Chapter 7

Finding Minimal Geodesics: A Coarse to Fine Approach

Finding the shortest path between points on a surface is a challenging global optimization problem. It is difficult to devise an algorithm that is computationally efficient, locally accurate and guarantees to converge to the globally shortest path. In this chapter a two stage coarse to fine approach of finding shortest paths is suggested [73]. In the first stage a fast algorithm is used to obtain an approximation to the globally shortest path. In the second stage the approximation is refined into a locally optimal path. In the first stage we use the fast algorithm introduced in [83] that combines a 3-D length estimator with graph search. This path is then refined to a shorter geodesic curve by an algorithm that deforms an arbitrary initial curve ending at two given surface points via geodesic curvature shortening flow. The $3D$ curve shortening flow is transformed into an equivalent $2D$ one that is implemented using an efficient numerical algorithm for curve evolution with fixed end points, introduced in [77, 78].

7.1 Chapter Introduction

Searching for the shortest path, also known as the *minimal geodesic*, between two points on a three dimensional surface is very important in robotic navigation and in three dimensional shape analysis. For example, it is the key to computational surface flattening [124], a potentially useful technique for brain research.

Standard computational procedures to obtain shortest paths on continuous surfaces involve numerical solutions of differential equations by numerical integration [5], and are computationally intensive. These methods are accurate but yield paths that are only locally optimal. When handling complex surfaces a good initial guess is needed to increase the likelihood of convergence to the globally optimal path.

In the *discrete geodesic problem*, the shortest paths between points on a *polyhedral* surface are to be determined. This is of importance in certain CAD-oriented and land-surveying applications where a polyhedral representation of solids and surfaces is natural. An algorithm to solve this problem was described in [98], in which the shortest path from a source point to a destination point is found in $O(n^2 \log n)$ time, where n denotes the number of edges in the polyhedral surface. The length of the shortest path to any other destination can then be determined in $O(\log n)$ time, and the shortest path can be listed in $O(k)$ time, where k is the number of edges crossed by the path. Wolfson and Schwartz [144] argued that the algorithm of [98] is difficult to implement, and suggested an algorithm that is simpler to implement, but is of exponential computational complexity.

Kiryati and Székely [83] considered the problem of finding the shortest path between points on a continuous three dimensional surface that is given in a digitized form. Its basis is the observation that actual interpolation of the digitized surface may not be needed in order to estimate properties of the underlying continuous structure to fairly high accuracy. Their approach combined recently developed length estimators for digitized three dimensional curves with well known algorithms for computing shortest paths in sparse graphs, to estimate the minimal distances and the corresponding shortest paths in a systematic, computationally efficient way. Note that shortest path algorithms that are based on surface discretization and graph search are fast, but are of inherently limited accuracy [99].

A framework that combines the advantages of two different approaches in order to obtain the minimal geodesic on an essentially continuous $3D$ surface quickly and accurately, is introduced in this chapter. A two stage coarse–to–fine approach is taken. The fast graph–search based algorithm [83] that operates on a voxel representation of the surface is used as a first stage to obtain a good initial approximation efficiently, as an input to an accurate but slower shortening flow differential geometry technique [77, 78]. In the second stage, the curve is evolved by a geodesic curvature flow to the geodesic curve closest to the initial approximation. In comparison to pure differential techniques, the speed of convergence is greatly improved and the risk of convergence to an insignificant local minima is alleviated.

Curve shortening flow has attracted great interest in recent years in the field of Differential Geometry. In [52], Grayson extended the theory of planar curve evolution and noted several global properties of curvature evolution of smooth curves immersed in a Riemannian surface. He proved that the curve remains smooth, and it either converges to a point, or becomes a geodesic curve when its geodesic curvature converges to zero in the C^∞ norm.

One of the most important properties of the flow–by–curvature curve evolution process is that it shrinks the curve as fast as possible, in the sense that flow lines in the space of closed curves are tangent to the gradient of the length functional [52]. For this reason, the flow is also called *curve shortening flow*. Therefore, deforming a curve by its geodesic curvature vector is a very efficient way of finding geodesic curves.

The curve shortening flow of the suggested combined approach operates on a $3D$ surface given as an elevation array. Given the surface and two points on it, the curve shortening flow algorithm evolves an arbitrary initial embedded curve between these points. Grayson's results show that the curve remains smooth and embedded, and if the end points are fixed it converges to a geodesic curve on the surface between the given points. See [26, 29, 27] for a similar approach.

For computerized implementation of the second (shortening flow) stage, the three dimensional curve flow is first transformed into an equivalent two dimensional curve flow. It is then implemented based on a numerical algorithm derived from [111], together with a simple algorithm, motivated by [26], for keeping the end points fixed.

In Section 7.2 the first stage in which an initial approximation is rapidly obtained is described. Section 7.3 presents the second stage that is based on a geodesic curvature shortening flow type of algorithm, with the proper numerical implementation. Examples of carrying out the two stages successively are shown in Section 7.4.

7.2 Rapid Approximation of the Shortest Path

The first stage of the suggested approach is based on the algorithm described in [83] and is used here to rapidly obtain an approximation to the shortest path between points on a 3D surface, that is then refined by the second, curve flow based stage. This section briefly summarizes the principles of the algorithm of Kiryati and Székely [83].

Suppose that the surface of an object is given in a digitized form as a set of voxels in a three dimensional array. Assume that a voxel belongs to the digitized surface if it is traversed by the underlying continuous surface and if at least one of its direct (*i.e.*, 6-directional) neighbors is a "background" voxel.

A path on the surface that connects two surface points can be represented in a digital form by the set of surface voxels that the path traverses. This set of voxels, the "digital path," can be represented by a 6-directional chain code. Alternatively, full diagonal connectivity can be admitted, to obtain a 26-directional chain code representation.

The $3D$ length estimation problem is to estimate the length of an underlying continuous $3D$ curve given its chain code. The estimator recently developed in [82] is based on link classification in the 26-directional chain code representation of the curve. It has the general form of:

$$\hat{L} = \Psi_1 N_1 + \Psi_2 N_2 + \Psi_3 N_3, \tag{7.1}$$

where N_1 is the number of direct links in the chain code, *i.e.*, links that are parallel to one of the three main axes, and N_2 and N_3 are respectively the number of minor and major diagonal links in the chain code (see Figure 7.1). Ψ_1, Ψ_2 and Ψ_3 are weights, and $\hat{L}$ is the estimated length.

Assuming that the digitization is sufficiently dense so that the curve is reasonably straight within one or two voxels, the naive selection of weights $\Psi_1 = 1$, $\Psi_2 = \sqrt{2}$ and $\Psi_3 = \sqrt{3}$ would lead to consistent overestimation of the length. Assuming a uniform distribution of orientations, an *unbiased* estimator that achieves the least RMS error possible for unbiased estimators, was found, see [82], to be:

$$\hat{L} = 0.9016 N_1 + 1.289 N_2 + 1.615 N_3. \tag{7.2}$$

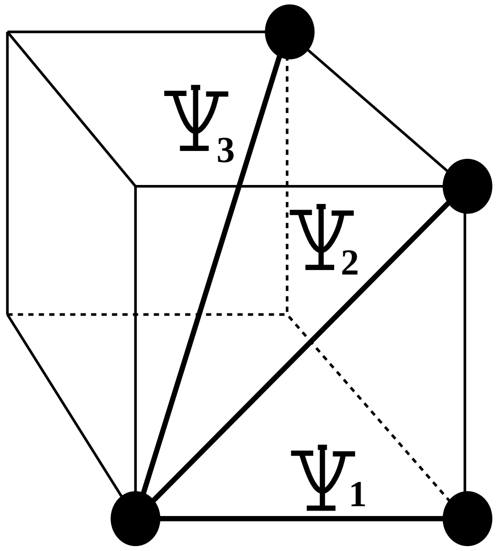


Figure 7.1: The link types and corresponding weights in a 26-directional $3D$ chain code.

The fact that the estimator is unbiased implies that if the direction of the tangent varies along the curve, local estimation errors cancel out and lower total estimation errors are obtained.

Thus, assuming that the digitization grid is fine with respect to the curvature of a path, such that the path is almost linear within most of the voxels, its length can be estimated from the digital representation using a three dimensional length estimator. This is the basis for the the algorithm of [83] that constitutes the first step in our approach. The shortest path between two points is assumed to be associated with the digital path whose length estimate is the shortest, and that estimate is suggested as an approximation to the minimal distance.

Let us view the digital surface as a three dimensional graph, in which each vertex corresponds to a surface voxel. Given any specific definition of surface connectivity, pairs of vertices that correspond to pairs of neighboring surface voxels are connected by arcs in the graph. In [83] the 26-directional connectivity is assumed, hence every surface voxel is connected by an arc to every surface voxel in its (26) neighborhood. If each arc is assigned a cost according to the weight of its link type (direct, minor diagonal or major diagonal) in the $3D$ length estimator, then estimating minimal distances and shortest paths on a continuous surface given in digitized form, reduces to finding the shortest path in a graph.

Algorithms for finding minimal distances and shortest paths in graphs are well known; see reference [50] for an overview. Here all arcs have positive weights, hence, Moore and Dijkstras' algorithm can be applied. Let N denote the number of vertices in the graph, *i.e.*, the number of surface voxels; then the shortest paths between a source vertex to all other vertices can be found in $O(N^2)$ time. It is however clear that surface graphs are sparse. The number of arcs emanating from any vertex is upper bounded by 26. (In practice, since it is assumed that the digitization is fine, a surface would be roughly planar within most small neighborhoods, so the number of arcs emanating from a vertex is about 8). The computational complexity can thus be reduced, and the minimal distance from a vertex to all others can be estimated in $O(N \log M)$ time, where M is the total number of arcs in the graph, and is proportional to N, [50]. This means that the minimal distances can be found in $O(N \log N)$ time. The minimal distances between *every* pair of vertices can obviously be determined in $O(N^2 \log N)$ time by N applications of the algorithm. Observe however, that the shortest paths and minimal distances in the graph are just an *approximation* to the shortest paths and minimal distances on the continuous surface. The quality of the approximation reflects the quality of the $3D$ length estimator [82].

Given the source voxel, the minimal distances to all other surface voxels are thus estimated in $O(N \log N)$ time. Note that if a single destination is specified in advance, actual computing time can be significantly reduced by simultaneous propagation from the source and destination voxels until the first meeting of the distance wavefronts. The time required to actually report the path to any specific destination is then just $O(K)$, where K is the number of voxels in the digital path.

7.3 Curve Shortening Path Refinement

The first stage of the suggested approach, described in the previous section, produces an approximation of the shortest path to be refined by the second curvature flow based stage described in this section. The curvature flow algorithm requires an elevation array representation of the surface to be available.

It is assumed that the reader is familiar with basic concepts of surface differential geometry, see *e.g.* [38]. However, in order to be self contained, a short description of each property is given before it is used in constructing the flow.

The path of minimal length connecting two points on a surface is always a geodesic. On the other hand, not every geodesic is a path of minimal length, it is only locally minimal. Using the results of the first stage as an initial approximation, the shortening flow algorithm will shorten the curve to a geodesic. In most practical situations, this geodesic will be a global minimal path, i.e. the *minimal geodesic*. If there are several significantly different possible paths of nearly the same length as the shortest one, the approximation obtained in the first stage may not correspond to the absolutely shortest path. In that case, the geodesic obtained in the second stage might be just slightly longer than the minimal geodesic.

Given a regular surface $\mathcal{S}$ and two points $\mathbf{a}$ and $\mathbf{b}$ on it, we want to compute a geodesic curve ending at these points. Let $\alpha_0 : [p_1, p_2] \to \mathcal{S}$, be a given initial embedded smooth curve such that $\alpha_0(p_1) = \mathbf{a}$ and $\alpha_0(p_2) = \mathbf{b}$. Based on Grayson's results [52], if α_0 is deformed via the curve shortening flow on $\mathcal{S}$, and if the two end points are kept fixed, the curve converges to a geodesic curve as fast as possible. The geodesic curvature flow is given by

$$\frac{\partial \alpha(p,t)}{\partial t} = \kappa_g \mathcal{N}^\alpha, \tag{7.3}$$

where p parameterizes the curve, t stands for time, and $\kappa_g \mathcal{N}^\alpha$ is the geodesic curvature vector of the curve α. The geodesic curvature vector may also be written as:

$$\kappa_g \mathcal{N}^\alpha = \alpha_{ss} - \langle \alpha_{ss}, N \rangle N,$$

where α_{ss} (the curvature vector) is the second derivative of the curve according to s, its arc-length parameterization, and N is the normal to the surface. From (7.3) it follows immediately that the only possible stable curves are geodesic curves[1]. Hence, applying (7.3) to the initial curve α_0 will give us the required geodesic curve.

As we have seen in Chapter 2, there is an efficient numerical algorithm for the computer implementation of planar curve flows. In order to use this numerical recipe the $3D$ flow (7.3) is transformed into a $2D$ one. Let the surface $\mathcal{S}$ be a graph surface, *i.e.*, a function $z(x, y)$, where x and y stand for the Cartesian coordinates on the (x, y)–plane. Then, the curve $\alpha(p, t)$ is given by the triplet $\{x(p), y(p), z(p)\}$, where $z(p) = z(x(p), y(p))$. Define the curve $\mathcal{C} := \{x(p), y(p)\}$ as the projection of the $3D$ curve $\alpha(p, t)$ on the (x, y)–plane. It was shown in [77, 78] that while α deforms according to (7.3) on the surface, its corresponding projection $\mathcal{C}$ evolves according to the following $2D$ curve flow

$$\frac{\partial \mathcal{C}}{\partial t} = \kappa_g \langle \pi \circ \mathcal{N}^\alpha, \mathcal{N} \rangle \mathcal{N},$$

where $\mathcal{N}$ is the unit normal vector of the planar curve, and π is the projection operation $\pi \circ (x, y, z) \equiv (x, y)$. Rewriting this evolution as a function of $\mathcal{C}$, the planar curve, and the surface's first derivatives, we obtain

$$\frac{\partial \mathcal{C}}{\partial t} = \left[\frac{1}{\eta} \left(\kappa + \psi \frac{\langle \nabla z, \mathcal{N} \rangle}{1 + z_x^2 + z_y^2} \right) \right] \mathcal{N} \ , \tag{7.4}$$

where $\mathcal{N} = (n_1, n_2)$ and κ are the unit normal and the curvature of the planar curve, respectively, and

$$\eta(\mathcal{N}, \nabla z) = n_1^2(1 + z_y^2) + n_2^2(1 + z_x^2) - 2 z_x z_y n_1 n_2$$
$$\psi(\mathcal{N}, z_{xx}, z_{xy}, z_{yy}) = z_{xx} n_2^2 - 2 z_{xy} n_1 n_2 + z_{yy} n_1^2.$$

Solving (7.4) is equivalent to solving (7.3).

A numerical algorithm for the computer implementation of (7.4) while keeping the end points fixed is presented in the following section.

7.3.1 Numerical implementation

Following the Eulerian formulation, if the curve $\mathcal{C}$ is a level set of a function ϕ, then as shown in Chapter 2, its curvature κ can be computed as [127]:

$$\kappa = \frac{\phi_{yy}\phi_x^2 - 2\phi_x\phi_y\phi_{xy} + \phi_{xx}\phi_y^2}{(\phi_x^2 + \phi_y^2)^{3/2}}.$$

[1]Along a geodesic $\alpha_{ss} \| N$ so that $\kappa_g = 0$.

The surface evolution equation corresponding to (7.4) is given by

$$\phi_t = \tag{7.5}$$

$$\frac{\phi_x^2\phi_{yy} - 2\phi_x\phi_y\phi_{xy} + \phi_y^2\phi_{xx} + (z_x\phi_x + z_y\phi_y)(z_{xx}\phi_y^2 - 2\phi_x\phi_y z_{xy} + z_{yy}\phi_x^2)\frac{1}{1+z_x^2+z_y^2}}{\phi_x^2(1+z_y^2) + \phi_y^2(1+z_x^2) - 2z_x z_y\phi_x\phi_y}.$$

The next step of the algorithm construction consists of the discretization of Equation (7.5). If singularities cannot develop during the evolution, as in the Euclidean shortening flows (diffusion equation), a straightforward discretization can be performed [111]. The discretization of the evolution equations is performed on a fixed *rectangular grid.*

The space derivatives are implemented using central approximations. Then, the $2D$ curve flow (7.4) reformulated to (7.5) is implemented by taking a forward derivative approximation in time, which yields the desired numerical scheme. Since the evolving curve is given by the level set of the function ϕ, we have to find this level set (ϕ is discrete now). This is done by using a very simple linear interpolation type of contour finding procedure as the one described in [128].

The algorithm as described by Osher and Sethian works on closed curves, or curve flows with boundary conditions. In our case, we have to keep the two end points fixed. This is done by adding a step to the algorithm, which alters the ϕ function after each iteration (this part of the algorithm was motivated by Chopp's work on minimal surface computation [26]). This change is done in order to ensure that the two end points will remain in their position, as two fixed points in the zero level set of the evolving function. The function ϕ is changed in such a way that the weighted average in the neighborhood of the end points is equal to zero. In our examples we have actually added two lines connecting the curve's end points to the boundary of the domain. This way, the curve divides the domain to two parts, and the ϕ function is defined to be positive on one part and negative on the other. Note that a stopping condition must be defined. It can be related either to a threshold value on the geodesic curvature or to small changes in the evolving curve.

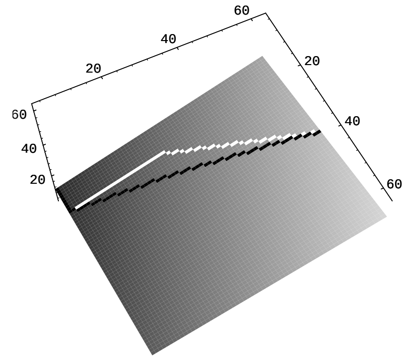
The proposed shortening flow stage can be summarized as follows:

1. Initialize ϕ_0, using $\pi \circ \alpha_0$.
2. Evolve ϕ according to (7.5), by using the corresponding numerical approximation.
3. Adapt ϕ, to keep the end points fixed.
4. If the stopping condition is not satisfied go to 2.
5. Find the zero level set, α is obtained as $z(\mathcal{C})$.
6. Stop.

7.4 Examples

The suggested two stage algorithm was applied to several test surfaces with two points. The test surfaces were contained in a $64 \times 64 \times 64$ voxel array for the first stage. A $64 \times 64 \times \mathbb{R}$ elevation array representation was used in the second stage.

In the first example the shortest path between two points on a tilted plane is computed. This simple example is very interesting since it is known[83] to demonstrate worst-case performance of the first stage of the algorithm. Indeed, the first graph search stage produced the broken white line in Figure 7.2. The second shortening flow algorithm corrected the path to the black straight line. In all the examples, the gray level of the surface corresponds to elevation, the white path is the outcome of the first stage, the black path is the result of the second refining stage.



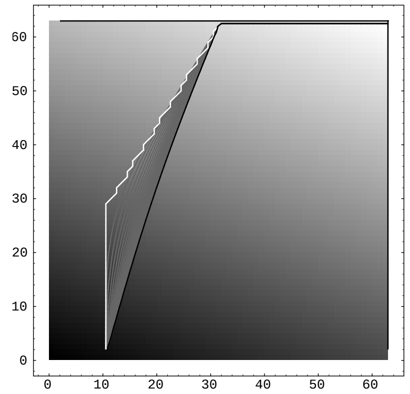


Figure 7.2: In the upper frame, the path obtained by the first stage is painted in white on the tilted plane. The white path is refined by the second stage of the algorithm into the black path. The lower frame describes an orthographic projection of the surface from above. The white path is again the result of the first stage, the gray intermediate paths converge to the black final refined path.

In the sequence of the next three examples, Figure 7.3, Figure 7.4 and Figure 7.5, the amplitude of an "egg–box" type of surface was increased, causing the estimated path as well as the refined shortened geodesic to slide away from the top of the sinusoidal hills. The length of the shortening paths along the iterations of the second stage is shown in Figure 7.6.

In the last example, the shortest path between two points on a complicated surface is located, see Figure 7.7. Observe the way the second algorithm refines the path produced by the first one, and thereby locates the accurate position of the shortest path.

7.5 Chapter Summary

This chapter combines two approaches to locate the minimal geodesic between two points on surfaces in an efficient way. The first graph–search based stage produces an initial approximation of the location of the minimal geodesic. This approximation is the initial condition for the second shortening flow based stage that refines the curve by shortening it to a geodesic.

A clear advantage of the first stage is its theoretical as well as practical computational efficiency. Another important feature is the direct operation on voxel data, without need for prior interpolation or polygonal approximation of the surfaces. Its main drawback is that due to the limited accuracy of the underlying $3D$ length estimator, it only produces approximate estimates of the shortest paths.

When different paths of lengths nearly equal to the minimal distance between the source and destination points exist, slight errors in minimum distance estimation could correspond to a large difference between the true shortest path and the estimated one. This means that the shortening flow algorithm might in the worst case converge to a local minimum that is distant from the global minimum but still almost as deep, and would not be trapped into insignificant local minima.

A curve evolution approach for the computation of geodesic curves on surfaces was used in the second stage. It is initialized by an approximation generated by the first stage, and a shorter geodesic curve is obtained by a curve shortening flow. The $3D$ curve flow is represented by an equivalent $2D$ one, which was implemented by an efficient numerical algorithm for curve evolution, together with a procedure for keeping the end points fixed. This projection of the $3D$ flow into a $2D$ equivalent one is an important characteristic of this algorithm since it permits an efficient computer implementation while keeping the curve on the surface.

The suggested combination of the two approaches exploits the computational efficiency of the graph search algorithm while compensating for estimation errors. Sub–voxel accuracy is achieved by using simple interpolation to find the location of the zero level set, at the last step of the shortening flow algorithm. Note that the two stage approach is rather general and allows various algorithms to be used as the first and second stages in the suggested framework.

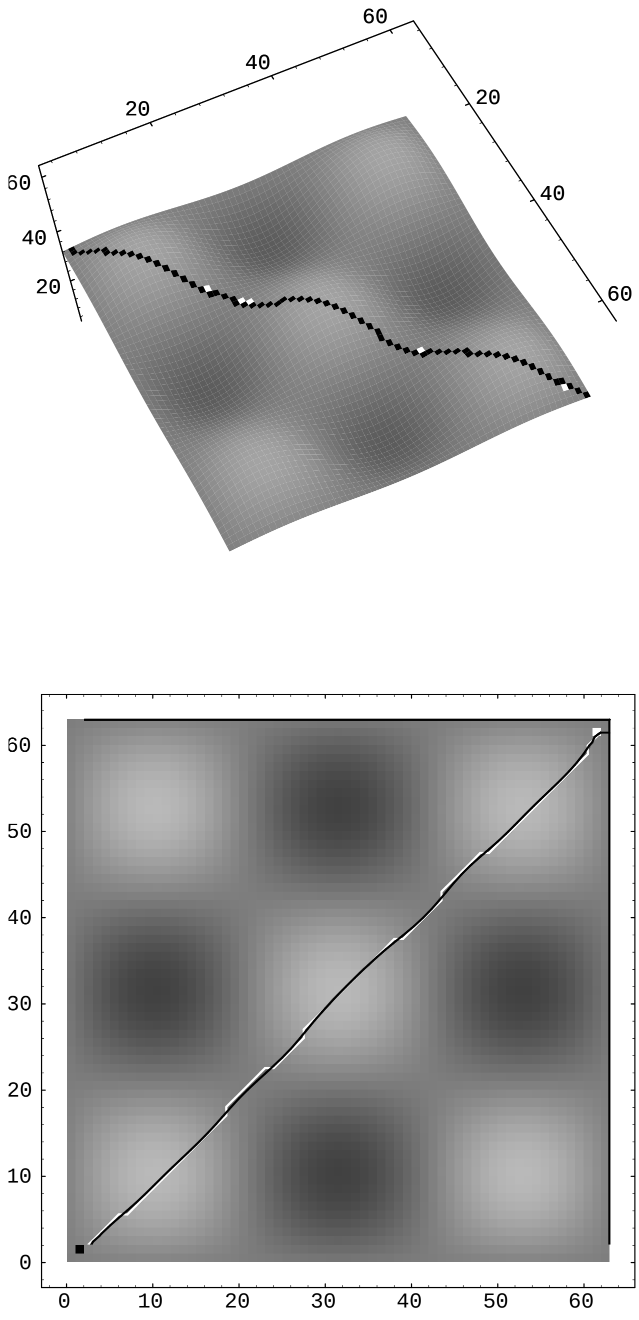
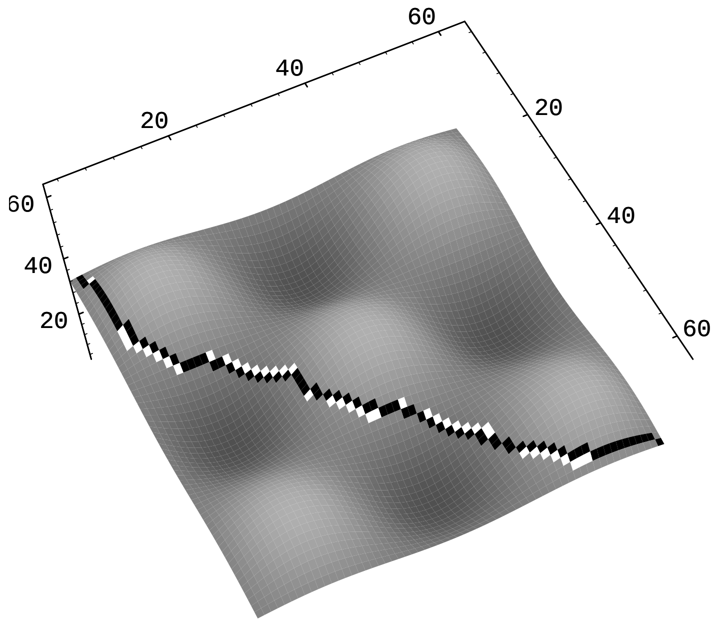


Figure 7.3: For the small amplitude egg–box, the estimated path is close to the refined one. The white pixels painted on the surface belong to the estimated path and indicate the places where the paths are different.



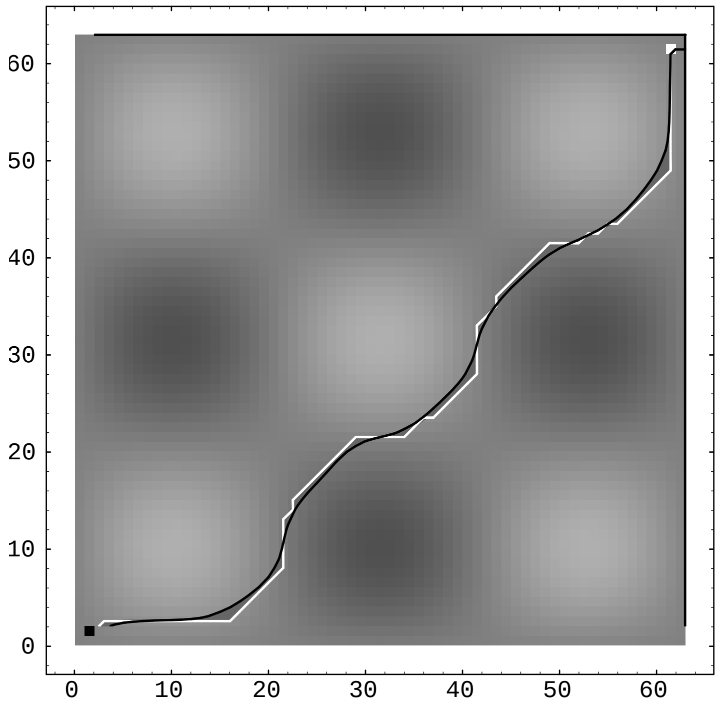


Figure 7.4: Increasing the amplitude of the egg–box, the (white) estimated path as well as the (black) refined one, slide away from the tops of the hills.

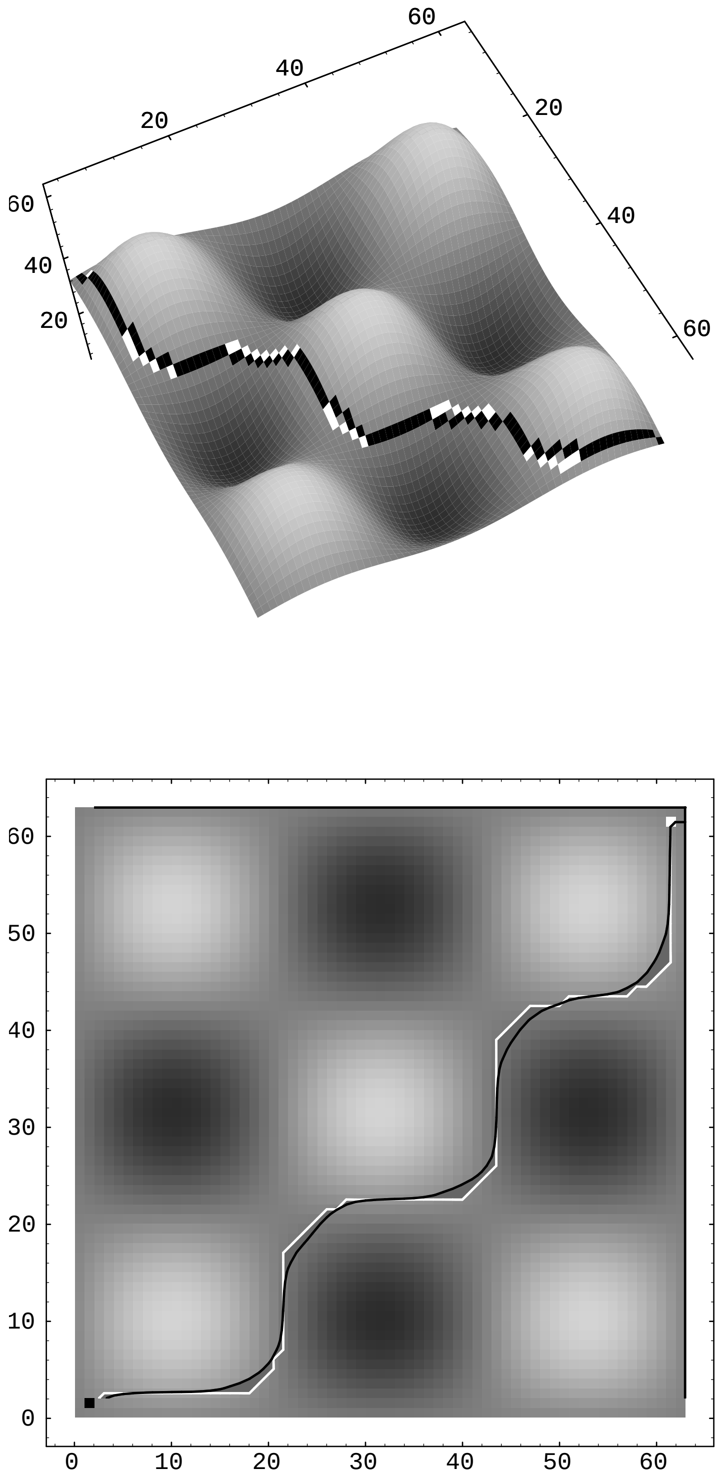


Figure 7.5: Increasing the amplitude, the estimated (white) path and the refined (black) one, nearly track an equal height contour, and avoid climbing on the hills.