Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

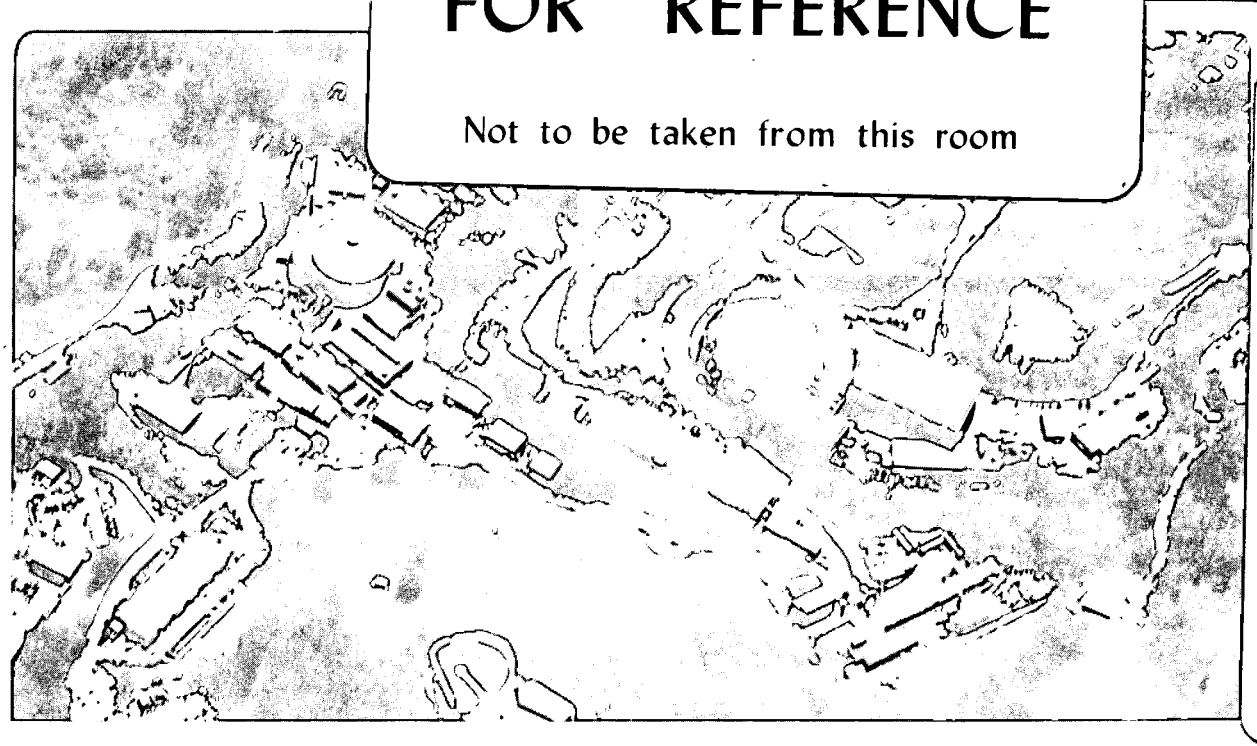
Physics Division

Mathematics Department

Computing Minimal Surfaces Via Level Set Curvature Flow

D.L. Chopp
(Ph.D. Thesis)

May 1991



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098

DISCLAIMER

LBL-30685

COMPUTING MINIMAL SURFACES VIA LEVEL SET CURVATURE FLOW[1]

David L. Chopp

Lawrence Berkeley Laboratory
and
Department of Mathematics
University of California
Berkeley, CA 94720

Ph.D. Thesis

May 1991

[1] This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

Dedication

This thesis is dedicated to my parents for all the love and support they have given me through my education.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my advisor, Professor James Sethian, for his invaluable assistance in the creation of this thesis.

I would like to thank Professor Ole Hald for his many helpful comments and suggestions.

I would also like to thank Steve Larson and Michael Stewart of Cray Research for their help with computing problems, and Wes Bethel of Lawrence Berkeley Laboratory for his help with graphics rendering.

This work was supported in part by the Applied Mathematics Subprogram of the Office of Enegry Research US Department of Energy under contract DE-AC03-76SF-00098 and by an International Business Machines Graduate Fellowship.

Chapter 1

Introduction

Given a finite union Γ of fixed closed curves in R^3, there is an associated problem of finding a surface with mean curvature zero which has Γ as its boundary. In this paper, a new approach to numerically solving this minimal surface problem is introduced. The surface is represented as a level set of a global function $\Phi : R^3 \rightarrow R$. The surface evolves according to mean curvature flow until a steady state solution is achieved. A new system of interpolatory boundary conditions are used to maintain the connection between the moving surface and the fixed boundary contour.

1.1 What is a Minimal Surface?

Given a finite union Γ of simple closed curves in R^3 a minimal surface is a surface with least area that has Γ as its boundary. The simplest example of a minimal surface is where Γ is a circle and the minimal surface is the disk bounded by Γ. A more interesting example is the classic catenoid surface discovered by Euler [6] shown in Figure 1.1.

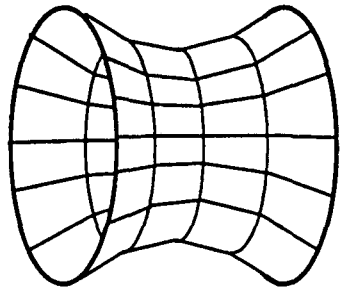


Figure 1.1: Classic catenoid solution of Euler

From a practical standpoint, minimal surfaces arise naturally in many physical models, the most familiar being soap films fitting to a wire boundary. They also appear in the study of statics of flexible and inextensible films, translation nets, relativity theory, quantum string theory, computer

aided design, medical technology, and even architecture. In general, the problem of producing an analytic representation of a minimal surface for a given arbitrary boundary Γ is exceedingly difficult. Since constructive analytical solutions are impractical, a numerical approximation to minimal surfaces becomes a valuable tool.

1.2 Lagrange Formulation

The study of the minimal surface problem was begun by J. L. Lagrange [11] in 1762. He reasoned as follows: Let γ be the projection of Γ onto the $xy-$plane, and let D be its interior. Assume that the minimal surface can be expressed as $z = m(x,y)$ where $m(x,y) \in \Gamma$ for $(x,y) \in \gamma$. Define $z(x,y) = m(x,y) + \epsilon\delta(x,y)$ where $\delta(x,y) = 0$ for $(x,y) \in \gamma$. Next, define the surface area functional $A(\epsilon) = \int\int_D \sqrt{1 + z_x^2 + z_y^2}\, dx\, dy$. Since m is a minimal surface, then A must have a minimum at $\epsilon = 0$, thus $A'(0) = 0$. By differentiating under the integral sign and setting $\epsilon = 0$, it follows that

$$A'(0) = \iint_D \frac{m_x\delta_x + m_y\delta_y}{\sqrt{1 + m_x^2 + m_y^2}}\, dx\, dy = 0$$

Integration by parts gives

$$A'(0) = -\iint_D \left\{ \frac{\partial}{\partial x}\left(\frac{m_x}{\sqrt{1 + m_x^2 + m_y^2}}\right) + \frac{\partial}{\partial y}\left(\frac{m_y}{\sqrt{1 + m_x^2 + m_y^2}}\right) \right\} \delta(x,y)\, dx\, dy$$

Because $\delta(x,y)$ can be any smooth function on D, then it follows that

$$\frac{\partial}{\partial x}\left(\frac{m_x}{\sqrt{1 + m_x^2 + m_y^2}}\right) + \frac{\partial}{\partial y}\left(\frac{m_y}{\sqrt{1 + m_x^2 + m_y^2}}\right) = 0$$

for all $(x,y) \in D$. Carrying out the differentiation produces the equation:

$$\frac{(1 + m_x^2)m_{yy} - 2m_{xy}m_xm_y + (1 + m_y^2)m_{xx}}{(1 + m_x^2 + m_y^2)^{3/2}} = 0. \tag{1.1}$$

Eliminating the denominator in Equation (1.1) produces what is called the minimal surface equation

$$(1 + m_x^2)m_{yy} - 2m_{xy}m_xm_y + (1 + m_y^2)m_{xx} = 0. \tag{1.2}$$

A geometric interpretation of equations (1.1) and (1.2) is given in the following section.

1.3 Minimal Surfaces and Mean Curvature Zero

The mean curvature of a surface at a point is defined as the average of the two principle curvatures of the surface at that point. A formula for the curvature in the case where $z = m(x,y)$ is given by the left hand side of Equation (1.1). For a complete derivation of the mean curvature

formula see [13]. A surface which has mean curvature equal to zero at every point is said to have the property *mean curvature zero.*

Given a boundary contour Γ, a surface is *locally minimal* with respect to Γ if any small perturbation of the surface increases the total area of the surface. A surface is *globally minimal* if it has the least area of any surface which has Γ for its boundary. Equation (1.1) above indicates that if a surface is locally minimal, then that surface has mean curvature zero. A proof of this fact can be found in Nitsche [13]. The converse is not necessarily true. In Section 1.5 a boundary is shown admitting a mean curvature zero surface which is not locally minimal. Surfaces of this type are called *unstable.* However, the accepted definition of minimal surface has been extended to include all surfaces with mean curvature zero. It is this extended definition which will be used in this paper. Note that numerical area minimizing algorithms will not ordinarily find unstable minimal surfaces.

For Euler's catenoid solution in Section 1.1, there are at least two solutions which topologically are like a cylinder. One surface is stable and locally area minimizing, while the other is not. This may be seen as follows. The formula for the catenoid is given by $r(x) = a\cosh(x/a)$, where $r(x)$ is the radius of the catenoid at the point x along the x-axis and $a > 0$ is the radius of the catenoid at the center $x = 0$. Suppose that the boundary consists of two rings of radius R located at $\pm b$ on the x-axis. From this, the parameter a is determined by the equation

$$R = a\cosh(b/a). \tag{1.3}$$

If no real value of a solves Equation (1.3), then no catenoid solution exists. Assuming that the catenoid solution exists, solving for b in Equation (1.3) gives $b = a\cosh^{-1}(R/a)$. Let R be fixed, then $\lim_{a\to 0} b(a) = 0$ and $b(R) = 0$, therefore there exists a value $a_{\max}$ such that $b_{\max} = b(a_{\max}) \geq b(a)$ for all $a \in (0, R]$. Furthermore, for every value $b_0 \in (0, b_{\max})$ there exists at least two values of a for which $b(a) = b_0$.

In terms of the catenoid, this means that given R, there is a maximal distance $2b_{\max}$ where the rings may be held apart and still give a catenoid minimal surface. When the rings are less than the maximal distance apart, there are two distinct catenoid solutions corresponding to the two different values of a. When the rings are exactly $2b_{\max}$ apart, only one catenoid solution exists. When the rings are farther apart, no catenoid solutions exist. For values of $b < b_{\max}$, the catenoid with the smaller inner radius is the unstable solution.

1.4 Plateau's Problem

In 1873, J. Plateau [15] published his experimental work on minimal surfaces. He conducted extensive studies of soap films and soap bubbles. Because of his famous experiments, the problem of finding a disk type minimal surface solution for a given single boundary curve has become known as "Plateau's problem." After numerous physical experiments, Plateau conjectured that for every given boundary contour there exists a minimal surface bounded entirely by that contour.

In 1930, T. Radó [16] and J. Douglas [4] were independently able to prove Plateau's conjecture. But while existence had finally been established, the proof did not lead to a practical way of computing a surface for a specific boundary.

1.5 Topological Types of Minimal Surfaces

In Section 1.3 it was shown that the general minimal surface problem does not have the uniqueness property. Even for the simpler problem of Plateau, uniqueness has been proven only under very strict conditions.

As an example, the catenoid solution by Euler is not the only minimal surface which has two rings as their boundary. Let the boundary Γ be given as in Section 1.3. The boundary Γ is shown in Figure 1.2. If the rings are not too far apart in relation to the radius of the rings, then

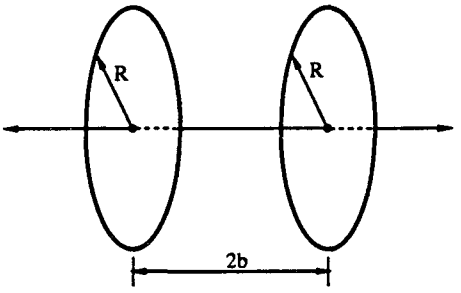


Figure 1.2: A two ring boundary arrangement

three topologically different solutions exist as in Figure 1.2. One solution consists of two flat disks

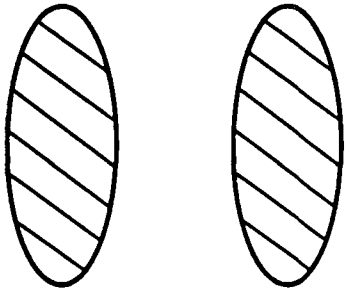


Figure 1.3: Two disks

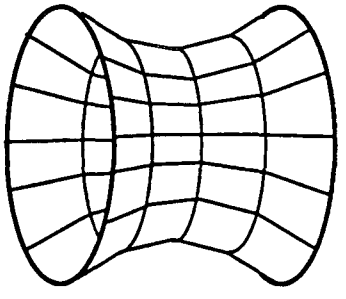


Figure 1.4: Euler's catenoid

(Figure 1.3), a second, called the catenoid is topologically equivalent to a cylinder (Figure 1.4), and the third is formed by a catenoid with a disk sewn into the center (Figure 1.5). For the third example, the radius of the surface is given by the function

$$r(x) = a\left(\cosh\left(\frac{|x|}{a}\right)\sqrt{3} + \frac{1}{\sqrt{3}}\sinh\left(\frac{|x|}{a}\right)\right) \tag{1.4}$$

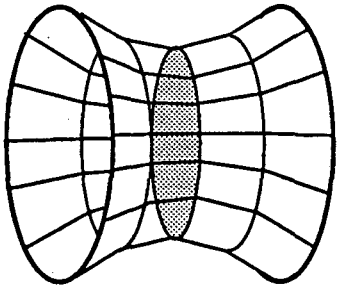


Figure 1.5: Catenoid with disk

where a is determined by the equation $R = r(b)$.

It is reasonable to think that a catenoid solution exists for rings that are any distance apart. It was shown in Section 1.3 that this is not the case. This example illustrates how it is not always possible to predict the topology of a potential solution based solely on the boundary. This is an important point which affects the way minimal surfaces are computed.

1.6 Existing Numerical Methods

There are two general types of numerical Plateau's problem solvers. The distinction between the types depends on the way surfaces are represented.

1.6.1 Non-Parametric Surface Solvers

The simplest way to try to solve the minimal surface problem is to represent the surface as a function $z = f(x, y)$ similar to the argument given in Section 1.2. Because the surface is represented in a non-parametric form, algorithms of this type are called non-parametric surface solvers. In general, they work by discretizing a piece of the xy–plane and using the value of f on those grid points. The local surface area, the mean curvature, or surface tension of f is then iteratively reduced by changing the values of f on the grid.

A large number of algorithms in this category have been published, for example, see Concus [2], Hoppe [10], Elcrat and Lancaster [5], and Greenspan [8].

Algorithms of this kind are typically simple to program and computationally inexpensive. However, they face certain shortcomings. The most obvious difficulty is how to solve for surfaces which cannot be projected one-to-one onto a plane. While special cases may be calculated by using symmetry arguments, this type of algorithm is not desirable in the more general case.

A more subtle, yet critical, failure of these methods is that they explicitly assume the general topological type of the final solution. It is not always possible to know *a priori* the topology of the solution as seen in Section 1.5. Since these methods are unable to change topology during computation, they have difficulty capturing the correct solution.

1.6.2 Parametric Surface Solvers

In order to solve for surfaces which cannot be projected onto a plane, a second class of algorithms has been developed which represent a surface as a collection of connected triangles. Similar to the non-parametric surface solvers, the vertices of the triangles are moved iteratively to reduce local area, mean curvature, or surface tension. Methods of this type have been developed by Brakke [1], Wagner [19], Hinata, Shimasaki, and Kiyono [9], and Coppin and Greenspan [3].

While this new class of algorithms do solve one shortcoming of the previous class, they must still predict the topology of the solution before computing. Again, the inability of the algorithms to allow topological changes makes it difficult to find the correct solution. For example, if the cylindrical type is predicted when only the disk solution exists in the example above, then these methods result in a final solution as depicted in Figure 1.6 (figure from [19]).

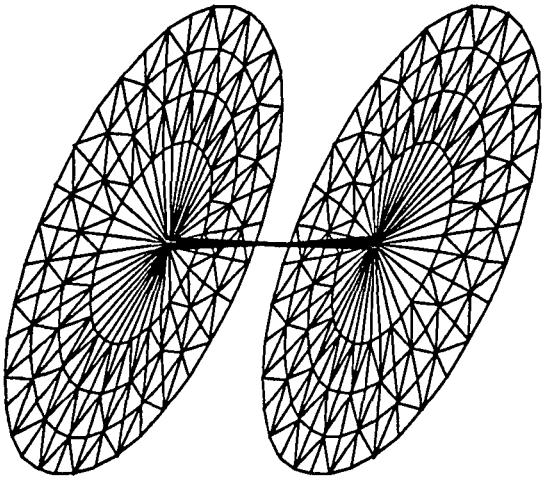


Figure 1.6: Erroneous solution of Euler's catenoid

This solution is clearly incorrect since the center portion is essentially a very thin cylinder which has arbitrarily high mean curvature. Ideally, if that central cylinder could be cut, in other words the topology of the surface could change during computation, then false results such as this could be avoided.

Chapter 2

Level Set Curvature Motion

In the previous chapter, the desirability of an algorithm that can predict topological changes was discussed. In a paper by Osher and Sethian [14], a technique for following interfaces propagating with curvature dependent speed was introduced which allows topological changes. This chapter is a summary of the relevant portions of that paper.

2.1 Lagrangian Representation of Curvature Flow

To begin, consider a smooth two-dimensional surface S in R^3. At each point x of S, for mean curvature flow, the velocity of x is in the direction normal to S with magnitude equal to the local mean curvature of S at x. The case of a one-dimensional surface in the plane along with the velocity vectors of the surface are depicted in Figure 2.1. More precisely, let S_0 be an initial smooth

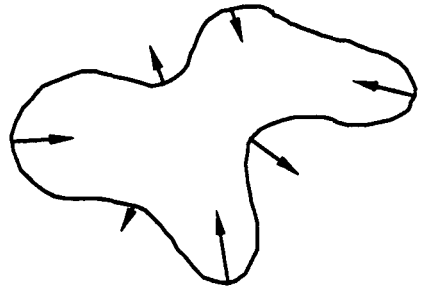


Figure 2.1: One-dimensional curvature flow velocity field

surface and $x(0) = x_0 \in S_0$. Then the equation of motion for the point $x(t)$ is given by

$$\begin{aligned} \frac{dx}{dt}(t) \cdot n &= \kappa(x(t)) \\ x(0) &= x_0 \end{aligned} \tag{2.1}$$

where n is the unit normal to S at the point x and $\kappa(x)$ is the mean curvature of S at x. Recall that mean curvature is the average of the two principle curvatures, the sign of which depends on the chosen normal to S. However, changing the sign of the normal vector only changes the sign of the curvature. Therefore, for this equation of motion it is not necessary to specify which normal is used. This formulation of the motion is said to be in Lagrangian terms because the coordinate system moves with the surface S.

For example, let S_0 be a sphere of radius R, and choose the inward pointing normal everywhere. Then at every point x_0, the curvature is given by $1/R$. The solution surface S_t

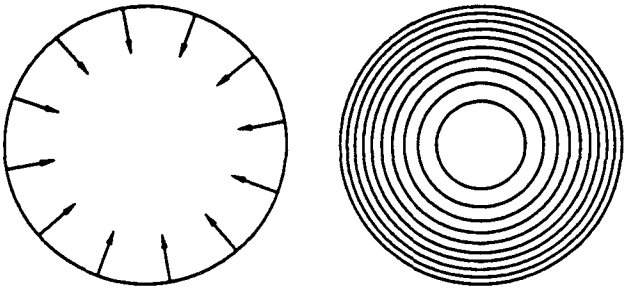


Figure 2.2: Shrinking sphere curvature flow

satisfying Equation (2.1) is the sphere of radius $\sqrt{R^2 - 2t}$. In this case,

$$x(t) = \frac{x_0}{\|x_0\|}\sqrt{R^2 - 2t}, \quad n(t) = -\frac{x_0}{\|x_0\|}, \quad \text{and} \quad \kappa(x(t)) = \frac{1}{\sqrt{R^2 - 2t}}.$$

It is easy to see that these satisfy Equation (2.1) for all $t \in [0, R^2/2)$.

For the parametric surface solvers which use local mean curvature for their equations of motion, the formulation in Equation (2.1) is used for the surface flow. But Sethian [17] noted that while this formulation for the curvature flow model is relatively easy to implement, it can eventually lead to difficulties.

One way to numerically model this type of flow is by using marker particles. These marker particles correspond to the vertices of the triangles in the non-parametric surface solvers. The idea is to pick a number of initial points $x_0, \ldots, x_n \in S_0$ and then allow them to move according to Equation (2.1) where the normal n and the curvature κ are computed for each x_i by looking at its nearest neighbors. Sethian [17] showed that this algorithm has two inherent flaws. First, if points should get too close together instability can result. Second, it is not clear what to do if the surface should change topology, in other words, if it should merge together or break apart. In this case, the concept of nearest neighbors between points must change.

Not coincidentally, these are the same difficulties that the existing parametric minimal surface solvers face. In some circumstances, maintaining stability in these methods is not trivial, and at present only *ad hoc* intervention by the user can alter the topology of the surface.

2.2 Level Set Representation of Curvature Flow

A different approach was introduced by Osher and Sethian is to model curvature flow in an Eulerian coordinate system. To begin, the surface is represented as a level set of some function $\Phi : R^3 \rightarrow R$. An equation of motion for Φ equivalent to the Lagrangian formulation may be found by treating every level set $\Phi^{-1}(C)$ as a separate curvature flow problem. This leads to an equation of motion for Φ over the entire domain. For the details see Osher and Sethian [14], and Sethian [17]

Consider an arbitrary level set $\{x \in R^3 : \Phi(x,t) = C\}$. Implicit differentiation with respect to t in this equation gives

$$\nabla\Phi(x,t) \cdot \frac{dx}{dt}(t) + \Phi_t(x,t) = 0 \tag{2.2}$$

Note that $-\nabla\Phi/||\nabla\Phi||$ is a unit normal to the level set $\Phi^{-1}(C)$, then by combining Equations (2.1) and (2.2),

$$\Phi_t = ||\nabla\Phi|| \frac{-\nabla\Phi}{||\nabla\Phi||} \cdot \frac{dx}{dt}(t) = ||\nabla\Phi|| n \cdot \frac{dx}{dt}(t) = \kappa(x(t))||\nabla\Phi||. \tag{2.3}$$

The mean curvature $\kappa(x(t))$ can be represented solely as a function of Φ and its derivatives by

$$\kappa = \frac{\Phi_{xx}(\Phi_y^2 + \Phi_z^2) + \Phi_{yy}(\Phi_x^2 + \Phi_z^2) + \Phi_{zz}(\Phi_x^2 + \Phi_y^2) - 2\Phi_{xy}\Phi_x\Phi_y - 2\Phi_{yz}\Phi_y\Phi_z - 2\Phi_{xz}\Phi_x\Phi_z}{2(\Phi_x^2 + \Phi_y^2 + \Phi_z^2)^{3/2}}. \tag{2.4}$$

Therefore, the equation of motion for the surface $\Phi^{-1}(C)$ has been changed into a quasi-linear second order parabolic partial differential equation which is independent of the level set value C.

Returning to the example of the shrinking sphere, suppose the function Φ is initialized by $\Phi(x,y,z,0) = \sqrt{x^2+y^2+z^2} - R$. In Section 2.1 the rate of change of the radius of each level set of Φ with respect to time was calculated to be $\sqrt{R^2 - 2t}$. Using that computation, then $\Phi(x,y,z,t)$ can be determined by looking back in time and determining where the level set including $\Phi(x,y,z,t)$ was at time $t = 0$. Doing this produces the solution

$$\Phi(x,y,z,t) = \sqrt{x^2+y^2+z^2+2t} - R. \tag{2.5}$$

This solution can be verified by inserting it into Equation (2.3).

Equation (2.5) shows that every level set of Φ is a separate shrinking sphere. No level set is any different than any other except for its initial radius determined by the constant R. In fact, the shrinking spheres represent a very simple form of changing topology. In a marker particle method, special considerations must be taken when fronts get too close together. In contrast, the level set formulation allows the spheres to disappear naturally.

A more interesting example of a topological change was computed by Sethian [18]. He considered an initial shape of a dumbbell, two large spheres connected by a cylinder. As predicted, the shrinking of the spheres is slower than the shrinking of the cylinder, so the dumbbell eventually breaks in the center. The picture of a slice of the object as it shrinks is depicted in Figure 2.3. To see how this relates to the level curves of Φ, Figure 2.4 shows the various stages of the surface

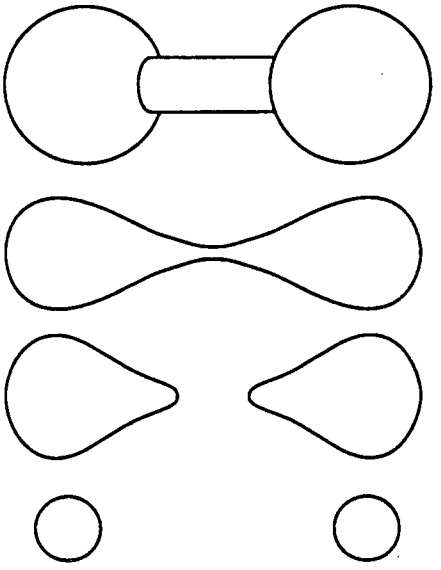


Figure 2.3: Motion of a dumbbell under mean curvature

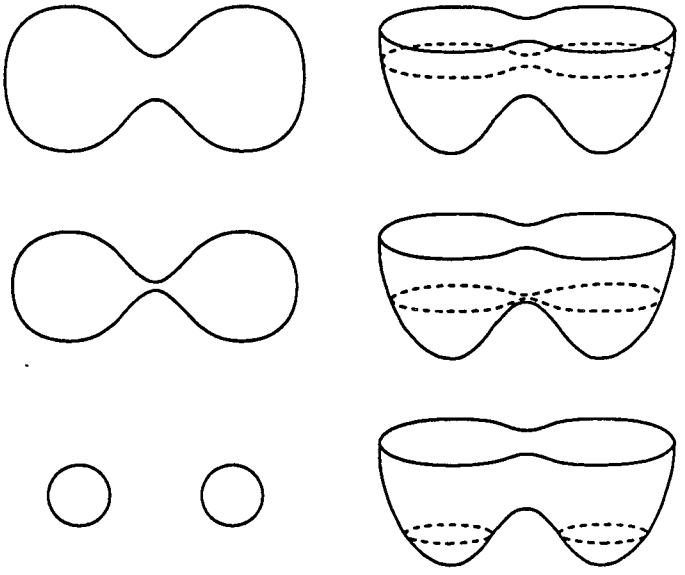


Figure 2.4: Breaking dumbbell vs. $w = \Phi(x, y, 0, t)$

$\Phi^{-1}(0)$ and the corresponding graph of $\Phi(x, y, 0, t)$. Note how the topology changes when the level set $\Phi^{-1}(0)$ contains the center critical point of Φ where $\nabla\Phi = 0$.

This new level set approach to curvature flow problems provides remedies to two difficulties in the previous formulations. First, marker particle instability is eliminated because the grid remains fixed for all time. Second, the surface is able to change topology naturally without the need for *ad hoc* decisions. These two properties can be exploited to create a new minimal surface algorithm.

2.3 Osher-Sethian Numerical Method

In [14], [17], a numerical method for modelling curvature flow using the level set representation for Φ was developed. In those papers a more general curvature dependent speed function was considered. For the case of speed equal to mean curvature (see [17]), any type of time derivative method, for example Euler's method, Runge-Kutta's method, or an implicit method can be employed. For the space derivatives, simple central differences are used. Thus, for Euler's method, the numerical scheme for modelling curvature flow reduces to

$$\Phi_{ijk}^{n+1} := \Phi_{ijk}^{n} + \Delta t * K(\Phi^n)$$

where $K(\Phi^n)$ is a finite difference approximation to the curvature from Equation (2.4). In this case, the finite difference approximation is simple central differences for all derivatives of Φ in the interior of the domain, thus

$$\Phi_x(x_{ijk}, t_n) \approx \frac{\Phi_{i+1,jk}^n - \Phi_{i-1,jk}^n}{2\Delta x}$$

$$\Phi_{xx}(x_{ijk}, t_n) \approx \frac{\Phi_{i+1,jk}^n - 2\Phi_{ijk}^n + \Phi_{i-1,jk}^n}{\Delta x^2}$$

$$\Phi_{xy}(x_{ijk}, t_n) \approx \frac{\Phi_{i+1,j+1,k}^n - \Phi_{i-1,j+1,k}^n - \Phi_{i+1,j-1,k}^n + \Phi_{i-1,j-1,k}^n}{4\Delta x \Delta y}$$

$$\vdots$$

At the boundary of the domain one sided finite differences are used.

In order to construct $\Phi(x, 0)$ for a given initial surface S, the signed distance function from the initial surface is used. For the signed distance, the magnitude of Φ is given by $|\Phi(x, 0)| = \text{dist}(x, S)$. Given an orientation for the normal n of S, the sign of Φ is determined by the requirement that $d\Phi/dn|_S = 1$. As noted above, for mean curvature flow it does not matter which side of the surface has positive values of Φ. In the sphere example, Φ was the signed distance from the initial surface of a sphere of radius R with positive values on the outside.

Chapter 3

Computation of Minimal Surfaces

So far it has been argued that a new type of minimal surface algorithm is needed to handle the topological complexity of finding minimal surfaces. In the previous chapter, an alternative representation of surfaces was described including how this representation would be used to model curvature flow. In this chapter a new minimal surface algorithm is described. It uses the level set curvature flow model discussed in the previous chapter.

3.1 Mean Curvature Flow and Minimal Surfaces

So far, curvature flow has been studied in the absence of any kind of obstacles such as boundaries. A possible way to compute minimal surfaces is to attach a surface to a given boundary and let it move according to its mean curvature. However, in terms of curvature flow this leads to a discontinuous speed function at the boundary.

For an example of the discontinuous speed, let the initial boundary curve be a circle of radius R in the plane and the initial surface be a hemisphere of radius R as in Figure 3.1. At all

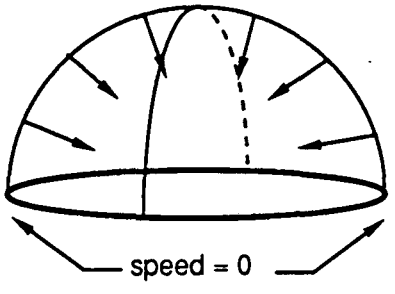


Figure 3.1: Example of a discontinuous speed function

points up to but not including the boundary, the initial speed of any point on the surface is $1/R$.

But at the boundary, we want the speed to be zero. A minimal surface algorithm must be able to handle this discontinuity.

Another difficulty faced by curvature flow concerns the case of when $\nabla\Phi = 0$, which is in the denominator of the curvature formula. Since we are primarily concerned with making the numerator of the curvature formula tend to zero, the minimal surface algorithm presented here leaves out the denominator for the curvature flow. Unless otherwise noted, curvature flow in this chapter refers to the curvature formula without the denominator.

3.2 Basic Algorithm for Minimal Surfaces

A solution for the discontinuous velocity property is simply to continually "reattach" the surface back to the boundary after it has moved away. A diagram of how this might be done using discrete time steps is depicted in Figure 3.2. In some sense, the surface is extended by a sort of

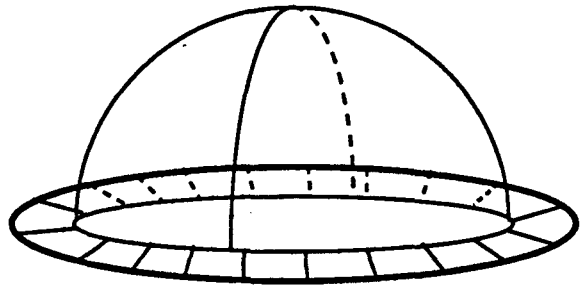


Figure 3.2: Reattaching the surface to the boundary

annulus consisting of line segments joining points on the fixed boundary to the corresponding points where the boundary would be were it allowed to move with the rest of the surface. In the case of the hemisphere attached to the circle, in time Δt, the edge of the hemisphere would move towards the center a distance of approximately $\Delta t/R$. Reattaching the boundary in this case would consist of adding to the surface the annulus of outer radius R and inner radius $R - \Delta t/R$. Thus, a first attempt at a minimal surface algorithm based on curvature flow might look like this:

Algorithm 1

Step 1. Move surface according to curvature flow.
Step 2. Reattach surface to boundary.
Step 3. Go to step 1.

The algorithm will use the level set formulation of curvature flow to follow the surface $\Phi^{-1}(0)$. In this formulation, the boundary contour divides $\Phi^{-1}(0)$ into two disjoint pieces, $\Phi^{-1}(0) = I \cup E$, where E is defined to be the *exterior set* and the compact *interior set* defined by I also contains the boundary.

Once again consider the boundary contour Γ consisting of a ring of radius R in the xy–plane. One might make the initial guess of a solution to be a hemisphere S of radius R attached to the ring. To realize S as a level set of some function Φ, the surface S must be continued out to the edges of the domain of Φ. One way to continue the surface is to let E be the xy–plane minus the disk bounded by Γ, so that $\Phi^{-1}(0)$ would appear as in Figure 3.3. Now Φ can be constructed

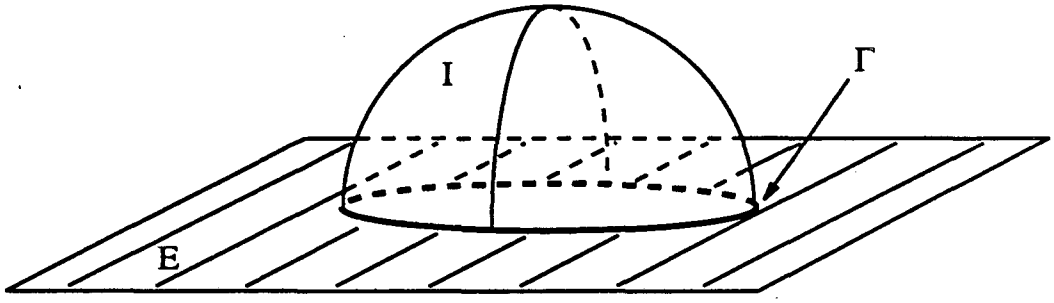


Figure 3.3: $\Phi^{-1}(0) = E \cup I$

by choosing the sign of Φ to be, say, negative below the surface and positive above. Clearly, the portion of the surface that is of interest is the set I. The set E is only needed to construct Φ.

This example also illustrates the procedure for initializing the minimal surface algorithm. First, an initial guess of the solution is made giving the set I. Second, the initial surface is extended in some way to the limits of the domain of Φ giving the set E. Finally, Φ is initialized by taking the signed distance to the surface $E \cup I$. We may assume without loss of generality that Φ is a C^∞ approximation of the signed distance function for purpose of ensuring the existence of the derivatives of Φ. Even though an initial guess is necessary to start the algorithm, it need not be of the same topological type as the final solution unlike the existing minimal surface algorithms.

In reference to step two, reattaching the boundary means changing the value of Φ in a neighborhood of the boundary so that $\Phi \equiv 0$ at the boundary. The reattaching process is described below.

3.3 Boundary Conditions

The process of reattaching the surface to the boundary in this numerical method follows from the idea that Φ must only be altered locally at the boundary so that $\Phi \equiv 0$ at the boundary. Doing this involves answering two questions. First, how does one model a one-dimensional contour on a three-dimensional rectangular grid? Second, how can Φ be altered locally so that $\Phi \equiv 0$ at the boundary?

To illustrate the strategy, consider the simpler problem of finding the shortest distance between two points A and B in the plane. The goal is to find conditions on Φ near the boundary

(the points A and B) so that $\Phi(A,t) = \Phi(B,t) = 0$ for all time. As an example, see Figure 3.4. The grid points around the point A are depicted in Figure 3.5. Suppose that the point A is located

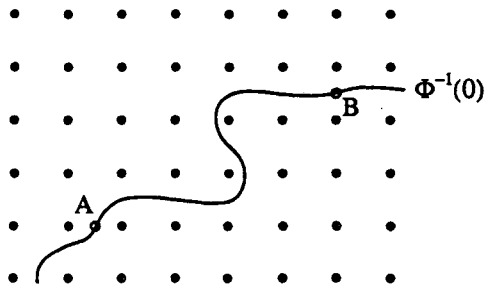


Figure 3.4: Grid for two points connected by a curve

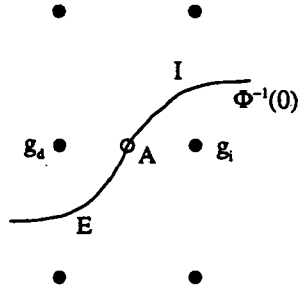


Figure 3.5: Grid points around the boundary

directly in the middle between the points g_i and g_d. In order for $\Phi(A) = 0$ on a grid, it follows from linear interpolation that Φ must be adjusted to satisfy the equation $\Phi(g_i) = -\Phi(g_d)$. Therefore, either the value of $\Phi(g_i)$ or the value of $\Phi(g_d)$ must be set for all time to be the negative of the other. Since the setting of the value of Φ at a particular point is in some sense artificial, in order to not influence the motion of the set I, we let the value of Φ closer to I be updated by the curvature flow and readjust the point nearer the exterior set E. In Figure 3.5, g_i is nearer to I. The point g_i is defined to be an *independent point* because the value of $\Phi(g_i)$ changes only according to curvature flow. The point g_d is defined to be a *dependent point* because at each time step we reattach the boundary at the point A by setting $\Phi(g_d) \leftarrow -\Phi(g_i)$. Here the left arrow indicates the value $\Phi(g_d)$ is being assigned the value $-\Phi(g_i)$.

In the following sections we explain how to label dependent and independent points and compute their dependency coefficients. Second, we show how labelling should be done to construct consistent boundary conditions. Third, we give an example of how to construct the boundary conditions for a circle. Finally, we describe an algorithm for computer-generated boundary conditions.

In general, the boundary conditions will be represented as a vector equation of the form

$v_{\text{dep}} \leftarrow A v_{\text{ind}}$ where

$$v_{\text{dep}} = \begin{bmatrix} g_{d,1} \\ g_{d,2} \\ \vdots \\ g_{d,m} \end{bmatrix}, \qquad v_{\text{ind}} = \begin{bmatrix} g_{i,1} \\ g_{i,2} \\ \vdots \\ g_{i,n} \end{bmatrix}$$

and A is an $m \times n$ matrix. Since the matrix A only depends on the fixed boundary Γ, then the coefficients of A are constant for all time. The computation of the coefficients for A can be broken down into a linear combination of smaller matrix blocks of size 1×1 and 2×2.

For the remainder of this section let $\Gamma = \{\gamma(s) \in R^3 : s \in [0, P]\}$, where $\gamma(s)$ is continuous with $\gamma(0) = \gamma(P)$, be a single boundary contour.

3.3.1 Definitions

As $\gamma(s)$ traverses through the grid, there are three ways it can interact with the grid. Define a *zero-point* to be a grid point g such that $g \in \Gamma$. Define an *edge* to be a line segment L connecting two adjacent grid points such that $L \cap \Gamma \neq \emptyset$. Define a *pane* to be a rectangular region R bounded by four connecting coplanar edges such that $R \cap \Gamma \neq \emptyset$. An example of each is depicted in Figure 3.6.

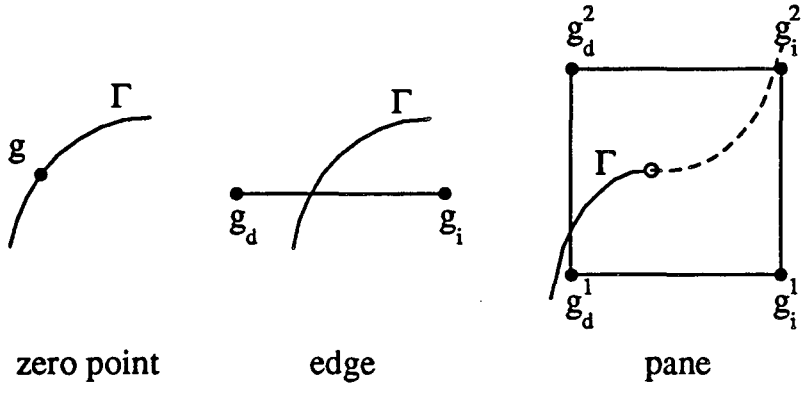


Figure 3.6: Interaction between a curve and a grid

3.3.2 Grid Point Weighting

The boundary Γ is discretized into a piecewise linear path with vertices consisting of all the intersections of Γ with zero-points, edges, and panes in the grid. Keeping Γ fixed then means fixing those vertices with respect to the grid. The boundary conditions needed for fixing the vertices require that, independent and dependent grid points are designated and that the resulting dependency coefficients are calculated.

For a zero-point g, the point itself is defined to be dependent with no dependency coefficient. Since g is always on the boundary and $\Phi = 0$ on the boundary, then $\Phi(g) \leftarrow 0$.

For an edge, one end point must be designated independent, the other dependent. In general, the point closer to the interior set I is chosen as independent. In this case, $\Phi(g_d) \leftarrow C\Phi(g_i)$ where $C < 0$ is a constant for all time. To calculate C, suppose the intersection point of Γ and the edge is given by $\alpha g_d + (1-\alpha)g_i$ with $\alpha \in (0,1)$. Linear interpolation implies that $C = (\alpha - 1)/\alpha$.

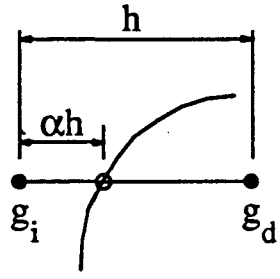


Figure 3.7: Dependency calculation of an edge

For a pane, two adjacent points must be chosen to be independent, the other two are then dependent. Again, the two points sharing a common edge nearest to the set I are chosen as independent. In this case the two points will straddle the intersection of I with the pane. The dependency relation is then a 2×2 matrix equation. Let the intersection point be given by $(1-\alpha-\beta)g_{i,1} + \alpha g_{i,2} + \beta g_{d,1}$. Then the dependency relation is given by

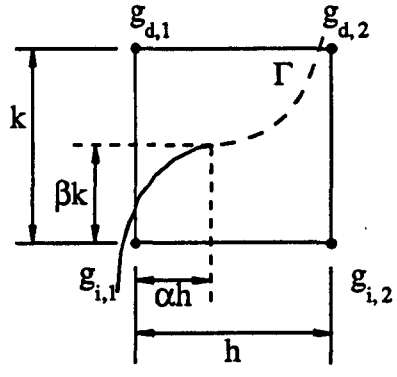


Figure 3.8: Dependency calculation of a pane

$$\begin{bmatrix} \Phi(g_{d,1}) \\ \Phi(g_{d,2}) \end{bmatrix} \leftarrow \frac{1}{\beta} \begin{bmatrix} \alpha+\beta-1 & -\alpha \\ \alpha-1 & \beta-\alpha \end{bmatrix} \begin{bmatrix} \Phi(g_{i,1}) \\ \Phi(g_{i,2}) \end{bmatrix}.$$

3.3.3 Chain Construction and Consistency

Unfortunately, the choices made for edges and panes are not independent. There may be grid points which belong to more than one pane or edge. Define a *link* to be either a pane or an edge. Define a *chain* to be an ordered list of links with any two consecutive links sharing at least

one common grid point. A simple example of a chain is depicted in Figure 3.9. As a point $\gamma(s)$

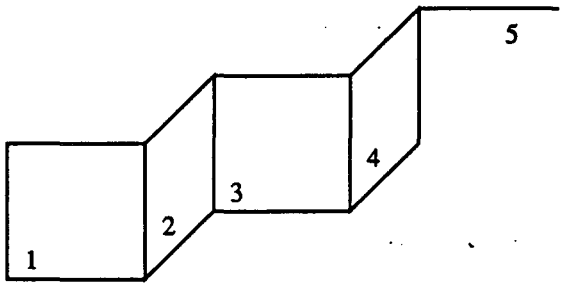


Figure 3.9: A sample chain

traverses the grid a string of connected panes and edges are intersected.

In general, the complete boundary conditions for Γ are not one large chain, but instead made up of any number of disjoint chains. These chains can be separated in any number of ways, for example, Γ may intersect opposite facing panes, a pane opposite a parallel edge, two opposite edges, etc. Some chains are shown in Figure 3.10

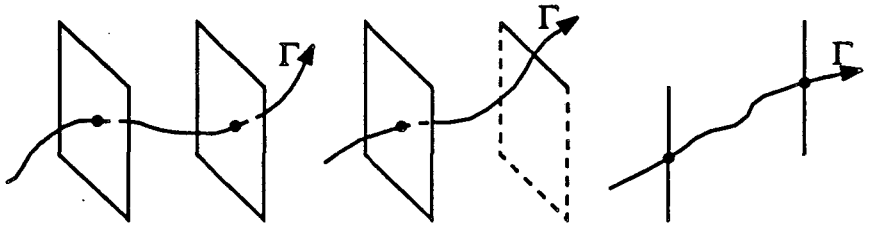


Figure 3.10: Example chain separators

Define a chain to be *consistent* if every grid point contained in the chain is labelled either independent or dependent but not both. The goal of choosing independent and dependent points is to construct consistent chains. In the chain shown in Figure 3.9 there are exactly the four consistent labellings shown in Figure 3.11.

Unfortunately, there exist chains which do not permit any consistent labellings. Such chains may arise when the grid is too coarse to adequately model a given boundary. To get around this problem, moving the boundary with respect to the grid by some small amount ϵ may help. If nudging the boundary does not help, then only refining the mesh will eliminate such chains.

Note that the intersection points between Γ and the edges and panes of the grid are strictly local events, i.e. the location of the intersection of Γ and pane one in Figure 3.9 should have no influence over the location of the intersection of Γ in pane four. However, an inconsistent chain can lead to dependencies stretching all along the length of a chain. For example, consider the inconsistent labelling of the two pane chain in Figure 3.12. In pane one, point A depends on points B and D.

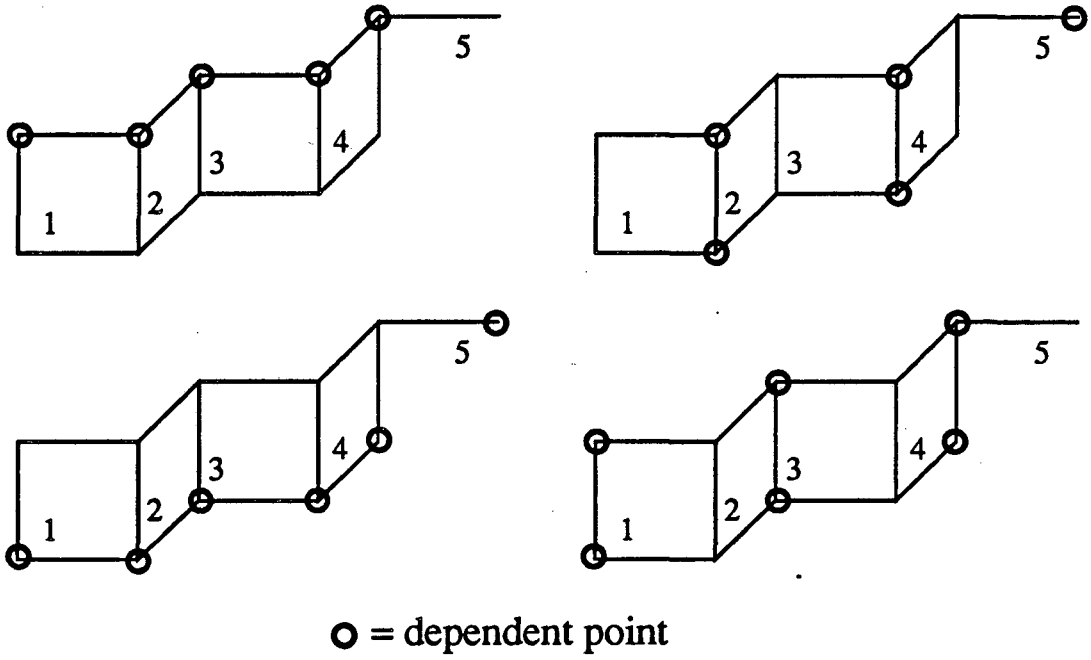


Figure 3.11: Four consistent labellings

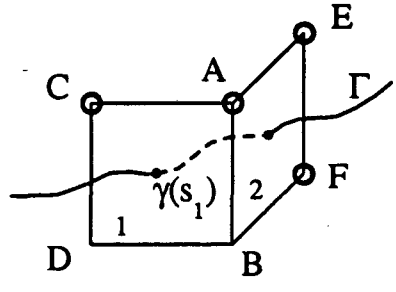


Figure 3.12: An inconsistent chain labelling

In turn, in pane two, point E depends on point A. At the point A, $\Phi(A)$ is artificially set to force $\Phi(\gamma(s_1)) = 0$. But now the values assigned to $\Phi(E)$ and $\Phi(F)$ are computed using a dependent value, so their values are influenced by $\Phi(D)$ which should not be the case. A better way to label the points might be to let the point D be dependent rather than A. This results in a consistent labelling. Notice that in this case, the dependent values only rely on local independent points.

An algorithm for choosing consistent labellings is given in Section 3.3.8.

3.3.4 Chains and the Dependency Coefficients

The structure of the chains also affect the calculation of the dependency coefficients. The number of dependent and independent points within a chain may not be the same, so the boundary conditions may be an overdetermined system of equations.

For example, consider the labelled two pane chain in Figure 3.13. The dependency relations

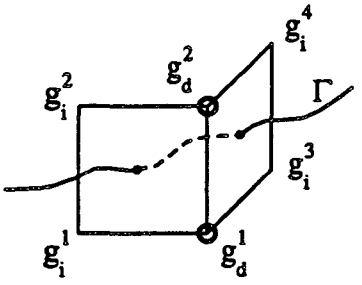


Figure 3.13: Overdetermined labelling

for such a system are given by

$$\begin{bmatrix} \Phi(g_{d,1}) \\ \Phi(g_{d,2}) \end{bmatrix} \leftarrow A_1 \begin{bmatrix} \Phi(g_{i,1}) \\ \Phi(g_{i,2}) \end{bmatrix} \qquad \begin{bmatrix} \Phi(g_{d,1}) \\ \Phi(g_{d,2}) \end{bmatrix} \leftarrow A_2 \begin{bmatrix} \Phi(g_{i,3}) \\ \Phi(g_{i,4}) \end{bmatrix}$$

The values $\Phi(g_{d,1})$ and $\Phi(g_{d,2})$ are overdetermined. One option for solving the overdetermined system is to average the values $\Phi(g_{d,1})$ and $\Phi(g_{d,2})$, thus

$$\begin{bmatrix} \Phi(g_{d,1}) \\ \Phi(g_{d,2}) \end{bmatrix} \leftarrow \frac{1}{2}\left(A_1 \begin{bmatrix} \Phi(g_{i,1}) \\ \Phi(g_{i,2}) \end{bmatrix} + A_2 \begin{bmatrix} \Phi(g_{i,3}) \\ \Phi(g_{i,4}) \end{bmatrix} \right)$$

Note that if the surface is a plane, then the averaging solution is correct. As the mesh is refined the surface locally becomes more like a plane. Therefore, as the mesh is refined, the error in this averaging process decreases.

3.3.5 Example Boundary Construction: A Circle

To illustrate, as an example, the boundary consisting of chains is computed for a circle. Consider a circle of radius $3\frac{1}{2}$ parameterized as

$$x(\theta) = \tfrac{3}{2}, \quad y(\theta) = 5 + \tfrac{7}{2}sin(\theta), \quad z(\theta) = \tfrac{9}{2} + \tfrac{7}{2}cos(\theta)$$

Define the grid to have dimensions $2 \times 9 \times 8$ centered about the origin with uniform space step $h = 1$. This is an exceedingly large space step, but is used here to simplify the arithmetic. For now, suppose that the interior set I is in the positive x direction. This would be the case if the initial surface is to be like a cylinder, for example, and this circle is on one end. The relationship between the boundary curve and the grid are shown in the two views in Figure 3.14.

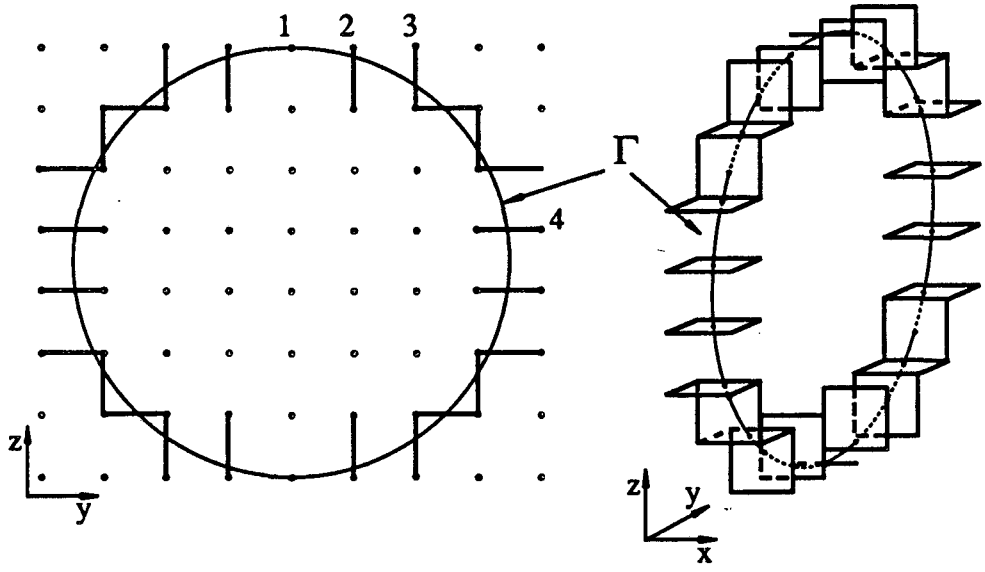


Figure 3.14: Example of a grid with a boundary curve

In this example there are fourteen chains, four of which are shown in Figure 3.15. By symmetry, it is only necessary in this example to compute the boundary conditions for the first four chains.

Chain 1

This chain consists of only one link, an edge connecting grid points $g_{1,5,8}$ and $g_{2,5,8}$. The intersection occurs when $\theta = 0$ which corresponds to the point $\frac{1}{2}g_{1,5,8} + \frac{1}{2}g_{2,5,8}$. Since the set I is in the direction of positive x, then the grid point $g_{2,5,8}$ should be the one labelled independent. Using the formula in Section 3.3.2 with $\alpha = 1/2$ gives the equation

$$\Phi(g_{1,5,8}) \leftarrow -\Phi(g_{2,5,8}).$$

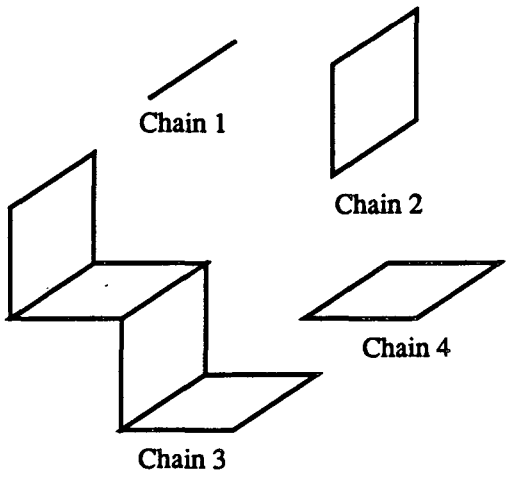


Figure 3.15: Example chains

Chain 2

This chain also consists of only one link, a pane connecting the grid points $g_{1,6,7}$, $g_{1,6,8}$, $g_{2,6,7}$, and $g_{2,6,8}$. The intersection point occurs at the point $(3/2, 6, (9+3\sqrt{5})/2)$. With respect to the grid, this corresponds to the point $(1-\alpha-\beta)g_{2,6,7} + \alpha g_{2,6,8} + \beta g_{1,6,7}$ where $\alpha = (3\sqrt{5}-5)/2$ and $\beta = 1/2$. As in the previous chain, the independent points should be $g_{2,6,7}$, and $g_{2,6,8}$. Plugging this into the equation in Section 3.3.2 gives the equation

$$\begin{bmatrix} \Phi(g_{1,6,7}) \\ \Phi(g_{1,6,8}) \end{bmatrix} \leftarrow \begin{bmatrix} 3\sqrt{5}-6 & 5-3\sqrt{5} \\ 3\sqrt{5}-7 & 6-3\sqrt{5} \end{bmatrix} \begin{bmatrix} \Phi(g_{2,6,7}) \\ \Phi(g_{2,6,8}) \end{bmatrix}.$$

Chain 3

This chain is made up of four connected panes. Choosing dependencies is similar to the previous one pane segment. In this case, the dependent points should be the grid points $g_{1,j,k}$ and the independent points should be $g_{2,j,k}$. It is easy to see that this chain is then consistent, i.e. no grid points are labelled both dependent and independent. Once the dependencies are established, then the calculation of the dependency matrices proceeds as in chain 2 to get the four matrix equations

$$\begin{bmatrix} \Phi(g_{1,7,7}) \\ \Phi(g_{1,7,8}) \end{bmatrix} \leftarrow A \begin{bmatrix} \Phi(g_{2,7,7}) \\ \Phi(g_{2,7,8}) \end{bmatrix}, \quad \begin{bmatrix} \Phi(g_{1,7,7}) \\ \Phi(g_{1,8,7}) \end{bmatrix} \leftarrow B \begin{bmatrix} \Phi(g_{2,7,7}) \\ \Phi(g_{2,8,7}) \end{bmatrix},$$

$$\begin{bmatrix} \Phi(g_{1,8,6}) \\ \Phi(g_{1,8,7}) \end{bmatrix} \leftarrow C \begin{bmatrix} \Phi(g_{2,8,6}) \\ \Phi(g_{2,8,7}) \end{bmatrix}, \quad \begin{bmatrix} \Phi(g_{1,8,6}) \\ \Phi(g_{1,9,6}) \end{bmatrix} \leftarrow D \begin{bmatrix} \Phi(g_{2,8,6}) \\ \Phi(g_{2,9,6}) \end{bmatrix}.$$

Let A have entries a_{ij}, and similarly for B, C, and D. Then the four matrix equations are combined

into the one matrix equation by averaging to get

$$\begin{bmatrix} \Phi(g_{1,7,8}) \\ \Phi(g_{1,7,7}) \\ \Phi(g_{1,8,7}) \\ \Phi(g_{1,8,6}) \\ \Phi(g_{1,9,6}) \end{bmatrix} \leftarrow \begin{bmatrix} a_{22} & a_{21} & 0 & 0 & 0 \\ \frac{a_{12}}{2} & \frac{a_{11}+b_{11}}{2} & \frac{b_{12}}{2} & 0 & 0 \\ 0 & \frac{b_{21}}{2} & \frac{b_{22}+c_{22}}{2} & \frac{c_{21}}{2} & 0 \\ 0 & 0 & \frac{c_{12}}{2} & \frac{c_{11}+d_{11}}{2} & \frac{d_{12}}{2} \\ 0 & 0 & 0 & d_{21} & d_{22} \end{bmatrix} \begin{bmatrix} \Phi(g_{2,7,8}) \\ \Phi(g_{2,7,7}) \\ \Phi(g_{2,8,7}) \\ \Phi(g_{2,8,6}) \\ \Phi(g_{2,9,6}) \end{bmatrix}. \quad (3.1)$$

Chain 4

The process for this single pane chain is identical to that of chain 2.

The remainder of the boundary can be constructed easily by using symmetry, or can be done the same way as above. In Section 3.3.8 a computer algorithm for computing these boundary conditions is described.

This example is fairly typical in that the final dependency matrix is in block diagonal form. Each block corresponds to each chain segment of the boundary. Provided that the boundary has at least one break in it (that is, separates into at least two chains), and that all chain segments have at least one consistent labelling, the dependency matrix can always be represented in this form. However, the matrix blocks are not always square, so the dependency matrix is not always square.

3.3.6 Two-Step Boundary Condition Process

One difficulty with creating the boundary conditions is that the final solution is implicitly assumed to be known before the boundary conditions are generated. Thus, finding the boundary conditions for a given boundary curve is a two-step process. First, an initial guess at the correct boundary conditions is made and the surface with those boundary conditions generated. The effect of poorly chosen boundary conditions results in small perturbations near the boundary, but the interior I of the surface will be close to the correct solution. Thus, the second step of the boundary generation process is to readjust the boundary using the computed surface as a good approximation to the correct solution. Once the boundary is properly adjusted, a better solution can be computed with the new boundary using the previously computed surface as the new initial surface.

3.3.7 Properties of Chains

Before any kind of computer-generated boundary conditions can be constructed, certain abstract properties of chains and the number of consistent labellings must be derived. Let a pane be represented by P and an edge by E. One representation of a chain is as an undirected graph with the nodes being the edges and panes. Two nodes are connected if the corresponding edges or panes share a common grid point. The chain in Figure 3.9 is depicted in graph form in Figure 3.16. Define a chain to have a *loop* if its graph representation contains a closed loop. In this section, assume that no chains form a loop. This assumption essentially guarantees that all chains have at least one

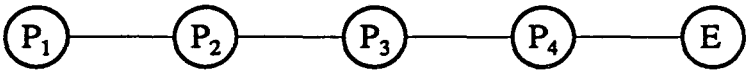


Figure 3.16: A sample graph representation

consistent labelling. By eliminating loops and using the continuity of Γ, the graph representation of chains reduces to a single string of nodes. Therefore, we can represent a chain as a sequence of P's and E's. For example, the chain in Figure 3.9 may be represented as $PPPPE$, or P^4E. Denote by C any string of P's and E's.

Let C be any chain segment which does not form a loop. Define the function $L(C)$ to be the number of consistent labellings allowed by the chain segment C. The following are a set of rules which describe how to compute $L(C)$.

Rule 1 $L(P^n) = 4$ *and* $L(E^n) = 2$ *for* $n > 0$.

We proceed by induction. Suppose that $C = P$. A consistent labelling for a single pane P corresponds to choosing a side of P to be the dependent side. Since P has four sides, then $L(P) = 4$. Now suppose $C = P^2$. The only way panes can be connected is if they share a common edge. Given a pane with one edge labelled, the rest of the labelling is uniquely determined. Therefore, for each of the four labellings of the first pane, the second pane's labelling is uniquely determined, hence there are no additional labellings. Therefore $L(P^2) = 4$. Now assume $C = P^nP$. By the inductive hypothesis, $L(P^n) = 4$. Since P^n and P share a common edge, then the labelling of P is uniquely determined by the labelling of P^n, therefore $L(P^{n+1}) = 4$.

The argument for edges is analogous. □

Rule 2 $L(CE) = L(EC) = L(C)$ *where* $C \neq \emptyset$.

The argument here is similar to the previous rule. The edge E and the chain C share a common gridpoint, therefore the labelling of E is uniquely determined by the labelling on C so $L(CE) = L(EC) = L(C)$. □

Rule 3 $L(C_1PE^nPC_2) = L(C_1PEPC_2)$ *and* $L(C_1EP^nEC_2) = L(C_1EPEC_2)$ *for* $n > 0$.

By Rule 2, $L(C_1PE^n) = L(C_1P) = L(C_1PE)$. The number of points in PC_2 labelled by C_1PE^n is one, similarly for C_1PE. The number of labellings for PC_2 given one point determined by the previous part of the chain is the same in either case. Therefore, $L(C_1PE^nPC_2) = L(C_1PEPC_2)$.

The proof for the second equation is similar. □

Rule 4 $L((PE)^nP) = 2^{n+2}$ *for* $n \geq 0$.

We proceed by induction. The case of $n = 0$ is given by Rule 1. For $n > 0$, by the inductive hypothesis, $L((PE)^{n-1}P) = 2^{n+1}$. By Rule 2, $L((PE)^n) = L((PE)^{n-1}(PE)) = L((PE)^{n-1}P)$.

The last edge of the chain $(PE)^n$ determines the labelling of one point of the final pane P. Given one determined point on a pane, there are two different consistent labellings which may be chosen. Thus, for each labelling of $(PE)^n$, there are two possible labellings of the final P which are consistent. Therefore, $L((PE)^nP) = 2L((PE)^n) = 2 \cdot 2^{n+1} = 2^{n+2}$. □

Rule one tells how many consistent labellings are possible on a string of strictly panes and a string of strictly edges. Rules two and three describe how to collapse a complicated chain into its simplest form of $(PE)^nP$. Rule four tells how to compute $L(C)$ for the collapsed chain.

For example, chain 3 above is represented as $C = PPPP$, so by rule one $L(C) = 4$. For a more complicated example, consider the chain $C = EPPEEPPPEPE$. By rule two, $L(C) = L(PPEEPPPEP)$. By rule three, $L(PPEEPPPEP) = L(PEPEP) = L((PE)^2P)$. Finally, by rule four, $L(C) = 2^{2+2} = 16$.

3.3.8 Automatic Generation of Chains

Unfortunately, because of the required two-step boundary condition process, an entirely automatic boundary generation procedure is not available. However, generating by hand the sometimes vast number of coefficients in the dependency coefficient matrix A can be tedious and prone to error. In order to create a computer-generated set of dependency coefficients, three issues must be resolved. First, the intersection points between Γ and the grid must be located. Second, for determining optimal dependencies, at each point $\gamma(s) \in \Gamma$ the vector $\tau(s)$ where $\tau(s)$ is orthogonal to Γ and tangent to and in the direction of the interior surface set I at $\gamma(s)$ must be computed. A diagram of the relationship between τ, Γ, E, and I is shown in Figure 3.17.

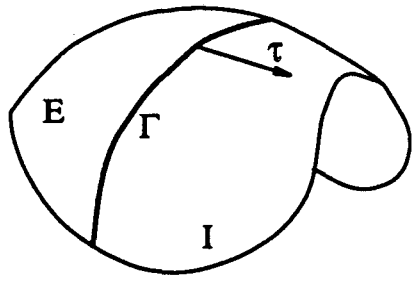


Figure 3.17: Relationship between τ, E, and I

Finally, after the choices for optimal dependencies are made, the chains of conditions must be reconciled so that all chains are consistent.

Finding the Intersection Points

Finding the intersection points between Γ and the grid essentially distills into a local root-finding problem. Let $\gamma(s) = (\gamma_1(s), \gamma_2(s), \gamma_3(s))$, then the function for which a root is to be found

is given by

$$f(s) = (\gamma_1(s) \bmod \Delta x)(\gamma_2(s) \bmod \Delta y)(\gamma_3(s) \bmod \Delta z).$$

The best way to solve this problem is with a bisection algorithm.

Instead of looking at individual grid points, label each cube of grid points by $c_{i,j,k}$ where $g_{i,j,k}$ is the bottom left corner of the cube. Thus, the cube $c_{i,j,k}$ consists of the grid points $g_{i,j,k}$, $g_{i+1,j,k}$, $g_{i,j+1,k}$, $g_{i+1,j+1,k}$, $g_{i,j,k+1}$, $g_{i+1,j,k+1}$, $g_{i,j+1,k+1}$, and $g_{i+1,j+1,k+1}$. Denote by $C(x)$ the function which takes a point in space and returns the cube coordinates which contain x and denote two distance functions by

$$\begin{aligned} d_\infty(c_{i,j,k}, c_{l,m,n}) &= \max(|i-l|, |j-m|, |k-n|) \\ d_1(c_{i,j,k}, c_{l,m,n}) &= |i-l| + |j-m| + |k-n|. \end{aligned}$$

Now, note that the "zeros" that are sought occur in an interval $[s_1, s_2]$ when $C(\gamma(s_1)) \neq C(\gamma(s_2))$. By using the method of bisection, this interval can be divided until the subinterval $[t_1, t_2]$ is found where $d_\infty(C(\gamma(t_1)), C(\gamma(t_2))) = 1$. At this point either bisection can be continued to find the location of the zero, or a more sophisticated algorithm can be used such as accelerating the bisection process by using straight line approximations to γ, or Newton's method. Since Newton's method requires the computation of the derivative of γ, it is suggested that bisection or accelerated bisection be used to cut down on the total input. Once the location t_0 of the zero is found, then the type of intersection is determined by computing $d_1(C(t_0 - \epsilon), C(t_0 + \epsilon))$. Here, ϵ may be half of the distance between the final pair of straddling values of t which are computed during the bisection process. The type of intersection is given in Table 3.1.

Value of d_1	Intersection Type
1	Pane
2	Edge
3	Zero-point

Table 3.1: Values of d_1 vs. intersection types

One final issue must be addressed with regard to finding the intersection points. Because of round-off errors the root finding section might find an edge even though $\gamma(s)$ theoretically passes through a zero point. If this happens and the wrong endpoint is chosen to be dependent, then it is possible to get an overflow when computing the dependency coefficient. Because of this, a reduction procedure is employed which first reduces a pane to an edge if the intersection point $\gamma(t_0)$ is within ϵ of an edge, and then further reduces an edge to a zero point if the intersection point is within ϵ of an endpoint of the edge. For the boundaries used at the end of chapter 4, ϵ was chosen to be 10^{-10}.

Approximation of τ

Recall that the vector τ is the vector which essentially points in the direction of the interior set I orthogonal to Γ. Before describing how to compute an approximation for τ it is important to understand how τ will be used to determine grid dependencies. For now, assume that the vector $\tau(s)$ is given. For a zero point, the vector τ is not necessary because the dependency coefficient is determined automatically. The cases of an edge and a pane will be handled separately.

For an edge E, let the endpoints of the edge be denoted by g_1 and g_2. To determine the dependent end, treat $\gamma(s)$ as a vector and orthogonally project the vector $\gamma(s) + \tau(s)$ onto the line determined by E. Call the projection point $\bar{\tau}$. An optimal dependent end satisfies the relation $(\bar{\tau} - \gamma(s)) \cdot (g_i - \gamma(s)) \leq 0$. This construction is displayed in Figure 3.18.

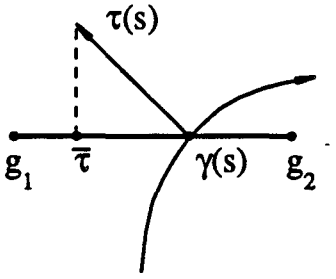


Figure 3.18: Edge dependency diagram

For a pane P, let the endpoints be denoted by g_1, g_2, g_3, and g_4 as in Figure 3.19. In this

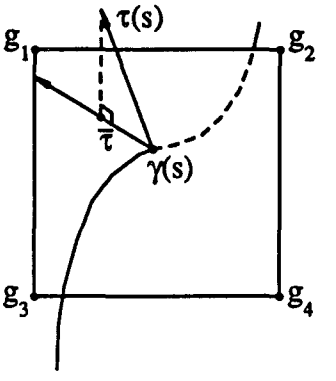


Figure 3.19: Pane dependency diagram

case, an optimal edge of P is to be chosen. Similar to the edge, project $\gamma(s) + \tau(s)$ orthogonally onto the plane determined by P and call the projection vector $\bar{\tau}$. Since $\gamma(s)$ is inside P, then there is a $t > 0$ such that $\gamma(s) + t(\bar{\tau} - \gamma(s)) \in \partial P$. An optimal independent edge is one which contains the point $\gamma(s) + t(\bar{\tau} - \gamma(s))$. In the unlikely event that $\bar{\tau} = \gamma(s)$, then all edges are treated as optimal.

The process of computing $\tau(s)$ *a priori* is difficult because of the non-uniqueness property

of minimal surfaces for most boundaries. It was noted in Section 3.3.6 that boundary condition generation is a two step process. These two steps correspond to the two ways of approximating τ. The first is used to generate the preliminary boundary conditions and the second is used to generate the corrected boundary conditions.

Experimentation with several different prediction methods was performed. For surfaces that are extremal, the most success was achieved by approximating $\tau(s) \approx \bar{\gamma} - \gamma(s)$ where $\bar{\gamma}$ is the center of mass of Γ. For more complicated surfaces, since the local surface may be approximated by a plane, τ may be crudely approximated by the acceleration vector $\ddot{\gamma}(s)$.

In order to correct the boundary conditions, an approximation of the minimal surface must be available. Consider a pane which is part of the boundary. Given a function Φ for which the minimal surface approximation is $S = \Phi^{-1}(0)$, there are three places on the pane where the surface can be determined: $\gamma(s)$ and the two points ι, $\eta \in S \cap \partial P$ (see Figure 3.20). Let n be normal to

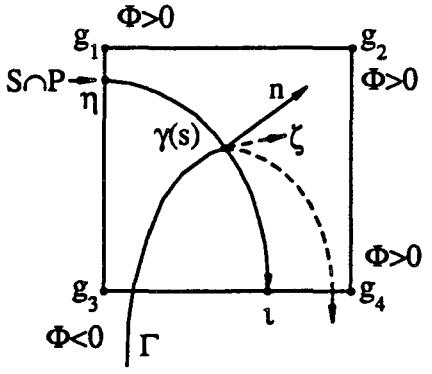


Figure 3.20: Diagram of $S \cap \partial P$

$S \cap P$ in the direction of $\nabla\Phi$. Since $\nabla\Phi|_S \neq 0$, then a natural orientation is induced upon S which makes it possible to locally distinguish between the sets I and E. In order to make the distinction, let ζ be orthogonal to P with $\zeta \cdot \dot{\gamma}(s) > 0$. Then the interior point ι is the point nearest to $\pm n \times \zeta$ where the sign depends on the direction of the parameterization of $\gamma(s)$ around the interior surface I. The optimal independent edge is the one which contains ι, thus $\tau(s) \approx \iota - \gamma(s)$. At the same time an optimal (external) dependent edge is found, the one containing the exterior point η. The optimal independent and dependent edge may conflict, but that is resolved during the construction of the chains.

For an edge, the correction algorithm is somewhat more complicated. The object is to choose the endpoint that is closer to the interior region I. In this case, let the edge be given by the segment g_1g_2 and consider the four panes that have g_1g_2 as an edge. The orientation induced for panes given above picks out two of those panes has having the exterior point η on g_1g_2 as shown in Figure 3.21. For each of the two panes P_1 and P_2 calculate ξ_i as follows. Each pane will now determine two points that are on the surface S, $\gamma(s)$ on the edge g_1g_2 and ι_i on one of the other

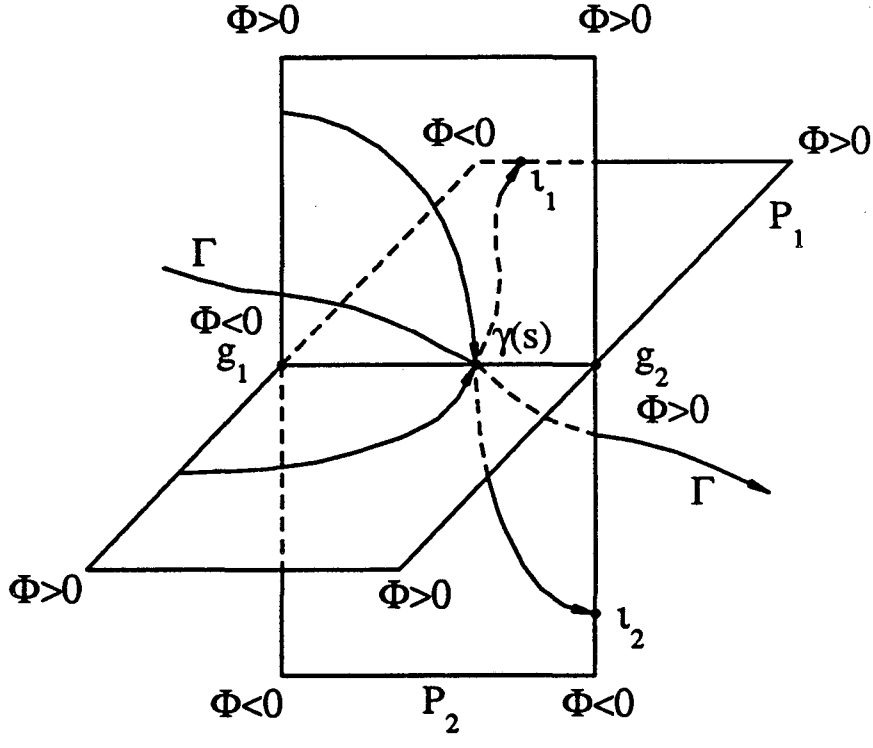


Figure 3.21: Panes intersecting an edge

edges. If ι_i is on the opposite edge, then ξ_i is the orthogonal projection of $\iota_i - \gamma(s)$ onto g_1g_2. If ι_i is not on the opposite edge, ξ_i is the orthogonal projection of the point where the line through $\gamma(s)$ and ι_i intersects the line determined by the opposite edge of P_i. The point g_j nearest to $\bar{\xi} = (\xi_1 + \xi_2)/2$ is selected as the point nearest I. Thus, $\tau(s) \approx g_j - \gamma(s)$.

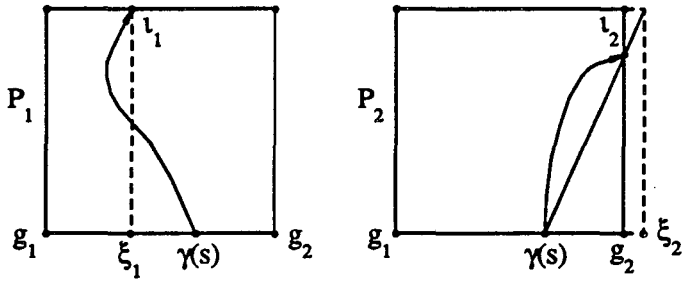


Figure 3.22: Construction of ξ_i

Chain Consistency

Finally, once $\tau(s)$ is approximated for each chain link, the chains must be constructed so that they are consistent. To do this, the total number of possible combinations must be found using the properties of chains in Section 3.3.7. For each of the consistent labellings, a score is given according to how well the chain is labelled. A typical way to score a chain is to assign a value to each element of the chain according to the choice of dependent points versus the optimal. One such scoring system is shown in Table 3.2.

	Dependent points are...	Score
Panes	opposite optimal interior edge	0
	on optimal exterior edge (correction method only)	1
	on optimal interior edge	100
	anywhere else	5
Edges	optimal	0
	not optimal	3

Table 3.2: Chain scoring table

The chain labelling that achieves the lowest score is the one that is used. Chains that score greater than 99 should be cause for concern. When chains score greater than 99, it means either dependent points are straddling the surface which could lead to inaccuracies at the boundary, or a very long chain exists. Particularly in the first case, it is important to reconfigure the grid to avoid such chains whenever possible.

3.4 Reinitialization and the "Tentpole Phenomenon"

Using curvature flow with boundary conditions to anchor the surface to the boundary contour produces a very simple first attempt at an algorithm for computing minimal surfaces. This basic algorithm is shown again here.

Algorithm 1

Step 1. Move the surface according to curvature flow.
Step 2. Reattach the surface to the boundary.
Step 3. Go to step 1.

Unfortunately, algorithm 1 doesn't work because of inaccuracy at the boundary. To resolve the inaccuracy, some investigation into the theory of curvature flow is necessary. For mean curvature flow without boundaries using the level set representation, Evans and Spruck [7] proved that the level sets of the function Φ do not change distance relative to each other as time progresses. When boundaries are introduced, however, this is not true. Instead, a structure resembling a tentpole can appear as in Figure 3.23.

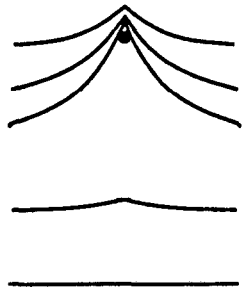


Figure 3.23: The tentpole phenomenon

Unless the surface $S = \Phi^{-1}(0)$ is extended so that all the level sets near the boundary have zero mean curvature, the level sets on one side of S will collapse together onto the boundary, while on the other side they separate and flow away from the boundary. In the wake of the boundary, as the level sets flow downstream, eventually catastrophic round-off affects the computation of the curvature near the boundary and instability ensues.

To avoid the tentpole phenomenon the surfaces are reinitialized. At regular intervals, the surface $\Phi^{-1}(0)$ is located and the signed distance from $\Phi^{-1}(0)$ to each point is computed. The formula for reinitialization can be expressed as

$$\Phi(x) \leftarrow \text{sign}(\Phi(x))\text{dist}(x, \Phi^{-1}(0)) \tag{3.2}$$

The process of reinitialization effectively moves the nonzero level sets so that they are equally spaced as they would be for flow without boundary. The zero level set remains fixed.

Adding reinitialization to algorithm 1 produces the final algorithm for computing minimal surfaces:

Algorithm 2

Step 1. Move by curvature flow
Step 2. Reattach the surface to the boundary
Step 3. Reinitialize Φ
Step 4. Go to step one.

3.4.1 Reducing Computing Costs

Considered as separate pieces, the steps for computing minimal surfaces given above can be computationally expensive. In the simplest case, one time step in curvature flow is an $O(n^3)$ operation, reattaching the boundary is $O(n)$, and reinitializing the surface is $O(n^6)$. To see that reinitialization is of order $O(n^6)$, note that it is an $O(n^3)$ operation to locate a level set on a grid, and an additional $O(n^3)$ operation to compute the distance from a point on the surface to each point on the grid. Even for relatively small values of n, an $O(n^6)$ order computation done every time step is for all practical purposes too costly.

By combining the three steps and making one observation, the overall computing cost can be decreased dramatically. To begin, note that with the introduction of reinitialization, the values of Φ outside of a neighborhood of $\Phi^{-1}(0)$ are extraneous and have no effect on the motion of the set $\Phi^{-1}(0)$. The only contribution of points outside the neighborhood is to hold a sign for the use of locating the zero level set. This observation leads to the conclusion that the only points that need to be reinitialized are those inside of the neighborhood of $\Phi^{-1}(0)$ that are used in computing the motion for $\Phi^{-1}(0)$.

Define the *stencil* of the surface to be those points. More specifically, a point g is in the interior of the stencil if it has a neighboring point g', for which $\Phi(g)\Phi(g') \leq 0$. A point g is in the stencil if there is a point g' in the interior of the stencil and for which $d_\infty(g, g') \leq 1$. In other words, a point g is in the stencil if it is next to the surface, or is one of the 26 grid points surrounding a point next to the surface. A diagram of the 2-dimensional case is shown in Figure 3.24.

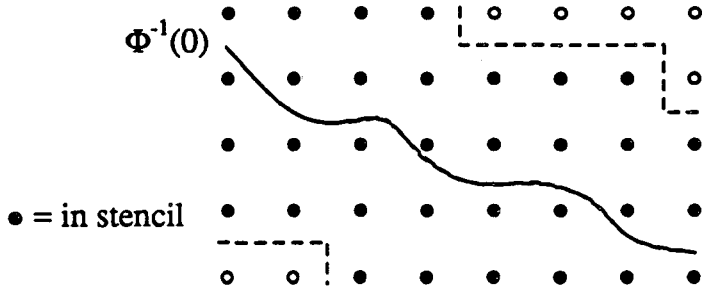


Figure 3.24: A 2-dimensional stencil diagram

By restricting reinitialization to the stencil of the surface, then the computational cost of reinitialization drops from $O(n^6)$ to $O(n^3)$. To see this reduction note that if restricted to the stencil,

the distance need only be computed for a fixed size neighborhood of each point, since only points near the surface are going to be reinitialized. Thus, the cost of reinitialization reduces to finding the surface and then computing for each point near the surface the distance to a fixed number of points (26) independent of n. Therefore, the total cost of reinitialization decreases to $O(n^3)$.

Furthermore, since the values of Φ outside of the stencil are only sign holders, it is not necessary to move Φ outside of the stencil. Thus, during reinitialization, a list of the points in the stencil is created and passed to the curvature flow step. The curvature flow subroutine computes the curvature motion for all points in the *interior* of the stencil. This allows the curvature flow step cost to drop from $O(n^3)$ for computing Φ_t everywhere, to $O(n^2)$ for computing Φ_t only within the stencil.

One final side benefit is gained by restricting computation to the stencil. It now no longer matters what type of boundary conditions are used for the boundary of the grid. Since computation is only done within the stencil, only the intersection of the stencil with the grid boundary can influence the motion of Φ. However, perturbations introduced by the grid boundary are stopped entirely by the contour boundary before they can affect the interior portion I. Thus, it is equally acceptable to use one sided differences at the grid boundary or simply assume $\Phi_t = 0$ on the grid boundary.

3.5 Initialization and Stopping Criterion

Two important issues remain: how to start and how to end the algorithm. For initialization, the object is to find a function $\Phi : R^3 \rightarrow R$ such that the boundary contour $\Gamma \subset \Phi^{-1}(0)$ and that $\nabla\Phi \neq 0$ on the surface $\Phi^{-1}(0)$. If a minimal surface of a particular topological type is sought, then the initial surface should have the same topological type. If no solution exists of that topological type, the algorithm will search for a different solution.

While several different initial surfaces are possible, in order to avoid the tentpole problem, a surface which has a small curvature on Γ is preferable. For example, if Γ consists of two rings as for the catenoid, then an initial surface of an infinite cylinder is better than an initial surface of a finite cylinder with disks on the ends.

The stopping criterion for the algorithm is a test of whether steady state is achieved. The test can be inserted anywhere in the main loop, but the best place is to put it directly after step two, reattaching the boundary. At that point a simple check of $L_\infty(\Phi^{n+2/3}, \Phi^{n+1+2/3}) < \epsilon$ determines whether steady state is achieved.

3.6 Stability and the Boundary

During a computation, the surface must remain attached to the boundary. If the surface moves too far during one time step, separation from the boundary can occur, similar to a Courant

condition. Once separation from the boundary happens at any point, it often follows rapidly along the rest of the boundary until the surface pulls completely away. By reducing the time step so that the surface moves no more than one grid cell, this can be avoided. In the next chapter is a discussion of time step sizes necessary for stability.

3.7 Summary

In this chapter, a new algorithm for computing minimal surfaces has been introduced. It is given again below with the basic equations listed.

Algorithm 2

Step 1. Move Φ with speed equal to the numerator of the local mean curvature.

$$\Phi^{n+1/3} \leftarrow \Phi^n + \Delta t K(\Phi^n)$$

Step 2. Reattach Φ to the boundary contour.

$$\Phi^{n+2/3} \leftarrow A\Phi^{n+1/3}$$

Step 3. Reinitialize.

$$\Phi^{n+1} \leftarrow \text{sign}(\Phi^{n+2/3})\text{dist}(\,\cdot\,, \Phi^{n+2/3\,-1}(0))$$

Step 4. Go to step one.

Chapter 4

Numerical Results

The algorithm presented in the previous chapter was designed to meet the following goals: it should produce accurate results with at least linear convergence and it should be able to naturally change topology without special intervention by the user. The first goal will be addressed in Section 4.1, and the second in Section 4.2.

4.1 Convergence

From tests against known solutions, the convergence of the algorithm to the exact solution appears to be nearly linear. In Figure 4.1 is shown a sequence of grid refinements and the exact solution for the radius of Euler's catenoid solution with $a = 0.4$, $b = 0.277259$, and $R = 0.5$. Table 4.1 shows the absolute error in the radius measured in the L^1, L^2, and L^∞ norms.

Grid Size	L^1	L^2	L^∞
$9 \times 17 \times 17$	$8.64E-02$	$1.63E-03$	$2.37E-02$
$17 \times 33 \times 33$	$6.13E-02$	$4.27E-04$	$9.33E-03$
$26 \times 46 \times 46$	$7.42E-02$	$3.34E-04$	$6.13E-03$

Table 4.1: Computed error table

4.2 Changes in Topology

Figures 4.4, 4.8, and 4.9 show how topological changes can occur. However, the topology can also be complex if a solution of that type exists as in Figure 4.16.

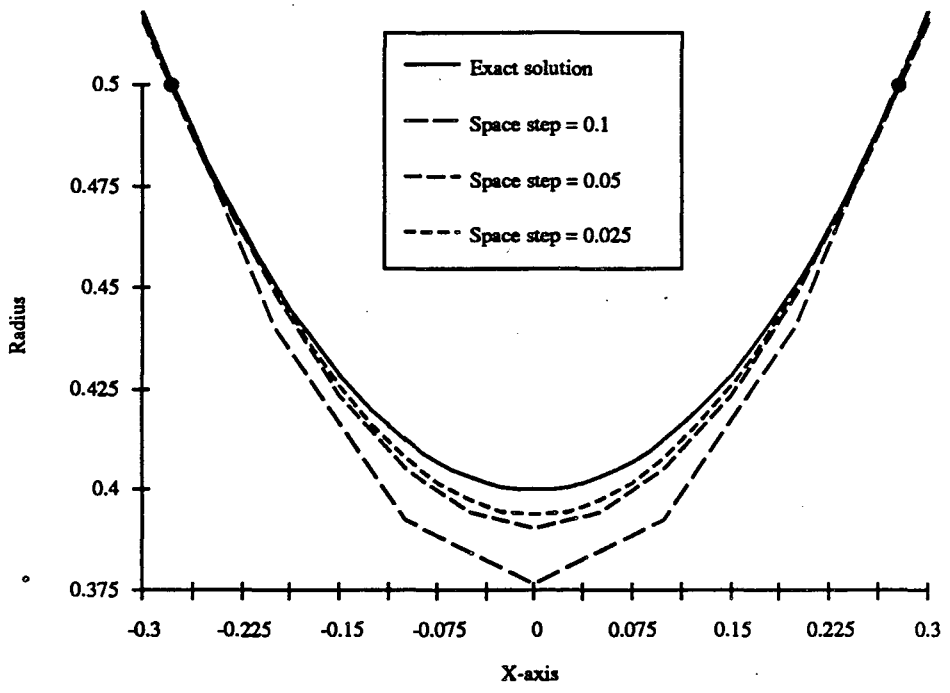


Figure 4.1: Convergence of solutions under grid refinement.

4.3 Computational Cost

As was noted last chapter, the computational cost of each time step is $O(n^3)$, where n is the number of grid points in each coordinate direction. Table 4.2 gives a comparison of the computational cost of the catenoid problem for various grid mesh sizes. A graph of this table shows

Grid Size	Total Grid Points	CPU Seconds
$9 \times 17 \times 17$	2,601	47.6
$17 \times 33 \times 33$	18,513	233.7
$27 \times 47 \times 47$	59,643	867.9
$29 \times 51 \times 51$	75,429	1082.8
$31 \times 57 \times 57$	100,719	1256.0
$35 \times 63 \times 63$	138,915	1572.1
$41 \times 73 \times 73$	218,489	3327.9
$49 \times 87 \times 87$	370,881	5673.2

Table 4.2: Computational cost for 50,000 iterations

more clearly the linear relationship between the total number of grid points and the computational cost.

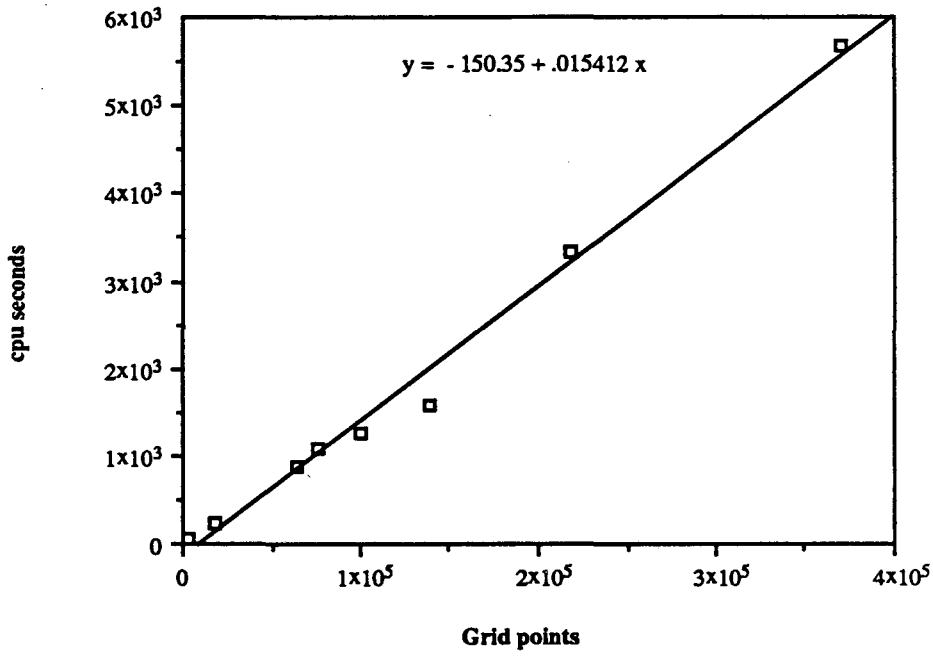


Figure 4.2: Graph of Computational Cost

4.4 Graphics Rendering and Triangle Trimming

In the next section a number of examples of minimal surfaces are displayed. In order to generate those figures, a set of triangles which approximate a level set were constructed. A triangle filter was then used to remove all triangles contained in the exterior set E. The remaining triangles were then rendered on a Stardent ST3000 computer. All computations were done within the Stardent graphics system shell called AVS (Application Visualization System).

4.4.1 Marching Cubes

In order to generate the level sets, a marching cubes (see [12]) type of algorithm was used. This is a standard method for rendering level sets of a function defined on a 3–dimensional grid. The general algorithm addresses many important graphics problems needed for a complete rendering package. In this section is a summary of only a part of that algorithm, the construction of triangles.

This part of the marching cubes algorithm consists of a small loop with a large number of cases. A list of triangles is then constructed which forms an approximation to the surface $\Phi^{-1}(0)$. To illustrate, as was done in Section 3.3.8, represent the grid as a collection of cubes labelled by the index of a corner grid point. Each point g in the grid is assigned the value of

$$f(g) = \begin{cases} 1 & \text{if } \Phi(g) \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

Given an arbitrary cube c_{ijk}, label the vertices numbers v_1–v_8. Next, define the function

$$F(c_{ijk}) = \sum_{m=1}^{8} 2^{m-1} f(v_m) \tag{4.2}$$

The main loop of the marching cubes algorithm searches for any cube c_{ijk} in the grid for which $F(c_{ijk}) \neq 0$ or 255. This indicates that $\Phi^{-1}(0)$ intersects that cube.

For each intersection there are 2^8 different possible combinations of 1's and 0's corresponding the 256 values of the function F. By exploiting symmetry arguments, the 256 different cases can be reduced to 14. Each case will add up to four triangles to the triangle list. The triangle vertices are given in real coordinates and can then be projected onto the 2-dimensional surface of a monitor screen.

4.4.2 Triangle Trimming Filter

The results of this standard approach were not satisfactory. The problem is that the only portion of the surface $\Phi^{-1}(0)$ that is of interest is the interior set I, but the standard approach in the previous section displayed $I \cup E$. What is needed is a filter which will throw out the triangles which are contained in E. An algorithm suggested by William Johnston of Lawrence Berkeley Laboratory is to start on the outside boundary of the grid, then do a recursive search inward towards the boundary, throwing away triangles until the boundary is hit.

For input this graphics module receives the values of Φ on the grid, the space step of the grid, and also the real coordinates of a large number of points on the boundary contour Γ. The first step of the algorithm is to mark each of the grid cubes as either boundary or non-boundary. Then the main triangle search and destroy algorithm looks for a starting point by scanning the cubes on the outside edge of the grid until it finds one for which $F \neq 0$ or 255. At each starting point a recursive search algorithm is implemented as in algorithm 3.

Algorithm 3

Subroutine Search(i, j, k)
If c_{ijk} is outside the grid then return
If c_{ijk} is a boundary cell then return
If $F(c_{ijk}) = 0$ or $F(c_{ijk}) = 255$ then return
Remove all triangles in cell c_{ijk}
Mark c_{ijk} as a boundary cell (to prevent an infinite loop)
Call Search$(i+1, j, k)$
Call Search$(i-1, j, k)$
Call Search$(i, j+1, k)$
Call Search$(i, j-1, k)$
Call Search$(i, j, k+1)$
Call Search$(i, j, k-1)$
Return

This algorithm leaves all the triangles in the cubes containing the boundary unchanged, however, so the pictures displayed have fringes around the boundary. Further pursuit of the triangle trimming algorithm to clean off the fringes is beyond the scope of this paper.

4.5 Examples of Computed Surfaces

The following pages contain pictures of a number of minimal surfaces computed by this algorithm as well as some sample evolutions demonstrating changes of topology.

In Figure 4.3, the computed solution of Euler's catenoid surface is shown. The rings are of radius 0.5 and positioned at $x = \pm 0.277259$. The radius at the center should be approximately 0.4. The mesh size is $27 \times 47 \times 47$ with space step 0.025 in all directions. The initial surface consisted of a cylinder of radius 0.5.

In Figure 4.4 is shown what happens if the initial surface is chosen to be a cylinder as in Figure 4.3, but the rings are too far apart for a catenoid solution to exist. In this case, the topology changes so that instead of a cylinder type surface, two disks are found as the solution. For this surface, the rings have radius 0.5 and are positioned at $x = \pm 0.345$, and the mesh size is $41 \times 41 \times 41$ with space step 0.05.

In Figure 4.5, the computed solution of a catenoid type of surface with square ends is shown from different angles. The squares have side length 1.0 and are positioned at $x = \pm 0.275$. The mesh size is $41 \times 41 \times 41$ with space step 0.05. The initial surface was a square tube of side length 1.0.

In Figure 4.6, a cylinder type of surface is shown where the circles are parallel, but with offset centers. The circles are of radius 0.5 with the centers located at $\pm(0.2625, 0.0, 0.25)$ parallel to the yz-plane. The mesh has dimensions $25 \times 45 \times 67$ with space step 0.025 in all directions. The initial surface was a cylinder with oval cross-section and principle radii 1/2 and 21/58.

In Figure 4.7, the boundary consists of three circles with centers equally spaced on a base circle of radius 2/3 on the xy-plane. The circles each have radius 2/3. The mesh has dimensions $41 \times 41 \times 41$ with space step 0.05. The initial surface was three half-cylinders joined in the center.

In Figure 4.8, the boundary consists of three circles with centers equally spaced on a base circle of radius 0.755 on the xy-plane. The circles each have radius 0.51. The mesh has dimensions $41 \times 51 \times 25$ with space step 0.05. The initial surface was three half-cylinders joined in the center. In this case the rings are too far apart to have a connected minimal surface, hence the surface breaks into three disks.

In Figure 4.9, the boundary consists of six squares of side length 1/2, centered on the coordinate axes. On the x-axis the squares are located at ± 0.375, on the y-axis at ± 0.775 and on the z-axis at ± 1.275. The different distances were chosen to cause the surface to break at three different times. The mesh has dimensions $25 \times 41 \times 61$ with uniform space step 0.05. The initial surface was the union of three cylinders with square cross section.

In Figure 4.10, the surface computed is known as Sherck's surface. The boundary consists of eight line segments of unit length. The mesh has dimensions $42 \times 42 \times 42$ with space step 2/41 in all directions. The initial surface consisted of the two parallel half-planes $\{(x, y, z) : y = \pm 0.5, z \leq 0.5\}$ and the strip $\{(x, y, z) : -0.5 \leq y \leq 0.5, z = 0.5\}$.

In Figure 4.11, the boundary is formed by the edge of a twisted rectangle. The width of the rectangle is 1, and the flat ends have length 0.5. The center twisted portion has length 0.5 as well. the grid dimensions are 42 x 42 x 42 with space step 2/41. The initial surface was the twisted strip from which the boundary was derived extended out to the edge of the grid.

In Figure 4.12, the boundary is an oval mapped onto the surface of a cylinder. The exact equation for the boundary curve is given by

$$\gamma(s) = \left(\frac{1}{2}\cos(s) + \frac{1}{40}, \sin\left(\frac{3}{2}\sin(s)\right), \cos\left(\frac{3}{2}\sin(s)\right)\right).$$

The dimensions of the grid are 46 x 46 x 46 with space step 2/45. The initial surface was a cylinder of radius 1/2.

In Figure 4.13, the boundary consists of two parallel "bowties" located at $x = \pm 0.1375$, rotated by $\pi/6$ with respect to each other. The bowties consist of two opposite quarter arcs of radius 1/2 along with the connecting diameter lines. The grid size is 25 x 49 x 49 with space step 0.025. This surface demonstrates the ability of the algorithm to handle singularities in the boundary contour. Two more examples of contour singularities follow.

In Figure 4.14, the boundary consists of a square of side length 1/2 on one end and two squares of diagonal length 1/2 rotated by $\pi/4$ and offset so that the corners touch on the x-axis. The two ends are located at $x = \pm 0.1375$. The mesh dimensions are 25 x 25 x 49 with space step 0.025.

In Figure 4.15, the boundary consists of a square of side length 1/2 on one end and a "pinwheel" of the same dimensions.. The two ends are parallel centered at $x = \pm 0.1375$. The grid dimensions are 25 x 45 x 45 with space step 0.025.

In Figure 4.16, the boundary consists of three parallel squares. The outer two squares have side length 1 and the middle square has side length 2. The mesh dimensions are 25 x 47 x 47 with space step 0.025. This surface is very similar to the Meeks-Yao free minimal surface. Through the surface there are four holes, two through each of the smaller end squares. The initial surface had the same topology as the final shape. This example demonstrates that topologically complex surfaces are not always changed so long as a minimal surface of that type exists.

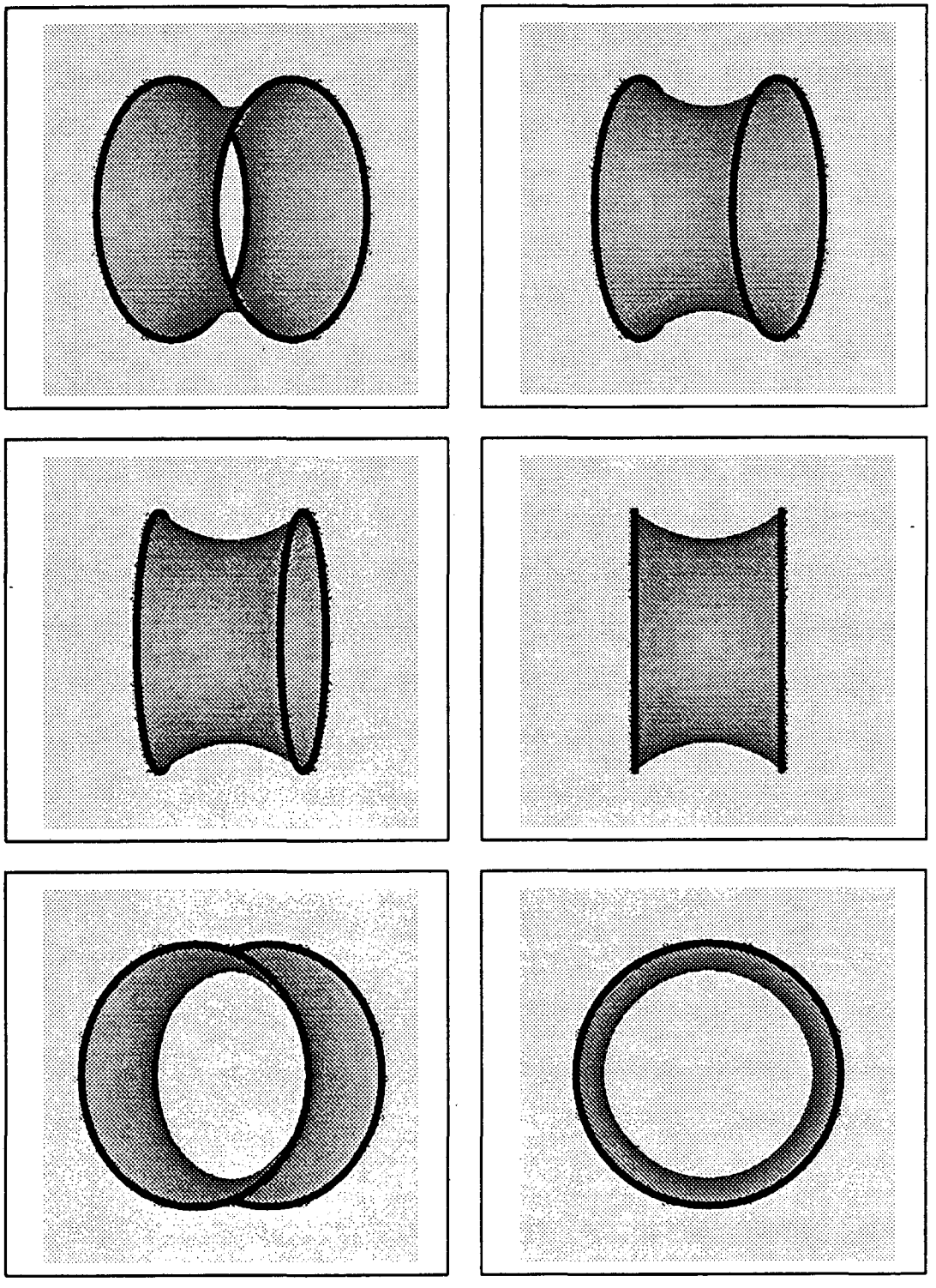


Figure 4.3: Euler's Catenoid Surface

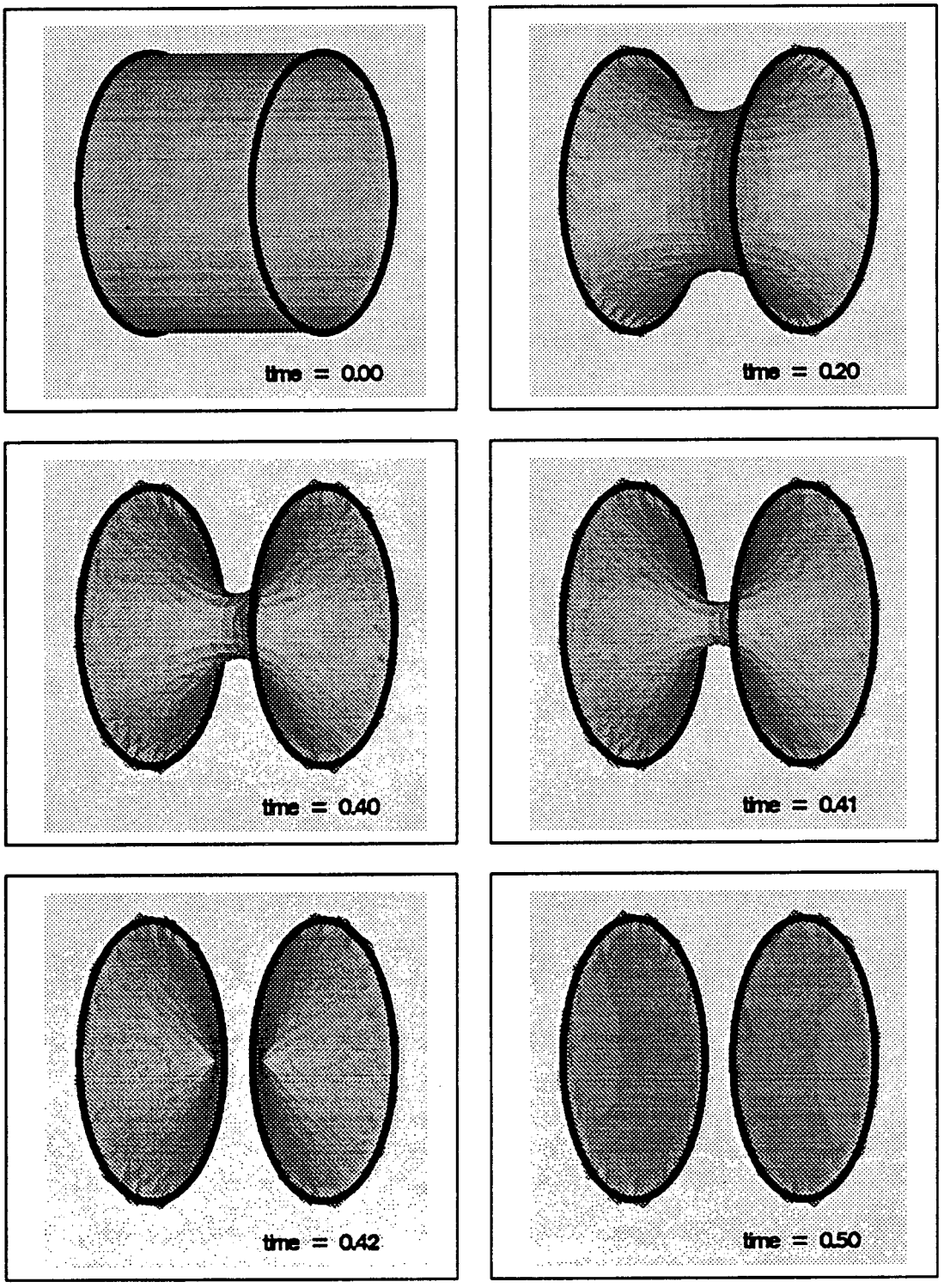


Figure 4.4: Splitting Catenoid Evolution

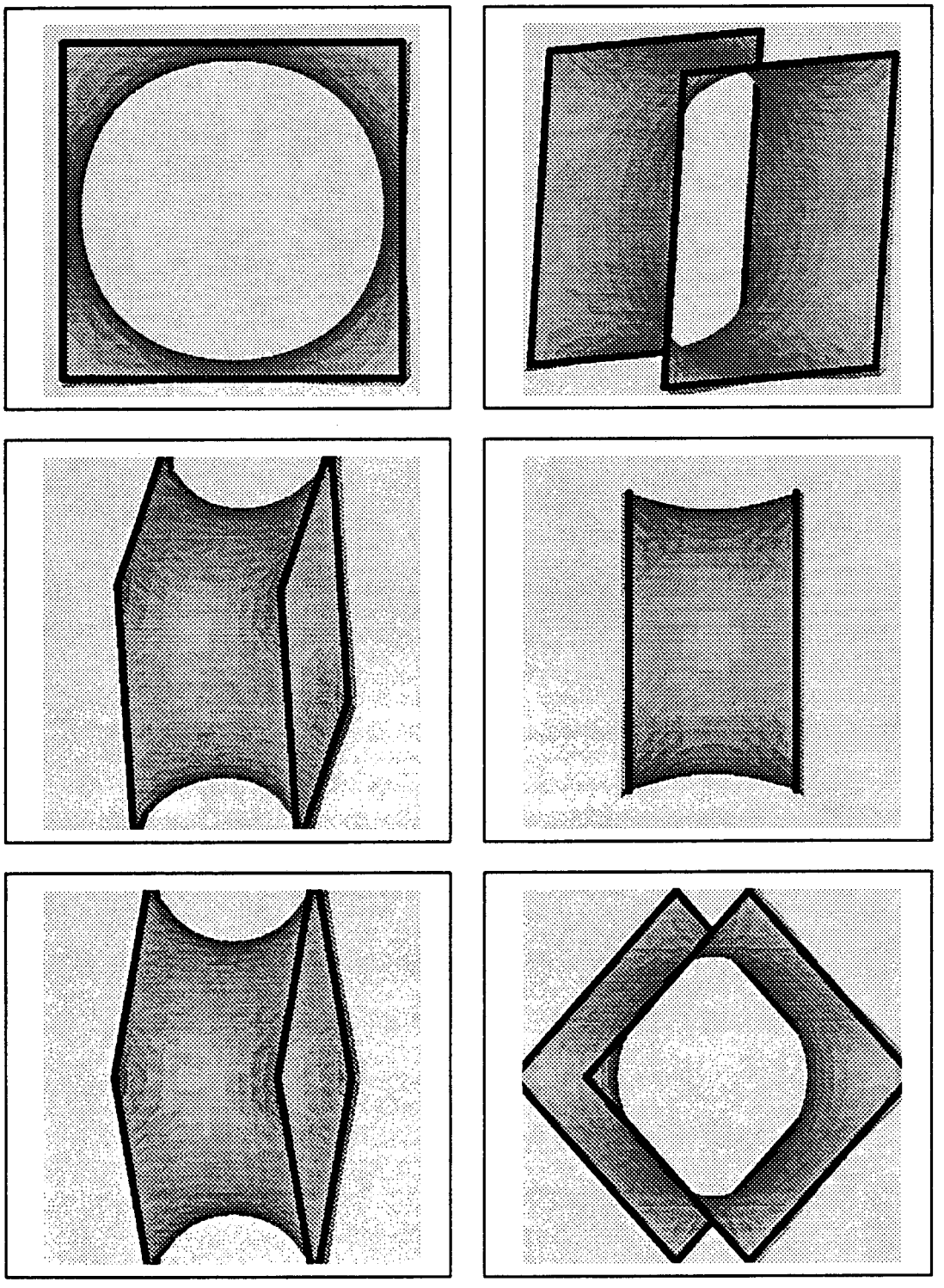


Figure 4.5: Square Catenoid Surface

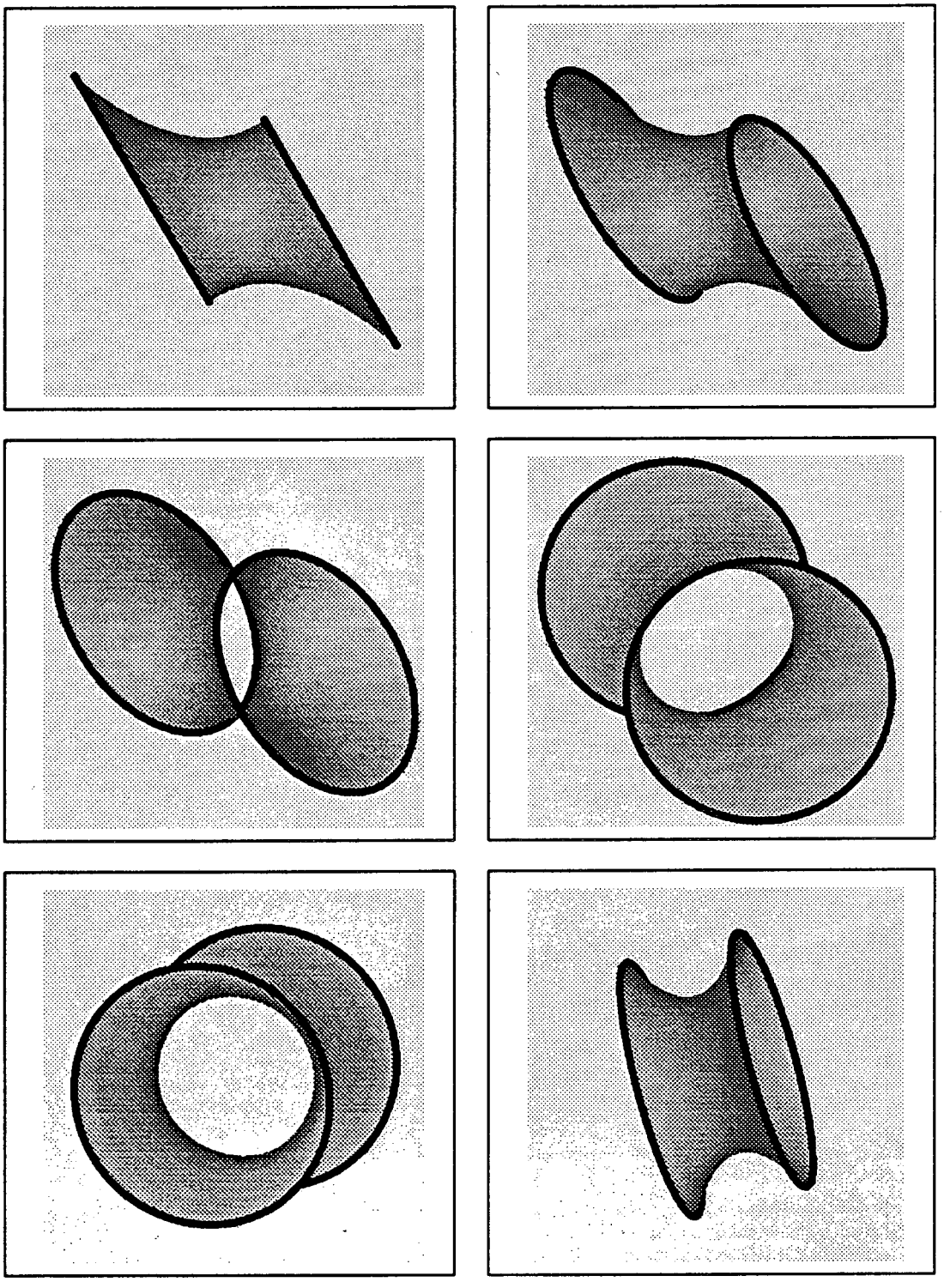


Figure 4.6: Offset Circles Surface

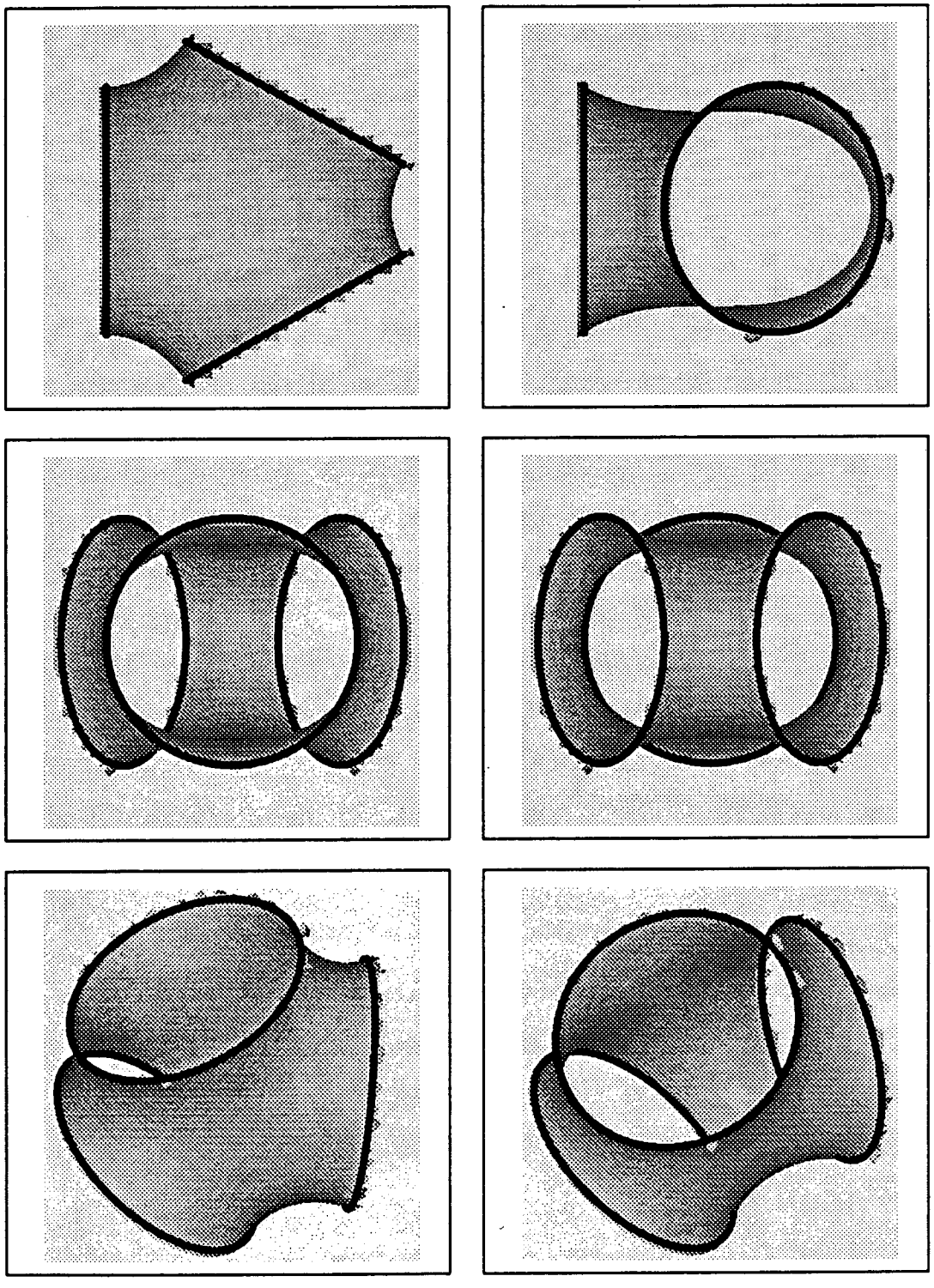


Figure 4.7: Three Circles Surface

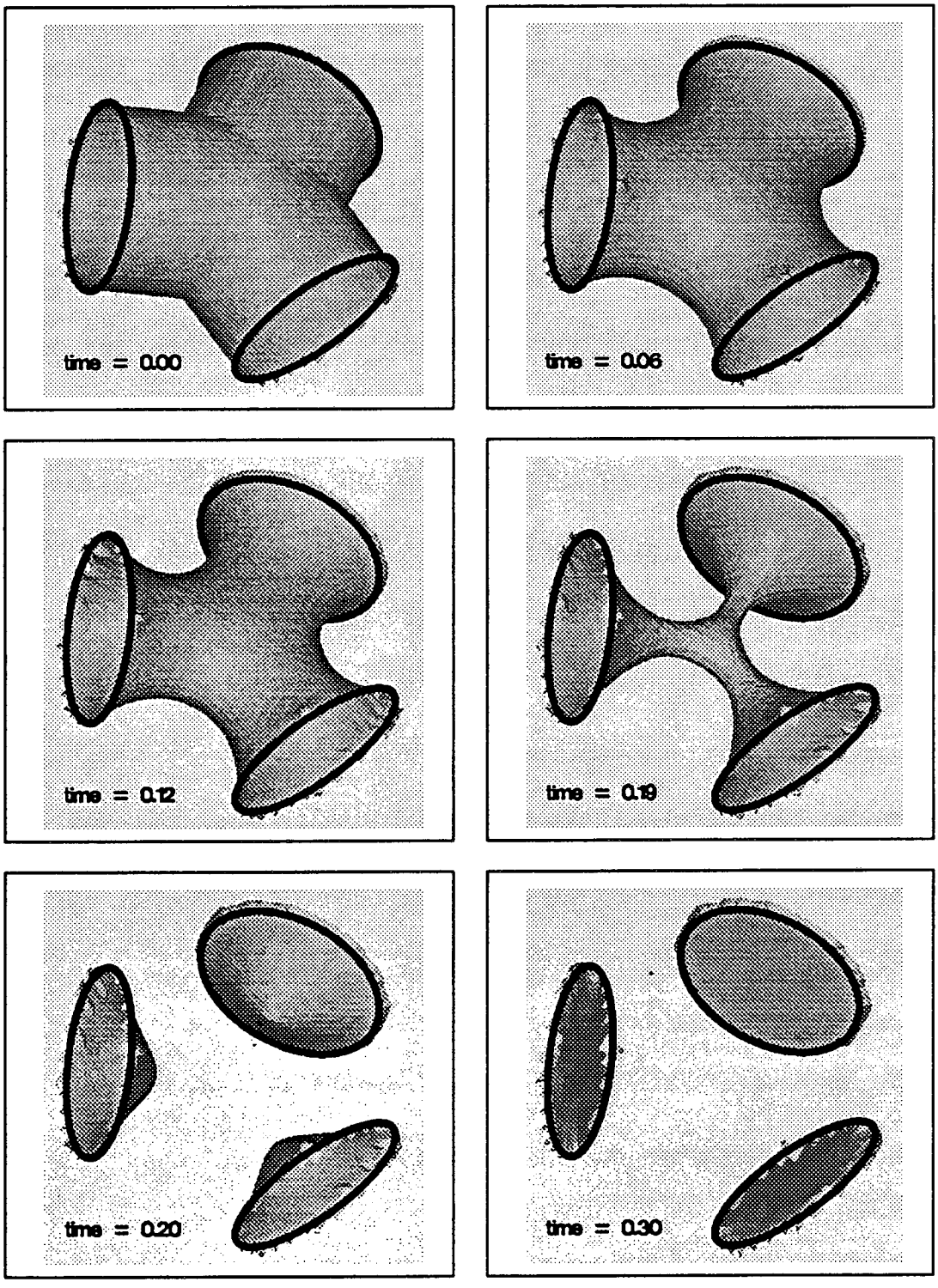


Figure 4.8: Three Rings Splitting Evolution

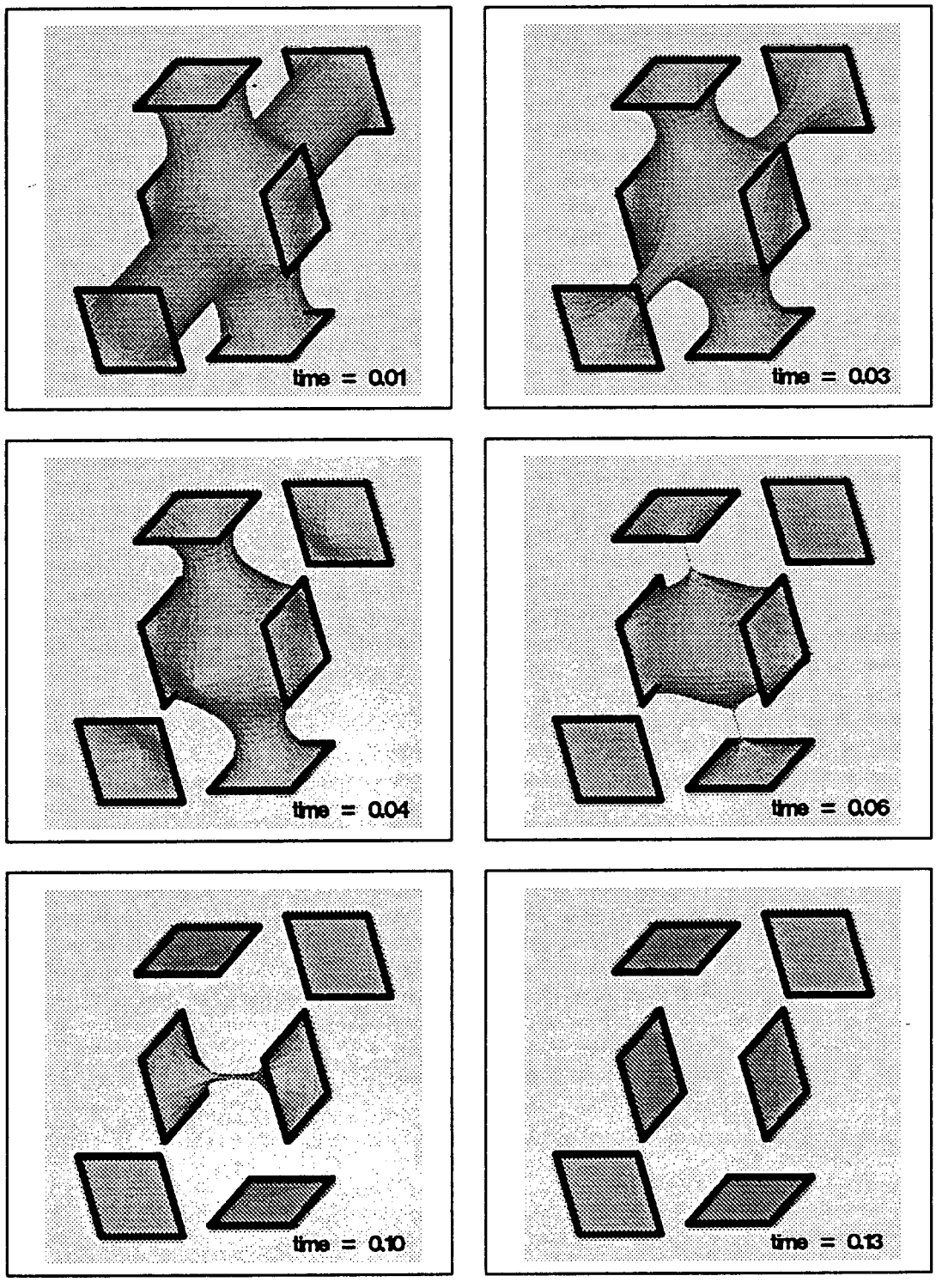


Figure 4.9: Six Squares Splitting Evolution

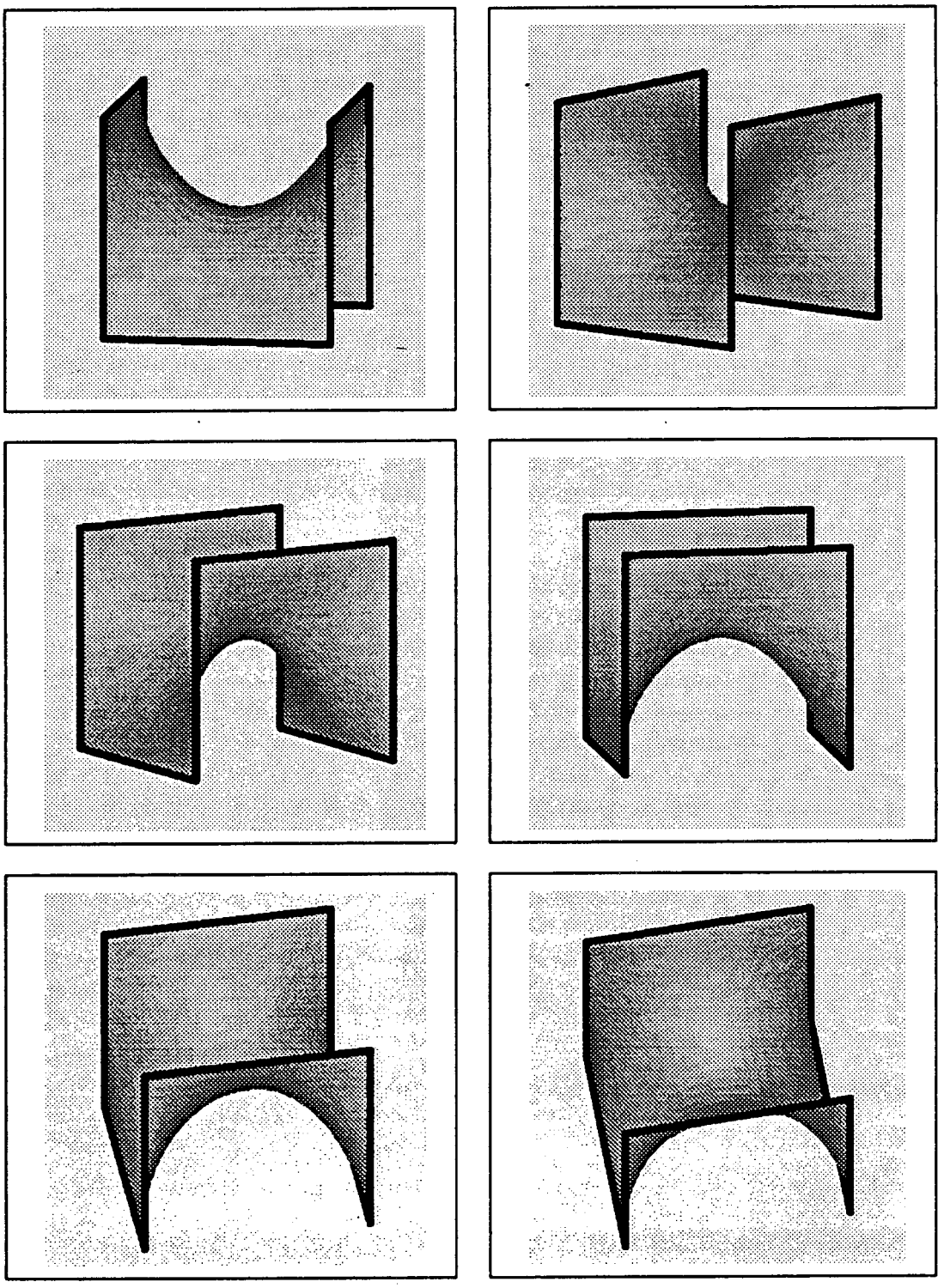


Figure 4.10: Square Sherck Surface

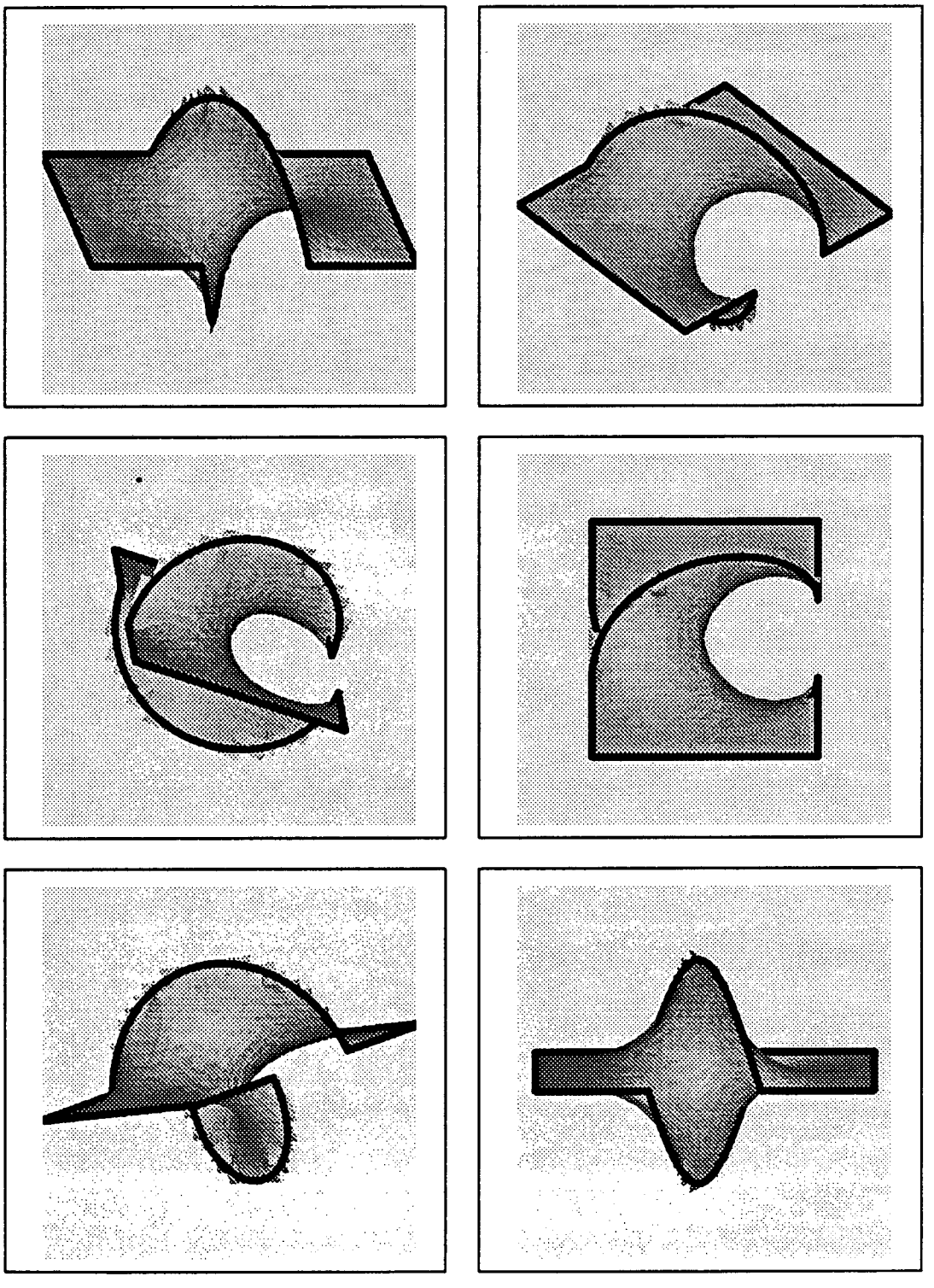


Figure 4.11: Twisted Rectangular Strip Surface

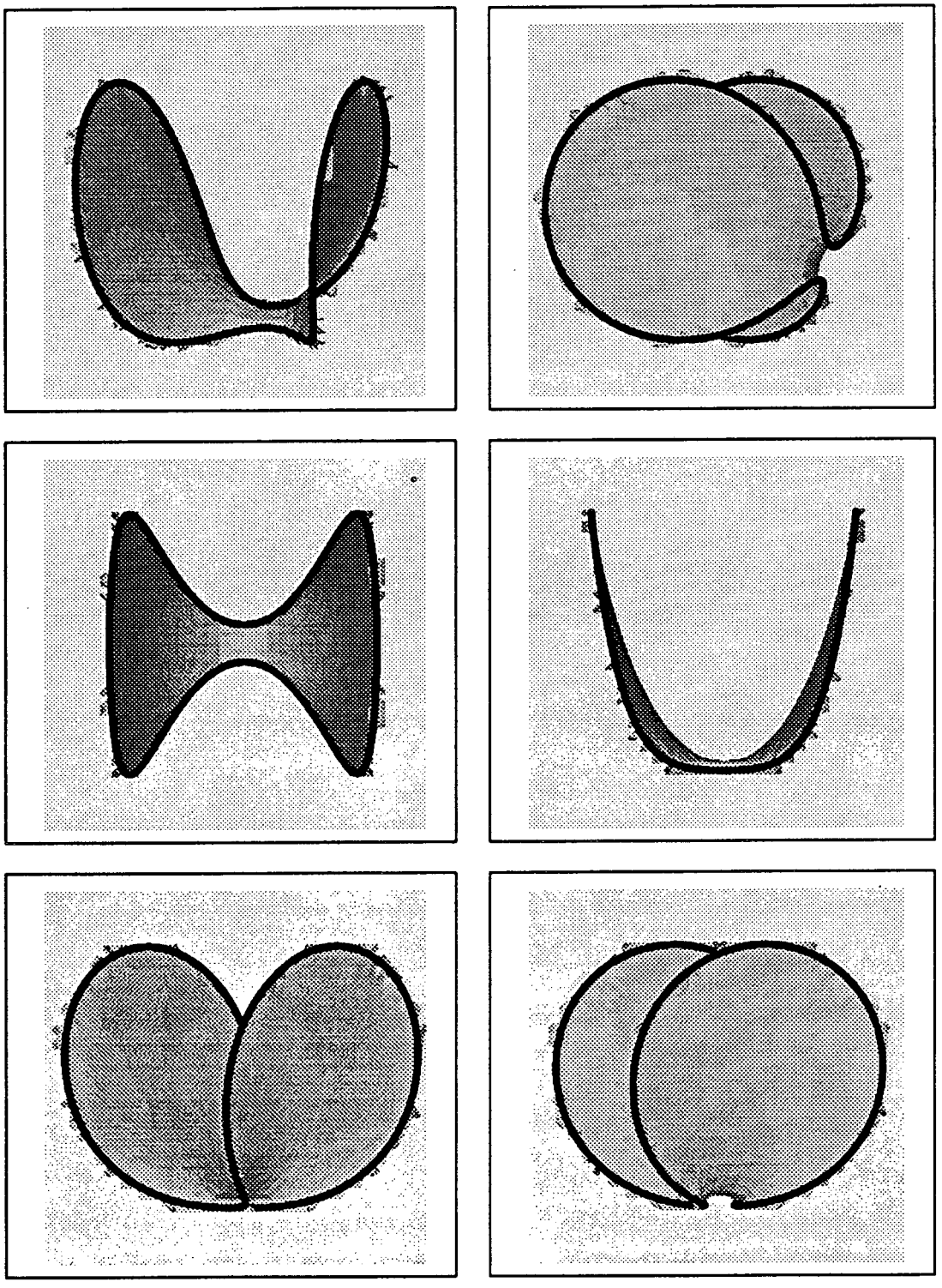


Figure 4.12: Oval on a Cylinder Boundary Surface

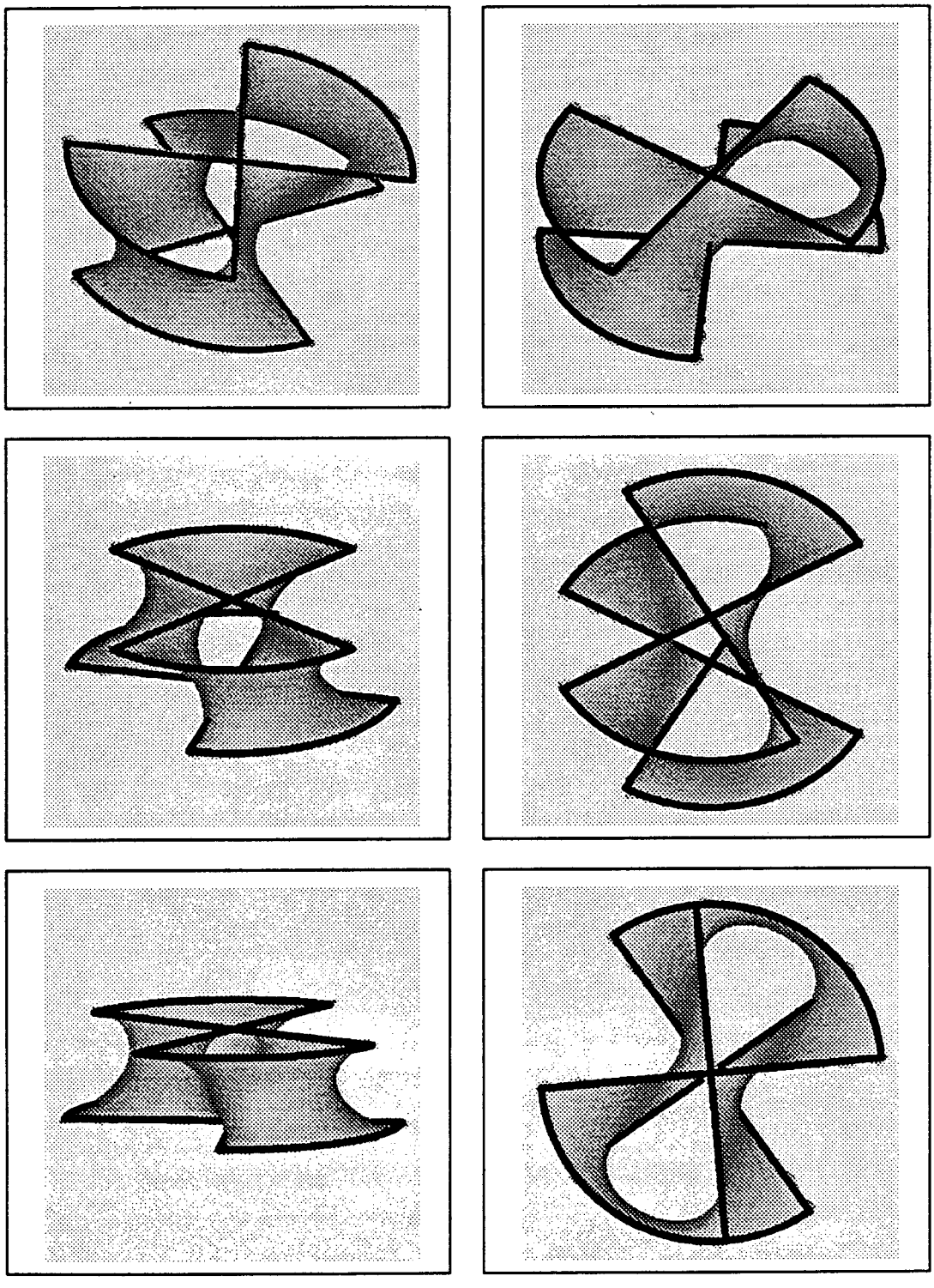


Figure 4.13: Twisted Bowtie Surface

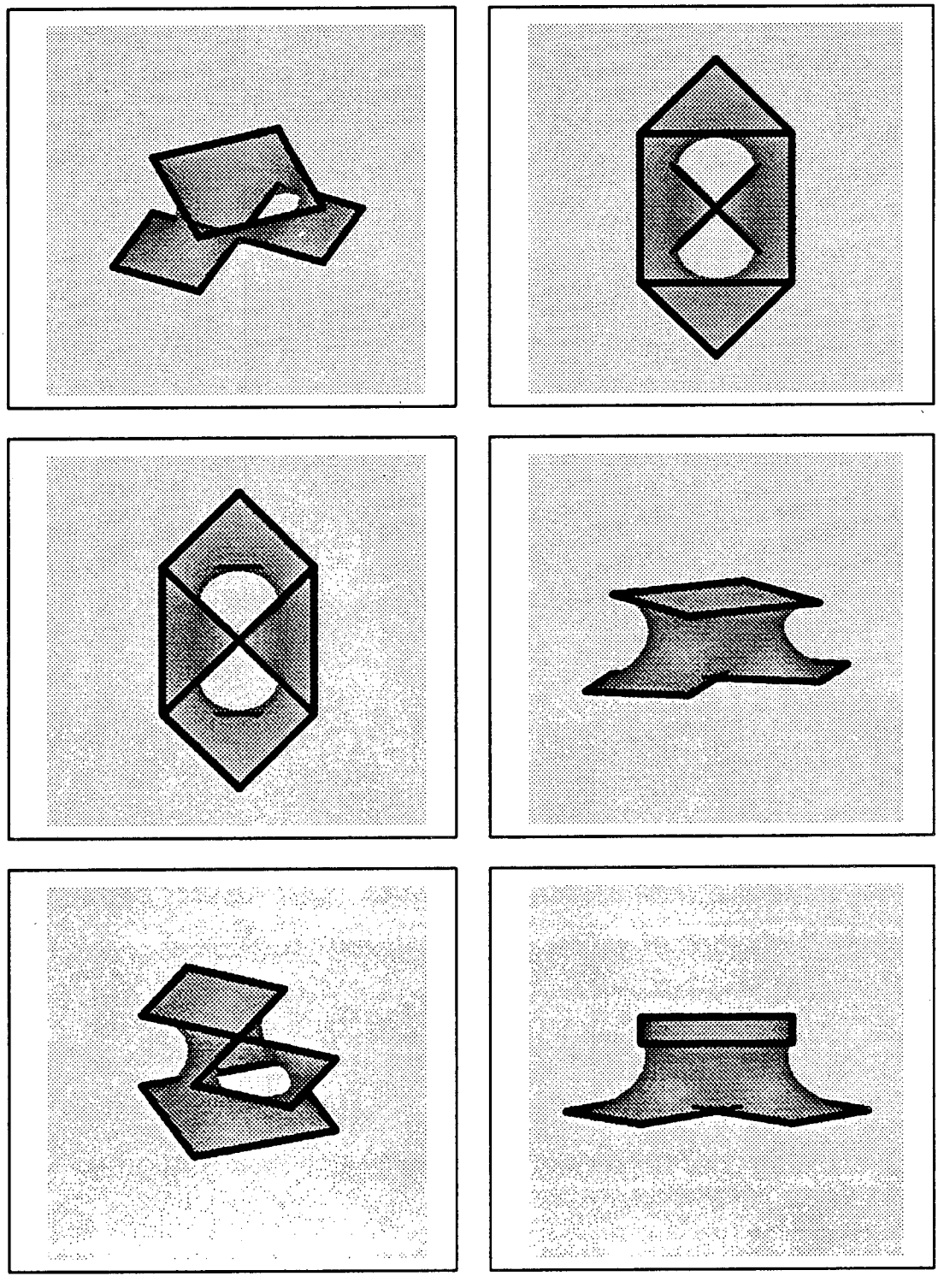


Figure 4.14: Square and Diamonds Surface

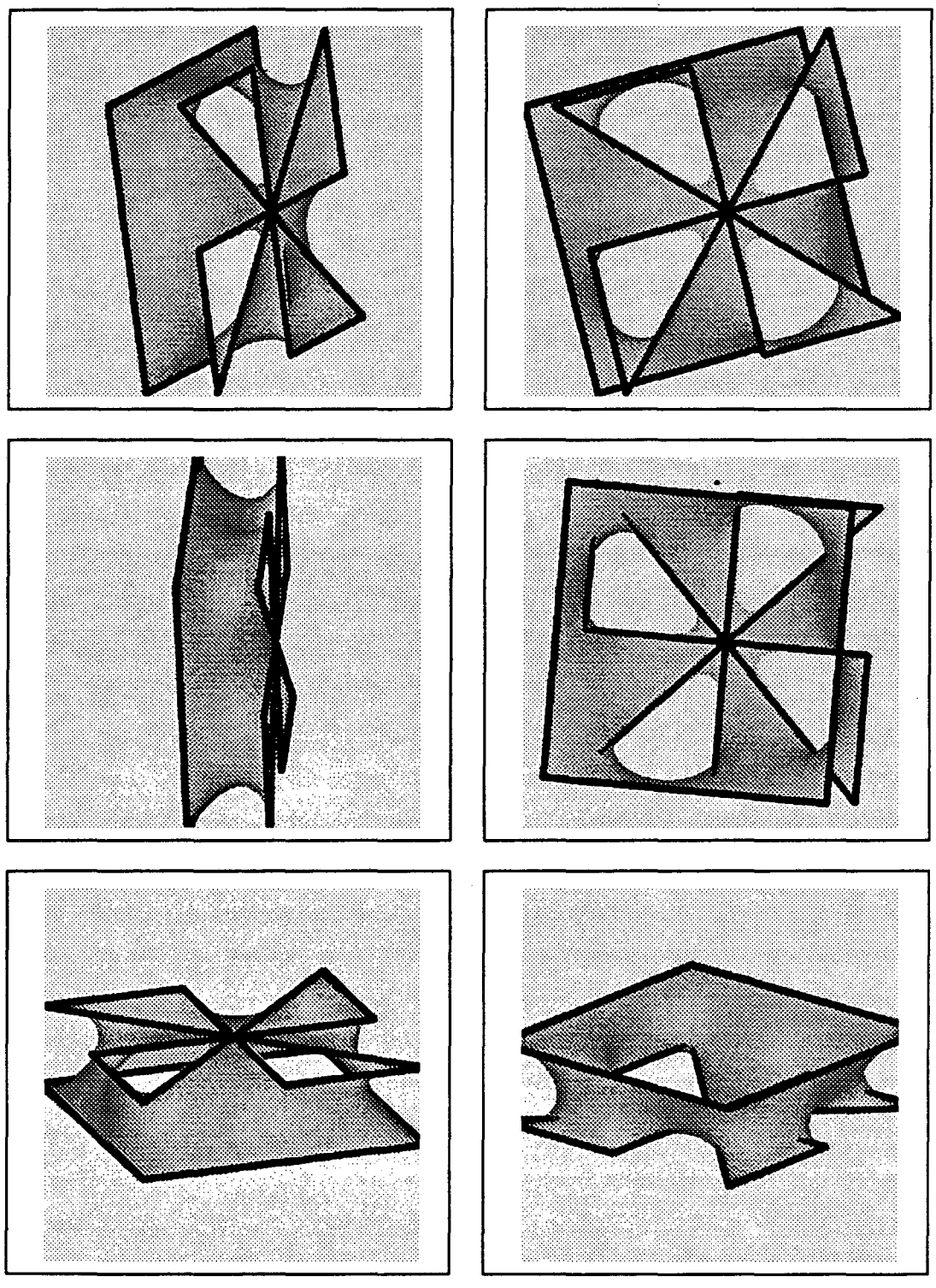


Figure 4.15: Square and Pinwheel Surface

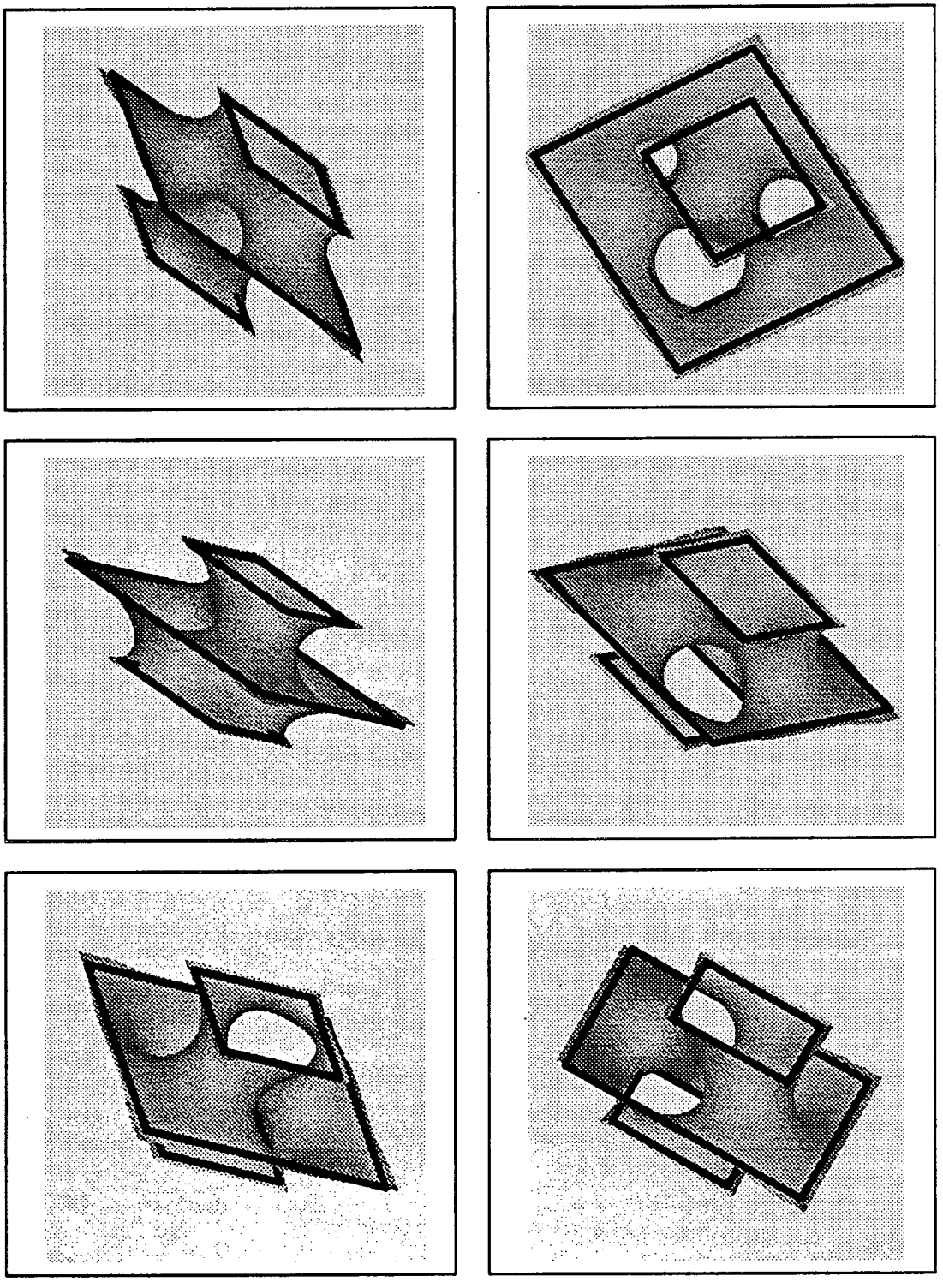


Figure 4.16: Square Meeks-Yao Surface

Chapter 5

Limitations and Future Extensions

In creating the new advantage of topological changes, certain desirable properties were sacrificed. At present, the new algorithm is not capable of studying minimal surfaces which have the boundary passing through an interior portion of the surface as in Figure 5.1. The obvious reason

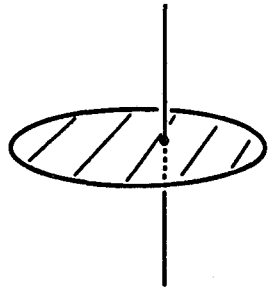


Figure 5.1: Surface with boundary passing through interior

is because the artificial values created by the interpolatory boundary conditions interfere with the natural motion of the surface where the boundary passes through. A more careful look at the boundary conditions under these circumstances is the subject of future work.

Also, the algorithm is not capable of handling triple points, for example, the catenoid with a disk in the center of Figure 1.5. Because of the level set formulation, it is not possible to have the sign of Φ change across each surface (see Figure 5.2). The possibility of extending the algorithm in this case is under investigation.

Several other extensions to the algorithm are also under investigation. One extension is in the direction of solving problems where a portion of the boundary is allowed to float freely within certain constraints, for example fixed arc length or derivative conditions. Another extension is for solving constant non-zero mean curvature surfaces.

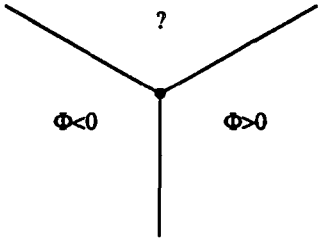


Figure 5.2: Example of a triple point

Bibliography

[1] K. A. Brakke. Surface evolver program. Research Report GCG 17, the Geometry Supercomputer Project, 1200 Washington Ave. South, Minneapolis, MN 55455, 1990.

[2] Paul Concus. Numerical solution of the minimal surface equation. *Mathematics of Computation*, 21:340–350, 1967.

[3] C. Coppin and D. Greenspan. A contribution to the particle modeling of soap films. *Applied Mathematics and Computation*, 26:315–331, 1988.

[4] J. Douglas. Solution of the problem of Plateau. *Transactions of the American Mathematical Society*, 33:263–321, 1931.

[5] A. Elcrat and K. Lancaster. On the behavior of a non-parametric minimal surface in a non-convex quadrilateral. *Archive for Rational Mechanics and Analysis*, 94(3):209–226, 1986.

[6] L. Euler. *Methodus inveniendi lineas curvas maximi minimive propreitate gaudeates sive solutio problematis isoperimetrici latissimo sensu accepti*, volume 24. Opera omnia (1), Füssli, Turici, 1952.

[7] L. C. Evans and J. Spruck. Motion of level sets by mean curvature I. *Journal of Differential Geometry*, to appear.

[8] D. Greenspan. On approximating extremals of functionals. *ICC Bulletin*, 4:91–98, 1965.

[9] M. Hinata, M. Shimasaki, and T. Kiyono. Numerical solution of Plateau's problem. *Mathematics of Computation*, 21:340–350, 1967.

[10] Ronald H. W. Hoppe. Multigrid algorithms for variational ineqaulities. *Siam Journal of Numerical Analysis*, 24(5), October 1987.

[11] J. L. Lagrange. *Essai d'une nouvelle méthode pour déterminer les maxima et les minima des formules intégrales indéfinies*, volume I. Gauthier-Villars, Paris, 1867. translated by D. J. Struick.

[12] William E. Lorenson and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4), July 1987.

[13] Johannes C. C. Nitsche. *Lectures on Minimal Surfaces*, volume 1. Cambridge University Press, 1989.

[14] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1), November 1988.

[15] J. Plateau. *Statique expérimentale et théorique des liquides soumis aux seules forces moléculaires.* Gauthier-Villars, Paris, 1873.

[16] T. Radó. On Plateau's problem. *Annals of Mathematics*, 31(2):457–69, 1930.

[17] J. Sethian. Curvature and the evolution of fronts. *Communications in Mathematical Physics*, 101:487–499, 1985.

[18] J. Sethian. A review of recent numerical algorithms for hypersurfaces moving with curvature-dependent speed. *Journal of Differential Geometry*, 31:131–161, 1989.

[19] H. J. Wagner. A contribution to the numerical approximation of minimal surfaces. *Computing*, 19:35–58, 1977.

LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
INFORMATION RESOURCES DEPARTMENT
BERKELEY, CALIFORNIA 94720