GridMesh: Fast and High Quality 2D Mesh Generation for Interactive 3D Shape Modeling

Andrew Nealen[1], Justus Pett[2], Marc Alexa[2,3], Takeo Igarashi[4]
[1]Rutgers University, [2]TU Berlin, [3]Walt Disney Animation Studios, [4]The University of Tokyo / PRESTO JST

Abstract—**In this paper we present an algorithm for watertight meshing of closed, sketched curves. The sketch is resampled as a piecewise linear (PWL) curve and placed onto a triangular grid. A small boundary (seed) that describes a closed path along grid points is placed inside the sketch and grown until it resembles the sketch. Vertices of the evolved grid boundary are projected onto the stroke to establish a bijective, ordered mapping. Finally, valences along the boundary are optimized while retaining the previously established mapping. The resulting mesh patch can be duplicated, stitched and inflated to generate a new shape, or used to fill a hole in an existing shape. We have implemented our algorithm in FiberMesh [1], an interactive sketch based interface for designing freeform surfaces, where it is used for the all mesh generation processes. The triangulation generated with our algorithm improves the quality of the model by reducing the number of irregular vertices, while running at real time rates.**

Keywords—**Meshing; Mesh Generation; Interactive Shape Modeling; Sketch Based Modeling; Fair Surface Design**

1. Introduction

Current tools for the design of 3D shapes and surfaces provide a vast amount of modeling operations. At the same time however their complexity has significantly increased. In related research, many tools that try to hide this complexity exist. Some of which use design metaphors like sketching [2], [3], [4], [5], [1], where, in order to create a 3D model, the planar user sketch is triangulated and inflated. In the case of FiberMesh, the mesh is generated by intersecting the sketch with a regular triangle grid [1], and thereafter inflated by means of functional optimization. In the original paper [1], the problem of mesh generation is described rather briefly. To fill this gap, we have designed a fast and robust meshing algorithm, which generates a semi-regular mesh with only few irregular vertices.

The algorithms presented in this paper are intended to improve the quality of 3D models in FiberMesh, however, they can also be easily adopted to other tools and replace existing surfacing algorithms. Our main contributions are:

- **A fast, robust algorithm for the meshing of planar sketches** (Section IV). We present an algorithm that uses a regular triangle grid to mesh a planar sketched curve in real time. Stitching the obtained patches results in a small number of irregular vertices (Figure 1).
- **A method for meshing 3D polylines which lie on developable surfaces** (Section V). We show how the algorithm can be adjusted to 3D curves that can easily be parameterized in 2D, and have a fixed number of boundary vertices. This case occurs after a cut operation, where a hole is filled with a boundary compatible triangle mesh (Figure 2).

Our approach uses a regular 2D triangle grid, as well as a closed boundary curve that passes through the grid vertices and is grown until it resembles the user sketch (Figure 1).

The general problem of meshing is widely covered in existing literature, and a comprehensive overview is given by Lamot and Zalik [6]. The first algorithm for polygon triangulation was introduced by Lennes [7], who proposed inserting diagonals between pairs of vertices of the polygon. The inductive proof for the existence of a diagonal triangulation was proposed by Meisters [8]. Thereafter, various diagonal insertion algorithms were published [9], [10], [11], [12], where the main focus lies in optimizing the worst case runtime. However, they do not measure the quality of the resulting triangulation. A Delaunay triangulation for a set of vertices is one where no other vertex lies inside the circumcircle of any triangle. Constrained Delaunay triangulation is more common than diagonal insertion approaches, because it maximizes the minimum angle of all triangles, resulting in a triangulation of higher quality [13], [14], [15]. Approaches using Steiner points add vertices inside the polygon in order to maximize the interior angle of all triangles in the triangulation. Consequently, the overall number of triangles is increased. Algorithms using Steiner points try to maximize the minimum angle while minimizing the number of additional triangles. Some of the existing implementations are based on Delaunay triangulation [16], [17], while others use different approaches [18], [19], [20].

Grid-based meshing algorithms are used in various domains such as tetrahedral mesh generation (see [21] and references therein), and have also been applied in the setting of 2D/3D shape deformation [22], [23], where they are used to create the deformation scaffolding.

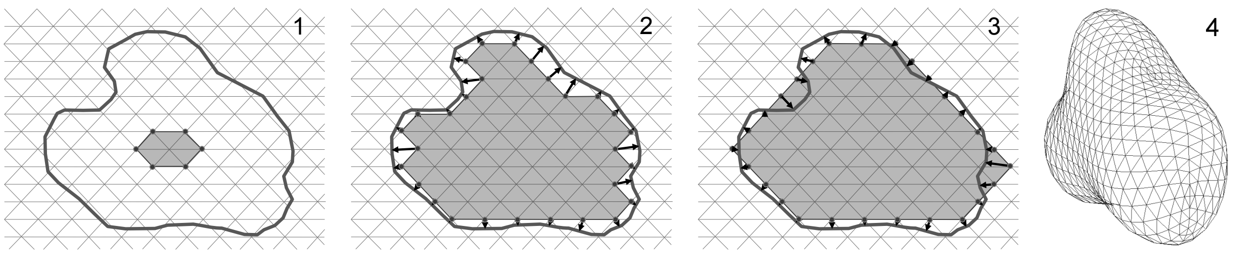

Fig. 1. Mesh generation for a sketched, planar curve: the user curve and the regular triangle grid overlay (1) with the seed 1-ring in dark gray. The seed is repeatedly grown until the sketch boundaries are reached and a bijective, ordered mapping between user curve and grid curve is established (2). Turning angles in the grid curve are optimized w.r.t. valences and resemblance of the user curve, while retaining the previous mapping (3), resulting in a watertight mesh with only few irregular vertices (4).

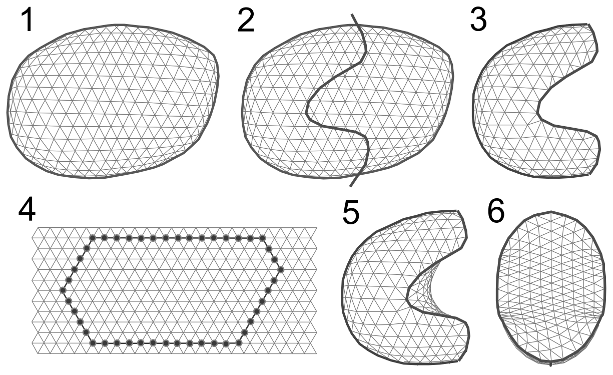

Fig. 2. The general process of cut meshing. A cut is applied to an existing 3D model (1). A cut stroke is drawn across the model (2). One part selected by the user is deleted (3). The hole is remeshed using a boundary compatible triangle grid (4). The obtained patch of triangles is stitched on the model along the cut stroke (5 and 6).

The quality of the resulting triangulation in many previous approaches is measured based on different requirements, and most do not rely on interactive/realtime response. Sketch-based modeling tools however heavily rely on interactive response, and furthermore require a *sufficient* number of triangles: the model is inflated after initial meshing, and we can achieve more smooth, visually pleasing results when using a larger number of triangles with regular vertices. In our grid-based method, the mesh resolution can be easily adjusted. This is also important, since the resolution has a direct impact on the response time of the surface optimization after a deformation operation.

2. Overview and preliminaries

We start with a short description of the general meshing procedure before describing the details. Please refer to Figure 1 for the steps described in the following. The user draws a closed curve that defines the silhouette of the mesh being created; this curve is intersected with a triangle grid (step one). A set of triangles lying inside the stroke is identified, and the boundary vertices of the retained triangle patch are mapped onto the curve (step two). Then the patch boundary is optimized w.r.t. valences (step three). The resulting patch of triangles is cloned and stitched together along the boundary (similar to a pillowcase) to obtain a watertight polyhedron. Finally, the interior vertex positions are computed, using the mapped locations on the sketched curve as positional boundary constraints (step four). FiberMesh computes the positions of interior vertices by functional optimization [1], resulting in a smooth shape (Figure 1, right).

For the robustness of the procedure, the identification of triangles is crucial: the boundary of the set of triangles must form a single and simple polygon. Note that merely identifying the triangles that are lying (fully or partially) inside the stroke could lead to disconnected, degenerate or nonmanifold patches.

If the triangles form a single, simply connected component, the quality of the resulting mesh is mainly determined by two factors: (1) the similarity to the original user sketch, and (2) the regularity of the vertices[1]. These factors can only be influenced when the curve is mapped to the triangle grid (steps 2 and 3 in Figure 1). At this point a boundary is defined on the grid that, ideally, should resemble the original user sketch. Furthermore, the grid points on the boundary are the only mesh vertices which can be irregular after cloning and stitching; all interior vertices are regular by construction.

2.1 Triangle grid and user sketch

We define a triangle grid as an isometric grid formed by tiling the plane regularly with equilateral triangles. We call the vertices of these triangles the *grid points* p_i and the set *GP*. One grid point is adjacent to six triangles. The distance d_{ij} between grid points p_i and p_j is defined as the minimum number of edges between these points. Furthermore we use the neighbors relation: $\mathcal{N} \subset GP \times GP = \{(p_i, p_j) | d_{ij} = 1\}$. Accordingly, each grid point has exactly six neighbors. When moving along a triangle grid we can define six different directions.

The process of sampling the user sketch results in a piecewise linear (PWL) curve. The samples are the curve *vertices* $(v_0, v_1, ..., v_n)$, which are connected by *segments* $(v_1 - v_0, v_2 - v_1, ..., v_n - v_{n-1})$. For closed curves there is an additional segment $v_0 - v_n$.

[1]A triangle mesh vertex is regular if it has valence six, meaning it has six incident edges

2.2 *Boundary and topology*

The boundary of a set of triangles in the grid is an ordered sequence B of grid points p_i which describes a closed path on the grid. The set of enclosed triangles forms a single, simply-connected component if the boundary is simple and closed, i.e. every grid point on the boundary appears only once in B ($p_i = p_j \Rightarrow i = j$) and the first and the last grid point B are neighbors $(p_0, p_{|B|-1}) \in \mathcal{N}$. We will call the elements of B *boundary vertices*. In Figure 3 the left image shows an example of a valid boundary, while the right image shows an example of an invalid boundary. A crucial point of the algorithm we describe later, is that it guarantees a valid boundary. This is achieved by starting with a small patch inside the user stroke, and growing the patch triangle by triangle while preserving the simple and closed boundary property.

Note that it is easy to induce a bijection between the boundary and the PWL user stroke: since both curves are simple, any parameterization over the circle of both curves is sufficient. If the two curves are close enough to each other (details on this are elaborated later) the bijection can also be established by orthogonally projecting the grid points onto the user stroke.

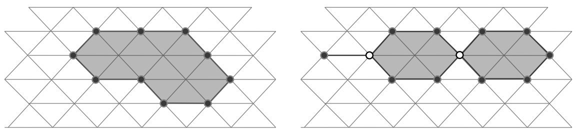

Fig. 3. Left: example of a valid boundary. Right: examples of invalid boundary. The boundary contains two loops. The white vertices would appear twice in the ordered sequence defining the boundary.

3. Mesh quality criteria

3.1 *Similarity to the original user sketch*

We want to obtain a 3D mesh with a silhouette as similar to the original curve drawn by the user as possible. However, for densely sampled curves, reproducing every detail can also become a problem. Choosing a triangle grid which allows reproducing all details of the user sketch could result in meshes with a very high resolution. In order to allow realtime interaction we therefore need to find a compromise between a good resemblance of the curve and a mesh resolution that still permits interactive feedback.

The similarity to the original user sketch is also influenced by the way the boundary points are mapped to the sketch. Depending on the mapping procedure, we may be able to preserve more details. This will be discussed in Section IV-C.

3.2 *Regularity of the mesh's vertices*

First of all, we examine how the different possible boundary configurations on the triangle grid relate to the valence of their corresponding vertices in the final watertight mesh. These configurations can be reduced to four valid cases, and one invalid case. Figure 4 shows the four valid configurations, where the inner stroke

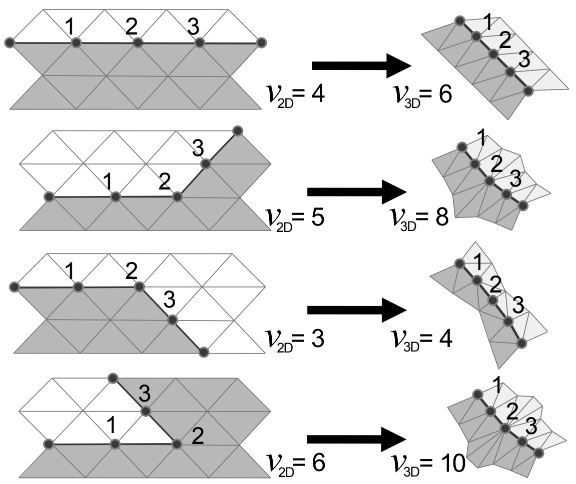

Fig. 4. The four valid boundary configurations and their resulting mesh vertices in the final watertight mesh.

area is highlighted dark gray. In the right column, the corresponding part of the 3D mesh is shown, and the duplicated part of the triangle patch is marked light gray.

If the center vertex (2) has valence v in the patch, it turns into a vertex with valence $2v - b$ in the duplicated and stitched mesh. Here, $b = 2$ is the number of incident boundary vertices. Ideally, each vertex in the mesh is regular, i.e. has valence 6, which means $v = 4$ and coincides with the case of no direction change as we move along the boundary from vertex 1 to 3. Since any vertex in a closed mesh has valence at least 3 we require $2v - b \geq 3 \Leftrightarrow v \geq (b + 3)/2$. In other words, the case of a boundary vertex in the patch on the tip of a triangle is prohibited (Figure 5). More generally, any edge that is incident upon two boundary vertices, but not part of the boundary, leads to a non-manifold situation in the final mesh. A more intricate case is shown in Figure 9, where the boundary comes close to itself and the edge connecting two boundary vertices is incident on four triangles in the resulting stitched mesh, leading to a non manifold edge. Concluding, mesh edges that are not part of the boundary, yet are connected to two boundary vertices, must be avoided.

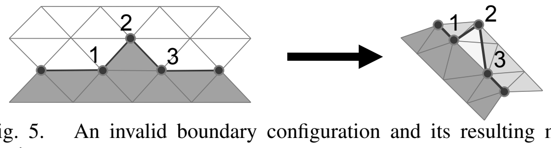

Fig. 5. An invalid boundary configuration and its resulting mesh vertices

We will attempt to minimize boundary vertices with valence different from four or, equivalently, direction changes along the boundary. However, elementary calculations using Euler's formula show that the sum of valences of all vertices on the boundary B is $4|B| - 6$, which means that not every vertex can have valence 4; we need at least 6 vertices with valence 3, and additional valence 3 vertices for each vertex with valence greater 4. Our goal is to balance this number (= vertex regularity) with geometric accuracy (= similarity to the user sketch).

4. Meshing a planar sketch

Our algorithm can be stated as follows

1) In a preprocessing step, the sketched curve is faired. This makes it easier to identify features along the curve based on curvatures, and also widens potential constrictions so that a coarser triangulation is sufficient to grow a triangle patch inside the stroke (Section IV-A).
2) We place a small initial boundary (the circularly connected 1-ring neighbors of a single vertex) inside the sketched curve and extend the patch until any further growth would cause it to intersect with the user sketch. This step ensures that a valid boundary is developed, which is close to the sketched curve (Section IV-B).
3) The grid points on the boundary of the patch are mapped to the sketched curve so as to improve the geometric accuracy of the boundary relative to the sketch (Section IV-C).
4) Vertex valences are optimized by adding additional triangles/vertices to the patch. This removes the invalid cases discussed earlier and tries to additionally remove boundary vertices with valence different from 4. The identification of boundary and sketched curve is induced by arc length parameterization of the modified parts (Section IV-D).

As a result, we obtain a boundary with an optimized number of direction changes, whose vertices have all been mapped onto the curve.

4.1 Fairing the sketch and handling narrow parts

Especially for low grid resolutions, sharp features or narrow parts in the user sketch can not be properly represented. These cases can be detected in a preprocess, and one possibility is to increase the grid resolution, such that all features can be approximated. In our system, the grid resolution is determined by the user. So, instead, we (locally) fair the user sketch to remove features which are larger than 1.5 times the grid edge length. Also, around narrow parts of a stroke, there are no triangles which lie fully inside the sketched curve. We fair the concerned parts of the curve to widen the area (Figure 6).

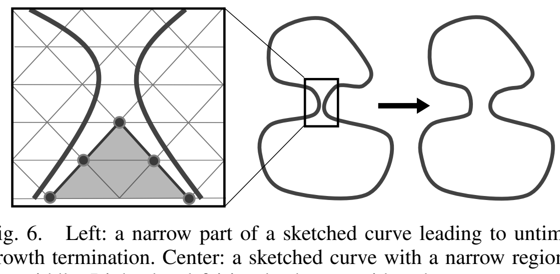
Fig. 6. Left: a narrow part of a sketched curve leading to untimely growth termination. Center: a sketched curve with a narrow region in the middle. Right: local fairing leads to a widened area.

4.2 Initialization and growing

We start with a small boundary consisting of six vertices, which is grown until the user sketch is closely resembled. The user is able to adjust the grid and therefore the mesh resolution according to smoothness requirements and the desired performance.

The boundary is grown in a breadth first manner. Expansion is terminated if a triangle edge would intersect the sketched curve. Depending on the actual requirements, the first grid point lying outside can either be added to the boundary or not. In a classic breadth first search implementation, any adjacent vertex can be traversed. Figure 7 (top row) illustrates why this is not the case in our setting; the grid point marked white may not be traversed from the current boundary, since adding it would lead to an invalid boundary. However when the boundary has grown further as depicted in Figure 7 (bottom row), adding the same point is now allowed since this leads to a valid expanded boundary. We call grid points that are valid to enqueue for a given boundary as *valid adjacent* to the boundary. A grid point is *valid adjacent* if at least two of its neighbors belong to the boundary, and none of the edges connecting the grid point to the boundary intersect the sketched curve.

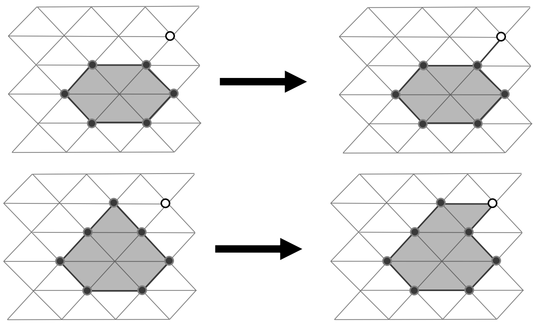
Fig. 7. Top: adding the white vertex to the given boundary (left) would invalidate it (right). Bottom: given the enlarged boundary (left) the growth expansion is now valid (right)

Note that this method is essentially equivalent to traversing triangles, which seems to be the obvious choice. But, as we will show further below, the vertex based formulation has advantages when optimizing the triangle patch for valences (Section IV-D), as well as for meshing curves with a fixed number of vertices (Section V), which is why we use it homogeneously throughout the algorithm. Due to the symmetry of the triangle grid, the number of different cases of expanding the boundary by a *valid adjacent* point can be reduced to five. These cases are illustrated in Figure 8.

In our setting of 3D modeling, narrow parts (described in Section IV-A) can be problematic after boundary growth has terminated. As noted earlier, if two boundary vertices are connected by an edge that is not part of the boundary, the resulting mesh will contain a nonmanifold edge. Therefore, after boundary growth has terminated, the boundary is tested for adjacent parts. Around all parts found, boundary growth is continued for at least one more step as shown in Figure 9.

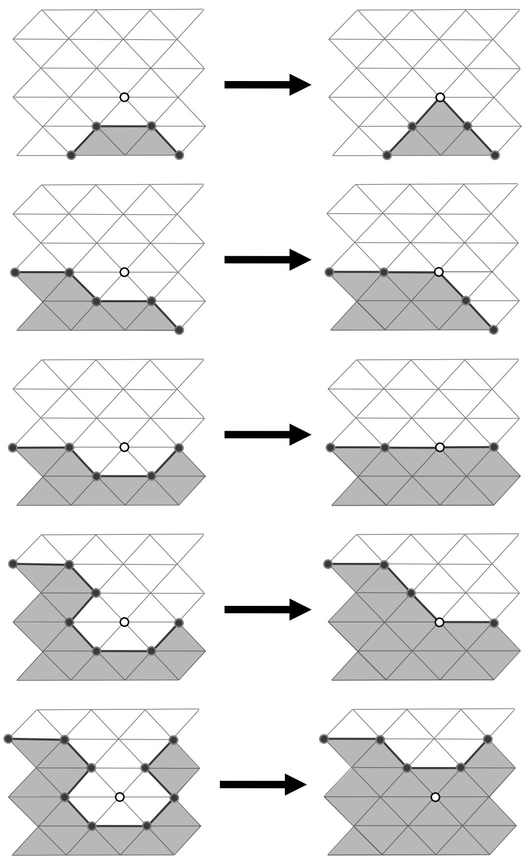
Fig. 8. The five different cases for expanding a boundary by a valid adjacent point.

4.3 Mapping boundary vertices to the sketched curve

After the boundary has been grown, the boundary grid points need to be mapped onto the user sketch, where they will serve as boundary conditions for subsequent surface optimization. If the boundary is close enough to the sketch, assigning each grid point to the closest point on the user sketch seems most plausible in order to minimize the distance of the boundary to the sketched curve. This mapping is a bijection only if the minimimum feature size of both curves is larger than $2-\sqrt{2}$ times the Hausdorff distance between them [24]. While there is no guarantee that this condition is satisfied, the resulting mapping is rarely not a bijection in practice, and the bijectivity is easy to check: the order of grid points along the boundary needs to be preserved. If we find grid points to be projected out of sequence we reassign them based on arc-length parameterization.

The number of grid points on the boundary is generally smaller than the number of vertices in the user sketch. To preserve significant features of the user sketch, such as a sharp tip, we identify these and ensure that grid points are identified with them. We consider features to be local maxima of curvature. The integrated curvature of a PWL curve can be defined as $\kappa = 2\ sin\frac{\theta}{2}$ [25] , where θ is the turning angle between two adjacent curve segments. For a pointwise measure, κ needs to be adjusted according to the length of the segments incident on the vertex.

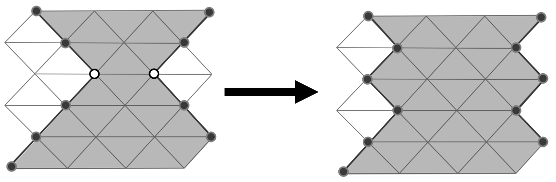
Fig. 9. Distinctive parts of the boundary on the left are adjacent (white vertices). Around this narrow area, boundary growth is continued.

From the vertices with curvature above a certain threshold, we identify maxima of pointwise curvature as significant features and order them according to their magnitude. We then take the feature points from this list as long as free boundary grid points are available and associate the closest grid point with the feature (Figure 10), unless the distance between vertex and grid point exceeds the length of a triangle edge. The remaining grid points are treated as explained above.

This mapping procedure is carried out before boundary optimization (see next section), since the optimization expands the boundary further, thereby moving it further from the user sketch. The mapping, however, must be performed when the boundary is close to the user sketch.

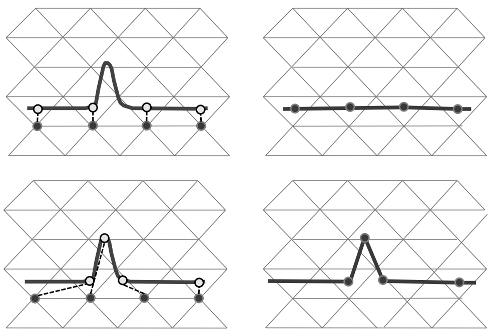
Fig. 10. Different mappings of boundary grid points to the user sketch. Top row: orthogonal projection onto the curve. Lower picture: orthogonal projection followed by snapping to feature points of the original curve.

4.4 Valence optimization

After the boundary has been grown, we have acquired a certain resemblance of the original user sketch. However the boundary is likely to be quite irregular, potentially containing the invalid cases described in Section III-B.

Therefore we grow the boundary further, in order to eliminate edges connecting boundary vertices (the cases illustrated in Figures 5 and 9) and to reduce the number of direction changes. This process may decrease the geometric proximity to the target curve, thus, the algorithm presented in the following provides the possibility to weigh the relevance of these two properties.

The optimization process grows the boundary further with different rules for the selection of expansion points and termination. In a first phase, edges connecting two boundary vertices (which are not part of the boundary) are eliminated by adding additional triangles. Note that the case of boundary vertices with valence 2 is unlikely, as the growth process is steered to avoid them. Nevertheless, remaining cases can be eliminated by adding a single triangle. When the boundary comes close to itself, the patch is grown on both sides of the boundary (Figure 9).

In a second phase, the number of direction changes along the boundary is reduced in order to improve the regularity of the resulting mesh along the boundary. It is checked if boundary expansion by a valid adjacent point decreases the total number of direction changes. For this we introduce the *dirDiff* function that measures the degree of a direction change along three boundary vertices as shown in Figure 11. Accordingly we can define an error function err_{dc}, that measures the number of direction changes along the boundary as follows:

$$err_{dc}(B) = \sum_{p \in B} \textit{dirDiff}(prev(p), p, next(p)) \tag{1}$$

where B is the boundary, and the functions $prev$ and $next$ return the predecessor and the successor of a point in B.

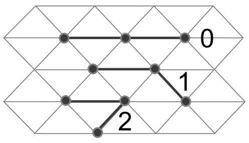

Fig. 11. The *dirDiff* function measures the degree of direction change along 3 boundary vertices. If no direction change occurs, the function returns 0, otherwise the number of turns is returned (1 or 2).

Using the function err_{dc}, we can easily determine if adding a point p to the boundary would result in a boundary with fewer direction changes. We only expand the boundary B by a valid adjacent point p if the resulting boundary B' has a decreased number of direction changes, i.e. $err_{dc}(B') < err_{dc}(B)$. We can also allow $err_{dc}(B') \leq err_{dc}(B)$, if we want to favor more triangles inside the boundary. This eliminates valence 6 vertices on the boundary (case 4 in Figure 4 with $v_{3D} = 10$), by removing the center vertex 2 and connecting vertices 1 and 3. Another example for an expansion that improves boundary straightness is given in Figure 12.

Growing the boundary terminates when no more decrease of err_{dc} can be achieved. For a convex boundary, this is the case when $err_{dc}(B) = 6$. However, for highly non-convex user-sketches, this optimal solution is likely to lack any resemblance to the sketch (the

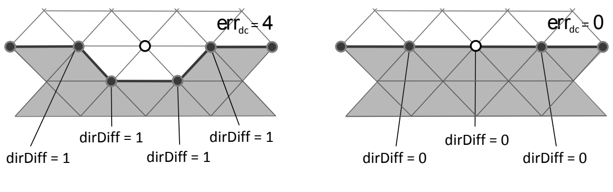

Fig. 12. The valid adjacent vertex marked white is a candidate for expansion. The original boundary (left) has 4 direction changes, while the new boundary has none.

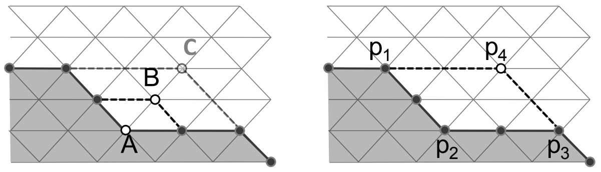

Fig. 13. Overcoming local minima of err_{dc}. See the text for details.

example in Figure 1 shows a non-convex user sketch where $err_{dc}(B) = 6$ can be achieved while staying close to the sketch). Therefore, as an additional constraint, we only allow expansion within a maximum distance from the user sketch. By adjusting the maximum distance, we are able to balance the boundary straightness against its similarity to the user sketch. A small value will lead to boundaries that are very similar to the user sketch, while a higher value will generate an optimal solution in terms of direction changes.

There is still a special case left that needs to be treated: due to the stepwise boundary expansion, the algorithm is not able to overcome a local minimum of err_{dc}. Figure 13 illustrates this: for the current boundary A, $err_{dc}(A)$ is lower than $err_{dc}(B)$. Accordingly, the algorithm will not grow the boundary around the vertex marked white, because no local optimization can be accomplished, even though growing to the final boundary C would result in an optimal boundary w.r.t. err_{dc}. This case can easily be identified by searching for a specific pattern of direction changes along the boundary (Figure 13, right): we need to search for a convex vertex (p_1), followed by a concave vertex (p_2), followed by convex vertex (p_3). Between these vertices, an arbitrary number of vertices with no direction change may occur. The boundary needs to be grown to point p_4 in order to reach the global minimum of err_{dc}, a case which can be easily detected. Note though that the boundary is only expanded if the distance of p_4 to the user sketch is smaller than the maximum distance.

The position of boundary points added during optimization on the sketched curve cannot be determined by closest point matching as described in Section IV-C, since we are moving further away from the sketched curve. Therefore, we compute the mapping of boundary points added during optimization by arc length parameterization from adjacent, unmodified points mapped previously (Section IV-C).

5. Meshing a cut or a hole

Figure 2 shows the steps of filling a hole in the mesh after a cut operation. In general, the process of cutting is similar to that of the initial meshing. However some requirements and constraints differ. In contrast to the initial stroke, the cut stroke generally does not lie on a plane. Since the cut curve always lies on a developable (ruled) surface, we can compute an isometric, planar mapping by simply "unrolling" the curve onto a plane. Furthermore, the number of boundary vertices on the triangle grid is fixed, since the patch needs to be attached to the cut stroke residing on the mesh.

Once we have obtained a planar version of the cut stroke, the meshing process can be handled similarly to the initial meshing process described in section IV with some modified constraints.

The patch cut out of the grid has to be attached to the cut stroke by merging the boundary vertices and the cut stroke vertices. Note that there exist various boundaries of the same length that enclose a different number of triangles, see Figure 14. To ensure a smooth approximation of the hole, we attempt to maximize the area enclosed by the boundary while preserving its prescribed length.

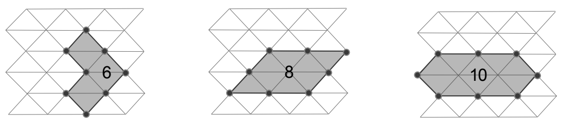

Fig. 14. Three boundaries with the same length of 8 vertices, each enclosing a different number triangles.

Given these constraints, we adapt the meshing algorithm as described in the following. The queue for boundary expansion is prioritized; candidates for expansion are ordered by their distance to the cut/hole curve. Grid points inside and far away from the curve receive the highest priority. For points outside the curve the priority is lowest and decreases further, as their distance to the curve increases.

Once again, the five different cases of boundary expansion are shown in Figure 8. The boundary length only increases in the first case. In all other cases, the boundary length remains the same or decreases. It is therefore guaranteed that the target length will be reached, since in every step of growth the length can increase at most by one. We make use of the fact that the last three cases of growing decrease the boundary length, in order to maximize the enclosed area, and achieve a more pleasing approximation of the cut surface.

Boundary expansion is performed until the target length is reached. At this point the boundary is expanded by all valid adjacent points that decrease the boundary length again (growing cases three, four and five) and lie within a predefined maximum distance to the curve. In our implementation a maximum distance of two edge lengths of the triangle grid represent a good compromise between proper resemblance and more triangles. If the boundary length was successfully decreased, expansion continues until the target length is reached again.

6. Results

In the following we present some results of our mesh generation algorithm. All figures have been taken using our FiberMesh implementation. We show the final mesh together with the curve sketched by the user, the triangle grid, and a wireframe view. On the triangle grid, all points visited during growth are marked dark gray. The final boundary is black. Grid points that have been added during boundary optimization are marked light gray. The center point enclosed by the initial boundary is marked white. Figure 15 shows a complex shape with many concavities along the silhouette.

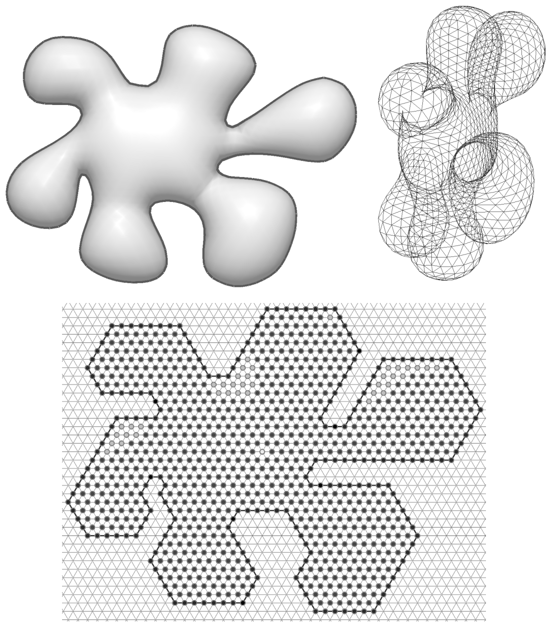

Fig. 15. A shape with many concavities along the silhouette.

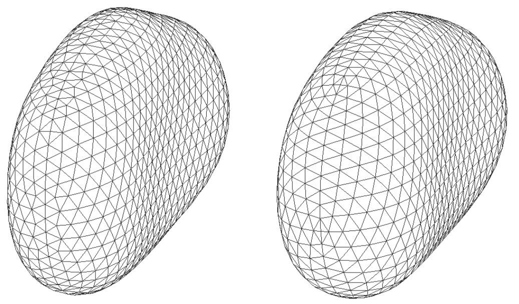

Fig. 16. Comparison with the previous meshing routines of FiberMesh (left: FiberMesh algorithm, right: our algorithm)

In Figure 16 the results of the previous prototype implementation of FiberMesh are compared to our algorithm. Both meshing algorithms were initialized with exactly the same sketched curve. It can clearly be seen that the previous approach results in more irregular mesh vertices, not only on the boundary but also in its vicinity.

Figure 17 shows a simple flat cut with the corresponding patch that was cut out of the triangle grid, while Figure 18 demonstrates the robustness of this method in the FiberMesh test case.

7. Discussion

We have presented an algorithm for meshing closed sketched curves using a triangle grid. Various properties of the resulting mesh can be easily influenced through different parameters: the number of triangles can be adjusted via the resolution of the triangle grid, and we have introduced a trade-off between geometric accuracy and mesh regularity.

We believe that this way of meshing a curve is intuitive and still provides various possibilities for further

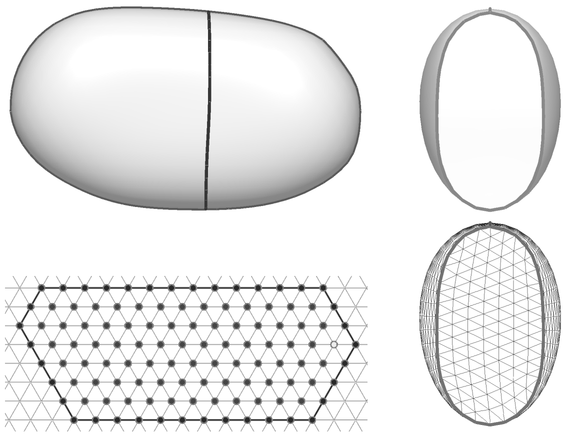

Fig. 17. Simple flat cut

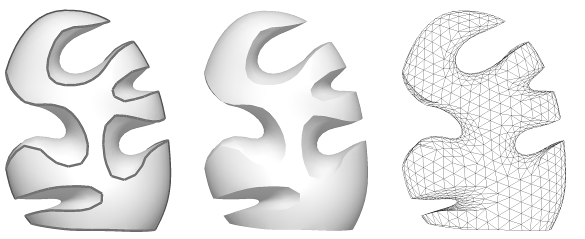

Fig. 18. Cuts inspired by modern sculpture.

improvements and new features. The algorithm could for example be extended in order to determine the best grid orientation for a given curve. Currently the results can vary significantly depending on the orientation of the curve relative to the grid. Even though the quality of the current results is good enough for our requirements, this behavior is clearly undesirable and exposes a possibility to increase the number of regular mesh vertices, and improve their locations on the silhouette.

The algorithm is furthermore adapted for a set of different constraints. It is possible to influence the number of triangles being cut out. This way of remeshing parts of an existing mesh is not only interesting for the FiberMesh application or similar sketch based interfaces using cut operations. It can also be applied to general meshes that need to be remeshed along an enclosed area. The cut stroke required for the algorithm can be obtained by defining a closed path along the mesh's vertices, which encloses the area in need of remeshing. However, the geometry of the inner vertices still needs to be computed if the algorithm is used outside the FiberMesh environment.

We believe that our algorithm provides a fast, robust and general method for meshing closed curves, including curves with a fixed number of vertices. Growing a boundary and optimizing it w.r.t. valences is fast, robust, easy to implement, and can easily be modified to meet other constraints and requirements.

References

[1] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. FiberMesh: Designing freeform surfaces with 3d curves. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 26(3), 2007. article No. 41.

[2] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. In *ACM SIGGRAPH*, pages 409–416, 1999.

[3] Takeo Igarashi and John F. Hughes. Smooth meshes for sketch-based freeform modeling. In *ACM Symposium on Interactive 3D Graphics*, pages 139–142, 2003.

[4] Olga A. Karpenko and John F. Hughes. SmoothSketch: 3D free-form shapes from complex sketches. *ACM Trans. Graph.*, 25(3):589–598, 2006.

[5] Levent Burak Kara and Kenji Shimada. Sketch-based 3D shape creation for industrial styling design. *IEEE Computer Graphics and Applications*, 27(1):60–71, 2007.

[6] Marko Lamot and Borut Zalik. An overview of triangulation algorithms for simple polygons. In *International Conference on Information Visualisation IV*, pages 153–159, 1999.

[7] N. J. Lennes. Theorems on the simple finite polygon and polyhedron. *Am. J. Math.*, 33:37–62, 1911.

[8] G.H. Meisters. Polygons have ears. *Amer. Math.*, 82:648–651, 1975.

[9] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *IPL: Information Processing Letters*, 7:175–179, 1978.

[10] Bernard Chazelle. A theorem on polygon cutting with applications. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 339–349, 1982.

[11] David G. Kirkpatrick, Maria M. Klawe, and Robert Endre Tarjan. Polygon triangulation in O(n log log n) time with simple data structures. *Discrete & Computational Geometry*, 7:329–346, 1992.

[12] Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry*, 1:51–64, 1991.

[13] B. A. Lewis and J. S. Robinson. Triangulation of planar regions with applications. *Computer Journal*, 21:324–332, November 1978.

[14] L. de Floriani and E. Puppo. An on-line algorithm for constrained delaunay triangulation. *Graphical Models and Image Processing*, 54(4):290–300, July 1992.

[15] Jonathan Richard Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *FCRC '96/WACG '96: Selected papers from the Workshop on Applied Computational Geormetry, Towards Geometric Engineering*, pages 203–222, 1996.

[16] P. Chew. Guaranteed-quality triangular meshes. Technical Report 89-893, Cornell University, Computer Science, 1989.

[17] Jim Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[18] Brenda S. Baker, Eric Grosse, and Conor S. Rafferty. Nonobtuse triangulation of polygons. *Discrete Comput. Geom.*, 3(2):147–168, 1988.

[19] Marshall Bern and David Eppstein. Mesh generation and optimal triangulation. Technical Report P92-00047, Xerox PARC, 1992.

[20] Marshall Bern, Scott Mitchell, and Jim Ruppert. Linear-size nonobtuse triangulation of polygons. In *SCG '94: Proceedings of the tenth annual symposium on Computational geometry*, pages 221–230, 1994.

[21] François Labelle and Jonathan Richard Shewchuk. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 57, 2007.

[22] Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Large mesh deformation using the volumetric graph laplacian. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 496–503, 2005.

[23] Yanlin Weng, Weiwei Xu, Yanchen Wu, Kun Zhou, and Baining Guo. 2d shape deformation using nonlinear least squares optimization. *Vis. Comput.*, 22(9):653–660, 2006.

[24] Frédéric Chazal, André Lieutier, and Jarek Rossignac:. Projection-homeomorphic surfaces. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 9–14, 2005.

[25] Eitan Grinspun, Mathieu Desbrun, Konrad Polthier, Peter Schröder, and Ari Stern. Discrete differential geometry: An applied introduction. SIGGRAPH 2006 Course Notes, `http://ddg.cs.columbia.edu`, 2006.