Engineering with Computers (2001) 17: 326–337
© 2001 Springer-Verlag London Limited

Parameterization of Faceted Surfaces for Meshing using Angle-Based Flattening

A. Sheffer and E. de Sturler

Computational Science and Engineering Program, University of Illinois at Urbana Champaign, Urbana, IL, USA

Abstract. *We propose a new method to compute planar triangulations of faceted surfaces for surface parameterization. In contrast to previous approaches that define the flattening problem as a mapping of the three-dimensional node locations to the plane, our method defines the flattening problem as a constrained optimization problem in terms of angles (only). After applying a scaling that derives from the 'curvature' at a node, we minimize the relative deformation of the angles in the plane with respect to their counterparts in the three-dimensional surface. This approach makes the method more stable and robust than previous approaches, which used node locations in their formulations. The new method can handle any manifold surface for which a connected, valid, two-dimensional parameterization exists, including surfaces with large curvature gradients. It does not require the boundary of the flat two-dimensional domain to be predefined or convex. We use only the necessary and sufficient constraints for a valid two-dimensional triangulation. As a result, the existence of a theoretical solution to the minimization procedure is guaranteed.*

Keywords. Flattening; Parameterization; Triangulation

1. Introduction

Most algorithms for meshing of three-dimensional surfaces, with either quadrilateral or triangular elements, are more efficient if a two-dimensional parameterization of the surface is available. In fact, given a spacing function which preserves the surface metric structures (lengths, angles, etc.), we can carry out the entire mesh generation procedure in the two-dimensional parametric domain, and project the resulting mesh to the three-dimensional surface. This makes the procedure much more efficient. Moreover, we generally obtain higher quality meshes if we use a two-dimensional parametric domain, because better quality guarantees exist for two-dimensional algorithms than for general three-dimensional surface algorithms [1].

Another important use for surface parameterization is anisotropic meshing (e.g. [2,3]). The only way to provide a smooth control spacing function across a mesh domain is by providing a proper domain parameterization. Unless the surface has an analytic description, it does not have an inherent parameterization. Hence, providing a two-dimensional surface parameterization becomes an essential preprocessing step for anisotropic meshing.

When meshing CAD models the analytic surface representation can often be used to provide the parameterization. However, analytic representation is not available when meshing composite surfaces, virtual topology [4], or faceted models. For all of those models we can provide an alternative surface representation using a tessellation that describes the surface as an unstructured triangular mesh. This description can easily be generated for models from any source. It can be used as the basis for the parameterization algorithm described below. For parameterization purposes the triangulation may be coarse and have lax quality constraints.

A two-dimensional parameterization of faceted surfaces has many other applications. These applications include texture mapping [5–7], surface reconstruction, multiresolutional analysis [8], formation of ship hulls, generation of clothing patterns [9], and metal forming.

An algorithm for two-dimensional parameterization of tessellated surfaces first constructs a two-dimensional mesh with a similar connectivity to the three-dimensional surface. Then, a parametric function is defined between the two-dimensional mesh facets and their three-dimensional counterparts.

Multiple approaches for parameterization of tessellated surfaces have been suggested. McCartney

Correspondence and offprint requests to: A. Sheffer, Computer Science dept., Technion, Technion City, Haifa 3200, Israel. E-mail: sheffa@cs.technion.ac.il

et al. [9] suggest a heuristic approach for triangulation flattening. The method is based on optimal local positioning of projected nodes, based on a sequential addition of the nodes. It is efficient and produces good results for nearly planar surfaces. However, the method does not guarantee the preservation of the metric structure of the two-dimensional mesh or even its validity.

Eck *et al.* [8] suggest the use of harmonic maps to generate the two-dimensional projection of the three-dimensional mesh. The algorithm produces approximations of good quality, and provides an accurate mapping function. A major disadvantage of the method is that it requires the boundary of the two-dimensional mesh to be predefined and convex. Another drawback is that the method does not guarantee the validity of the resulting flat mesh (i.e. it can generate inverted elements).

Floater [10] describes a parameterization method that computes the positions of the nodes in the flat mesh using the solution of a linear system based on convex combinations. Floater is the first, to our knowledge, to formally define what types of surfaces can be parameterized and to provide a parameterization method that is provably correct for any such surface. Similarly to the algorithm in Eck *et al.* [8], the method requires the boundary of the two-dimensional mesh domain to be predefined and convex.

Marcum [11] introduces the use of finite-element techniques to compute the locations of the flat mesh nodes. The method computes the boundary as part of the solution, using an iterative procedure where the boundary and interior are recomputed in separate consecutive steps. However, no validity guarantees are provided for the method.

A large amount of research on providing a parameterization for tessellated surfaces has been done in the context of computer graphics, since parameterization is required to generate non-distorted texture mappings. Some papers like Haker *et al.* [5] provide a mapping to a sphere, which is less useful in the context of mesh generation. Zigelman *et al.* [6] provide a method for flattening surfaces using multi-dimensional scaling. It computes the two-dimensional domain boundaries as part of the solution. The method does not guarantee the validity of the resulting flat mesh. Levy and Mallet [7] suggest a method that gives the user control on the spread of the distortion across the flattened mesh. The method uses some of the formulations introduced in Floater [10], and as a result has similar limitations.

1.1. Contribution

We [12] presented a new method for parameterization of tessellated three-dimensional surfaces via a mapping to a two-dimensional domain. The procedure is based on the observation that for a triangular mesh preserving the size of the angles on each of the faces is sufficient to maintain the surface metric structures up to a global scaling factor. Therefore, the method defines the flattening problem in terms of angles. The algorithm minimizes the relative deformation of the angles in the plane with respect to their counterparts in the three-dimensional surface, while satisfying a set of constraints on the angles that ensure the validity of the flat mesh. In order to account for the 'curvature' at each node in the three-dimensional surface (the angles around an interior node do not sum to 2π), we apply a scaling to the angles in the three-dimensional surface, and we measure the deformation relative to these scaled angles. Our minimization problem is entirely formulated in terms of the angles; the locations of the mesh nodes do not play a role. Note that after the angles have been computed, the two-dimensional mesh is fully determined after fixing the position of one interior node and the length and direction of one edge connected to that node. This formulation does not require the two-dimensional boundary to be predefined and does not place any restrictions on the boundary shape or the surface curvature. At the same time the solution method based on the formulation is provably correct as shown below. The result of the procedure is a valid two-dimensional mesh, which maintains the original surface connectivity and minimizes the distortion of the mesh angles resulting from the mapping to a plane.

In this work, we provide a more detailed description of the method and address several aspects not handled before. Those include: proof of convergence and validity of the resulting mesh; prevention of boundary intersection; and application of the method to meshing of tree-dimensional surfaces. We provide several examples of parameterization and surface meshing using the generated parameterization.

The main advantages of our method are that: (1) the method provides a parameterization for any surface for which a parameterization exists; (2) we prove that for any such surface a solution to the optimization problem exists and that the numerical algorithm converges to a solution; (3) the method guarantees that the resulting parameterization is valid; (4) we compute the boundary of the two-dimensional domain as part of the projection procedure; we do not need to define the boundary in advance; (5) the two-dimensional domain can have any shape; specifically, it does not have to be convex; (6) the robustness of the method is not affected by the input mesh quality due to the use of angles in the formulation; (7) the angle-based optimization avoids the scale problems often associated with working on meshes with different feature sizes, i.e. the tolerances and error measurements do not need to be modified when different mesh element sizes are used.

2. Algorithm

In order to preserve the surface metric structures (up to a global scaling factor) of a triangular mesh it is sufficient to maintain the sizes of the angles for each face. Therefore, we formulate the mesh flattening problem in terms of the flat mesh angles. The mapping to the plane is computed through a constrained minimization problem based on this formulation. The constraints are the necessary and sufficient requirements for a valid two-dimensional mesh, and we minimize the (relative) deviation of the angles from their *optimal* two-dimensional projections.

Our procedure for surface parameterization for meshing has three stages:

1. Solve the constrained minimization problem (defined below).
2. Check for intersections of the boundary. If a boundary intersection is found, augment the constraints and solve the augmented minimization problem again (starting with the current solution).
3. Compute the mesh spacing function based on the ratios between the areas of triangles in the three-dimensional surface and their counterparts in the flat surface.

Using the parameterization a surface mesh is generated by first meshing the two-dimensional domain using the spacing function and then projecting the resulting mesh to the three-dimensional surface.

We will now discuss each parameterization stage in detail.

2.1. The Constrained Minimization Problem

This section consists of two parts. First we will define the constrained minimization problem, and then we will discuss its solution.

2.1.1. Definition of the Constrained Minimization Problem

We use the following notation to define the objective function to be minimized and the constraints. The index i always indicates faces, the index j indicates angles inside a face, and the index k indicates nodes.

- $f_i, i = 1 \dots P$, are the triangulation faces (either in the flat mesh or in the original mesh, as will be clear from the context).
- $\alpha_i^j, i = 1 \dots P$, $j = 1, 2, 3$, are the flat mesh angles; the angles in a face are numbered counterclockwise in increasing order. The vector of all angles is denoted α.
- β_i^j, $i = 1 \dots P$, $j = 1, 2, 3$, are the corresponding original mesh angles.
- $N_k, k = 1 \dots M$, are the mesh nodes, where $k = 1 \dots M_{int}$ ($< M$) indexes the interior nodes.
- $\alpha_i^{j(k)}$ is the angle in face f_i at node N_k.
- $\beta_i^{j(k)}$ is the angle in the original mesh corresponding to $\alpha_i^{j(k)}$.

In the following, we implicitly assume the obvious relations among the indices that derive from the connectivity of the mesh. So when we write $\Sigma i\ \beta_i^{j(k)}$, the index i runs only over those faces that contain node N_k. The sum is therefore over all angles adjacent to node N_k. The objective function (to be minimized) is defined by

$$F(\alpha) = \sum_{i=1}^{P} \sum_{j=1}^{3} (\alpha_i^j - \phi_i^j)^2 w_i^j \tag{1}$$

where ϕ_i^j is the *optimal angle* for α_i^j in the two-dimensional mesh, and the $w_i^j > 0$ are weights. Our standard initial choice for the weights is $w_i^j = (\phi_i^j)^{-2}$. We derive the *optimal* angles ϕ_i^j from the angles β_i^j by computing a scaling factor per node:

$$\phi_i^j(k) = \begin{cases} \beta_i^{j(k)} \dfrac{2\pi}{\Sigma_i \beta_i^{j(k)}}, & N_k \text{ is an interior node,} \\ \beta_i^{j(k)}, & N_k \text{ is a boundary node,} \end{cases} \tag{2}$$

where N_k is the mesh node to which the face f_i is attached at the corner j. Since the input mesh is supposed to be valid, we assume that

$$\beta_i^j \geq \varepsilon_1 > 0 \tag{3}$$

where ε_1 is an arbitrarily small (input) parameter. The following constraints are necessary and sufficient to ensure that the resulting mesh is valid:

(1) $g_{i,j}^{(1)} \equiv \alpha_i^j \geqslant \varepsilon_2 > 0$, for $i = 1 \dots P, j = 1 \dots 3$, and some $\varepsilon_2 > 0$;
(2) $g_i^2 \equiv \alpha_i^1 + \alpha_i^2 + \alpha_i^3 - \pi = 0$, for $i = 1 \dots P$;

(3) $g_k^{(3)} \equiv \Sigma_i\ \alpha_i^{j(k)} - 2\pi = 0$, for N_k: $k = 1 \ldots M_{int}$ (interior nodes);

(4) $g_k^{(4)} \equiv \dfrac{\Pi_i \sin(\alpha_i^{j(k)+1})}{\Pi_i \sin(\alpha_i^{j(k)-1})} - 1 = 0$, for N_k: $k = 1 \ldots M_{int}$, where $j(k) + 1 = 1$ if $j(k) = 3$ and $j(k) - 1 = 3$ if $j(k) = 1$. The symbol Π indicates the product of its arguments.

The precise meaning of the variable ε_2 will be made clear in Section 3. We note that our constraints do not avoid potential crossing of the boundary edges. We discuss our solution to this potential problem in Section 2.2. The first two constraints deal with the validity of individual faces. Constraint (1) maintains the orientation of a face (up or down) with respect to the mesh, and (2) ensures that each face is valid. Constraints (3) and (4) are necessary to ensure the topological validity of the mesh, because the connectivity of the mesh is not an explicit constraint.

To explain this we introduce the following terminology. We define the *wheel* associated with an interior node as the set of all faces that share this interior node. We refer to the edges in the wheel connected to the interior node as the spokes.

Constraint (3) guarantees that after fixing the position of an interior node and the direction of one spoke, going over all spokes in counterclockwise order around the wheel, all shared edges (spokes) of neighboring faces coincide. Constraint (4) guarantees that after fixing the length of one (arbitrary) spoke, going over all spokes in counterclockwise order around the wheel, that the length of the last spoke (coinciding with the first) agrees with the length of the first spoke. Note that, since we only compute angles, the lengths of the intermediate spokes can be made to agree by stretching or shrinking the faces appropriately. So we only need one constraint. Going over all interior nodes, it is easy to see that the interior of the mesh is valid. The necessity of the third constraint is obvious. The necessity of the fourth constraint is best illustrated by Fig. 1, where the constraint is violated. Since the connectivity of the mesh is not explicit in the computation of the angles of all faces, we must make sure that the (arbitrarily chosen) first face in a wheel and the last face agree on the length of the shared edge.

The fourth constraint is derived as follows. Consider an interior node as shown in Fig. 1, and assume we fix the edge length l_1. Now from the angles α_1^1 and α_1^2 in face f_1 we can compute l_2 using the relation

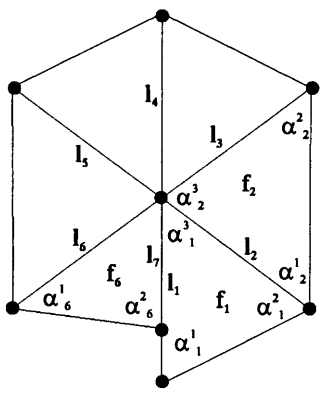

Fig. 1. Incompatibility of edge length in a wheel.

$$\frac{l_1}{l_2} = \frac{\sin\alpha_1^2}{\sin\alpha_1^1} \tag{4}$$

Using l_2 and the angles α_2^1 and α_2^2 in face f_2 we can compute l_3 to obtain an equation analogous to (4)

$$\frac{l_2}{l_3} = \frac{\sin\ \alpha_2^2}{\sin\ \alpha_2^1} \tag{5}$$

Combining (4) and (5), we can relate l_3 to l_1

$$\frac{l_1}{l_3} = \frac{l_1 l_2}{l_2 l_3} = \frac{\sin\ \alpha_1^2}{\sin\ \alpha_1^1}\frac{\sin\ \alpha_2^2}{\sin\ \alpha_2^1} \tag{6}$$

Applying (4) for each face as we go in a counterclockwise direction around the wheel, we can compute $l_4, l_5, \ldots, l_7$. However, for the mesh to be valid we need $l_7 = l_1$. Using an equation analogous to (6) to relate l_7 to l_1 we get

$$\frac{l_1}{l_7} = \frac{l_1}{l_2}\cdot\frac{l_2}{l_3}\ldots\frac{l_6}{l_7} = \frac{\sin\ \alpha_1^2}{\sin\ \alpha_1^1}\cdot\frac{\sin\ \alpha_2^2}{\sin\ \alpha_2^1}\ldots\frac{\sin\ \alpha_6^2}{\sin\ \alpha_6^1} = 1 \tag{7}$$

which demonstrates the necessity of our fourth constraint.

2.1.2. *Solution of the Constrained Minimization Problem*

We solve the constrained minimization problem as follows. As argued in Section 3, for any valid input to our algorithm a valid planar mesh exists. Since the *optimal angles* ϕ_i^j are strictly positive and our objective function measures the relative distance between the angles α_i^j and the *optimal angles* ϕ_i^j, we may prevent the algorithm from making angles too small by increasing the weight in the objective function for those angles rather than explicitly preventing the algorithm from moving outside the feasible set. This corresponds to a change in the norm that measures the distance to our optimal angles. Moreover, because of their simple form, the inequality constraints are very easy to check. Therefore, we formulate the optimization problem without explicitly taking the inequality constraints into account. If the optimization algorithm makes a certain angle too small we reject the iterate, adjust the weight of that angle, and continue.

We use a standard Lagrange multiplier formulation for the optimization problem with the equality constraints. The auxiliary objective function then becomes

$$F(\alpha) + \sum_{i=1}^{P} \lambda_i g_i^{(2)}(\alpha) + \sum_{k=1}^{M_{int}} \mu_k g_k^{(3)}(\alpha) + \sum_{k=1}^{M_{int}} \nu_k g_k^{(4)}(\alpha) \quad (8)$$

We use Newton's method to solve for a stationary point of the auxiliary function that satisfies the equality constraints. We modify Newton's method as indicated above to make sure we satisfy the inequality constraints. In most of our examples Newton's method converges in a few iterations and we do not need to adapt the weights. Although more analysis is needed we conjecture this happens for the following reasons: (1) the equations are linear, apart from those derived from constraint (4); (2) we start with the *optimal* angles as an initial guess, which is close to the solution. The sparse linear systems of equations that arise in Newton's method are solved by a preconditioned iterative solver. We use either GMRES(m) [13] or BiCGSTAB [14]; as preconditioner we use Saad's ILUT preconditioner [15]. The results provided in Table 1 were computed using BiCGSTAB.

2.1.3. Node Placement

After the algorithm computes the angles of the projected mesh, we compute the placement of the mesh nodes on the $z = 0$ plane. As stated earlier the angles of a two-dimensional mesh define the mesh up to a linear transformation (translation, rotation or scaling). Hence, by placing the end nodes of a single mesh edge in the plane, we fully define the placement of the other nodes, based on the mesh angles.

The procedure is done as follows:

- Choose a mesh edge $e^1 = (N_a^1, N_b^1)$.
- Project N_a^1 to (0, 0, 0).
- Project N_b^1 to ($\|e^1\|$ 0, 0).
- Push e^1 on the stack S.
- While S not empty, pop an edge $e = (N_a, N_b)$. For each face $f_i = (N_a, N_b, N_c)$ containing e:
  - If f_i is marked as *set* continue.
  - If N_c is not yet projected to the $z = 0$ plane, compute its position based on N_a, N_b and the face angles α_i^1, α_i^2 and α_i^3.
  - Mark f_i as *set*, push (N_a, N_c) and (N_b, N_c) on the stack.

The node placement gives us the parameterization of the three-dimensional surface.

2.2. Preventing Boundary Intersections

The set of constraints defined in Section 2.1.1 guarantees the validity of the flat mesh at any interior node. However it does not prevent the flat surface from generating self-intersections at the boundary. It is also very difficult, if at all possible, to predict in advance when the basic flattening procedure described above will lead to self intersections. Hence in this work we take the approach of handling self-intersections as a post-processing step. A flat surface mesh is generated and then tested for self intersections, by checking intersections of boundary edges.

If an intersection exists the following procedure is performed (Fig. 2):

- Given the two intersecting edges $e^1 = (N_a, N_b)$ and $e^2 = (N_c, N_d)$, find a list of nodes forming the interior loop of the intersection $N_l^1, N_l^2, \ldots, N_l^m$ where $N_l^1 = N_b$, $N_l^m = N_c$ (N_a and N_d are outside the loop).
- For each concave node N_l on this loop compute a new angle θ_l such that using those angles the intersection will no longer occur. The computation of θ_l is explained below.
- Add to the system of equations above an additional constraint $g_l^{(5)}$ for each concave node N_l: $g_l^{(5)} \equiv \sum_i \alpha_i^{j(l)} - \Theta_l = 0$ (In Fig. 2 this means adding a constraint for all the loop nodes except N_l^4).
- Solve the combined constrained system of equa-

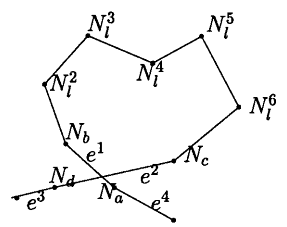

Fig. 2. Intersection boundary.

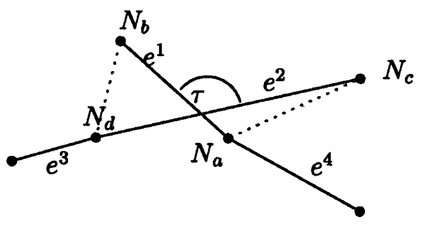

Fig. 3. Zoom-in on the intersecting edges. Note that the choice of $\tilde{\tau}$ should try to anticipate the intersection of e^3 and e^4.

tions using the previous solution as the initial guess for the solver.

The angle θ_c is computed so as to remove the point of intersection between the intersecting edges e^1 and e^2. This can be done by reducing the intersection angle τ between the edges (Fig. 3) to angle $\tilde{\tau}$. To avoid the intersection of the edges it is sufficient to define $\tilde{\tau}$ less than the minimum of $\angle((N_b, N_a), (N_c, N_a))$ and $\angle((N_c, N_d), (N_b, N_d))$ (Fig. 3). However, this does not prevent an intersection to occur between another pair of edges on two sides of e^1 and e^2. A conservative approach is to make $\tilde{\tau}$ equal zero, i.e. make the two edges parallel. This makes the intersection between the adjacent edges (e^3 and e^4 in Fig. 3) very unlikely, but can lead to unnecessary distortion (Fig. 4(d)). An intermediate value can be based on the trade-off between distortion and the number of iterations necessary to fix intersections.

The desired $\tilde{\tau}$ is achieved by setting θ_l for each concave N_l on the interior loop to:

$$\theta_l = \sum_i \alpha_i^{j(l)} (1 - c) + \pi c$$

where

$$c = \frac{\tau - \tilde{\tau}}{L\pi - \Sigma_l (2\pi - \Sigma_i \alpha_i^{j(l)})},$$

L is the number of concave nodes in the loop, Σ_l goes over those nodes and the $\alpha_i^{j(l)}$ are the angles at nodes N_l in the current solution.

To avoid all boundary intersections it may be necessary to repeat the procedure more than once. And theoretically avoiding an intersection in one part of the model may generate an intersection in another place. In practice intersections are very rare and a single iteration with additional constraints is sufficient if they do occur. Examples of intersection treatment are shown in Figs 4 and 5.

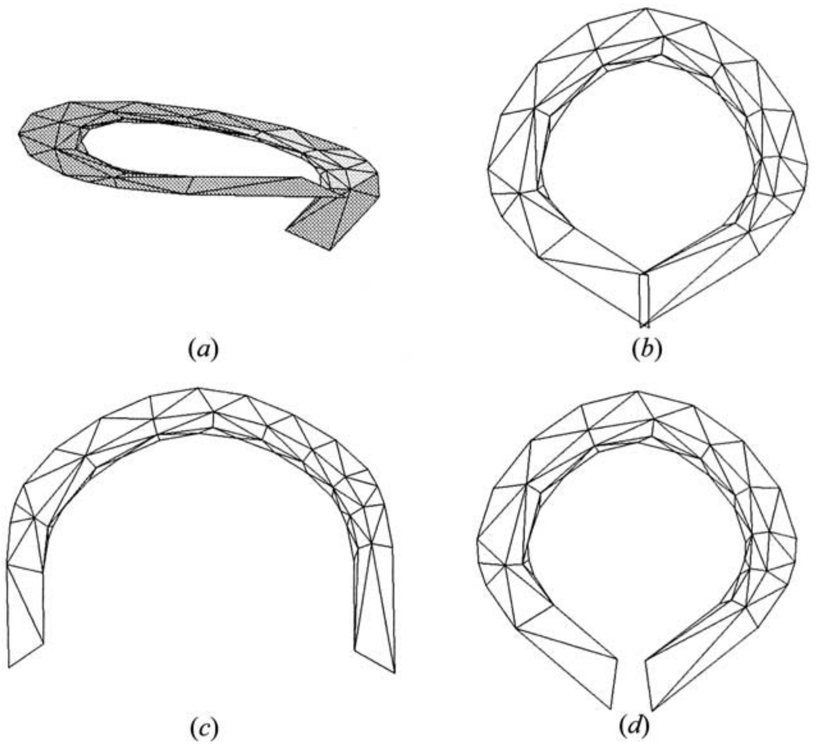

Fig. 4. Flattening of a spiral surface. (a) Original surface, (b) initial flattening (no intersection prevention), (c) preventing intersection by enforcing the original intersecting edges to be parallel, (d) preventing intersection by moving the intersection point beyond the length of the two edges.

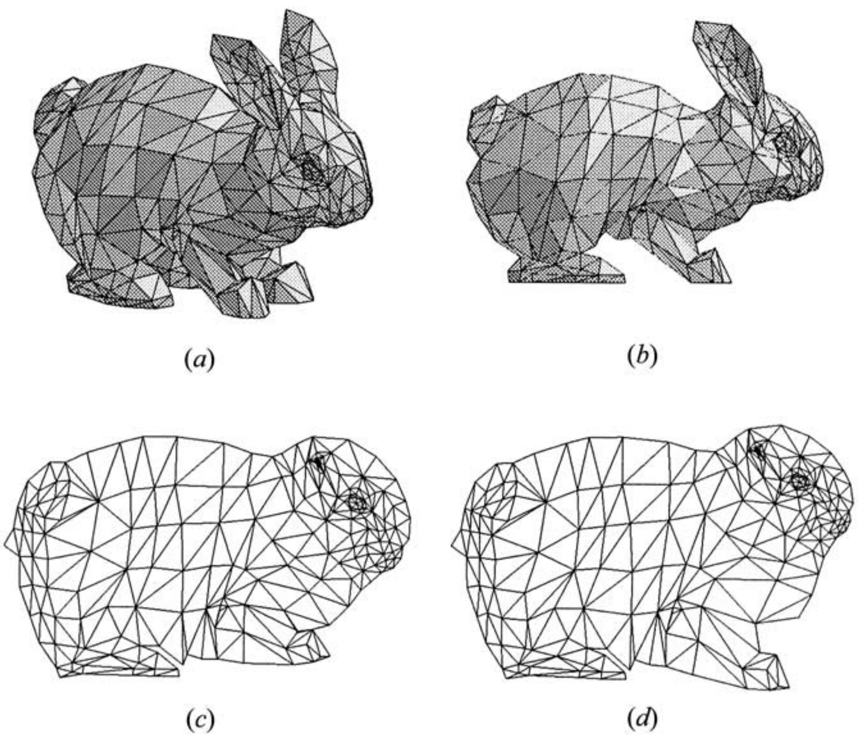

Fig. 5. Flattening a rabit model. (a) Full model, (b) the surface of half a rabit being flattened, (c) initial flattening (no intersection prevention). The model intersects near the front leg, (d) final flattening, with no intersections.

2.3. Spacing Function and Surface Meshing

After a flat triangulation has been generated, the surface can be meshed by first meshing the two-dimensional flat domain and then projecting the mesh back to the original surface. Since the flattening procedure causes some mesh distortion, a spacing function has to be used when generating the two-dimensional mesh to account for the distortion.

Since the parameterization procedure presented above minimizes the deformation of the angles, the main purpose of the spacing function is to preserve area. Hence, we can use an isotropic spacing. The spacing function S is computed as follows:

- For each f_i, $i = 1 \dots P$ compute the distortion ratio as $R_i = \sqrt{\frac{2D\ Area(f_i)}{3D\ Area\ (f_i)}}$, the ratio of the face area in the flat mesh to the area of the face in the original surface.
- For each N_k, $k = 1 \dots M$ compute the distortion ratio at the node R_{N_k} as the average of face distortions at it.
- Now we can compute the spacing function at each point p in the domain. For each point p we locate the mesh triangle $f_i = (N_a, N_b, N_c)$ it belongs to. We compute $S(p)$ using the barycentric coordinates of p in f_i. For $p = N_a u + N_b v + N_c w$, $u + v + w = 1$ we set $S(p) = R_{N_a} u + R_{N_b} v + R_{N_c}\ w$.

This spacing function scaled by the desired mesh size is then used as an argument to a two-dimensional mesh generator. In our examples we used the triangulation code described in Li *et al.* [3].

Due to the use of the spacing function when generating the two-dimensional mesh, the final surface mesh has a uniform spacing. If a non uniform spacing is desired, the two spacing functions can be combined to achieve the desired mesh.

An additional feature added to the meshing procedure is preservation of significant surface mesh features. This is achieved by explicitly adding the surface nodes that have very high curvature to the mesh node set. For this purpose we define the (discrete) curvature at node N_k as

$$C_k = \frac{\Sigma_i \beta_i^{j(k)}}{2\pi}.$$

The (discrete) curvature measures the deviation of the sum of the angles at an interior node in the surface from the sum of the corresponding angles in the plane. In the surface meshes in Fig. 6 the tips of the cat's ears are preserved this way.

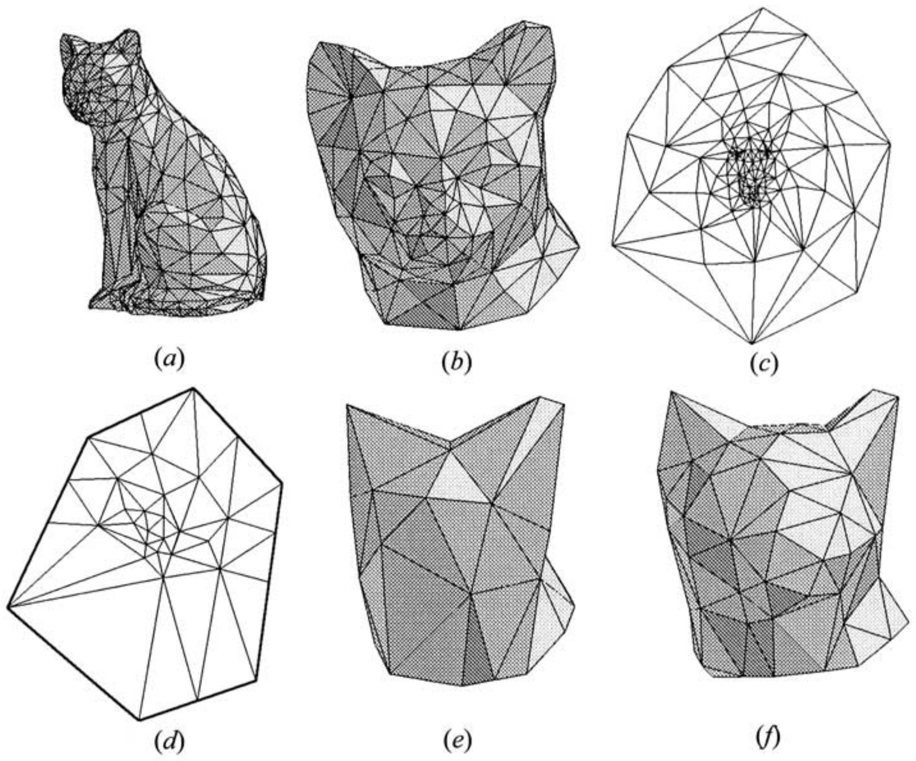

Fig. 6. Remeshing a cat head model. (a) Complete model, (b) the head (256 elements), (c) flat parameterization, (d) (e) coarse remeshing: (d) mesh in parameter space, (e) surface mesh with 53 elements. (Note that the extreme points at the ears are preserved); (f) medium size surface mesh (118 elements).

3. Proof of Algorithmic Correctness

To define the input to the algorithm and prove the existence of a solution to the constrained minimization problem defined above, we use the following standard definitions from graph theory [16]. A (simple) graph is defined as $G = (V, E)$, where $V = \{i: i = 1 \dots M\}$ are the graph nodes and E the graph edges. $E = \{(i, j), i \neq j\}$ is a subset of all unordered pairs of nodes.

A planar embedding of a graph G is defined as follows:

- each node i is mapped to a point in $\mathbb{R}^2$,
- each edge (i,j) is mapped to a curve whose endpoints are i and j,
- the only intersections between curves are at the common endpoints.

A graph G is said to be *planar* if it has a planar embedding.

A *surface triangulation* [10] $S = S(G, X)$ is an embedding in $\mathbb{R}^3$ of a triangulated planar graph G, with node set $X = \{x_i : x_i \in \mathbb{R}^3, i = 1 \dots M\}$, with straight lines for edges and triangular facets for faces.

We require the input to our algorithm to be a *surface triangulation*. Intuitively, a three-dimensional triangular surface mesh is a *surface triangulation* if it is manifold, has a single boundary loop and has no *tunnels*. Figure 7 shows a surface mesh which is not a *surface triangulation*. By definition if a triangular surface mesh is not a *surface triangulation*, it does not have a valid two-dimensional parameterization, since a parameterization is a planar embedding.

In fact as mentioned in the previous section, we allow the introduction of input meshes which have more than one boundary loop, allowing a single outside boundary and multiple interior boundary

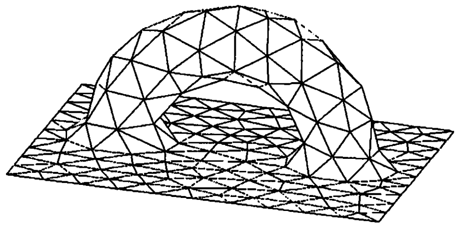

Fig. 7. A manifold surface with a single boundary loop, which is not a *surface triangulation*.

loops. In such a case, the interior loops are triangulated by connecting their boundary nodes prior to starting the minimization procedure (e.g. Fig. 10(b)). However, if the (straightforward) triangulation of an interior loop produces an invalid surface mesh, the algorithm will fail. A more robust approach to triangulating interior loops is an area of further research.

Fary's Theorem [17] states that every planar graph has an embedding in $\mathbb{R}^2$ in which all edges are straight line segments. This means that for every valid input to our algorithm a valid flat triangulation exists. Since our constraints for a valid triangulation are necessary and sufficient, the solution space for the minimization procedure is not empty.

Next, we want to show that our objective function has at least one local minimum on the closure of the constrained set. If the nonlinear function is smooth, globally convergent extensions of Newton's method are guaranteed to find a solution [18]. Hence, demonstrating the smoothness properties will conclude our proof. Rather than giving a formal proof we will only give an outline. The outline will also highlight the theoretical importance of our set of weights w_i^j.

We denote the set of all vectors α that satisfy constraints $\alpha_i^j > 0$, (2), (3), and (4) as Ω. Furthermore, we denote the set of all vectors α that satisfy constraints (1), (2), (3), and (4) as Ω_{ε_2}, where the subscript indicates the dependence on the parameter ε_2. If no singularities exist (see below), the set Ω is bounded and open. The set Ω_{ε_2} is bounded and closed. We want to show that $\varepsilon_2 > 0$ exists such that Ω_{ε_2} contains a continuous and differentiable surface. From Fary's Theorem [17] we know that a vector (of angles) $\omega \in \Omega$ exists that satisfies our constraints. So ω represents a valid planar mesh with the same structure/topology as the three-dimensional triangulated surface represented by β. From the geometry of the two-dimensional mesh it is clear that there exists a neighborhood of ω that satisfies all our constraints. This neighborhood represents all small perturbations of the mesh nodes that maintain the validity of the mesh. From this heuristic argument we may infer that the gradients of the constraints (4)

$$\nabla g_k^{(4)}(\alpha), \text{ for } k = 1, \dots M_{int},$$

are never dependent with the gradients of the linear equality constraints (2) and (3). So we have no singularities, where the surfaces defined by these constraints touch. Now we choose an arbitrarily small value ε_2 such that $\frac{1}{2}\min(\omega_i^j) > \varepsilon_2 > 0$. Therefore, some neighborhood of ω exists that is a subset of Ω_{ε_2} and that is continuous and differentiable. We denote this *surface* by S_ω. Note that the gradients of the functions $g_k^{(2)}$ and $g_k^{(3)}$ are constant and non-zero, that the gradients of the functions $g_k^{(4)}$ are differentiable and continuous, as long as $\alpha_i^j \geq \varepsilon_2 > 0$ for all i,j, and that the gradients of $g_k^{(4)}$ never vanish. Now from continuity we can extend S_ω until either it crosses the boundary $\alpha_{i,j} = \varepsilon_2$ or it closes. Note that $S_\omega \subset \Omega_{\varepsilon_2}$ is a bounded set, since we have $\varepsilon_2 \leq \alpha_i^j \leq \pi - \varepsilon_2$, for all i, j. Concluding, if no singularities occur, S_ω is a bounded, closed set in a finite dimensional normed space (we may assume any norm) and therefore it is compact. Clearly, our objective function is continuous and so it assumes a minimum (and a maximum) on $S_\omega \subset \Omega\varepsilon_2$. It is easily verified that the gradient of our auxiliary objective function is continuous and differentiable as long as $\alpha_i^j > 0$. If necessary, we adaptively choose a set of weights w_i^j such that the auxiliary objective function has a stationary point that satisfies the constraints (1). Then, globally convergent extensions of Newton's method [18] are guaranteed to find the solution.

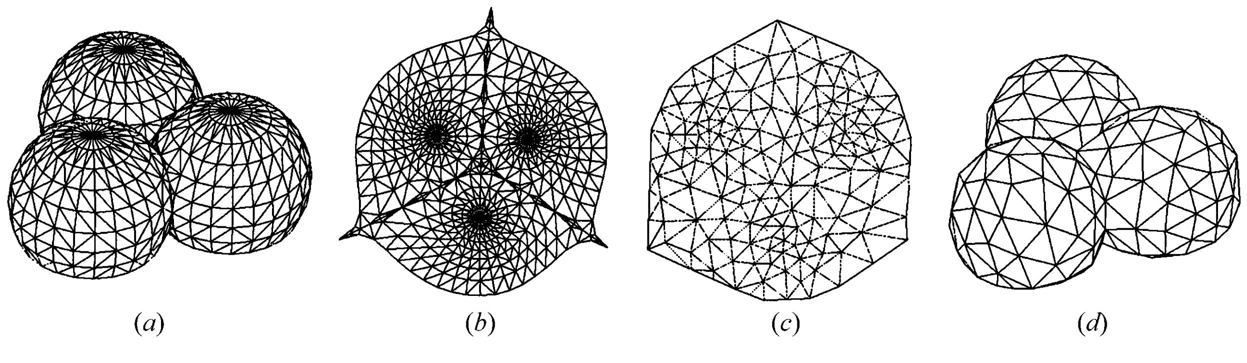

Fig. 8. Remeshing a three ball model. (a) Original surface (1032 elements), (b) flat parameterization, (d) (e) Remeshing: (d) mesh in parameter space, (e) surface mesh with 279 elements.

Finally, note that our standard initial choice of $w_i^j = (\phi_i^j)^{-2}$ puts the vector ϕ at equal distances to the boundaries defined by the constraints (1). This makes it very unlikely that the Newton algorithm tries to approximate a stationary point outside this domain.

4. Examples

Throughout the paper the flattening procedure is demonstrated on several examples of varying complexity. The examples' properties are summarized in Table 1. To measure the distortion of the flat surface with regard to the original mesh we use $F(\alpha)$ (Eq. (1)) with the weights w_i^j set to the initial algorithm choice of $(\phi_i^j)^{-2}$.

Figures 9 and 10 show relatively simple meshes with few elements. Figure 9 shows the flattening of a nearly developable surface. It demonstrates the distortion minimization property where the resulting flat surface has near zero distortion compared to the original three-dimensional surface. In Fig. 10, we demonstrate the method's ability to handle surfaces with interior holes, and significant variation in element sizes.

In Fig. 6 we show a model of a cat's head (cut around the neck). The model is quite complex and has a large overall curvature, hence the distortion is relatively high. Figure 8 displays a surface build from three spheres positioned at 120° around a joint axis. The surface is cut at about a quarter of the diameter up from the sphere base, to create a surface which can be parameterized. The surface mesh is highly non-smooth with high local curvature changes and with multiple sliver triangles, but despite this the parameterization converges in a small number of iterations and provides good results. This model

Table 1. Flattening examples data. The numbers in brackets for the spiral and rabbit examples represent the values of the first solution iteration (before the intersection prevention)

Model	Num. faces	Runtime (sec.)	Num. iterations	Distortion $F(\alpha)$
Spiral	68	0.3	5	0.022
			(2)	(4e-6)
Rabbit	380	108	10	31.28
			(6)	29.9
Cat	257	10	4	21
3 Balls	1032	158	4	36.118
Folded Plane	280	3	2	5e-4
Face	472	21	3	2.36

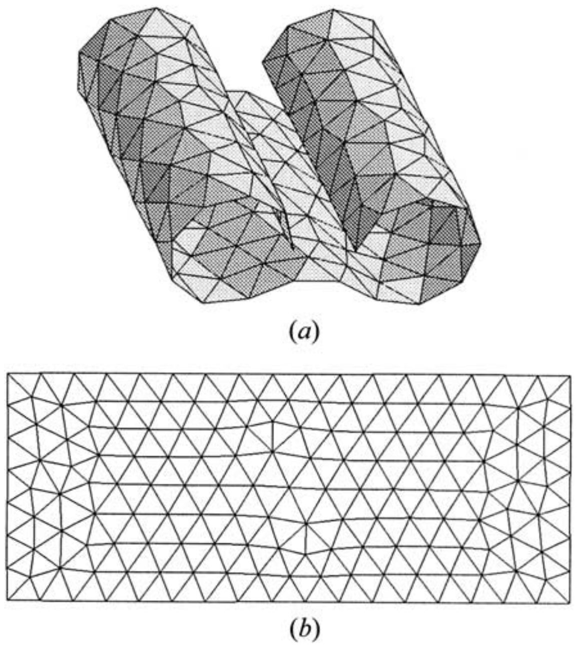
(a)
(b)

Fig. 9. Flat triangulation of nearly developable surface. (a) Original model, (b) flat triangulation.

is about four times larger than the cat model. Despite the increase in model size, the number of Newton iterations required to obtain a solution does not increase. The increase in solution time (roughly a factor of fifteen) is the result of a linear increase in the cost of the matrix-vector product and an increase in the number of iterations in the linear solver.

Figures 4 and 5 show examples, where the initial parameterization resulted in boundary intersections, and an additional intersection removal step was required. The rabbit model in Fig. 5 consists of half of the original closed and symmetric model. The model contains a region of very high curvature near the rabbit's ear. This produces a relatively high distortion in this region during parameterization. A possible way to disperse this type of local distortion throughout a larger region is by adopting the weights at the mesh angles, increasing the weight near nodes with high curvature.

5. Summary

We have proposed a new method for computing a surface parameterization that can be used for generating three-dimensional surface meshes and for several other purposes. We have discussed the properties and underlying theory of our algorithm. Our method can be applied to more general prob-

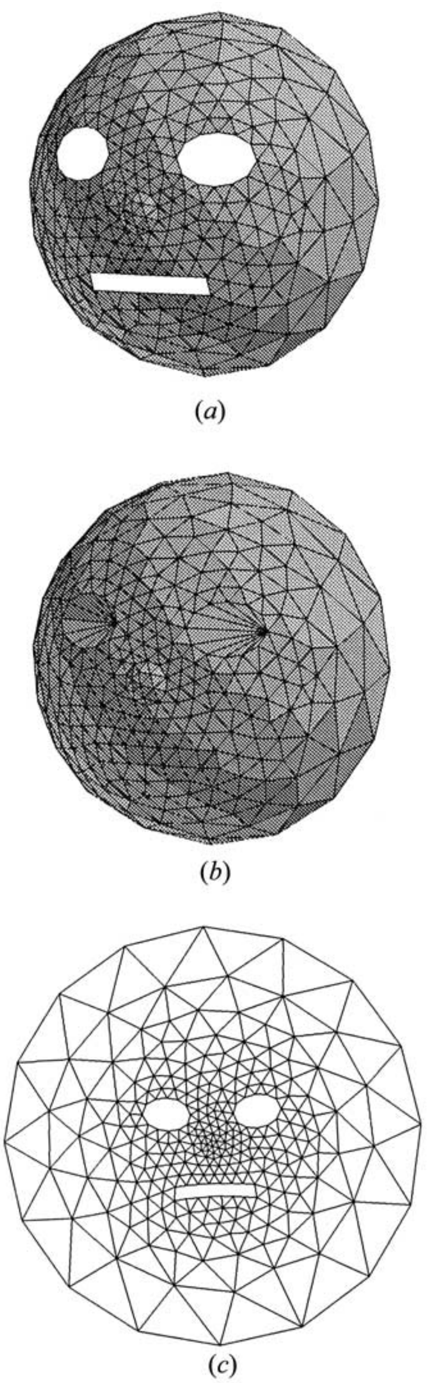

Fig. 10. Flat triangulation of surface with holes. (a) The original model, (b) model with the additional triangles filling the holes, (c) flat triangulation of the model.

lems than other methods, has fewer restrictions, and does not require the user to predefine boundaries. The method is based on minimizing the (relative) distortion of the mesh angles in each face subject to the necessary and sufficient conditions of a valid two-dimensional mesh. In our examples we demonstrate that the proposed method is capable of generating good parameterizations of highly complex surfaces in a modest amount of time.

Although we must solve a nonlinear system of equations, in practice the work involved is modest. For all our test cases convergence seemed to be very fast. The convergence seems loosely dependent on the curvature and complexity of the three-dimensional mesh, which is to be expected.

The efficiency of the current implementation can be further improved in a number of ways. This will be especially useful when handling larger meshes (tens of thousands of elements). First, we do not need to recompute the entire Jacobian, since the part that derives from the linear constraints is constant. Second, as part of the Jacobian is constant, we only need to update the preconditioner for those rows that are changed. This can be done easily if we order the rows of the Jacobian so that the rows that change come last. The computation of a good preconditioner accounts for a significant part of the computational cost. So, we expect this will make a large difference in run-time. Third, the various parts of the solution algorithm, especially the convergence tolerances, should be tuned better for overall performance. For example, we currently solve the linear systems in each Newton iteration to high relative accuracy. High accuracy is unnecessary in the initial Newton iterations. Reducing the tolerance in the initial iterations may therefore reduce costs without affecting convergence.

Another area of future research is parameterization of very large models. For models with hundreds of thousands of elements, solving the flattening problem directly as an optimization problem may be too expensive. For such surfaces a mesh simplification procedure has to be used first to produce a mesh with fewer elements. The parameterization obtained for the simplified mesh, can then be adjusted to handle the original.

An interesting research issue raised by this research is automatic surface subdivision to facilitate low-distortion parameterization. Subdivision has to be used to parameterize closed surfaces or other surfaces which are not *surface triangulations* as defined in Section 3. It can also improve parameterization of highly distorted surfaces like the rabbit model in Fig. 5. Subdivision methods similar to Sheffer [19] can be used, but they need to be adopted to follow parameterization based subdivision criteria.

Acknowledgements

The work reported here was supported, in part, by the Center for Process Simulation and Design (NSF DMS 98-73945) and the Center for Simulation of Advanced Rockets (DOE LLNL B341494).

References

1. Bern, M., Eppstein, D. (1997) Quadrilateral meshing by circle packing. 6th International Meshing Roundtable, 7–19
2. Borouchaki, H., Frey, P. J. (1998) Adaptive triangular-quadrilateral mesh generation. Int J Numerical Methods in Eng, 41, 915–934
3. Li, X. Y., Teng, S. H., Üngör, A. (1999) Biting ellipses to generate anisotropic mesh. 8th International Meshing Roundtable, 97–108
4. Sheffer, A., Blacker, T. D., Bercovier, M. (2000) Virtual topology operators for meshing. Int J Computational Geometry and Applic, 10(2)
5. Haker, S., Angenent, S., Tannenbaum, A., Kikinis, R., Sapiro, G., Halle, M. (2000) Conformal surface parameterization for texture mapping. IEEE Trans Visualization and Computer Graphics, 6(2), 181–189
6. Zigelman, G., Kimmel, R., Kiryati, N. (2000) Texture mapping using surface flattening via multi-dimensional scaling. CIS report CIS-2000–01. IEEE Trans Visualization and Computer Graphics (submitted)
7. Levi, B., Mallet, J. L. (1998) Non-distorted texture mapping for sheared triangulated meshes. Proc SIGGRAPH 1998, 343–352
8. Eck, M., DeRose, T., Duchamp, T., et al. (1995) Multiresolution analysis of arbitrary meshes. Computer Graphics (Annual Conference Series, 1995. SIGGRAPH '95), 173–182
9. McCartney, J., Hinds, B. K., Seow, B. L. (1999) The flattening of triangulated surfaces incorporating darts and gussets. Computer-Aided Design, 31, 249–260
10. Floater, M. S. (1997) Parametrization and smooth approximation of surface triangulations. Computer Aided Geometric Design, 14, 231–250
11. Marcum, D. L., Gaiter, J. A. (1999) Unstructured surface grid generation using global mapping and physical space approximation. 8th International Meshing Roundtable, 397–406
12. Sheffer, A., de Sturler, E. (2000) Parameterization of CAD surfaces for meshing by triangulation flattening. Proc 7th Int Conf Numerical Grid Generation in Computational Field Simulations, 699–708
13. Saad, Y., Schultz, M. (1986) GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. SIAM J Scientific and Statistical Computing, 7, 856–869
14. Van der Vorst, H. A. (1992) BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. SIAM J Scientific and Statistical Computing, 13, 631–644
15. Saad, Y. (1994) ILUT: a dual threshold incomplete ILU factorization. Numerical Linear Algebra and Applications, 1, 387–402
16. Marshall, C. W. (1971) Applied Graph Theory. Wiley, New York
17. Fary, I. (1948) On the straight line representations of planar graphs. Acta Sci Math 11, 229–233
18. Dennis, Jr, J. E., Schnabel, R. E. (1996) Numerical Methods for Unconstrained Optimization and Nonlinear Equations. SIAMpub, Philadelphia, PA, USA
19. Sheffer, A. (2000) Model simplification for meshing using face glustering. Computer Aided Design, 33, 925–934