# Filling Holes in Triangular Meshes by Curve Unfolding

Alan Brunton[1], Stefanie Wuhrer[1,2], Chang Shu[1], Prosenjit Bose[2], Erik D. Demaine[3]

[1]National Research Council of Canada, Ottawa, Canada
[2]Carleton University, Ottawa, Canada

[3]Massachusetts Institute of Technology, Massachusetts, USA

*Abstract*—**We propose a novel approach to automatically fill holes in triangulated models. Each hole is filled using a minimum energy surface that is obtained in three steps. First, we unfold the hole boundary onto a plane using energy minimization. Second, we triangulate the unfolded hole using a constrained Delaunay triangulation. Third, we embed the triangular mesh as a minimum energy surface in $\mathbb{R}^3$. The running time of the method depends primarily on the size of the hole boundary and not on the size of the model, thereby making the method applicable to large models. Our experiments demonstrate the applicability of the algorithm to the problem of filling holes bounded by highly curved boundaries in large models.**

*Keywords*—**Computational Geometry; Object Modeling; Hole Filling; Curve Unfolding**

## 1. INTRODUCTION

The wide use of laser range sensors and other types of 3D sensing devices has produced increasingly detailed 3D models. However, holes, due to a variety of reasons, are usually present in models built from range scans. Some holes are caused by the intrinsic limitations of the sensors. Others are the result of object self-occlusion, insufficient view coverage, and shallow grazing angles. Davis et al. [7] give an account of holes and their causes in different situations.

Many methods have been proposed for hole filling. The majority of them fill holes by interpolating the nearby geometry [7]. This method is only effective when the holes are small. Manual, interactive tools are often used to fix large holes. To keep the surface smoothness the same as the nearby areas, interactive methods were proposed in a way that imitates image touch-up tools such as those found in Adobe Photoshop where users "cut and paste" surface geometry from nearby areas [22].

We propose a novel approach to automatically fill holes in triangulated models. We assume that the holes are bounded by a simply connected loop that is an *unknot*. That is, it can be continuously deformed to a circle without introducing self-intersections. The hole can be large and the boundary loop can be highly curved. We fill the hole with an approximate minimum energy surface. A *mimimum energy surface (MES)* is a surface with fixed boundary $b$ that minimizes a given

energy functional over all surfaces with boundary $b$. For the examples shown in this paper we two energy functions: the least-squares mesh function [24] and the discrete fairing energy [14]. However, other energy functionals, for example to preserve smooth change in surface normal, could be used instead. The complexity of our algorithm depends primarily on the complexity of the hole boundary instead of on the complexity of the triangular mesh. Note that this is a significant improvement over algorithms that operate on the full mesh because the complexity of the boundary is often small even for large models.

To fill a hole bounded by a loop of boundary edges, the proposed approach proceeds in three steps. In the first step, the boundary loop is gradually unfolded to a simple planar polygon. During this unfolding, we constrain the motion such that the loop does not self-intersect.

In the second step, the simple planar polygon is triangulated using a *constrained Delaunay triangulation* algorithm. The resulting triangulation has three properties. First, the triangulation contains all of the edges of the unfolded polygon. Second, the triangulation does not add any Steiner points. Third, the triangulation maximizes the minimum angle over all triangulations that have the first two properties.

In the third step, the triangulated patch is embedded in $\mathbb{R}^3$ using the known boundary positions. As the ordering of the vertices is maintained during the unfolding, the resulting patch closes the hole. We refine the patch to have the same resolution as the surrounding surface. The interior vertices are positioned to approximate a MES.

## 2. RELATED WORK

Geometric methods for filling holes in a mesh model interpolate the hole boundary or extrapolate the surface geometry from the surrounding areas. Two types of representations are used: volumetric representations and triangular meshes.

The volumetric representation discretizes the surface mesh into regular 3D grids or an octree structure either locally or globally. Davis et al. [7] diffuse the geometry from the hole boundary to the interior until the fronts meet. This method handles complex topological configurations such as holes with islands. However, it may change the existing mesh. Podolak and Rusinkiewicz [20] embed the incomplete mesh in an

octree and use a graph cut method to decide the connections between pairs of the hole boundaries. It resolves difficult boundary topologies globally. Ju [11] constructs a volume using an octree grid and reconstructs the surface using contours. Volumetric approaches work well for complex holes. However, they are time consuming. Furthermore, the topology of the generated result may be incorrect in case of large holes.

The triangle-based approaches to hole filling work directly on the surface mesh. The advantage of working directly with the surface mesh is that the rest of the surface is unchanged when the holes are filled. Let the mesh contain $n$ vertices and let the mesh boundary contain $m$ vertices. This class of algorithms usually only deal with holes bounded by a simple loop that is unknot. That is, holes with islands can usually not be filled using these algorithms.

Barequet and Sharir [2] give an $O(n + m^3)$ algorithm for triangulating a 3D polygonal boundary which represents the boundary of a hole. When triangulating the hole boundary, a divide and conquer technique is used. No new vertices are added to the mesh. Liepa [18] extends this method to include a surface fairing step. The high complexity of the algorithm limits the use of this method. Jun [12] proposes another method based on subdividing the hole into simple regions. Each simple hole is filled with a planar triangulation. This algorithm is not guaranteed to find a result. Li et al. [17] extend the method to achieve higher efficiency and stability. However, the algorithm is not guaranteed to find a result for arbitrary holes.

Dey and Goswami [8] use a Delaunay triangulation-based method, called Tight Cocone, in which tetrahedrons are labeled as in or out. In this method, no extra vertices are added to the mesh. The method is shown to perform well to fill small holes. However, the method is inappropriate to fill large holes because the geometry is extrapolated from the nearby boundary.

Carr et al. [4] use radial basis functions (RBF) to compute an implicit surface covering the hole. One RBF is computed for the full surface. Hence, the complexity of the algorithm is a function of $n$. That is, in cases where the surface is large and the hole boundary is small, the algorithm is inefficient. To overcome this problem, Branch et al. [3] extend this approach to use local RBF for each hole. Chen et al. [5] use an RBF-based approach to fill holes and recover sharp features in the hole area.

Tekumalla and Cohen [25] propose an approach that fills the hole by repeatedly using moving least squares projection. The approach iteratively adds layers of triangles onto the boundary until the hole is filled. Zhao et al. [27] fill holes in a similar way. After finding an initial triangulation by iteratively adding layers to the boundary, the position of the vertices is optimized by solving a Poisson equation. The goal of the optimization is to achieve smooth normal changes across the mesh.

Lévy [16] proposes a general technique for surface editing based on global parameterization. The method can fill holes in a surface by parameterizing the surface in the plane, filling the hole in the parameter domain, and placing the added vertex coordinates in three dimensions to approximate a MES. Unlike the previously discussed approaches, this approach can fill holes with arbitrary boundaries. However, the complexity of the algorithm is a function of $n$. That is, in cases where the surface is large and the hole boundary is small, the algorithm is inefficient. Furthermore, it is hard to ensure that no global overlap occurs during the parameterization of an incomplete mesh. If the parameterization overlaps, then the holes cannot be filled.

In this paper, we propose a novel approach to fill holes in triangular meshes. Our approach is similar to the approach by Lévy [16] in that we construct a planar parameterization. However, unlike Lévy, we do not parameterize the full mesh in the plane, but only the boundary of the hole. Note that this restricts our algorithm to operate on holes bounded by simple loops. However, if the triangular mesh and the boundaries of the holes are given, our algorithm is independent of the complexity $n$ of the mesh and depends primarily on the complexity $m$ of the hole boundary. If the boundaries of the holes are not given, our algorithm can extract the boundaries in $O(n)$ time. Furthermore, if our algorithm finds a solution, then the parameterization of the boundary is not self-intersecting.

## 3. ALGORITHM

Given a triangular manifold $S$ with $n$ vertices with partially missing data, we aim to fill the holes of $S$ by a triangular manifold meshes of minimum energy.

We first identify the boundaries of holes of $S$. Since $S$ is a manifold, we can find the edges of $S$ bounding a hole as edges that touch exactly one triangle. We fill each hole separately. Filling a hole bounded by $m$ edges with a triangulation that does not have self-intersections may require an exponential number of Steiner points in $m$ [9]. Furthermore, the problem of deciding whether such a triangulation exists is an $NP$-complete problem [1]. Hence, we do not require that the mesh avoids self-intersection. However, unlike the approach by Barequet and Sharir [2], we add Steiner points to the mesh to reduce the occurrence of self-intersections.

Our approach proceeds in three steps. First, the boundary loop is gradually unfolded to a simple planar polygon. During this unfolding, the loop does not self-intersect. Second, the simple planar polygon is triangulated using a constrained Delaunay triangulation algorithm. Third, the triangular patch is embedded in $\mathbb{R}^3$ and refined to match the resolution of the surrounding mesh. The following sections give a detailed description of these steps.

## 3.1 Unfolding the Boundary

We aim to unfold a boundary loop in $\mathbb{R}^3$ into a simple planar polygon with similar curvature as the boundary loop by gradually moving the vertices of the loop without causing any self-intersections of the loop. This is only possible if the original loop is an unknot. Hence, in the following, we assume that an unknotted boundary loop is given.

The aim is to unfold the loop to a planar polygon. Minimizing an energy that encourages all sets of four vertices on the loop to be planar is computationally expensive because it takes $O(m^4)$ time to evaluate the energy. Hence, we minimize an energy that encourages all sets of four consecutive vertices on the loop to be planar. This energy can be evaluated in $O(m)$ time. As we unfold the curve gradually, we expect that the resulting planar curve has similar curvature as the boundary loop. In particular, we expect that concavities of the curve are preserved. Maintaining concavities helps to obtain a triangulation that does not self-intersect.

Let $p_0, p_1, \ldots, p_{m-1}$ denote the $m$ vertices of the boundary loop. To unfold the boundary loop, we aim to minimize $E_{PE} = \sum_{i=0}^{m-1} d_{PE}(p_i)$ subject to the constraint $d_{MD} > \epsilon$ for an arbitrary threshold $\epsilon$. We specify these term below:

$$d_{PE}(p_i) = \angle(n_i, n_{(i+1) \bmod m}), \text{ where}$$

$$n_i = (p_i - p_{(i+1) \bmod m}) \times (p_{(i+2) \bmod m} - p_{(i+1) \bmod m})$$

and $\angle(a, b)$ denotes the angle between the two vectors $a$ and $b$. An illustration of $d_{PE}(p_i)$ is shown in Figure 1. Here, $d_{MD}$ denotes the minimum distance between any two segments on the boundary loop, which can be computed in $O(m^2)$ time. The minimum distance between two segments can be computed using dot products [15].
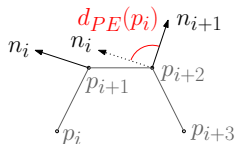


Fig. 1. *Illustration of the distance $d_{PE}(p_i)$. In the illustration, we assume that $i + 3 < m$. Otherwise, all subscripts need to be taken modulo $m$.*

Note that we can compute the gradient of $E_{PE}$ analytically, which allows to minimize $E_{PE}$ using a gradient descent approach. However, experimental evidence suggests that such an approach is likely to get trapped in a local minimum.

Therefore, we solve the minimization problem using simulated annealing [13], which proceeds by trying random steps. If the new solution achieves lower energy than the previous one, the step is always accepted. Otherwise, the step is accepted according to a probability distribution that depends on the time that elapsed since the algorithm was started. In the beginning of the algorithm, steps that increase the energy are more likely to be accepted than in the later stages of the algorithm.

This is analogous to the way liquids cool and crystallize. We use the approach outlined by Press et al. [21, Chapter 10.9] that uses the Boltzmann distribution to decide whether a step is accepted or rejected. This approach accepts a new step with probability $exp(\frac{-(E_{PE}^{t+1} - E_{PE}^t)}{kT})$, where $E_{PE}^t$ is the energy in the last step, $E_{PE}^{t+1}$ is the energy in the current step, $k < 1$ is a constant that describes the rate of cooling, and $T$ is the start temperature. We enforce the constraint $d_{MD} > \epsilon$ by restricting the maximum step size of the algorithm to $\max(\epsilon, d_{MD})$.

As outlined above, a solution that achieves lower energy than the previous one is always accepted. In this case, we choose the next random step close to the current one. Namely, we move each point along a direction within 10 degrees of the previous random direction.

After the simulated annealing step, we force the boundary loop to be planar by projecting it to its best-fitting plane. If it is possible to move the boundary loop to the projection by linear motions of the vertices without introducing self-intersections, we accept the unfolding. Otherwise, we restart the simulated annealing process. If we cannot find a solution after starting the simulated annealing process 100 times, we consider the algorithm inappropriate to fill the hole.

## 3.2 Triangulating the Planar Patch

After unfolding the boundary as outlined in the previous section, we aim to triangulate the simple planar polygon. We use the method and available code by Shewchuck [23] to complete this step. The algorithm computes a constrained Delaunay triangulation of the input polygon. A constrained triangulation of a polygon is a triangulation that is constrained to contain each of the edges of the polygon. That is, no Steiner points are added along the edges of the polygon. The constrained Delaunay triangulation of a polygon has the property that it maximizes the minimum angle over all constrained triangulations of the polygon. During the triangulation, we do not add Steiner points.

## 3.3 Embedding the Triangular Mesh in $\mathbb{R}^3$

The previous section constructs a planar triangular mesh. This section outlines how we move this mesh to the boundary of the hole to obtain a watertight model.

First, we embed the mesh in $\mathbb{R}^3$ by moving the vertices of the mesh to their corresponding vertices on the hole boundary. As the order of the vertices along the boundary polygon is maintained during unfolding, this yields a watertight model, and, as we expect concavities to be preserved during the gradual unfolding, we expect to obtain a triangulation that does not self-intersect. For the non-pathological examples shown in our experiments, no self-intersections occur.

The result of this step is similar to the result by Barequet and Sharir [2]. Our approach takes $O(m^2 c d)$ time to compute this result, where $c$ is the number

of unfolding steps during each SA run and $d$ is the number of SA runs required to find the result, while the approach by Barequet and Sharir takes $O(m^3)$ time. The number of simulated annealing steps counts how often we need to restart the simulated annealing process while the number of unfolding steps counts the number of random steps taken in one simulated annealing step.

The resolution from the filling mesh may be different from the resolution of the surrounding mesh, so we refine the mesh using the approach by Chew [6]. Steiner points are added such that the Delaunay triangulation of the added points is guaranteed to have all the angles between $30°$ and $120°$ and where the edge lengths are at most twice as long as the edges of the mesh surrounding the hole. The running time of the algorithm is linear in the number of generated triangles.

Finally, we embed the interior vertices $P_{Int}$ of the mesh, such that $P_{Int}$ minimize an energy function. In this paper, we use two energy functions: a Laplacian energy to obtain a least-squares mesh [24] and a discrete fairing energy [14]. Note that these energies can be replaced by any desired energy function. For example, different boundary conditions can be used. Or, if images are captured for texture mapping, a photo consistency energy can be minimized.

To obtain a MES using the Laplacian energy, the newly added interior vertices of the mesh filling the hole are repositioned to minimize the area of the triangular mesh subject to the boundary constraints provided by the positions of the boundary vertices. This can be achieved by repeatedly applying Laplacian smoothing because Laplacian smoothing is equivalent to minimizing the surface area [10].

To improve the efficiency of the algorithm, we formulate Laplacian smoothing as an optimization problem. For each vertex $p$ of the mesh, define

$$U(p) = \frac{1}{|N_1(p)|} \sum_{q \in N_1(p)} q - p,$$

where $N_1(p)$ is the 1-ring neighborhood of $p$ and $|N_1(p)|$ is the cardinality of the set $N_1(p)$. We aim to minimize

$$E_{AREA} = \sum_{p \in P_{Int}} (U(p))^2,$$

where $P_{Int}$ denotes the set of interior vertices of the mesh. Note that the gradient of $E_{AREA}$ with respect to $p$ can be expressed explicitly. We solve the optimization problem using a quasi-Newton method [19]. Note that this is equivalent to computing a least-squares mesh [24].

To obtain a MES using discrete fairing, we minimize a second-order Laplacian energy. For each vertex $p$ of the mesh, we compute $U(p)$ as before. Next, compute for each interior vertex of the mesh

$$U^2(p) = \frac{1}{|N_1(p)|} \sum_{q \in N_1(p)} U(q) - U(p).$$

We aim to minimize the energy

$$E_{DF} = \sum_{p \in P_{Int}} \left( U^2(p) \right).$$

We minimize this energy using an iterative approach in our experiments.

## 4. EXPERIMENTS

This section presents experiments using the algorithm presented in this paper. The experiments were conducted using an implementation in C++ using OpenMP on an Intel Pentium D with 3.5 GB of RAM. We set $k = 0.9, T = 0.5$, and $\epsilon = 10^{-5}$ in all of our experiments. In all of the figures showing holes in the models, back faces are shown in blue.

We first apply the algorithm to holes arising from the limitations of range scanners. The first experiment fills the holes present in the scan of a chicken model. The model was scanned using a ShapeGrabber laser range scanner. The scanned model contains 135233 vertices. The algorithm filled eight holes with a total of 666 vertices on the boundaries. The model along with the filled holes is shown in Figure 2. The model is grey and the filled holes are colored. Figures 2 (a) and (b) show the front and back of the chicken with five and three holes respectively. The most complex of the five holes on the front is located underneath the little chicken and shown in detail in Figure 2 (c). Another complex hole on the front of the chicken is located under the eye and shown in detail in Figure 2(d). Figures 2 (e) and (f) show large complex holes at the back base of the chicken model. Our algorithm took the longest to fill the two highly curved holes shown in Figure 2 (f). We can see that the proposed algorithm fills all of the holes present in the scan with approximate MES of similar resolution as the surrounding mesh.

The unfolding of all the holes took about 6.5 minutes. The embedding of all the holes while minimizing $E_{AREA}$ took about 3 seconds. Note that due to the high complexity of the model, we only ran the algorithm once. Hence, the running times are not averaged over multiple runs for this experiment.

The second experiment based on laser range scans fills the well-known holes present in the Stanford bunny [26]. There are 5 holes and they contain 79 vertices total. We fill the holes while minimizing $E_{AREA}$. Four of the holes before and after filling are shown in Figure 3.

We next apply the algorithm to a number of artificial holes. We created holes of large curvature in complete models to show the applicability of our algorithm in this case. The first experiment fills the hole present in the armpit of a human model. The hole boundary has high curvature. The initial hole as well as the result of our algorithm are shown in Figure 4.

The following three experiments fill holes present in the head of a human model. The first hole is shown in Figure 5. The hole is large and the hole boundary
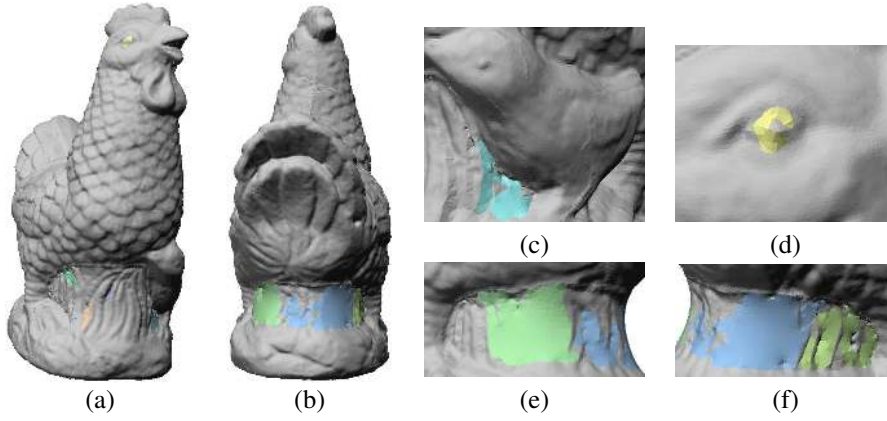
Fig. 2. *Chicken model filled by minimizing $E_{AREA}$. (a): Front view of the chicken with five filled holes. (b): Back view of the chicken with three filled holes. (c): Detail view of the filled hole under the little chicken. (d): Detail view of the filled hole under the eye. (e): Detail view of a filled large hole at the base of the model. (f): Detail views of two filled complex holes at the base of the model.*
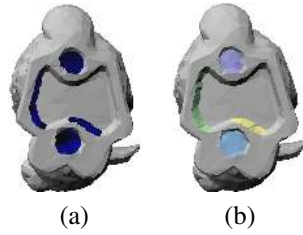


Fig. 3. *Stanford bunny model filled by minimizing $E_{AREA}$. (a): Four of the holes in the bunny model. (b): Filled holes.*
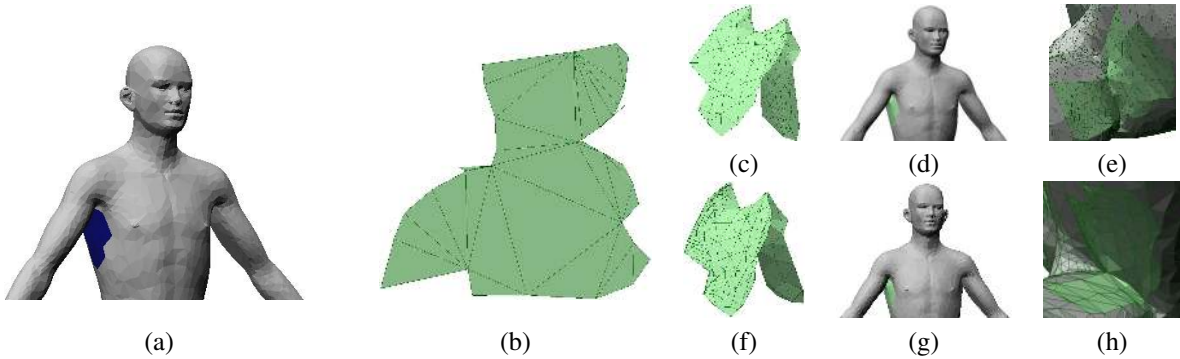


Fig. 4. *Armpit model. (a): Hole in the armpit of a human model. (b): Unfolded mesh. (c)-(e): Final embedded mesh obtained by minimizing $E_{AREA}$. (f)-(h): Final embedded mesh obtained by minimizing $E_{DF}$.*

is highly curved. Nonetheless, our algorithm finds a visually pleasing solution.

The second hole is shown in Figure 6. The hole is large. The hole boundary is highly curved and highly twisted. Nonetheless, our algorithm finds a visually pleasing solution.

The third hole is shown in Figure 7. The hole is large. The hole boundary is curved in all three dimensions. Nonetheless, our algorithm finds a visually pleasing solution.

The running times of the experiments are given in Table 1. We average the running time over 10 runs. Note that due to the random component in simulated annealing, the running times of the 10 runs vary significantly. The running time of the unfolding step depends on the number of times simulated annealing is restarted and on the number of steps required to unfold the boundary. The running time of the embedding step depends on the

number of Steiner points added to the triangular mesh that fills the hole. Note that the efficiency of the algorithm may be improved by using a more sophisticated simulated annealing technique than the one described by Press et al. [21, Chapter 10.9].

## 5. CONCLUSION

We presented a novel approach to automatically fill holes in triangulated models. The approach fills the hole using a minimum energy surface that is obtained by unfolding the hole boundary into the plane using an energy minimization approach. The planar curve is then triangulated and embedded to the three-dimensional position of the boundary loop. In this paper, we embed the triangular patch as a minimal surface. Note that this could be replaced by a prior distribution of the surface's geometry to embed the triangular patch. We leave this for future work.
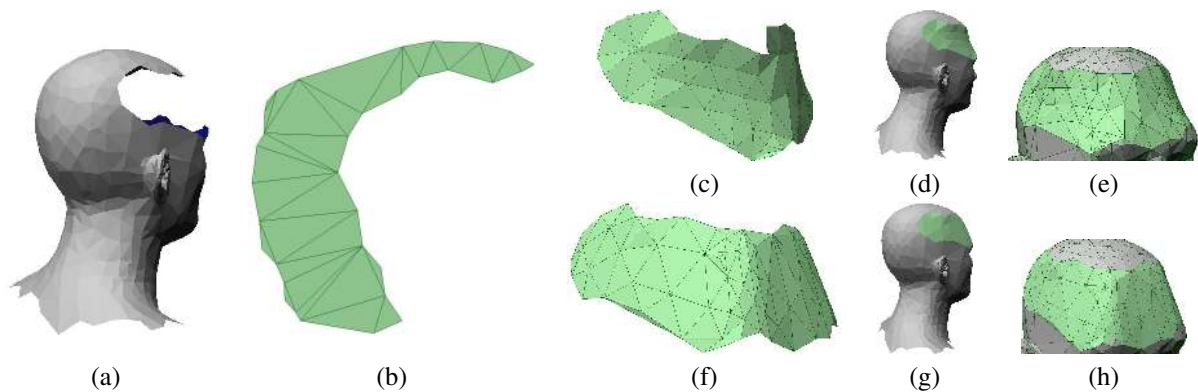
Fig. 5. *Head model 1. (a): Hole in the head of a human model. (b): Unfolded mesh. (c)-(e): Final embedded mesh obtained by minimizing $E_{AREA}$. (f)-(h): Final embedded mesh obtained by minimizing $E_{DF}$.*
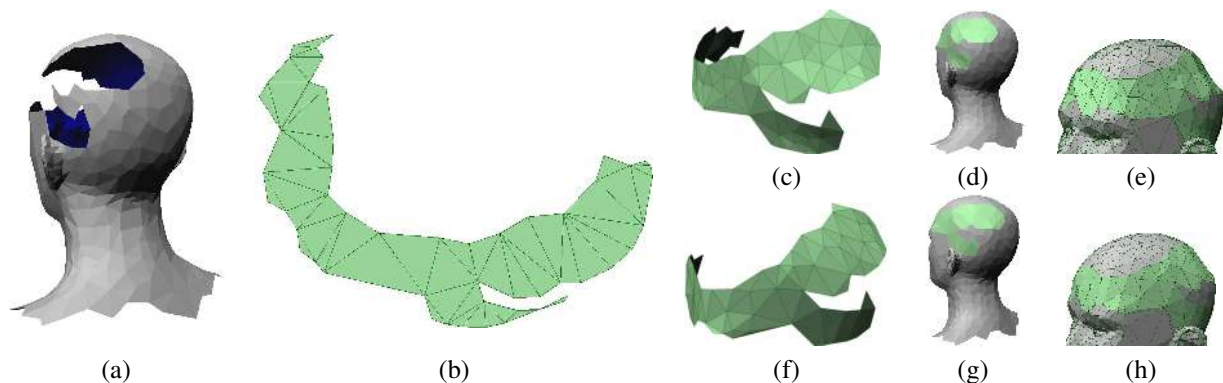


Fig. 6. *Head model 2. (a): Hole in the head of a human model. (b): Unfolded mesh. (c)-(e): Final embedded mesh obtained by minimizing $E_{AREA}$. (f)-(h): Final embedded mesh obtained by minimizing $E_{DF}$.*
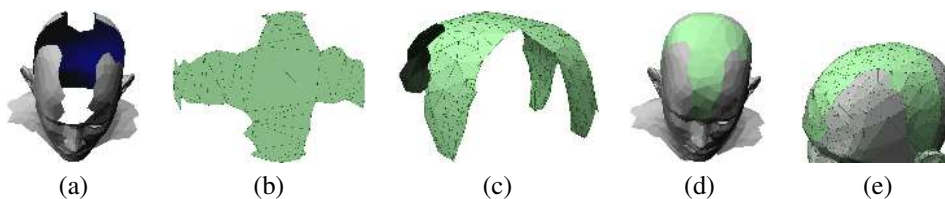


Fig. 7. *Head model 3 filled by minimizing $E_{DF}$. (a): Hole in the head of a human model. (b): Unfolded mesh. (c): Final embedded mesh. (d): Filled hole. (e): Detail view of the filled hole.*

The energy used to unfold the boundary loop encourages all sets of four consecutive vertices on the loop to be planar. This energy can be evaluated efficiently in $O(m)$ time. We use simulated annealing to minimize this energy. We leave applying more sophisticated SA variants to this problem for future work.

### REFERENCES

[1] Gill Barequet, Matthew Dickerson, and David Eppstein. On triangulating three-dimensional polygons. In *Symposium on Computational Geometry*, pages 38–47, 1996.

[2] Gill Barequet and Micha Sharir. Filling gaps in the boundary of a polyhedron. *Computer Aided Geometric Design*, 12(2):207–229, 1995.

[3] John Branch, Flavio Prieto, and Pierre Boulanger. Automatic hole-filling of triangular meshes using local radial basis function. In *Third International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT'06)*, pages 727–734, 2006.

[4] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell, Richard W. Fright, Bruce C. Mccallum, and Tim R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *ACM Transactions on Graphics (SIGGRAPH'01)*, 2001.

[5] Chun-Yen Chen, Kuo-Young Cheng, and Hong-Yuan Mark Liao. Fairing of polygon meshes via bayesian discriminant analysis. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2004.

[6] L. Paul Chew. Guaranteed-quality triangular meshes. Technical Report TR 89-983, Computer Science Department, Cornell University, 1989.

[7] James Davis, Stephen R. Marschner, Matthew Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. In *First International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT'02)*, pages 438–433, 2002.

[8] Tamal K. Dey and Samrat Goswami. Tight cocone: A water-tight surface reconstructor. In *ACM Symposium on Solid Modeling and Applications 2003*, pages 127–134, 2003.

| Model | $n$ | $m$ | $d$ | Unfolding time (s) | Embedding time for $E_{AREA}$ (s) | Embedding time for $E_{DF}$ (s) |
|-------|-----|-----|-----|-------------------|-----------------------------------|--------------------------------|
| Bunny | 1494 | 79 | 1.0 | 2 | 1 | - |
| Armpit | 9951 | 35 | 4.8 | 3 | $< 1$ | $< 1$ |
| Head 1 | 1177 | 33 | 2.0 | 2 | $< 1$ | $< 1$ |
| Head 2 | 1189 | 53 | 2.9 | 5 | $< 1$ | $< 1$ |
| Head 3 | 1157 | 63 | 16.4 | 2 | - | 5.3 |

TABLE 1

*Running times. The number of vertices in the model is denoted by $n$, the number of vertices on the hole boundary is denoted by $m$, and the number of simulated annealing steps is denoted by $d$. All running times are averaged over 10 runs.*

[9] Joel Hass, Jack Snoeyink, and William P. Thurston. The size of spanning disks for polygonal curves. *Discrete and Computational Geometry*, 29(1):1–17, 2003.

[10] Klaus Hildebrandt and Konrad Polthier. Anisotropic filtering of non-linear surface features. *Computer Graphics Forum*, 23:391–400, 2004.

[11] Tao Ju. Robust repair of polygonal models. In *ACM Transactions on Graphics (SIGGRAPH'04)*, pages 888–895, 2004.

[12] Yongtae Jun. A piecewise hole filling algorithm in reverse engineering. *Computer-Aided Design*, 37(2):263–270, 2005.

[13] Scott Kirpatrick, Dan Gelatt, and Mario Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[14] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Conference on Computer graphics and interactive techniques (SIGGRAPH'98)*, pages 105–114, 1998.

[15] Andrew Ladd and Lydia Kavraki. Using motion planning for knot untangling. *The International Journal of Robotics Research*, 23(7–8):797–808, 2004.

[16] Bruno Lévy. Dual domain extrapolation. In *ACM Transactions on Graphics (SIGGRAPH'03)*, pages 364–369, 2003.

[17] Gen Li, Xiu-Zi Ye, and San-Yuan Zhang. An algorithm for filling complex holes in reverse engineering. *Engineering with Computers*, 24(2):119–125, 2008.

[18] Peter Liepa. Filling holes in meshes. In *Eurographics Symposium on Geometry Processing*, pages 200–205, 2003.

[19] Dong C. Liu and Jorge Nocedal. On the limited memory method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[20] Joshua Podolak and Szymon Rusinkiewicz. Atomic volumes for mesh completion. In *Eurographics Symposium on Geometry Processing*, 2005.

[21] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, 1992.

[22] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. *ACM Transactions on Graphics (SIGGRAPH'04)*, 23(3):878–887, 2004.

[23] Jonathan R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *First Workshop on Applied Computational Geometry*, pages 124–133, 1996.

[24] Olga Sorkine, Daniel Cohen-Or. Least-Squares Meshes. In *International Conference on Shape Modeling and Applications (SMI)*, pages 191–199, 2004.

[25] Lavanya Sita Tekumalla and Elaine Cohen. Hole-filling algorithm for triangular meshes. Technical Report UUCS-04-019, School of Computing, University of Utah, 2004.

[26] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of SIGGRAPH'94*, pages 311–318, 1994.

[27] Wei Zhao, Shuming Gao, and Hongwei Lin. A robust hole-filling algorithm for triangular mesh. *The Visual Computer*, 23(12):987–997, 2007.

[28] Pavel Chalmoviansky and Bert Juttler. Filling Holes in Point Clouds. In M. Wilson and R.R. Martin (eds.) *The Mathematics of Surfaces X*, Lecture Notes in Computer Science, Springer, 2003, 196–212.