FORCE-DIRECTED METHODS FOR SMOOTHING UNSTRUCTURED TRIANGULAR AND TETRAHEDRAL MESHES

Hristo N. Djidjev,

Department of Computer Science
University of Warwick
Coventry CV4 7AL, UK

ABSTRACT

We develop and implement new algorithms for smoothing triangular and tetrahedral unstructured meshes. Our approach is based on a variation of the force-directed method used in graph drawing. This method assumes that on each vertex a certain force is applied that moves the vertex relative to its neighbors so that the shapes of its incident elements are improved. The final stable configuration often corresponds to a graph with good global properties. In this paper we show that this method can be successfully applied to mesh smoothing and describe some details of our implementation and test results.

Keywords:mesh smoothing, force-directed methods, unstructured, triangular, tetrahedral

1 Introduction

The solution of partial differential equations arising in simulations in computational fluid dynamics and physics requires the representation of a two or three-dimensional domain as a mesh of elements of simple shapes.

For the stability and the convergence of the resulting numerical method the mesh used should be of high quality. Often used quality criteria for isotropic meshes are small aspect ratio of elements and no angles very close to $0°$ or to $180°$.

Many mesh generation algorithms use the so called constrained Delaunay triangulation [8, 5] of a set of points in the domain. In the case of a planar domain, the Delaunay triangulation of a given point set maximizes the minimum angle among all triangulations of that set of points. However, a different set of points in the same domain could yield a Delaunay triangulation which is a mesh of a better quality. While Delaunay triangulation is believed to be a relatively good approach to mesh generation in the two-dimensional case and several popular mesh generators are based on it (e.g. [26]), in three-dimensions it often produced meshes of very low quality. Other often used mesh generation methods include quadtrees and octrees, advancing fronts, circle-packing, and others [3, 4, 5].

There are two basic approaches to the problem of improving the properties of an existing mesh. The first approach allows the topology of the mesh to be changed by inserting or deleting new points or by local reconnections. The other approach, called *mesh smoothing*, that we study in this paper allows the positions of the mesh points to be changed, but keeps the topology of the original mesh (i.e. the underlying graph) unchanged.

One of the simplest and most often used mesh smoothing methods is the so called *Laplacian smoothing*

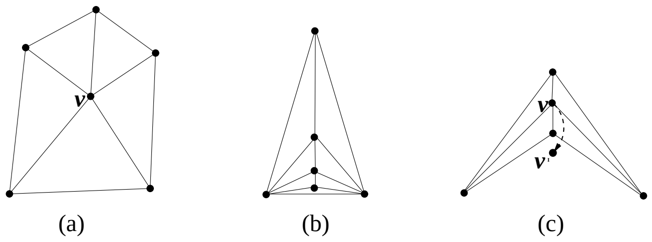

Figure 1: Laplacian smoothing. (a) Point v at the center of mass (good case). (b) Laplacian smoothing produces exponentially small angles. (c) The center of gravity v' is outside the domain.

[28, 12, 20]. By this method, interior mesh points are successively moved to the geometric center of their neighbors (Figure 1(a)). This operation is applied on all mesh points in a given order and is repeated a certain number of times or until the maximum displacement of a mesh point during the last iteration becomes smaller than a given error constant. The exact fixed point of Laplacian smoothing can be computed by finding the eigenvector of the Laplacian of the graph corresponding to the mesh (hence the name). A weighted version of the Laplacian smoothing assigns weights to nodes depending on the properties of their adjacent elements, e.g., the average edge length of the elements surrounding the corresponding node.

Laplacian smoothing is relatively inexpensive, but it does not guarantee any improvement in mesh quality and can even produce invalid meshes (see Figure 1(c)). An example of a k-point mesh for which Laplacian smoothing produces a valid mesh, but with angles as small as $O(2^{-k})$, is given on Figure 1(b). An often used variant of Laplacian smoothing, known as a *smart Laplacian smoothing*, moves a mesh point only if that change improves the mesh quality according to the chosen criterion.

The fixed point of Laplacian smoothing minimizes the sum of the squares of edge lengths of the mesh, a criterion not directly related to any relevant mesh quality parameter. In a different approach, Freitag et al. [16] proposed an optimization based alternative to Laplacian smoothing that always results in a valid mesh. Their algorithm sweeps over the mesh, computing a new position for each point that maximizes the minimum angle between the edges incident to that node. Generalizing that approach, Amenta et al. [1] show that a variety of other optimization problems such as maximizing the minimum angle, minimizing the maximum angle, minimizing the maximum aspect ratio (defined as the ratio of the length of the longest side of a triangle to its altitude), minimizing the maximum circumradius and the radius of the containing circle of the incident triangles, can be used as objective functions in the place of the minimum angle. They show that all these optimization problems can be formulated as *quasi-convex programs* and thus can be solved in time linear to the number of incident elements using a generalized linear programming approach. Although this is an interesting theoretical result, the practicality of the method is unclear because of the large constants involved in the time analysis [1] and, more importantly, the sequence of locally optimal placements does not guarantee any optimality property of the final mesh (except that the last point to be moved will be in optimal position with respect to its neighbors).

Our approach tries to combine the advantages of both the Laplacian smoothing (efficiency and convergence of the solution) and the optimization based smoothing (meaningful optimization criterion and better quality of the result). Our aim is not to apply a Laplacian smoothing on nodes whose adjacent elements have better shapes and an optimization based smoothing on the remaining nodes, an approach used in, e.g., [7, 14, 15, 25]. Our method is based on a technique developed in graph drawing. Eades [11] describes a *force-directed* method for finding an "aesthetical" drawing of a graph by representing the graph as a mechanical system with vertices replaced by steel rings and edges by springs that exert repulsive or attractive forces on their endpoints depending on the current lengths of the edges. Additional repulsive forces are usually assumed between each pair of nonadjacent vertices. The equilibrium (minimum energy) configuration of the idealized mechanical system often displays well any symmetries of the input graph or has other nice properties. Note that when there are no repulsive forces and the attractive force is proportional to the edge length, the resulting algorithm computes Laplacian smoothing. A similar approach was used in the 1960s by Tutte [27], who applied the model to show that any triconnected planar graph has a straightline planar drawing. A number of refinements and extensions of the basic method have been recently reported, see e.g. [17, 18]. All of these algorithms, however, use repulsive forces between any pair of nonadjacent vertices, which greatly reduces their speeds and hence makes them inapplicable to mesh smoothing, where meshes typically correspond to very large graphs. Another typical disadvantage of this method is that, depending on the forces used, there might be multiple equilibrium states, which makes the corresponding optimization procedure less effective.

In this paper we describe a method for mesh smoothing based on the above idea. We apply a force at any node that depends only on its incident elements and hence can be computed relatively fast. The simulation of the forces and the minimum energy (stable) configuration is determined by using an optimization procedure. Hence, from a technical point of view, our method can be classified as an optimization-based mesh smoothing and compared with previous work in this area, e.g. [6, 14, 9, 23, 25]. Important feature of our approach is that, by putting the accent on the forces that will be simulated, we can ensure that the resulting optimization procedure can be solved relatively fast and its solution will minimize a global objective function. Moreover, unlike Laplacian smoothing, our objective function will be a meaningful mesh quality criterion.

The paper is organized as follows. In Section 2, we justify the choice of an objective function and forces applied on vertices and outline the algorithm that computes an improved mesh. We study the cases of triangular and tetrahedral meshes. In Section 3, we discuss some details of our implementation. In Section 4, we describe the results of experiments with several implementations of our algorithm on test meshes. In the concluding section, we summarize our experience and list problems for future research.

2 Computation of the forces

In this section we will determine the forces that have to be applied on each vertex so that the mesh after smoothing with those forces satisfies certain quality requirements. We separately consider the 2d and the 3d cases.

2.1 Triangular meshes

Let M be a triangular, connected, unstructured, planar mesh and let B be the set of the *boundary* nodes of M that include the nodes on the outer face of M and possibly the nodes of some other faces of M ("holes") called *boundary faces*. All non-boundary faces of M are triangles. In the mesh-smoothing problem we are going to consider the case in which the non-boundary nodes are allowed to move freely as long as the planarity of the underlying graph is preserved. The boundary nodes are allowed to move along the boundary if the boundary polygons are not changed. The apex nodes of the boundary faces are not allowed to move.

We define the following objective function

$$\mathcal{F}(M) = 3 \sum_{\alpha \in \mathcal{A}} |\cot \alpha|,$$

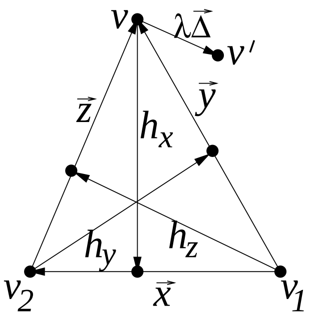

Figure 2: Computing the force at v in 2d.

where $\mathcal{A}$ is the set of all element angles and the coefficient 3 is chosen in order to simplify the following formulas.

Note that when the angle α tends to 0 or π, $|\cot(\alpha)|$ tends to $+\infty$, hence the optimal solution w.r.t. this function will avoid both very small and very large angles.

We can rewrite the objective function as

$$\mathcal{F}(M) \quad = \quad \sum_{v \in V} \sum_{t \in N(v)} f(t), \qquad (1)$$

where $f(t)$ denotes the sum of the absolute values of the cotangents of the angles of t, V is the set of the nodes of M, and $N(v)$ denotes the set of elements incident to v. Observe that $f(t)$ is a number between $\sqrt{3}$ and $+\infty$ which is minimized for equilateral triangles.

Since the sign of $\cot \alpha$ depends on whether $\alpha \in (0, \pi/2)$ or $\alpha \in (\pi/2, \pi)$, the portion of the gradient contributed by $f(t)$ depends on whether t is an acute triangle or an obtuse triangle. Here we show only the computations in the case of an acute triangle.

For any vector $\vec{q}$, let q denote the Euclidean length of $\vec{q}$. In order to simplify the notation, we will not distinguish hereafter between a mesh node and the point in the plane where that node is embedded.

Let us first consider the case where v is a non-boundary node. Let t be the triangle vv_1v_2 and assume that all angles of t are acute. Let $\vec{x} = \overrightarrow{v_1v_2}$, $\vec{y} = \overrightarrow{v_1v}$, $\vec{z} = \overrightarrow{v_2v}$, and let $\overrightarrow{h_x}$, $\overrightarrow{h_y}$, and $\overrightarrow{h_z}$ be the heights to v_1v_2, v_1v, and v_2v, respectively (Figure 2). By definition

$$f(t) = \frac{x}{h_x} + \frac{y}{h_y} + \frac{z}{h_z} = \frac{x^2+y^2+z^2}{xh_x}.$$

In order to compute the gradient at v we denote by $\overrightarrow{\triangle}$ any unit vector and choose $\lambda \geq 0$. Denote by v' the point in the plane for which $\overrightarrow{vv'} = \lambda\overrightarrow{\triangle}$ and by t' the triangle $v'v_1v_2$. Define a function $\phi(v, t, \lambda, \overrightarrow{\triangle}) =$

$f(t')$. Straightforward calculations show that the directional derivative of f at v is

$$\phi'_\lambda(v,t,\lambda,\overrightarrow{\triangle})|_{\lambda=0} = \left(\frac{x^2+y^2+z^2}{2h_x^2}\overrightarrow{h_x}+\vec{y}+\vec{z}\right)\frac{\overrightarrow{\triangle}}{S(t)},$$

where $S(t) = xh_x/2$ is the area of t.
Thus, the contribution of the pair (v,t) to the gradient of $\mathcal{F}(M)$ is $\vec{g}(v,t) = \frac{f(t)}{h_x^2}\overrightarrow{h_x} + \frac{1}{S(t)}(\vec{y}+\vec{z})$. Hence the steepest-descent force applied to v w.r.t. $f(t)$ will be

$$\overrightarrow{force}(v,t) = -\frac{f(t)}{h_x^2}\overrightarrow{h_x} - \frac{1}{S(t)}(\vec{y}+\vec{z}). \quad (2)$$

If v is a boundary node different from the apices of the corresponding boundary polygon, i.e., if v is an internal point of a straightline segment on the boundary of M, then we will be allowed to move v on l as long as the planarity of M is preserved. The force on v in this case is computed as the projection on l of the force on v found by equation (2). More information about the 2d case can be found in [10].

2.2 Tetrahedral meshes

Now we consider the three-dimensional case where all elements of M are tetrahedra.
In analogy with the two-dimensional case, we can use as an objective function the sum of the absolute values of the cotangents of the dihedral angles of all tetrahedra. In such a case, however, the formula for computing the forces at the nodes will be much more complex and that will make the algorithm too slow and impractical. For this end, we will choose objective functions that may be not as precise, but will lead to simpler computations.We assume that the objective function is given in a similar form as (1), i.e.

$$\mathcal{F}(M) = \sum_{v\in V}\sum_{\tau\in N(v)} f(\tau), \quad (3)$$

where V and $N(v)$ have the same meaning as in (1) and $f(\tau)$ has yet to be defined. Next we discuss two choices for the function $f(\tau)$.

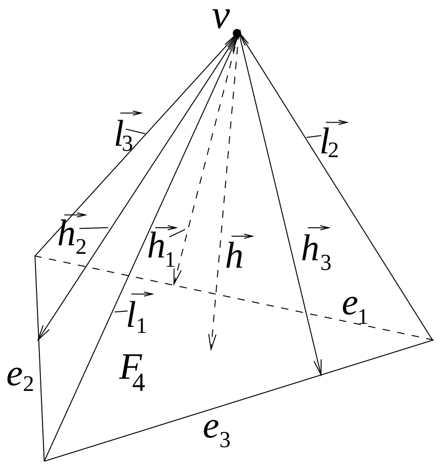

Figure 3: Computing the force at v in 3d.

Ratio of element face area to volume (AV force)
Let τ be a tetrahedron of M, let the four triangular faces of τ be $F_1,\cdots,F_4$ and their areas be $S_1,\cdots,S_4$, respectively. We define

$$f(\tau) = \frac{\sum_{i=1}^4 S_i}{(3V)^{\frac{2}{3}}}, \quad (4)$$

where V is the volume of τ.
The exponent $\frac{2}{3}$ was chosen instead of the other natural choices 1 and 2 because it leads to slightly simpler expression for the force.
Now let v be that node of τ that is not incident with face F_4. Let for $i = 1,2,3$ each face F_i has an edge e_i in common with F_4, let h_i be the distance from v to e_i, and let h be the height of τ from v (see Figure 3). Then the contribution of τ to the force on v corresponding to the function (4) can be computed as

$$\overrightarrow{force}(v,\tau) = -\frac{2}{3}\frac{f(\tau)}{h^2}\vec{h} + \frac{1}{2(3V)^{\frac{2}{3}}}\sum_{i=1}^{3} e_i\vec{n_i}, \quad (5)$$

where $\vec{n_i} = \frac{\vec{h_i}}{h_i}$.

Sum of squared edge lengths over volume (LV force)
The formula (5) requires computation at each iteration of a number of variables for each tetrahedron: the volume, 4 heights, lengths of 3 edges, the areas of 4 faces. We will define an alternative force that will be easier to compute by using the objective function

$$f(\tau) = \frac{\sum_{i=1}^6 l_i^2}{2V^{\frac{2}{3}}}, \quad (6)$$

where l_i are the edges of the tetrahedron τ.
The force corresponding to (6) is

$$\overrightarrow{force}(v,\tau) = -\frac{2}{3}\frac{f(\tau)}{h^2}\vec{h} - \frac{\sum_{i=1}^{3}\vec{l_i}}{V^{\frac{2}{3}}}, \quad (7)$$

which is clearly easier to compute than the force (5).

3 Implementation

In this section we briefly describe the structure of our smoothing algorithm and some implementation issues.

Optimization procedure Force-based graph drawing algorithms simulate the effect of a set of forces on the nodes of the graph or mesh in a continuous manner. The initial forces make the nodes move slightly in a certain directions, the new node positions cause the forces to possibly change direction, the vertices move a small distance in the new direction, etc., until the forces at all vertices become zero.

A convenient way to study such a simulation is to use optimization theory terminology and techniques. Our goal is to minimize an objective function of type (1). We apply an iterative search procedure with the following structure. For each internal mesh point v we compute a search direction $\vec{d}(v)$ using formula (2), (5), or (7). Next we determine the optimal displacement for v in that direction, i.e., a constant $\alpha > 0$ minimizing $\sum_{t \in N(v_\alpha)} f(t)$, where $\overrightarrow{vv_\alpha} = \alpha\vec{d}(v)$. That constant α can be found by a binary search or by a more complex line-search procedure (see e.g. [13]). After v is repositioned, the same procedure is applied to all other internal nodes. Depending on the termination criterion used, the whole procedure is repeated a predetermined number of times or until the improvement of the objective function during the previous iteration becomes smaller than some precision (error) parameter.

The algorithm can be divided into the following modules: (i) a procedure for computing the direction of the force for any single node; (ii) a procedure for determining the distance to move in the direction of that force; (iii) a procedure for determining the order into which the nodes are moved during the progression of the algorithm as well as a stopping criterion; (iv) other procedures as input/output, visualization of results, evaluation of mesh quality, etc., which will not be discussed here.

Computing a descent direction. We have a choice of several methods for computing the direction $\vec{d}(v)$. The forces given in the formulas (2), (5) and (7) correspond to the method of the steepest descent. That method is the simplest one and chooses $\overrightarrow{force}(v) = -\vec{g}(v)$, where $\vec{g}(v)$ is the gradient at v. The more complex Newton method [13] is generally more efficient in terms of the convergence rate as it uses a quadratic approximation of the objective function. However, each iteration of the Newton method is computationally much more expensive, which makes it inapplicable to our case. A third option, which is more acceptable than the Newton method for our application, is the conjugate gradient method [13]. By this method, the new search direction $\vec{d}(v)$ is computed as a linear combination of the old direction and the new gradient as

$$\vec{d}(v) = -\vec{g}(v) + \frac{\|\vec{g}(v)\|^2}{\|\vec{g}(v)_{prev}\|^2}\vec{d}(v)_{prev},$$

where $\vec{g}(v)_{prev}$ and $\vec{d}(v)_{prev}$ are the corresponding values of $\vec{g}(v)$ and $\vec{d}(v)$ from the previous iteration ($\vec{d}(v) = -\vec{g}(v)$ for the first iteration). The conjugate gradient method is almost as inexpensive as the steepest descent method and in theory is much faster for certain type of smooth objective functions.

Computing a step size. We use a line-search procedure, which, for a given direction, iteratively adjusts the step by checking the values of the objective function and the gradient at different points. One can implement such a procedure as a binary search, but we preferred a slightly faster modification of that procedure similar to the one described in [13].

Determining the order of node adjustment. The straightforward approach would be to move all nodes of the mesh once in any iteration and to repeat until the stopping criterion is met. The problem with such a solution is that the nodes may need different amounts of adjustment – typically most of the nodes will need minor adjustments and only in a few regions of the original mesh bigger improvements may be needed. Hence, we chose to maintain a priority queue of the nodes to be adjusted, where the priority of each node is determined by the Euclidean norm of the gradient at that point. Maintaining the queue takes logarithmic time per insert, minimum, or delete operation and, in all of our experiments, was much smaller than the time for computing the step size.

Other implementation issues. Another heuristic we use is related to combining the Laplacian and the optimization smoothing methods. The idea, originally used in [25, 15], is to use the faster Laplacian smoothing only for nodes adjacent to triangles with

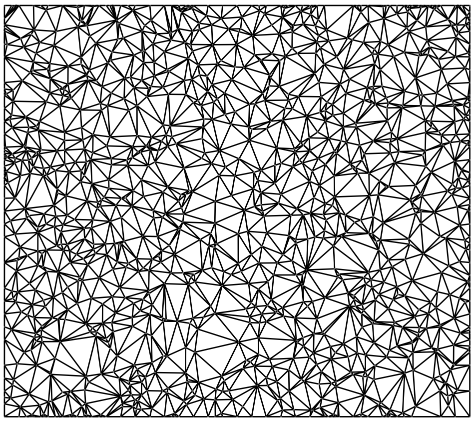

Figure 4: The test mesh Constrained Delaunay (CD).

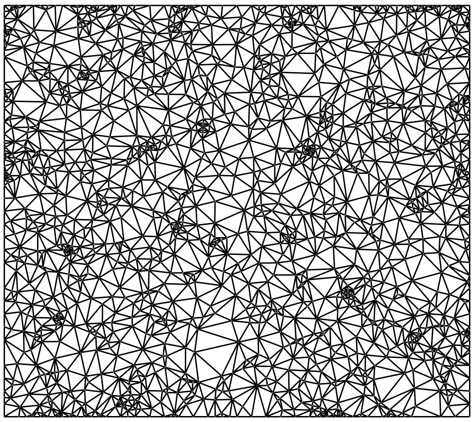

Figure 5: The test mesh Quality Delaunay (QD).

nice shapes, i.e., for nodes that are not likely to need big adjustments, and the slower but more precise force-based smoothing discussed above for the few remaining nodes.
The algorithm is coded in C++ and is based upon the LEDA software library of efficient data structures and algorithms [21]. A few simple data structures are used including a priority queue from LEDA for determining the order into which the nodes will be adjusted and look-up tables for faster computation of angles and cotangents (for the 2d case). For computing the angle between two vectors, for instance, we store in the table the values of arccosines for 1000 uniformly distributed points in the interval $(-1, 1)$. Then, for determining the angle between the query vectors, each of them is normalized. If x_0 is the x-coordinate of one of the normalized vectors, its y-coordinate is positive, and γ is the angle between that vector and the x-axis, then clearly $x_0 = \cos\gamma$. Hence γ can be approximately computed by finding in the table the entries for the two closest to x_0 points in $(-1, 1)$ and interpolating γ as a linear combination of the arccosines of those two points.

4 Experimental results

4.1 The two-dimensional case

We will study the effectiveness of the proposed method and its different implementations by comparing the performances of the Laplacian smoothing and the several variants described above of our force-driven method based on the steepest descent and the conjugate gradient methods, respectively.

Test meshes. We will use the following meshes for our tests:
(i) *Constrained Delaunay (CD).* This is a mesh produced by a constrained Delaunay triangulation algorithm [8] of a set of 1000 random points inside a square region of the plane. The mesh has 1232 points and 2217 faces (the additional points are on the boundary of the square) and is shown in Figure 4.
(ii) *Quality Delaunay (QD)* This mesh was generated by the Carnegie-Mellon `triangle` mesh generator [26] using the same set of random points as for the mesh CD. That generator is based on the algorithm of Ruppert [24] that inserts additional (Steiner) points in the original point set as to guarantee that the mesh produced by a constrained Delaunay triangulation for the resulting point set does not have an angle smaller than some positive constant $\varepsilon \leq 20.7°$. The larger the ε, the larger the number of Steiner points that have to be added. QD test mesh shown in Figure 5 was produced with $\varepsilon = 15°$ and has 1664 nodes and 2986 faces.
(iii) *Airfoil.* This is a typical mesh for fluid dynamics simulations and is produced by the meshing algorithm of Barth and Jespersen [2]. It features well shaped elements and high grading (maximum ratio of longest to shortest mesh edge). We use it to test the performance of our 2d algorithms on high quality meshes. Airfoil (Figure 6) has 4720 nodes and 9004 faces.

Algorithms. We have made experiments with the

mesh	method	minimum angle	maximum angle	worst distortion	average distortion
Constrained Delaunay (CD)	original	1.21	169.74	0.03	0.49
	SL	1.99	155.36	0.09	0.81
	SD	5.63	132.71	0.11	0.84
	CG	6.00	130.53	0.12	0.85
Quality Delaunay (QD)	original	15.00	146.13	0.21	0.66
	SL	12.21	144.11	0.21	0.84
	SD	15.05	131.97	0.32	0.86
	CG	15.25	129.85	0.34	0.86
Airfoil	original	8.12	117.42	0.25	0.96
	SL	16.74	112.35	0.42	0.98
	SD	19.24	106.11	0.59	0.98
	CG	19.24	104.44	0.61	0.98

Table 1: Comparison of improvements in 2d test meshes.

mesh	method	Laplacian adjustments	force-based adjustments	time (sec.)
Constrained Delaunay (CD)	SL	5281	0	0.48
	SD	1232	7329	3.97
	CG	1232	5131	2.28
Quality Delaunay (QD)	SL	9329	0	0.97
	SD	1664	7862	4.28
	CG	1664	6137	2.66
Airfoil	SL	20265	0	1.81
	SD	4253	12285	4.76
	CG	4253	10571	2.97

Table 2: Comparison of the speeds of the optimization methods.

following mesh smoothing algorithms:

(i) *Smart Laplacian (SL).* Our implementation is a variation of the Laplacian smoothing where a node is moved only if that change will locally improve the quality of the mesh.

(ii) *Steepest Descent (SD).* This is a version of the force-directed algorithm described in Sections 2 and 3 that uses steepest descent method for the computation of the force direction. The algorithm initially performs one Laplacian adjustment for each node in order to determine appropriate initial values for the step size in the linesearch procedure and to set the initial priorities of the nodes for the priority queue.

(iii) *Conjugate gradient (CG).* This algorithm is the same as SD, except that the conjugate gradient method is used to determine the descent direction.

Results. In Table 1 we compare the effectiveness on quality improvement of the three mesh smoothing algorithms on the test meshes. The four quality criteria we use are the minimum and the maximum angles and the minimum and the average distortion of faces. We adopt the distortion metric used in [19, 7], where the *distortion* of a triangle t with edge lengths a, b, c and area $S(t)$ is defined as

$$\text{distortion}(t) = 2\sqrt{3}\frac{S(t)}{a^2 + b^2 + c^2} .$$

The coefficient $2\sqrt{3}$ is used as a normalization factor such that equilateral triangles have a distortion one (the maximum possible distortion).
For each algorithm in Table 1 the average number of node adjustments is three . Recall that the actual number of times any individual node is adjusted may

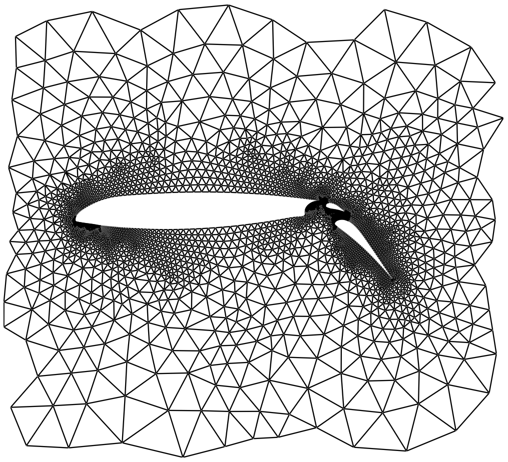

Figure 6: The test mesh Airfoil.

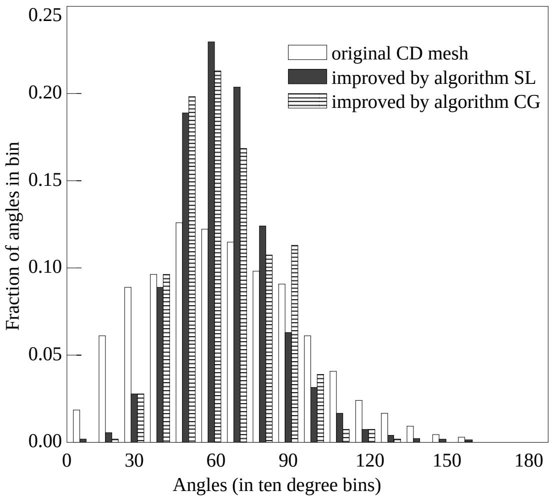

Figure 7: Distribution of angles for test mesh Constrained Delaunay improved by algorithms SL and CG.

not be the same as for the other nodes and depends, roughly, on the quality of the triangles in its neighborhood.

It can be observed from the table that in all cases the force-directed algorithms improve the quality of the original mesh and perform better than the Laplacian smoothing. The difference between SD and CG compared to SL is more noticeable for the lower quality mesh CD. But even on Airfoil, the highest quality mesh, CG produces a mesh with worst-triangle distortion 0.61, compared to 0.42 for algorithm SL, which is a considerable improvement. Notice that the improvements in the average distortions for each of the algorithms are smaller than for the other parameters. That can be explained by the fact that in a typical mesh the number of the badly shaped triangles is much smaller than the number of well-shaped ones. (But even a few low-quality triangles can cause problems in numerical simulations based on such a mesh.)

Table 1 shows also a slightly better performance of CG algorithm compared to SD. But, as we will see in Table 2, that difference becomes more noticeable when the number of adjustments increase.

Figure 7 illustrates the distribution of angles for the test mesh CD after being smoothed by algorithms SL and CG as in Table 1. It can be observed that SL produces a larger number of angles close to $60°$, while CG favors angles in the vicinity of $90°$. (A similar behavior has been established for the other test meshes.) Moreover, CG eliminates a larger portion of the obtuse angles.

Finally, Table 2 contains data on the number of iterations of the tested algorithms required to achieve a relative error (distance of the last displacement of a node divided by the square root of the total area of its incident triangles) of any node less than 0.05. All tests have been run on an Ultra 5 Sparc workstation.

For all the test meshes algorithm SL is faster than CG, which is faster than SD. It is natural for the Laplacian smoothing to be faster than the other two algorithms, since the determination of the new position for a node involves a simpler computation. Furthermore, there is no need for the line-search step, since each node can be directly moved to the center of the mass of its neighbors.

Another difference between algorithm SL and the other two algorithms is that, for SL, the time per node for a single adjustment is roughly the same for all meshes. On the other hand, the time per adjustment for SD and CG depends on the quality of the initial mesh, as can be seen in Table 2. Thus, the difference in speeds between algorithms SL and CG is greater (between 4 and 5 times) for the worse mesh CD and smaller (about 1.6 times) for the high quality mesh Airfoil. This can be explained by the fact that for the latter type of meshes the line-search procedure is more efficient. For such meshes often the first guess for the length of the adjustment step is correct.

One can also observe that for all meshes algorithm CG works noticeably faster than algorithm SD, although the difference between the performance of these two algorithms is not so distinct in Table 1. This is due to the fact that for the tests in Table 1 we chose to make only three adjustments per node on average, the first of them being a Laplacian one and thus the same for

mesh	method	minimum angle	maximum angle	worst distortion	average distortion
Random Cube (RC)	original	0.00	179.99	0.00	0.29
	SL3	0.36	179.28	0.04	0.52
	AV	4.78	166.65	0.22	0.58
	LV	5.23	169.73	0.22	0.58
SF Quake (SFQ)	original	16.14	146.17	0.44	0.65
	SL3	16.14	156.69	0.43	0.65
	AV	17.30	147.10	0.45	0.69
	LV	16.63	151.17	0.44	0.68

Table 3: Comparison of improvements in 3d test meshes.

mesh	method	Laplacian adjustments	force-based adjustments	time (sec.)
Random Cube (RC)	SL3	3000	0	1.32
	AV	0	3000	11.78
	LV	0	3000	4.05
SF Quake (SFQ)	SL3	3000	0	15.23
	AV	0	3000	76.10
	LV	0	3000	30.06

Table 4: Speeds of algorithms for different force choices in 3d .

all nodes. Since the first descent direction for the conjugate gradient method is the same as for the steepest descent method, the second adjustments are also the same for algorithms SD and CG. This leaves to CG only one adjustment per node on average to outperform SD.

The above test results indicate that algorithm CG is a better choice than SD, working about 50% faster than SD on the test examples. The average time for an adjustment for CG decreases with increasing the size and the quality of the original mesh. For poorly shaped input meshes, CG works about 4–5 times slower than SL, but may achieve considerable improvement on quality. For high quality big input meshes (typical for many applications), the speed of CG can be comparable to SL (1.6 times slower in our test), while CG produces better smoothed meshes than SL (achieved about 50% improvement in the minimum distortion parameter in our tests).

4.2 The three-dimensional case

Test meshes. We will give the experiment results with the following two test meshes:

(i) *Random Cube (RC).* As the name suggests this is a mesh corresponding to a constrained Delaunay triangulation of a set of points in the unit three-dimensional cube. The mesh has 1000 points and 6394 tetrahedral elements (Figure 8).

(ii) *SF Quake (SFQ)* This mesh has been used for finite element simulations for earthquake-induced underground motion in the San Fernando Valley and has been described in [22]. Because of the regular shape of the domain for which that mesh was generated, the quality of mesh SFQ is quite high. The mesh is highly graded and has 7294 points and 35025 elements.

Algorithms. We have carried out experiments with the following mesh smoothing algorithms:

(i) *Smart Laplacian 3d (SL3).* This is the 3 dimensional version of algorithm SL described above.

(ii) *Area to Volume (AV).* This algorithm uses the ratio of element area to volume force (5).

(ii) *Lengths to Volume (LV).* The force used for this algorithm is the ratio of sum of the squared edge lengths of a tetrahedron over the volume (7).

The optimization procedure chosen for both algorithms AV and LV is the conjugate gradient method.

Unlike algorithms SD and CG from the 2d case, in algorithms AV and LV we do not do Laplacian smoothing on nodes whose adjacent tetrahedra are of higher quality. This will help us better evaluate the smoothing efficiency of our force-based algorithms in the 3d case.

Results. The results of the tests are summarized in Table 3, Figure 9, and Table 4. We use similar quality criteria as in the 2d case. The distortion metric chosen is

$$\text{distortion}(\tau) = \frac{27\sqrt{3}}{2} \frac{V^2}{\sum_{i=1}^{4} S_i^3} ,$$

where V and S_i are the volume and the areas of the faces of τ, respectively. The coefficient $\frac{27\sqrt{3}}{2}$ is chosen so that the distortion of any tetrahedron will be a number in the interval $(0, 1]$.

In Table 3 we have compared the effectiveness of the three algorithms on improving the quality of the test meshes.

The original mesh Random Cube is of a very low quality, both in respect to dihedral angles and to the distortion metrics. The smart Laplacian algorithm SL3 performs surprisingly well in terms of average distortion (which is not an important criterion), but essentially does not improve the other quality measures. The force-based algorithms AV and LV both perform reasonably well and produce smoothed meshes of roughly the same quality.

The mesh SF Quake as evident from Table 3 is of very high quality, the original mesh having much better quality measures than the smoothed mesh RC. Algorithm SL3 makes a very small number of changes and actually reduces the quality of the mesh. Algorithms AV and LV improve slightly on the minimum dihedral angle and the worst and the average distortion criteria, but create a little bit larger maximum angles. (However the maximum angles of the original mesh SFQ and its 3 smoothed variants are all in acceptable range.) Algorithm AV works slightly better on the mesh SFQ than Algorithm LV.

Table 3 does not provide enough information to distinguish between the performances of Algorithms AV and LV on mesh SFQ. A more detailed information given in Figure 9 shows a more noticeable improvement on the angles of SFQ by both of these algorithms and a greater difference between the effectiveness of AV and LV.

In Table 4 we compare the speeds of the three methods. Our current versions are not optimized at this time for time performance, hence our goal is only to compare the relative speeds of the algorithms.

As expected, Algorithm SL3 is much faster than the other two algorithms. Yet it is not as fast as its 2d variant since more time is needed in 3d for computing the qualities of the mesh elements in order to be able to decide whether smoothing a particular node by the Laplacian method improves the local quality. The most important observation is that Algorithm LV is about 2.5 times faster than AV, which can be explained by the relative simplicity of formula (7) compared to formula (5). That fact suggests that for big meshes, where time performance is a critical factor, one might prefer that algorithm.

5 Conclusion

We described an algorithm for mesh smoothing based on a variation of the force-directed method used in graph drawing. Compared to existing force-directed algorithms, ours is faster (since the force on a node is computed using only the positions of its neighbors) and leads to configurations optimizing meaningful global quality criterion.

A feature of our approach is that we do not aim at optimizing the quality of the mesh using the standard minmax or maxmin objective functions that lead to NP hard optimization problems. Instead we use objective functions that are sums of quality measures of individual mesh elements. By choosing appropriate quality measures for the elements, one could efficiently solve problems of the latter type. Furthermore, our results suggest that the quality of the meshes are clearly improved with respect to the corresponding minmax or maxmin type criteria, although we certainly can not claim optimality for such criteria.

Our experiments showed much better quality performance of the force-driven method compared to the smart Laplacian smoothing. The conjugate gradient method yields a better performance than the steepest gradient method when the average number of iterations per vertex is 3 or larger and roughly the same performance for a smaller number of iterations.

Algorithm LV is significantly faster than Algorithm AV and should be used for larger meshes. The slightly better quality of smoothing of Algorithm AV does not justify its use. One could consider using a combination of the two methods, where AV is used only in the final iterations or for the lowest quality elements.

There are several ways to improve the efficiency of our algorithms, especially in the 3d case. For instance, using the approach of [15], one can use different smoothing methods for individual nodes depending on the quality of the adjacent elements. Also the use of the

LEDA library of data structures greatly simplifies programming and improves the readability of the code. However, its use affects negatively the speed of the algorithm and has an even worse effect on the use of memory. A significant savings will result if new data structures are designed specifically for use by our algorithms.

We plan also to do experiments on more tetrahedral application meshes and to extend our method and run experiments on other types of 2d and 3d meshes (e.g. quadrilateral and hexahedral).

Acknowledgement. The author would like to thank the referees for the helpful and insightful comments, to John Gilbert for providing the test mesh Airfoil, to Jonathan Shewchuk for the test mesh SF Quake, and to Rory Driscol for helping with some of the coding. This work was partially supported by the EPA grant R82-5207-01-0, EPSRC grant GR/M60750, and RTDF grant 98/99-0140.

References

[1] Nina Amenta, Marshall Bern, and David Eppstein. " Optimal point placement for mesh smoothing. " *Journal of Algorithms*, 30(2):302–322, February 1999.

[2] T. J. Barth and D. C. Jespersen. " The design and application of upwind schemes on unstructured meshes. " In *Proc. AIAA 27th Aerospace Sciences Meeting, Reno*, 1989.

[3] Marshall Bern and Paul Plassmann. " Mesh generation. " In Jörg Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*. Elsevier Science, to appear.

[4] Marshall W. Bern and David Eppstein. " Mesh generation and optimal triangulation. " In F. K. Hwang and Ding-Zhu Du, editors, *Computing in Euclidean Geometry*, pages 23–90. World Scientific, 1992.

[5] Houman Borouchaki and Paul Louis George. " *Delaunay Triangulation and Meshing: Application to Finite Elements*. " Hermes, 1998.

[6] S. Canann, M. Stephenson, and T. Blacker. " Optismoothing: An optimization-driven approach to mesh smoothing. " *Finite Elements in Analysis and Design*, 13:185–190, 1993.

[7] Scott A. Canann, Joseph R. Tristano, and Matthew L. Staten. " An approach to combined laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes. " In *7th International Meshing Roundtable, Sandia National Labs*, pages 479–494, October 1998.

[8] L. Paul Chew. " Constrained Delaunay Triangulations. " *Algorithmica*, 4:97–108, 1989.

[9] E.B. de l'Isle and P. L. George. " Optimization of tetrahedral meshes. " In I. Babushka, W. D. Henshaw, J. E. Oliger, J. E. Flaherty, J. E. Hopcroft, and T. Tezduyar, editors, *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, pages 97–127. Springer-Verlag, 1995.

[10] Hristo Djidjev. " Force-directed methods for mesh improvement. " In *Proceedings of the 2nd Workshop on Algorithm Engineering and Experiments*, pages 29–41, San Francisco, USA, January 2000.

[11] Peter Eades. " A heuristic for graph drawing. " *Congressus Numerantium*, 42:149–160, 1984.

[12] D. A. Field. " Laplacian smoothing and Delaunay triangulations. " *Comm. in Applied Numer. Analysis*, 4:709–712, 1988.

[13] R. Fletcher. " *Practical Methods of Optimization. Volume 1: Unconstrained Optimization*. " John Wiley & Sons, 1980.

[14] L. Freitag and P. Knupp. " Tetrahedral element shape optimization via the jacobian determinant and condition number. " In *Proceedings of the 8th International Meshing Roundtable*, pages 247–258, October 1999.

[15] Lori Freitag. " On combining Laplacian and optimization-based mesh smoothing techniques. " In S. A. Canann and S. Saigal, editors, *Trends in Unstructured Mesh Generation*, pages 37–44. ASME Applied Mechanics Division, 1997.

[16] Lori Freitag, Mark Jones, and Paul Plassmann. " A parallel algorithm for mesh smoothing. " In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, MN, 1997)*, Philadelphia, PA, 1997. SIAM.

[17] Thomas M. J. Fruchterman and Edward M. Reingold. " Graph drawing by force-directed placement. " *Software — Practice and Experience*, 21(11):1129–1164, November 1991.

[18] Tomihisa Kamada and Satoru Kawai. " An algorithm for drawing general undirected graphs. " *Information Processing Letters*, 31(1):7–15, April 1989.

[19] C. K. Lee and S. H. Lo. " A new mesh scheme for the generation of a graded quadrilateral mesh. " *Computers and Structures*, 52(5):847–857, 1994.

[20] S. H. Lo. " A new mesh generation scheme for arbitrary planar domains. " *International Journal for Numerical Methods in Engineering*, 21:1403, 1985.

[21] Kurt Mehlhorn and Stefan Näher. " LEDA: A platform for combinatorial and geometric computing. " *Communications of the ACM*, 38(1):96–102, January 1995.

[22] D. O'Hallaron and J. Shewchuk. " Properties of a family of parallel finite element simulations. " Technical Report CMU-CS-96-141, School of Computer Science, Carnegie Mellon University, 1996.

[23] V.N. Parthasarathy and S. Kodiyalam. " A constrained optimization approach to finite element mesh smoothing. " *Finite Elements in Analysis and Design*, 9:309–320, 1991.

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

[25] M. S. Shephard and M. K. Georges. " Automatic three-dimensional mesh generation by the finite octree technique. " *Int. J. Num. Meth. Engng.*, 32(4):709–749, 1991.

[26] J. Shewchuk. " Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. " In *WACG: 1st Workshop on Applied Computational Geometry*, pages 124–133. LNCS, 1996.

[27] W.T. Tutte. " Convex representations of graphs. " *Proc. Lond. Math. Soc.*, 10:304–320, 1960.

[28] A.M. Winslow. " An irregular triangle mesh generator. " Technical Report UCXRL-7880, National Technical Information Service, Springfield, VA, 1964.

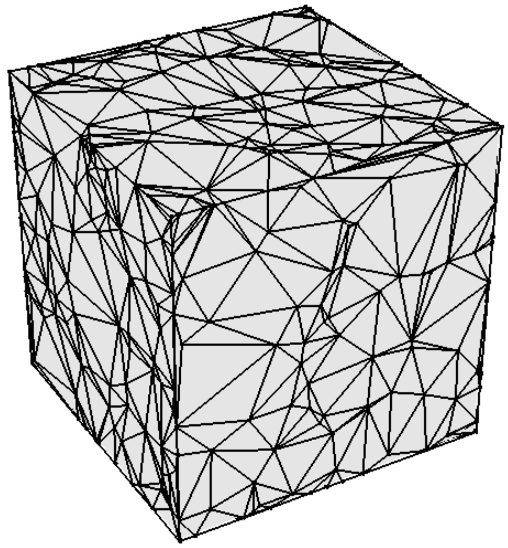

Figure 8: The surface of the test mesh Random Cube (RC).

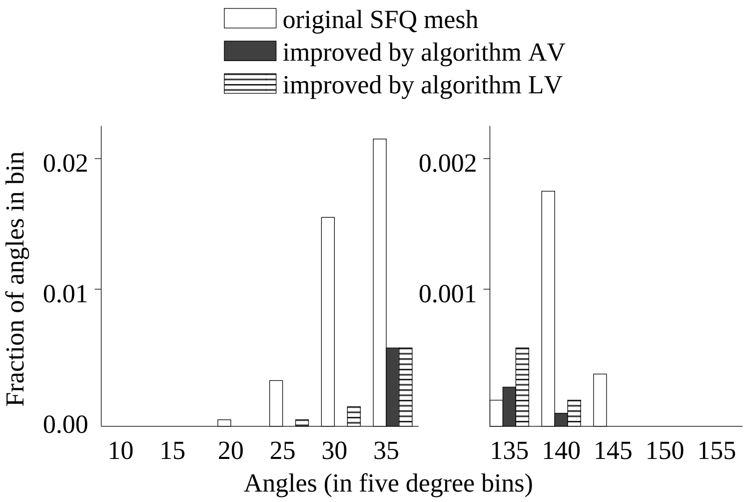

Figure 9: Distribution of the bad angles for 3d test mesh SF Quake smoothed by algorithms AV and LV. There are no angles greater than 152° or smaller than 15°.