Automatic visualization of two-dimensional cellular complexes

Rober Marcone Rosi `marcone@inf.ufes.br`

Jorge Stolfi `stolfi@dcc.unicamp.br`

Instituto de Computação
Universidade Estadual de Campinas

Relatório Técnico IC–96-02

Maio de 1996

Automatic visualization of two-dimensional cellular complexes

Rober Marcone Rosi
marcone@inf.ufes.br

Jorge Stolfi
stolfi@dcc.unicamp.br

Instituto de Computação
Universidade Estadual de Campinas *

May 31, 1996

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 consists of 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.

1 Introduction and motivation

1.1 Cellular complexes

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 one or more *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 building or using such a model, it is generally convenient 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 [15, 23].

A *two-dimensional cellular complex* is a mathematical structure that captures the topological aspects of such a boundary model, freed from all geometrical data. Formally, it can be defined as a finite set S, a permutation σ of the elements of S, and a set η of triples (r, d, s), where $r, s \in S$, $r \neq s$, and d is a sign (either + or −); and such that every element of S occurs exactly twice in these triples.

*This reasearch was supported in part by the National Council for Scientific and Technological Development of Brazil (CNPq), the Foundation for Research Support of the State of São Paulo (FAPESP), and by UNICAMP's Teaching and Research Support Fund (FAEP).

Intuitively, the elements of S represent the sides of one or more polygons; $\sigma(s)$ is the side following s around the same polygon, in counterclockwise order; and each triple (r, d, s) specifies that side r must be identified with side s, taken in the same sense if $d = +$, or in opposite senses if $d = -$.

1.2 Topological visualization

The topology of a cellular complex, even a small one, may be quite hard to understand from its combinatorial description. For instance, a complex with a single four-sided face may define four different surface topologies (sphere, torus, Klein bottle, and projective plane), depending on how the sides are glued to each other [16, 1]. See figure 1.

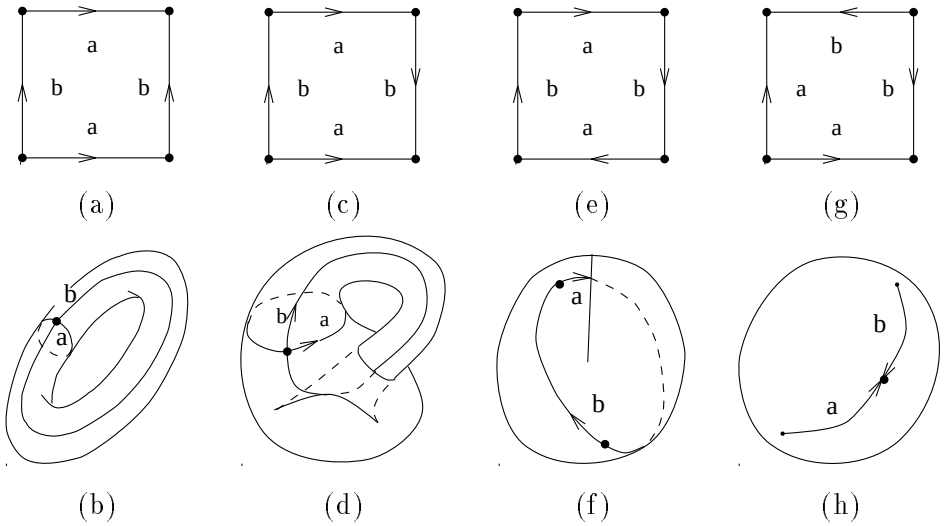


Figure 1: The four distinct cellular complexes with one four-sided face: (a,b) torus, (c,d) Klein bottle, (e,f) projective plane, (g,h) sphere. Sides with same label are identified as shown by the arrows.)

The large psychological distance between combinatorics and topology is a major obstacle when developing and debugging algorithms that deals with boundary models. Since the topological and geometrical sections of such an algorithm are largely independent of each other, one cannot rely on the visual appearance of the object to verify the topological information. Faces that look adjacent on the screen may not be adjacent in the topological data structure, and vice-versa.

Motivated by this difficulty, we developed and implemented an agorithm for the *automatic visualization of the topology of a cellular complex*. Given only the number of elements in the complex, and the incidence relations among them, our algorithm constructs a specific geometric surface in three-space, and a specific partition of it into discs, arcs and points, that displays those incidence relations—hopefully, in a visually effective way. Thus, for example, given a combinatorial description of the complex shown in figure 1(a), our algorithm will automatically generate a picture like 1(b).

1.3 What is a good realization?

Any cellular complex has infinitely many geometric representations; and not all of them are helpful for understanding its structure. In order to choose a "good" representation, we will rely on some general geometric properties which, according to intuition and experiment, seem to be associated with visual clarity.

For one thing, 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 are not features of the complex itself. For the same reason, it is 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. Note that these requirements may indireclty 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), seems less appropriate than the sausage-shaped surface of figure 2(c).

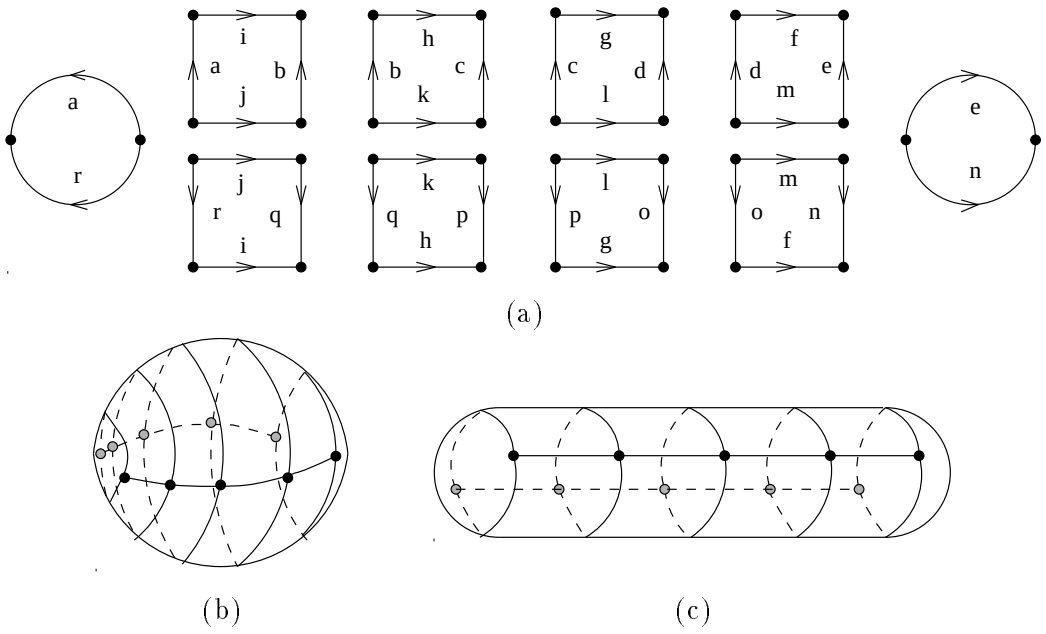


Figure 2: A cellular complex (a), and two drawings of the same: on a spherical surface (b) and on a sausage-like surface (c).

1.4 Energy functions

In order to formalize those requirements, we borrowed a tool from our colleagues who have been working on the plane graph drawing problem: the concept of an *energy function* [20, 9], a quantitative measure of how badly a solution fails to meet a certain visual effectiveness criterion. For example, we can define numerical functions that measure the total curvature of the surface, the amount of self-intersection, the variance of edge lengths, and so forth. Once we select such a function as being *the* measure of badness, we only need to find, among all possible realizations of the complex, the one which has the minimum energy.

1.5 Related work

This work can be viewed as a three-dimensional extension of the plane graph-drawing problem [11, 21]. Such extensions have been considered before; however, it is generally assumed that the input is an ordinary graph, and the output is merely a collection of points and line segments in three space, without any surface elements.

Ferguson, Rockwood and Cox [13] addressed the problem of automatically generating a surface with given topology. Since the topology of a *surface* is completely determined by its connectivity (an integer) and its orientability (a single bit), they were able to solve the problem by a direct construction. We cannot use this approach for our problem, since we must find both a "good" surface with the right topology, and "good" drawing of a specified graph on that surface. As we observed in section 1.3, these two sub-problems cannot be solved separately.

Another related work is Brakke's `Surface Evolver` [6, 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 determine empirically the surfaces of minimum energy for various topologies [19]. However, the energies used in those experiments were chosen for mathematical and physical significance, rather than visual effectiveness. Furthermore, the energies depended only on the shape of the surface, there being no graph to be drawn on it.

A related problem, with obvious practical interest, is that of optimizing a mesh of given topological type so as to best approximate a given set of data points in $\mathbb{R}^3$. A fairly general solution was given by Hoppe, DeRose, and others [17]. Their approach is based, like ours, on minimizing an "energy" that is a function of the mesh coordinates. However, the similarity ends there. Their energy function measures only the closeness of fit to the data points and the "economy" of the mesh (number of vertices and total squared edge length), without concern for smoothness or visual effectiveness. More importantly, they assume that a good initial guess for the mesh shape is given as input to their algorithm.

Energy minimization of a fixed topology has also been proposed as a paradigm for solid modeling, for example in the design of smooth filets and chamfers; for the smooth interpolation of space curves [8, 30]; and for interpolation of scattered data points in $\mathbb{R}^3$. However, in those works the surface to be optimized generally has a very simple topology—a disk, possibly with a few holes—and its boundary curve is fixed and known. Thanks to these conditions, those authors could model the surface by a regular grid of triangles or

Bézier patches; and a simple interpolation of the boundary curves provides a good initial approximation to the optimum shape. Thus, the energy optimization does not have to worry about multiple minima or degenerate solutions, which are a major concern in our case (see section 3).

Moreton and Séquin [27, 26] considered the problem of realizing a general cellular complex with a collection of parametric surface patches, joined with $\mathbb{C}_1$ continuity, given the cordinates and tangent planes at the vertices of the complex. They solve the problem in two stages, first choosing polynomial curves for the edges of the complex, then fitting surface patches to those edges. Our approach differs from theirs in the nature of the surface model (we use a mesh of flat triangles) and in the input data required by the program (we do not require any geometrical data, not even vertex coordinates).

The general idea of using general "energy functions" to quantify the "ugliness" of a drawing apparently became popular after the paper by Kamada [20]. Indeed, some of the energy functions that we use are very similar to Kamada's "spring" models—in spirit, if not in detail. Other energy functions that we use are similar to the integral of the bending energy (generally assumed to be the square of the surface's curvature), which is often used as *the* penalty function in minimal surface research [18, 8, 5, 19, 27, 28].

2 Visualization model

2.1 Encoding the cellular complex

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 [3, 12, 7, 31, 23]. For our work, we selected the *quad-edge* data structure [15, 24], a variant of the *winged edge* and *half-edge* structures [3], widely used in CAD and computer graphics. The main reason for this choice was that the quad-edge structure allows the encoding of non-orientable cellular complexes, such as the one of figure 1(c,d).

The quad-edge structure also allows degree-1 vertices, loop edges, multiple edges with the same two endpoints, and faces which are incident to both sides of the same edge. These features require special attention in our algorithms, as we shall see.

On the other hand, the quad-edge structure requires that every face be equivalent to a disk—just as in the mathematical definition of a cellular complex. This restriction greatly simplifies the data structure and its use.

Some solid modeling data strutures are more liberal in this respect, allowing faces with two or more border loops (i.e., with one or more holes). Models that include such features can still be handled by our algorithm, provided we first "cut open" any face with $k \geq 2$ loops, by adding $k-1$ extra edges between those loops. Fortunately, this preprocessing step is quite trivial, since it is purely topological—it is not necessary to specify the shape and position of those extra edges.

Some data structures also allow models with "free borders", namely edges that have only one face incidence, instead of two. In other words, those structures allow some sides of some faces to remain unglued. The resulting surface has one or more "holes", each of them

surrounded by a closed ring of unglued edges. For example, if we glue two opposite sides os a square, leaving the other two unglued, we get either an open-ended cylinder (two holes) or a Möbius strip (one hole).

In order to handle such models with out tool, we must first "close" each hole with an extra "ghost" face. This operation can be easily performed in linear time; the resulting complex defines a borderless surface (technically, a compact two-dimensional topological manifold), and can be represented by the quad-edge structure. Our tool is programmed to ignore those ghost faces during shape optimization, and also when rendering the optimized surface.

2.2 Tiles

An obvious way to model the geometry of the surface is to model each face of the complex as a piece of polynomial surface, implicit or parametric, with continuity constraints imposed on pairs of adjacent patches [25, 2]. However, the faces of the complex may have any number of sides, which may be glued among themselves in almost any fashion. In general, it is not possible to realize such a face as a single polynomial surface patch of bounded degree.

Therefore, instead of viewing the surface as the union of faces, we view it as the union of *tiles*. Each tile is a four-sided surface patch, containing exactly on edge e of the complex. See figure 3.

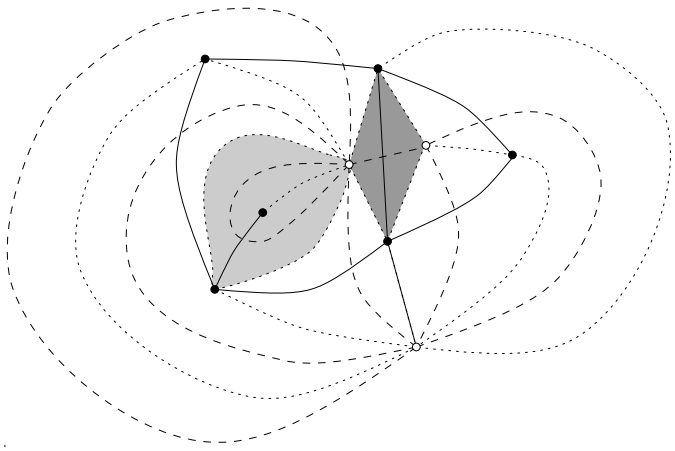


Figure 3: A two-dimensional complex on the plane, showing the primal edges (solid), dual edges (dashed), and tile boundaries (dotted).

The other diagonal of the tile can be taken to represent the edge e^* of the *dual complex* [15]. The dual C^* of a complex C is a repartition of the same surface: it has one vertex inside each face of C, and vice-versa; and each edge of one complex crosses, once and transversally, exactly one edge of the other.

2.3 Realizing a tile

Since each tile has only four sides, and therefore only four neighboring tiles, we can in principle realize it as a geometric object of bounded complexity.

One obvious alternative is to use a parametric polynomial patch [25, 27]: a polynomial mapping of the unit square $[0 \,..\, 1]^2$ of $\mathbb{R}^2$ to $\mathbb{R}^3$. The main difficulty of this approach is to guarantee a smooth join (with tangent plane continuity, and possibly curvature continuity) between adjacent tiles. Moreover, we must be careful to avoid degeneracies—points where the tangent plane is undefined—inside the tiles.

These constraints are rather difficult to enforce in our tiling model. The main source of difficulty is that a tile may have to be joined to itself (see, for example, the light gray tile in figure 3. In order to get continuity without degeneracy at the corner between those two sides, we must use polinomials of degree 5 or more. Note that the first partial derivatives of the polynomials are necessarily zero at that corner, so the tangent-plane continuity constraint must be expressed in terms of higher-order derivatives.

Moreover, if the complex has faces and vertices of high degree, it may be necessary to use higher-degree polynomials merely to allow the tiles to connect to each other in the required topology. Higher degrees may also be required in order to reduce the number of self-intersections and to keep the elements of the complex well-separated.

Finally, many of the energy functions we considered would be hard to evaluate for polynomial patches, since they would be expressed as complicated surface integrals which probably not admit a closed form.

2.4 Triangulated tiles

Given these difficulties of the polynomial approach, we decided to model each tile as a polyhedral surface; specifically, a grid of $k \times k$ four-sided *cells*, each consisting of four plane triangles. See figure 4.

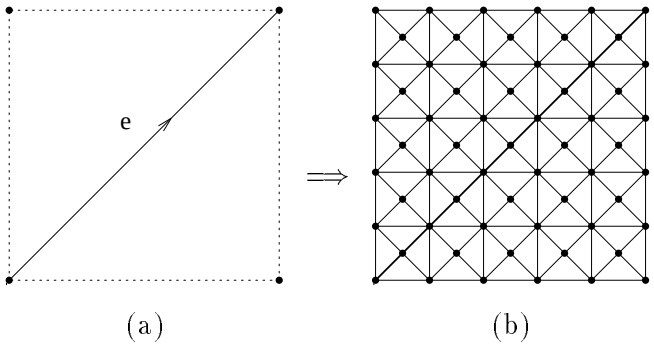


Figure 4: The tile (a) for some edge e, modeled as a 5×5 grid (b).

In other words, we replace the original cellular complex C, with m edges, by a refined complex T having $6mk^2$ edges and $4mk^2$ triangles. The elements of C are unions of elements

of T. In particular, the vertices of C are a subset of the vertices of T, and the edges of C are polygonal paths in T, running diagonally across the corresponding tiles.

Obviously, having adopted a polyhedral model, we must give up any hope of obtaining a really smooth surface; instead, we may seek to reduce and equalize the external dihedral angles between adjacent triangles. In this way, by using a large enough tile order k, we can in principle obtain surfaces that are arbitrarily "smooth" almost everywhere. Moreover, we can always blur out the corners when the surface is rendered, by standard computer graphics tricks such as Gouraud shading [14].

The best value for the parameter k depends on the complex and on the intended application. In some cases, we may be able to get by with $k = 1$ or $k = 2$. However, as we shall see, we must take $k \geq 3$ if the complex includes vertices or faces of degree 1; and we may want to use a larger k in order to obtain a smoother surface. Also, for complexes with intricate topology, we may need to use a larger k to allow the surface to fold properly, or to reduce the number of self-intersections, or to keep the elements of the complex well separated.

Note that, if we had opted for polynomial patches instead of triangular meshes, these same reasons would force us to use polynomials of higher degree. Indeed, a triangulated tile of order $k \times k$ has about the same number of degrees of freedom as a polynomial patch of degree $k\sqrt{2}$ in each variable.

We do not yet have any reliable rules for choosing the value k for agiven complex. We can only remark that that $k = 5$ was sufficient for all the examples we tried.

2.5 Building the triangulation

We will now describe in detail how to construct the triangulation T from the given complex C. The first step is to build a triangulated $k \times k$ tile for each edge of C. We will denote by T_{u} the union of all these triangulated tiles.

Next the sides of these tiles are glued in pairs, as prescribed by the adjacency relation between the edges of C. If, in the complex C, edge b follows edge a in counterclockwise order out of some vertex v, then the left side of a's tile, from bottom to top, gets glued to the bottom side of b's tile, from left to right. The "gluing" entails the identification of all corresponding vertices and edges of T_{u} that make up those two sides. We shall denote the triangulated mesh that results by T_{g}.

Finally, the mesh T_{g} is subjected to a "cleanup" procedure, that removes any topological "degeneracies" which may have been introduced by the gluing step. The nature of those degeneracies, and their removal, are described in the next section. The outcome of this step is the desired triangulation T.

To encode the topology of the mesh T, and of the partial meshes that arise during its construction, we use the same quad-edge representation that we use for the original complex. (We could have saved some space by using a data structure specialized for triangular meshes; however, the generality of the quad-edge is quite valuable here, since the partial meshes often have non-triangular faces. In particular, because the quad-edge structure does not allow free borders, each tile of T_{u} includes a dummy exterior face with $4k$ sides, that completes it to a sphere. These exterior faces disappear automatically during the gluing step.)

2.6 Degeneracies

Note that the gluing procedure above may end up gluing a tile to itself, sometimes even forming a one-sided surface. In those cases the resulting mesh may contain degeneracies of two types: *twin edges*—two or more distinct edges with the same two endpoints—and *twin triangles*—two or more distinct triangles with the same three vertices.

Figure 5 illustrates these problems, in this case when the left and bottom sides of a tile of order $k = 2$ (a) get glued together (b), resulting in the mesh of figure 5(c). Note that vertices q and r were identified with with t and s, respectively; and the triangulation edges qr and rv were identified with ts and sv. As a consequence, triangles ruv and ruw became twins of suv and suw; and edges ru and rw became twins of su and sw.

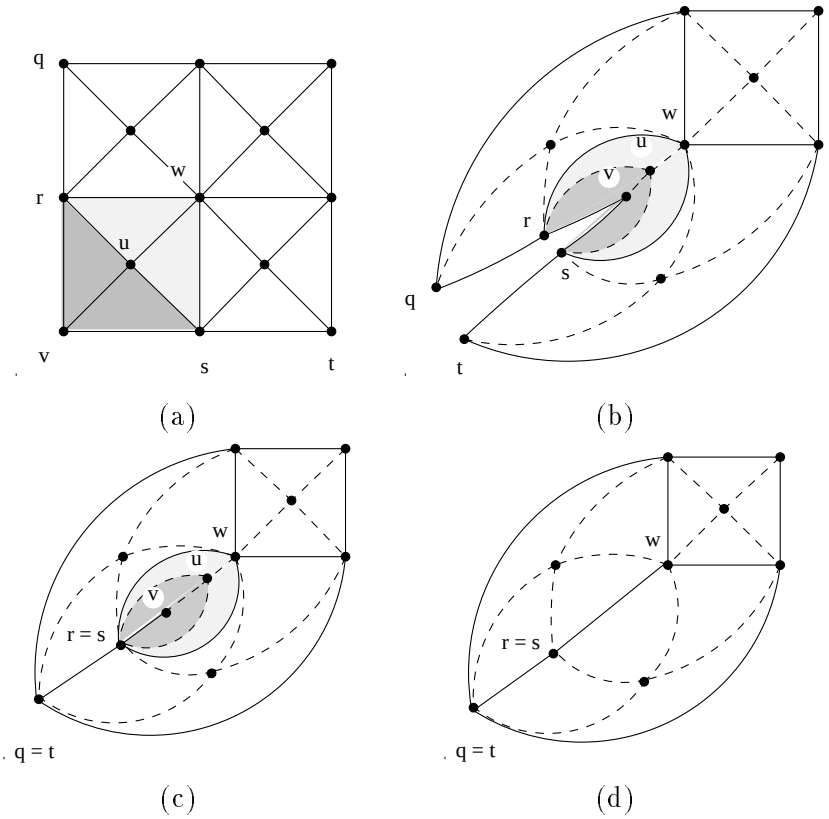


Figure 5: Gluing a tile to itself.

Recall that the mesh T is assumed to have straight edges and planar faces. Therefore, twin edges and triangles will always coincide no matter what coordinates we assign to their vertices. In particular, the twin triangles of figure 5(c) would become to a loose flap hanging out from the surface. Therefore, they must be removed, as shown in figure 5(d).

Fortunately, locating and removing such degeneracies is quite easy. Let's say that a tile cell is *critical* if it is a corner cell that is adjacent to two sides that must be glued together

(such as the shaded cell in figure 5(a)). Let's also say that a vertex of a triangulated tile is of type `C`, `S`, or `I`, according to whether it lies at the corner of the tile, along one of its sides, or in the tile's interior. See figure 6.

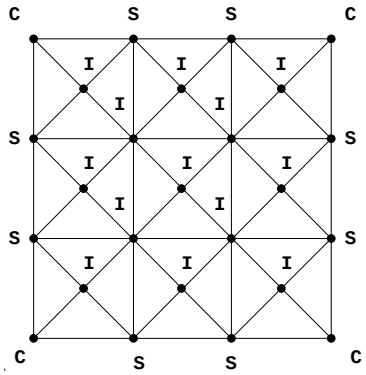


Figure 6: Vertex types in T_{u}: `C` = corner, `S` = side, `I` = interior.

We then have:

Theorem 1 *If $k \geq 2$, the triangles of T_{u} that become twins in T_{g} are precisely those contained in the critical cells of T_{u}.*

Proof: It is easy to check that each critical cell of T_{u} will always gives rise to two pairs of twin triangles in T_{g}, as shown in figure 5. We need to prove that all twin triangles are of this sort.

Let R'_{g} and R''_{g} be distinct twin triangles of T_{g}, and let $a_{\mathrm{g}}, b_{\mathrm{g}}, c_{\mathrm{g}}$ be their common vertices. Let R'_{u} and R''_{u} be the corresponding triangles in T_{u}, with vertices $a'_{\mathrm{u}}, b'_{\mathrm{u}}, c'_{\mathrm{u}}$ and $a''_{\mathrm{u}}, b''_{\mathrm{u}}, c''_{\mathrm{u}}$.

Observe that the vertices of a triangle of T_{u} may have only one of these six type combinations: `III`, `IIS`, `ICC`, `ISC`, `ISS`. The gluing process identifies only `C`–`C` or `S`–`S` vertex pairs; so R'_{u} and R''_{u} must have the same type.

Note that all triangle types include at least one `I` vertex, and that no `I` vertices get identified in the gluing step. It follows that the triangles already had at least one `I` vertex in common in T_{u}; and, therefore, they belonged to the same tile.

Now, the type of the two triangles cannot be `III`, otherwise they would not have become twins. The type cannot be `ISS`, since two triangles of that type never have an `I` vertex in common. And triangles of type `ICC` do not arise when $k \geq 2$.

So, the only possible types are `IIS` and `ISC`. In the `ISC` case, R'_{u} and R''_{u} must share the `I` vertex from the structure of the tiles, it is obvious that the `C` vertex must be shared, too. In the `IIS` case, they must have two `I` vertices shared. In either case, pairs of triangles with the required pattern of shared vertices occur only in corner cells.

To conclude the proof, observe that in both cases the unshared vertices are of type `S`. Since tile sides are glued in pairs, the only way two S vertices can get identified is when the containing sides get glued together. We conclude that the cell is a critical one.
QED.

Theorem 2 *If $k \geq 3$, the edges of T_u that become twins in T_g are precisely the* `S`–`I` *edges of critical cells.*

Proof: Clearly, all those edges will become twins in T_g. We need to show that they are the only ones to become so.

Let e'_g and e''_g be distinct twin edges of T_g, with endpoints a_g and b_g; and let e'_u and e''_u be the correspondinng edges of T_u, with endpoints a'_u, b'_u and a''_u, b''_u, respectively.

Since gluing identifies only `S`–`S` or `C`–`C` pairs, corresponding endpoints must be of the same type. If $k \geq 2$, the possible type combinations for an edge of T_u are `SS`, `SC`, `SI`, `IC`, or `II`. The combination `II` can be excluded, otherwise the two edges would not have become twins. The combinations `SS` or `SC` can be excluded, too; such edges lie on the tile boundary, and the identification of two `S` vertices implies the identification of all edges and vertices along the corresponding tile sides—which would contradict the assmption $e'_g \neq e''_g$.

Therefore, each of the edges e'_u and e''_u must have one `I` endpoint, which is shared; and one `S` or `C` endpoint, which is not shared. If $k \geq 3$, the only pairs of edges that satisfy these requirements lie in a corner cell; and, in that case, the unshared endpoints must be of type `S`. Since `S` vertices get identified only when the corresponding sides get glued together, we conclude that the cell must be a critical one.
QED.

2.7 The cleanup procedure

Thanks to theorems 1 and 2, the cleanup procedure is quite simple (provided $k \geq 3$). Whenever we glue together two adjacent sides of the same tile, we need only remove the four triangles in the corresponding critical cell, together with six of its eight edges; and then identify the two remaining edges, as shown in figure 5(d).

Since a tile has only four sides, it contains at most two critical cells, at opposite corners of the tile. If $k \geq 3$, the critical cells are pairwise disjoint, not only in T_u but also in T_g. Therefore, the critical cells can be excised from T_g independently from each other, in any order—even as the tiles are being glued.

Moreover, the cleanup procedure does not cause any additional vertices to be identified, and therefore it cannot itself create new twin edges or twin triangles.

Finally, if $k \geq 3$, at least two edges of T_u along each tile diagonal will survive the cleanup procedure. Thanks to this property, no edge of the original complex will be completely obliterated by the cleanup.

3 The energy function

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 $\mathcal{V}T = \{v_1, \ldots v_n\}$. Assuming a fixed numbering of the vertices, we define a *configuration* of the mesh as an n-tuple $x_1, x_2, \ldots x_n$ of points in $\mathbb{R}^3$. An *energy function*, which measures some kind of "badness" or "ugliness" of the surface, is therefore a function from $(\mathbb{R}^3)^n$ to $\mathbb{R}$. We have thus reduced the problem to that of finding the minimum of this function.

There are may ways in which a solution may be "ugly" or confusing, and a good energy function must be sensitive to all of them. Therefore, it is natural to consider energy formulas that combine several simpler functions, each designed to penalize a particular defect. In all our experiments, we always used some linear combination $E = \sum_i w_i E_i$ of the basic energy functions described below, with non-negative weights w_i chosen separately for each experiment.

3.1 Basic energy functions

Let $\mathcal{V}C$, $\mathcal{E}C$, $\mathcal{D}C$, and $\mathcal{F}C$ denote the vertices, primal edges, dual edges, and faces of a complex C. Let also $\vec{\mathcal{E}}C$ and $\vec{\mathcal{D}}C$ denote the set of all directed edges (every edge taken in its two directions), respectively primal and dual. The basic energy formulas that we have used in our tests were:

- the *bending energy* $E_\mathrm{b} = \sqrt{|\mathcal{F}T|} \sum_{e \in \mathcal{E}T} l_e \theta_e^2$

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

- the *excentricity energy* $E_\mathrm{x} = |\mathcal{V}T| \sum_{v \in \mathcal{V}T} |v - b_v|^2$

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

- the *repulsion energy* $E_\mathrm{r} = \dfrac{1}{|\mathcal{F}T|^2} \sum_{\substack{r,s \in \mathcal{F}T \\ r \neq s}} \dfrac{1}{|c_r - c_s|^2 + \rho_r^2 + \rho_s^2}$

  where c_r, c_s are the centroids of triangles r and s, and ρ_r, ρ_s are their average radii. (This energy can be understood as the eletrostatic potential of a set of fuzzy electric charges, located at the triangle centroids). Minimizing E_r tends to spread out the triangles in space, thus avoiding self-intersections and fold-overs.

- and the *patch area energy* $E_\mathrm{a} = \sum_{e \in \vec{\mathcal{E}}C \cup \vec{\mathcal{D}}C} \left(\dfrac{A_e}{A_0} + \dfrac{A_0}{A_e} - 2 \right)$

  where A_e is the area of the quarter-tile associated with the directed edge e of the original complex, and $A_0 = \pi / |\mathcal{E}C|$ is its "ideal" area. Minimizing E_a tends to keep the total surface area close to 4π, and equalize the tile areas, so that the edges of the original complex are spread out uniformly over the surface.

Note that none of these terms can be used by itself, since each attains its minimum only at degenerate configurations. For instance, the repulsion energy E_r is minimized when the surface spreads out to infinity; whereas the bending energy E_b is minimum when the surface has contracted to a point. 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_b, is to make the formulas largely insensitive to the number of triangles in the mesh. This property allows us to use the same weights for optimizing meshes of different resolution (different values of k). In particular, this property would allow us to use multiscale techniques—computing the energy minimum for a coarse mesh, and using the result as the starting point for optimizing a finer mesh.

3.2 Bad energy functions

We tried and discarded several other energy formulas, such as

- and the *vertex spread energy* $E_\mathrm{v} = |\mathcal{V}T|^{-1} \sum_{v \in \mathcal{V}T} |v|^2$
- the *edge stretch energy* $E_\mathrm{e} = \sum_{e \in \mathcal{E}T} (\frac{l_e}{l_0} + \frac{l_0}{l_e} - 2)$
- the *triangle area energy* $E_\mathrm{t} = \sum_{s \in \mathcal{F}T} (\frac{A_s}{A_0} + \frac{A_0}{A_s} - 2)$

where l_e is the length of edge e, A_s is the area of triangle s, $l_0 = 2\pi/\sqrt{|\mathcal{E}T|}$ is the "fair" edge length, and $A_0 = 4\pi/|\mathcal{F}T|$ is the "fair" area of each triangle.

Minimizing E_v tends to pull all vertices towards the origin, and therefore this energy term could be used to counteract the inflationary tendencies of E_r. However, this same effect also tends to crumple long complexes like that of figure 2 into compact knots.

The problem with E_e and E_t is that they are more sensitive to the shape and size of the *triangles of* T than to the shape of the surface and of the elements of C. As a result, each tile tends to assume the shape of a flat square, and the surface tends to exhibit extraneous bumps and ridges at the joints between tiles.

3.3 Selecting the weights

How do we choose the weights w_i of the various energy terms? This is still a major open question, for which we have no answer better than trial-and-error. At best, we can our intuitive understanding of the energy functions to guess the direction in which to change the weights so as to effect a given change in the surface. (See section 6.)

It would be trivial to implement a GUI-based tool that allowed direct adjustment of the weights by the user, with visual feedback. At present, our optimization algorithm is still too slow for this sort of interactive use: it would take tens of minutes to (re)compute

the optimum configuration after a small change in the weights. However, considering that still know very little about the "shape" of the energy functions, and have little expertise in numerical optimization, it is likely that the computation time can be reduced by by several orders of magnitude.

A more speculative solution is to let the computer learn the weights from examples, as Eades and Mendonça [10] did for the plane graph drawing problem. 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.

3.4 Critical weights: $E_r \times E_b$

To illustrate the kind of issues that affect weight selection, consider the following simplified situation. The triangular mesh is a single tile of order $k = 3$, all of whose vertices but one are fixed at their "natural" positions in a square, as shown in figure 7. Only the center vertex is free to move, and only in the direction perpendicular to the square.

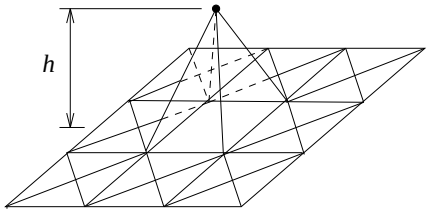


Figure 7: A testbed for energy weights.

Figure 8(a,b) shows qualitative graphs of the repulsion energy E_r and bending energy E_b of this configuration, as a function of the height h of the variable vertex above the square.

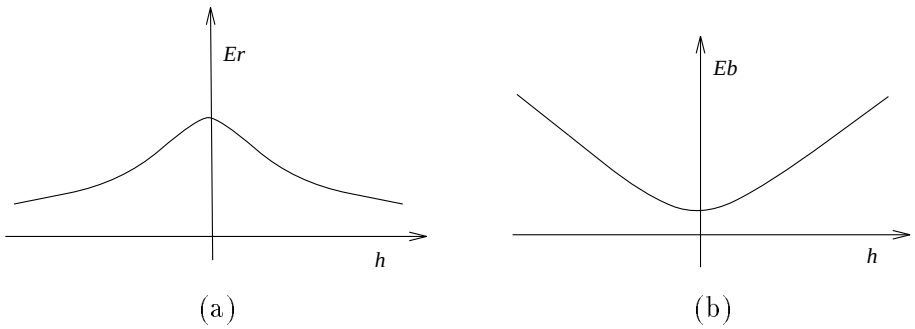


Figure 8: Graphs of (a) E_r and (b) E_b for figure 7.

The important details are that E_r has a quadratic maximum at $h = 0$, and then flattens out as h goes to infinity; whereas E_b has a quadratic minimum at $h = 0$, and grows asymptotically like $O(|h|)$.

Now consider the graph of the mixed energy $E = \alpha E_{\mathrm{r}} + (1-\alpha) + E_{\mathrm{b}}$. If α is small enough so that $\partial^2 E/\partial h^2 > 0$, the graph of E against h looks pretty much like that of E_{b}, with a single minimum at $h = 0$. However, if α is large enough to make $\partial^2 E/\partial h^2 < 0$, the graph will have a maximum at $h = 0$, and two symmetric minima at finite and non-zero values of h. See figure 9(a).

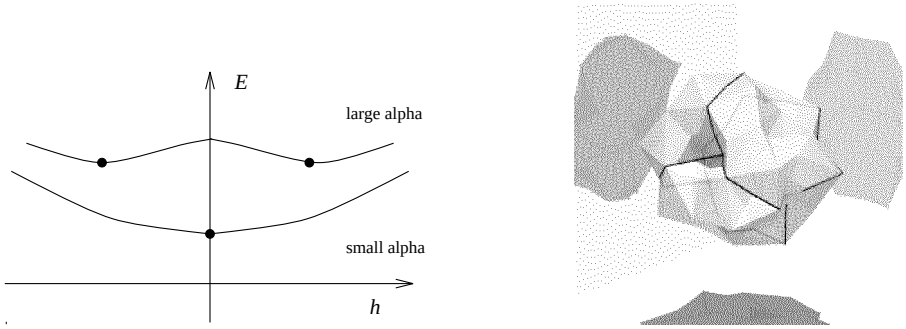


Figure 9: (a) Graph of $E = \alpha E_{\mathrm{r}} + (1-\alpha)E_{\mathrm{b}}$ for the mesh of figure 7.
(b) The "pineapple" effect.

In the more realistic case of a closed triangular mesh whose vertices are all free to move, an excess of E_{r} over E_{b} may result in minimal configurations with a "pineapple" texture, in which the the vertices lie alternately above and below a smoother "mean" surface. See figure 9(b), for example, where the energy function was $E = 9E_{\mathrm{r}} + 0.02E_{\mathrm{b}} + 5E_{\mathrm{v}} + 9E_{\mathrm{v}}$.

4 Optimization

Having constructed the triangular mesh, and chosen an energy function, we are faced with problem of finding the configuration of minimum energy.

Recall that a complex with m edges results in a mesh with about $4mk^2$ triangles. Recall also that k must be at least 3 to ensure that the original edges survive the removal of twin triangles. It follows that even small complexes lead to meshes with hundreds of vertices, and thus to energy functions with hundreds of variables. Moreover, the functions are highly non-linear and convoluted, with many local minima.

Obviously there is little hope of finding the *global* minimum of such a function. Even finding a *local* minimum is a non-trivial task. Fortunately, a true minimum is not necessary for visualization purpopses; all we need is a configuration whose enery is low enough. Moreover, since the program cannot tell when the energy is "low enough", 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 various general purpose numerical optimization techniques, especially:

- Kirkpatrick's *simulated annealing* [22, 29];
- Nelder and Mead's *downhill simplex method* [29, p.289];
- Powell and Brent's *principal directions* method [29, p.298];
- the naive *single coordinate* optimization, with periodic diagonal steps [29, p.294];
- a *gradient descent* method with adaptive stepsize.

Our tests showed that the effectiveness of these methods generally increases from top to bottom. It is a unfortunate that we also implemented the methods roughly in that order; so that the one we found most effective—gradient descent—is also the one we have less experience with.

The tests were performed on various triangulated meshes, with up 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 total CPU time. The ranking of the methods was generally the same on all tests, and consistent over time.

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 still not fast enough to be usable: even after several hours of CPU time, the best configurations found were still far from smooth. Moreover, those two methods require $\Omega(n^2)$ storage for a function of n variables, and therefore are limited to complexes with a few tens of edges.

The naive minimization method consists of optimizing one variable at a time (using, for instance, Brent's univariate minimization algorithm [29, p.283]), while all other variables are held fixed. This process is applied to each coordinate in turn, in cyclic fashion. The minimum configuration thus evolves by a sequence of steps parallel to the coordinate axes. A simple but very effective improvement is to take a "diagonal" step every $n+1$ such "axial" steps, along the line connecting the outcomes of the first and the last of these 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 main reason, which was obvious on hindsight, is that 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.

The general gradient descent method keeps improving the best configuration p found so far along the path defined by the differential equation $dp/dt = -\nabla E(p)$. We avoided

this method for a long time, since we wrongly assumed that computing the gradient of our energy functions (hundreds of partial derivatives of a formula with hundreds of operations) would be too difficult and expensive.

Only much later did we realize that, since our functions are sums of simple terms, the gradient too has the same structure, and thus is relatively easy to compute. Indeed, as Baur and Strassen [4] have proved, 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.

We eventually implemented this method too, using a simple adaptive Euler integrator to follow the steepest descent path. This turned out to be the most effective of all general methods, producing fairly smooth surfaces from scratch in tens of minutes instead of tens of hours. Still, we believe there is much room for improvement, for instance in the step-size adjustment logic. In particular, there are several gradient-based algorithms that are theoretically better than gradient-descent, such as Fletcher-Reeves and Polak-Ribière [29, p.301], which we haven't had the time to try.

4.2 Heuristic methods

The general-purpose methods above can be applied to a large class of functions, and in principle will converge to a true local optimum if allowed to run for a sufficiently long time. However, exactness is relevant only towards the end of the optimization. When starting from a random configuration, the first task of any optimization algorithm is to untangle the surface and smooth out its largest wrinkles, which are bad under any reasonable energy function. This initial stage can be carried out by an *ad hoc* (heuristic) method that does not depend on the particular energy function being minimized.

We have successfully used two such 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 in the star of v (those edges which are incident to v, or connect two neighbors of v). To simplify the computations, the new position of v is restricted to the straight line $a + t\vec{n}$, where a is the barycenter of the neighbors of v, $\vec{n}$ is the normal to the surface at v (also computed from those neighbors), and t is a free parameter.

The *spreading heuristic* keeps the vertex v fixed, but rotates its neighbors $u_1, \ldots u_k$ around the axis $a + t\vec{n}$ so as to better equalize the angles $u_i v u_{i+1}$, when projected on a plane perpendicular to $\vec{n}$.

Applying one pass of either heuristic to all vertices has the effect of distributing the corresponding "stress" (exterior dihedral angles and projected edge angles) more uniformly throughout the whole mesh, and allowing neighboring defects of opposite direction to cancel each other out.

In our experience, these heuristics work very well, at least for triangular meshes with a few hundred vertices. Usually, the surfaces obtained after 20-50 passes were already smooth enough for topology visualization purposes.

However, the smoothing action of these heuristics is a diffusion-like process, in the sense that the rate at wich stress "flows" over the surface decreases as its distribution becomes more uniform. As in any diffusion process, the number of passes needed to achieve a given level of surface smoothness should scale as the square of the graph-theoretic diameter of the triangulation.

Therefore, after a hundred or so passes, the heuristics generally become ineffective, even though the configuration may still be visibly non-optimal. To advance beyond this point, we must switch to more intelligent methods, such as gradient descent. Moreover, the heuristics are not sensitive to certain defects, such as self-intersections or unequal tile areas; and, obviously, they pay no attention to the weights w_i. To take these factors into account, we must apply an "exact" method, using the heuristically smoothed configuration merely as a good starting point.

4.3 Local minima and global minima

Generally speaking, the methods above will only find a *local* minimum of the energy function, near the starting configuration. Of course, what we would like to get is the *global* minimum.

Unfortunately our energy functions are full of local minima, separated by high energy barriers. Typically, these local minima correspond to configurations with unnecessary self-intersections. As a rule, removing such self-intersections would require going through configurations of high curvature, and therefore high energy.

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 it to work fast enough. At present, our best strategy is simply to repeat the search several times, starting from different random configurations, and select the best result by visual inspection.

This simple strategy may seem hopeless, considering the large number of "bad" local minima that we expect our functions to have. However, in our tests is has worked suprisingly well; five to ten trials were enough to find the global optimum, even for meshes with hundreds of vertices (see section 5). A possible explanation, which seems geometrically plausible, is that the "attraction basin" of the global optimum—the set of starting points from which the optimizer coverges to that configuration–is likely to be much wider than the basins of other local minima.

5 Results

In this section we show some geometric realizations of topological complexes that we produced with our tools. The output of our tool is a text file that describes a computer graphics model of the final configuration: a set of translucent triangles in $\mathbb{R}^3$, plus some thin cylinders and small spheres, painted black, that trace out the edges and vertices of the original complex. These files were then rendered with `POVRAY`, a public domain ray tracer [32]. Most of these images were rendered with Gouraud shading [14] to smooth out the edges of the triangulation.

5.1 Torus

Figure 10 shows several stages in the optimization of the simplest complex with toroidal topology, illustrated in figure 1(a). The original complex has one vertex, one square face, and two edges; when modeled with tiles of order $k = 5$, it becomes a triangular mesh with 200 triangles, 300 edges, and 100 vertices.

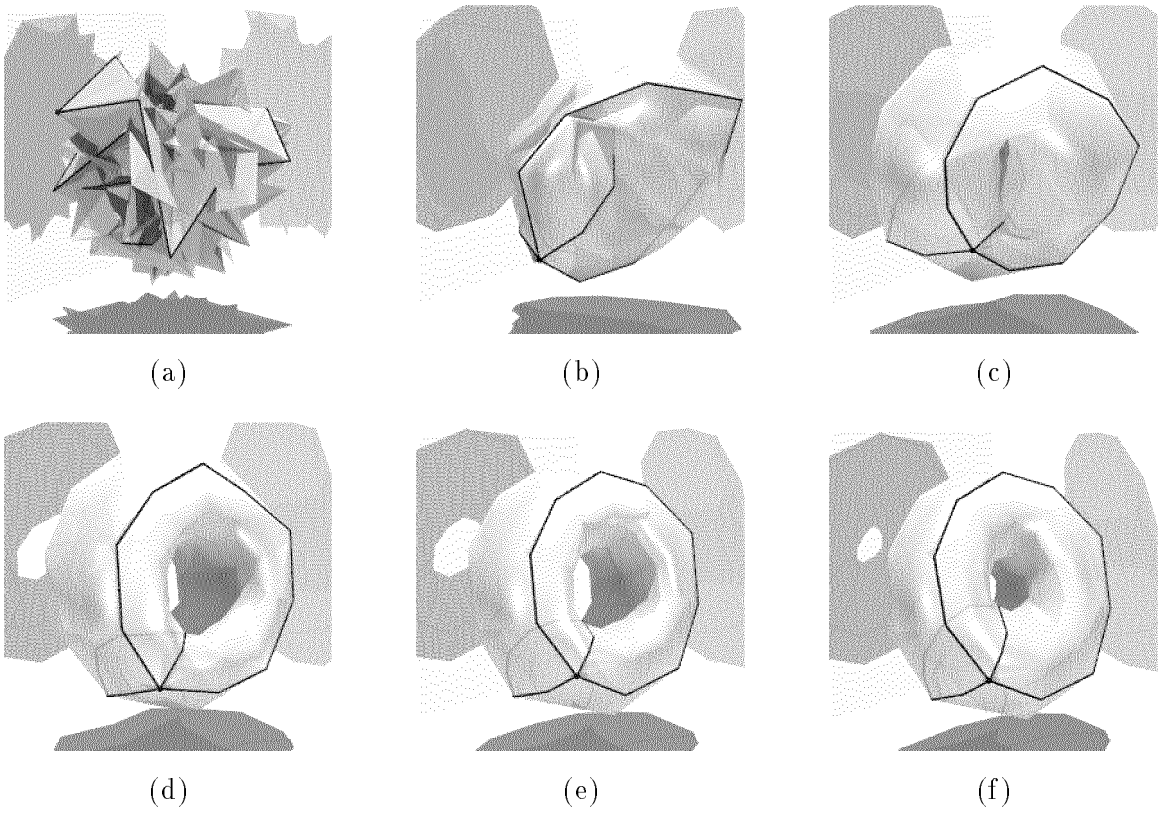


Figure 10: Heuristic smoothing (b–d) and energy optimization (e–f) of a simple torus complex.

Figure 10(a) shows the initial random configuration. Figures 10(b–d) show the results of 5, 30, and 100 passes of the smoothing heuristic, interleaved with that many passes of the spreading heuristic, applied to configuration (a). Figures 10(e,f) show the results of optimizing configuration (d) with the `Grad` method, respectively after 100 and 1000 energy evaluations. The energy function used was $E_{\mathrm{b}} + E_{\mathrm{a}} + E_{\mathrm{x}} + 5E_{\mathrm{r}}$.

We performed this test five times, starting from different random configurations. Of the five final configurations, two had the "correct" shape, like te one shown above; the other three were still self-intersecting in various ways.

5.2 Sausage

The next example is based on a "sausage" complex similar to that of figure 2(a), with only two "stages" instead of four. The triangulated model had 360 faces, 540 edges, and 182 vertices. Figure 11 shows the state after 0, 15, and 100 passes of the smoothing and spreading heuristics (a–c); and then after 2000 steps of the `Grad` method (d), with energy function $E_\mathrm{b} + E_\mathrm{a} + E_\mathrm{x}$.

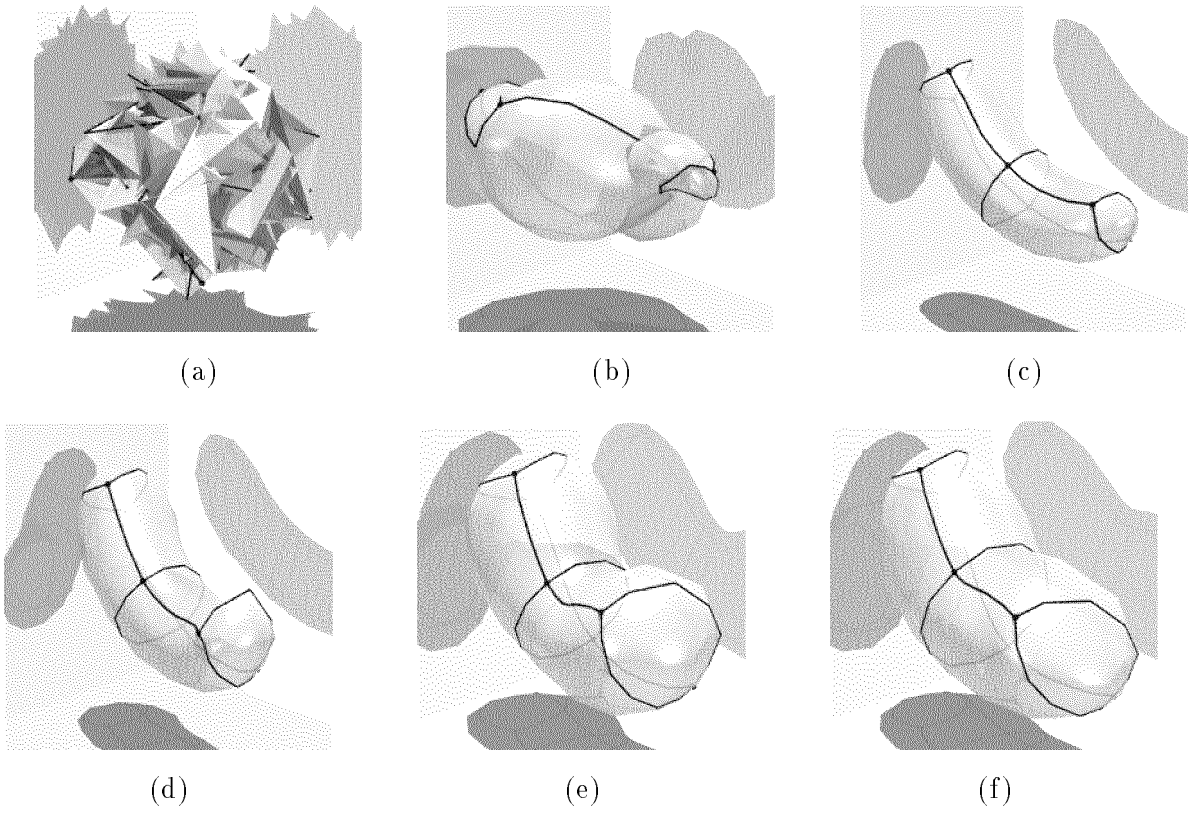


Figure 11: Visualization of a sausage-like complex.

Apparently, the heuristics and the energy optimization were both terminated before they had time to converge to their respective optima (which, almost certainly, should be "straight" sausages). Even so, it is already evident that the optimum shapes for the two methods are quite different.

5.3 Klein bottle

Figure 12 refer to the "Klein bottle" complex of figure 1(c). Modeling this complex with 5×5 tiles resulted in a triangulation with 200 triangles, 300 edges, and 100 vertices.

The resulting shapes are shown in figure 12. To each of 10 independent random configurations, we applied 100 passes of each heuristic (a,c), followed 1000 steps of the `Grad`

optimizer (b,d), with energy $E_{\rm b} + E_{\rm a} + E_{\rm x} + 5E_{\rm r}$. Seven of these trials yielded the "crossed-tube" solution (a,b); the other three yielded the "crossed hole" solution (c,d).

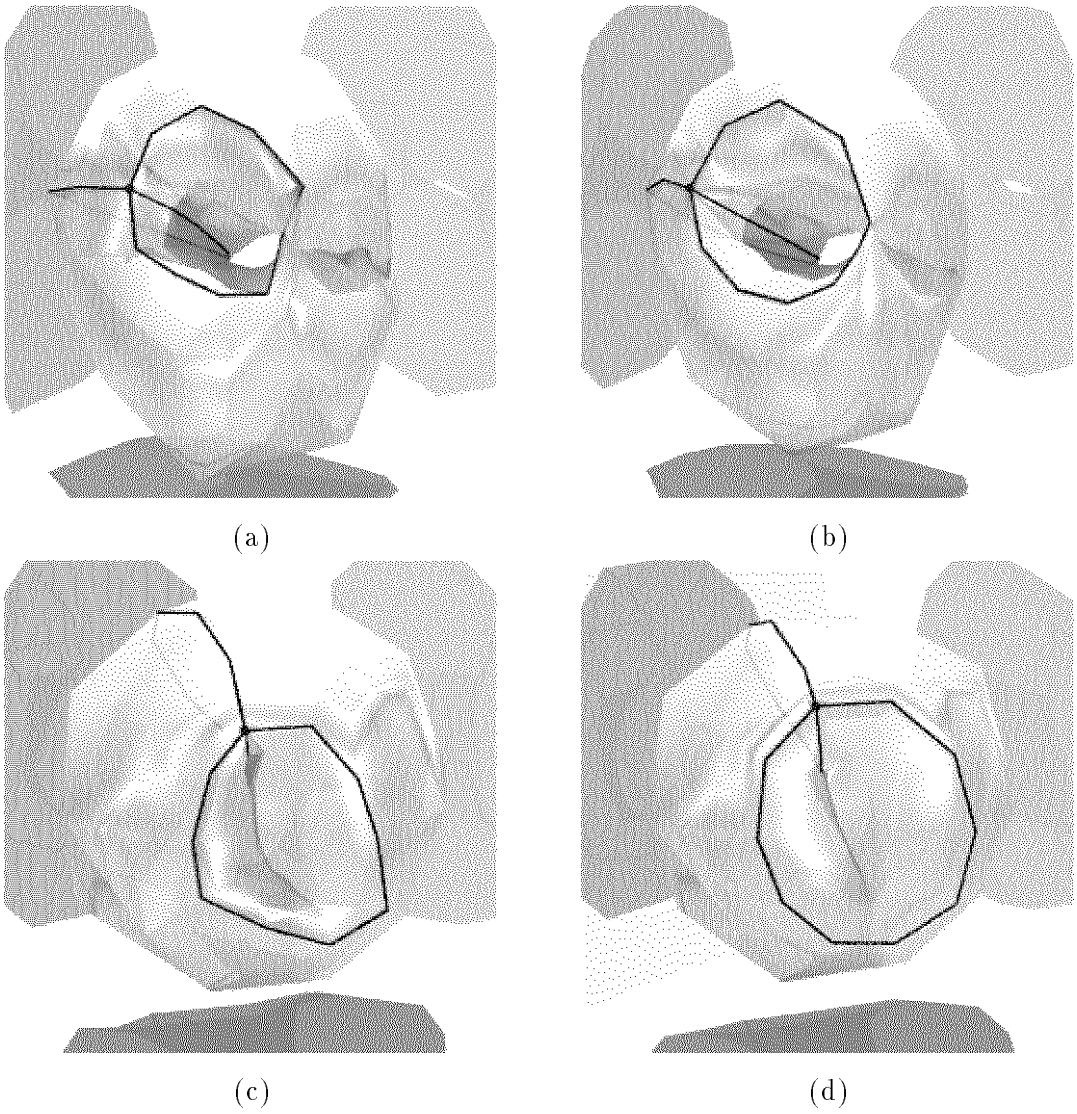


Figure 12: Klein bottle representations: "crossed tube" (a,b) and "crossed hole" (c,d).

Figure 12(b) can be understood as a torus that has been cut open and then reglued in a "crossed" fashion, as shown in figure 13(a). Note that the resulting surface is self-interesecting and has two "pinch points" ($A = A'$ and $B = B'$) where the curvature is infinite. Figure 12(d) is similar, except that the cut goes around the hole of the torus, rather than through it.

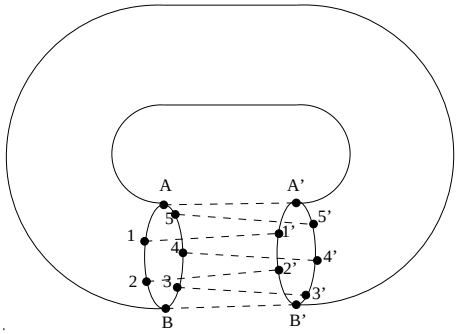


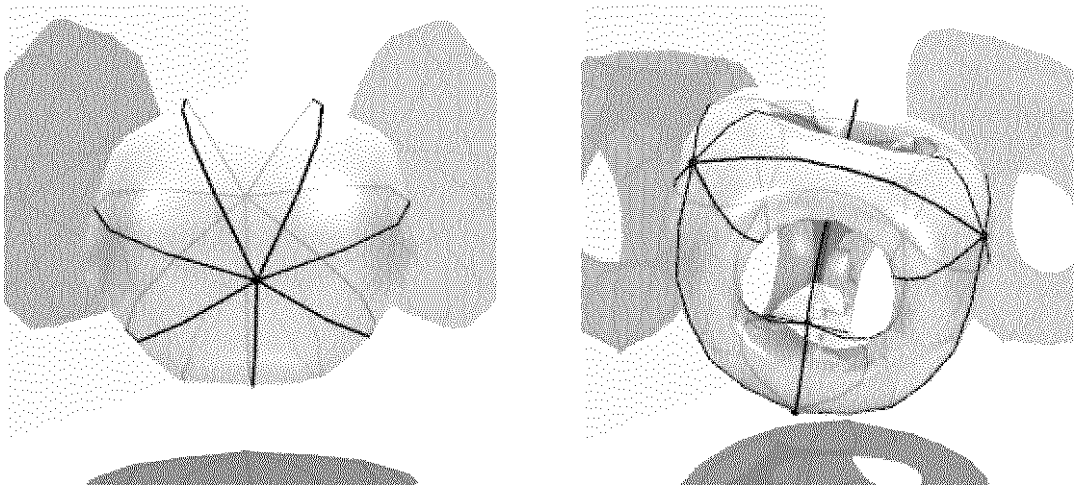
Figure 13: Understanding the "crossed-tube" solution.

Of course, since the Klein surface is non-orientable, it is not possible to realize it in $\mathbb{R}^3$ without self-intersections. Still, we could hope to obtain at least a "smooth" surface, whose curvature is finite everywhere—as in the "classical" Klein bottle, shown in figure 1(d).

Unfortunately, our tools are still not smart enough to do that. . The energy optimization phase is generally incapable of transforming a configuration with pinch points (such as figures 12(b,d)) into one without them (such as figure 1)(d). Even if the pinch-free solution has lower energy (which may not be true, depending on the energy function), the two solutions are probably separated by higher-energy barriers.

5.4 Orange and tritorus

Figure 14 shows two additional realizations produced by our tool.



(a) (b)

Figure 14: Two additional examples: orange and tritorus.

The "orange-like" complex on the left is a sphere sliced into seven sectors (wedges) by seven meridians. Its modeling with 3×3 tiles resulted in 252 triangles, 378 edges, e 128 vertices. The configuration shown was obtained from a random one after 100 passes of the smoothing and spreading heuristics, without energy optimization.

The complex on the right has 12 square faces, 24 edges, and 8 vertices, glued so as to make a tritorus (a sphere with three handles). More precisely, the faces are glued in pairs to form six open-ended cylinders, and these are glued toghether so that their axes are the edges of a tetrahedron. Its model with 3×3 tiles had 864 triangles, 1296 edges, and 428 vertices. We generated 10 random configurations of this model, and we subjected each of them to 100 passes of the smoothing and spreading heuristics. The resulting shapes were all reasonably smooth, but only one of them was free from self-intersections. We manually selected that configuration and subjected it to a couple thousand steps of the `Grad` optimizer, with the result shown above.

5.4.1 Stars...?

The examples that follow record an unplanned test of our motivating hypothesis, namely that an automatic topology visualizer can be a valuable tool for debugging solid modeling algorithms.

One of our test cases was the complex with 15 faces, 25 edges, and 12 vertices shown in figure 15. Informally, this complex can be described as five "half-sausages" (see section 5.2) joined so as to make a five-pointed star.

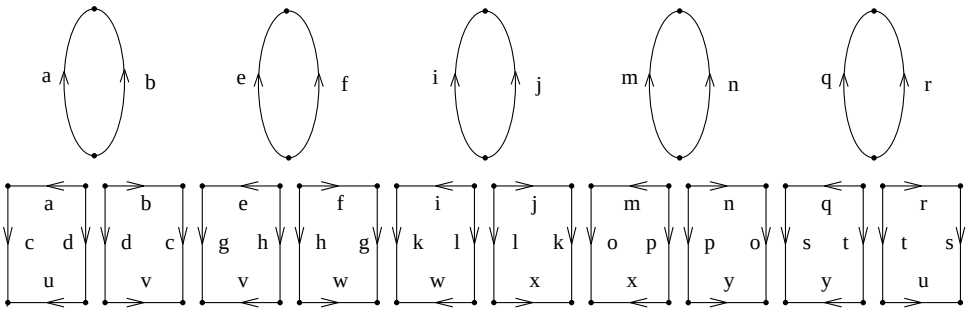


Figure 15: A star-like complex.

Modeling this complex with 3×3 tiles produced a triangulation with 900 triangles, 1350 edges, and 452 vertices. Starting from a random configuration, after 100 passes of the spreading heuristic and 300 passes of the smoothing heuristic we got the configuration shown in figure 16:

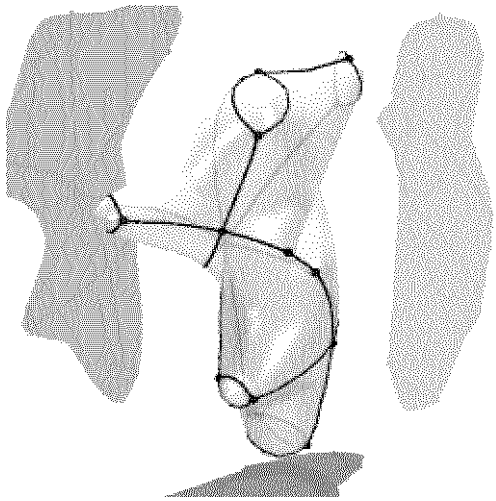


Figure 16: The star complex, first try...

Looking at this image, we immediately noticed that the complex had the wrong topology. (Note, for example, the vertices of degree 2 near the center.)

Eventually we found a mistake in the program that created the input complex. We fixed this mistake and repeated the test, with the result shown in figure 17.

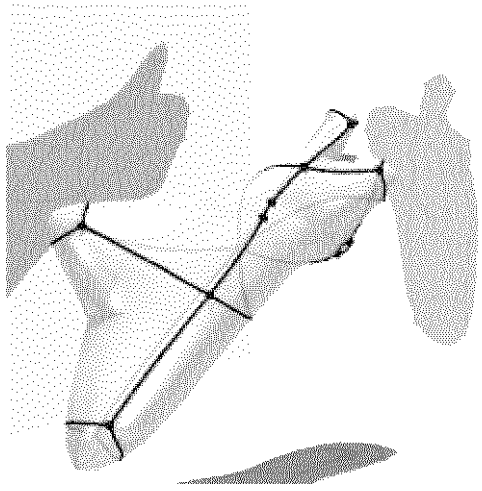


Figure 17: The star complex, second try....

From this image we could see that the complex still had the wrong topology. Indeed, we soon found a second mistake in our code.

Having fixed this second mistake, we got the picture shown in figure 18 (after 100 spreading passes and 900 smoothing passes):

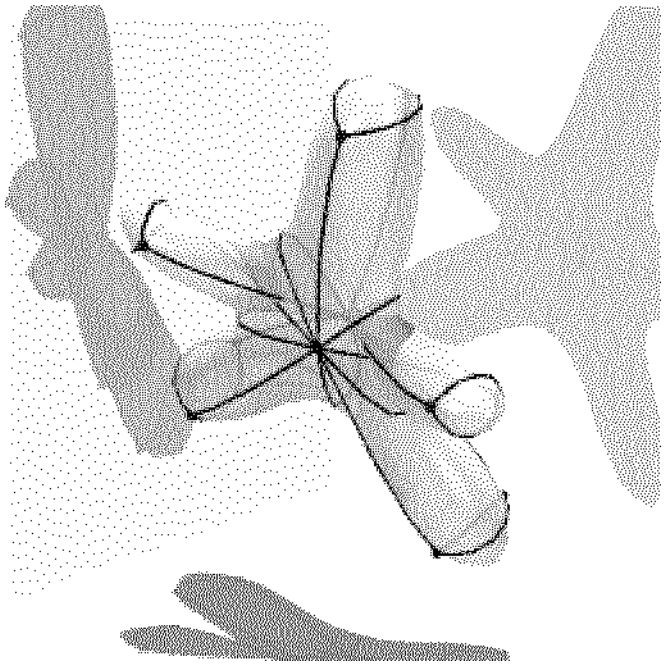


Figure 18: The star complex—at last...

This configuration is still not energy-optimal: the five rays of the star have unequal sizes, and two of them (the "arms") are turned inside-out with respect to the other three. Even so, it already allows us to visually check that all but a few of the incidences of the input complex are correct.

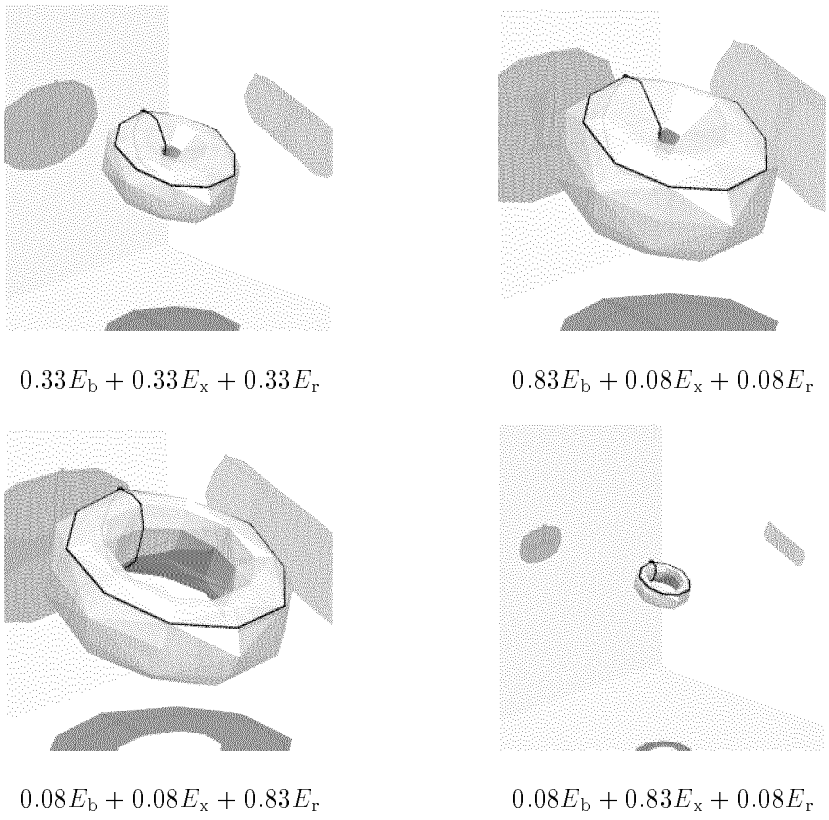
For the record, here is the `Modula-3` code that we used to build this complex. The main procedure is `MakeStar`, which builds a star with `n` arms, each having `s` bands; thus the complex of figure 15 would be the result of `MakeStar`$(5, 1)$. The auxiliary procedure `MakeOrange` builds a complex like that of figure 14, left, with n slices. The procedure `BuildTower` is then called repeatedly, in order to erect on each face of this "orange" a tower of h storeys, with 2 quadrilaterals each, capped by a 2-sided face. An `Arc` is a directed and oriented edge. The procedures `Onext`, `Oprev`, `Lnext`, `Sym`, `Rot`, `Tor` (= `Rot`$^{-1}$), `Splice`, and `MakeEdge` are standard quad-edge operators; see the Guibas and Stolfi paper [15] for their definition.

```
PROCEDURE MakeStar (n, h: CARDINAL): Arc =
  VAR o: Arc;
  BEGIN
    o := MakeOrange(n);
    FOR k := 1 TO n DO
      o := Onext(o);
      BuildTower(2, h, o);
    END;
    RETURN o
  END MakeStar;

PROCEDURE MakeOrange(n: CARDINAL): Arc =
  VAR fst, a, b: Arc;
  BEGIN
    a := Rot(MakeEdge());
    Splice(a, Sym(a));
    fst := a;
    FOR i := 2 TO n DO
      b := Rot(MakeEdge());
      Splice(b, Sym(b)); Splice(b, Sym(a));
      a := b;
    END;
    Splice(fst, Sym(a));
    RETURN Tor(fst)
  END MakeOrange;

PROCEDURE BuildTower(h: CARDINAL; a: Arc) =
  VAR s, e, t: Arc;
  BEGIN
    t := a;
    FOR i := 1 TO h DO
      t := Oprev(t);
      s := Rot(MakeOrange(2)); (* A ring of 2 edges *)
      FOR j  := 1 TO 2 DO
        e := MakeEdge();
        Splice(t, e); Splice(Sym(e), Oprev(s));
        t := Lnext(t);
        s := Lnext(s)     (* was "s := Rnext(s)" (error 1) *)
      END;
      t := Onext(s);      (* was "t := s" (error 2) *)
    END;
  END BuildTower;
```

Figure 19: The star-building code.

6 Varying the weights

The images in this section illustrate the variety of effects that can be achieved merely by adjusting the weights in a mixed energy formula. In all these examples, we used the torus complex of section 5.1, modeled with 3×3 tiles. An initial random configuration was subjected to 50 passes of the smoothing heuristic interleaved with 20 passes of the spreading heuristic, and then optimized with 5000 steps of the `Grad` method. The energy combination used is listed under each image.



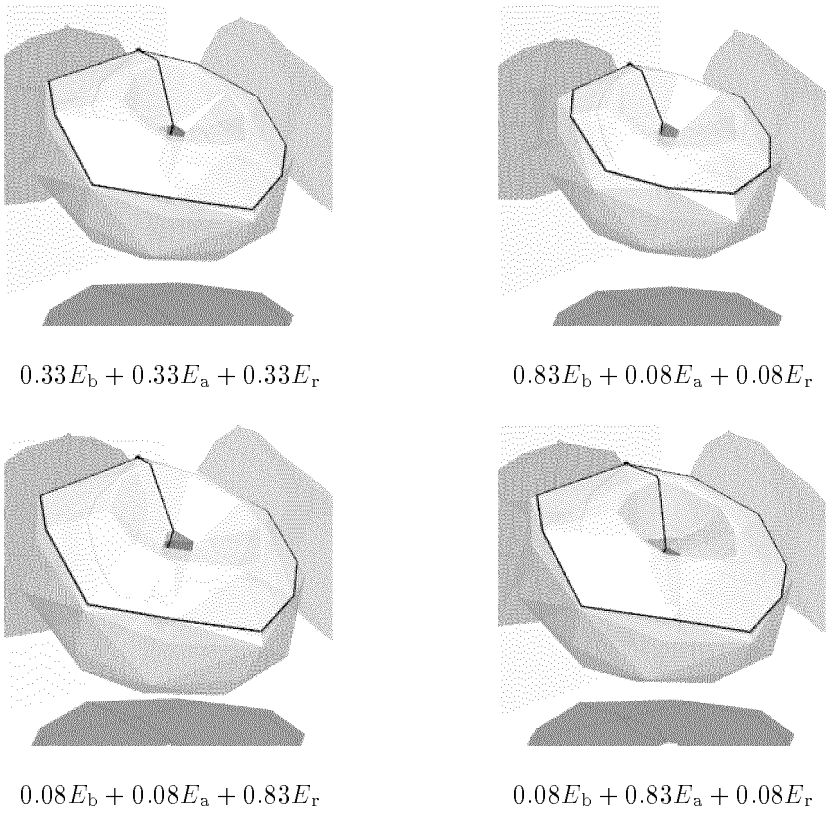
$0.33E_\mathrm{b} + 0.33E_\mathrm{x} + 0.33E_\mathrm{r}$ $0.83E_\mathrm{b} + 0.08E_\mathrm{x} + 0.08E_\mathrm{r}$

$0.08E_\mathrm{b} + 0.08E_\mathrm{x} + 0.83E_\mathrm{r}$ $0.08E_\mathrm{b} + 0.83E_\mathrm{x} + 0.08E_\mathrm{r}$

Figure 20: Combinations of E_b, E_x, and E_r

Notice how increasing the weight of E_x causes the surface to shrink, while increasing E_r causes it to expand.

Notice also how the hole of the "donut" tends to collapse when the weight of E_b is increased at the expense of E_r. Recall that E_b is the sum of the exterior dihedral angles, squared, weighted by the respective edge lengths. One way to reduce E_b is to compress the hole to a narrow cylinder, as shown in the top right image. That way, a good portion the surface's curvature gets confined to the edges around the hole's rim, which are very short.

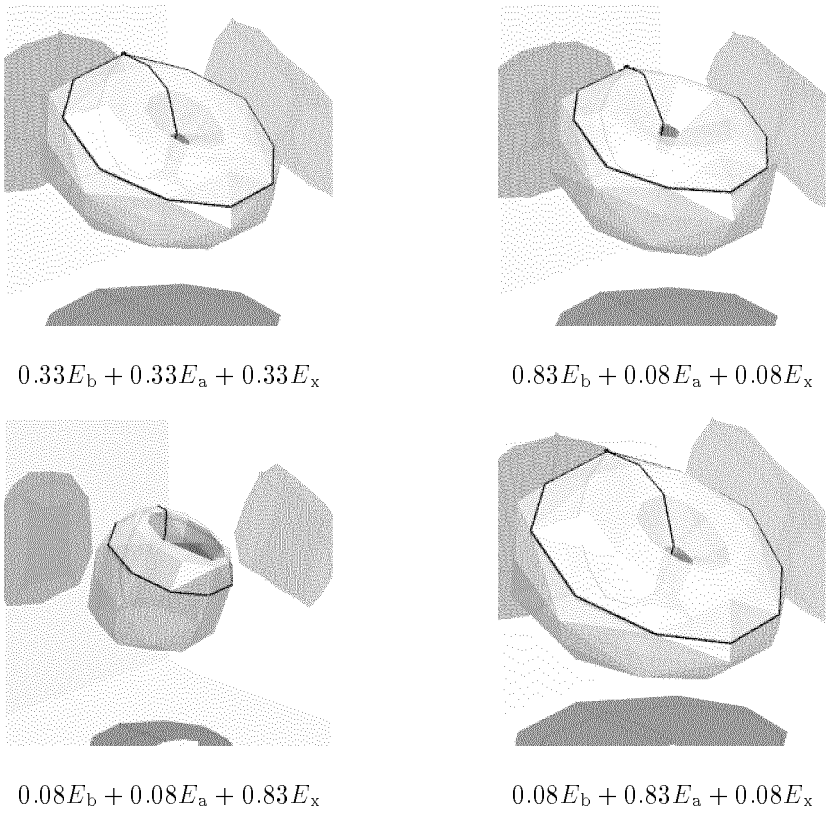


$0.33E_\mathrm{b} + 0.33E_\mathrm{a} + 0.33E_\mathrm{r}$ $0.83E_\mathrm{b} + 0.08E_\mathrm{a} + 0.08E_\mathrm{r}$

$0.08E_\mathrm{b} + 0.08E_\mathrm{a} + 0.83E_\mathrm{r}$ $0.08E_\mathrm{b} + 0.83E_\mathrm{a} + 0.08E_\mathrm{r}$

Figure 21: Combinations of E_b, E_a, and E_r.

In figure 21, E_x has been replaced by E_a, which is sensitive to the area of the surface but not to its shape. The hole-shrinking tendency of E_b is stronger than before, and we must use a lot of E_r just to keep the hole open.

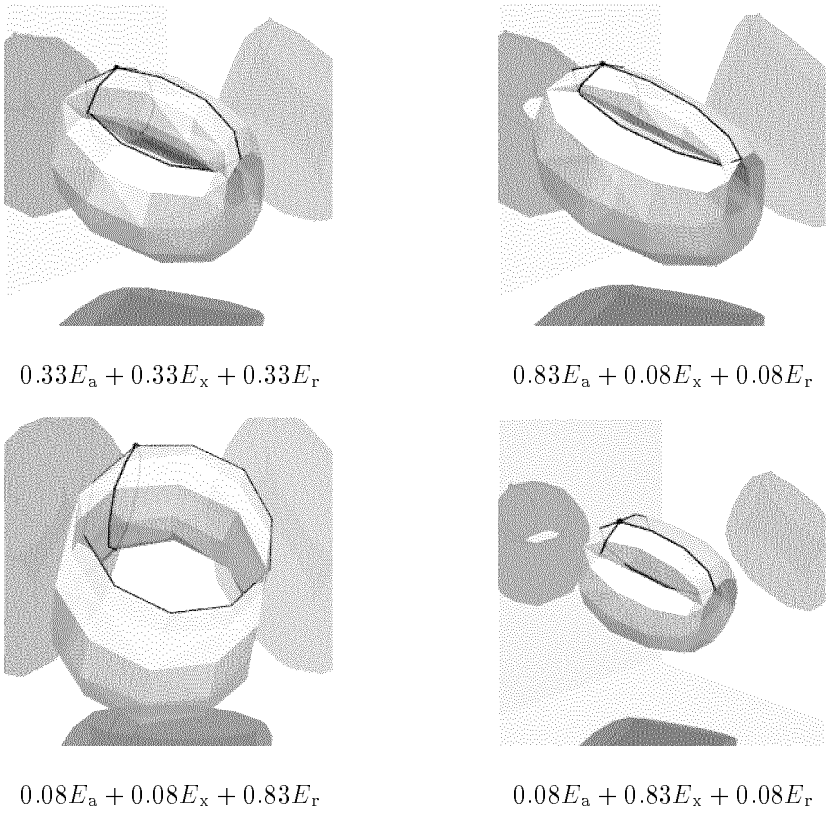


$0.33E_{\mathrm{b}} + 0.33E_{\mathrm{a}} + 0.33E_{\mathrm{x}}$ $0.83E_{\mathrm{b}} + 0.08E_{\mathrm{a}} + 0.08E_{\mathrm{x}}$

$0.08E_{\mathrm{b}} + 0.08E_{\mathrm{a}} + 0.83E_{\mathrm{x}}$ $0.08E_{\mathrm{b}} + 0.83E_{\mathrm{a}} + 0.08E_{\mathrm{x}}$

Figure 22: Combinations of E_{b}, $E_{\mathbf{a}}$, and $E_{\mathbf{x}}$.

In figure 22, where E_{r} has been omitted, it is the joint action of $E_{\mathbf{a}}$ and $E_{\mathbf{x}}$ that keeps the hole from collapsing under the influence of E_{b}.



$0.33E_\mathrm{a} + 0.33E_\mathrm{x} + 0.33E_\mathrm{r}$ $0.83E_\mathrm{a} + 0.08E_\mathrm{x} + 0.08E_\mathrm{r}$

$0.08E_\mathrm{a} + 0.08E_\mathrm{x} + 0.83E_\mathrm{r}$ $0.08E_\mathrm{a} + 0.83E_\mathrm{x} + 0.08E_\mathrm{r}$

Figure 23: Combinations of E_a, E_x, and E_r.

In all four combinations of E_a, E_x, and E_r that we tried (figure 23), the final surface had two self-intersections (and four pinch points), so that one half of the torus was inside-out.

7 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; speedups by several orders of magnitude seem possible, just by using better optimization algorithms. We need methods that can overcome the barriers between local minima; either combinatorial optimization tools, such as simulated annealing and genetic algorithms, or (more likely) multiscale techniques.

We still undertand very little about the effects of the various energy functions, and the proper weights to use. Both experiment and intuition suggest that we redefine our energy functions to penalize extreme cases more strongly. For example, the bending energy E_b should probably be modified go to infinity when the dihedral angle θ_e tends to 180°.

We also need to develop energy functions that penalize self-intersections more strongly than the functions we have got. One possibility is to use the total length of the self-intersection curves, perhaps weighted by some function of the dihedral angle. On the same line, we need to improve the heuristic methods so that they are more effective at "uncrossing" and untangling the triangulation. Presently, both heuristics visit the vertices of the mesh in random order; perhaps they would be more effective if applied in some neighbor-to-neighbor order (breadth-first or depth-first).

References

[1] A. V. Arkhangel'skiĭ and L. S. Pontryagin. *General Topology I*, volume 17 of *Encyclopaedia of Mathematical Sciences*. Springer-Verlag, 1990.

[2] Chandrajit L. Bajaj. Smoothing polyhedra using implicit algebraic splines. In *SIGGRAPH'92 Conference Proceedings*, volume 26, pages 79–88, 1992. In Computer Graphics 26 (2).

[3] Bruce G. Baumgart. A polyhedron representation for computer vision. In *Proceedings of 1975 AFIPS National Computer Conference*, volume 44, pages 589–596, 1975.

[4] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983.

[5] Kenneth A. Brakke. The Surface Evolver. *Experimental Mathematics*, 1(2):141–165, 1992.

[6] Kenneth A. Brakke. *Surface Evolver Manual*. The Geometry Center, The Geometry Center, Minneapolis, MN, December 1993.

[7] Erik Brisson. Representing geometric structures in d dimensions: Topology and order. *Discrete Comput. Geom.*, 9:387–426, 1993.

[8] George Celniker and Dave Grossard. Deformable curve and surface finite-elements for free-form shape design. In *SIGGRAPH '91 Conference Proceedings*, volume 26, pages 257–266, 1992. In Computer Graphics 25 (4).

[9] Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. Technical report, The Weizmann Institute of Science, Department of Applied Mathematics and Computer Science, 1989.

[10] Cândido X. F. de Mendonça Neto and Peter Eades. Learning aesthetics for visualization. In *Anais do XX Seminário Integrado de Software e Hardware*, pages 76–88, Florianópolis, SC (Brazil), 1993.

[11] G. Di Battista, P. D. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry: Theory and Applications*, 4:235–282, 1994.

[12] David P. Dobkin and Michel J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica*, 4:3–32, 1989.

[13] H. Ferguson, A. Rockwood, and J. Cox. Topological design of sculpted surfaces. In *SIGGRAPH '92 Conference Proceedings*, pages 149–156, 1992. In Computer Graphics 26 (2).

[14] James D. Foley, Andries Van Dam, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison Wesley, 2 edition, 1990.

[15] 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.

[16] D. Hilbert and S. Cohn-Vossen. *Geometry and the Imagination*. Chelsea, New York, 1952.

[17] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, August 1993.

[18] B. K. P. Horn. The curve of least energy. *ACM Transactions on Mathematical Software*, 9(4):441–460, December 1983.

[19] 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.

[20] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, pages 7–15, April 1989.

[21] G. Kant. A more compact visibility representation. In *Proc. of Graph Drawing '93 — ALCOM Workshop on Graph Drawing*, 1993.

[22] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[23] Pascal Lienhardt. Subdivisions of n-dimensional spaces and n-dimensional generalized maps. *Proc. 5^{th} Annual ACM Symp. on Computational Geometry*, pages 228–236, June 1989.

[24] D. Lischinski. Incremental Delaunay triangulation. In P. S. Heckbert, editor, *Graphics Gems IV*. Academic Press, 1994.

[25] Charles Loop and Tony DeRose. Generalized B-spline surfaces of arbitrary topology. In *SIGGRAPH '90 Conference Proceedings*, pages 347–356, August 1990. In Computer Graphics 24 (4).

[26] H. P. Moreton. Minimum curvature variation curves, networks, and surfaces for fair free-form shape design. Technical report, Computer Science Division, University of California, Berkeley, March 1993.

[27] Henry P. Moreton and Carlo H. Séquin. Functional optimization for fair surface design. In *SIGGRAPH '92 Conference Proceedings*, pages 167–176, 1992. In Computer Graphics 26 (2).

[28] D. B. Parkinson and D. N. Moreton. Optimal biarc-curve fitting. *Computer-Aided Design*, 23(6):411–419, 1991.

[29] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.

[30] William Welch and Andrew Witkin. Variational surface modeling. In *SIGGRAPH'92 Conference Proceedings*, volume 26, pages 157–166, 1992. In Computer Graphics 26 (2).

[31] Fujio Yamaguchi and Toshiya Tokieda. A solid modelling system: Freedom-II. *Computers & Graphics*, pages 225–232, 1983.

[32] C. Young, D. K. Buck, and A. C. Collins. POV-Ray version 2.0. US Mail: 3119 Cossel Drive - Indianapolis, IN 46224 U.S.A., 1993.

Relatórios Técnicos – 1995

95-01 **Paradigmas de algoritmos na solução de problemas de busca multidimensional,** *Pedro J. de Rezende, Renato Fileto*

95-02 **Adaptive enumeration of implicit surfaces with affine arithmetic,** *Luiz Henrique de Figueiredo, Jorge Stolfi*

95-03 **W3 no Ensino de Graduação?,** *Hans Liesenberg*

95-04 **A greedy method for edge-colouring odd maximum degree doubly chordal graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

95-05 **Protocols for Maintaining Consistency of Replicated Data,** *Ricardo Anido, N. C. Mendonça*

95-06 **Guaranteeing Full Fault Coverage for UIO-Based Methods,** *Ricardo Anido and Ana Cavalli*

95-07 **Xchart-Based Complex Dialogue Development,** *Fábio Nogueira de Lucena, Hans K.E. Liesenberg*

95-08 **A Direct Manipulation User Interface for Querying Geographic Databases,** *Juliano Lopes de Oliveira, Claudia Bauzer Medeiros*

95-09 **Bases for the Matching Lattice of Matching Covered Graphs,** *Cláudio L. Lucchesi, Marcelo H. Carvalho*

95-10 **A Highly Reconfigurable Neighborhood Image Processor based on Functional Programming,** *Neucimar J. Leite, Marcelo A. de Barros*

95-11 **Processador de Vizinhança para Filtragem Morfológica,** *Ilka Marinho Barros, Roberto de Alencar Lotufo, Neucimar Jerônimo Leite*

95-12 **Modelos Computacionais para Processamento Digital de Imagens em Arquiteturas Paralelas,** *Neucimar Jerônimo Leite*

95-13 **Modelos de Computação Paralela e Projeto de Algoritmos,** *Ronaldo Parente de Menezes e João Carlos Setubal*

95-14 **Vertex Splitting and Tension-Free Layout,** *P. Eades, C. F. X. de Mendonça N.*

95-15 **NP-Hardness Results for Tension-Free Layout,** *C. F. X. de Mendonça N., P. Eades, C. L. Lucchesi, J. Meidanis*

95-16 **Agentes Replicantes e Algoritmos de Eco,** *Marcos J. C. Euzébio*

95-17 **Anais da II Oficina Nacional em Problemas Combinatórios: Teoria, Algoritmos e Aplicações,** *Editores: Marcus Vinicius S. Poggi de Aragão, Cid Carvalho de Souza*

95-18 **Asynchronous Teams: A Multi-Algorithm Approach for Solving Combinatorial Multiobjective Optimization Problems,** *Rosiane de Freitas Rodrigues, Pedro Sérgio de Souza*

95-19 **wxWindows: Uma Introdução,** *Carlos Neves Júnior, Tallys Hoover Yunes, Fábio Nogueira de Lucena, Hans Kurt E. Liesenberg*

95-20 **John von Neumann: Suas Contribuições à Computação,** *Tomasz Kowaltowski*

95-21 **A Linear Time Algorithm for Binary Phylogeny using PQ-Trees,** *J. Meidanis and E. G. Munuera*

95-22 **Text Structure Aiming at Machine Translation,** *Horacio Saggion and Ariadne Carvalho*

95-23 **Cálculo de la Estructura de un Texto en un Sistema de Procesamiento de Lenguaje Natural,** *Horacio Saggion and Ariadne Carvalho*

95-24 **ATIFS: Um Ambiente de Testes baseado em Inje,c ao de Falhas por Software,** *Eliane Martins*

95-25 **Multiware Plataform: Some Issues About the Middleware Layer,** *Edmundo Roberto Mauro Madeira*

95-26 **WorkFlow Systems: a few definitions and a few suggestions,** *Paulo Barthelmess and Jacques Wainer*

95-27 **Workflow Modeling,** *Paulo Barthelmess and Jacques Wainer*

Relatórios Técnicos – 1996

96-01 **Construção de Interfaces Homem-Computador: Uma Proposta Revisada de Disciplina de Graduação,** *F'abio Nogueira Lucena and Hans K.E. Liesenberg*

96Abs **DCC-IMECC-UNICAMP Technical Reports 1992–1996 Abstracts,** *C. L. Lucchesi and P. J. de Rezende and J.Stolfi*

Instituto de Computação
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br