Automatic visualization of two-dimensional cellular complexes

L. A. P. Lozada,[⋆] *C. F. X. de Mendonça,*[⋆] *R. M. Rosi,*[⋆] and *J. Stolfi*[⋆]

Institute of Computing, Unicamp
{lplozada,xavier,marcone,stolfi}@dcc.unicamp.br

Abstract

A *two-dimensional cellular complex* is a partition of a surface into a finite number of elements—faces (open disks), edges (open arcs), and vertices (points). The *topology* of a cellular complex is the abstract incidence and adjacency relations among its elements. Here we describe a program that, given only the topology of a cellular complex, computes a geometric realization of the same—that is, a specific partition of a specific surface in three-space—guided by various aesthetic and presentational criteria.

Keywords: Computer graphics, visualization, graph drawing, solid modeling, minimum-energy surfaces, computational topology.

1 Introduction

1.1 Motivation

In the *boundary representation* technique commonly used in computer graphics and engineering, a solid object is defined indirectly by its surface, which is in turn described as the union of a certain number of *faces*—simple surface patches, flat or curved. Faces are bounded by *edges*—line segments, straight or curved—whose endpoints are the *vertices* of the model.

When manipulating such models, it is advisable to handle separately its *topological* properties (the contacts between faces, edges, and vertices) from its *geometrical* properties (vertex coordinates, face equations, etc.) This separation greatly improves the modularity and versatility of geometric algorithms [12,18].

A *two-dimensional cellular complex* is a mathematical structure that captures the topological aspects of such a boundary model, freed from all geometrical data. Informally, it consists of one or more polygons, whose sides are conceptually identified in pairs, in a specified direction. The pairing is conventionally indicated by labeling each side with a letter and an arrow, as in figure 1(a).

⋆ This research was supported in part by the National Council for Scientific and Technological Development (CNPq) of Brazil and the Foundation for Research Support of the State of São Paulo (FAPESP)

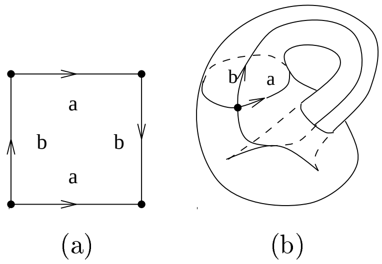


Fig. 1. A Klein bottle.

1.2 Topological visualization

The topology of a cellular complex, even a small one, may be hard to understand intuitively, which makes it hard to debug programs that deal with boundary representations. The topological and geometrical aspects are often handled by independent procedures; so one cannot rely on the visual appearance of the object to verify the correctness of its topology. Faces that may look adjacent on the screen may not be adjacent in the topological data structure, and vice-versa.

Motivated by these difficulties, we set out to develop a graphical tool for *automatically visualizing the topology of a cellular complex*: a program which, given only the incidence relations among the elements of a complex, will choose a surface in three-space, partitioned into points, arcs, and disks, that clearly displays those incidence relations. Thus, for example, given a combinatorial description of the complex shown in figure 1(a), the program should ideally output a picture like 1(b).

1.3 What is a good realization?

A cellular complex has infinitely many representations, but not all of them are useful for understanding its structure. Therefore, in order to develop an automatic visualization tool, we must figure out how to recognize a “good” representation.

Obviously we cannot expect a definitive solution to this problem. We can only list some general criteria, suggested by intuition or experiment, which seem to be associated with visual clarity. For example, it seems desirable that the surface be as smooth and flat as possible, in order to minimize self-occlusions and avoid distracting the viewer's attention with graphical artifacts (folds, shadows, silhouette edges, etc.) which have no topological significance. For the same reason, it is generally desirable that the surface be free from self-intersections; or, if that is not possible (as in the case of a Klein bottle), that the extent of self-intersection be somehow minimized.

Not only must the *surface* be easy to understand, but also the edges of the complex must be drawn on that surface in a visually effective way: they must be well-separated, smooth, as straight as possible, and neither too long nor too short. Notice that these requirements may indirectly affect the shape of the surface. For example, the complex shown in figure 2(a) must be drawn on a surface with the topology of a sphere; however, a truly spherical surface, as in figure 2(b), is visually less effective than the sausage-shaped surface shown in figure 2(c).

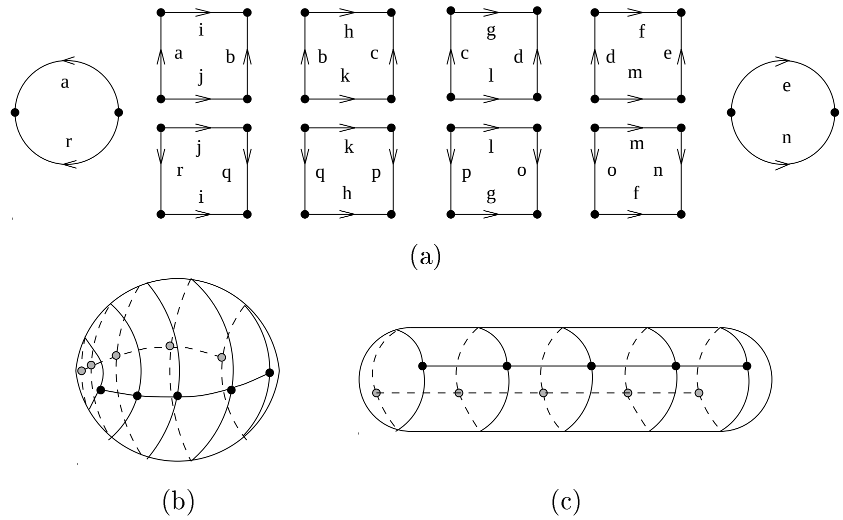


Fig. 2. A complex is more than a surface.

In order to automate the search for a good realization, we borrow a well-established concept from plane graph drawing: that of an *energy function*, a quantitative measure of how badly a solution fails to meet certain visual effectiveness criteria — like the total curvature of the surface, the amount of self-intersection, the variance of edge lengths, and so forth. The problem reduces then to finding, among all possible realizations of the complex, the one which has the minimum energy.

1.4 Related work

This work can be viewed as a three-dimensional extension of the plane graph-drawing problem [8]. Such extensions have been attempted before, but usually by assuming the input to be an ordinary graph, and the output to be only a collection of points and line segments in space, without any surface elements.

Ferguson, Rockwood and Cox [10] addressed the problem of automatically generating a surface with given topology. Since the topology of a surface is completely determined by its genus and orientability, they were able to solve the problem by a direct construction. Our problem differs from theirs because we must not only find a "good" surface with the right topology, but also a "good" drawing of a specified graph on that surface; as we have seen, this requirement influences indirectly the shape of the surface itself.

Another related work is Brakke's `Surface Evolver` [4,5], a general program to study the evolution of surfaces under arbitrary force laws — such as surface tension, elastic bending, gas pressure, etc. Brakke's evolver has been used to empirically determine the surfaces of minimum energy for various topologies [14]. However, the energies used in those experiments were chosen for their mathematical and physical significance, rather than their visual properties. Furthermore, the energy function depended only on the shape of the surface; there was no graph to be drawn on it.

Energy minimization of surfaces with a given topology has also been proposed as a tool for interpolating a surface over a network of curves [15]. An advanced example of this approach is the work by Moreton and Séquin [21], who consider the problem of realizing a cellular complex with a collection of parametric surface patches, joined with $\mathbf{C}_1$ continuity, given the coordinates and tangent planes at the vertices of the complex. Our approach differs from theirs in the choice of surface model (we use a triangular mesh), and in the amount of geometrical data required from the user (we do not require any).

The general idea of using energy functions to quantify the "ugliness" of a drawing apparently became popular after the graph-drawing paper by Kamada [16,7]. Indeed, some of the energy functions that we use are very similar to his "spring" energies—in spirit, if not in detail. Also, some of our energy functions can be viewed as approximations to the bending energy of a thin elastic membrane (generally assumed to be the square of the surface's curvature), which is often used in minimal surface research [13,15,22,4,14,21].

2 Visualization model

The input to our tool is a purely combinatorial data structure that describes the incidence relationships between the faces, edges and vertices of the complex.

In the literature one can find dozens of data structures that were developed for this purpose [6,9,18,25]. For our work we selected the *quad-edge* data structure [12], a variant of the *winged edge* and *half-edge* structures [2] which are widely used in CAD and computer graphics. The main reason for this choice was that the quad-edge allows the encoding of non-orientable cellular complexes, such as the one of figure 1; as well as degree-1 vertices, loops, bridging edges, and multiple edges with same endpoints.

An obvious way to realize a complex geometrically is to model each face as a polynomial surface patch, implicit or parametric, with suitable continuity constraints between adjacent patches [19,1]. However, in a general complex a face may have any number of sides (including 1 or 2), and may be glued in arbitrarily complex ways. Thus, in general, it is not possible to realize each face as a single polynomial surface patch of bounded degree.

Therefore, instead of using the faces as modeling units, we use what we call the *tiles* of the complex. Each tile is a four-sided surface patch, containing exactly one edge e of the complex and the corresponding edge e' of the dual complex, both running diagonally across the tile. See figure 3.

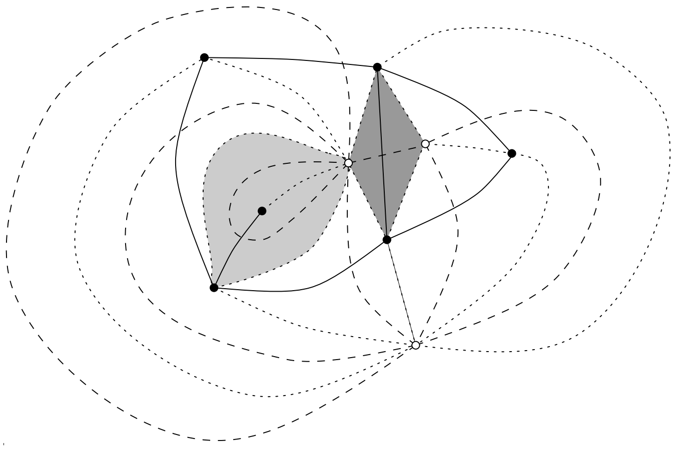


Fig. 3. A two-dimensional complex drawn on the plane (solid), overlaid with its dual complex (dashed), and the tile boundaries (dotted).

2.1 Modeling a tile

Since every tile has only four sides, and therefore only four neighboring tiles, it is possible to realize it as a geometric object of bounded complexity.

A obvious candidate for this role would be a parametric polynomial patch [19,21]. This approach would allow us in principle to obtain a truly smooth surface, with continuity of tangent plane (and possibly curvature) between adjacent tiles. However, it is not at all trivial to enforce those constraints for complexes arbitrary topology. Also, some of the energy functions we use seem hard to evaluate in this model. Given these difficulties, we decided to model each tile as a polyhedral surface; specifically, as a grid of $k \times k$ four-sided *cells*, each consisting of four plane triangles. See figure 4. We thus replace the original cellular complex C, with m edges, by a refined complex T having $6mk^2$ edges and $4mk^2$ faces, all triangles. The elements of C are unions of elements of T; in particular, the edges of C are polygonal paths in T, running diagonally across the corresponding tiles.

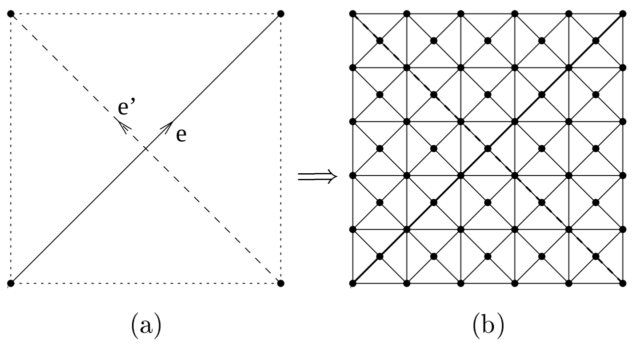


Fig. 4. The triangulated tile model.

Obviously, having adopted a polyhedral model, we must give up any hope of obtaining a really smooth surface, and seek instead to reduce and equalize the external dihedral angles between adjacent triangles. Still, by using a large enough tile order k, we can in principle obtain surfaces that are arbitrarily smooth almost everywhere. Moreover, we can always use standard tricks of computer graphics, such as Gouraud shading [11], to smooth out the corners when rendering the final image.

2.2 Building the triangulation

We use the quad-edge data structure to represent not only the input complex C, but also the refined mesh T. (There are specialized data structures for triangular meshes which use less space than the quad-edge; but the latter is more convenient in the intermediate stages of the tiling procedure, when the mesh T still contains some non-triangular faces.)

After building a triangulated $k \times k$ tile for each edge of the original complex C, we glue the sides of those tiles in pairs, as prescribed by the adjacency relation between the corresponding edges of C.

Note that it is possible for a tile to get glued to itself; that happens, for instance, when the complex C includes faces or vertices of degree 1. In those cases, the resulting mesh may contain pairs of *twin triangles* with the same three vertices, or pairs of *twin edges* having the same endpoints.

Figure 5 illustrates this problem, in this case when the bottom side of a 2×2 triangulated tile (a) gets glued to the right side of the same tile (b,c). Note that in figure 5(c) there are two pairs of twin triangles, highlighted in gray; and two pairs of twin edges, inside and surrounding the gray area..

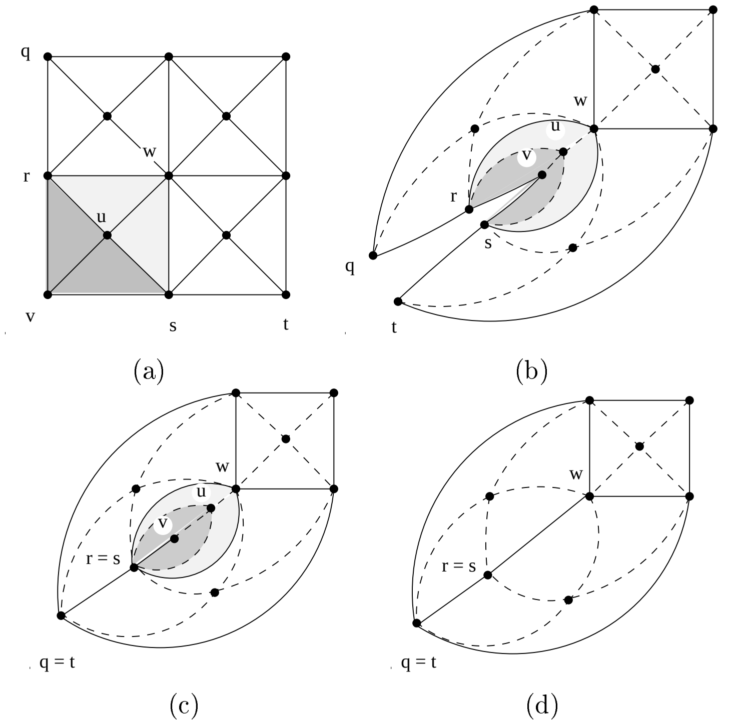


Fig. 5. Gluing a tile to itself.

Obviously, two such triangles will coincide no matter what coordinates we assign to their vertices. If they are left in the mesh, they will look like a loose flap hanging out from the surface. Thus, whenever gluing a tile to itself, we must locate and remove any twin pairs, as shown in figure 5(d).

Fortunately, this cleanup is quite easy. It can be shown [24] that any twin triangles or edges must belong to a grid cell that is adjacent to the two sides of a tile that are being glued together, like the highlighted corner of figure 5(a). As long as k is three or more, the removal of these corner cells does not create any new twin pairs, and original edges of the complex—represented by the tile diagonals—will not be entirely obliterated.

3 Energy functions

Since each tile is modeled by a set of *flat* triangles, the shape of the triangulated mesh is completely determined by the coordinates of its vertices $\mathcal{V}T = \{v_1, .. v_n\}$; we say that these n points of $\mathbf{R}^3$ are a *configuration* of the mesh. Our energy function, which measures the "ugliness" of the surface, is therefore a function from $(\mathbf{R}^3)^n$ to $\mathbf{R}$.

The energy functions we have tried were convex combinations $E = \sum_i w_i E_i$ of the functions E_i described below, with fixed weights w_i. In these formulas, $\mathcal{V}C$, $\mathcal{E}C$, $\mathcal{D}C$, and $\mathcal{F}C$ denote the vertices, primal edges, dual edges, and faces of a complex C. Also $\vec{\mathcal{E}}C$ and $\vec{\mathcal{D}}C$ denote the set of all directed edges, respectively primal and dual.

Bending energy:

$$E_{\text{bnd}} = \sqrt{|\mathcal{F}T|} \sum_{e \in \mathcal{E}T} l_e \theta_e^2$$

where l_e is the length of edge e, and θ_e is the external dihedral angle at that edge. Minimizing E_{bnd} tends to flatten out the surface, and distribute its curvature evenly among all edges.

Eccentricity energy:

$$E_{\text{ecc}} = |\mathcal{V}T| \sum_{v \in \mathcal{V}T} |v - b_v|^2$$

where b_v is the barycenter of all neighbors of vertex v. Minimizing E_{ecc} also tends to flatten out the surface, and equalize the edge lengths.

Angle variance energy:

$$E_{\text{ang}} = \frac{1}{|\mathcal{V}T|} \sum_{e \in \vec{\mathcal{E}}T} \left| \phi_e - \frac{2\pi}{d_e} \right|^2$$

where ϕ_e is the angle between e and the next edge out of the same vertex, projected onto the tangent plane at that vertex; and d_e is the degree of that vertex. Minimizing E_{ang} tends to equalize the angles between adjacent edges, which helps to unfold the surface and reduce its self-intersections.

Spreading energy:

$$E_{\text{spr}} = \frac{1}{|\mathcal{V}T|} \sum_{v \in \mathcal{V}T} |p_v|^2$$

where p_v is the Cartesian coordinates of vertex v. Minimizing this energy tends to keep all vertices close to the origin of $\mathbf{R}^3$.

Proximity energy:

$$E_{\text{prx}} = \frac{1}{|\mathcal{F}T|^2} \sum_{\substack{r,s \in \mathcal{F}T \\ r \neq s}} \frac{1}{|c_r - c_s|^2 + \rho_r^2 + \rho_s^2}$$

where c_r, c_s are the centroids of triangles r and s, and ρ_r, ρ_s are their average radii (in the root-mean-square sense). This energy can be understood as the electrostatic potential of a set of fuzzy electric charges, located at the triangle centroids. Minimizing E_{prx} tends to spread out the triangles in space, thus avoiding self-intersections (especially grazing ones) and fold-overs.

Patch area energy The two diagonals of a tile divide it into four triangular pieces, each consisting of k^2 triangles. Let $A_1, \ldots A_{4m}$ be the total areas of these quarter-tiles. Then

$$E_{\mathrm{par}} = \frac{1}{4m} \sum_{i=1}^{4m} \left(\frac{A_i}{A_*} + \frac{A_*}{A_i} - 2 \right)$$

where $A_* = \pi/m$ is the "ideal" area of a quarter-tile. Minimizing E_{par} tends to keep the total surface area close to 4π, and to equalize the quarter-tile areas, so that the edges of the original complex are spread out uniformly over the surface.

Note that the energy functions above *must* be used in combination, rather than alone, in order to avoid degenerate configurations. For instance, the proximity energy E_{prx} tends to its minimum value (zero) when the vertices tend to infinity; whereas the bending energy E_{bnd} is minimum when all vertices are at the origin. However, the asymptotic behavior of the two formulas is such that any nontrivial convex combination of them will attain its minimum at configurations of bounded and nonzero radius.

The purpose of the scaling factors in the formulas above, such as $\sqrt{|\mathcal{F}T|}$ in the formula of E_{bnd}, is to make the numerical value of each energy largely insensitive to the number of triangles in the mesh. This scaling allows us to use the same weights for optimizing meshes of different resolutions (different values of k). In particular, it allows us to use the *multi-scale approach*, which consists in running the optimization algorithm simultaneously on several meshes of varying resolution, which are loosely coupled by interpolation and local averaging.

3.1 Selecting the weights

When it comes to selecting the weights, we still do not have any answer much better than trial-and-error, perhaps guided by some physical intuition.

It would be trivial to implement a GUI-based tool that allowed direct adjustment of the weights by the user, with visual feedback. Unfortunately, our optimizer is still too slow for interactive use: it would take tens of minutes to (re)compute the optimum configuration after a change in the weights. (On the other hand, considering our limited expertise in numerical methods, it seems likely that this time can be reduced, by several orders of magnitude, merely by using better optimization algorithms.)

A more speculative solution is to let the computer "learn" the weights from examples, as Eades and Mendonça did for the plane graph drawing problem [20]. The idea is to give the computer a collection of "good" realizations of cellular complexes—generated, say, by ad-hoc programs or manual editing with solid modeling tools—and let the computer find the combination of weights that comes closest to reproducing those shapes.

4 Optimization

Having constructed the triangular mesh, and chosen an energy function, we are faced with problem of finding the configuration of minimum energy. This is a nontrivial optimization problem, since even a small complex determined an energy function with hundreds of independent variables.

We approach this problem at two levels. We use general-purpose non-linear optimization techniques to move from given configuration to a nearby *local* minimum. We repeat this search for a couple dozen random starting points, and (select the local minimum with lowest energy as the answer. It should be possible to use combinatorial heuristics, such as simulated annealing, to extend the search beyond the nearest local minimum; but we haven't been able to get such methods to work fast enough.)

When searching the local minimum, we need a stopping criterion of some sort. Unfortunately, there is no numerical criterion that can tell when the energy is "low enough" to provide a good visualization. So, in practice, we fix a computation budget, and use the best configuration we can find within that limit.

4.1 General optimization methods

To solve this problem, we have tried several general purpose numerical optimization techniques, including Kirkpatrick's *simulated annealing* [17,23], Nadler and Mead's *downhill simplex method* [23, p.289], Powell's *principal directions* method [23, p.298], the naive *single coordinate* optimization, with periodic diagonal steps [23, p.294] and a *gradient descent* method with adaptive step-size.

The tests were performed on various triangulated meshes, ranging from a few tens to a few hundred vertices. The initial guess for the optimization was either a random configuration—where each vertex was chosen independently and uniformly in the unit cube—or a reasonably smooth configuration obtained by the heuristic methods described in the next section. The effectiveness of an algorithm was judged from its energy evolution curve: the energy of the minimum configuration found, as a function of accumulated CPU time.

The ranking of the methods was generally the same on all tests, and consistent over time. Not surprisingly, we found that gradient descent was the most effective. Simulated annealing was so slow that we gave up on it after a few tests. The Nadler-Mead and Brent-Powell algorithms were faster, but not enough to be usable. Moreover, they require $\Omega(n^2)$ storage for a function of n variables, and therefore are restricted to complexes with a few tens of edges.

The naive minimization method consists of adjusting one variable at a time (using, for instance, Brent's univariate minimization algorithm [23, p.283]), while all other variables are held fixed. As recommended in the literature, every $n+1$ such "axial" steps we perform a "diagonal" step, where we seek the minimum configuration along the line connecting the outcomes of the first and the last of those axial steps.

We implemented this naive method only for the sake of comparison, since textbooks generally claim it is slower than Brent-Powell. To our surprise, it turned out to be much faster. The reason, which was obvious on hindsight, is that all our energy functions are the sum of many terms, each depending on a few vertices only. When varying one coordinate at a time, we could save time by recomputing only the terms that depended on that coordinate. Thus, while the naive algorithm performed somewhat more energy evaluations than Powell's method, each evaluation was faster by one or two orders of magnitude.

In the gradient descent method, we compute the partial derivatives of the energy function by Baur and Strassen's method [3]. The latter, which is basically a systematic use of the chain derivation rule, reduces the cost of computing the gradient of *any* algebraic or transcendental formula to a small constant times the cost of computing the energy itself.

4.2 Heuristic methods

Besides these general-purpose optimization algorithms, we have also used two specialized heuristic methods, the *smoothing heuristic* and the *spreading heuristic*. Both are local operations that are applied to each vertex v in turn, cyclically, up to a prescribed number of passes. The *smoothing heuristic* adjusts the position of vertex v, keeping all other vertices fixed, so as to approximately minimize the bending energy of the edges between the triangles that are incident to v. The *spreading heuristic* keeps the vertex v fixed, but rotates its neighbors $u_1, .. u_k$ around the surface normal r at the vertex so as to better equalize the angles $u_i v u_{i+1}$, when projected on a plane perpendicular to r.

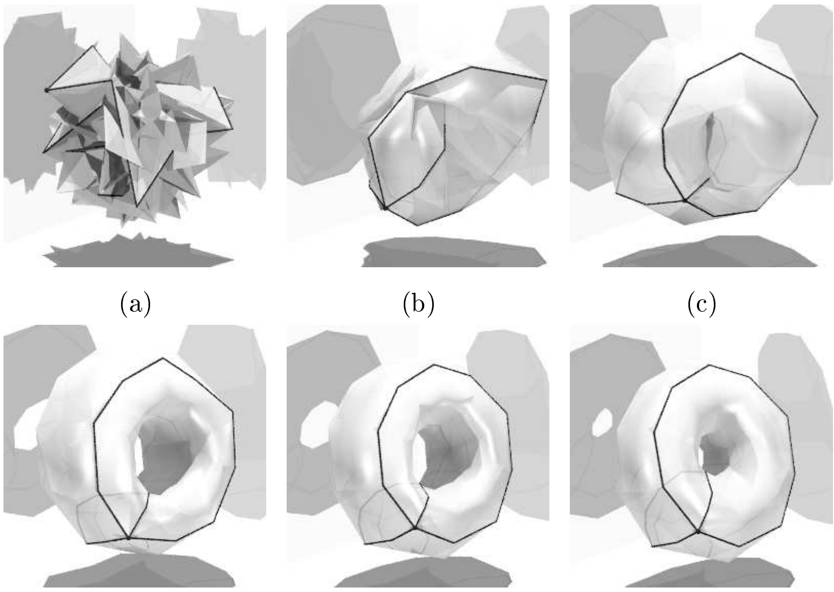
These heuristics are not sensitive to the energy functions or their weights. Therefore, they are useful only as a preprocessing step, when starting from a random configuration, where the goal is merely to untangle the surface and smooth out its largest wrinkles — which are bad under any reasonable energy function. If we want to compare the effect of different energy functions and varying weights, we must eventually follow these heuristics with one of the generic optimization methods above.

Each pass of the heuristic redistributes the "stress" energy of each element among the neighboring elements. It should be noted that, as in any diffusion process, the number of passes needed to achieve a given surface smoothness is expected to scale as the square of the graph-theoretic diameter of the triangulation.

In our test runs, with complexes of a few tens of edges, we found that 20-100 passes of each heuristic already provided a reasonably smooth surface; additional passes were hardly worth their cost.

5 Results

Figure 6 shows the heuristic smoothing of a simple torus-like complex. The original complex had one face, two edges, and one vertex; it was modeled with tiles of order $k = 5$, resulting in a mesh with 300 edges, 200 triangles and 100 vertices. Shown are the initial random configuration (a), and the states after 5, 30, and 100 passes of each heuristic, respectively (b-d).



(a) (b) (c)

(d) (e) (f)

Fig. 6. Smoothing of a torus complex.

The configuration of figure 6(d) was then optimized with the gradient-descent method, for the energy $E = E_{\mathrm{bnd}} + E_{\mathrm{par}} + E_{\mathrm{ecc}} + 5E_{\mathrm{prx}}$ Figures 6(e-f) show the state after 100 and 1000 energy evaluations, respectively.

Figure 7 shows renderings of the final stage of optimization for three different complexes with the topology of a sausage, a five-star and a tritorus (a sphere with three handles).

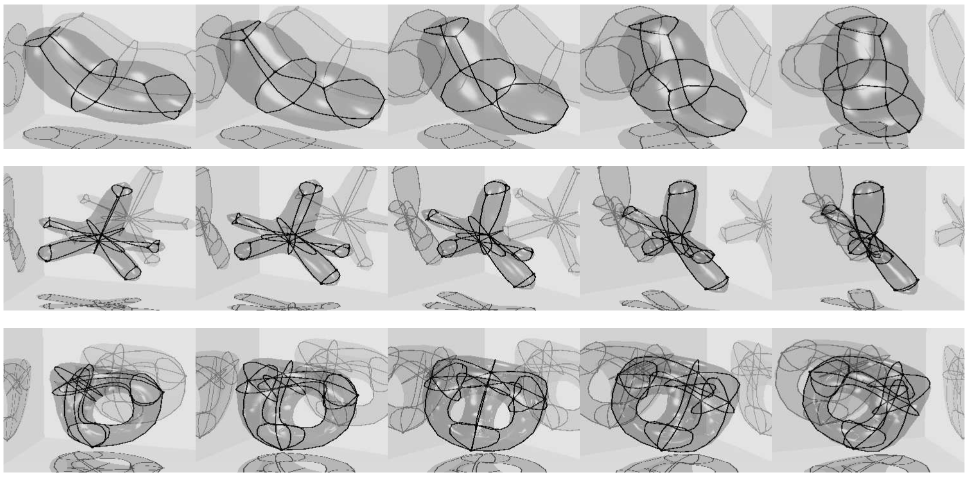


Fig. 7. Realizations of a sausage, a five-star, and a tritorus.

The renderings are further enhanced by a little animation displaying several views of the complexes. The series of pictures are presented as a sequence of stereoscopic pairs. The 3D rendering may be seen by holding the paper about 50 cm away, and converging the eyes toward a point behind it, so that pairs of consecutive drawings are fused into one image. The resulting stereo illusion should compensate for the the small size of the images.

The images were produced with `POV-Ray`, a freely available ray tracer [26]. Their apparent smoothness is due to Gouraud-shading of the triangles [11]. The edges and vertices are modeled by thin cylinders and small spheres.

6 Conclusions and future work

Our experiments to date are encouraging, but there is still a lot of work to be done here. To begin with, we need to work more on the energy minimization code; speedup several orders of magnitude seem possible, just by using better optimization algorithms.

We still understand very little about the effects of the various energy functions, and the proper weights to use. Among other things, we need to develop energy functions that penalize self-intersections more strongly than the functions we have got. We also need to improve the heuristic methods so as to avoid generating self-intersecting shapes.

Finally, we need to develop combinatorial optimization tools or heuristics to extend the search over several local minima.

References

1. Chandrajit L. Bajaj. Smoothing polyhedra using implicit algebraic splines. In *Proc. SIGGRAPH'92*, pages 79–88, 1992.
2. Bruce G. Baumgart. A polyhedron representation for computer vision. In *Proc. 1975 AFIPS National Computer Conference*, volume 44, pages 589–596, 1975.
3. Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983.
4. Kenneth A. Brakke. The Surface Envolver. *Experimental Mathematics*, 1(2):141–165, 1992.
5. Kenneth A. Brakke. *Surface Evolver Manual*. The Geometry Center, University of Minnesota, December 1993. Eletronic Address: brakke@geom.umn.edu.
6. Erik Brisson. Representing geometric structures in d dimensions: Topology and order. *Proc. 5th Annual ACM Symp. on Computational Geometry*, pages 218–227, June 1989.
7. Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. Technical report, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, 1989.
8. G. Di Battista, P. D. Eades, and R. Tamassia. Algorithms for drawing graphs: An annotated bibliography. Technical report, Department of Computer Science, University of Newcastle, 1993.
9. David P. Dobkin and Michel J. Laszlo. Primitives for the manipulations of three-dimensional subdivisions. *Proc. 3rd ACM Symp. on Comp. Geometry*, pages 86–99, June 1987.
10. H. Ferguson, A. Rockwood, and J. Cox. Topological design of sculpted surfaces. In *Proc. SIGGRAPH'92*, pages 149–156, 1992.
11. James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, second edition, 1990.
12. Leonidas Guibas and Jorge Stolfi. Primitives for the manipulations of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.
13. B. K. P. Horn. The curve of least energy. *ACM Transactions on Mathematical Software*, 9(4):441–460, December 1983.
14. Lucas Hsu, Rob Kusner, and John Sullivan. Minimizing the squared mean curvature integral for surfaces in space forms. *Experimental Mathematics*, 1(3):191–207, 1992.
15. Michael Kallay and Bahram Ravani. Optimal twist vectors as a tool for interpolation a network of curves with a minimum energy surface. *Computer Aided Geometric Design*, pages 465–473, 1990.
16. Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, pages 7–15, April 1989.
17. S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
18. Pascal Lienhardt. Subdivisions of N-dimensional spaces and N-dimensional generalized maps. *Proc. 5th Annual ACM Symp. on Computational Geometry*, pages 228–236, June 1989.
19. Charles Loop and Tony DeRose. Generalized B-spline surface of arbitrary topology. In *Proc. SIGGRAPH'90*, pages 347–356, August 1990.
20. Cândido X. F. Mendonça and Peter Eades. Learning aesthetics for visualization. In *Anais do XX Seminário Integrado de Software e Hardware*, pages 76–88, Florianópolis, SC (Brazil), 1993.

21. Henry P. Moreton and Carlo H. Séquin. Functional optimization for fair surface design. In *Proc. SIGGRAPH'92*, pages 167–176, 1992.
22. D. B. Parkinson and D. N. Moreton. Optimal biarc-curve fitting. *Computer-Aided Design*, 23(6):411–419, 1991.
23. William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.
24. Rober Marcone Rosi and Jorge Stolfi. Automatic visualization of two-dimensional cellular complexes. Technical Report IC-96-02, Institute of Computing, University of Campinas, May 1996.
25. Fujio Yamaguchi and Toshiya Tokieda. A solid modelling system: Freedom-II. *Computers & Graphics*, pages 225–232, 1983.
26. C. Young, D. K. Buck, and A. C. Collins. POV-Ray - Persistence of Vision Raytracer, version 2.0. 1993.