# Making Papercraft Toys from Meshes using Strip-based Approximate Unfolding

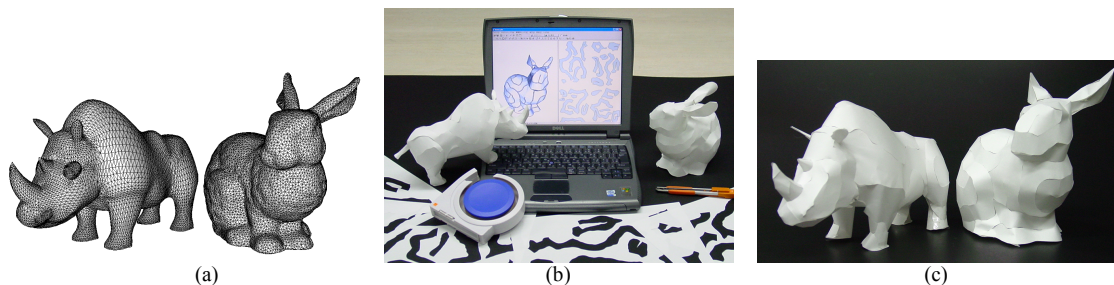Jun Mitani[*]        Hiromasa Suzuki[†]
University of Tokyo

Figure 1. (a) Mesh models.  (b) Making papercraft toys with a computer.  (c) Papercraft toys of the mesh models.

## Abstract

We propose a new method for producing unfolded papercraft patterns of rounded toy animal figures from triangulated meshes by means of strip-based approximation. Although in principle a triangulated model can be unfolded simply by retaining as much as possible of its connectivity while checking for intersecting triangles in the unfolded plane, creating a pattern with tens of thousands of triangles is unrealistic. Our approach is to approximate the mesh model by a set of continuous triangle strips with no internal vertices. Initially, we subdivide our mesh into parts corresponding to the features of the model. We segment each part into zonal regions, grouping triangles which are similar topological distances from the part boundary. We generate triangle strips by simplifying the mesh while retaining the borders of the zonal regions and additional cut-lines. The pattern is then created simply by unfolding the set of strips. The distinguishing feature of our method is that we approximate a mesh model by a set of continuous strips, not by other ruled surfaces such as parts of cones or cylinders. Thus, the approximated unfolded pattern can be generated using only mesh operations and a simple unfolding algorithm. Furthermore, a set of strips can be crafted just by bending the paper (without breaking edges) and can represent smooth features of the original mesh models.

**CR Categories**: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling -- Geometric algorithms, languages, and systems

**Keywords**: Mesh, Simplification, Unfold, Papercraft

------------------------------------------

[*] Currently with RIKEN (The Institute of Physical and Chemical Research); jmitani@riken.jp

[†] suzuki@den.rcast.u-tokyo.ac.jp

[*†] http://www.den.rcast.u-tokyo.ac.jp/~{mitani,suzuki}/

## 1  Introduction

3D figures displayed on a computer screen are just digital data stored in a computer - we cannot touch them. Realizing such computer models as physical objects is not only useful for engineering but is also entertaining in itself. Making papercraft from an unfolded pattern is one of the simplest methods of achieving this, and is a hobby which many people greatly enjoy. As one example of this, the unfolded pattern of the Utah Teapot, a well known test dataset, was created by Elber as a keepsake for the 25th anniversary of SIGGRAPH [SIGGRAPH 1998].

Unfoldable surfaces are a subset of ruled surfaces, and methods of approximating other surfaces by sets of developable ruled surfaces are well-studied [Elber 1995; Pottmann and Farin 1995; Hoschek 1998]. These approaches approximate parametric surfaces such as B-spline surfaces or rational Bézier surfaces by sets of ruled surfaces such as parts of cones or cylinders. The method of Chen et al. [1999] adapts such approximations to point clouds. However, it is hard to handle free form models of triangulated meshes such as Fig.1a with these approaches.

Methods for unfolding mesh models have been proposed by researchers in the field of mesh parameterization in recent years [Sorkine et al. 2002; Lévy et al. 2002; Sheffer 2002], but these approaches are not applicable for papercraft because they allow distortion of faces.

Because the Gaussian curvature of ruled surfaces (unfoldable surfaces) is zero, use of a curvature-based approach could be considered. However partitioning a free-form mesh model by means of its curvature is generally difficult. Our method is completely new in that it does not rely on curvature and instead converts a mesh to a set of strips, which are always unfoldable.

In principle, making an unfolded pattern from a mesh model without distortion is not difficult. We can generate the pattern simply by placing triangular faces on a plane, retaining as much as possible of its connectivity while checking that the resulting triangles do not intersect in the unfolded plane. However, in view of the size and intricacy of patterns such as Fig.2, physically creating such a pattern is unrealistic (as is assembling one!).
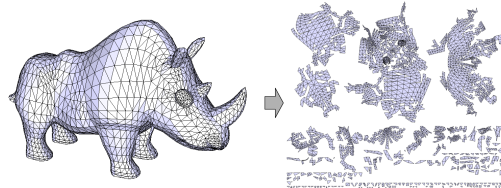


Figure 2. A mesh model and its unfolded pattern (4624 faces).

A more practical approach to making papercraft from a mesh model is to simplify the model using a mesh optimization method and then to unfold. Fig.3 shows an example of this approach. (c) is the unfolded pattern of the polyhedron (b) generated by simplifying the mesh model (a) using Garland and Heckbert's [1997] method, and (d) is a photo of the crafted model. This approach is practical and a commercial system for papercraft that generates unfolded patterns by unfolding low-polygonal models exists [ABNET Corp. 2003]. However, the papercraft models it generates from simple polygonal models become more angular, and the smoothness of the surface is lost.
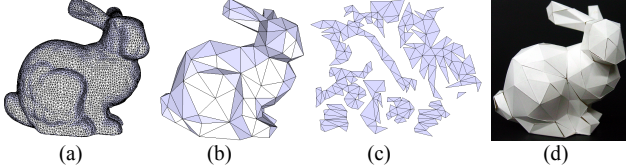


| (a) | (b) | (c) | (d) |

Figure 3. Making papercraft with a simplified mesh model.

This paper describes and illustrates our method. Section 2 reviews triangle strips. Section 3 describes our proposed method in detail. Section 4 shows our results, including illustrations of assembled papercraft toys. Section 5 outlines ideas for future work.

## 2  Triangle strip

A triangle strip, a common concept in graphics APIs [Neider et al. 1993], is a sequence of triangles such as the one shown on the left of figure in Fig.4a. Unfolding a polyhedron requires cuts, but we can easily unfold a triangle strip without cuts (right of Fig.4a) as long as the unfolded triangles do not intersect in the plane (when they intersect, we must separate them). Triangle strips can be connected by branching triangles to form a triangle tree; in general, triangle trees can be also unfolded without cuts. More generally, a triangle graph including loops can be unfolded by adding cuts for loops (Fig.4b). In this paper, we use *strip* to refer to any triangle graph which can be unfolded: triangle strips, triangle trees and simple triangle graphs with 'removable' loops.
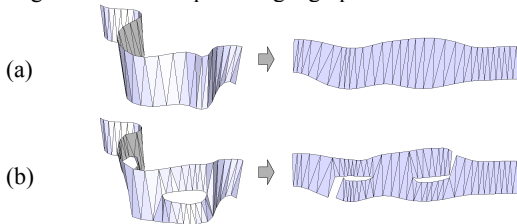


Figure 4. *Strips* and unfolded patterns.

Normally, each edge of an unfolded pattern will be bent when the figure is crafted, but sometimes the two mesh triangles are almost coplanar and the edge need not be bent, leaving a smooth surface in the crafted figure. Avoiding bending in this way also helps to reduce crafting time. It is also better if the strips are wide – this also helps to reduce crafting time. Hence, one goal of our method is to approximate the triangulated mesh model by a set of 'smooth and wide' triangle strips. Although building triangle trees from meshes is well studied in the fields of data compression and rendering [Taubin and Rossignac 1998; Evans et al. 1996], the strips which we want to generate differ from these in that ours are generated by changing the geometry and topology of the original meshes. Compared with approaches that approximate a model by a set of ruled surfaces such as parts of cones or cylinders, our new method has the advantage that no approximation by ruled surfaces is needed – it is purely a mesh method.

## 3  Detailed Method

We propose a method for making papercraft by strip-based approximation, and an overview of our method is shown in the top half of Fig.10. Fig.10a is a mesh model. We convert it into a set of strips (Fig.10f) and unfold these strips (Fig.10i), and then manually craft them to produce a figure (Fig.10h).

To make the strips, we must determine which edges will become strip borders (Fig.10e). We call these edges *cut-line*s - they will be cut after the model approximated by a set of strips is unfolded.

Each step of our method is described in detail below. We first segment a mesh model into parts (Sec.3.1) and then generate zonal regions (Sec.3.2). The borders between zonal regions become cut-lines. We add additional cut-lines (Sec.3.3) to ensure that important features of the original model are retained. We make these cut-lines smooth (Sec.3.4) and then generate strips the borders of which correspond to cut-lines by applying mesh optimization operations (Sec.3.5).

Our input models are manifold triangulated meshes, with or without borders, with tens of thousands of rather evenly distributed triangles.

### 3.1  Feature Line Extraction and Partitioning

Initially, we segment a mesh model into parts based on features to make it easy to craft. For example, when the target figure is an animal, we divide it into head, body, arms and legs. Methods for dividing a mesh model into parts have been proposed by several researchers (e.g. [Garland et al. 2001; Katz and Tal 2003; Lévy et al. 2002]); of these, we adopt the method of Lévy et al. Although this method was proposed for texture mapping, it is also good for papercraft – it detects sharp edges, and the feature boundaries tend to coincide with sharp edges whereas feature interiors contain smoother edges. This method extracts feature lines: lines with sharp edges and which are longer than some predefined length. *Charts* (what we call *parts*), sets of faces, are seeded with the faces whose distances from the feature lines are locally maximal, and these charts are expanded simultaneously from their seeds to meet each other at feature lines (for details, see [Lévy et al. 2002]). The method has parameters that affect the result; determining suitable values for these parameters is not straightforward. Rather than resort to trial and error to determine appropriate parameter values, we first set the parameters to generate small charts and then applied the following algorithm to merge them:

1.  Select the chart $C$ with the smallest number of triangles. If $C$ has more than predefined number of triangles (e.g. 3% of all triangles), finish.
2.  For each chart $H$ other than $C$, count the number of edges lying on the border between $H$ and $C$ (excluding the extracted feature lines).
3.  Merge $C$ with the chart for which the count is maximal, and repeat from step 1.

This operation divides a mesh model into parts, and the extracted feature lines tend to lie on the borders between these parts. Although Lévy's original method always produces charts which are homeomorphic to a disc, our additions do not always preserve this; this does not cause problems for the later steps we describe below. Fig.10(1b) shows the parts generated for the mesh in (1a) by this method. However, some extracted feature lines may not lie on borders, so we add these features to the cut-lines in Sec.3.3.

## 3.2 Generation of Zonal Regions

In order to approximate each part generated in Sec.3.1 by a set of strips, we segment the part into zonal regions as shown in Fig.10(1c). The triangles in each zonal region will be converted to a strip (Sec.3.5).

To generate zonal regions for each part, we assign a value to each triangle and segment the part by placing region borders on edges according to the values of neighbor triangles. The assigned value we use is the topological distance from the nearest part border or feature line (as extracted in the previous section) to the triangle. Zonal region borders are added along edges that connect triangles with assigned values $nw$ and $nw+1$ (for n=$1,2,3,...$, and $w$ is a positive integer). By this means, we can segment parts into zonal regions whose widths are $w$, leaving innermost internal areas. $w$, the width of zonal regions, affects the accuracy of the approximation - smaller $w$ generates more precise regions but make it harder to craft the resulting pattern.

The sizes of the resulting internal areas are unpredictable. Since we do not want narrow or tiny, difficult-to-handle pieces in our papercraft patterns, after the internal areas are generated, we merge those areas by the same algorithm as was previously applied to parts in Sec.3.1. For example we can merge those areas in which the difference between the minimum and maximum values assigned to triangles is smaller than $w/3$ or which contain fewer than 0.5% of all triangles.

The borders of zonal areas correspond to cut-lines, and we call them *border cut-lines*.

## 3.3 Addition of Internal Cut-Lines

Our method uses mesh optimization which retains cut-lines (see Sec.3.5). If we retain only border cut-lines, internal features may disappear (as in Fig.5a). To avoid this, we add extra cut-lines to zonal regions to ensure that these features are retained (e.g. the center red lines in Fig.5b). Adding cut-lines to a region corresponds to adding holes to a strip. The method of Lévy et al. generally extracts feature lines that lie on part borders (see Sec.3.1), but if there are any feature lines lying in the interior of a zonal region - away from the border more than appropriate distance (e.g. $0.3w$) - we record them as *feature cut-lines*.
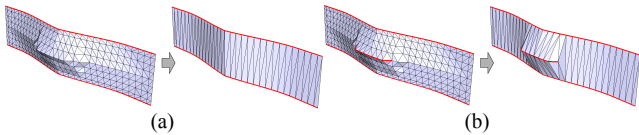


Figure 5. Mesh simplification without and with feature cut-line.

Additionally, we consider those regions homeomorphic to a disc that do not have any feature lines (Fig.6a, left). Even though smooth rounded forms are important for representing rounded figures such as animal toys, mesh simplification destroys these features (Fig.6a) for the same reason as given above. In order to avoid this, we apply an algorithm to extract core lines of regions homeomorphic to a disc by shrinking their outer loops as follows:

1. Add all triangles in the region to a list $T$, and make an outer loop $L$ of the region that is a list of edges in counterclockwise order.
2. Update $L$ by removing a triangle from $T$. The triangle to be removed is the nearest to the border of the region. The way $L$ is updated differs depending on to how the triangle touches $L$ (see Fig 7) - Fig.7a to Fig.7f illustrate the various possibilities. The thick red lines are edges of the outer loop $L$ and the arrows show the direction of the loop. Same-colored

arrows indicate adjacent edges in $L$. Triangles in $T$ are shown in pink, and the target triangle to be removed is shown in red. Fig.7a shows the case where a triangle touches $L$ along a single edge, (b) and (c) show the case with two touching edges and (d), (e) and (f) the cases with three touching edges.
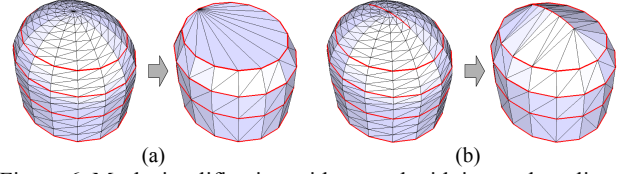3. Repeat step 2 until $T$ is empty.



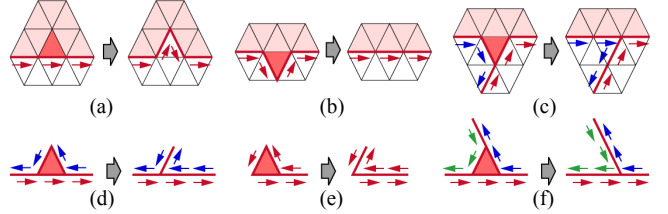Figure 6. Mesh simplification without and with internal cut line.



Figure 7. Rules for updating outer loops.

The red lines in Fig.8a show borders of zonal regions and feature cut-lines. The core lines added by the above algorithm are shown in Fig.8b. With this algorithm a region loses all of its triangles and converges to a set only of edges and their vertices, which we call core lines. The algorithm shrinks a region continuously from its border without making holes (so the core lines do not have loops), to form a tree structure. Since these lines would be too complicated to craft, we simplify each core line using the following algorithm:

1. Create an edge-vertex tree of the core lines derived from the previous algorithm.
   (a) Make a list $L$ of the vertices that are leaves of the edge-vertex tree.
   (b) Repeatedly remove one vertex in $L$ from the list and the core line. When the number of vertices in the core line is reduced to a predefined ratio (e.g. 15%), go to step 2. If $L$ becomes empty, update the edge-vertex tree of the core line and repeat from (a).
2. If there are any vertices nearer than a predefined distance (e.g. $0.3w$) to the outer loop, remove the edges that connect these vertices from the core line.

Fig.8c shows the result of this algorithm – it can be seen that the resulting core lines have been simplified. We call these core lines *center cut-lines*.
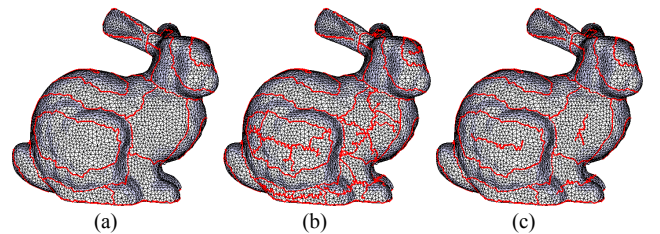


Figure 8. Generation of center cut-lines

3

## 3.4  Smoothing of Cutting Lines

In order to make it easy to cut out parts from a sheet of paper and to paste them, we smooth all cutting lines (border cut-lines, feature cut-lines and center cut-lines) generated in previous sections. To do this, we apply both *connectivity smoothing* and *geometrical smoothing* as described below.

### 3.4.1  Connectivity smoothing

*Connectivity smoothing* is illustrated in Fig.9. Where there is a triangle (such as the gray triangle in Fig.9a) with two edges on the same cut-line, we replace that part of the cut-line by the third edge of the triangle as shown in Fig.9b. This replacement is not applied to triangles where the cut-line branches out.
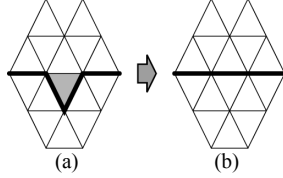


(a)                    (b)

Figure 9. Connectivity smoothing

### 3.4.2  Geometrical Smoothing

In order to make cut-lines smooth, after applying connectivity smoothing, we move those vertices on cut-lines which have two neighbor vertices on the same cut-line by using the one-dimensional Laplacian operator shown in equation (1).

$$p[i] = \frac{1}{4}p[i-1] + \frac{1}{2}p[i] + \frac{1}{4}p[i+1] \qquad (1)$$

$p[i]$ is the coordinates of a vertex to be moved, and $p[i-1]$, $p[i+1]$ are the coordinates of the neighbor vertices connected to $p[i]$ on the cut-line. Repetition of this operation makes the cut-line smoother still, but also makes the line shrink. We apply this operation just twice, enough to make it smooth without shrinking it too much.

## 3.5  Simplification to Generate Strips

We simplify the mesh model and generate strips by a method of constrained mesh simplification that removes all the internal vertices in a zonal region while retaining the edges on cut-lines extracted by the methods described in the previous sections (see Fig.6b). Cohen [1999] has collected several methods for simplifying meshes. Of these, we used the mesh simplification algorithm proposed by Garland and Heckbert [1997], which uses the *edge-collapse* operation. During this operation, we add a check that face normals are not inverted - if such undesirable side-effects occur, we undo the *edge-collapse*. The edge-collapse operation is repeated as far as it can be applied. After that, if vertices still remain that are not on any cut-line, we use a *vertex-removal* operation [Turk 1992]. As a result of this simplification, we obtain a set of strips which is a simplified mesh with vertices only on cut-lines. After simplification, we use an *edge-swap* operation [Hoppe et al. 1993] to make strips smooth. This *edge-swap* is applied only to edges that are not on cut-lines and only when the second order difference (i.e. the angle between the normals, as in [Hubeli and Gross 2001]) becomes smaller.

## 3.6  Unfolding and Packing

Unfolding a strip is trivial. We place one of the triangles in the strip on the plane and recursively add those triangles connected to triangles already in the plane. If and when they intersect, we divide the unfolded strip into two.

The problem of packing pieces has been studied elsewhere (see, e.g. [Milenkovic 1999]). We do not discuss this problem here, and in producing the results shown in the next section we placed the pieces manually.

## 4  Results

Fig.10 shows the various stages of our method applied to a bunny mesh model with 19996 triangles (upper), and a rhinoceros mesh model with 18496 triangles (lower). Fig.10a is the initial mesh, (b) is the result of partitioning and (c) is the result of generation of zonal regions. We used 12 and 11 respectively for the parameter *w* (the width of the zonal regions), and merged regions smaller than 60 and 250 triangles (we determined these parameter values by trial and error). (d) shows cut-lines, added feature cut-lines and center cut-lines. (f) shows the set of strips generated by mesh simplification and (g) is the model with the cut-lines enhanced so as to make it easy to see the strips. (h) and Fig.1c are photos of the assembled papercraft toys and (i) is the unfolded pattern. We used a cutting machine to cut out the pieces; the unfolded patterns were cut from four sheets of A4 paper. We assembled the figures using scotch tape (automatically generating flaps on cut-lines is straightforward, but we find that using scotch tape without flaps is easier than gluing flaps). The heights of the papercraft models are 17cm (bunny) and 14cm (rhinoceros), and it took 2¼ hours and 3½ hours respectively to assemble them - this is about the same as the time taken to craft the simple polygonal model in Fig.3 with about 300 faces. As mentioned in Sec.2, if we do not have to bend internal edges of strips, we not only generate figures with smooth surfaces but also greatly save crafting time, and in practice we did not have to bend almost all internal edges. Fig.1b shows the equipment required for crafting – note that the correspondence between parts of the 3D model and those of the pattern is shown by a system implemented on a PC.

In terms of accuracy, the geometrical error between an input mesh model and its simplified model is rather large compared to those with other simplification methods. The RMS (root mean square) errors as measured by Metro [Cignoni et al. 1998] between the original models and those optimized by our method are 0.0126 and 0.0113 respectively (size of models are normalized to 1). These error values are almost same as those when we simplify the original meshes to 270 faces and 200 faces respectively using Garland and Heckbert's [1997] method.

## 5  Conclusion and Future Work

We proposed a new method for producing unfolded papercraft patterns from triangulated meshes by means of strip-based approximation. The effectiveness of the method was demonstrated by examples that kept the smoothness of original meshes. The time consumed in crafting was reasonable. Although the accuracy of approximation our method achieves may not appear particularly good, the most important criterion is the visual difference between an input mesh model and its papercraft figure (papercraft figures will usually include additional large geometrical errors due to manual assembly). It can be seen that judged by this criterion our method works well for both models generated originally from ranged data (bunny) and designed manually (rhinoceros).

With our approach, we have to find appropriate parameters by trial and error (in practice, this is not difficult), and we cannot specify the approximation tolerance to the input mesh model. In future, we hope to find method for automatically finding applicable parameters for predefined tolerance. Additionally,

methods for quantifiably evaluating the generated patterns are required.

Furthermore, when papercraft is designed by humans, thin parts of models (such as ears of animals or wings of birds) are sometimes simply a single sheet of paper. We should wish to enhance our method so that such non-manifold models can also be simplified.

## Acknowledgements

## References

ABNET CORP, 2003. Pepakura Designer. *http://www.e-cardmodel.com/pepakura-en/*.

CHEN, H., LEE, I., LEOPOLDSEDER, S., POTTMANN, H., RANDRUP, T., AND WALLNER, J. 1999. On Surface Approximation Using Developable Surfaces. *Graphical Models and Image Processing*, *61*, 2, 110-124

CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 1998. Metro: measuring error on simplified surfaces. *Compute Graphics Forum*, *17*, 2, 167-174

COHEN, J. D. 1999. Concepts and Algorithms for Polygonal Simplification. *SIGGRAPH 99 Course Tutorial #20*, C1-C34.

ELBER, G. 1995. Model Fabrication using Surface Layout Projection. *CAD 27*, 4, 283-291.

EVANS, F., SKIENA, S., AND VARSHNEY, A. 1996. Optimizing Triangle Strips for Fast Rendering. *IEEE Visualization '96*, 319-326.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proc. of ACM SIGGRAPH 1997*, 209-216.

GARLAND, M., WILLMOTT, A., AND HECKBERT, P. 2001. Hierarchical face clustering on polygonal surfaces. In *Proc. of ACM Symposium on Interactive 3D Graphics*, 49-58.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh Optimization. In *Proc. of ACM SIGGRAPH 1993*, 19-26.

HOSCHEK, J. 1998. Approximation of surfaces of revolution by developable surfaces. *CAD*, *30*, 10, 757-763.

HUBELI, A., AND GROSS, M. 2001. Multiresolution features extraction from unstructured meshes. In *Proc. of IEEE Visualization*, 287–294.

KATZ, S., AND TAL, A., 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics*, *22*, 3, 954-961.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, *21*, 3, 362-371.

MILENKOVIC, V. J. 1999. Rotational polygon containment and minimum enclosure using only robust 2D constructions. *Computational Geometry*, *13*, 1, 3-19.

NEIDER, J., DAVIS, T., AND WOO, M. 1993. OpenGL Programming Guide. *Addison-Wesley*, 1993.

POTTMANN, H., AND FARIN, G. 1995. Developable rational Bézier and B-spline surfaces. *CAGD*, *12*, 5, 513-531.

SHEFFER, A. 2002. Spanning Tree Seams for Reducing Parameterization Distortion of Triangulated Surfaces. *Shape Modelling International*, 61-66.

SIGGRAPH 1998. SIGGRAPH 98 Teapot Assembly. http://www.siggraph.org/s98/conference/teapot/

SORKINE, O., COHEN-OR, D., GOLDENTHAL, R., AND LISCHINSKI, D. 2002. Bounded-distortion Piecewise Mesh Parameterization. In *Proc. of the conference on Visualization '02*, 355-362.

TAUBIN, G., AND ROSSIGNAC, J. 1998. Geometric Compression Through Topological Surgery. *ACM Transactions on Graphics*, *17*, 2, 84–115.

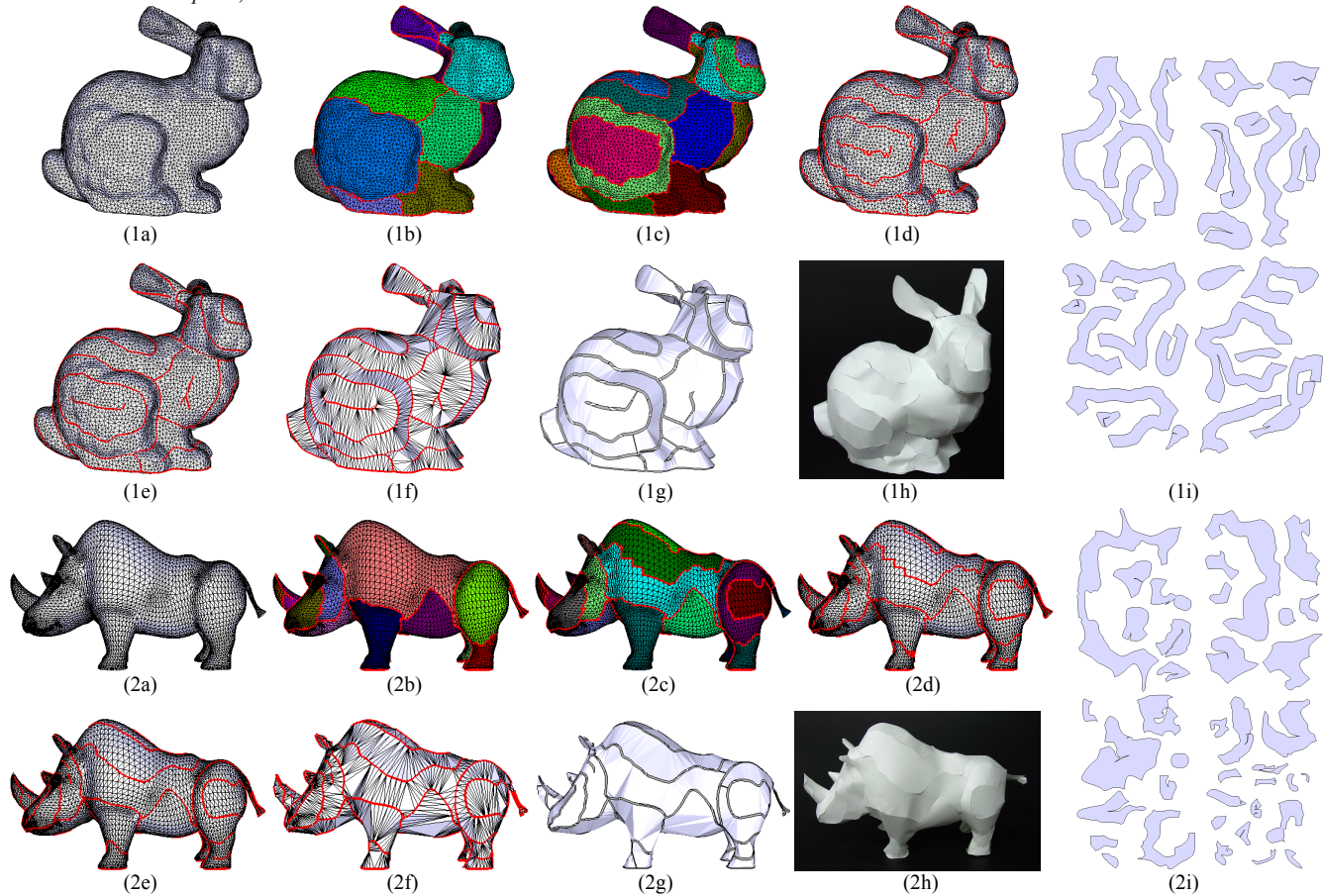TURK, G. 1992. Re-tiling Polygonal Surfaces. In *Proc. of ACM SIGGRAPH 1992*, 55-64.

Figure 10. (a) Initial mesh  (b) Partitions  (c) Zonal regions  (d) Cut-lines
(e) Smoothed cut-lines  (f) Set of strips  (g) Enhanced cut-lines  (h) Photo of papercraft  (i) Unfolded pattern