Papercraft Models using Generalized Cylinders

Fady Massarwi
Technion
fadym@cs.technion.ac.il

Craig Gotsman
Technion
gotsman@cs.technion.ac.il

Gershon Elber
Technion
gershon@cs.technion.ac.il

Abstract

We introduce an algorithm for approximating a 2-manifold 3D mesh by a set of developable surfaces. Each developable surface is a generalized cylinder represented as a strip of triangles not necessarily taken from the original mesh. Our algorithm is automatic, creates easy-to-assemble pieces, and provides L_∞ global error bounds. The approximation quality is controlled by a user-supplied parameter specifying the allowed Hausdorff distance between the input mesh and its piecewise-developable approximation. The strips generated by our algorithm may be parameterized to conform with the parameterization of the original mesh, if given, to facilitate texture mapping. We demonstrate this by physically assembling papercraft models from the strips generated by our algorithm when run on several polygonal 3D mesh data sets.

1. Introduction

Producing a physical replica of a 3D digital model is an important task in computer-aided manufacturing. One common way to do this computes developable versions of pieces of the given surface geometry, which are then cut, bent and assembled from flat sheets of material to approximate the original surface. This manufacturing paradigm is popular in the ship and aircraft industries, as well as cloth and shoe making, to name a few.

This flattening approach can be reduced to the problem of generating a set of developable patches (sheets), that together approximate the 3D model. This approximation problem poses several challenges, like constructing intuitive and easy-to-assemble patches, minimizing the number of patches and the distortion/error in the boundaries between the patches, and minimizing the error between the original surface and its approximation.

A solution to this problem also depends on the material the model is made of. Building a papercraft model is more challenging than building a rubber model, since, due to the physical limitations of paper, stretch cannot be accommodated. Furthermore, some materials are unisotropic (e.g. fabric) and stretch differently in different directions.

A good solution should be easy to use and intuitive to the user. It should also take into account and support different properties of the model, such as color and texture coordinates.

In this work, we present an algorithm that creates a piecewise-developable approximation to a given 3D polygonal mesh. The input is a 3D polygonal mesh M, and a tolerance value $\tau > 0$ that controls the approximation quality. The algorithm consists of three stages. In the first stage, which may be considered a pre-processing stage, the mesh is segmented into meaningful components. This segmentation is necessary only for complex meshes. Meaningful components are frequently separated by sharp edges and edges with high curvature. If segmentation is possible, using it typically leads to better results and a smaller number of strips. In the second stage, each of the mesh components are approximated in 3D by a set of triangle strips with smooth (non-jagged) boundaries that guarantee a maximal error of τ from M. These triangle strips are tessellations of *generalized cylinders*. The strip vertices are also assigned texture coordinates derived from those of the mesh if they were supplied. In the final stage these strips are unfolded to the plane, resulting in the final flat patterns, which may be cut and assembled from paper.

2. Related Work

The quality of an algorithm that approximates a mesh by a set of developable surfaces can be measured using the following criteria:

Error control and the number of generated strips. The algorithm should provide easy control over the tradeoff between the number of strips generated and the accuracy of the approximation. A large approximation error will result in a small number of strips, and vice-versa.

Strip quality. The strips should be easy to cut and glue together. This means smooth boundaries are preferable, and cutting and gluing hints should be supplied.

Automation. The algorithm should be as automatic as possible and the input parameters should be as intuitive as possible. This includes the approximation error parameter, which should have a geometric meaning.

Several results have already been obtained in this area. Elber [7] proposed a method for approximating NURBs by a set of developable strips. That work treats only a single surface patch (and hence simple rectangle topology), as opposed to models with complex geometry and topology. Elber also uses quite a loose upper bound on the Hausdorff distance as an error approximation. This bound can be far from the true Hausdorff distance, resulting in a much larger number of strips than necessary.

Julius et al. [11] segment a given mesh into compact charts by minimizing the length of the chart boundaries. Their algorithm is greedy, as it starts from some initial patch and applies region-growing techniques, trying to merge faces that fit best to the same part of a cone. It produces non-developable patches of the original mesh polygons that can be flattened and connected together only if distortion and stretching is allowed, thus are not really appropriate for paper. In terms of the criteria listed above, there is no user control of the approximation, and there are several non-intuitive input parameters involved.

Mitani et al. [18] propose a strip-based segmentation which involves simplification of the model and extraction of foldable strips from the simplified triangles. They segment the model into several components as proposed by Levy et al. [16], and then each component is cut in strips at several offset distances from the component boundary. This method is appropriate for paper-craft, but requires a few threshold parameters that are not intuitive to the user. Moreover, there is no control over the approximation quality, thus the difference between the original mesh and the approximation can be quite large. Finally, since each surface strip is approximated as a triangle strip whose boundaries consist of mesh edges, these boundaries are jagged, making it harder to assemble from paper.

Shatz et al [21] approximate the mesh by a set of cones and planes. As an initial stage they assign a plane to each face. A greedy iterative process then merges neighboring faces if they can be approximated together by a part of cone or plane, such that the total error between the faces and the approximation does not exceed the given tolerance. This method guarantees that neighboring cones have a common boundary. Thanks to this, analytical boundaries between the cones are extracted, which ensure smooth boundaries.

This method works well, but it is limited to only cones and planes, a small subset of the class of developable surfaces. Consider a cylinder with an elliptic cross section. The flat pattern developed by Shatz et al. for this cylinder will consist of many pieces while the original surface is clearly developable. Shatz et al. employ a metric which is similar to the L_2 norm in the sense that it is the sum of the Euclidean distances between the mesh vertices and the approximating cones. This means that the final result could introduce an arbitrarily large error in the L_∞ sense. This method also creates best-fit cones that approximate the original mesh. These cones are not directly related to the mesh and not triangulated at all. Thus deriving the correct texture information from the model for the generated cones is not easy.

Kolmanic and Guid [14] approximate a mesh by a set of developable strips using given cross-section curves of the input. This method can handle only very simple inputs, and, in fact, was primarily designed to approximate shapes of shoes. It cannot handle models with complex geometry or topology nor models containing holes. Further, it requires the use to supply additional inputs (the cross section curves).

Work has been done on 3D model segmentation, but most of this is not appropriate for our problem, since these methods typically generate non-developable patches [2, 12, 13, 15, 17, 23, 24]. Other segmentation work [19] identifies tubular surfaces of the mesh that can be easily approximated with paper, but does not handle parts of the mesh that are not tubular. We use segmentation methods as a preprocessing stage, preferring to segment the model into meaningful components with smooth boundaries. There are several mesh segmentation algorithms suitable for our purposes. We prefer the methods proposed by Katz et al. [12] and Zhang et al. [24], as we describe later.

We measure approximation error by the maximal symmetric Hausdorff distance between the input mesh M_1 and the approximating surface M_2:

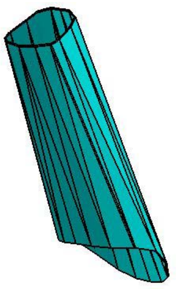
$$H(M_1, M_2) = \max\left(\begin{array}{c}\max\limits_{p \in M_1}\left(\min\limits_{q \in M_2}\|p-q\|_2\right)\\ \max\limits_{q \in M_2}\left(\min\limits_{p \in M_1}\|p-q\|_2\right)\end{array}\right).$$

To the best of our knowledge there is no algorithm for computing the exact Hausdorff distance between triangular meshes. In their absence, some methods [1,3,9] approximate the Hausdorff distance by sampling points on one mesh and calculating their distances from the other. These methods apply different optimization techniques to reduce the time complexity of the computation, which otherwise can be quite lengthy.

3. Our Algorithm

Our algorithm approximates the input mesh by a set of generalized cylinders, which are a subset of the developable surfaces - those whose Gaussian curvature vanishes everywhere [6].

A central concept in our algorithm is the *ruled surface*. A ruled surface between two parametric curves $C_1(t)$ and $C_2(t)$ is $S(u,t) = (1-u)C_1(t)+uC_2(t)$ for $u \in [0,1]$. Our ruled surfaces will actually be between two polylines, in the form of *triangle strips*. A triangle strip is a sequence of triangles in space where each triangle shares an edge with its predecessor and a different edge with its successor, as in the image on the right. If the strip is not closed, the first and last triangles are exceptions (they are incident on just one other triangle). We will deal mostly with closed triangle strips, which have the topology of a generalized cylinder (or generalized cone if degenerate). Thus we can define a *triangulated ruled surface* (TRS) as follows. Consider a ruled surface $S(u,t)$ defined by two curves, $C_1(t)$ and $C_2(t)$, and let $P_1(t)$ and $P_2(t)$ be piecewise linear approximations to $C_1(t)$ and $C_2(t)$, respectively. The TRS is the triangle strip that is a discrete approximation to $S(u,t)$, achieved by piecewise-linear interpolation between $P_1(t)$ and $P_2(t)$.



The class of developable surfaces is a subclass of the class of ruled surfaces [6], thus a general ruled surface is not developable, and is typically hyperbolic (having negative Gaussian curvature). However, due to their piecewise-linear structure, the TRS's we generate are always piecewise-developable strips.

Our algorithm receives as input only two parameters: a 2-manifold mesh, M, and a tolerance value $\tau > 0$. The output is a set of flat triangle strips which, when folded, are guaranteed to approximate M by a Hausdorff distance of less than τ. The algorithm consists of three main stages:

Mesh segmentation: The input mesh M is segmented into meaningful components, typically cylindrical in the topological sense. This stage is considered a preprocessing stage, applied to complex inputs only. The rest of the stages are applied on each of the resulting components recursively. The segmentation stage is described in detail in Section 3.1.

Mesh approximation: The mesh is approximated by a set of TRS's by a recursive refinement process, which terminates when the Hausdorff distance between the approximation and the mesh is below τ. This is described in detail in Section 3.2.

Unfolding and merging: In this stage each of the 3D strips generated in the previous stage is unfolded onto a 2D plane, while neighboring strips are merged if possible. This stage is described in detail in Sections 3.3 and 3.4.

3.1 Mesh Segmentation

Meaningful parts in a typical mesh M are frequently separated by sharp edges, those with high curvature. Attempting to approximate these sharp folds with developable surfaces will typically result in an unnecessarily large number of small developable sheets. To achieve better results, we segment the model into meaningful parts, and apply our approximation algorithm to each part independently. Since the proposed scheme interpolates the boundaries, the end result will be a C^0 continuous surface, despite the segmentation.

While not mandatory, the produced flat sheets will preferably have smooth boundaries. Since the smoothness of these boundaries is influenced by the boundaries of the input mesh M and the boundaries introduced by the segmentation algorithm, a good candidate to be a segmentation algorithm for our purposes is one that produces meaningful components with smooth boundaries.

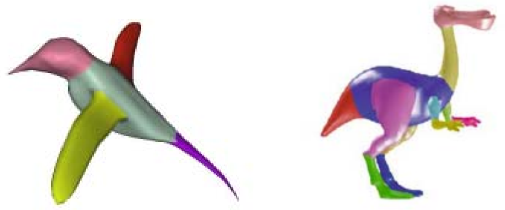


Figure 1. Segmentation to meaningful parts. (left) Results of [24] on a bird model. (right) Results of [12] on a Dinopet model.

There are quite a few algorithms for mesh segmentation [2, 12, 13, 15, 17, 19, 23, 24] that segment the mesh into meaningful components with smooth boundaries. We found that the algorithms from [12,24] were the most appropriate. See Fig. 1 for an example.

3.2 Mesh Approximation

We are now ready to approximate the input mesh *M* by a set of TRS's with smooth boundaries, while guaranteeing an approximation error of at most τ, measured by the Hausdorff distance between the piecewise developable approximation and *M*.

The approximation algorithm is a recursive refinement algorithm, having four main steps at each iteration:

1. Two boundaries of *M* are extracted, as described in Section 3.2.1.
2. A TRS is defined between the two boundaries as an approximation to *M*, as described in Section 3.2.2.
3. The Hausdorff distance δ between the TRS and *M* is calculated using the Metro tool [3].
4. If $\delta<\tau$, the TRS is adopted as an approximation to *M* and the algorithm terminates. Otherwise, *M* is cut at the maximal (Hausdorff) error point into two parts, and the algorithm is run recursively on each part. This cutting step is described in Section 3.2.3.

3.2.1. Selecting approximating TRS boundaries

For a given mesh *M*, we seek two boundaries that are most appropriate to serve as the boundaries of the TRS that will approximate *M*. To provide reasonable approximations for the geodesic distances between vertices (and boundary edges), the shortest path distance between all vertices is computed using the Dijkstra algorithm [5]. Then the problem of selecting the two boundaries is classified as one of the following cases:

Zero boundaries: If *M* is closed, we select as (degenerate) boundaries two vertices that present the maximal geodesic distance among all pairs of vertices of *M*.

One boundary: If *M* has one boundary, we select it as the first boundary and select the vertex that presents the maximal Hausdorff distance from the first boundary as the second (degenerate) boundary.

Two or more boundaries: In all the other cases, we consider all boundaries of *M* as candidates. We also consider as candidates vertices that present the maximal Hausdorff distance from each boundary. We employ several heuristics to select two good boundaries from the candidate set. The first simply selects the two candidates that result in the best approximation (minimal Hausdorff error) among all TRS's of all the possible pairs of boundary candidates. While this first approach is locally optimal, it is also very slow (because it constructs and evaluates the distance for every possible pair of boundaries). A faster, but not optimal, scheme selects the two boundaries that have the maximal geodesic distance from each other. See Figure 2 for simple examples of these three cases.

The TRS will have texture coordinates taken from its boundary polylines that are part of the model.

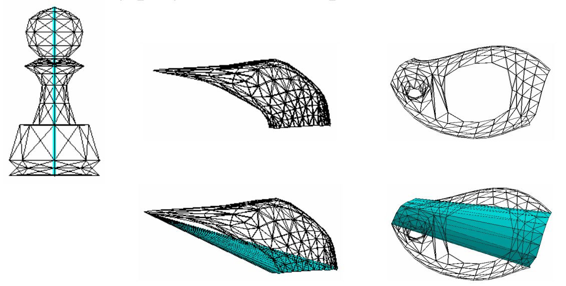


Figure 2. Selection of TRS boundaries. The blue faces are the TRS that approximates the mesh at the first iteration of our algorithm. (left) The pawn model is closed – has no boundaries. (center) The bird's head has only one boundary. (right) The body of the dinopet model has 6 boundaries**.**

3.2.2. Building a TRS between two 3D polylines

Having selected two 3D piecewise linear (polyline) closed boundaries, we are now ready to build a TRS that connects them as a piecewise-linear ruled surface. This boils down to finding a correspondence between points on both boundaries, which can then be joined to form a triangle strip. The correspondence is usually determined by optimizing some cost function. This typically requires O(*nm*) time using dynamic programming, where *n* and *m* are the number of vertices in the two boundaries. We build the TRS that minimizes the *twist* in the surface normal along the ruling lines, following, for example, the approach suggested in [22], with the difference that in our case the polylines are closed.

A ruled surface $S(u,t)$ between curves $C_1(t)$ and $C_2(t)$, is developable if for all *t*, the tangent vectors of $C_1(t)$ and $C_2(t)$ and the line that connects them, are co-planar. More formally, when

$$\left[C_1(t)-C_2(t),C_1'(t),C_2'(t)\right]=\left(C_1(t)-C_2(t)\right)\cdot\left(C_1'(t)\times C_2'(t)\right)$$

vanishes for all $t\in[0,1]$. This condition also means that the normal of the surface at the points $C_1(t)$ and $C_2(t)$ should be the normal of the plane that contains the line $C_1(t)$-$C_2(t)$.

For a TRS, if vertex a_i of the first boundary and vertex b_j of the second boundary are connected (i.e they are vertices in some triangle in the TRS), one can approximate the unit normal at point a_i as:

$$n_{a_i}=\frac{(a_i-b_j)\times d_{a_i}}{\left\|(a_i-b_j)\times d_{a_i}\right\|},$$

where d_{ai} is the tangent vector of C_1 at a_i. Similarly one can approximate the unit normal at b_j as:

$$n_{b_j} = \frac{(a_i - b_j) \times d_{b_j}}{\left\| (a_i - b_j) \times d_{b_j} \right\|}.$$

Twist is minimized by minimizing the total difference between these normal vectors, or minimizing

$$\sum_{i=1}^{n} \left(1 - \left\langle n_{a_i}, n_{b_j} \right\rangle\right)$$

This may be done using dynamic programming in a standard manner [4,8], applied to a graph on a set of vertices which are pairs of points from both polylines, where each path in the graph defines a TRS.

The boundaries of the TRS are polylines that are part of the mesh. If the mesh is also endowed with texture coordinates, these are linearly interpolated from their values along the boundary to the vertices of the TRS.

3.2.2. Splitting the mesh

At each iteration of the approximation algorithm, if the approximation error between the TRS and M is larger than τ, M is split into two pieces and the algorithm recursively approximates each of the two pieces. Only one exception exists to this rule. When M contains internal boundaries we always split at these internal boundaries at the end of the process, regardless of the error, seeking the topology of a cylinder.

Recall that we require the cut of M to be smooth. To obtain this, we compute a continuous function f over the mesh, such that f is constant and obtains its minimum on the first boundary of the approximating TRS, and is constant and obtains its maximum on the second boundary of the TRS. By emulating a harmonic potential field obeying the Poisson equation, f can be made continuous and monotonic between the two boundaries and on the entire mesh M, satisfying the given boundary conditions. Without lose of generality, assign the value f=0 to all vertices of the first boundary and f=1 to all vertices of the second boundary. Then, the function f on each internal vertex v_i of M, should satisfy

$$\sum_{v_j \in Neigh(v_i)} w_{ij}(f(v_i) - f(v_j)) = 0$$

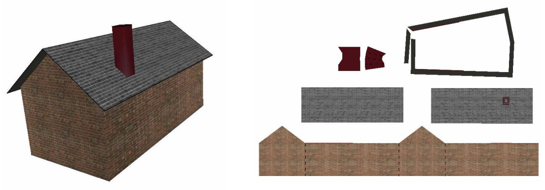
where w_{ij} are the standard cotangent weights [20] of the Laplacian operator on a manifold 3D mesh.

We compute a smooth cut through the point that presents the maximum Hausdorff error (as computed in the previous stage) by using the iso-contour of f determined by the value of f at that same maximal error location. If the maximal Hausdorff error occurs in more than one point, one is chosen arbitrarily. This requires cutting the mesh along edges, which is done by introducing new vertices. These new vertices are then assigned texture coordinates by linear interpolation of the texture coordinates of the original neighboring vertices.

3.3 Unfolding the TRS

As the final stage, the 2D strips are generated by unfolding the TRS created in the previous stage. Later, neighboring strips can be merged if possible, as described in Section 3.4.

Since the patterns could globally self intersect in the plane, we check at each step of the unfolding whether the current triangle projected to the plane intersects previously projected triangles. If so, a new triangle strip is initiated. This rarely happens, yet an example of such a case can be found in the house pattern below (see the roof's thickness strip).



3.4 Merging Strips

It is sometimes possible to merge neighboring strips to reduce their number. We check if pairs of flat strips have a common identical planar boundary, up to rigid motion, and merge them into one strip if possible. This should not happen for a mesh that does not have internal boundaries, because our algorithm cuts the mesh at the maximal error point at each iteration. Thus, neighboring strips cannot belong to the same TRS, implying that their flat strips cannot have common boundaries.

Nonetheless, these situations are possible with meshes containing internal boundaries (holes). In this case and regardless of the error, we first split along all the internal boundaries, even if the neighborhood can be approximated by a single TRS with holes.

Consider the cylinder in Fig. 3 (left) having two holes. One piece of paper (right) is sufficient to construct this cylinder, but because the cylinder contains holes, the algorithm will generate three strips, as in Fig. 3 (center). The flat versions of these three 3D strips have common boundaries, and can be merged to one flat strip, as in Fig. 3 (right).

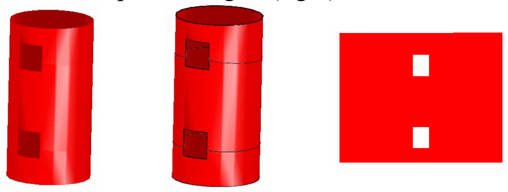


Figure 3. (left) Input model of cylinder containing two holes. (center) Result of algorithm on (left), generating 3 strips. (right) The final approximating strip after unfolding and merging neighboring strips.

5. Experimental Results

In this section we present some results of our algorithm run on several mesh models. We show the original mesh, the approximation result as a 3D mesh, and the final flat strips. In a few examples, the original mesh is color-coded to indicate the error between the mesh and its approximation. Red regions represent the largest error whereas green regions depict very small error and blue depicts intermediate error. Feature lines are marked as dashed lines, projections of intersection curves between strips are drawn in red, and cut lines (where the TRS is cut before it is unfolded) are drawn in blue (see Fig. 9). A photograph of physical paper-craft models constructed from some of the layouts appears at the beginning of the paper. Table 1 lists the model attributes, the number of generated strips and the runtimes of the algorithm (without the segmentation stage) on a 3 GHZ machine with 1GB RAM. Our implementation was in C++ in the Windows XP environment. Approximation error was measured as a percentage of the largest dimension of the model bounding box.

Table 1: *Results of our algorithm on a number of 3D mesh models.*

Model	Num Faces	approx error (%)	num flat strips	time (sec)
Whale	11,034	5	8	39
		2	20	63
Fish	6,380	2	22	55
Flamingo	8,951	2	30	142
Rex	8,694	2	37	52
Teapot	26,820	2	20	98
Eight	1,536	2	16	20
Me262	2,508	5	24	11
Bird	3,000	5	7	8
		2	15	13
B58	10,568	2	46	35
Pawn	304	1	17	6

Computations that are influenced by the model's resolution, such as the geodesic distance between all pairs (if needed) and/or the Hausdorff distance bound, are the most expensive computations at each iteration.

The behavior of our algorithm may be observed through these examples. First, Figs. 5, 6, 7 and 12 show how the texture coordinates are computed and properly inherited by the flat 2D strips. The flamingo model results in several boundaries that must be considered, thus the selection of the TRS consumes more time. The advantages of using the TRS as an approximation primitive are evident in several examples, e.g. the b58 and the me262 in Figs. 6 and 10: most of the parts of the b58 and me262 are approximated by just one strip, as the surfaces are (almost) developable. Methods that are restricted to using cones and planes will approximate these surfaces by several planes- or cone-parts.

Our algorithm can handle complex meshes such as the Dinopet in Fig. 13 and the Rex in Fig. 12. The algorithm is also capable of handling meshes with non-trivial topology such as the Figure Eight model in Fig. 8, whose triangle strip structure is also shown. Fig. 4 illustrates various stages of the approximation process for this input.

The smoothness of the boundaries and other algorithmic features like the feature lines, cut lines control, and connectivity information makes it easier to connect and assemble the strips. The layout of the 2D strips in our examples reflects the 3D approximation. This was simple to achieve using the connectivity information, and controlling the cut lines position.

Different values of the approximation tolerance τ lead to different results, as can be seen in the bird model in Fig. 9. A large tolerance will result in a less accurate, yet smaller number of strips. While the pattern still reflects the shape and the texture of the original mesh, a tighter τ yields a more accurate result at the price of more strips.

The smooth boundaries, the approximation quality control and the automation of our algorithm gives it clear advantages over the method of Mitani et al [18] that is not automatic, creates jaggy boundaries and does not provide control over the error and the approximation quality.

Using a large family of developable surfaces creates a clear advantage over the method proposed by Shatz et al [21]. This is demonstrated in Figs. 13 and 14. To make a fair comparison with [21], in these examples we employed the error metric used in [21] (the square sum of distances of the original vertices from the approximation result).

In the dinopet example (Fig. 13), our algorithm generates 50 strips with an error of 4.9% with smooth boundaries, while the algorithm of Shatz et al [21] results in 61 strips with comparable error. With 60 strips, our algorithm approximates with error of 3.7%.

In the Venus example (Fig. 14) our algorithm results in 32 strips with error of 4.5%, while the algorithm of Shatz et al. [21] results in 39 strips for an error of 5%.

For a very small error, our algorithm could generate long and skinny strips, as in Fig. 14. Each strip is connected to only two other strips (one on each side of it), thus it is not difficult to fold and glue together. Addiitonally, because these strips are connected along smooth boundaries, they can better approximate feature areas of the model. These strips are preferred over small and compact pieces in practice, as can be seen, for example, in the manually made paper-models in [10].

6. Conclusions and Future Work

We have presented a scheme to approximate an arbitrary 2-manifold 3D mesh using piecewise developable surfaces with a global error bound. No distortion in the created flat pattern was allowed, making it an ideal solution for papercraft.

Efficient computation of the exact Hausdorff distance between two meshes is an open question and its answer will allow one to provide sharp bounds on the error parameter used in this work - τ. We are currently looking into this problem.

Our algorithm could be adapted to handle smooth freeform surfaces, synthesizing smooth boundary (i.e. C^1 or better) curves, only to yield smooth ruled surfaces. Finding the proper matching between the two boundaries to yield a C^1 developable sheet, if one exists, is still an open question. Moreover, the Hausdorff distance computation between two free-form surfaces is a necessary building block for such an extension.

A better TRS boundary extraction method can enhance the results, especially for an open model without cylindrical topology. The current scheme does not handle such open models efficiently, but we foresee little difficulty in extending it to handle them as well.

References

[1] Aspert, N., Santa-Cruz, D., Ebrahimi, T. MESH: Measuring error between surfaces using the Hausdorff distance. *Proc. of the IEEE International Conference on Multimedia and Expo* 2002 (ICME), Vol. I, pp.705-708, 2002.

[2] Attene, M., Katz, S., Mortara, M., Patane, G., Spagnuolo M., Tal A. Mesh segmentation - A comparative study. *Proc. Shape Modeling International,* July 2006.

[3] Cignoni, P., Rocchini, C., Scopigno, R. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum,* 17(2):167-174, 1998.

[4] Cohen, S., Elber, G. and Bar Yehuda, R. Matching of free-form curves. *CAD*, 29(5):369-378, 1997.

[5] Cormen, T., Leiserson, C., Rivest, R., Stein, C. *Introduction to Algorithms.* McGraw-Hill, 2001.

[6] Do Carmo, M.P. *Differential Geometry of Curves and Surfaces.* Prentice-Hall 1976.

[7] Elber G. Model fabrication using surface layout projection. *Computer-Aided Design* 27(4):283-291, 1995.

[8] Fuchs, H., Kedem, Z.M. and Uselton, S.P. Optimal surface reconstruction from planar contours. *Comm. ACM*, 20(10):693-702, 1977.

[9] Guthe, M. , Borodin, P., Klein, R. Fast and accurate Hausdorff distance calculation between meshes. *Proc. WSCG*, 41-48, 2005.

[10] http://www.card-models.com/

[11] Julius, D., Kraevoy, V., Sheffer, A. D-Charts: Quasi-developable mesh segmentation. *Computer Graphics Forum* (*Proc. Eurographics*), 24(3):981-990, 2005.

[12] Katz, S., Leifman, G. Tal, A: Mesh segmentation using feature points and core extraction, *The Visual Computer* (*Proc. Pacific Graphics*), 2005.

[13] Katz, S., Tal, A. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph.* (*Proc. SIGGRAPH*) 22(3):954-961, 2003.

[14] Kolmanic, S. and Guid, N. A new approach in CAD system for designing shoes. *Journal of Computing and Information Technology 11*(4),319-326,2003.

[15] Lai, Y, Zhou, Q., Hu, S., Martin, R. Feature sensitive mesh segmentation. *Proc. Symp. Solid and Physical Modeling,* pp. 7-16, 2006.

[16] Levy, B., Petitjean, S., Ray, N., Maillot, J. Least-squares conformal maps for automatic texture atlas generation. *Proc. SIGGRAPH* 2002, pp. 362–371.

[17] Lien, J.M, Keyser, J., Amato, N. Simultaneous shape decomposition and skeletonization *Proc. Symp. Solid and Physical Modeling,* pp. 219-228, June 2006.

[18] Mitani, J., Suzuki, H. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Trans. Graph.* (*Proc. SIGGRAPH*) 23(2):259-263, 2004.

[19] Morata, M., Patane, G., Spagnuolo, M., Falicidieno B., Rossignac J. Plumber: A method for a multi-scale decomposition of 3D shapes into tubular primitives and bodies. *Proc. Symp. Solid Modeling and Appl.,* 2004, pp. 339-344.

[20] Pinkall, U. and Polthier, K. Computing discrete minimal surfaces. *Exp. Math.* 2(1):15–36, 1993.

[21] Shatz, I., Tal, A., Leifman G. Papercraft models from meshes *The Visual Computer* (*Proc. Pacific Graphics*), 22(9):825-834, 2006.

[22] Wang, C., Tang, K. Optimal boundary triangulations of an interpolating ruled surface. *Journal of Computing and Information Science in Engineering,* 5(4):291-301, 2005.

[23] Yamauchi, H., Gumhold, S., Zayer, R., Seidel, H. Mesh segmentation driven by Gaussian curvature. *The Visual Computer,* 21:659–668, 2005.

[24] Zhang, H., Liu, R. Mesh segmentation via recursive and visually salient spectral cuts. *Proc. Vision, Modeling, and Visualization,* pp. 429-436, 2005.

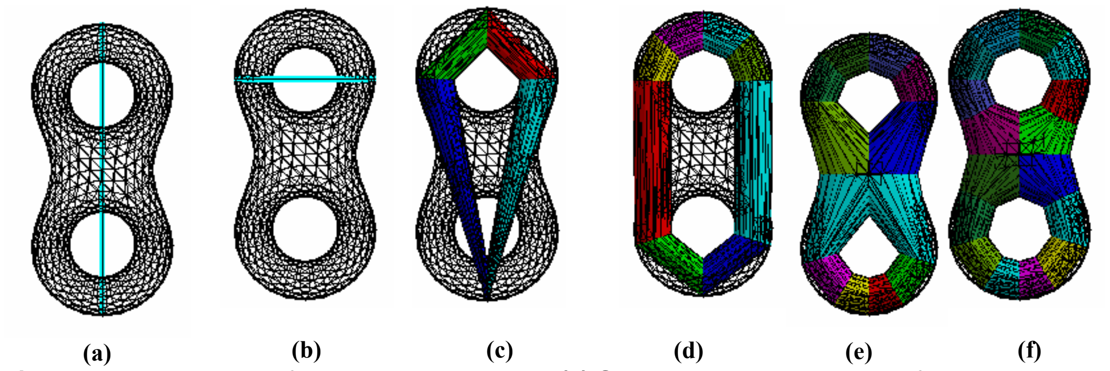


Figure 4: Approximation of the Figure Eight model. (a) Since the mesh is closed the first approximation is built between the two vertices having maximal geodesic distance. (b) The mesh is split at the maximal error and two sub-meshes created, each having two boundaries (c)+(d) Each sub-mesh from the previous stage has more than two boundaries, the best approximating TRS is chosen. (e) The Hausdorff distance on the top and bottom parts of the model do not exceed the specified threshold (2%) (f) Other parts with error exceeding the threshold are further approximated.

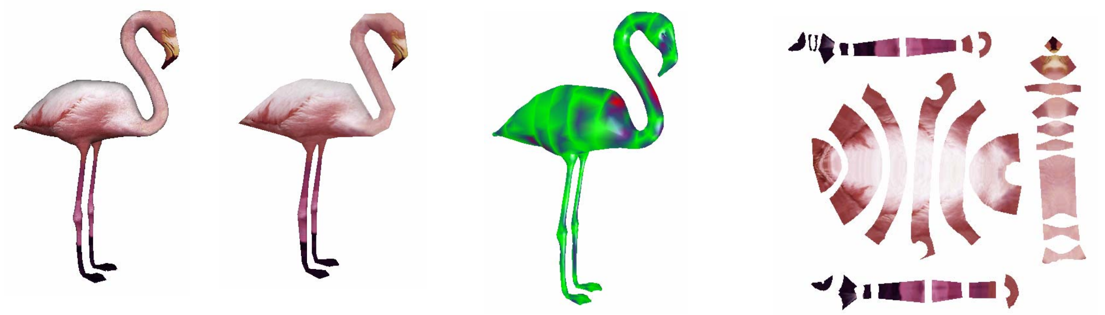


Figure 5: Flamingo model. (left to right) Input mesh. Approximation with 2% error. Color-coded approximation error. Final flat strips.

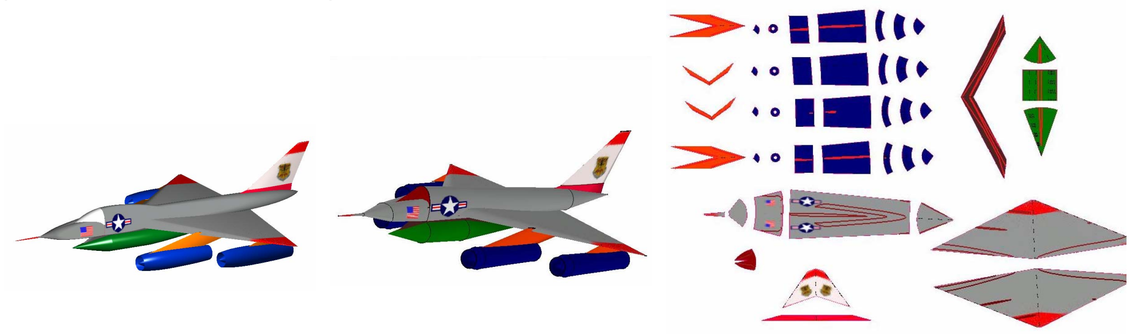


Figure 6: B58 model. (left to right) Input mesh. Approximation with 2% error. Final flat strips. This model is built mainly from cylindrical components approximated by a small number of TRS. The intersection curves and feature lines are drawn on the layout.

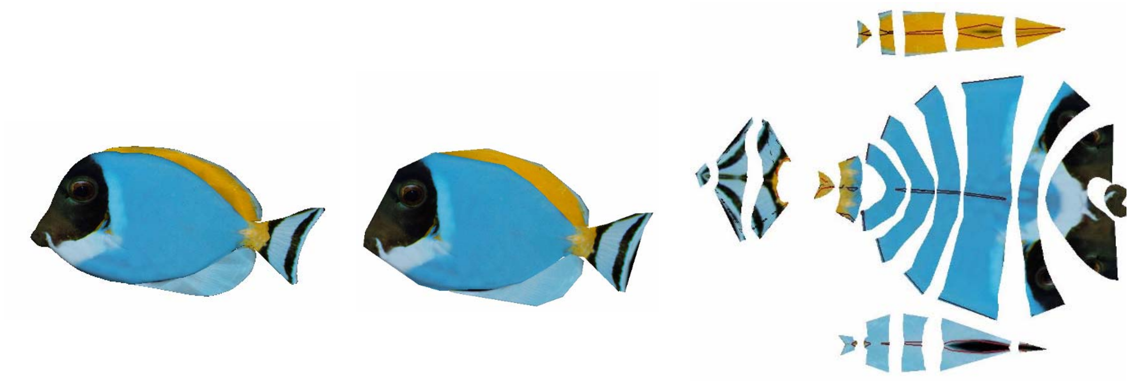


Figure 7: Fish model. (left to right) Input mesh. Approximation with 2% error. The flat strips.

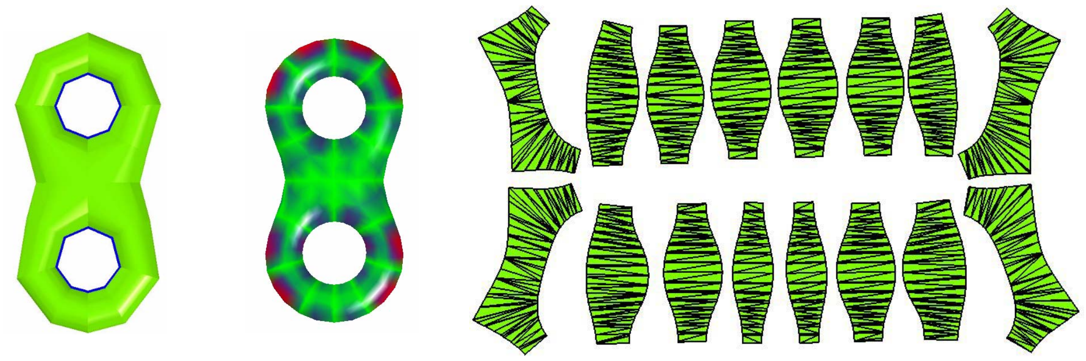


Figure 8: Figure Eight model. (left to right) Approximation with 2% error. Color-coded error map. Flat strips with triangles marked.

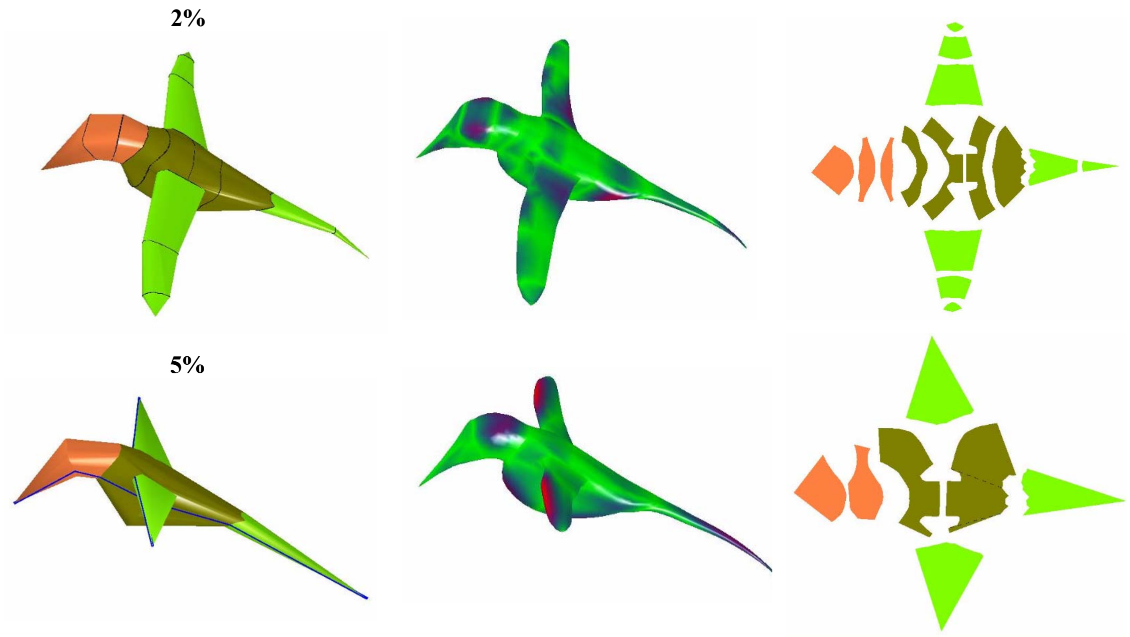


Figure 9: Bird model. (left to right, top to bottom) Approximation with 2% error. Color-coded error map. Flat strips. Approximation with 5% error. Color-coded error map. Flat strips.

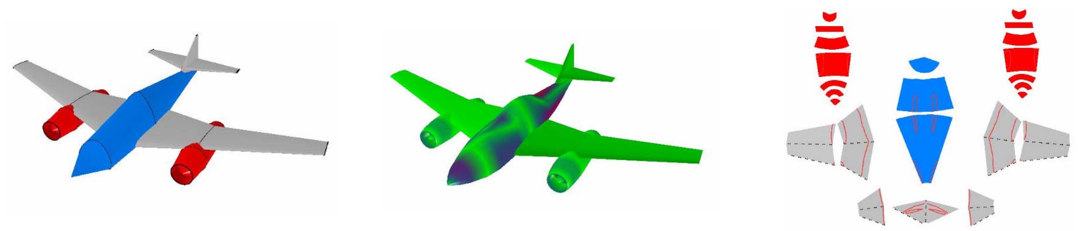


Figure 10: Me262 model. (left to right) Approximation with 5% error. Color-coded error map. Flat strips.

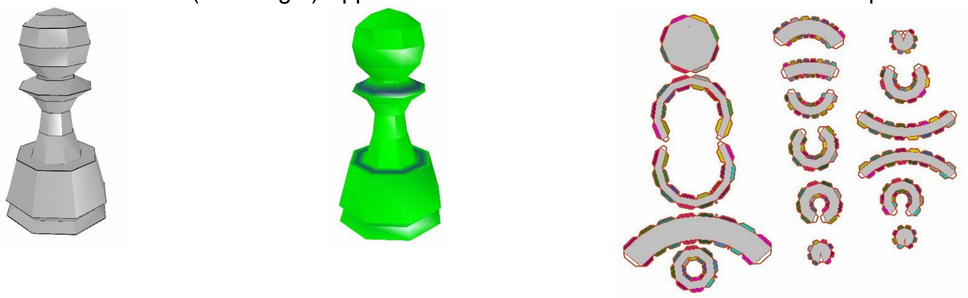


Figure 11: Pawn model. (left to right) Approximation with 1% error. Color-coded error map. Flat strips with tabs (for gluing).

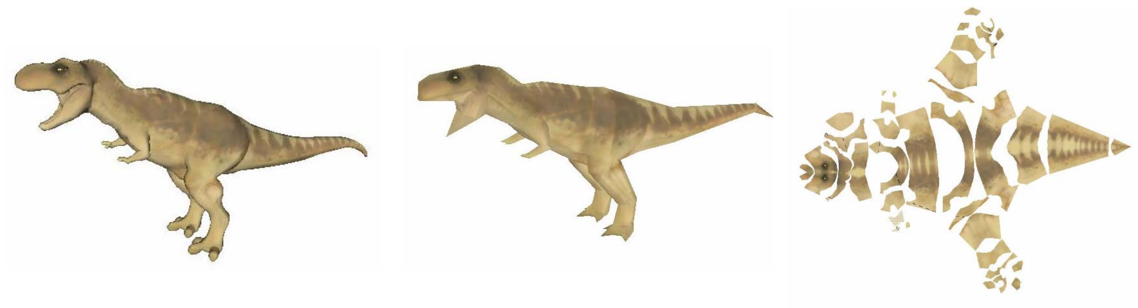
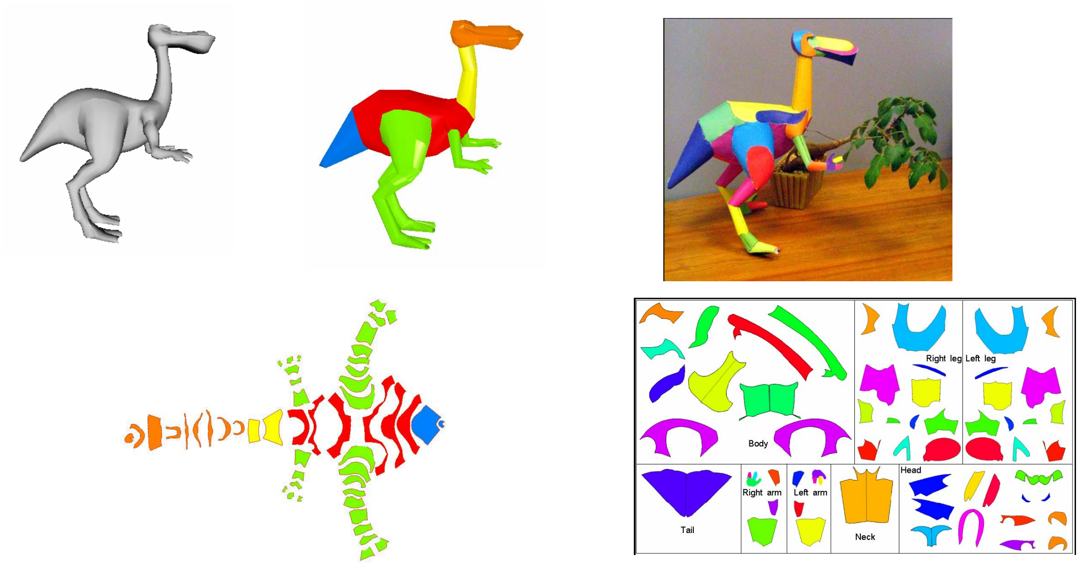
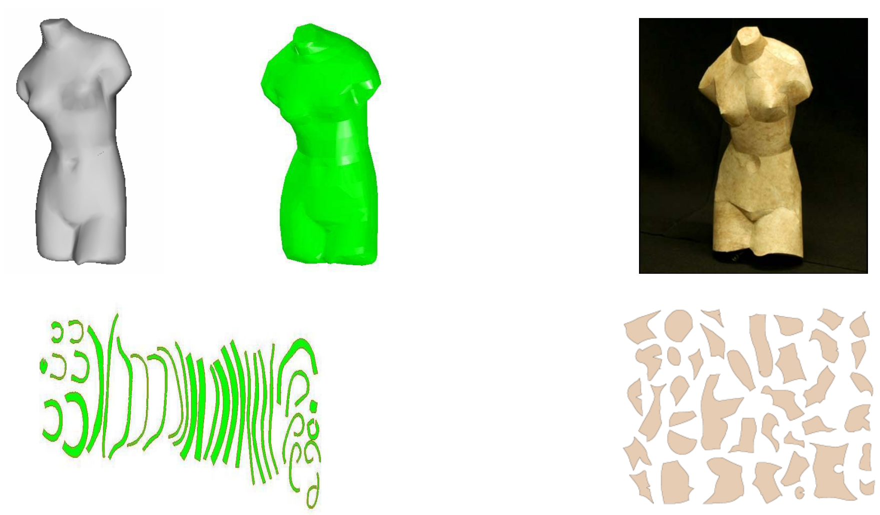


Figure 12: Rex model. (left to right) Input mesh. Approximation with 2% error. Flat strips.



Our method Shatz et al (reproduced from [21])

Figure 13: Comparison with Shatz et al [21]. (top to bottom, left to right) Input dinopet mesh. Output of our method with error of 4.9%. Flat strips. Output of Shatz et al with error of 5%. Their flat pieces.



Our method Shatz et al (reproduced from [21])

Figure 14: Comparison with Shatz et al [21]. (top to bottom, left to right) Input Venus mesh. Output of our method with error of 4.5%. Flat strips. Output of Shatz et al with error of 5%. Their flat pieces.