Using Symmetry Features of the Surface Evolver to Study Foams

K. A. Brakke[1] and J. M. Sullivan[2]

[1] Mathematics Dept., Susquehanna University, Selinsgrove, PA, USA
[2] Mathematics Dept., University of Minnesota, Minneapolis, MN, USA

Summary. This paper describes the use of various symmetry features, including periodic boundary conditions, mirror boundaries, and rotational symmetry, in the Evolver. As a test case, we use these features to study foams, in particular the equal-volume foams of Kelvin and Weaire–Phelan. To compute the shape and energy of one of these compound surfaces, it is most efficient to work with only the smallest possible fundamental domain.

1. Introduction

The Surface Evolver [2] is a software package[1] for interactive study of curves and surfaces shaped by energy minimization. In its most basic mode, the energy in question is surface tension, and the resulting shapes are minimal (or constant-mean-curvature) surfaces, mathematical models of soap films and soap bubbles. The Evolver works with triangulated surfaces, and can easily handle those with complicated topology, like the triple junctions and Plateau singularities found in real soap films.

The Evolver has been extended to deal with many other energies, including physical terms like gravity, and also mathematical energies like the elastic bending energy of Willmore surfaces [4] and various Möbius-invariant knot energies [7].

Here we will discuss extensions in a different direction, allowing the Evolver to compute a symmetric surface using only a single fundamental domain. For instance, for a triply periodic surface, we can look at a fundamental domain for the lattice translations, imposing periodic boundary conditions. In fact, we prefer to think of describing a surface in the quotient torus, which is essentially equivalent.

[1] Available at the URL `http://www.geom.umn.edu/software/locate/evolver/` . Our examples may require version `1.99w` or later, run with the `-q` option.

Some further symmetries can also be modeled easily. Any minimal surface with a straight-line boundary can automatically be extended, by 180° rotation around this line, to a surface with two-fold symmetry. Similarly, a minimal surface with a free boundary curve in a plane can be extended by mirror reflection. Thus, surfaces with mirror symmetry or two-fold rotation symmetry can be modeled naturally in the Evolver simply by one fundamental piece with boundaries in mirror planes or along rotation lines. For mirror planes, the Evolver will automatically, by area-minimization, give a surface meeting this plane perpendicularly. It can also deal with a soap-film triple junction where one sheet is in the mirror plane and the others meet it at 120° angles.

For more general symmetries, we use the Evolver's general purpose symmetry-group feature: the user can write code to deal with any group. This has been used in the past to try to model periodic surfaces in hyperbolic space.[2] There are, however, certain special difficulties when the group in question does not act freely. We have now overcome these problems, and have implemented the group of rotations of arbitrary order about a single axis, as well as the cubic point group, as Evolver symmetry groups.

As an example illustrating all these symmetry features, we will look at two foams, or partitions of space into cells with (locally) minimal interface area, and see how to compute them using successively more and more symmetry (and thus smaller fundamental domains). Our newly implemented rotational symmetries will also be useful for a minimax sphere eversion [3] where a halfway surface with rotational symmetry (but no mirrors) is evolved to minimize its elastic bending energy.

2. The foams of Kelvin and Weaire–Phelan

Lord Kelvin [5] posed in 1887 the problem of partitioning space into equal-volume cells, while using the least interface area. The solution will be a foam, like an infinite cluster of soap bubbles. Kelvin's proposed solution (Fig. 2.1) was a relaxation of the truncated octahedra which tile space in a body-centered cubic lattice. Weaire and Phelan [8] discovered in 1994 a new equal-volume foam (Fig. 2.2). Using the Evolver, they concluded numerically that it used less area than Kelvin's foam, and thus is a new candidate for the optimal partition.

Kusner and Sullivan [6] discuss the construction and symmetries of both of these foams, and outline a mathematical proof that in fact Weaire–Phelan does beat Kelvin. However, neither foam can be described by explicit formulas, and we do not even know an existence proof for Weaire–Phelan. Thus, it is still important to be able to use the Evolver to efficiently calculate approximations to both these foams.

[2] See the file `khyp.c` in the Evolver source code.

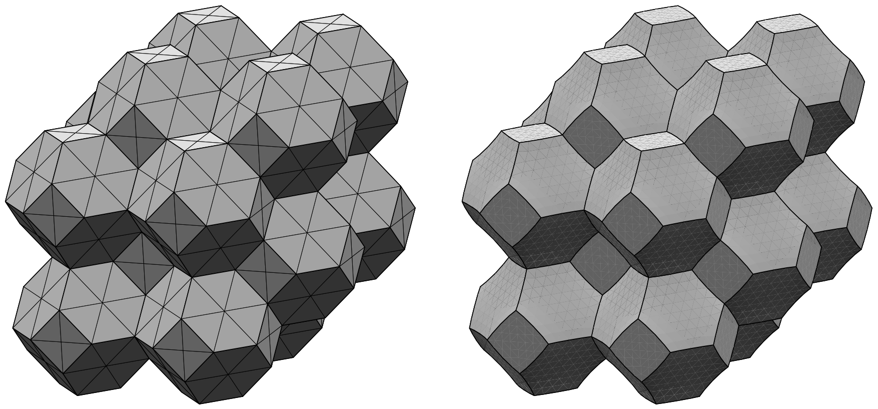

Fig. 2.1. The Kelvin foam has congruent cells tiling space in a BCC lattice. Here we see two cells from a cubic torus, each replicated eight times in a $2 \times 2 \times 2$ array. The original polyhedral cells (left) will relax to a foam (right) satisfying the Plateau rules, with somewhat lower area. The Evolver divides initial polygons into triangles (left); the new edges (diagonals) drawn are exactly the two-fold axes of symmetry of the foam – note that they continue in infinite straight lines square-to-square or hexagon-to-hexagon. The relaxed foam (right) has a much finer triangulation, but only the triple junctions are drawn. All the symmetries are still present.

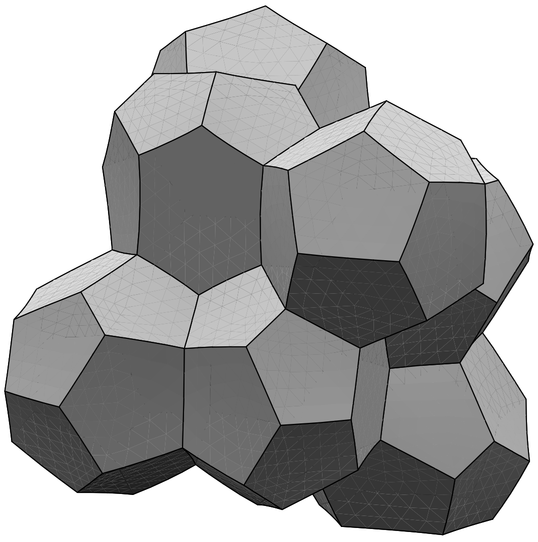

Fig. 2.2. Here we see a relaxed Weaire–Phelan foam. These eight cells (two dodecahedra and six 14-hedra) fill space when replicated in a cubic lattice. In front at the upper right is a dodecahedron; the other visible cells are 14-hedra, which occur in columns along the three coordinate directions. (For instance, the two cells in the back right are in a vertical column, and the two at the bottom left in a horizontal column.) Each 14-hedron has two opposite, parallel hexagonal faces (which separate cells within each of these columns) and twelve pentagonal faces.

Although all cells in a given foam will have equal volumes, we may want to compare different foams with different choices for this volume, so we define the scale-invariant *cost* of a foam to be $\mu := A^3/V^2$, where V is the volume of each cell, and A is the average interface area per cell (which is half the surface area of a typical cell, since interfaces are shared). This cost is what we will ask the Evolver to report, while it minimizes the area. Note that it is important to constrain the cell volumes to be equal, as there are certainly more efficient packings of cells of varying sizes (though it is not even clear how to properly measure efficiency then).

All cells in Kelvin's foam are congruent, so they have equal pressure, and the interfaces are minimal surfaces despite the volume constraints. The Weaire–Phelan foam, on the other hand, uses two shapes of cells (dodecahedra, and 14-hedra of a particular type). The pressures in these cells are different, so that the interfaces between them have nonzero constant mean curvature. Of course, in both cases, the cells in the foams meet according to Plateau's rules for soap bubbles: along each singular line, three cells meet at 120° angles, and at each vertex, four cells come together tetrahedrally.

As discussed in [6], the cell combinatorics implied by these rules for foams are the same as those found in generic Voronoi partitions. (Given a list of points, called sites, in space, we partition space into the corresponding Voronoi cells by allocating each other point in space to the cell of its nearest site.) Thus it is not surprising that both foams are relaxations of Voronoi partitions for simple point sets. Starting with sites in a body-centered cubic (BCC) lattice, the Voronoi cells are the truncated octahedra which relax to the Kelvin foam. If we use the BCC lattice points together with half the vertices of Kelvin as our Voronoi sites, we obtain a partition which relaxes to Weaire–Phelan. These partitions will give our first Evolver models of the foams.

3. Triply periodic foams in the Evolver

In general, surfaces are presented to the Evolver in a datafile which lists first the vertices with their coordinates in space, then the edges (which connect pairs of vertices), and finally the faces (specified by the rings of oriented edges around their boundaries). Optionally, bodies with fixed or varying volume can be specified, each given by the set of oriented faces enclosing it.

To implement a triply periodic surface or foam in the Evolver, we use the `torus` mode of the program. We specify three `periods` or vectors generating a lattice, and the surface then lives in the quotient torus of $\mathbb{R}^3$ by this lattice. Vertices are free to move throughout the torus, and are not restricted to any particular fundamental domain. (One could imagine other models, where a fundamental domain would be fixed. Then certain vertices would be constrained to the boundary; these would be replicated on the opposite boundary to give periodic conditions. But such a scheme is needlessly restrictive.)

For each edge in `torus` mode, we must specify a `wrap` value, which tells which way around the torus the edge goes, or how it crosses between fundamental domains when lifted to $\mathbb{R}^3$. In the datafile, the wrap is given as a triple of symbols `-`, `*`, or `+`, representing the sum of -1, 0, or 1 times the three period vectors, respectively. If an edge connects vertices a and b (as listed with coordinates in $\mathbb{R}^3$) and its wrap vector is given as w, then this edge in the torus is one that has a lift connecting a to $b+w$ in $\mathbb{R}^3$. Faces and bodies are specified as usual in the datafile, with the requirement that the net wrap around the boundary of any face will be zero.

An Evolver datafile[3] for the Kelvin foam is shown in Fig. 3.1. This foam lives in the cubic torus $\mathbb{R}^3/4\mathbb{Z}^3$. The twelve vertices given are those from two opposite hexagonal faces (numbers `30` and `80`) of the Voronoi cell around the origin. (Of course all other vertices of the foam are lattice translates of these.) Those two faces can be specified with edges of wrap zero, but the other faces all involve nonzero wraps. Here, body `1` is the cell around the origin, while the other body is its translate by $(2, 2, 2)$.

When in `torus` mode, bodies are specified by oriented faces with no notion of their relative positions in the torus. Because of this, the Evolver's computation of a body volume might be off by some multiple of 1/6 the total torus volume. (Note for instance that the list of faces for body `1` in Fig. 3.1 includes `12` and `-12`. These are in fact opposite square faces of the cell, at different heights and should thus contribute differently to the volume, obtained by integrating $z\,dx\,dy$. But the Evolver does not know their relative positions, and we could just as well have left these two faces off the list.) To avoid these errors it is important to give the Evolver the approximate starting value for each body volume, as we do. However, the end of this datafile tells the Evolver to unfix one of the volumes. This is because, in a torus filled with bodies of fixed volumes, these volume constraints are in fact linearly dependent. Here the two constraints are equivalent, and we must keep one, removing the other.

If we run the Evolver on this datafile, we see initially the foam on the left in Fig. 2.1, with `cost` $\mu \approx 18.7653$. The command `transform_expr "abc"` at the end of the datafile tells the Evolver to show eight copies of the surface, translated by (sums of) the torus period vectors. Even though the Evolver has, as always, introduced new vertices in the centers of the hexagons and squares to get a triangulated surface (whose edges are drawn in the left figure), there is no freedom to improve this foam until we refine the triangulation. If we run the commands `r; g 10; r; g 10` to refine and iterate, we obtain the relaxed foam on the right in Fig. 2.1, with `cost` $\mu \approx$ 18.6824, modeled in the Evolver with slightly over one thousand triangles.

To check that we are now essentially at the minimum energy for this level of refinement, we can use the `hessian` command (first turning on the

[3] This datafile and all the files described or listed here are available at URL `http://www.geom.umn.edu/~sullivan/symfoam/` .

```
torus periods
4 0 0  0 4 0  0 0 4

vertices
10 1 2 0
11 0 1 2
12 2 0 1
20 1 0 2
21 2 1 0
22 0 2 1
60 -1 0 -2
61 -2 -1 0
62 0 -2 -1
70 -1 -2 0
71 0 -1 -2
72 -2 0 -1

edges
121 12 61 + * *
102 10 62 * + *
110 11 60 * * +
201 20 71 * * +
212 21 72 + * *
220 22 70 * + *
301 10 21 * * *
312 11 22 * * *
320 12 20 * * *
401 20 11 * * *
412 21 12 * * *
420 22 10 * * *
621 62 11 * - -
602 60 12 - * -
610 61 10 - - *
701 70 21 - - *
712 71 22 * - -
720 72 20 - * -
801 60 71 * * *
812 61 72 * * *
820 62 70 * * *
901 70 61 * * *
912 71 62 * * *
920 72 60 * * *
```

```
faces

10 121 812 -212 412   color red
11 102 820 -220 420   color green
12 110 801 -201 401   color blue

60 621 312 -712 912   color red
61 602 320 -720 920   color green
62 610 301 -701 901   color blue

20 301 212 720 401 -621 -102
21 312 220 701 412 -602 -110
22 320 201 712 420 -610 -121

70 801 712 220 901 -121 -602
71 812 720 201 912 -102 -610
72 820 701 212 920 -110 -621

30 301 412 320 401 312 420

80 801 912 820 901 812 920

bodies

1 10 11 12 20 21 22 -30 \
  -10 -11 -12 -70 -71 -72 80 volume 32

6 60 61 62 70 71 72 -80 \
  -60 -61 -62 -20 -21 -22 30 volume 32

read

connected;
conj_grad;
optimize 100;

unset body[1] target;
transform_expr "abc";
cost := {
    print (total_area/body_count)^3 /
        (sum(body,volume)/body_count)^2
};
```

Fig. 3.1. This Evolver file describes two cells of the Kelvin foam, which fill a cubic torus ($\mathbb{R}^3$ mod a lattice given by the three `period` vectors). The edges are marked with `wrap` values; edge `121`, for instance, is wrapped by the first period vector, and thus goes from $(2,0,1)$ to $(-2,-1,0)+(4,0,0)$. The square faces are colored red, green or blue, depending on which coordinate plane they are parallel to, while the hexagons are left colored white.

The commands at the end of the file tell the Evolver to display connected bodies and to use the conjugate gradient optimization method with maximum scale 100. Then we remove the redundant volume constraint for the first cell, and ask for the display of eight copies of the graphics, as in Fig. 2.1. Finally, the command `cost` is defined to report the cost μ of the foam at any time. Note also that elements whose identification numbers differ by 5 in the first digit are translates of each other by $(2,2,2)$; this will be used to generate our next datafile.

```
torus periods
-2 2 2  2 -2 2  2 2 -2

vertices
10 1 2 0
11 0 1 2
12 2 0 1
20 1 0 2
21 2 1 0
22 0 2 1

edges
121 12 11 - * *
102 10 12 * - *
110 11 10 * * -
201 20 21 * * -
212 21 22 - * *
220 22 20 * - *
301 10 21 * * *
312 11 22 * * *
320 12 20 * * *
401 20 11 * * *
412 21 12 * * *
420 22 10 * * *
```

```
faces
10 121 312 -212 412     color red
11 102 320 -220 420     color green
12 110 301 -201 401     color blue

20 301 212 220 401 -121 -102
21 312 220 201 412 -102 -110
22 320 201 212 420 -110 -121

30 301 412 320 401 312 420

bodies
1 10 11 12 20 21 22 -30 \
  -10 -11 -12 -20 -21 -22 30 volume 32

read
connected;
conj_grad; optimize 100;
unset body[1] target;
transform_expr "abc";
cost := {
  print (total_area/body_count)^3 /
      (sum(body,volume)/body_count)^2
};
```

Fig. 3.2. This Evolver file describes one cell of the Kelvin foam, in a BCC torus. It is essentially just half of the previous cubic datafile, but note the changes in edge wraps. Edge `121` is still the same edge from $(2, 0, 1)$ to $(2, -1, 0)$, for instance, but it is represented differently with the new period vectors.

`hessian_normal` mode, which ignores tangential motion of vertices). This performs one iteration of Newton's method, converging toward the critical point, but we see little if any change.

The Hessian commands are also useful for testing stability; with the file as it is, we find that we are at a local minimum for energy. But we can now consider removing the volume constraint with `unset body[2] target`; since the cells have equal pressure, the configuration will still be an equilibrium. Now that the two cells in the torus are free to assume different volumes, `hessian` reports that the Hessian matrix has one negative eigenvalue. The `saddle` command will move in the direction of this lowest eigenvector, and if we do this we see immediately that one cell is growing at the expense of the other: without the volume constraint, this foam will collapse, even with enforcement of the cubic translational symmetry.

Of course, this foam actually has further translational symmetry, by the body-centered cubic lattice BCC. We have numbered the vertices, edges, and faces in our original file so that elements differing by the body-centering translation $(2, 2, 2)$ have numbers differing by 5 in the first digit. We can implement a simpler Evolver file for the same foam as in Fig. 3.2, with half as many elements, now enforcing the BCC symmetry. Of course, imposing this symmetry forces all cells to have equal volumes, so we no longer need

```
8
0 0 0:1
12 12 12:5
6 0 12:2
-6 0 12:3
12 6 0:4
12 -6 0:6
0 12 6:7
0 12 -6:8
```

Fig. 3.3. This input file `phw.v` lists the eight Voronoi sites for the Weaire–Phelan foam: two at the lattice points of BCC and the others at half the vertices of the Kelvin foam. They are marked as the centers of bodies `1` through `8`. To generate an Evolver file, we invoke `vcs` with the options `-t d 24 24 24` to specify the torus periods, `-s phw.v -c` to read in the sites and compute the Voronoi partition, and `-p k phw.fe` to print the results in Evolver format.

a volume constraint. (We still list the `body` to make use of the Evolver's `connected` body display option.)

It should be clear that generating input files, even for such a simple foam with a single type of cell, can be difficult to do by hand. In fact, we have written software[4] `vcs` which computes Voronoi partitions in $\mathbb{R}^3$ or in any quotient torus, and can format its output as an Evolver file. To generate an Evolver file for the Weaire–Phelan foam, we can use `vcs` with the input file of Fig. 3.3; the resulting Evolver file is shown in Fig. 3.4. (For didactic purposes the datafiles shown here have been edited by hand, changing for instance the numbering scheme for vertices.)

The Voronoi cells for these sites, computed by `vcs` and described in this file, actually have slightly differing volumes. To describe a polyhedral equal-volume partition in the Weaire–Phelan pattern, we could use weighted Voronoi cells. We have computed [6] that a weighted Voronoi partition on these same sites with equal volumes has $\mu \approx 18.5775$. (This is the only equal-volume polyhedral partition with this symmetry and combinatorics.)

Because the current version of `vcs` does not compute weighted Voronoi partitions, we start with the ordinary Voronoi partition, and let the Evolver equalize the volumes. The commands at the end of our datafile set all bodies to have equal target volumes. After a few steps of iteration, these new constraints will be satisfied, and we get an equal-volume partition, with `cost` $\mu \approx 18.5099$. This is less that the cost computed for the weighted Voronoi partition, because we no longer have planar faces, even though we have not refined the triangulation yet. The Evolver always introduces a new vertex in the center of each given face, to give a triangulation, and our iteration steps have moved these vertices into better positions.

If we continue to iterate and refine (with `g 10; r; g 10; r; g 10`) we end up with the approximation to the Weaire–Phelan foam pictured in Fig. 2.2. It has `cost` $\mu \approx 18.4885$, and uses 4416 triangles, requiring enough

[4] Available from URL `http://www.geom.umn.edu/~sullivan/software/` .

```
torus
periods
24 0 0
0 24 0
0 0 24

vertices
4   5 5 5
401 5 5 -5
402 5 -5 5
403 5 -5 -5
404 -5 5 5
405 -5 5 -5
406 -5 -5 5
407 -5 -5 -5
6   7 7 7
601 7 7 -7
602 7 -7 7
603 7 -7 -7
604 -7 7 7
605 -7 7 -7
606 -7 -7 7
607 -7 -7 -7
11  0 3.75 7.5
111 0 3.75 -7.5
112 0 -3.75 7.5
113 0 -3.75 -7.5
12  7.5 0 3.75
121 -7.5 0 3.75
122 7.5 0 -3.75
123 -7.5 0 -3.75
13  3.75 7.5 0
131 3.75 -7.5 0
132 -3.75 7.5 0
133 -3.75 -7.5 0
17  12 4.5 8.25
171 12 4.5 -8.25
172 12 -4.5 8.25
173 12 -4.5 -8.25
18  8.25 12 4.5
181 -8.25 12 4.5
182 8.25 12 -4.5
183 -8.25 12 -4.5
19  4.5 8.25 12
191 4.5 -8.25 12
192 -4.5 8.25 12
193 -4.5 -8.25 12
34  6 12 0
341 -6 12 0
35  0 6 12
351 0 -6 12
36  12 0 6
361 12 0 -6
```

```
edges
1 406 121 * * *
2 121 123 * * *
3 121 36 - * *
...
90 19 191 * + *
91 603 173 * * *
92 123 361 - * *

faces
1 1 2 5 10 19
2 1 3 7 13 24
...
53 87 85 13 -76 -88
54 72 85 25 -91 -86

bodies
1 1 -3 -4 -5 -9 10 11 -12 -15 \
    -23 24 -32 volume 1687.5
2 15 -28 29 30 31 45 -46 -24 -25 \
    47 7 54 -41 -42 volume 1741.5
3 -2 5 -6 -7 13 -14 -16 -17 -18 \
    -29 19 4 34 -37 volume 1741.5
4 23 -31 -39 -40 41 -48 44 -50 \
    51 6 17 21 8 3 volume 1741.5
5 -47 -49 50 -43 16 53 -54 -51 \
    -52 33 18 -20 volume 1687.5
6 -26 -27 32 42 -44 -45 48 52 -34 \
    -53 36 2 -1 38 volume 1741.5
7 9 14 -21 22 -30 -33 -38 12 27 \
    40 -35 -13 28 49 volume 1741.5
8 -8 -10 -19 20 -22 25 35 -36 37 \
    39 43 46 -11 26 volume 1741.5

read
set facet color 0
set body[2].facet color color+blue
set body[3].facet color color+blue where
 color < blue
set body[4].facet color color+green
set body[6].facet color color+green where
 color < green
set body[7].facet color color+red
set body[8].facet color color+red where
 color < red
set body[1].facet color color+darkgray
set body[5].facet color color+darkgray
vol := sum(body,volume)/body_count;
connected; conj_grad; optimize 100;
set body target vol; unset body[1] target;
cost := {print
    (total_area/body_count)^3 / vol^2 };
```

Fig. 3.4. This Evolver file describes the Weaire–Phelan foam in a cubic lattice. The full listings of edges and faces are not shown. This file was automatically generated, but we have renumbered the vertices by hand to show some of the symmetry. The commands at the end color the faces based on which bodies they separate, and equalize the targets for the volume constraints.

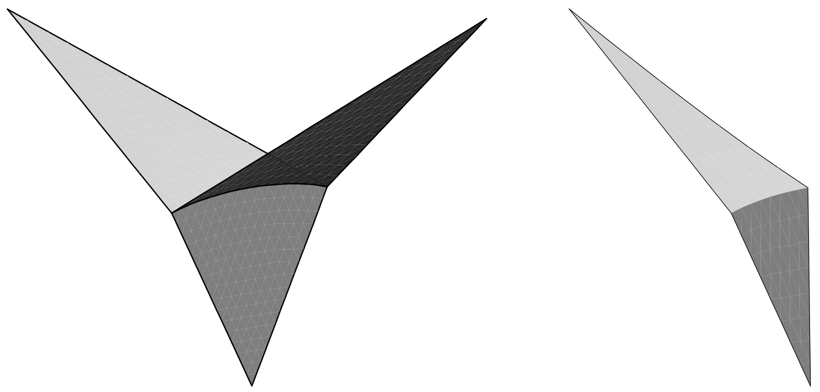

Fig. 4.1. This piece of the Kelvin foam (left) is bounded by six axes of two-fold rotational symmetry. Extending it by these symmetries would recreate the whole foam. The piece we see is described by the datafile in Fig. 4.2(left), and consists of one quarter of a square (at the bottom) plus two sixths of hexagons; it is the unit obtained by cutting the Kelvin foam along the diagonals drawn in Fig. 2.1(left), which are fixed axes of symmetry. This unit still has two perpendicular mirror planes, and thus can be generated by a piece (right) one quarter as big, modeled with half density for the face in the mirror plane.

memory to start to slow down most computer systems. To get better approximations, we will implement some further symmetries of the foam.

4. Using two-fold symmetries of minimal surfaces for the Kelvin foam

Whenever a minimal surface has a straight line boundary, it can be extended across this line by 180° rotation. Thus, a minimal surface with two-fold rotational symmetries can be modeled by using just a piece bounded by the symmetry lines. If this piece is then rotated by the symmetry motions, the rest of the surface will be recovered.

For instance, consider the Kelvin foam of Fig. 2.1. The diagonals of each face, be it a square or hexagon, are axes of two-fold symmetry for the entire structure. If we cut the foam along these axes, we get congruent pieces as in Fig. 4.1(left), each consisting of three sheets, meeting along a triple junction. (Each piece in fact consists exactly of those parts of the entire foam surface closer to this particular triple-junction arc than to any other.) An Evolver datafile for this symmetric piece is shown in Fig. 4.2(left). It is easy to take this piece, repeatedly iterate and refine, and get a surface using under one thousand triangles with `cost` $\mu \approx 18.6762$, a noticeably better approximation than we got with our first Kelvin datafile.

Whenever a minimal surface has a free boundary curve in a plane, it will meet this plane perpendicularly, and can be extended across the plane

```
vertices
1   0 1 0
2   1 0 0
10  1 1 1
20  1 1 -1
30  0 0 0

edges
1   1 2
11 10 1
12 10 2
21 20 1
22 20 2
31 30 1
32 30 2

faces
1 11 1 -12
2 21 1 -22
3 31 1 -32

read

conj_grad;

fix vertex;
fix edge where
    valence<2;

cost := {print
 (3*total_area)^3/16};
```

```
constraint 1 formula x=y

vertices
1  .5 .5 0
2   1 0 0
10  1 1 1
20  1 1 -1
30  0 0 0

edges
1   1 2
11 10 1
12 10 2
21 20 1
22 20 2
31 30 1
32 30 2

faces
1 11 1 -12
2 21 1 -22
3 31 1 -32

read
conj_grad;
set vertex constraint 1 where x=y;
set edge ee constraint 1 where
  min(ee.vertex, on_constraint 1);
fix edge where
  not on_constraint 1 and valence<2;
foreach edge ee where fixed do
  fix ee.vertex;
cost := {print (6*total_area)^3/16}
```

Fig. 4.2. This listing shows two datafiles for Kelvin which use two-fold symmetry. On the left, we describe the piece of the foam shown in Fig. 4.1(left) by fixing the bounding edges along two-fold axes of symmetry. This unit consists of one quarter of a square (face 3) and two sixths of hexagons.

There is a mirror symmetry in the plane $x = y$, so we can instead describe (right) just half of this piece, using the `constraint 1` for the edges in the mirror plane. We could mark vertices and edges to be fixed or constrained when listing them in the datafile, but in both files we choose instead to use the Evolver query language to select the appropriate ones. We constrain all the vertices in the mirror plane to stay there, and also constrain any edge between two such vertices. Any remaining boundary edges are fixed, as are their vertices. Note that, in both cases, we define new `cost` commands to properly account for the area of a whole cell.

by a mirror symmetry. For instance, the piece of Kelvin we have just seen has a mirror plane perpendicular to all three sheets. If we invoke this mirror symmetry, we get the datafile shown in Fig. 4.2(right). Some of the edges of this piece are now not fixed (on rotation axes) but merely constrained to the mirror plane (so that when they are refined, the new vertices and edges created will inherit this constraint).

When dealing with compound soap films or foams with mirror symmetry, we must also consider the case when one piece of the surface lies in the mirror plane, bounded by triple junction curves where other sheets come in symmetrically at 120° angles. To model this, we endow the sheet in the mirror plane with half the density of the other sheets. (It lives in a mirror boundary of the quotient orbifold of $\mathbb{R}^3$ modulo the symmetry group, and thus counts half.) Minimizing the total energy then will automatically result in the correct 120° contact angle. In the datafile of Fig. 4.3 adjusted our cost function to account for the fact that the energy is now different from simply the `total_area`, and is half that of the previous datafile. This piece no longer has any symmetry, so this is a minimal fundamental domain for calculation of the Kelvin foam.

The Evolver has nice features for displaying several copies of a symmetric object. For datafiles in `torus` mode, there are automatically three generators `a`, `b` and `c`, set up to translate the view by the torus period vectors (as used in the example of Fig. 3.1). Then the command `transform_expr` can be used to control the display of one or more fundamental units. The argument `"abc"`, for instance, represents all possible ordered subproducts of this generator string, so we see a $2 \times 2 \times 2$ array. With `"3ab2c"`, defined to equal `"aaabbc"`, we would see instead a $4 \times 2 \times 3$ array. Of course, the lattice translations in torus mode always give a commutative group, so the order of generators in such an expression is irrelevant.

In Fig. 4.3, we see that several generators are defined explicitly at the top, as 4×4 matrices. As is usual in computer graphics, these matrices are used projectively, so that in practice, the upper-left 3×3 block gives a rotation of $\mathbb{R}^3$, while the last column gives any translation. Here our four matrices (automatically assigned the names `a` through `d`) represent mirror reflections in constraints `1` and `2`, and rotations about edges `32` and `12`, in that order. For example we see that the fourth line represents the transformation $(x, y, z) \mapsto (2-x, z, y)$, which is the desired 180° rotation about the edge from $(1, 0, 0)$ to $(1, 1, 1)$. The command `rotpiece` sets the `transform_expr` to be `"ab"`, which gives four copies of the surface, transformed by the two mirror symmetries (which commute). More complicated expressions will generate larger pieces of the complete foam, as in Fig. 4.4, where we see a complete square and hexagon. The `sqrot` command uses expression `"2(ca)b"`, which is equivalent to `"cacab"`. Reading from the right, this tells us to first reflect in the mirror `b`, then repeatedly apply the mirror `a` and the rotation `c` until the square is unfolded. There is no need to apply `b` again at all, as this reflection

```
view_transform_generators 4 //read across, one per line
0 1 0 0   1 0 0 0   0 0 1 0   0 0 0 1
1 0 0 0   0 1 0 0   0 0 -1 0  0 0 0 1
1 0 0 0   0 -1 0 0  0 0 -1 0  0 0 0 1
-1 0 0 2  0 0 1 0   0 1 0 0   0 0 0 1

constraint 1 formula x=y
constraint 2 formula z=0

vertices
1  .5 .5 0
2   1 0 0
10  1 1 1
30  0 0 0

edges
1   1 2
11 10 1
12 10 2
31 30 1
32 30 2

faces
1 11 1 -12
3 31 1 -32

read
conj_grad; hessian_normal;
set vertex constraint 1 where x=y;
set vertex constraint 2 where z=0;
set edge ee constraint 1 where min(ee.vertex, on_constraint 1);
set edge ee constraint 2 where min(ee.vertex, on_constraint 2);
foreach facet ff where min(ff.edge, on_constraint 2) do
    {set ff constraint 2; set ff density 1/2};
fix edge where not on_constraint 1 and valence<2;
foreach edge ee where fixed do fix ee.vertex;
cost := {print (12*total_energy)^3/16}
rotpiece := {transform_expr "ab"};
sqrot := {transform_expr "2(ca)b"};
hexrot := {transform_expr "3(da)b"};
```

Fig. 4.3. This Evolver datafile describes a minimal fundamental unit for the Kelvin foam, bounded by mirror planes and axes of two-fold rotation. We see here a subset of the vertices and edges from our previous datafile. Again we use commands at the end of the datafile to constrain the appropriate elements. Note that it is face number 3 which will be constrained to the mirror boundary $z = 0$. It is set to have half the density of the other face, leading to the correct contact-angle behavior.

Displaying just one copy of the data in this file gives Fig. 4.1(right). The various view transforms can be used to create graphics of larger pieces of the foam by replicating this unit, as shown in Fig. 4.4. The four 4×4 matrices (each listed across one line at the top of the datafile) represent mirror reflections in the two constraints and two-fold rotations about the two fixed edges, in that order.

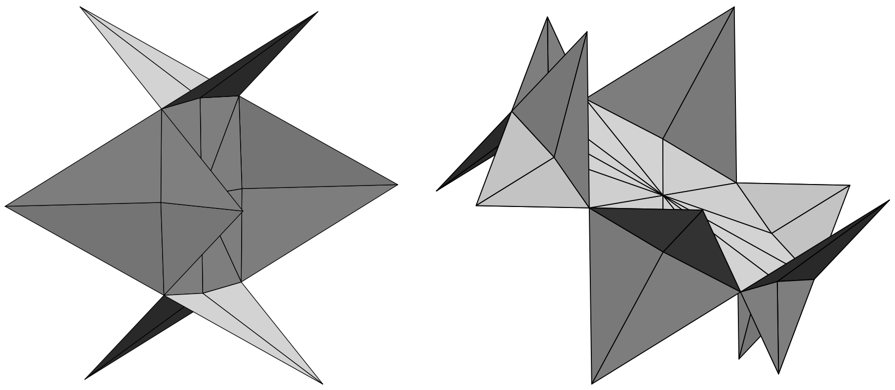

Fig. 4.4. Using view transformations, we can use the fundamental unit from the datafile of Fig. 4.3 to display larger pieces of the Kelvin foam. The command `rotpiece` would reproduce the unit of Fig. 4.1(left). Here we see even larger pieces, shown in the same orientation. If we use `sqrot`, then we see (left) a complete square, together with the adjacent pieces of hexagons; the previous unit appears at the top. If on the other hand, we use `hexrot`, then we see (right) a complete hexagon, together with the adjacent pieces of hexagons and squares; the previous unit appears at the lower right.

in the plane of the square commutes with the other generators. Similarly, `hexrot` uses the rotation `d` in place of `c` to unwrap the hexagon. In these examples, of course, the order of the generators is important.

5. Other rotational symmetries and the Weaire–Phelan foam

The Weaire–Phelan foam also has several symmetries beyond the cubic translation. For instance, a body-centering translation composed with a 90° rotation preserves the foam; since this symmetry has no fixed points, there is a quotient manifold (smaller than the cubic torus) from which the foam lifts. However, the Evolver does not have this manifold built in, as it does the `torus` mode, so we will instead look first at some mirror symmetries. The vertices listed in Fig. 3.4 live in the cubic fundamental domain $(-12, 12]^3$ for the torus used there, and they have been numbered to show the mirror symmetries in the three coordinate planes. The translational symmetries imply that there are also mirrors in the bounding planes of that cube. Therefore, we can model the foam by a unit inside a smaller cube (of side 12) bounded by six mirror planes. The datafile, now containing only the vertices from the original file with positive coordinates, and not their mirror images, is shown in Fig. 5.1. We have in fact also subtracted 6 from all the coordinates here, to make the further symmetries more apparent. Fig. 5.2(left) shows this unit,

```
constraint 1 formula x=6
constraint 2 formula y=6
constraint 3 formula z=6
constraint 4 formula x = -6
constraint 5 formula y = -6
constraint 6 formula z = -6

#define fvi method facet_vector_integral \
 vector_integrand:
#define fvz fvi q1:0 q2:0 q3:
method_instance volx  fvi q1:x  q2:0  q3:0
method_instance volmx fvi q1:-x q2:0  q3:0
method_instance voly  fvi q1:0  q2:y  q3:0
method_instance volmy fvi q1:0  q2:-y q3:0
method_instance volz  fvz   z
method_instance volmz fvz  -z
method_instance volp  fvz   z-6
method_instance volm  fvz  -z-6

quantity volb1 fixed=432 method volx method volmx
quantity volb2 fixed=432 method voly method volmy
quantity volb3 fixed=432 method volz method volmz
quantity volb4 fixed=216 method volp
quantity volb5 info_only method volm
```

```
vertices
4 -1 -1 -1
6  1  1  1
11 -6 -2.25 1.5
12 1.5 -6 -2.25
13 -2.25 1.5 -6
17 6 -1.5 2.25
18 2.25 6 -1.5
19 -1.5 2.25 6
21 1.5 -6 -6
22 -6 1.5 -6
23 -6 -6 1.5
27 -1.5 6 6
28 6 -1.5 6
29 6 6 -1.5
34 0 6 -6
35 -6 0 6
36 6 -6 0
41  6 -6 -6
42 -6 6 -6
43 -6 -6 6
47 -6 6 6
48 6 -6 6
49 6 6 -6
```

```
edges
5 4 6
11 11 4
12 12 4
13 13 4
17 17 6
18 18 6
19 19 6
31 11 35
32 12 36
33 13 34
37 36 17
38 34 18
39 35 19
41 21 41
42 22 42
43 23 43
47 27 47
48 28 48
49 29 49
51 41 36
52 42 34
53 43 35
57 47 35
58 48 36
59 49 34
112 11 22
223 22 13
131 13 21
212 21 12
123 12 23
231 23 11
178 17 28
289 28 19
197 19 27
278 27 18
189 18 29
297 29 17
```

```
faces
31 -31 -231 43 53 volx
32 -32 -212 41 51 voly
33 -33 -223 42 52 volz
37 37 178 48 58 volx
38 38 189 49 59 voly
39 39 197 47 57 volz

11 123 231 11 -12 volx volm
12 131 212 12 -13 voly volm
13 112 223 13 -11 volz volm
17 178 289 19 -17 volmx volp
18 189 297 17 -18 volmy volp
19 197 278 18 -19 volmz volp

14 -13 33 38 18 -5 volz volmy
15 -11 31 39 19 -5 volx volmz
16 -12 32 37 17 -5 voly volmx

read
set vertex constraint 1 where x=6;
set vertex constraint 2 where y=6;
set vertex constraint 3 where z=6;
set vertex constraint 4 where x = -6;
set vertex constraint 5 where y = -6;
set vertex constraint 6 where z = -6;
ii := 6; while (ii>0) do {
  set edge ee constraint ii where
    min(ee.vertex,on_constraint ii);
  foreach facet ff where
    min(ff.edge,on_constraint ii) do
      {set ff constraint ii; set ff density 1/2;};
  ii := ii-1}
set edge color green where original>100;
set edge color red where original>30 and original<40;
set edge color blue where original=5;
show_expr edge where original>0 and valence>1;
cost := {print (total_energy)^3/1728^2;};
```

Fig. 5.1. A datafile for the Weaire–Phelan foam with mirrors, as shown in Fig. 5.2(left). Read first the two columns at the top, then the two at the bottom.

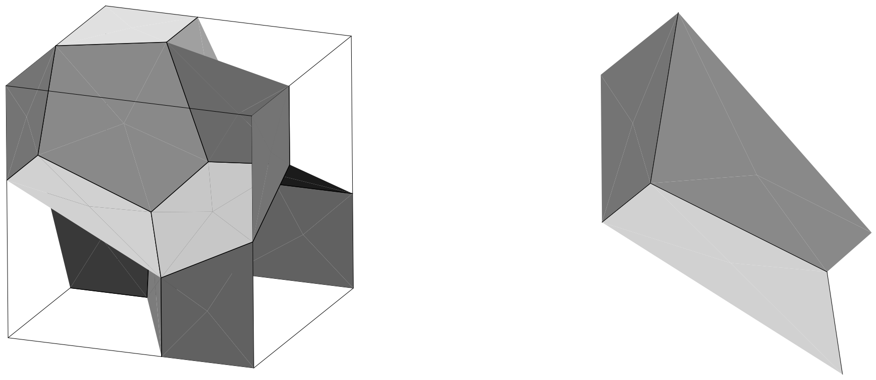

Fig. 5.2. A cube bounded by mirror planes (left) contains a unit of the Weaire–Phelan foam, which includes quarters of three 14-hedra (like the one at the top front), and eighths of two 12-hedra (at opposite cube corners). The edge in the center of the picture connects corners of the two dodecahedra, and is an axis of three-fold rotation, a triple junction between the three 14-hedra. The lines joining its midpoint to the opposite vertices of the three attached pentagons are axes of two-fold symmetry. The faces in the mirror planes are quarters of hexagons (each separating two 14-hedra). To generate this picture we added `bare` edges, bounding the cube, to the datafile of Fig. 5.1. A unit one sixth as big (right) accounts for the rotational symmetries. It includes a quarter of a hexagon, and halves of each kind of pentagon from the foam, and comes from the upper left corner of the cube.

which includes a quarter of a 14-hedron along each of three edges of the cube, and an eighth of a dodecahedron at each of two opposite corners.

In the datafile, note that we have a constraint for each of the mirror planes; the commands at the end automatically constrain vertices, edges and faces based on which mirrors they lie in. We must take special care now to compute cell volumes, which are usually integrals over the faces of a closed body. When a body is bounded partly by planes (like our mirrors) not given as faces in the datafile, we can most easily compute its volume with an integrand that evaluates to zero on the missing faces, using our freedom to choose any two-form with exterior derivative $dx\,dy\,dz$. For instance, the former `body 1` is the 14-hedron along an edge of the cube in the x direction. It would be closed off by faces in mirror planes parallel to the xy- and xz-planes. We integrate its volume with `volx`, the form $x\,dy\,dz$, which vanishes along these missing faces. For `body 4`, one eighth of a dodecahedron, we use `volp`, the form $(z-6)\,dx\,dy$, which integrates to zero along all three missing faces, which are in the mirror planes $x=6$, $y=6$, and $z=6$.

This unit has no further mirrors, but there is rotational symmetry. The edge joining near corners of the two dodecahedra lies along a body diagonal of the cube, which is an axis of three-fold rotation. There are also two-fold rotations interchanging the ends of this same edge. We could make a model half as big, fixing the three edges of two-fold symmetry as for the Kelvin

```
#include "include.h"
#define stdwrap(w) ((((w)<0? -2*(w):(w))+1)%3 - 1)

void xyz_wrap(x,y,wrap)
REAL *x; REAL *y; WRAPTYPE wrap;
{
  memcpy((char*)y,(char*)x,SDIM*sizeof(REAL));
  if (!(wrap = stdwrap(wrap))) return;
  if (wrap==-1) y[0]=x[1], y[1]=x[2], y[2]=x[0];
  else /*wrap 1*/ y[0]=x[2], y[1]=x[0], y[2]=x[1];
}

WRAPTYPE xyz_compose(wrap1,wrap2)
WRAPTYPE wrap1,wrap2;
{ return stdwrap(wrap1 + wrap2); }

WRAPTYPE xyz_inverse(wrap)
WRAPTYPE wrap;
{ return stdwrap(-wrap); }

void xyz_form_pullback(x,xform,yform,wrap)
REAL *x; REAL *xform; REAL *yform; WRAPTYPE wrap;
{ xyz_wrap(yform,xform,-wrap); }
```

Fig. 5.3. Adding this code to the Evolver source will implement the `xyz` symmetry group of three-fold rotation about the $(1, 1, 1)$-axis. Elements are represented by the integers $\{0, \pm 1\}$ mod 3. Included are routines to compose, invert, and apply these symmetries, and also one to pull back one-forms. The names of these routines must be entered in the table in `registry.c` to use the new symmetry.

foam, without using any new symmetry features. The edge along the three-fold axis is similarly fixed, but because the foam is not cut into three pieces by this edge, we need to teach the Evolver explicitly about the symmetry. A datafile using a named symmetry group will have edges marked with `wrap` values as in a torus, but these wraps are now integers encoding the elements of a group. Some symmetry groups are built in to the Evolver, but new ones can be included if we add routines to the source code to compose and apply these wrap values.

A simple example is the group of rotations about the $(1, 1, 1)$-axis. This is a cyclic group of order three, which we can represent by integers modulo 3. Code (in `C`) implementing this group is shown in Fig. 5.3. The group can be used to model a cube (Fig. 5.4) which will flow to a spherical bubble. Under this symmetry, there are four orbits of vertices, four of edges and two of faces, and these each appear once in the datafile. Note that one edge is described with a `wrap` value, meaning that its second vertex has coordinates cycled by the group element.

This group, unlike the fundamental group in the `torus` mode, has fixed points along the axis, and we need to allow a surface to cross this axis. To implement this, we include a vertex constrained to be on the axis, and mark it as an `axial_point`. A surface through this point is then given by a single wedge-shaped face, replicated by rotation to fill out a disk around

```
symmetric_content
view_transform_generators 1
0 1 0 0  0 0 1 0   1 0 0 0   0 0 0 1
symmetry_group "xyz"
constraint 1 formula x=y
constraint 2 formula y=z
#define axis axial_point constraint 1 2
vertices
1   1  1  1 axis
2  -1  1  1
3  -1 -1  1
4  -1 -1 -1 axis
edges
1   1 2
2   2 3
3   3 2 wrap 1
4   4 3
faces
1   1 2 3 -1
2   4 -2 -3 -4
body
1  1 2 volume 8/3
read
transform_expr "2a"; raw_cells
```

Fig. 5.4. Like the file `cube.fe` included with the Evolver distribution, this file represents an initial cube, which will flow to a spherical bubble as its area is decreased with volume fixed. This file, however, uses the three-fold `xyz` symmetry.

the axial vertex. This face (like each face in the cube example) will start with an edge out of the axial vertex, and end with the same edge in opposite orientation, after a wrap by some group element. Mathematically, we could think of closing off the face with an edge from the axial vertex to itself, merely to undo the net `wrap` of the face. But the Evolver does not like zero-length edges, so instead we leave out that edge, with the understanding that a face can have nonzero net `wrap` if it starts at an axial point. In this case, it is required that the first edge listed for the face be a positive edge, outward from the axial point. Listing any other faces with nonzero net wrap will give errors, but in this one case, the Evolver takes special care to track the wrap needed to close off the face.

Note also that we specify `symmetric_content`, which means that the volume of a body is computed by integrating not $z\,dx\,dy$ over its faces, but instead the symmetrized version $\frac{1}{3}(x\,dy\,dz+y\,dz\,dx+z\,dx\,dy)$. This, integrated over just one fundamental piece of the surface under our symmetry group, will give one-third the total volume of the cube. We start with a cube of total volume 8, and fix this volume by telling the Evolver that the volume computed for the fundamental piece is 8/3.

We can use this same `xyz` group to implement the Weaire–Phelan foam with all of its symmetries, as in Fig. 5.5. This file contains elements from the previous datafile (Fig. 5.1): one selected from each orbit under the sym-

```
symmetry_group "xyz"

view_transform_generators 2
0 1 0 0    0 0 1 0  1 0 0 0   0 0 0 1
0 -1 0 0  -1 0 0 0  0 0 -1 0  0 0 0 1

constraint 1 formula x=6
constraint 3 formula z=6
constraint 5 formula y = -6
constraint 7 formula x=y
constraint 8 formula x+y=2*z
constraint 9 formula z=0

#define fvi method facet_vector_integral vector_integrand:
method_instance volx  fvi q1:x   q2:0   q3:0
method_instance volmx fvi q1:-x  q2:0   q3:0
method_instance voly  fvi q1:0   q2:y   q3:0
method_instance volp  fvi q1:x-6 q2:y-6 q3:z-6

quantity volb1 info_only method volx method voly method volmx
quantity volb4 fixed=216 method volp

vertices
5 0 0 0 fixed constraint 7 8 9
6 1 1 1 axial_point constraint 7 8
36 6 -6 0 fixed constraint 1 8 9
17 6 -1.5 2.25 constraint 1
28 6 -1.5 6 constraint 1 3
48 6 -6 6 fixed constraint 1 3 5

edges
48 28 48 constraint 1 3
58 36 48 constraint 1 5
178 17 28 constraint 1 color green
389 17 28 constraint 1 wrap 1 color green
37 36 17 constraint 1 color red
17 6 17
5 6 5 constraint 7 8 color blue
6 5 36 constraint 8 9

faces
37 37 178 48 -58 density .5 constraint 1 volx
15 5 6 37 -17 voly volmx
17 17 178 -389 -17 volmx volp

read
show_expr edge where original>0 and (valence>1 or color=blue)
cost := {print (6*total_energy)^3/1728^2;};
conj_grad; hessian_normal; transform_expr "b2a";
```

Fig. 5.5. This datafile implements the Weaire–Phelan foam with full symmetry. The unit described here is that shown in Fig. 5.2(right), though the view transformations and expression will show six copies of the unit, reproducing Fig. 5.2(left). The new constraints `7` and `9` are used to constrain edge `5` along the three-fold axis of `xyz` symmetry. In this unit we see one eighth of a dodecahedron, and one eighth of a 14-hedron, corresponding to bodies 1 and 4 of our last datafile. The quantities `volb1` and `volb4` are supposed to measure their volumes, but only `volb4` is correctly invariant under the symmetry.

metry. We have also introduced vertex 5 at the midpoint of an earlier edge, and edge 389 (replacing 289) is now listed with a wrap. Note that when we constrain this edge to have $x = 6$, the constraint is applied to the edge as unwrapped from its tail: the head vertex 28 is on this constraint when wrapped to $(6, 6, -1.5)$ by the symmetry group. Also, two new constraints have been added to fix edge 5 (and the axial_point vertex) along the axis of xyz symmetry, while letting the edge get longer or shorter. (Three other vertices are marked fixed; we might have chosen instead to list three constraints for each of them.)

Our previous datafile computed the volume of the dodecahedron by integrating $(z - 6)\,dx\,dy$ over face 17 and over its rotations under the symmetry group. Here, equivalently, we integrate a symmetrized integrand volp over just that one face, the only face of a dodecahedron present. Although the quantity volb1 computes the complementary volume correctly at first, note that its integrands (volx, etc.) are not invariant under the group. As we refine, some new triangles will appear not where their parents were, but rotated under the group. Therefore this quantity integrated over them will no longer be correct. Thus the invariant volb4 is the quantity we can safely pick to fix; the complementary volume is then automatically fixed. We leave volb1 in for info_only purposes merely to demonstrate this problem with noninvariant integrands.

The current Evolver distribution does not include our group xyz because it is subsumed in the symmetry cubocta, which implements the full point group of the cube or octahedron. The group elements for cubocta are encoded as integers, with the three low-order bits indicating sign changes of the coordinates, then a two-bit value giving the xyz rotation, and finally a single bit indicating if the x and y coordinates are swapped. To change a datafile to use this built-in group instead of xyz, we can simply change the name "xyz" at the top to "cubocta" and then replace wrap values 1 and -1 by 8 and 16 respectively.

There is also a built-in rotation group, which can implement rotational symmetry of any order around the z-axis. The order of the group is given as a parameter in the datafile, and can even be changed interactively; the wrap values are integers in the obvious interpretation. A variant, flip-rotate, replaces the rotational generator by one which also does a mirror in the xy-plane.

6. Using edge energies to eliminate faces in mirror planes

We have demonstrated Evolver datafiles for both foams which implement minimal fundamental units, corresponding to the respective full symmetry groups. In both cases, these files include faces in mirror symmetry planes.

Such faces, of course, must stay planar, and can never really evolve. As we refine the surface, there will be more and more vertices within these faces, but they will never have any reason to move. In some similar situations, such vertices will bunch up as a contact line moves across the plane, leading to a bad triangulation. Our foams do not move much from their initial positions, so the triangulation does not get hung up in this way. But, still, these extra elements gain us nothing and will slow down the computations.

To remove them, we can recognize that the area of any face in a fixed plane can be computed by an edge integral. Typically just one or two edges are free (within the mirror plane), and their positions determine the size of the face.

For the Kelvin foam, Fig. 6.1(left) shows a file equivalent to that in Fig. 4.3, but with face `3` (and the vertices and edges required only there) removed. That face was in the plane $z = 0$, bounded by edge `1` and the lines $x = y$ and $y = 0$. Thus we look for a one-form vanishing on those lines, with exterior derivative $dx\,dy$; one such form is $y(dx - dy)$. The area of the face we removed can be computed by integrating this form along edge `1`; as before, we add half this area to the energy.

Our most recent datafile for the Weaire–Phelan foam, Fig. 5.5, also had one face in a mirror plane. In Fig. 6.1(right) we remove this face `37` (and the vertex and edges no longer needed), and compute its area again with an edge integral. This time the face is in the plane $x = 6$, and it is bounded by edges `37` and `178` as well as two lines $y = -6$ and $z = -6$ of the mirrored cube in Fig. 5.2(left). Thus the area can be computed by integrating $(y + 6)\,dz$ (which vanishes on the two lines) along those two edges.

These two files allow the most efficient calculation of the optimal shape and cost of the two foams. For instance, the commands at the end of the datafile for Kelvin tell the Evolver to refine a total of six times, leading to a foam with `cost` $\mu \approx 18.6758$, using 4096 triangles. The commands in the other datafile refine a total of four times, leading to an approximation of the Weaire–Phelan foam with `cost` $\mu \approx 18.4871$, using 2048 triangles. (These computations take about 2.4 seconds and 8 seconds, respectively, on a Silicon Graphics R4400 machine running at 175 MHz.) In both cases, we believe the resulting approximation is good enough to give the cost of the true foam accurately to the four decimal places we report. The symmetry features of the Evolver allow us to make these computations in a very short time, and also allow further calculations to even greater accuracy, which would be impractical without using the symmetry.

Color Plates 1.2 and 1.3, pp. 25 and 26 in the Appendix, show views from inside these foams. These images were created with a soap-bubble shader written for the RenderMan package [1]. It uses the laws of thin-film optics to render the sheets of the foam as if they were soap-films of randomly varying thickness.

```
quantity wall
energy
modulus 1/2
method edge_vector_integral
vector_integrand
q1:y
q2:-y
q3:0

constraint 1 formula x=y

constraint 2 formula z=0

vertices
1  .5 .5 0 constraint 1 2
2   1 0 0  fixed
10  1 1 1  fixed

edges
1   1 2 constraint 2 wall
11 10 1 constraint 1
12 10 2 fixed

faces
1 11 1 -12

read

conj_grad;
optimize 100;
hessian_normal;

cost := {
 print
  (12*total_energy)^3/16;
};

g5; {r;u;g12;cost}4;
r;u;g2; r;g;cost;
```

```
symmetry_group "xyz"

constraint 1 formula x=6
constraint 3 formula z=6
constraint 7 function x=y
constraint 9 function y=z

quantity volb4 fixed=216
method facet_vector_integral
vector_integrand q1:x-6 q2:y-6 q3:z-6

quantity wall energy modulus 1/2
method edge_vector_integral
vector_integrand q1:0 q2:0 q3:y+6

vertices
5  0 0 0       fixed
6  1 1 1 axial_point constraint 7 9
36 6 -6 0      fixed
17 6 -1.5 2.25 constraint 1
28 6 -1.5 6    constraint 1 3

edges
178 17 28 constraint 1 wall
389 17 28 constraint 1 wrap 1
37  36 17 constraint 1 wall
17   6 17
5    6  5 constraint 7 9
6    5 36 fixed

faces
15 5 6 37 -17
17 17 178 -389 -17 volb4

read
conj_grad;optimize 100;hessian_normal
cost := print total_energy^3/8/1728;
g20; {r;u;g10;cost} 2;
r;u;g5;cost; r;u;g;cost
```

Fig. 6.1. These datafiles for the Kelvin (left) and Weaire–Phelan (right) foams are equivalent to those in Fig. 4.3 and Fig. 5.5, respectively. Here we have removed the faces in mirror planes, and compute their areas instead by integrals along their edges. These are added to the energy computed by the Evolver through the `quantity wall`, using an appropriate one-form in each case. In both files, we have omitted the viewing transforms, colors, and so forth, which could be implemented as before. The last two lines give sample scripts for evolving the surfaces down to good approximations to the mathematical foams.

References

1. F. J. Almgren, Jr. and J. M. Sullivan, *Visualization of soap bubble geometries*, Leonardo **24**:3/4 (1992), 267–271, reprinted in *The Visual mind: Art and mathematics.*
2. K. A. Brakke, *The surface evolver*, Experimental Mathematics **1**:2 (1992), 141–165.
3. G. Francis, J. M. Sullivan, R. B. Kusner, K. A. Brakke, C. Hartman, and G. Chappell, *The minimax sphere eversion*, Mathematics and Visualization (K. Polthier and H.-C. Hege, eds.), Springer Verlag, Berlin, 1996, pp. ???–???
4. L. Hsu, R. Kusner, and J. M. Sullivan, *Minimizing the squared mean curvature integral for surfaces in space forms*, Experimental Mathematics **1**:3 (1992), 191–207.
5. W. Thompson, Lord Kelvin, *On the division of space with minimum partitional area*, Acta Math. **11** (1887), 121–134.
6. R. Kusner and J. M. Sullivan, *Comparing the Weaire-Phelan equal-volume foam to Kelvin's foam*, Forma (1996), to appear.
7. R. B. Kusner and J. M. Sullivan, *Möbius energies for knots and links, surfaces and submanifolds*, Geometric Topology: The Proceedings of the Georgia International Topology Conference (W. H. Kazez, ed.), International Press, Cambridge, MA, 1996, pp. 603–637.
8. D. Weaire and R. Phelan, *A counter-example to Kelvin's conjecture on minimal surfaces*, Phil. Mag. Lett. **69**:2 (1994), 107–110.

Appendix: Color Images

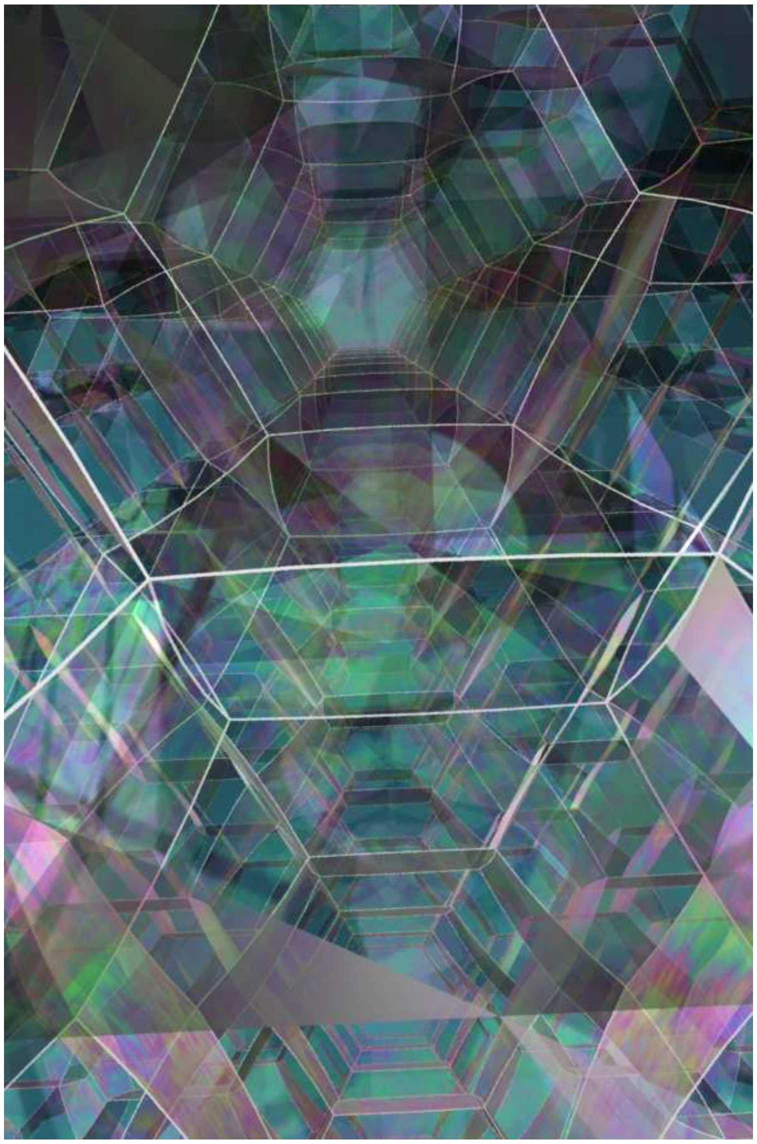

Fig. 1.2. This is a view from inside the Kelvin foam, using a fairly wide-angle lens. Near the top, we look along a body diagonal of the cubic lattice, through successive parallel hexagons of the foam. Near the bottom, we look in a $(1,1,0)$ direction, through hexagons which alternately tilt towards us and away from us. Our viewpoint is slightly below the center of our cell. (Brakke and Sullivan, p. 21.)

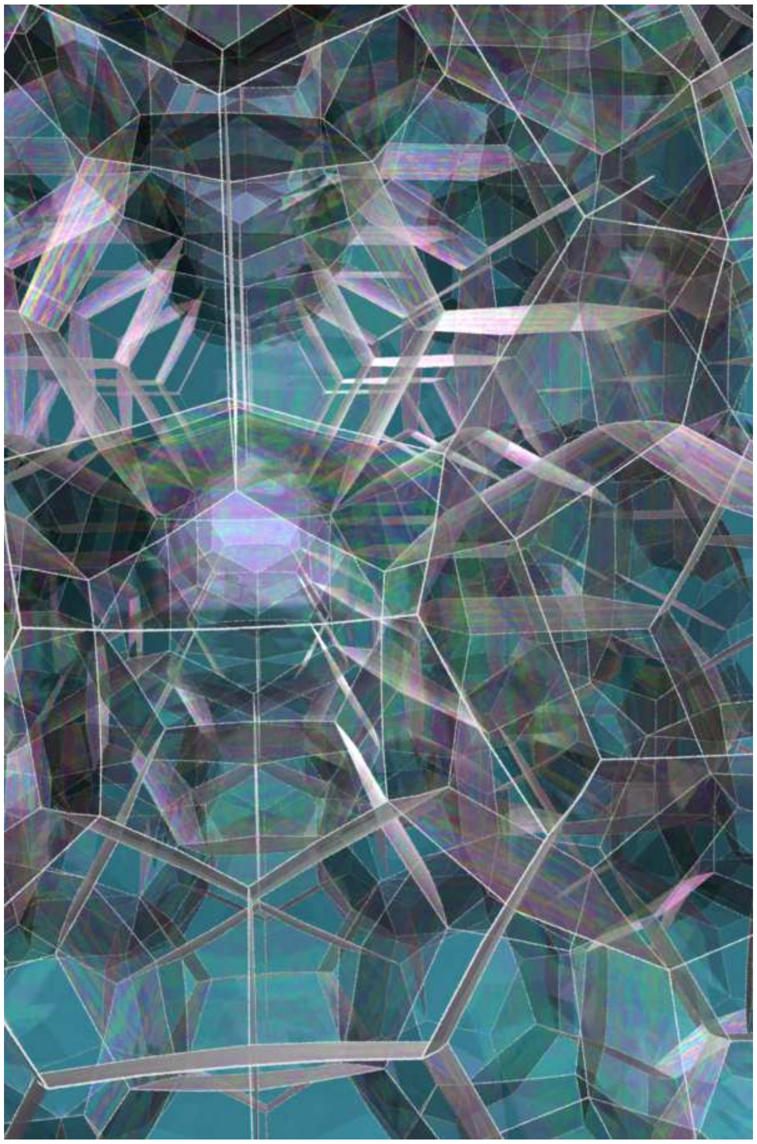

Fig. 1.3. This is a view from inside the Weaire–Phelan foam, using a wide-angle lens. We are inside one of the 14-hedra, and near the bottom left, we see one of its hexagonal faces. Behind that (and a bit above) is parallel hexagon turned a quarter turn, part of an infinite sequence of parallel hexagons receding towards the bright spot in the middle left. (We are a bit too high in our cell to see all the way through these.) Near the top left, we look through an infinite number of receding dodecahedra, separated by hexagons seen edge-on. (Brakke and Sullivan, p. 21.)