The Surface Evolver

Kenneth A. Brakke

CONTENTS

The Surface Evolver is a computer program that minimizes the energy of a surface subject to constraints. The surface is represented as a simplicial complex. The energy can include surface tension, gravity and other forms. Constraints can be geometrical constraints on vertex positions or constraints on integrated quantities such as body volumes. The minimization is done by evolving the surface down the energy gradient. This paper describes the mathematical model used and the operations available to interactively modify the surface.

INTRODUCTION

One of the fundamental problems in the calculus of variations is to find a surface minimizing some energy subject to constraints. A soap film on a wire frame minimizes its area subject to its boundary staying on the frame. A cluster of soap bubbles minimizes the total soap film area subject to enclosing fixed volumes in each bubble. A capillary surface minimizes the gravitational energy of a fluid in a vessel plus the surface energy of its free surface and its contact energy with the vessel walls. Other examples of surfaces are grain boundaries in metals, crystal facets, fluid interfaces and cell membranes [Almgren 1982; Almgren and Taylor 1976]. Naturally occurring surfaces need not be simply connected, need not be orientable (as in a Möbius-band soap film) and need not be manifolds (as in bubble clusters).

The Surface Evolver is an interactive program for the study of surfaces shaped by surface tension and other energies. The user specifies an initial surface, the constraints that the surface should satisfy throughout the evolution, and an energy function that depends on the surface. The Evolver then modifies the surface, subject to the given constraints, so as to minimize the energy. The user can intervene during the evolution, changing the sur-

1058-6458/92 $0.50 per page

face's properties or applying certain operations to keep the evolution well-behaved.

The action of the Evolver is meant to model the process of evolution by mean curvature, which was studied in [Brakke 1977] for surface tension energy in the context of varifolds and geometric measure theory. The energy in the Evolver can be a combination of surface tension, gravitational energy, squared mean curvature or user-defined surface integrals. The Evolver can handle complicated topology (as seen in real soap bubble clusters), volume constraints, boundary constraints, boundary contact angles, prescribed mean curvature, crystalline integrands, gravity, and constraints expressed as surface integrals.

The main focus of the Surface Evolver is on two-dimensional surfaces in three-dimensional space, the so-called soap-film model. The principal data structures are set up with this in mind, but they are designed in such a way that the dimension of the ambient space and of the "surfaces" of interest can be arbitrary. Thus, one-dimensional strings and higher-dimensional surfaces can also be handled. Moreover, the ambient space can be endowed with an arbitrary Riemannian metric, and even be a quotient space under a group action.

The Evolver has a graphical interface that allows the user to follow the evolution of the surface on the screen. The graphics can also be output to files in several formats, including PostScript.

The Surface Evolver is freely available (see "Software Availability" at the end of this article) and is in use by a number of researchers. Some of the applications of the Evolver so far include modeling the shape of fuel in rocket tanks in low gravity [Tegart 1991], calculating areas for the opaque cube problem [Brakke 1991b], computing capillary surfaces in cubes [Mittelmann and Hornung] and in exotic containers [Callahan et al. 1991], simulating grain growth, studying grain boundaries pinned by inclusions, searching for partitions of space more efficient than Kelvin's tetrakaidecahedra, modeling the shape of molten solder on microcircuits [Racz et al.], studying polymer chain packing and classifying minimal-surface singularities. Section 9 of this paper gives a proof, based on the use of the Evolver, that a conjectured area-minimizing cone in $\mathbf{R}^4$ is not area-minimizing.

The strength of the Surface Evolver program is in the breadth of problems it handles, rather than in the optimal treatment of some specific problem. It is under continuing development, and not every feature is described in this paper. Users are invited to suggest new features.

This paper gives an overview of the capabilities of the Surface Evolver so that readers can evaluate its usefulness in their research and so that users have a published description to cite. It is not a substitute for the manual [Brakke 1991a], although it does mention certain operational details so that readers can find the corresponding features in the program or manual.

ACKNOWLEDGEMENTS

The Surface Evolver is a result of my participation in The Geometry Center at the University of Minnesota (formally The National Science and Technology Research Center for Computation and Visualization of Geometric Structures, formerly The Geometry Supercomputer Project). I would like to thank Fred Almgren, Jean Taylor and Al Marden for making my participation in The Geometry Center possible and for their enthusiasm. I would also like to thank Charlie Gunn, Toby Orloff, Stuart Levy and the rest of the staff of The Geometry Center for technical assistance, especially in graphics. Thanks also to John Sullivan, who, having used the Evolver since its inception, has made many suggestions and found many bugs.

1. GENERAL OVERVIEW

As discussed above, the primary concepts in the Surface Evolver are the surface, the energy function and the constraints. In this section I briefly discuss each of these elements, and also the operations available to the user for controlling the evolution.

1.1. Representing Surfaces

Surfaces have been represented mathematically as graphs of functions, level sets of functions, images of maps, measures, simplicial complexes, polyhedral complexes, spline patches, and so forth. Each way has its strengths and weaknesses. The Surface Evolver uses a finite-element method, representing a surface as a union of simplices. This permits the representation of surfaces like soap films and bubbles, which may have complicated topologies and may not be orientable. It allows the specification of a surface in terms of a finite amount of combinatorial and geometric information, the latter consisting of the vertex coordinates. Many of the figures accompanying this paper show the component simplices.

The Evolver represents a soap-film model surface as a simplicial complex consisting of vertices, edges and facets. A vertex is a point, and its principal attributes are its coordinates. Each edge has a head and tail vertex, and each facet is defined by a chain of three oriented edges. In addition, it is possible to define bodies by giving for each body a list of oriented facets that make up its boundary. It is not necessary to have a simplicial decomposition of the interior of a body, since the Evolver never attempts to integrate over a body, only over its boundary. The initial surface is defined in a data file that lists the combinatorial information (see Section 8).

The units of measurement are dimensionless. If the user wishes to model a specific physical problem, all values should be in one consistent set of units such as cgs or MKS.

Section 3 describes what is available in the Evolver in terms of alternatives or elaborations to the basic soap-film model: "surfaces" of arbitrary dimension, ambient spaces with an arbitrary Riemannian metric, and so on.

1.2. Energies

Broadly speaking, the energies that the Evolver minimizes are any quantities that may be expressed as integrals over the surface. Foremost among them is surface tension. Soap films and interfaces between different fluids have an energy proportional to their area, which can also be regarded as a surface tension, or force per unit length. That is, across any line in the surface, there is a tension whose value is the same as the surface energy density. In the Evolver, the user can specify a value of the surface tension for each facet.

Another common energy is gravitational potential energy, which can be written as a surface integral by means of the divergence theorem. Capillary surfaces may be modeled by including both surface tension and gravitational energy in the total energy. For more details, see Section 4.

1.3. Constraints

Several types of constraints are available in the Evolver. Vertices may be fixed in place, reflecting the fact that the edge of a soap film should be attached to a wire. Vertices may be constrained to lie on smooth manifolds, reflecting the case when the edge of a soap film should lie on a wall, but is free to move on the wall. Edges and facets may likewise be constrained, which simply means that any vertices generated on them will inherit those constraints. Bodies may be constrained to have fixed volumes, as in the case of the volume of the column of liquid underneath a capillary surface. Section 5 explains the exact mathematical form of the various types of constraints.

1.4. Basic Operations

The fundamental operation of the Evolver is the iteration step, which reduces energy while obeying any constraints. A gradient descent method is used. The force at each vertex is the gradient of the total energy as a function of the position of that vertex. Every vertex is moved simultaneously by a global multiple of its force. This multiple, called the *scale factor*, can either be fixed by the user or be the factor that optimizes the decrease in energy. Details of all the options available for the iteration step are given in Section 6.

Several operations are available for manipulating the triangulation. *Refinement* is the subdivision of each facet into four similar facets, for the better approximation of curved surfaces. *Equiangulation* readjusts the triangulation of a surface to make the facets as nearly equilateral as possible. *Vertex averaging* moves each vertex to the average position of its neighboring vertices. These and other operations, including some that change the topology of the surface, are more fully described in Section 7.

2. THREE EXAMPLES OF THE EVOLVER IN ACTION

2.1. The Catenoid

The catenoid is the minimal surface whose boundary consists of two parallel rings not too far apart. It is an extremely simple surface, yet it illustrates some of the subtleties of evolving triangulated surfaces. Stages in its evolution are shown in Figure 1. The surface in the initial data file consists of six rectangles forming a cylinder between the two

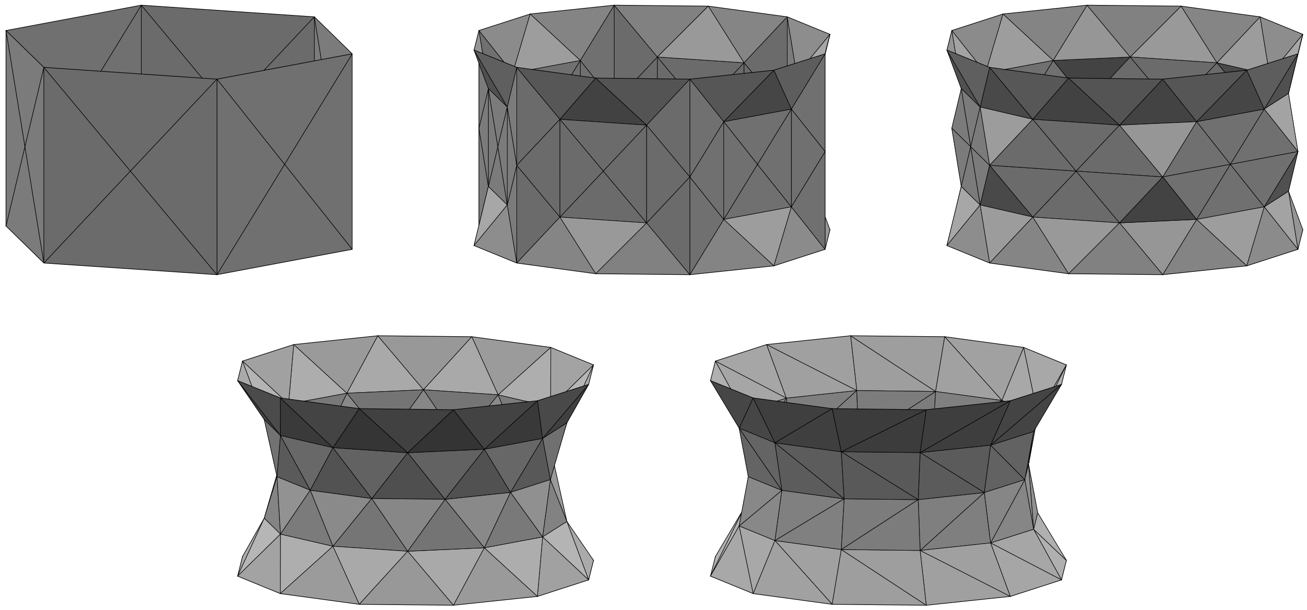

FIGURE 1. The evolution of a stable catenoid. Top left: The initial surface. The boundary wire circles are not shown. The rectangular faces of the data file have been automatically triangulated. Top middle: After one refinement. Note how the vertices on top and bottom edges follow the boundary wire circles. Top right: After equiangulation. Note the edges that have switched direction. Bottom left: After iterating fifty times. This is a saddle point in the area. Bottom right: Ultimate endpoint of iteration, with edges following the lines of curvature, which are horizontal and vertical.

rings. In general, a data file contains only the minimum amount of information needed to correctly define the topology of the surface. When initially read in, the rectangles are automatically triangulated into facets (top left). The vertices and edges on the rings are fixed. The rings themselves are not shown. With so few facets, the initial surface cannot shrink, so the user refines the surface (top middle). Here, only one refinement is done to keep the facets large enough to be seen easily. Normally there would be alternating stages of refinement and iteration. Note that the vertices created by subdividing the edges on the rings are themselves fixed on the rings. Equiangulation gives the much nicer triangulation shown top right, by switching the diagonals of some quadrilaterals to make the facets more equiangular. Fifty iterations with optimizing scale factor result in an area of 6.458483 (bottom left). At this point, each iteration is reducing the area by only .0000001, the triangles are all nearly equilateral, everything looks nice, and the innocent user might conclude that the surface is very near its minimum. But we are really near a saddle point of energy. Another 300 iterations get the area down to 6.4336 (bottom right), near the true local minimum. We know it is a minimum because iteration produces no change, and the Evolver can calculate the Hessian matrix to be positive definite. One can see that the triangulation really wants to be twisted around so that there are edges following the lines of curvature.

If the two rings are too far apart, the neck of the catenoid will shrink down to a point, as shown in Figure 2. Upon iteration, the neck forms a ring of very short edges (top middle). These edges can be removed by identifying their endpoints, in a step (taken by the user) known as tiny-edge weeding (§ 7.5). This produces a single vertex at the neck (top right). The Evolver can recognize the topology around the neck vertex as improper for a soap film and split the vertex into two (bottom left), when instructed to do so by the user. This is called vertex popping (§ 7.7). The two parts of the surface then quickly collapse to disks (bottom right).

2.2. Capillary Surface Meeting a Wall

Figure 3 shows the evolution of an example where we use constraints and varying surface tension to model a surface meeting a wall at a given contact angle—here 60 degrees. This is the situation for a capillary surface, but this example does not include volume constraints or gravity.

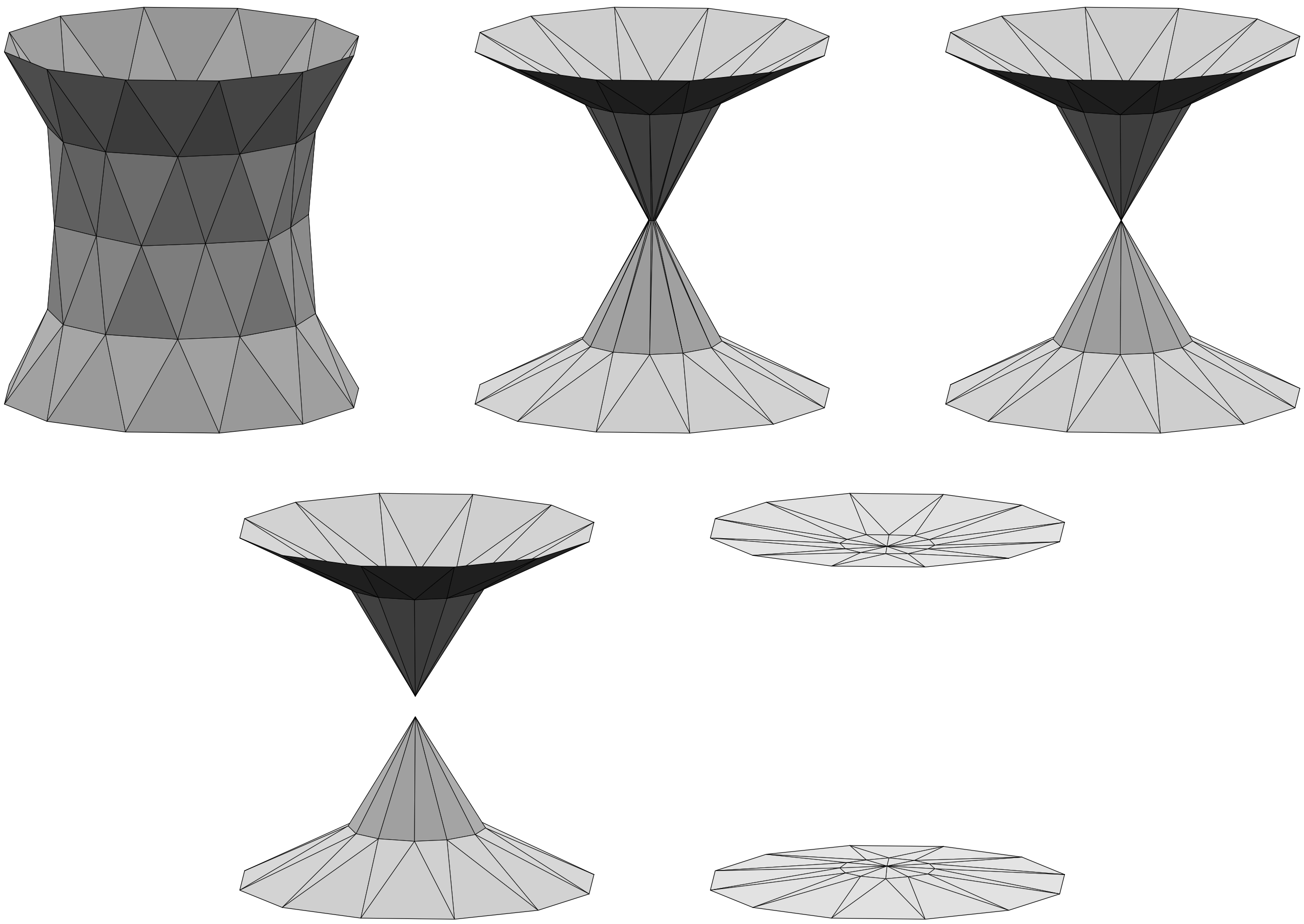

FIGURE 2. The evolution of an unstable catenoid. The neck pinches down to a point, and the surface splits into two pieces.

The initial configuration is shown on the left. The fluid surface is the light gray horizontal surface, which has one edge movably attached to the dark gray vertical wall. The junction of the surface and wall consists of edges belonging to both surface and wall facets. We give the wall facets half the surface tension of the surface facets, so the equilibrium contact angle is 60 degrees ($0.5 = \cos 60^\circ$). All vertices on the wall are constrained to stay in a vertical plane. The three vertices at the top of the wall are fixed in place, as are the vertices of the surface on the edge opposite the wall. The two lateral sides of the surface and wall are constrained to lie in vertical planes (not shown) perpendicular to the wall. These planes contain no facets and contribute no energy, so the equilibrium contact angle is 90 degrees.

After several iterations (center left), the contact line has moved up the wall, seeking the equilibrium contact angle. The shrinking of the wall facets more than offsets the stretching of the surface facets. The interior vertices of the wall do not move, since there are no net forces on them. This can give rise to problems if the contact line overruns interior wall vertices, as is about to happen. At this point, the user must intervene to adjust the triangulation, using vertex averaging, after which (center right) the contact line can continue to move up the wall. The final equilibrium state consists of a plane surface (far right).

The wall facets serve two purposes: to generate the correct contact angle and to help visualize the surface. But they also are the source of problems as the surface moves up the wall, requiring repeated vertex averaging. They occupy more computer memory and calculation time than necessary. It is possible to omit them. In § 4.5 I show how to use edge integrals to compute the equivalent energy of the facets. The visualization function could be served by having fixed facets on the wall that do not participate in the calculations and do not refine.

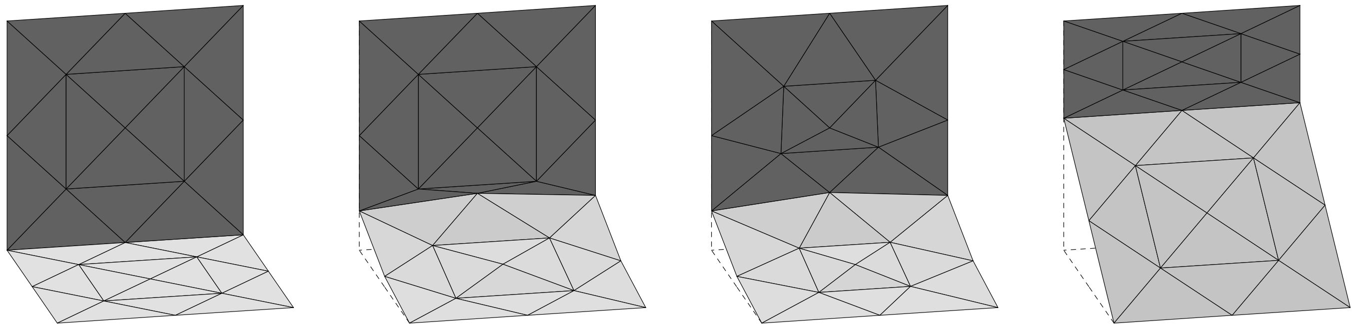

FIGURE 3. Capillary surface making contact with a wall. Left: The initial surface. The facets on the vertical wall (darker shading) have surface tension 0.5, while the ones on the horizontal free surface have surface tension 1.0. Thus the equilibrium contact angle is 60 degrees. The free surface is bounded by a fixed wire in front and plane constraints on either side. Center left: After some evolution, the free edge has crept up the wall. The dashed lines indicate the original position. Note that several wall facets are getting very narrow, which will soon cause problems in the evolution. Center right: After vertex averaging, the free edge has more room to migrate without overrunning vertices. Right: The final equilibrium surface.

2.3. Grain Growth

When liquid metal solidifies, crystallization generally starts at many nuclei, with random orientations. The crystal lattices are mismatched where the grains meet, and the atoms along the grain boundaries are in a higher energy state than interior atoms. To a good approximation, the energy is independent of the orientation of the boundary. In the process of annealing, the metal is warmed enough for boundary atoms to switch from one lattice to the next through thermal motions, and the boundaries migrate at a rate proportional to their curvature, assuming that impurities or other obstacles do not interfere.

Figure 4 shows this process for a two-dimensional metal in a unit flat torus that initially crystallizes from 100 random nuclei, resulting in an initial grain configuration of 100 Voronoi cells (top left). The ultimate product of evolution consists of four unequal hexagonal grains. A video of the evolution is available in [Brakke 1992b].

Modeling the dynamics of the evolution requires using a fixed scale factor (the time step) much smaller than the optimizing scale factor. Here, a time step of 5×10^{-6} was used. The Evolver has several features used to automate the evolution, including automatic topology changes in the string model. With a feature called *autopopping*, any edge whose length is projected to become less than a critical length is deleted and any resulting improper vertices are popped. The critical length is automatically set to the critical length for instability described in § 6.4, which works out to 0.003 here. Only edges whose lengths are decreasing are tested, so the very short edges generated by vertex popping will not be eliminated. With *autochopping*, edges that become longer than a chosen cutoff length of 0.015 are automatically subdivided.

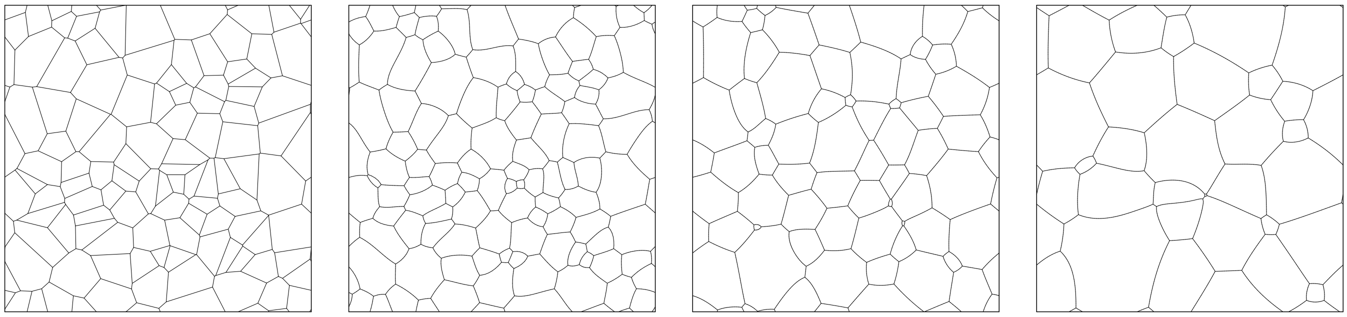

FIGURE 4. The evolution by mean curvature of 100 Voronoi cells in a two-dimensional flat torus. The time step is 5×10^{-6}. From left to right: The initial configuration; configuration after 200 steps, still with 100 cells; after 1600 steps, with 60 cells left; and after 5000 steps, with 28 cells left.

3. SURFACE MODELS

This section describes several variations on the basic soap-film model discussed in § 1.1.

3.1. One-Dimensional "Surfaces": The String Model

The "surface tension" can be declared to reside in edges instead of facets. The surface then becomes a network of elastic strings, hence the term *string model* is applied to this mode of operation. The strings may reside in a space of any dimension, but if the domain is two-dimensional, the strings may bound regions. In this case, a region is defined with a facet structure and a body structure. The facet may have any number of sides. The body has just one facet on its boundary. The effect is to stretch the string network into a cylindrical surface of height 1, whence the mechanisms for surfaces can be applied with minimal changes. The grain growth example in § 2.3 uses the string model.

3.2. Quadratic Model

In an attempt to approximate curved surfaces better than by flat facets, there is a mode in which each facet is a quadratic spline patch. A midpoint is added to each edge, giving a total of six control points. Each coordinate is then quadratically interpolated from these six points to form the surface. An edge becomes a curve that depends only on the control points on the edge, thus guaranteeing that neighboring facets meet without a gap. The disadvantages of the quadratic mode are that it is slower, surface area is calculated approximately by numerical integration, facets are displayed as if flat, and some Evolver features are not implemented.

3.3. Higher-Dimensional Surfaces

Higher-dimensional surfaces cannot be represented by the basic vertex–edge–facet scheme. There is a mode of operation in which the facets making up the surface are represented directly as simplices (vertex lists). This permits a surface of arbitrary dimension in a space of arbitrary dimension. However, many features are not yet implemented for this mode.

3.4. Quotient Spaces

The ambient space can be a quotient space of $\mathbf{R}^n$ under some symmetry group. The vertex coordinates are taken to be in a fundamental region, and each edge is marked with a group element to tell how its head vertex should be transformed (wrapped) relative to its tail. The user invents an integer representation for the group elements to be used in marking edges. The flat torus quotient space is built-in, and can be specified in the data file in terms of the fundamental parallelepiped and the wraps of the edges. Other quotient spaces require the user to write C-language functions that handle group transformations and compositions.

The display of a surface in a quotient space poses some interesting problems, since the display space is Euclidean. The Evolver offers three options, two of which are illustrated in Figure 5, which shows

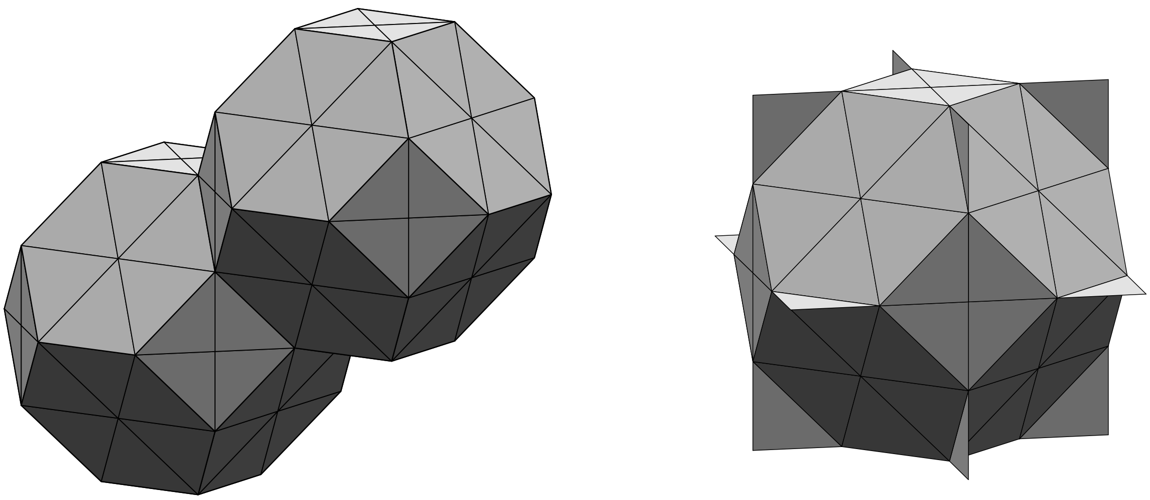

FIGURE 5. Two of Kelvin's tetrakaidecahedra in a flat torus. The fundamental region is a unit cube. Left: The surface plotted as the boundaries of connected bodies. Right: The surface clipped to the fundamental region, a unit cube.

a surface in a flat three-dimensional torus whose fundamental parallelepiped is a unit cube. The three options are:

Raw facets. Each facet is displayed as it is located in the fundamental domain. Location is based on the first vertex in the facet, with other vertices being unwrapped as needed.

Connected bodies. All facets on the boundary of each body are gathered in a list. One point is chosen as a base point, and unwrapping of vertices spreads out from neighbor to neighbor facet until the entire body surface is done. This nicely displays each logical body as a unit. An example is shown in Figure 5 (left).

Clipping. For a torus domain, each facet is clipped to the fundamental parallelepiped. Pieces lying outside are wrapped around so they lie inside. This option clearly shows the fundamental domain and how it wraps around. An example is shown in Figure 5 (right).

The surface shown in Figure 5 bounds two tetrakaidecahedra. Lord Kelvin [Thompson 1987] conjectured that the optimal way to partition space into equal-volume cells with least area is a packing of very slightly curved tetrakaidecahedra. Several people (including the author) have used the Evolver in attempts to beat Kelvin's partition, but nobody has succeeded yet.

3.5. Background Metric

The ambient space can be endowed with a Riemannian metric. Only one coordinate patch is allowed, but quotient spaces are possible. Basically, the Evolver operates as usual in Euclidean coordinates, except that the metric is used for the calculation of edge lengths and facet areas. Edges are not taken to be geodesics, nor are facets geodesic surfaces; rather, they keep their Euclidean shape. Surfaces are displayed as if their coordinates were in Euclidean space.

An example using a metric is given in Section 9. Other possible ways to use a metric are: modeling a cylindrically symmetric surface by means of the string model, and implementing a spatially varying scalar surface energy, as in a surface whose own weight is not negligible.

The metric need not be positive definite. One can do minimal surfaces in a Minkowski metric, but it takes a little care.

3.6. Internal Representation

Each geometric element (vertex, edge, facet, body) is implemented as one data structure. An element is stored as an oriented entity, but may be referred to with this orientation or the inverse one. There is a data type *element_id* that contains a pointer to an element structure and a relative orientation (normal or inverted). This type is used for all references to elements. The connectivity of the surface is specified by having links in each element structure to the next higher- or lower-dimensional elements it intersects. One design principle followed is that each element should contain links to a fixed number of other elements. Thus the body structure does not record bounding facets itself; rather, the facet structure has two slots to record which body (if any) is on each of its sides.

The surface connectivity is completed by introducing *facet-edge* structures, which are a simplification of the scheme described in [Dobkins and Laszlo 1987]. A facet-edge structure contains links to a facet and an edge on the facet's perimeter (with proper boundary orientation), plus links to the previous and next facet-edges around the facet and to the previous and next facet-edges around the edge. This permits a quick run-through of all facets containing a given edge, and also the representation of facets with an arbitrary number of sides, which is necessary in certain situations.

4. ENERGIES

This section describes the various forms of energy that may be combined into the Evolver's total energy function.

4.1. Surface Tension

Soap films and interfaces between different fluids have an energy proportional to their area. In the Evolver, each facet has a surface tension of 1 unless otherwise specified. Different facets may have different surface tensions. It is possible to endow both facets and chosen edges with tension in order to model surfaces where singular curves have energy, as in [Morgan].

Contact angles between free surfaces and walls, as in capillary problems, can be specified by introducing facets that are confined to the wall and have a different surface tension, as in the wall example in § 2.2. Negative tensions are allowed, so that all contact angles are possible. However, this method has the drawback that a moving free boundary on the wall can overrun wall facets, as shown in Figure 3. Another method of prescribing contact angles, described in § 4.5, uses edge-energy integrals.

There is no general mechanism yet to include the integral over the surface of a general scalar integrand that may depend on position and tangent plane orientation. However, the use of a Riemannian metric on the ambient space can often achieve the same effect.

4.2. Crystalline Integrands

The Evolver can model energies of crystalline surfaces. A crystalline surface energy density depends on the direction of the normal vector to the surface. Such a quantity is known as a *crystalline integrand* [Taylor 1983; 1988]. The energy density in this case is given by the largest dot product of the surface normal with a set of vectors known as the *Wulff vectors*. Surface area can be regarded as a crystalline integrand with respect to a set of Wulff vectors coinciding with the unit sphere. In the Evolver, a finite set of Wulff vectors may be specified, and the corresponding crystalline energy computed.

A surface can have either crystalline energy or surface tension, not both.

4.3. Surface Integrals

A facet may contribute an energy resulting from integrating a vector field over the facet as a surface integral. Multiple vector fields may be defined in the data file as functions of the coordinates, and any facet may use any number of them. Besides surface energies, surface integrals can be used to handle volume energies, such as gravitational energy, thanks to the divergence theorem. Integrals are calculated numerically using Gaussian quadrature.

4.4. Gravity

A body B having density ρ contributes its gravitational energy to the total. The acceleration of gravity G is under user control. The gravitational energy is defined as

$$E = G\rho \iiint_B z\,dV,$$

but is calculated by the divergence theorem as

$$E = G\rho \iint_{\partial B} \frac{z^2}{2}\vec{k} \cdot d\vec{S}.$$

The integral is taken over each facet that bounds a body. If a facet bounds two bodies of different densities, the appropriate difference in density is used. Vertical facets or facets lying in the plane $z = 0$ make no contribution and may be omitted if they are not needed otherwise. Facets lying in constraints may be omitted if their contributions to the gravitational energy are contained in edge energy integrals.

Gravity is a special case of a surface integrand, but it is implemented internally for several reasons: it can be evaluated exactly without numerical integration; it is common enough to be worth saving the user the trouble of setting it up; and, in general, evaluation of user-defined integrand expressions is slower than that of compiled-in energies.

The built-in gravity does not apply to Riemannian metrics or quotient spaces; users must define their own gravitational energy integrands in such cases.

Gravity applies to bodies, not surfaces. Surfaces are weightless. If one does want heavy surfaces, one can use the metric mechanism to simulate a scalar surface integrand.

4.5. Edge Integrals

An edge may contribute an energy resulting from integrating a vector field over the edge as a line integral. An edge integral can be associated with a constraint: it then applies to every edge subject to that constraint. The objective of this is to let the free edges of a surface have energy. This is useful in controlling the contact angle of a surface on a wall. As mentioned in § 4.1, the contact angle can be specified by giving an energy density to the wall on one side of the surface edge. Alternatively, an edge integral can be defined that gives an energy equivalent to the wall energy, thanks to Stokes' Theorem. This eliminates the need to coat the wall with facets. Likewise, edge integrals can be used to replace other facet energies, such as gravitational energy, for facets on constraints.

An edge integral is evaluated once for each edge associated with it—regardless of how many facets it is on—using the orientation inherited from the data file.

4.6. Prescribed Mean Curvature and Pressure

For an equilibrium surface with a constant surface tension, the mean curvature is proportional to the pressure difference across the surface. Therefore, prescribing the mean curvature is equivalent to prescribing the pressure difference. The Evolver permits the user to prescribe pressures in bodies. Pressure can be defined as the rate of change of energy with respect to volume, so the pressure feature is implemented by having each body with a prescribed pressure P contribute energy $E = -PV$, where V is the actual volume of the body. The energy is actually calculated by a surface integral

$$E = -P \iint_{\partial B} z\, \vec{k} \cdot d\vec{S}.$$

The desired surface need not really be the entire boundary of a body; it may have fixed edges. The energy contributed by the omitted boundary is constant and so does not affect the shape of the surface.

This method will only work if the desired surface is stable at the prescribed pressure. If curvature decreases as volume increases, the surface will either blow up or implode. For example, if a round soap bubble of surface tension T and initial radius R_0 is prescribed to have pressure $P > 2T/R_0$, the pressure force will cause the bubble to expand. This reduces the curvature, so the pressure can never be balanced by the curvature, and the bubble expands indefinitely.

4.7. Squared Mean Curvature

There are circumstances under which one wants the energy to include the integral over the surface of the squared mean curvature. Surfaces that minimize this integral are called *Willmore surfaces*. This presents a problem, in that, for our piecewise linear surfaces, the mean curvature (in the form of first-variation measure) is singular and concentrated on the edges. Its square integral is therefore always infinite. However, it is possible to come up with a usable approximation. An average mean curvature around each vertex can be calculated, and the integral of the square of this average can be counted as energy.

The definition of mean curvature used here is a variational one, and corresponds to the average of the sectional curvatures rather than their sum. The integral of squared mean curvature in the soap-film model is calculated as follows: Each vertex v has a star of facets around it of total area A_v. The force on the vertex is

$$\vec{F}_v = -\frac{\partial A_v}{\partial v}.$$

Mean curvature is proportional to force per unit area. Since each facet has three vertices, the area associated with v is $\frac{1}{3}A_v$. Hence the average mean curvature at v is taken as

$$\vec{h}_v = -\frac{3}{2}\frac{\vec{F}_v}{A_v},$$

and this vertex's contribution to the total integral is

$$E_v = \tfrac{1}{3}h_v^2 A_v = \frac{3}{4}\frac{F_v^2}{A_v}.$$

E_v can be written as an exact function of the vertex coordinates, so the gradient of E_v can be fed into the total force calculation.

The alternative to locating curvature at vertices is to locate it on the edges, where it really is, and to average it over the neighboring facets. But this has the problem that a least-area triangulated surface would have nonzero squared curvature, whereas in the vertex formulation it has zero squared curvature.

Squared mean curvature is also implemented for the string model, but not for quadratic models. In the string model, let L_v be the sum of the lengths of the edges adjacent to v, so the force on a vertex is

$$\vec{F}_v = -\frac{\partial L_v}{\partial v}.$$

Each edge has two endpoints, so the length associated with v is $\frac{1}{2}L_v$, the curvature is

$$\vec{h}_v = -\frac{2\vec{F}_v}{L_v},$$

and the vertex's contribution to the total integral is

$$E_v = \tfrac{1}{2}h_v^2 L_v = \frac{2F_v^2}{L_v}.$$

4.8. Gaps

Consider a soap film spanning a circular wire. The Evolver must approximate this surface with a collection of facets, as shown in Figure 6. The straight edges of these facets cannot conform to the curved

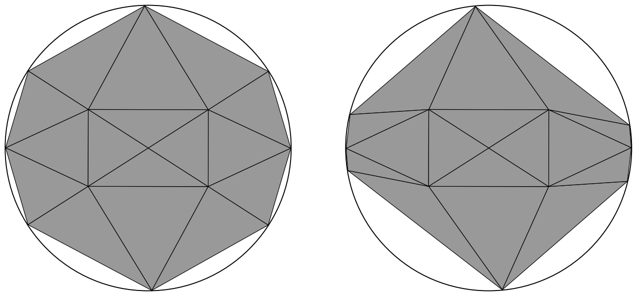

FIGURE 6. Surface in a ring, showing the gap problem. Left: The initial surface. The vertices are free to move along the boundary wire. Right: After one iteration, the gaps have grown and the surface area has shrunk.

wire, so the computed area of the surface leaves out the gaps between the outer edges and the wire. If the vertices are free to move along the wire, but not off of it, the Evolver will naturally try to minimize area by moving the outer vertices around so that the gaps increase, which is what is happening in Figure 6. This is not good. Sometimes the vertices can be fixed on the wire, but at other times this is not possible, for example, when the surface is spanning the inside of a cylinder and the edge of the surface is free to move on the cylinder. Therefore, there is provision for a "gap energy" to discourage growing gaps. A constraint of the type defined in § 5.1 may be declared *convex* in the data file. For an edge on such a constraint, an energy is calculated as

$$E = \tfrac{1}{6}k\,\|\vec{S} \times \vec{Q}\|,$$

where $\vec{S}$ is the edge vector and $\vec{Q}$ is the projection of the edge tangent to the constraint at the tail vertex of the edge. The global constant k is called the *gap constant*. A gap constant of 1 gives the best approximation to the actual area of the gap. A larger value of k minimizes gaps and gets vertices nicely spread out along the wire.

Another way to handle gaps is to define an edge integral (see § 4.5) that is zero on the constraint and positive on the convex side of the constraint. Edge integrals are evaluated on interior points of the edge, so the bigger the gap, the bigger the integral. This encourages the edges to be of equal length.

Of course, the introduction of any additional energy component changes the problem slightly. But the gap energy decreases quadratically with the fineness of the triangulation, and so should not change the solution significantly. By changing the constant associated with this energy, one can see whether the problem is being significantly altered.

In actual practice, gap energy is seldom used with a wire boundary, since it is much simpler just to declare the vertices on the wire fixed. The real use for gap energy comes with surface edges on walls, where the vertices cannot be fixed.

5. CONSTRAINTS

Energy minimization takes place subject to constraints of two types: constraints on the motion of vertices and constraints on the value of surface integrals. Vertices can be individually constrained by declaring them fixed, by confining them to level sets of functions (this is Evolver's narrow meaning of the term *constraint*), or by defining their position in terms of parameters (Evolver *boundaries*). Surface integrals in general are called *quantities*, and the particular case of body volumes is implemented internally.

5.1. Level-Set Constraints

A vertex may be confined to the zero level set of one or more functions. Such a function is called a *constraint* in Evolver terminology. In this paper, it should be clear from the context when the term constraint is used in this narrow sense and when in the broader mathematical sense. The default is the narrow sense. Constraint functions are defined by the user in the initial data file, and numbered for reference. Vertices may be declared to be on one or more constraints simultaneously, but it is the user's responsibility to ensure that the constraint function gradients at a vertex are linearly independent. There are also one-sided constraints; this means that a vertex is restricted to the region where the constraint function has nonnegative (or nonpositive) values.

When a vertex is moved for whatever reason, Newton's method is used to project it back to its constraints. There is a global constraint tolerance parameter that the user can set in order to control the accuracy of constraint satisfaction. The coordinates of a vertex in the initial data file do not have to satisfy its constraints exactly; the vertex will be automatically projected. A vertex on constraints may also be declared fixed, which means that it will not move after its initial projection. If a constraint is modified during runtime by changing an adjustable parameter, all vertices are projected again so as to satisfy the new constraint.

Edges and facets may be declared to be on constraints. This means that all vertices generated by subdividing them will be on the same constraints.

A single constraint is the best way to attach a free edge of a surface to a wall. Two constraints confine a vertex to a curve; but if a one-dimensional wire is desired instead of a two-dimensional wall, it may be better to use the parametrized boundary feature described below.

5.2. Boundaries

Evolver *boundaries* are one- or two-dimensional parametrized manifolds; they are an alternate way to constrain the position of vertices. A vertex on a boundary cannot also have constraints. Vertices, edges and facets may be deemed to lie in a boundary. For a vertex, this means that the fundamental parameters of the vertex are the parameters of the boundary, and its coordinates are calculated from these. When a vertex on a boundary moves, the motion is projected back to parameter space and applied to the parameters. Edges and facets on a boundary bequeath the boundary to descendant vertices.

A delicate question is how to deal with wraparounds on a boundary such as a circle or cylinder. Subdividing a boundary edge requires a midpoint, but taking the average parameters of the endpoints can give nonsense for an edge where the parameter value is discontinuous. Therefore the average coordinates are calculated, and the resulting point is projected on the boundary parameters as continued from one endpoint. The rings in the catenoid example in § 2.1 are represented as one-parameter circles and show a case where the endpoint extrapolation is necessary in the refining operation.

A general guideline is to use constraints for two-dimensional walls, and boundaries for one-dimensional wires. If one uses a boundary wire, the vertices and edges on the boundary can probably be declared to be fixed. Then the boundary becomes just a guide for refining the boundary edges.

5.3. Quantities

A *quantity* in the Evolver is a value that can be expressed as the sum of integrals of vector fields over surfaces and edges. Quantities can be evaluated for informational purposes, or they can be used as constraints (in the mathematical sense). Body volume is an example of a built-in quantity. Surface area is not a quantity in this sense, since it cannot be written as a vector integral.

User-defined quantities are supported—for example, one might use as quantities the center of mass, the moment of inertia, the magnetic flux, and so on. The key in many of these cases is the divergence theorem, which permits volume integrals to be written as surface integrals. One can also use a quantity in place of a body volume that is awkward to handle with the built-in volume mechanisms, as when a background metric is used.

Each quantity is specified in the data file by the surface integrand and the facets it is to be integrated over, plus the edge integrand and the edges it is to be integrated over. A quantity acts as a mathematical constraint when it is declared fixed and given a specified value. Its actual value may be displayed and its target value changed interactively. Quantities have the same mathematical form as the surface and edge energy integrals discussed in Section 4, but they are used for constraints or information rather than as part of the objective function.

5.4. Volumes

The term *volume* is used here for the highest-dimensional measure of a region in n-space: area in $\mathbf{R}^2$, volume in $\mathbf{R}^3$, etc. A body may have a volume specified in the data file, which then becomes a volume constraint. The volume of a body B can be written as

$$V = \iiint_B 1\, dV,$$

which by the divergence theorem can be written a surface integral:

$$V = \iint_{\partial B} z\vec{k} \cdot d\vec{S}.$$

This integral is evaluated over all the boundary facets of a body.

The part of the boundary of a body lying on a constraint need not be given in terms of facets. If it is not, the user can use Stokes' Theorem to convert the part of the surface integral on the constraint to a line integral over the edges where the body surface meets the constraint. The line integrands are given as part of the constraint definition in the data file. These edge volume integrals can also be used to overcome the volume calculation problems caused by the gaps between curved constraints and flat facets.

Volumes are a special case of quantities, and are implemented internally for Euclidean space and flat torus domains only. In general quotient spaces or in Riemannian metrics, it is up to the user to define volume constraints by using the general quantity constraint mechanism.

5.5. Volumes in a Torus Domain

The volume of a body can be automatically calculated in a torus domain, but the wrapping of the edges across the faces of the fundamental region makes the calculation tricky. Ideally, we would like to adjust the vertices by multiples of the fundamental region basis vectors to get a body whose volume we can find with regular Euclidean methods. Unfortunately, all we know are the edge wraps, that is, the differences in the adjustments to endpoints of edges. But this turns out to be enough if we are careful with the initial volumes in the data file.

Let the facets of a body be indexed by m, and let v_{mi}, for $i = 0, 1, 2$, be the vertices of facet m. Let $\vec{A}_{mi}$ be the (unknown) vertex adjustment for vertex v_{mi}, and $\vec{T}_{mi}$ the (known) wrap vector (difference in endpoint adjustments) for edge i of facet m. Then

$$\begin{aligned} V &= \tfrac{1}{6}\sum_{\text{facets } m}(\vec{v}_{m0}+\vec{A}_{m0})\cdot(\vec{v}_{m1}+\vec{A}_{m1})\times(\vec{v}_{m2}+\vec{A}_{m2}) \\ &= \tfrac{1}{6}(S_1+S_2+S_3+S_4), \end{aligned} \tag{5.1}$$

where

$$\begin{aligned} S_1 &= \sum_{\text{facets } m} \vec{v}_{m0}\cdot\vec{v}_{m1}\times\vec{v}_{m2}, \\ S_2 &= \sum_{\text{facets } m}{}' \ \vec{v}_{m0}\cdot\vec{v}_{m1}\times\vec{A}_{m2}, \\ S_3 &= \sum_{\text{facets } m}{}' \ \vec{v}_{m0}\cdot\vec{A}_{m1}\times\vec{A}_{m2}, \\ S_4 &= \sum_{\text{facets } m} \vec{A}_{m0}\cdot\vec{A}_{m1}\times\vec{A}_{m2}. \end{aligned}$$

In S_2 and S_3, the notation $\sum'$ means that each facet is included three times in cyclic permutation, once with each vertex as base point.

The first of these sums is straightforward. The second sum can be regrouped with one term for each edge j, pairing the two facets j and j' for each edge together:

$$\begin{aligned} S_2 &= \sum_{\text{edges } j} \vec{v}_{j0}\cdot\vec{v}_{j1}\times\vec{A}_{j2} + \vec{v}_{j1}\cdot\vec{v}_{j0}\times\vec{A}_{j'2} \\ &= \sum_{\text{edges } j} \vec{v}_{j0}\cdot\vec{v}_{j1}\times(\vec{A}_{j2}-\vec{A}_{j'2}) \\ &= \sum_{\text{edges } j} \vec{v}_{j0}\cdot\vec{v}_{j1}\times\frac{1}{2}(\vec{T}_{j'2}+\vec{T}_{j1}-\vec{T}_{j2}-\vec{T}_{j'1}). \end{aligned}$$

We can regroup this into a sum that can be done facet by facet, each facet appearing three times:

$$S_2 = \frac{1}{2}\sum_{\text{facets } m}{}' \ \vec{v}_{m0}\cdot\vec{v}_{m1}\times(\vec{T}_{m1}-\vec{T}_{m2}).$$

In S_3 we group terms with a common vertex together, with the inner facet sum over facets with vertex k as base vertex:

$$\begin{aligned} S_3 &= \sum_{\text{vertices } k}\left(\vec{v}_k\cdot\sum_{\text{facets } i}\vec{A}_{i1}\times\vec{A}_{i2}\right) \\ &= \sum_{\text{vertices } k}\left(\vec{v}_k\cdot\sum_{\text{facets } i}(\vec{A}_{i1}-\vec{A}_k)\times(\vec{A}_{i2}-\vec{A}_k)\right) \\ &= \sum_{\text{vertices } k}\left(\vec{v}_k\cdot\sum_{\text{facets } i}\vec{T}_{i0}\times-\vec{T}_{i2}\right) \\ &= \sum_{\text{facets } m}{}' \ \vec{v}_{m0}\cdot\vec{T}_{m2}\times\vec{T}_{m0}, \end{aligned}$$

which again can be done facet by facet. The step introducing A_k is valid, since the A_{i1} are just a relabeling of the A_{i2}, and so $\sum_i A_{i1} = \sum_i A_{i2}$.

The sum S_4 is a constant, and so only needs to be figured once. Also, it is an integer multiple of the fundamental region volume V_c, so its contribution to the body volume is a multiple of $\frac{1}{6}V_c$, by equation (5.1). Therefore, if we assume that the volume prescribed in the data file is within $\frac{1}{12}V_c$ of the actual volume, we can calculate the other sums and figure out what the fourth sum should be.

The body volume gradient at vertex v can be easily found from the above sums, since the base vertex is dotted with other terms. The gradient is a sum over all facets with base vertex v:

$$\frac{\partial V}{\partial v} = \frac{1}{6}\sum_{\text{facets } m \text{ on } v}\Big(\vec{v}_{m1}\times\vec{v}_{m2} + \tfrac{1}{2}\vec{v}_{m1}\times(\vec{T}_{m1}-\vec{T}_{m2}) + \vec{T}_{m2}\times\vec{T}_{m0}\Big).$$

6. ITERATION

The heart of the Evolver is the iteration step that reduces energy while obeying any constraints. The surface is changed by moving the vertices. No changes in topology or triangulation are made. The idea is to calculate the force at each vertex and to move the vertex in that direction, thus using a gradient descent method of minimization.

6.1. Force Calculation

The first iterative step is the calculation of the forces on the vertices. The total energy of the surface is viewed as a function of the coordinates of the vertices. The negative gradient of the energy as a function of the position of a single vertex gives the force on that vertex. Collectively, all the forces on all the vertices make up the negative of the total gradient of energy. No new approximations are introduced by the force calculation; the energy may not be exact due to numerical integrations, but the gradient is the exact gradient of the approximate energy.

Vertices on constraints have their forces projected to the tangent spaces of the constraints. Vertices on boundaries have their forces mapped back to forces on the boundary parameters. Fixed vertices have their forces set to zero.

6.2. Volume and Quantity Constraints

The second iterative step is to enforce constraints on body volumes and other integrated quantities. It has two parts. The first part consists of correcting for any errors in the current values of the quantities. The second part consists of projecting the vertex forces to be orthogonal to the quantity gradients. Both parts use the gradients of the quantities as functions of the vertices. Let $\vec{W}_{vk}$ be the gradient of quantity k as a function of the position of vertex v. These gradients are projected on constraint level sets or boundaries or are set to zero in the same manner as forces.

The value correction consists of applying a single step of Newton's method. Let the current excess value of quantity k be δ_k (which may be negative, of course). We assume that the correcting motion $\vec{R}_v$ at vertex v is of the form

$$\vec{R}_v = \sum_k c_k \vec{W}_{vk}$$

and satisfies

$$\sum_v \vec{R}_v \cdot \vec{W}_{vk} = -\delta_k \quad \text{for each } k.$$

This leads to the following linear system for the c_k:

$$\sum_k c_k \sum_v \vec{W}_{vk} \cdot \vec{W}_{vk'} = -\delta_{k'} \quad \text{for each quantity } k'.$$

This system is solved for the c_k, and hence for the motions $\vec{R}_v$. The motion is not carried out immediately, but as part of the overall motion described below. The quantity values are not perfectly corrected by this step, but over several iterations they should converge to the target values.

The second part is the projection of the forces. Let $\vec{F}_v$ be the total force at vertex v. We want a projected force $\vec{F}'_v$ of the form

$$\vec{F}'_v = \vec{F}_v - \sum_k a_k \vec{W}_{vk}$$

such that

$$\sum_v \vec{F}'_v \cdot \vec{W}_{vk} = 0 \quad \text{for each quantity } k,$$

which leads to the following linear system for a_k:

$$\sum_k a_k \sum_v \vec{W}_{vk} \cdot \vec{W}_{vk'} = \sum_v \vec{F}_v \cdot \vec{W}_{vk'}$$

for each quantity k'. The coefficients a_k are the Lagrange multipliers for the quantity constraints, and can be interpreted as pressures for body volume constraints. Whenever the user asks for body volumes to be displayed, these pressure values are also shown.

6.3. Motion

Each vertex is moved by the quantity correction motion plus a scale factor times the force at the vertex. The scale factor is a global constant, the same for all vertices. Its physical interpretation is the time step over which the velocity acts. The user may set the scale factor explicitly or let the Evolver seek the optimal value. In the latter mode, the Evolver will successively double or halve the scale factor until a minimum in energy is bracketed. Then quadratic interpolation is used to estimate the optimum scale factor, and that value is used in the final motion.

Each time a motion is done, all vertices on constraints are projected back to their constraints by repeated application of Newton's method, until the constraint function value is smaller than a constraint tolerance factor, which the user may set. If vertices subject to one-sided constraints are on the wrong side of the constraint, they are projected to the constraint. If such a vertex wants to move to the proper side of the constraint, it is freed from the constraint.

From experience, it seems that, for two-dimensional surfaces driven by surface tension, the optimum scale factor is around 0.2, independent of the fineness of the triangulation, as long as the surface is evolving without problems. The universality of the scale factor is expected in this case, since the scale factor is unitless in length for two-dimensional area only. When the scale factor dives toward zero, this usually signifies some impending problem, like an edge length or a facet area becoming zero. Unfortunately, the value 0.2 applies best to minimizing area; other models, such as the string model and squared mean curvature, have "normal" scale factors that vary with triangulation size and other factors, so it is hard to tell when the surface is evolving properly and when it is getting into trouble. In these circumstances, other methods have to be used, such as monitoring the surface visually and checking edge length and facet area histograms.

6.4. Motion by Mean Curvature

The mean curvature vector field $\vec{h}$ of a surface S is defined to be the gradient of the area of S, in the sense that if S is deformed with an instantaneous velocity $\vec{u}$, the rate of change in its area is

$$\frac{dA}{dt} = \iint_S \vec{u} \cdot \vec{h}\, dA.$$

By definition, $\vec{u} = -\vec{h}$ under motion by mean curvature, so that

$$\frac{dA}{dt} = -\iint_S h^2\, dA. \tag{6.1}$$

By default, the gradients of quantities like energy and volume are calculated simply as the gradient of the quantity as a function of vertex position. This gives the force on a vertex, for example. But to simulate motion by mean curvature, it is necessary to have force per area instead. In the triangulation formulation, let A_v be the area of the star of facets around vertex v and let

$$\vec{F}_v = -\frac{\partial A_v}{\partial v}$$

be the force on v. Then

$$\frac{dA}{dt} = \sum_v \vec{u}(v) \cdot -\vec{F}_v.$$

Take the area associated with a vertex to be one-third of the total areas of the facets surrounding the vertex, $dA = \frac{1}{3}A_v$. Since each facet has three vertices, this allocates all area. Hence, as the approximation to $\vec{h}$, we take

$$\vec{h}_v = -\frac{3\vec{F}_v}{A_v}.$$

If the Evolver is operated in "area normalization" mode, the vertex motions are calculated using this formula. In this mode, the user should set the scale factor, which is the timestep for the evolution, to a constant small enough for the iteration to be a good approximation to the continuous evolution. Using an optimizing scale factor makes the time step too large.

The string model is more amenable to close scrutiny of how the Evolver does motion by mean curvature. In § 2.3 an example was given of grain boundaries evolving in two dimensions. In approximating motion by mean curvature, there are two discretizations that must be made, in space and in time. The space discretization here consists of representing the grain boundaries as a set of line segments, and the time discretization consists of the iteration steps. There is an intermediate stage of discretization, namely, the continuous time evolution of the discrete segments. The equation of motion for this problem has been chosen to approximate the fully continuous problem in the following respect. Two-dimensional grain evolution has the property that the rate of change of area of a grain depends only on the number of its sides. If $\vec{h}$ is the mean curvature of the boundary of a grain G, so that each point on the boundary moves with velocity $-\vec{h}$, the rate of change of area of the grain is

$$\frac{dA}{dt} = \int_{\partial G} -h\, ds = -\int_{\partial G} \frac{d\theta}{ds} ds = -\int_{\partial G} d\theta.$$

The total turning angle around the boundary of a grain is 2π, but each triple vertex contributes a turning angle of $\pi/3$, so that

$$\frac{dA}{dt} = (N-6)\frac{\pi}{3}$$

for an N-sided grain. The space-discrete, time-continuous problem preserves this property. The motion of each vertex is such that the change in area of the grain due to the motion of that vertex is proportional to the turning angle at that vertex (or to the excess turning angle at triple vertices).

A property that is not exactly preserved is that the rate at which work is performed as area is swept is proportional to the rate of length loss,

$$\frac{dW}{dt} = \int_{\partial G} \vec{v} \cdot \vec{F}\, ds = \int_{\partial G} h^2\, ds = -\frac{dL}{dt}.$$

The last equality is the string version of equation (6.1). To preserve this property would require solving a system of linear equations linking together the motions of all the vertices.

The space-discretized evolution is guaranteed to be dissipative, but the time-discretized evolution can suffer from instabilities. Consider a boundary made up of segments of length L zigzagging about a straight midline with small amplitude y. The velocity of a vertex will be $4y/L^2$, and if the time step is Δt, the amplitude will grow if $4y\Delta t/L^2 > 2y$. Hence the maximum timestep allowable is $\Delta t = L^2/2$, or, conversely, the minimum edge length is $L = \sqrt{2\Delta t}$. Exact damping of the zigzag occurs when $\Delta t = L^2/4$.

6.5. Conjugate Gradient

For minimizing a quadratic function, the technique known as the *conjugate gradient method* [Press et al. 1988, § 10.6] is far more efficient than gradient descent. With exact arithmetic, it minimizes an n-dimensional quadratic function in at most n iterations. This method does not follow the gradient downhill, but makes an adjustment using the past history of the minimization.

The Evolver uses the Fletcher–Reeves variant of the conjugate gradient method. At iteration step i, let S_i be the surface, E_i its energy, $\vec{F}_i(v)$ the force at vertex v as described above, and $\vec{h}_i(v)$ the "history vector" of v. Then

$$\vec{h}_i(v) = \vec{F}_i(v) + \gamma \vec{h}_{i-1}(v),$$

where

$$\gamma = \frac{\sum_v \vec{F}_i(v) \cdot \vec{F}_i(v)}{\sum_v \vec{F}_{i-1}(v) \cdot \vec{F}_{i-1}(v)}.$$

For the actual motion, a one-dimensional minimization is performed in the direction of $\vec{h}_i$, using the bracketing method described in § 6.3. It is important that all volumes and constraints be enforced during the one-dimensional minimization, or else the method can go crazy.

The energy function of a surface is not exactly quadratic, but the method can still be applied, and sometimes it yields very good results. But sometimes it's worse than regular iteration. The saddle point of energy in the catenoid example of § 2.1 seems to confuse the conjugate gradient method. With conjugate gradient, in effect, the saddle point is passed at iteration 17 and the area decreases again until iteration 30, when it reaches 6.4486. But at this point further iteration produces no change, and the conjugate gradient mode has to be turned off and on to erase the history vector. Once restarted, another 20 iterations will get the area down to 6.4334. This shows that conjugate gradient mode can work much better than ordinary mode, but it can also have problems.

6.6. Hessian Minimization

Minimization by gradient descent or even by conjugate gradient can take many iterations. A more direct way to try to minimize is to calculate the Hessian matrix of second derivatives of energy and solve for the motion that gives zero gradient. This method is currently implemented only for the case where surface tension is the only energy and where the only constraints are fixed vertices. It assumes that the surface is close enough to a local minimum for the Hessian to be positive definite, which is not always true. When the method works, it can find the minimum energy to 15 decimal places in three or four iterations. But if the Hessian is not positive definite, the method blows up. It is easy to find examples with a saddle point of energy, such as the catenoid example. There are checks in place to ensure that the calculated motion does indeed reduce energy.

6.7. Diffusion

In real soap bubble clusters, air can diffuse across the soap films, driven by pressure differences. Since smaller bubbles have higher curvature and hence higher pressure, they tend to shrink. One can watch a foam evolve over the course of minutes, changing its topology as bubbles disappear. The Evolver can simulate diffusion. If the diffusion mode is on, target volume is transferred across each facet at the start of each iteration cycle, in an amount equal to the area of the facet times the difference in body pressures times the global diffusion constant. The iteration step then corrects the actual volumes to the target volumes and does its normal energy minimization step. Topology changes are not done automatically yet; it is up to the user to carry them out, using the operations described in Section 7.

7. SURFACE OPERATIONS

This section describes the main operations available to the user for modifying a surface, aside from the iteration step described in the previous section.

7.1. Refining

To *refine* a triangulation is to subdivide each facet to create a finer triangulation. The Evolver does this by creating new vertices at the midpoints of edges, which it then uses to subdivide each facet into four new facets, each similar to the original.

The first stage of refining is to subdivide all edges by inserting a midpoint. Hence all facets temporarily have six sides. For an edge on constraints, the midpoint gets the same set of constraints and is projected to them. For an edge on a boundary, the parameters of the midpoint are calculated by projecting the vector from the edge tail to the midpoint back into the parameter space and by adding that to the tail parameters. This avoids averaging parameters of endpoints, which gives bad results when done with boundaries that wrap around themselves, such as circles. In the second stage, each facet is subdivided into four facets by connecting the new midpoints.

Certain attributes of new elements are inherited from the old elements from which they were created. The new facets inherit the surface tension of their parent facets. Fixity, constraints and boundaries are always inherited by offspring of all dimensions. In a quotient space, some, but not all, new edges inherited symmetry group wrapping, so that the surface is correctly embedded.

Refining can change surface area, energy and volumes if there are curved constraints or boundaries. For example, refining the surface of Figure 6 increases its area, because it decreases the gap area.

In seeking the minimum energy, it is best to evolve with a coarse triangulation as far as possible. Each iteration can propagate a position adjustment only one edge at a time, so the finer the triangulation, the longer adjustments take to travel across the surface.

7.2. Equiangulation

Triangulations work best when the facets are as close to equilateral (that is, equiangular) as possible for a given set of vertices. Given a set of vertices, how does one make a triangulation for those vertices that has triangles as nearly as possible equilateral? In the plane, the answer is the Delaunay triangulation, in which the circumcircle of each triangle contains no other vertex [Sibson 1978]. It is almost always unique. It can be constructed by local operations beginning with any triangulation. Consider any edge as the diagonal of the quadrilateral formed by its adjacent triangles. If the angles of the two vertices off of the diagonal add to more than π, the circumcircle criterion is violated, and the diagonal should be switched to form a replacement pair of triangles (Figure 7). When no more switches can be done, we have a Delaunay triangulation.

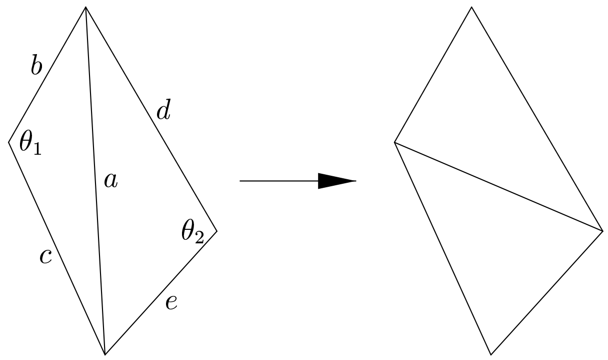

FIGURE 7. The two adjacent triangles on the left violate the equiangulation criterion, since we have $\theta_1 + \theta_2 > \pi$. Equiangulation flips the quadrilateral diagonal, making the triangles more nearly equiangular.

Now suppose that we have a triangulation of a curved surface in space. For any edge with two adjacent facets, we switch the edge to the other diagonal of the skew quadrilateral if the sum of the angles at the off-vertices is more than π. As in Figure 7, let a be the length of the common edge, b and c the lengths of the other sides of one triangle, and d and e the lengths of the other sides of the other triangle. Let θ_1 and θ_2 be the off-angles. By the law of cosines,

$$a^2 = b^2 + c^2 - 2bc\cos\theta_1 = d^2 + e^2 - 2de\cos\theta_2.$$

The condition $\theta_1 + \theta_2 > \pi$ is equivalent to $\cos\theta_1 + \cos\theta_2 < 0$. So we switch if

$$\frac{b^2 + c^2 - a^2}{bc} + \frac{d^2 + e^2 - a^2}{de} < 0.$$

The equiangulation procedure over the whole surface may have to be repeated several times to get complete equiangulation, but almost never more than three or four times. The process is guaranteed to terminate, since a switch reduces the radii of the circumcircles, and a finite set of vertices has a finite number of triangulations.

Equiangulation can have an almost magical effect in improving a triangulation, and I highly recommend its regular use. It may temporarily increase area and change volumes, but the magnitudes of these effects are within the approximation error of using flat facets for a curved surface. Equiangulation was used between the second and third scenes in Figure 1.

7.3. Vertex Averaging

An evolving surface can get into trouble if some of the vertices of the triangulation get too scrunched together, as in the second scene in Figure 3. To get vertices to spread out, one can use *vertex averaging*. For each vertex, this operation computes a new position as the area-weighted average of the centroids of the facets adjoining the vertex. Fixed vertices are not moved, and vertices on boundaries, constraints or singular curves are averaged only with neighboring vertices of the same type. Also, to keep the new surface as close as possible to the old one, volumes on both sides of the surface are preserved. If vertex v is on facets f_i with centroids $\vec{x}_i$, the new position is calculated as

$$\vec{v}_{\text{avg}} = \frac{\sum_i \text{area } f_i \cdot \vec{x}_i}{\sum_i \text{area } f_i}.$$

The volume on one side of all the facets around the vertex calculated as a cone from the vertex is

$$V = \sum_{\text{facets } f} \vec{v} \cdot \vec{N}_f,$$

where $\vec{N}_f$ is the facet normal representing its area. The total normal $\vec{N}$ is

$$\vec{N} = \sum_{\text{facets } f} \vec{N}_f.$$

To preserve volume, we subtract a multiple λ of the total normal from the average position:

$$(\vec{v}_{\text{avg}} - \lambda\vec{N}) \cdot \vec{N} = \vec{v} \cdot \vec{N},$$

so

$$\lambda = \frac{\vec{v}_{\text{avg}} \cdot \vec{N} - \vec{v} \cdot \vec{N}}{\vec{N} \cdot \vec{N}}.$$

Then the new vertex position is

$$\vec{v}_{\text{new}} = (\vec{v}_{\text{avg}} - \lambda\vec{N}).$$

Constrained vertices are then projected to their constraints.

Vertex averaging may slightly increase area, but this is usually offset by its benefits. It is useful in getting the vertices spread out evenly. Evolution can be awkward when facets are of very different sizes, since the same scale factor applies to the whole surface.

7.4. Notching Edges

A surface can be locally highly curved, resulting in facets forming pronounced ridges along edges. One way to selectively refine the surface is to refine only around those edges whose adjacent facets are too far from parallel, putting a notch in the edge to make it more saddle-shaped. There is a command that lets the user do this with a cutoff angle of his choosing. The refinement is actually done by subdividing each adjacent facet by putting a new vertex in the center. Equiangulation then completes the process. Formerly, the Evolver did notching by just subdividing the offending edges, but that tended to create lots of long skinny triangles and not always help matters. The new method seems to work better.

7.5. Edge and Facet Operations

One way to improve a triangulation is to simply eliminate all edges that have become too short.

This operation is known as *tiny-edge weeding*. Every edge shorter than a user-set cutoff length that can legitimately be removed is deleted by identifying its endpoints.

Sometimes there are very skinny triangles that should be eliminated, but which don't have a short edge to be found by tiny-edge weeding. One can then use *area weeding*, an operation that removes triangles whose area is smaller than some cutoff, by finding the shortest edge of the triangle and eliminating it by the same process as regular tiny-edge removal.

There is a command that will bisect all edges longer than a user-chosen length. All facets adjoining the edges are also subdivided into pairs of facets. If the new edges are still longer than the cutoff length, they are not further subdivided. It is suggested that this step be followed by equiangulation.

Histograms of edge lengths and facet areas can be displayed in conjunction with any of these commands.

7.6. Annealing, or Jiggling

Sometimes it may be desirable to perturb the surface to get it off a metastable position. Both random and user-definable perturbations are possible. Because of its similarity to the thermal perturbations responsible for annealing in metals, the characteristic magnitude of the perturbation is called *temperature*.

Under a random permutation, or *jiggle*, each coordinate of each nonfixed vertex is moved by $\delta x = TLg$, where g is a random value from the standard Gaussian distribution (calculated from the sum of five random values from the uniform distribution on [0,1]), T is the current temperature, and L is a characteristic length that starts as the diameter of the surface and is cut in half at each refinement.

A *long jiggle* is a sinusoidal displacement of each vertex v by $\vec{A}\sin(\vec{v}\cdot\vec{w}+\psi)$. The amplitude $\vec{A}$, the wave vector $\vec{w}$ and the phase ψ may be specified by the user or be chosen at random.

7.7. Popping Edges or Vertices

The Evolver does not change the topology of a surface on its own, but there are many times when a naturally evolving surface will need to change its topology. A neck might pinch out in a catenoid whose boundary rings are too far apart, or two growing metal grains might meet. Fortunately, the types of singularities possible in soap-film surfaces in three-dimensional space were classified in [Taylor 1976] for uniform surface tension. Three surfaces may meet along a curve, or four triple curves may meet at a point. The Evolver has procedures known as *edge popping* and *vertex popping* to detect improper singularities and to reduce them to proper types. These routines are designed only for surfaces with uniform surface tension. Many more types of singularities are possible if the different component surfaces meeting at a singularity have different surface tensions.

Edge popping looks for edges that are not fixed, are not on boundaries or constraints, and lie on more than three facets. When found, such an edge is split longitudinally, with a new facet in between. The two old facets with the smallest dihedral angle between them are attached to the new edge. This is repeated until only three facets are on the original edge. Each split is propagated along the multiple junction line as far as possible. If it is impossible to propagate the split beyond either endpoint, the edge is subdivided to provide a vertex that can be split.

Vertex popping assumes that each edge belongs to at most three facets, so it should be preceded by edge popping. The facet and edge structure around each vertex is analyzed to find which vertices have the wrong topology. This is done by looking at the *link* of the vertex: the intersection of the facets and edges containing the vertex with the surface of a small sphere around the vertex. The numbers of sides of the cells in the link are counted. A simple plane vertex has two cells of one side each. A triple-edge vertex has three cells of two sides each. A tetrahedral point has four cells with three sides each. Any other configuration is popped. The popping is done by replacing the vertex with a hollow formed by truncating each cell-cone except the cell with the largest solid angle. If the link is disconnected, the solid angles of all the cells will add up to over 4π. Then the vertex is duplicated and the different components are assigned to different vertices. This lets necks shrink to zero thickness and pull apart.

In the string model, vertices with more than three edges are popped by finding the pair of edges making the least angle, then pulling them out a short distance with a new vertex and joining the new and old vertex with a short edge. This is repeated until the original vertex has only three edges.

Improper vertices may exist in the original data file, or they may be introduced by short-edge elimination. For example, the pinching neck in a catenoid must have all the short edges around its waist eliminated to pinch the waist down to one vertex, which can then be popped. The other operations described in this section (refining, vertex averaging, equiangulation, notching) do not change the global topology and so do not introduce improper vertices. Improper vertices are not automatically detected unless the autopop feature is on.

7.8. Zooming

Sometimes the detail of a surface may require a closer look. The graphics display can magnify a surface, but that doesn't change the triangulation to follow the detail. There can be cases, as when a boundary wire passes through a soap film, where the detail around a point is on a scale 100,000 times smaller than the whole surface [Brakke 1992a]. For this reason the Evolver allows the user to zoom in on a vertex, throwing away the rest of the surface to save memory and time and to keep all triangle sizes reasonably close together.

The user specifies the vertex that the program should zoom in on and a cutoff distance. All vertices beyond the cutoff distance from the given vertex are deleted. Then all edges and facets containing any deleted vertices are deleted. Any remaining edge from which a facet was deleted is made fixed in order to anchor the cut edges of the surface.

8. USER INTERFACE DETAILS

This section describes the Evolver's user interface, including the initial data file, the command mechanism and the graphics interface.

8.1. The Initial Data File

The initial configuration of a surface is read from a text file referred to as the *data file*. The data file has five sections: general definitions, vertices, edges, faces and bodies. The catenoid data file `cat.fe` is presented in the sidebar on the next page, to give the flavor. This file is slightly atypical in that none of the vertices are given directly by their coordinates. The `.fe` filename extension is a relic of early versions of the Evolver in which facet-edges had to be explicitly listed in the data file; I continue to use it out of habit to identify data files.

The data-file syntax provides several features for flexibility and ease of use. Simple macros can be defined to do text substitution. Compound expressions can be used wherever a real number or a formula is expected. Normal arithmetic and standard functions are available. For functions that are evaluated during runtime, such as constraints and quantities, expressions are stored as syntax trees that are interpreted when the expression needs to be evaluated. If interpretation is too slow, user-defined functions may be written in C and compiled into the Evolver. Named variables, called *adjustable parameters*, may be declared and used in runtime expressions, and changed interactively at runtime. They are useful for moving constraints and boundaries around, modifying the metric or contact angles, and so forth.

The definitions section contains data not pertaining to particular geometric elements, such as

- declarations and initial values for adjustable parameters;
- the dimension of the surface and the dimension of the ambient space containing the surface;
- the specification of a quotient space, or, for a flat torus domain, the vectors defining the fundamental parallelepiped;
- Riemannian metric tensor components;
- for a crystalline surface energy, the name of the Wulff vector file;
- constraint function formulas, together with energy and volume integrands for edges on constraints;
- boundary definitions via formulas of coordinates in term of parameters;
- quantity integrands, with target values for constrained quantities;
- initial values for the gravitational constant, the diffusion constant, the weighting factor for the squared mean curvature in the energy, the Gaussian integration order, and linear or quadratic mode.

None of these items are required. If an item is missing, the feature is not used, or is used with a natural default value.

```
// cat.fe: Evolver datafile for catenoid.

// ring radius and height
// adjustable at runtime
PARAMETER radius = 1
PARAMETER height = 0.55

// upper ring, parametrized by p1
boundary 1 parameters 1
x1:  radius * cos(p1)
x2:  radius * sin(p1)
x3:  height

//   lower ring
boundary 2 parameters 1
x1:  radius * cos(p1)
x2:  radius * sin(p1)
x3:  -height

vertices /* second column = value of p1 */
1    0*pi/3  boundary 1   fixed
2    1*pi/3  boundary 1   fixed
3    2*pi/3  boundary 1   fixed
4    3*pi/3  boundary 1   fixed
5    4*pi/3  boundary 1   fixed
6    5*pi/3  boundary 1   fixed
7    0*pi/3  boundary 2   fixed
8    1*pi/3  boundary 2   fixed
9    2*pi/3  boundary 2   fixed
10   3*pi/3  boundary 2   fixed
11   4*pi/3  boundary 2   fixed
12   5*pi/3  boundary 2   fixed

edges  /* given by endpoint vertices */
1    1  2  boundary 1   fixed
2    2  3  boundary 1   fixed
3    3  4  boundary 1   fixed
4    4  5  boundary 1   fixed
5    5  6  boundary 1   fixed
6    6  1  boundary 1   fixed
7    7  8  boundary 2   fixed
8    8  9  boundary 2   fixed
9    9  10 boundary 2   fixed
10   10 11 boundary 2   fixed
11   11 12 boundary 2   fixed
12   12 7  boundary 2   fixed
13   1  7
14   2  8
15   3  9
16   4  10
17   5  11
18   6  12

faces  /* given by oriented edge list */
1   1 14 -7 -13
2   2 15 -8 -14
3   3 16 -9 -15
4   4 17 -10 -16
5   5 18 -11 -17
6   6 13 -12 -18
```

The vertices section lists the vertices, one per line. Each vertex is numbered for later reference, and is defined by its coordinates (or boundary parameters), which constraints or boundaries it is on, and whether it is fixed.

The edges section lists the edges, one per line, also numbered for reference. Each edge is defined by its tail and head vertex numbers, the constraints or boundaries it is on, which quantities it contributes to, and whether it is fixed. If a quotient space is being used, the group element for wrapping the head vertex to the proper place with respect to the tail vertex is also given.

The faces section lists polygons forming the initial surface. Each polygon is given by its edge numbers, in order around its circumference. Edges traversed in opposite direction from that given in the edges section are given as negative numbers. The polygons need not be planar, and they need not be triangles (which is why the section is called "faces" instead of "facets"). The Evolver will immediately triangulate nontriangular faces by putting a new vertex at the average position of the original vertices and by putting in edges from the new vertex to each original vertex. Each face may be on constraints, on boundaries, or be fixed. It may be given a specific surface tension; the default is 1.0. It may be deemed to contribute to certain quantity integrals, and to have certain surface integrands contribute to the total energy.

In the bodies section, each body is defined by listing its bounding faces by number, the number being negative if the orientation of the face in the face list has an inward normal. There may be any number of faces in any order. Faces do not have to completely enclose a body; they are used to compute volume and other integrals, and if certain faces are not needed for that, they may be omitted. A body may be declared to have a fixed volume of a certain value. The actual initial volume need not be that exact value; the volume will be adjusted during the iteration process. A body may also be given a density, which will cause the total energy to include the gravitational potential energy of the body with that density.

8.2. Command Interface

The user command interface is built on a simple terminal-type model for maximum portability. The main prompt is "`Enter command:`". There are two types of commands. The first consists of one letter occasionally followed by a number; the second is an embryonic query language. Currently, queries are supported that list, display, refine or delete elements by various criteria. Commands may be read from a file with the command "`read` *filename*". The output of any command can be piped to a system command. Commands that change the surface or the model will cause energies and volumes to be recalculated. Commands can be logged to a file for later repetition with the `read` command.

8.3. Graphics

It is possible to run the Surface Evolver without any graphics. But being able see the surface is always nice, and often essential to understanding. Unfortunately, every computer system has its own way of displaying graphics, and there is no universal standard. The Evolver isolates the system-dependent graphics to drawing line segments and triangles in two dimensions. This cuts down the effort in porting to new systems. The Evolver's main graphics routine calculates the triangles to be displayed and calls the device-dependent subroutine to do the display. The device could be a screen display or a graphics output file writer.

Two classes of graphics devices are provided for: those that can do their own viewing transformations and hidden surface removal, and those that can't. The former are simply provided with a list of triangles with vertex coordinates in three dimensions. For the latter, the Evolver keeps an internal viewing transformation matrix, sorts the transformed triangles from back to front, and feeds them to the display routine (painter's algorithm). The sorting algorithm will not subdivide intersecting triangles. If two triangles overlap, it will just find one point in the overlap and compare depths there. This can lead to some strange-looking displays for strange surfaces, but it works well for the types of surfaces for which the Evolver is designed.

My favorite graphics system is the geomview program on Iris workstations. Geomview is an interactive viewer that lets the user rotate, translate, zoom, and otherwise move the surface by dragging a mouse cursor over the window. A high-end Iris workstation can light, shade and smoothly rotate a surface consisting of several thousand triangles. The Evolver is interfaced with geomview so that the display is automatically updated whenever the surface changes. Geomview was written at The Geometry Center and is freely available (see "Software Availability" at the end of this article).

Other types of screen displays do not have such fancy view control as geomview. Instead, there is a terminal-type command interface that lets the user control the viewing angle and size of the display. This viewing transformation is also used for the graphics output files.

There are several graphics output file formats, most notably PostScript. The surfaces illustrating this article were done with the PostScript format. There are also formats that list transformed or untransformed triangles as text, suitable for input to other programs.

8.4. Other Commands

It is possible to reset the values of many parameters during runtime, including the gravitational constant, body volumes, constrained quantity targets, the diffusion constant and the user-defined variables used in formulas for constraints, boundaries, quantities and metrics.

The current surface can be dumped to a text file in the same format as the data file. This is the only way to save a file; there is no binary save format. The text format has the advantages that it is portable, editable and not too much larger than a binary file would be.

After minimizing energy at several levels of refinement, it is possible to extrapolate the energy to an infinitely fine refinement. The extrapolation uses the final energies of three successive refinements and assumes a power law approach to the ultimate minimum.

9. APPLICATION: THE HOPF CONE CONJECTURE

In this section I present an example of a conjecture that was settled (negatively) through the use of the Evolver.

In constrast to the situation in $\mathbf{R}^3$, the classification of area-minimizing hypersurface cones in $\mathbf{R}^4$ is unknown. Frank Morgan once conjectured that a certain cone in $\mathbf{R}^4$ is absolutely area minimizing [Morgan 1986, p. 1278]. Use of the Evolver showed that this is not the case: a comparison surface was found that has less area for the same boundary. This example illustrates the use of a Riemannian metric to permit a surface in three dimensions to represent one in four dimensions, by projecting out a symmetry.

Morgan's cone is based on the Hopf fibration of the 3-sphere S^3. Let S^3 be parametrized by $\alpha \in [0, \pi/2]$, $\beta \in [0, 2\pi]$ and $\gamma \in [0, 2\pi]$, by means of the formulas

$$x_1 = \cos\alpha\cos\beta, \quad x_2 = \cos\alpha\sin\beta,$$
$$x_3 = \sin\alpha\cos\gamma, \quad x_4 = \sin\alpha\sin\gamma,$$

where (x_1, x_2, x_3, x_4) are the Euclidean coordinates of a point in $S^3 \subset \mathbf{R}^4$. The boundary of Morgan's Hopf cone consists of the three surfaces

$$\begin{aligned} \beta - \gamma &= 0 \quad (\text{mod } 2\pi),\\ \beta - \gamma &= \tfrac{2}{3}\pi \quad (\text{mod } 2\pi),\\ \beta - \gamma &= \tfrac{4}{3}\pi \quad (\text{mod } 2\pi). \end{aligned}$$

These three surfaces have zero mean curvature and meet at 120-degree angles along the two orthogonal circles $x_1^2 + x_2^2 = 1$ and $x_3^2 + x_4^2 = 1$.

Theorem. Morgan's Hopf cone is not absolutely area minimizing.

Proof of . The idea of the proof is to take the quotient space of $\mathbf{R}^4$ modulo the Hopf fibers S^1, the result being $\mathbf{R}^3$ with a metric such that the area of a surface in $\mathbf{R}^3$ is the same as the 3-area of the lift of the surface back into $\mathbf{R}^4$. The metric turns out to have a natural interpretation as a cone space, leading to a simple counterexample that can be verified by the Evolver.

The metric on $\mathbf{R}^4$ in Hopf spherical coordinates $(r, \alpha, \beta, \gamma)$ is

$$ds^2 = dr^2 + r^2 d\alpha^2 + r^2\cos^2\alpha\, d\beta^2 + r^2\sin^2\alpha\, d\gamma^2.$$

The coordinates of the quotient space $\mathbf{R}^3$ will be (r, α, θ), where $\theta = \beta - \gamma$. The orthogonally projected metric is

$$ds^2 = dr^2 + r^2 d\alpha^2 + r^2\sin^2\alpha\cos^2\alpha\, d\theta^2.$$

The lift of a point in $\mathbf{R}^3$ is a circle of circumference $2\pi r$. Hence we can make the 3-area of the lift of a surface have 2π times the 2-area in $\mathbf{R}^3$ by multiplying the linear metric ds by $\sqrt{r}$, giving an effective metric of

$$ds^2 = r\, dr^2 + r^3 d\alpha^2 + r^3\sin^2\alpha\cos^2\alpha\, d\theta^2.$$

This can be made to look more like the ordinary spherical coordinate metric by using coordinates $\rho = \frac{1}{2}r^{3/2}$ and $\varphi = 2\alpha$. Then

$$ds^2 = \tfrac{16}{9}d\rho^2 + \rho^2 d\varphi^2 + \rho^2\sin^2\varphi\, d\theta^2.$$

This is the Euclidean spherical coordinate metric, except for the factor $\frac{16}{9} > 1$, which makes $\mathbf{R}^3$ into a cone space.

Morgan's Hopf cone projects to three planes that meet at 120 degrees, a configuration known to be absolutely area minimizing in the standard metric of $\mathbf{R}^3$. Its area inside the unit sphere (in the Hopf metric) is 2π. However, the cone factor makes it more expensive for a surface to go radially inward than to go sideways. In a two-dimensional cone, it is easily seen (by unrolling the cone) that geodesics avoid the origin. A similar phenomenon happens here. To improve on Hopf's cone, one starts by deforming the three planes by pushing the point at the origin out toward one of the boundaries. When fed into the Surface Evolver, this configuration evolves to the surface of Figure 8, which has

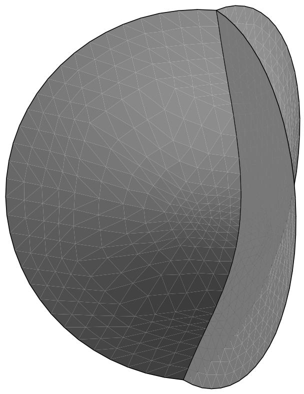

FIGURE 8. This surface is a counterexample to Morgan's Hopf cone conjecture. The three outer edges are equally spaced half–great-circles on the unit sphere. Note how the surface avoids the center of the sphere.

an area of 6.14, less than that of Morgan's Hopf cone. The numerical errors in this area are due to the numerical integration used to calculate the area of the facets and the gap between the curved boundary and the facets. Both of these can easily be estimated to be less than the improvement. □

The Hopf cone that projects to the tetrahedral cone is also not minimizing. The Evolver gives 7.44 for the area of the comparison surface in Figure 9, while the area of the cone is $4\cos^{-1}(-1/\sqrt{3}) =$

7.6425. The only other minimizing cone in standard $\mathbf{R}^3$, the flat plane, also deforms to avoid the origin. Thus none of the area-minimizing cones in standard $\mathbf{R}^3$ lifts via the Hopf fibration to a minimizing cone in $\mathbf{R}^4$.

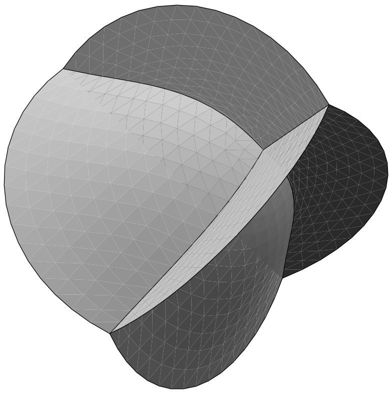

FIGURE 9. Comparison surface for the tetrahedral Hopf cone. The three outer edges lie on the unit sphere. Again, the surface avoids the center of the sphere.

10. FUTURE DIRECTIONS

The Surface Evolver is under continual development. I welcome suggestions from users for new features. If they are reasonable, they will be added as time permits.

Some mathematical questions and programming projects for the future are the following:

- How close in various senses is an Evolver minimal surface to the true smooth minimal surface for a given problem?
- The Evolver gives an upper bound for the area of a minimal surface. The technique of *calibrations*, which generalizes the min-cut-max-flow duality from network theory, can give lower bounds. A near-minimal surface should be able to generate a near-maximal calibration. Hence a goal is to have the Evolver generate such calibrations.
- How close is an Evolver evolution by mean curvature to an ideal smooth evolution? Given an initial smooth surface, is it possible to construct an Evolver approximation that stays close to the ideal evolution? A more permissive notion of approximation would say that for each Evolver evolution there is an ideal smooth evolution that stays near it.
- The current method of approximation to motion by mean curvature needs further investigation. The gradient of energy is a covector, and motion is a vector. The conversion from covector to vector requires a metric or inner product. The inner product used by the Evolver is the Euclidean inner product at vertices, weighted by the vertex star area. Other inner products are possible and may have desirable properties.
- Instabilities of the type described in § 6.4 often limit the size of the time step in an evolution. These instabilities need to be understood and methods have to be developed to speed evolution.
- Automatic triangulation management needs to be extended. Currently, users have to monitor the surface triangulation closely and intervene manually when it gets fouled up. I hope to be able to have any initial surface evolve for any length of time without user intervention, as is now the case for string evolution as described in § 2.3.
- An interactive graphical interface could let users select with a mouse the geometric elements (vertices, edges, etc.) they wish to work with, and it could be an interactive tool for the design of initial surfaces and data files.

REFERENCES

[Almgren 1982] F. Almgren, "Minimal surface forms", *The Mathematical Intelligencer* **4** (1982), 164–172.

[Almgren and Taylor 1976] F. Almgren and J. E. Taylor, "The geometry of soap films and soap bubbles", *Scientific American* (July 1976), 82–93.

[Brakke 1977] K. A. Brakke, *The Motion of a Surface by Its Mean Curvature*, Princeton University Press, Princeton, NJ, 1977.

[Brakke 1991a] K. A. Brakke, *Surface Evolver Manual*, Research Report GCG 31, The Geometry Center. See "Software Availability" below for information on how to obtain it.

[Brakke 1991b] K. A. Brakke, "The opaque cube problem", in *Computing Optimal Geometries* (video and booklet, edited by J. E. Taylor), American Mathematical Society, Providence, RI, 1991.

[Brakke 1992a] K. A. Brakke, "Minimal surfaces, corners, and wires", *Journal of Geometric Analysis* **2** (1992), 11–36.

[Brakke 1992b] K. A. Brakke, "Grain growth with the Surface Evolver", in *Video Proceedings of the Workshop on Computational Crystal Growing* (edited by J. E. Taylor), American Mathematical Society, Providence, RI, 1992.

[Callahan et al. 1991] M. Callahan, P. Concus and R. Finn, "Energy minimizing capillary surfaces for exotic containers", in *Computing Optimal Geometries* (video and booklet, edited by J. E. Taylor), American Mathematical Society, Providence, RI, 1991.

[Dobkin and Laszlo 1987] D. Dobkin and M. Laszlo, "Primitives for the manipulation of three-dimensional subdivisions", Technical Report CS-TR-089-87, Department of Computer Science, Princeton University, Princeton, NJ, 1987.

[Mittelmann and Hornung] H. Mittelmann and U. Hornung, "Symmetric capillary surfaces in a cube" (preprint).

[Morgan 1986] F. Morgan, "Harnack-type mass bounds and Bernstein theorems for area-minimizing flat chains modulo ν", *Comm. Part. Diff. Eq.* **11** (1986), 1257–1283.

[Morgan] F. Morgan, "Surfaces minimizing area plus length of singular curves" (preprint).

[Press et al. 1988] W. Press, B. Flannery, S. Teukolsky and W. Vetterling, *Numerical Recipes in C*, Cambridge University Press, New York, 1988.

[Racz et al.] L. M. Racz, J. Szekely and K. A. Brakke, "On some meniscus problems in materials processing" (submitted to *Trans. Iron and Steel Inst.*).

[Sibson 1978] R. Sibson, "Locally equiangular triangulations", *Comput. J.* **21** (1978), 243–245.

[Taylor 1976] J. E. Taylor, "The structure of singularities in soap-bubble-like and soap-film-like minimal surfaces", *Ann. Math.* **103** (1976), 489–539.

[Taylor 1983] J. E. Taylor, "Constructing crystalline minimal surfaces", pp. 271–288, in *Seminar on Minimal Submanifolds* (edited by E. Bombieri), Annals of Math. Studies **105**, Princeton University Press, Princeton, NJ, 1983.

[Taylor 1988] J. E. Taylor, "Constructions and conjectures in crystalline nondifferentiable geometry", *Proceedings of the Conference in Differential Geometry, Rio de Janeiro, 1988*, Pittman Publishing Ltd.

[Tegart 1991] J. Tegart, "Three-dimensional fluid interfaces in cylindrical containers", AIAA paper AIAA-91-2174, 27th Joint Propulsion Conference, Sacramento, CA, June 1991.

[Thompson 1987] W. Thompson (Lord Kelvin), "On the division of space with minimum possible error", *Acta Math.* **11** (1887), 121–134.

SOFTWARE AVAILABILITY

The Surface Evolver program is available free of charge. It is written in C, in such a way as to be portable between systems. So far it has been ported to Sun, Iris, NeXT, Xenix and MS-DOS systems. The major effort in porting to a new system consists of writing a screen graphics interface. However, this is fairly simple, since the system-dependent routines need only display triangles. The program can also be run without any screen graphics, which makes it possible to run remotely.

A package containing source code, manual and sample data files is available by anonymous ftp in the file `pub/evolver.tar.Z` on the machine `geom.umn.edu`. A separate file, `pub/evolver.next.tar.Z`, contains a version for the NeXT computer, including Interface Builder files. The Evolver is also available on floppy disk from the author. The manual in TEX dvi format is included in the ftp archive. A hardcopy version of the manual can be requested separately from the author, or directly from The Geometry Center, 1300 South Second Street, Minneapolis, MN 54554.

Geomview is likewise available from `geom.umn.edu`, in directory `pub/geomview`.

Kenneth A. Brakke, Mathematics Department, Susquehanna University, Selinsgrove, PA 17870
1992–1993 academic year: Geometry Center, 1300 So. Second Avenue, Minneapolis, MN 55454
(brakke@geom.umn.edu)

Received November 5, 1991; revised May 13, 1992