Geodesics on Triangle Meshes

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/

Created: September 16, 2008

Contents

1 Introduction

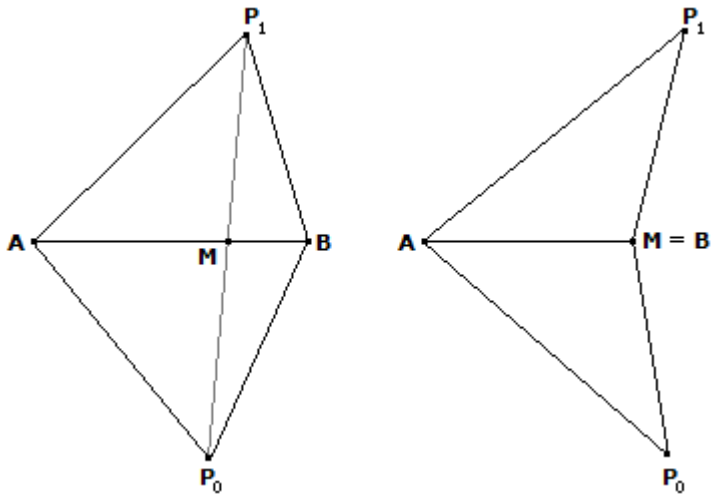
A geodesic on a surface is a curve connecting two points that is (locally) shorter than any other curve (nearby the geodesic) that connects the two points. There may be multiple geodesics connecting two points. For example, an ellipsoid for which there are two paths between antipodal points and a sphere for which there are infinitely many paths between two antipodal points. In the plane, there is a unique path. The geodesic of shortest length gives you the shortest path between two points.

Generally, this is a difficult problem for smooth surfaces (see my Riemannian PDF). The problem is somewhat simpler when restricted to manifold triangle meshes, but it is still difficult to implement.

2 Two Triangles

The simplest example consists of two triangles sharing an edge. Figure 2.1 illustrates [left has **M** interior to edge $\langle \mathbf{A}, \mathbf{B} \rangle$, right has **M** at endpoint].

Figure 2.1 Shortest paths between two vertices, $\mathbf{P}_0$ and $\mathbf{P}_1$, of a pair of triangles. The left image shows the shortest path, $\langle \mathbf{P}_0, \mathbf{M}, \mathbf{P}_1 \rangle$, that passes through an interior point of the shared edge of the triangles. The right image shows the shortest path, $\langle \mathbf{P}_0, \mathbf{B}, \mathbf{P}_1 \rangle$, that consists solely of edges of the triangles.



We want the shortest path from $\mathbf{P}_0$ to $\mathbf{P}_1$, which will consist of a polyline with two segments. Suppose that the shortest path *along only the edges of the triangles* is $\langle \mathbf{P}_0, \mathbf{A}, \mathbf{P}_1 \rangle$. We need to determine if there is a point **M** along the edge $\langle \mathbf{A}, \mathbf{B} \rangle$ for which the path $\langle \mathbf{P}_0, \mathbf{M}, \mathbf{P}_1 \rangle$ is shorter than the path $\langle \mathbf{P}_0, \mathbf{A}, \mathbf{P}_1 \rangle$.

2.1 Setting Up the Minimization Problem

We may set this up as a minimization problem that is solvable using the methods of calculus. Parameterize the shared edge points as

$$\mathbf{M}(t) = \mathbf{A} + t(\mathbf{B} - \mathbf{A}), \quad t \in [0, 1] \tag{1}$$

The length of the path $\langle \mathbf{P}_0, \mathbf{M}(t), \mathbf{P}_1 \rangle$ is

$$L(t) = |\mathbf{M}(t) - \mathbf{P}_0| + |\mathbf{P}_1 - \mathbf{M}(t)| = |t\mathbf{D} - \mathbf{\Delta}_0| + |t\mathbf{D} - \mathbf{\Delta}_1| \tag{2}$$

where $\mathbf{D} = \mathbf{B} - \mathbf{A}$ and $\mathbf{\Delta}_i = \mathbf{P}_i - \mathbf{A}$ for $i = 0, 1$. The length of the path is rewritten as

$$L(t) = \left(at^2 - 2b_0 t + c_0\right)^{1/2} + \left(at^2 - 2b_1 t + c_1\right)^{1/2} \tag{3}$$

where $a = |\mathbf{D}|^2$, $b_i = \mathbf{D} \cdot \mathbf{\Delta}_i$, and $c_i = |\mathbf{\Delta}_i|^2$ for $i = 0, 1$.

The value of t that minimizes $L(t)$ is a root of the derivative $L'(t)$ or is an interval endpoint 0 or 1. The derivative is

$$L'(t) = \frac{at - b_0}{\left(at^2 - 2b_0 t + c_0\right)^{1/2}} + \frac{at - b_1}{\left(at^2 - 2b_1 t + c_1\right)^{1/2}} \tag{4}$$

Setting $L'(t) = 0$ and subtracting one term to the right-hand side of the equation,

$$\frac{at - b_0}{\left(at^2 - 2b_0 t + c_0\right)^{1/2}} = -\frac{at - b_1}{\left(at^2 - 2b_1 t + c_1\right)^{1/2}} \tag{5}$$

Squaring both sides, multiplying by the denominators, cancelling common terms and grouping the remaining terms leads to the quadratic equation

$$a\left[a\left(c_0 - c_1\right) + \left(b_1^2 - b_0^2\right)\right] t^2 + 2\left[a\left(b_0 c_1 - b_1 c_0\right) + b_0 b_1 \left(b_0 - b_1\right)\right] t + \left(b_1^2 c_0 - b_0^2 c_1\right) = 0 \tag{6}$$

If $\bar{t}_j$ are the real-valued roots in $[0, 1]$ ($j = 0$ for a single root, $j = 0$ and $j = 1$ for two distinct roots), then the minimum length is the smallest of $L(0)$, $L(1)$, and $L(\bar{t}_j)$. The corresponding t-value is used to generate $\mathbf{M}(t)$.

Equation (6) may be degenerate. In particular, all coefficients are identically zero when $b_0 = b_1$ and $c_0 = c_1$. In this case, the equation $L'(t) = 0$ is equivalent to $at - b_0 = 0$, so $t = b_0/a$.

2.2 Analysis of Convexity

More analysis of the minimization problem will be helpful for understanding the general case when the number of edges emanating from $\mathbf{A}$ and between $\langle \mathbf{A}, \mathbf{P}_0 \rangle$ and $\langle \mathbf{A}, \mathbf{P}_1 \rangle$ is more than one. The second derivative of the length function is

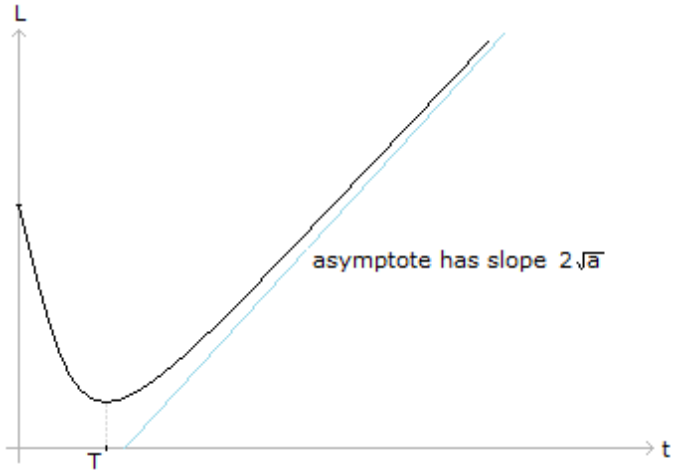
$$L''(t) = \frac{ac_0 - b_0^2}{\left(at^2 - 2b_0 t + c_0\right)^{3/2}} + \frac{ac_1 - b_1^2}{\left(at^2 - 2b_1 t + c_1\right)^{3/2}} \tag{7}$$

Notice that

$$ac_i - b_i^2 = (\mathbf{D} \cdot \mathbf{D})(\mathbf{\Delta}_i \cdot \mathbf{\Delta}_i) - (\mathbf{D} \cdot \mathbf{\Delta}_i)^2 = (\mathbf{D} \times \mathbf{\Delta}_i) \cdot (\mathbf{D} \times \mathbf{\Delta}_i) > 0 \tag{8}$$

Consequently, $L''(t) > 0$ for all $t \geq 0$. A function $L(t)$ for which $L''(t) > 0$ is said to be a *convex function*. Convex functions have at most one value T for which $L'(T) = 0$. If such a T exists, then $L''(T) > 0$, so $L(T)$ is a local minimum. By the convexity, $L(T)$ must be the unique global minimum. If $L(t)$ is restricted to an interval $[t_{\min}, t_{\max}]$, then the minimum of L on the interval is either $L(t_{\min})$, $L(t_{\max})$, or $L(T)$ when $T \in [t_{\min}, t_{\max}]$. Figure 2.2 shows the graph of $L(t)$ for $t \geq 0$ when $L'(0) < 0$.

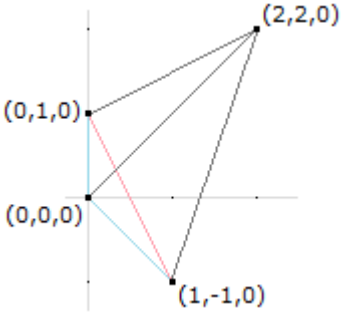
Figure 2.2 The graph of the path-length function $L(t)$ for which the global minimum occurs at $T > 0$. In the limit, $L(\infty) = \infty$, $L'(\infty) = 2\sqrt{a}$, and $L''(\infty) = 0$, which imply that the graph has an asymptote as shown.



2.3 Examples

Example 2.1 Let $\mathbf{P}_0 = (1, -1, 0)$, $\mathbf{P}_1 = (0, 1, 0)$, $\mathbf{A} = (0, 0, 0)$, and $\mathbf{B} = (2, 2, 0)$. Figure 2.3 shows the configuration.

Figure 2.3 Finding the shortest path from $\mathbf{P}_0 = (1, -1, 0)$ to $\mathbf{P}_1 = (0, 1, 0)$. The blue segments are the initial path, which is shortest among the edge-only paths connecting $\mathbf{P}_0$ and $\mathbf{P}_1$. The red segment is the shortest path.



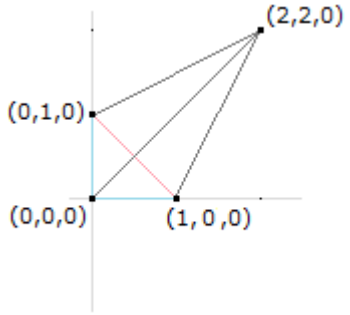
The derived quantities are $\mathbf{D} = (2, 2, 0)$, $\mathbf{\Delta}_0 = (1, -1, 0)$, $\mathbf{\Delta}_1 = (0, 1, 0)$, $a = 8$, $b_0 = 1$, $b_1 = 2$, $c_0 = 2$, and $c_1 = 1$. The length function and its derivative are

$$L(t) = \left(8t^2 + 2\right)^{1/2} + \left(8t^2 - 4t + 1\right)^{1/2}, \quad L'(t) = \frac{8t}{\left(8t^2 + 2\right)^{1/2}} + \frac{8t - 2}{\left(8t^2 - 4t + 1\right)^{1/2}} \tag{9}$$

Observe that $L'(0) = -2 < 0$, so the graph of $L(t)$ is similar to that shown in Figure 2.2. Consequently, there must be a shorter path from $\mathbf{P}_0$ to $\mathbf{P}_1$ that passes through an interior point of the edge $\langle \mathbf{A}, \mathbf{B} \rangle$. The quadratic equation in Equation (6) is $96t^2 - 64t + 8 = 0$ and has roots $r_0 = 1/6$ and $r_1 = 1/2$. The root r_0 leads to the global minimum shown in Figure 2.2. The root r_1 is extraneous and was generated by the squaring operation that led to the quadratic equation. However, in a numerical program we do not know which are the extraneous roots, so we evaluate all: $L(r_0) = \sqrt{5}$ and $L(r_1) = 3$. The length of the shorter path is $L(r_0) = \sqrt{5}$, which is consistent with our expectation that the path should be a line segment connecting $(1, -1, 0)$ and $(0, 1, 0)$. The middle point of the path is $\mathbf{M}(r_0) = (1/3, 1/3, 0)$. $\bowtie$

Example 2.2 This example shows that the quadratic equation of Equation (6) can be degenerate. Let $\mathbf{P}_0 = (1, 0, 0)$, $\mathbf{P}_1 = (0, 1, 0)$, $\mathbf{A} = (0, 0, 0)$, and $\mathbf{B} = (2, 2, 0)$. Figure 2.4 shows the configuration.

Figure 2.4 Finding the shortest path from $\mathbf{P}_0 = (1, 0, 0)$ to $\mathbf{P}_1 = (0, 1, 0)$. The blue segments are the initial path, which is shortest among the edge-only paths connecting $\mathbf{P}_0$ and $\mathbf{P}_1$. The red segment is the shortest path.



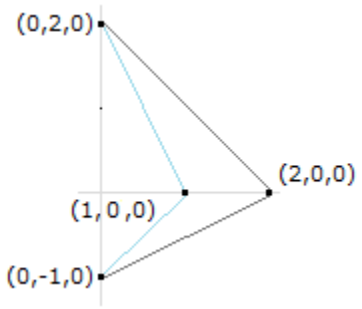
The derived quantities are $\mathbf{D} = (2, 2, 0)$, $\mathbf{\Delta}_0 = (1, 0, 0)$, $\mathbf{\Delta}_1 = (0, 1, 0)$, $a = 8$, $b_0 = b_1 = 2$, and $c_0 = c_1 = 1$. The length function and its derivative are

$$L(t) = 2\left(8t^2 - 4t + 1\right)^{1/2}, \quad L'(t) = \frac{2(8t - 2)}{\left(8t^2 - 4t + 1\right)^{1/2}} \tag{10}$$

Observe that $L'(0) = -4 < 0$, so the graph of $L(t)$ is similar to that shown in Figure 2.2. Consequently, there must be a shorter path from $\mathbf{P}_0$ to $\mathbf{P}_1$ that passes through an interior point of the edge $\langle \mathbf{A}, \mathbf{B} \rangle$. The quadratic equation in Equation (6) degenerates to the tautology $0 = 0$. Using Equation (10) directly, the derivative is zero at $t = 1/4$. The minimum length is $L(1/4) = \sqrt{2}$, which is consistent with our expectation that the path should be a line segment connecting $(1, 0, 0)$ and $(0, 1, 0)$. The middle point of the path is $\mathbf{M}(1/4) = (1/2, 1/2, 0)$. $\bowtie$

Example 2.3 This example shows that the initial path consisting of edges only might be the shortest path. Let $\mathbf{P}_0 = (0, -1, 0)$, $\mathbf{P}_1 = (0, 2, 0)$, $\mathbf{A} = (1, 0, 0)$, and $\mathbf{B} = (2, 0, 0)$. Figure 2.5 shows the configuration.

Figure 2.5 Finding the shortest path from $\mathbf{P}_0 = (0, -1, 0)$ to $\mathbf{P}_1 = (0, 2, 0)$. The blue segments are the initial path, which is shortest among the edge-only paths connecting $\mathbf{P}_0$ and $\mathbf{P}_1$. Those segments also form the shortest path that lies inside the triangles.



The derived quantities are $\mathbf{D} = (1, 0, 0)$, $\mathbf{\Delta}_0 = (-1, -1, 0)$, $\mathbf{\Delta}_1 = (2, -1, 0)$, $a = 1$, $b_0 = b_1 = -1$, $c_0 = 2$, and $c_1 = 5$. The length function and its derivative are

$$L(t) = \left(t^2 + 2t + 2\right)^{1/2} + \left(t^2 + 2t + 5\right)^{1/2}, \quad L'(t) = \frac{t+1}{(t^2 + 2t + 2)^{1/2}} + \frac{t+1}{(t^2 + 2t + 5)^{1/2}} \tag{11}$$

Observe that $L'(0) = 1/\sqrt{2} + 1/\sqrt{5} > 0$. The convexity of $L(t)$ guarantees that $L(t)$ is increasing for $t \geq 0$, in which case the minimum length is $L(0)$ and the blue-colored path in Figure 2.5 is the shortest path. $\bowtie$

2.4 Numerical Implementation

As indicated previously, the t-value that minimizes $L(t)$ on $[0, 1]$ is either $t = 0$, $t = 1$, or a value $t \in (0, 1)$ that is a root to the quadratic polynomial of Equation (6). Example 2.2, however, shows that the quadratic polynomial may be degenerate. To avoid the tedious details of processing the quadratic polynomial to trap degeneracies, and to be robust, a numerical implementation can use bisection to locate the root to $L'(t) = 0$.

The inputs to the algorithm are $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{A}$, and $\mathbf{B}$. The outputs are $t_{\min} \in [0, 1]$, the t-value at which $L(t)$ is minimum; the length of the shortest path, $L_{\min}$; and the point $\mathbf{M} = \mathbf{A} + t_{\min}(\mathbf{B} - \mathbf{A})$ for which $\langle \mathbf{P}_0, \mathbf{M}, \mathbf{P}_1 \rangle$ is the shortest path.

If $L'(0) \geq 0$, then $t_{\min} = 0$ and $\mathbf{M} = \mathbf{A}$. If $L'(0) < 0$, then evaluate $L'(1)$. If $L'(1) \leq 0$, by the convexity of $L(t)$, it must be that $L(1) < L(0)$. In this case, $t_{\min} = 1$ and $\mathbf{M} = \mathbf{B}$. The final case is $L'(1) > 0$. Because $L'(0) < 0$ and $L'(1) > 0$, there must be a root to $L(t) = 0$ on the interval $(0, 1)$. By the convexity of $L(t)$, the root must be unique. Bisection is used to locate the root. The implementation allows you to specify the number of digits of accuracy for the root, call this $\delta > 0$. The maximum number of iterations to locate the root using bisection is $n = \lceil \delta \log_2(10) \rceil$, where the notation $\lceil x \rceil$ refers to the smallest integer larger than x. For example, if you want 8 digits of accuracy, then the maximum number of iterations is

$n = \lceil 8 \log_2(10) \rceil = 27$. The source code computes all iterations and does not attempt to exit early based on a convergence criterion. Alternate implementations may be designed to speed up the root finding.

Pseudocode for computing the shortest path through two triangles is shown next. The variables prefixed with an `m` are assumed to be accessible by the function `FDer`. The ampersands in the `GetPath` function indicate that the corresponding variables are the outputs of the function.

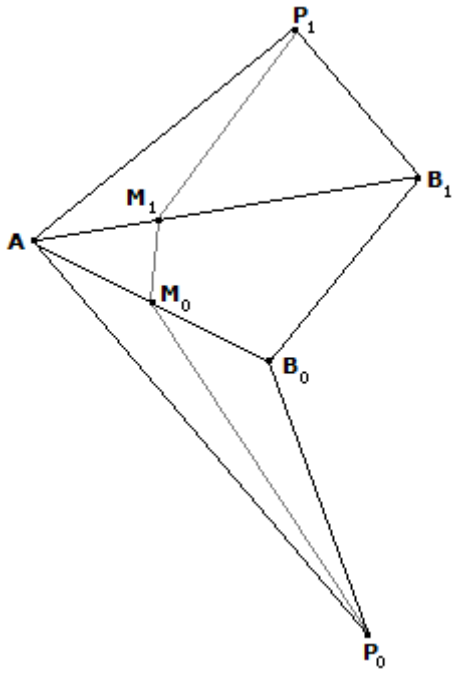
```
void GetPath (Vector3 P0, Vector3 P1, Vector3 A, Vector3 B, Real& tmin, Real& lmin, Vector3& M)
{
    Vector3 Delta0 = P0 - A;
    Vector3 Delta1 = P1 - A;
    Vector3 D = B - A;
    Real mDelta0SqrLength = Dot(Delta0,Delta0);
    Real mDelta1SqrLength = Dot(Delta1,Delta1);
    Real mDSqrLength = Dot(D,D);
    Real mDDotDelta0 = Dot(D,Delta0);
    Real mDDotDelta1 = Dot(D,Delta1);

    Real der0 = FDer(0);
    if (der0 >= 0)
    {
        tmin = 0;
        lmin = sqrt(mDelta0SqrLength) + sqrt(mDelta1SqrLength);
        M = A;
        return;
    }

    Real der1 = FDer(1);
    if (der1 <= 0)
    {
        tmin = 1;
        lmin = Length(P0 - B) + Length(P1 - B);
        M = B;
        return;
    }

    // der0 < 0 and der1 > 0, so bisect to find the root
    tmin = GetFRoot(0,der0,1,der1);
    M = A + tmin * D;
    lmin = Length(P0 - M) + Length(P1 - M);
}

Real FDer(Real s) const
{
    Real numer0 = mDSqrLength * s - mDDotDelta0;
    Real numer1 = mDSqrLength * s - mDDotDelta1;
    Real denom0 = sqrt(s*(numer0 - mDDotDelta0) + mDelta0SqrLength);
    Real denom1 = sqrt(s*(numer1 - mDDotDelta1) + mDelta1SqrLength);
    return numer0/denom0 + numer1/denom1;
}

Real GetFRoot (Real s0, Real der0, Real s1, Real der1)
{
    int numIterations = <user-defined parameter>;
    Real root = 0;
    for (int i = 0; i < numIterations; i++)
    {
        root = (s0 + s1)/2;
        Real derRoot = FDer(root);
        Real product = derRoot * der0;
        if (product < 0)
        {
            s1 = root;
            der1 = derRoot;
        }
        else if (product > 0)
        {
            s0 = root;
            der0 = derRoot;
```

```
        }
        else
        {
            break;
        }
    }
    return root;
}
```

3 Three Triangles

A typical configuration of three triangles is shown in Figure 3.1.

Figure 3.1 Shortest paths between two vertices, $\mathbf{P}_0$ and $\mathbf{P}_1$, of a triple of triangles. The shortest path is either the left-edge path, $\langle \mathbf{P}_0, \mathbf{A}, \mathbf{P}_1 \rangle$, the right-edge path, $\langle \mathbf{P}_0, \mathbf{B}_0, \mathbf{B}_1, \mathbf{P}_1 \rangle$, or a path through the interior, $\langle \mathbf{P}_0, \mathbf{M}_0, \mathbf{M}_1, \mathbf{P}_1 \rangle$.



3.1 Setting Up the Minimization Problem

As in the two-triangle example, this may be set up as a minimization problem. Parameterize the points on the shared edges as

$$\begin{aligned} \mathbf{M}_0(t_0) &= \mathbf{A} + t_0(\mathbf{B}_0 - \mathbf{A}), \quad t_0 \in [0,1] \\ \mathbf{M}_1(t_1) &= \mathbf{A} + t_1(\mathbf{B}_1 - \mathbf{A}), \quad t_1 \in [0,1] \end{aligned} \tag{12}$$

The length of the path $\langle \mathbf{P}_0, \mathbf{M}_0(t_0), \mathbf{M}_1(t_1), \mathbf{P}_1 \rangle$ is

$$\begin{aligned} L(t_0,t_1) &= |\mathbf{M}_0(t_0) - \mathbf{P}_0| + |\mathbf{M}_1(t_1) - \mathbf{M}_0(t_0)| + |\mathbf{P}_1 - \mathbf{M}_1(t_1)| \\ &= |t_0\mathbf{D}_0 - \mathbf{\Delta}_0| + |t_1\mathbf{D}_1 - t_0\mathbf{D}_0| + |t_1\mathbf{D}_1 - \mathbf{\Delta}_1| \end{aligned} \tag{13}$$

where $\mathbf{D}_i = \mathbf{B}_i - \mathbf{A}$ and $\mathbf{\Delta}_i = \mathbf{P}_i - \mathbf{A}$ for $i = 0, 1$.

The pair (t_0, t_1) that minimizes $L(t_0, t_1)$ satisfies one of the conditions

1. The gradient is zero, $\nabla L(t_0, t_1) = (0, 0)$.
2. The point is interior to one of the four edges on the parameter domain: $(t_0, 0)$, $(t_0, 1)$, $(0, t_1)$, or $(1, t_1)$, where $t_0 \in (0, 1)$ and $t_1 \in (0, 1)$.
3. The point is one of the four corners of the parameter domain: $(0, 0)$, $(1, 0)$, $(0, 1)$, or $(1, 1)$.

We can rule out immediately that the minimum value occurs on the edges $(t_0, 0)$ and $(0, t_1)$. For example, suppose the minimum occurs at $(t_0, 0)$ for some $t_0 \in (0, 1]$. The geodesic path would be $\langle \mathbf{P}_0, \mathbf{M}_0(t_0), \mathbf{A}, \mathbf{P}_1 \rangle$. However, notice that the triangle edge $\langle \mathbf{P}_0, \mathbf{A} \rangle$ has smaller length than the sum of the lengths of the triangle edges $\langle \mathbf{P}_0, \mathbf{M}_0(t_0) \rangle$ and $\langle \mathbf{M}_0(t_0), \mathbf{A} \rangle$, so in fact $\langle \mathbf{P}_0, \mathbf{A}, \mathbf{P}_1 \rangle$ is a shorter path than the one implied by choosing $t_0 \in (0, 1]$ and $t_1 = 0$. A similar argument rules out the minimum occuring for $t_0 = 0$ and $t_1 \in (0, 1]$.

The gradient of L is a two-tuple consisting of the first-order partial derivatives of L with respect to t_0 and t_1. These are

$$\frac{\partial L}{\partial t_0} = \frac{|\mathbf{D}_0|^2 t_0 - \mathbf{D}_0 \cdot \mathbf{\Delta}_0}{|t_0\mathbf{D}_0 - \mathbf{\Delta}_0|} + \frac{|\mathbf{D}_0|^2 t_0 - \mathbf{D}_0 \cdot \mathbf{D}_1 t_1}{|t_0\mathbf{D}_0 - t_1\mathbf{D}_1|} \tag{14}$$

and

$$\frac{\partial L}{\partial t_1} = \frac{|\mathbf{D}_1|^2 t_1 - \mathbf{D}_0 \cdot \mathbf{D}_1 t_0}{|t_0\mathbf{D}_0 - t_1\mathbf{D}_1|} + \frac{|\mathbf{D}_1|^2 t_1 - \mathbf{D}_1 \cdot \mathbf{\Delta}_1}{|t_1\mathbf{D}_1 - \mathbf{\Delta}_1|} \tag{15}$$

Although $L(t_0, t_1)$ is a continuous function at $(0, 0)$, its partial derivatives are discontinuous at $(0, 0)$.

To solve $\nabla L = (0, 0)$, it is convenient to define $a_i = |\mathbf{D}_i|^2$, $b_i = \mathbf{D}_i \cdot \mathbf{\Delta}_i$, $c_i = |\mathbf{\Delta}_i|^2$, $\ell_i = |t_i\mathbf{D}_i - \mathbf{\Delta}_i|$, $d_0 = \mathbf{D}_0 \cdot \mathbf{D}_1$, and $\lambda_0 = |t_0\mathbf{D}_0 - t_1\mathbf{D}_1|$. The two equations to solve are

$$\frac{a_0 t_0 - b_0}{\ell_0} + \frac{a_0 t_0 - d_0 t_1}{\lambda_0} = 0, \quad \frac{a_1 t_1 - d_0 t_0}{\lambda_0} + \frac{a_1 t_1 - b_1}{\ell_1} = 0 \tag{16}$$

In each equation, subtracting a term to the right-hand side, squaring the resulting sides, and subtracting to the left-hand sides leads to the polynomial equations

$$\begin{aligned} p_0(t_0, t_1) &= \lambda_0^2 (a_0 t_0 - b_0)^2 - \ell_0^2 (a_0 t_0 - d_0 t_1)^2 = 0 \\ p_1(t_0, t_1) &= \lambda_0^2 (a_1 t_1 - b_1)^2 - \ell_1^2 (a_1 t_1 - d_0 t_0)^2 = 0 \end{aligned} \tag{17}$$

Some algebra will show that

$$\begin{aligned} p_0 &= (a_0c_0 - b_0^2)(2d_0t_1 - a_0t_0)t_0 + (a_1b_0^2 - c_0d_0^2)t_1^2 + (a_0a_1 - d_0^2)(a_0t_0 - 2b_0)t_0t_1^2 \\ p_1 &= (a_1c_1 - b_1^2)(2d_0t_0 - a_1t_1)t_1 + (a_0b_1^2 - c_1d_0^2)t_0^2 + (a_0a_1 - d_0^2)(a_1t_1 - 2b_1)t_1t_0^2 \end{aligned} \tag{18}$$

It may also be shown that

$$\begin{aligned} \sigma &= a_0a_1 - d_0^2 &&= |\mathbf{D}_0 \times \mathbf{D}_1|^2 \\ s_0 &= a_0c_0 - b_0^2 &&= |\mathbf{D}_0 \times \boldsymbol{\Delta}_0|^2 \\ s_1 &= a_1c_1 - b_1^2 &&= |\mathbf{D}_1 \times \boldsymbol{\Delta}_1|^2 \\ w_0 &= a_1b_0^2 - c_0d_0^2 &&= c_0|\mathbf{D}_0 \times \mathbf{D}_1|^2 - a_1|\mathbf{D}_0 \times \boldsymbol{\Delta}_0|^2 \\ w_1 &= a_0b_1^2 - c_1d_0^2 &&= c_1|\mathbf{D}_0 \times \mathbf{D}_1|^2 - a_0|\mathbf{D}_1 \times \boldsymbol{\Delta}_1|^2 \end{aligned} \tag{19}$$

where the leftmost equalities define the leftmost variables. The polynomials are rewritten as

$$\begin{aligned} p_0 &= s_0(2d_0t_1 - a_0t_0)t_0 + w_0t_1^2 + \sigma(a_0t_0 - 2b_0)t_0t_1^2 \\ p_1 &= s_1(2d_0t_0 - a_1t_1)t_1 + w_1t_0^2 + \sigma(a_1t_1 - 2b_1)t_1t_0^2 \end{aligned} \tag{20}$$

The polynomials are each degree 4, so by eliminating t_1, we obtain a polynomial in t_0 whose degree is at most 16. You may use the *resultant* of two polynomials for the elimination. As it turns out in this example, the degree is 8 but the polynomial contains a factor of t_0^4:

$$p(t_0) = t_0^4(k_0 + k_1t_0 + k_2t_0^2 + k_3t_0^3 + k_4t_0^4) \tag{21}$$

where

$$\begin{aligned} k_0 &= (a_0a_1s_0s_1 - w_0w_1)^2 - 4s_0s_1d_0^2(a_0s_1 + w_1)(a_1s_0 + w_0) \\ k_1 &= 4\sigma(b_0w_1(a_0a_1s_0s_1 - w_0w_1) + d_0b_1s_0(a_0s_1(a_1s_0 + w_0) + w_0(a_0s_1 + w_1)) \\ &\quad +2d_0^2b_0s_0s_1(a_0s_1 + w_1)) \\ k_2 &= 2\sigma(2\sigma(b_0^2w_1^2 - a_0b_1^2s_0w_0) - a_0(a_1s_0 + w_1)(a_0a_1s_0s_1 - w_0w_1) \\ &\quad -4d_0b_0b_1s_0\sigma(2a_0s_1 + w_1) + 2d_0^2s_0(a_1s_0 - a_0s_1)(a_0s_1 + w_1)) \\ k_3 &= 4a_0\sigma^2(b_0(2b_1^2s_0\sigma - w_1(a_1s_0 + w_1)) + d_0b_1s_0((a_0s_1 - a_1s_0) + (a_0s_1 + w_1))) \\ k_4 &= a_0^2\sigma^2[(a_1s_0 + w_1)^2 - 4b_1^2\sigma s_0] \end{aligned} \tag{22}$$

For each root $\bar{t}_0$ of p, the polynomials

$$q_0(t_1) = p_0(\bar{t}_0, t_1), \quad q_1(t_1) = p_1(\bar{t}_0, t_1) \tag{23}$$

are (at most) quadratic in t_1. Compute roots $\bar{t}_1$ that are common to both q_0 and q_1. As long as $(\bar{t}_0, \bar{t}_1) \in [0,1]^2$, the value $L(\bar{t}_0, \bar{t}_1)$ is a candidate for the minimum of L.

To check for a minimum on the boundary of the (t_0, t_1) domain, we have already ruled out the boundary when $t_0 = 0$ and when $t_1 = 0$. Now choose $t_1 = 1$. The minimization along this boundary has already been discussed previously. We are back to the case of two triangles, in this case, $\langle \mathbf{P}_0, \mathbf{A}, \mathbf{B}_0, \rangle$ and $\langle \mathbf{B}_0, \mathbf{A}, \mathbf{B}_1 \rangle$. The reduction is similar for the boundary $t_0 = 1$.

3.2 Analysis of Convexity

As in the case of two triangles, we may show that the length function $L(t_0, t_1)$ is convex. For a function of two variables, we do so by showing that the *Hessian matrix*, the matrix of second-order partial derivatives, is positive definite.

Recall that the first-order partial derivatives are

$$\begin{aligned} L_{t_0} &= (a_0 t_0 - b_0)/\ell_0 + (a_0 t_0 - d_0 t_1)/\lambda_0 \\ L_{t_1} &= (a_1 t_1 - d_0 t_0)/\lambda_0 + (a_1 t_1 - b_1)/\ell_1 \end{aligned} \tag{24}$$

The second-order partial derivatives are

$$\begin{aligned} L_{t_0 t_0} &= (a_0 c_0 - b_0^2)/\ell_0^3 + (a_0 a_1 - d_0^2) t_1^2/\lambda_0^3 = s_0/\ell_0^3 + \sigma t_1^2/\lambda_0^3 \\ L_{t_0 t_1} &= -(a_0 a_1 - d_0^2) t_0 t_1/\lambda_0^3 = -\sigma t_0 t_1/\lambda_0^3 \\ L_{t_1 t_1} &= (a_0 a_1 - d_0^2) t_0^2/\lambda_0^3 + (a_1 c_1 - b_1^2)/\ell_1^3 = \sigma t_0^2/\lambda_0^3 + s_1/\ell_1^3 \end{aligned} \tag{25}$$

where s_0, s_1, and σ are defined in Equation (19). Vector algebra identities lead to Thus, $L_{t_0 t_0} > 0$, $L_{t_0 t_1} < 0$, and $L_{t_1 t_1} > 0$ for all $t_0 > 0$ and $t_1 > 0$.

Now define $e_i = s_i/\ell_i^3 > 0$ and $f_0 = \sigma/\lambda_0^3 > 0$. For $t_0 > 0$ and $t_1 > 0$, the second-order derivatives are continuous, so the Hessian matrix is the symmetric matrix

$$H = \begin{bmatrix} L_{t_0 t_0} & L_{t_0 t_1} \\ L_{t_0 t_1} & L_{t_1 t_1} \end{bmatrix} = \begin{bmatrix} e_0 + f_0 t_1^2 & -f_0 t_0 t_1 \\ -f_0 t_0 t_1 & e_1 + f_0 t_0^2 \end{bmatrix} \tag{26}$$

The determinant is

$$\det(H) = e_0 e_1 + f_0 t_0^2 e_0 + f_0 t_1^2 e_1 > 0 \tag{27}$$

The following theorem from Matrix Algebra is an important classification of positive definite matrices.

> THEOREM. Let H be a real-valued symmetric matrix. Let H_i be the leading principal submatrix of H determined by the first i rows and columns of H. H is positive definite if and only if $\det(H_i) > 0$ for all i.

In our example,

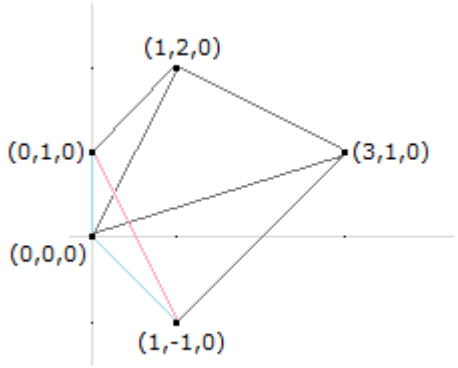
$$H_1 = \begin{bmatrix} e_0 + f_0 t_1^2 \end{bmatrix}, \quad H_2 = H \tag{28}$$

It is clear that $\det(H_1) = e_0 + f_0 t_1^2 > 0$, and we already showed that $\det(H_2) = \det(H) > 0$. The theorem guarantees that H is positive definite, in which case $L(t_0, t_1)$ is a convex function.

3.3 Examples

Example 3.1 Let $\mathbf{P}_0 = (1, -1, 0)$, $\mathbf{P}_1 = (0, 1, 0)$, $\mathbf{A} = (0, 0, 0)$, $\mathbf{B}_0 = (3, 1, 0)$, and $\mathbf{B}_1 = (1, 2, 0)$. Figure 3.2 shows the configuration.

Figure 3.2 Finding the shortest path from $\mathbf{P}_0 = (1, -1, 0)$ to $\mathbf{P}_1 = (0, 1, 0)$. The blue segments are the initial path, which is shortest among the edge-only paths connecting $\mathbf{P}_0$ and $\mathbf{P}_1$. The red segment is the shortest path.



The derived quantities are $\mathbf{D}_0 = (3, 1, 0)$, $\mathbf{D}_1 = (1, 2, 0)$, $\mathbf{\Delta}_0 = (1, -1, 0)$, $\mathbf{\Delta}_1 = (0, 1, 0)$, $a_0 = 10$, $a_1 = 5$, $b_0 = 2$, $b_1 = 2$, $c_0 = 2$, $c_1 = 1$, and $d_0 = 5$. The length function is

$$L(t_0, t_1) = \sqrt{(3t_0 - 1)^2 + (t_0 + 1)^2} + \sqrt{(t_1 - 3t_0)^2 + (2t_1 - t_0)^2} + \sqrt{t_1^2 + (2t_1 - 1)^2} \tag{29}$$

The auxiliary constants in Equation (19) are $\sigma = 25$, $s_0 = 16$, $s_1 = 1$, $w_0 = -30$, and $w_1 = 15$. The coefficients of Equation (22) are $k_0 = -437500$, $k_1 = 3750000$, $k_2 = 4875000$, $k_3 = -91250000$, and $k_4 = 164062500$. For an example whose points have components on the order of 1, the magnitudes of the coefficients of $p(t_0)$ should already be of concern in a numerical program.

The polynomial actually factors as

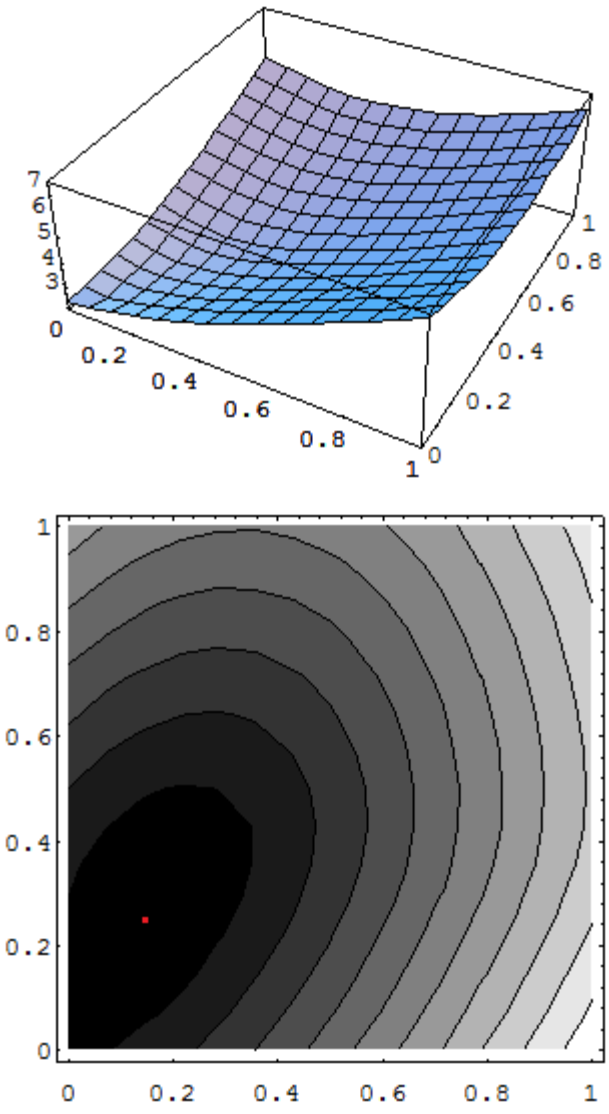
$$p(t_0) = -62500t_0^4(1 + 5t_0)(7 - 25t_0)(1 - 7t_0)(1 - 3t_0)$$

and has t_0-roots (listed in increasing order): $-1/5$, 0, $1/7$, $7/25$, and $1/3$. The only t_0-roots of interest are those for which $t_0 \geq 0$, which amounts to 0, $1/7$, $7/25$, and $1/3$.

The value $t_0 = 1/7$ leads to $q_0(t_1) = 160(-1 + 7t_1 - 12t_1^2)/49$ and $q_1(t_1) = 15(1 - 2t_1 - 8t_1^2)/49$. The common root is $t_1 = 1/4$ and the corresponding length is $L = \sqrt{5}$. The value $t_0 = 7/25$ leads to $q_0(t_1) = 32(-49 + 175t_1 - 150t_1^2)/125$ and $q_1(t_1) = 3(49 - 210t_1 + 200t_1^2)/125$. The common root is $t_1 = 7/10$ and the corresponding length is $L = 2\sqrt{13/5}$. The value $t_0 = 1/3$ leads to $q_0(t_1) = 160(-1 + 3t_1 - 2t_1^2)/9$ and $q_1(t_1) = 5(3 - 14t_1 + 16t_1^2)/9$. The common root is $t_1 = 1/2$ and the corresponding length is $L = 8/3$. The smallest of all the lengths is $L = \sqrt{5}$, and the minimum occurs at $(t_0, t_1) = (1/7, 1/4)$. The interior edge points are $\mathbf{M}_0 = (3/7, 1/7, 0)$ and $\mathbf{M}_1 = (1/4, 1/3, 0)$.

Figure 3.3 shows a 3D plot and a contour plot of $L(t_0, t_1)$.

Figure 3.3 Top: A 3D plot of $L(t_0, t_1)$ from Equation (29). The t_0-axis is the slanted line from the origin and goes diagonally down the page. The t_1-axis is the slanted line from the origin and goes diagonally up the page. The plot is for the domain $(t_0, t_1) \in [0, 1]^2$. Bottom: A contour plot of $L(t_0, t_1)$ on the domain $[0, 1]^2$.

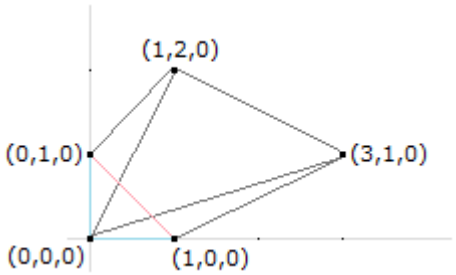


The location of the minimum is shown as a red dot. ⋈

Example 3.2 This example shows that the quadric equation of Equation (21) can be degenerate. Let $\mathbf{P}_0 = (1, 0, 0)$, $\mathbf{P}_1 = (0, 1, 0)$, $\mathbf{A} = (0, 0, 0)$, $\mathbf{B}_0 = (3, 1, 0)$, and $\mathbf{B}_1 = (1, 2, 0)$. Figure 3.4 shows the

configuration.

Figure 3.4 Finding the shortest path from $\mathbf{P}_0 = (1, 0, 0)$ to $\mathbf{P}_1 = (0, 1, 0)$. The blue segments are the initial path, which is shortest among the edge-only paths connecting $\mathbf{P}_0$ and $\mathbf{P}_1$. The red segment is the shortest path.



The derived quantities are $\mathbf{D}_0 = (3, 1, 0)$, $\mathbf{D}_1 = (1, 2, 0)$, $\mathbf{\Delta}_0 = (1, 0, 0)$, $\mathbf{\Delta}_1 = (0, 1, 0)$, $a_0 = 10$, $a_1 = 5$, $b_0 = 3$, $b_1 = 2$, $c_0 = 1$, $c_1 = 1$, and $d_0 = 5$. The length function is

$$L(t_0, t_1) = \sqrt{(3t_0 - 1)^2 + t_0^2} + \sqrt{(t_1 - 3t_0)^2 + (2t_1 - t_0)^2} + \sqrt{t_1^2 + (2t_1 - 1)^2} \tag{30}$$

and the first-order partial derivatives are

$$\begin{aligned} L_{t_0}(t_0, t_1) &= \frac{10t_0 - 3}{\sqrt{(3t_0-1)^2+t_0^2}} + \frac{10t_0 - 5t_1}{\sqrt{(t_1-3t_0)^2+(2t_1-t_0)^2}} \\ L_{t_1}(t_0, t_1) &= \frac{5t_1 - 5t_0}{\sqrt{(t_1-3t_0)^2+(2t_1-t_0)^2}} + \frac{5t_1 - 2}{\sqrt{t_1^2+(2t_1-1)^2}} \end{aligned} \tag{31}$$

The auxiliary constants in Equation (19) are $\sigma = 25$, $s_0 = 1$, $s_1 = 1$, $w_0 = 20$, and $w_1 = 15$. The coefficients of Equation (22) are $k_0 = k_1 = k_2 = k_3 = k_4 = 0$, so the equation is a tautology.

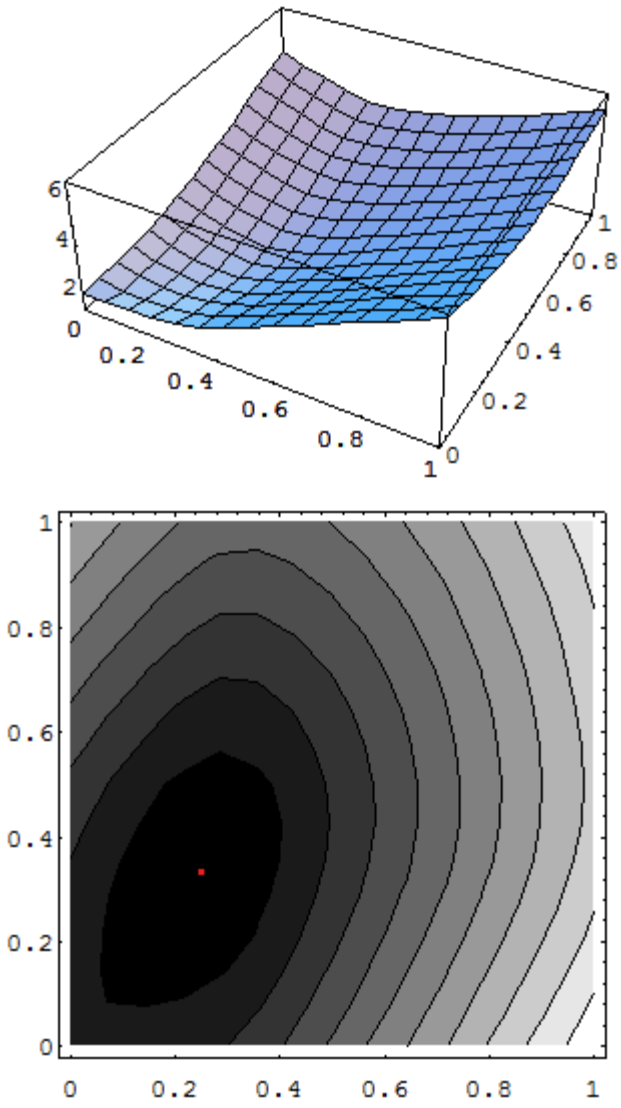
A closer analysis of the polynomials of Equation (20) shows that they factor as

$$\begin{aligned} p_0 &= 10(5t_0t_1 - t_0 - t_1)(5t_0t_1 + t_0 - 2t_1) \\ p_1 &= 5(5t_0t_1 - t_0 - t_1)(5t_0t_1 - 3t_0 + t_1) \end{aligned} \tag{32}$$

The degeneracy of Equation (21) is due to the common factor $5t_0t_1 - t_0 - t_1$ of the two polynomials. There are infinitely many solutions to $p_0 = p_1 = 0$ due to this factor. In particular, make the change of variables $t_1 = rt_0$ for $r > 0$; then the roots are $(t_0, t_1) = ((r + 1)/(5r), (r + 1)/5)$. Substituting this into the partial derivatives of Equation (31), $L_{t_0} = 0$ for $r = 2$ and $L_{t_1} = 0$ for $r = 1$, but both partial derivatives cannot be made zero for a common value of r. That is, there is no value of r for which both partial derivatives are zero.

The other roots of $p_0 = p_1 = 0$ are found by solving $5t_0t_1 + t_0 - 2t_1 = 0$ and $5t_0t_1 - 3t_0 + t_1 = 0$. Subtracting the second equation from the first produces $4t_0 - 3t_1 = 0$. Thus, $t_1 = 4t_0/3$. Substitute this in the first equation to obtain $t_0 = 0$, in which case $t_1 = 0$, and $t_0 = 1/4$, in which case $t_1 = 1/3$. Indeed, the global minimum of L occurs at $(t_0, t_1) = (1/4, 1/3)$ and is $L_{\min} = \sqrt{2}$. The interior edge points are $\mathbf{M}_0 = (3/4, 1/4, 0)$ and $\mathbf{M}_1 = (1/3, 2/3, 0)$. Figure 3.5 shows a 3D plot and a contour plot of L.

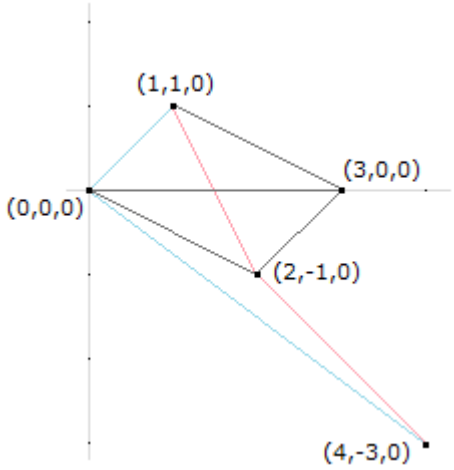
Figure 3.5 Top: A 3D plot of $L(t_0, t_1)$ from Equation (30). The t_0-axis is the slanted line from the origin and goes diagonally down the page. The t_1-axis is the slanted line from the origin and goes diagonally up the page. The plot is for the domain $(t_0, t_1) \in [0, 1]^2$. Bottom: A contour plot of $L(t_0, t_1)$ on the domain $[0, 1]^2$.



The location of the global minimum is shown as a red dot. ⋈

Example 3.3 Let $\mathbf{P}_0 = (4, -3, 0)$, $\mathbf{P}_1 = (1, 1, 0)$, $\mathbf{A} = (0, 0, 0)$, $\mathbf{B}_0 = (2, -1, 0)$, and $\mathbf{B}_1 = (3, 0, 0)$. Figure 3.6 shows the configuration.

Figure 3.6 Finding the shortest path from $\mathbf{P}_0 = (4, -3, 0)$ to $\mathbf{P}_1 = (1, 1, 0)$. The blue segments are the initial path, which is shortest among the edge-only paths connecting $\mathbf{P}_0$ and $\mathbf{P}_1$. The red segments are the shortest path.



The derived quantities are $\mathbf{D}_0 = (2, -1, 0)$, $\mathbf{D}_1 = (3, 0, 0)$, $\mathbf{\Delta}_0 = (4, -3, 0)$, $\mathbf{\Delta}_1 = (1, 1, 0)$, $a_0 = 5$, $a_1 = 9$, $b_0 = 11$, $b_1 = 3$, $c_0 = 25$, $c_1 = 2$, and $d_0 = 6$. The length function is

$$L(t_0, t_1) = \sqrt{(2t_0 - 4)^2 + (t_0 - 3)^2} + \sqrt{(3t_1 - 2t_0)^2 + t_0^2} + \sqrt{(3t_1 - 1)^2 + 1} \tag{33}$$

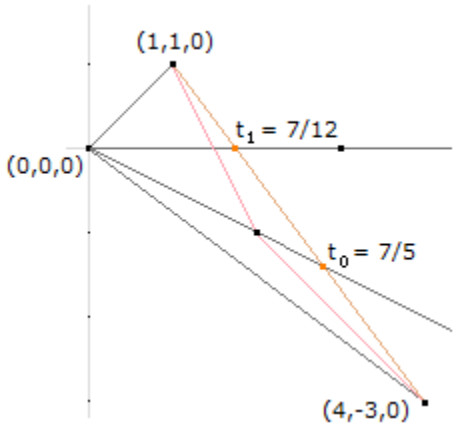
The auxiliary constants in Equation (19) are $\sigma = 9$, $s_0 = 4$, $s_1 = 9$, $w_0 = 189$, and $w_1 = -27$. The coefficients of Equation (22) are $k_0 = 24203529$, $k_1 = -18344556$, $k_2 = -21060810$, $k_3 = 19026900$, and $k_4 = -2460375$. For an example whose points have components on the order of 1, the magnitudes of the coefficients of $p(t_0)$ should already be of concern in a numerical program.

The polynomial actually factors as

$$p(t_0) = -6561t_0^4(t_0 + 1)(5t_0 - 31)(5t_0 - 7)(15t_0 - 17)$$

and has t_0-roots (listed in increasing order): -1, 0, 17/15, 7/5, and 31/5. The only t_0-roots of interest are those for which $t_0 \geq 0$, which amounts to $t_0 = 0$. The positive roots are actually related to choosing $t_0 > 0$ for the ray $t_0(2, -1, 0)$ and choosing $t_1 > 0$ for the ray $t_1(3, 0, 0)$ so that the corresponding points are on the line segment connecting $(4, -3, 0)$ and $(1, 1, 0)$. Figure 3.7 illustrates this.

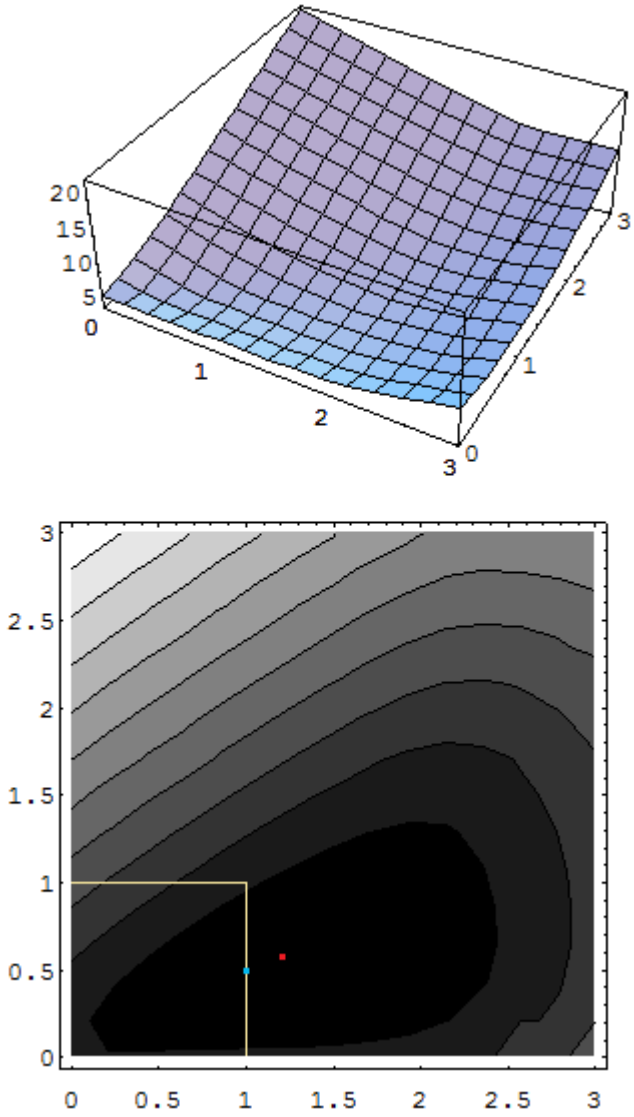
Figure 3.7 Ignoring the triangles, the shortest path from $\mathbf{P}_0 = (4, -3, 0)$ to $\mathbf{P}_1 = (1, 1, 0)$ is the orange segment. The red segments are the shortest path through the triangles.



The value $t_0 = 7/5$ leads to $q_0(t_1) = (336t_1 - 196)/5$ and $q_1(t_1) = (1944t_1^2 + 1134t_1 - 1323)/25$. The common root is $t_1 = 7/12$. A similar construction for $t_0 = 17/15$ leads to $t_1 = -17/6$, which may be discarded since t_1 is negative. Also, $t_0 = 31/5$ has a companion $t_1 = 31/36$, which is in $[0, 1]$ but the path length is longer than that for $(t_0, t_1) = (7/5, 7/12)$.

Figure 3.8 shows a 3D plot and a contour plot of $L(t_0, t_1)$, courtesy of Mathematica. I have hand-added some additional information to the contour plot.

Figure 3.8 Top: A 3D plot of $L(t_0, t_1)$ from Equation (33). The t_0-axis is the slanted line from the origin and goes diagonally down the page. The t_1-axis is the slanted line from the origin and goes diagonally up the page. The plot is for the domain $(t_0, t_1) \in [0, 3]^2$. Bottom: A contour plot of $L(t_0, t_1)$ on the domain $[0, 3]^2$. The required domain, $(t_0, t_1) \in [0, 1]^2$, is bounded by the yellow line segments.



The 3D plot is not sufficiently detailed to show the convexity of L, but the contour plot does illustrate this. In the contour plot, the red dot is at $(t_0, t_1) = (7/5, 7/12)$ and is the location of the global minimum of L, which is $L(7/5, 7/12) = 5$. This location is outside the required domain $(t_0, t_1) \in [0, 1]$. We must clamp

$t_0 = 1$. On that edge of the domain,

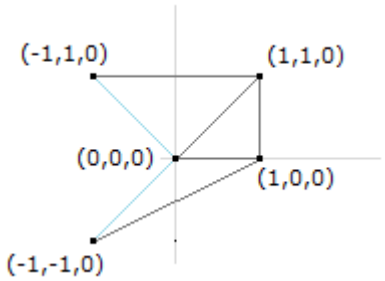
$$L(1, t_1) = \sqrt{8} + \sqrt{(3t_1 - 2)^2 + 1} + \sqrt{(3t_1 - 1)^2 + 1} \tag{34}$$

The minimum of this restricted function occurs at $t_1 = 1/2$, the location shown as a blue dot in Figure 3.8. A quick glance at Figure 3.6 will convince you that the upper red segment intersects the edge $\langle(0, 0, 0), (3, 0, 0)\rangle$ at the point $(3/2, 0, 0)$.

In fact, you should notice that once you clamp $t_0 = 1$, the construction of t_1 is based on the two-triangle configuration of Section 2. This reduction in dimension occurs generally–once you clamp a t_i value to 1, you are at a vertex of the triangle mesh and you may decompose the n-triangle problem into an m_1-triangle and an m_2-triangle problem, where $m_1 + m_2 = n$. In the current example, $n = 3$, $m_1 = 1$ (triangle $\langle \mathbf{A}, \mathbf{P}_0, \mathbf{B}_0\rangle$), and $m_2 = 2$ (triangles $\langle \mathbf{A}, \mathbf{B}_0, \mathbf{B}_1\rangle$ and $\langle \mathbf{A}, \mathbf{B}_1, \mathbf{P}_1\rangle$). The one-triangle case ($m_1 = 1$) is simply a matter of choosing the edge opposite $\mathbf{A}$ as the shortest path connecting the other two vertices of the triangle. ⋈

Example 3.4 This example shows that the minimum length occurs at $(t_0, t_1) = (0, 0)$. Let $\mathbf{P}_0 = (-1, -1, 0)$, $\mathbf{P}_1 = (-1, 1, 0)$, $\mathbf{A} = (0, 0, 0)$, $\mathbf{B}_0 = (1, 0, 0)$, and $\mathbf{B}_1 = (1, 1, 0)$. Figure 3.9 shows the configuration.

Figure 3.9 Finding the shortest path from $\mathbf{P}_0 = (-1, -1, 0)$ to $\mathbf{P}_1 = (-1, 1, 0)$. The blue segments are the initial path, which is shortest among the edge-only paths connecting $\mathbf{P}_0$ and $\mathbf{P}_1$. This path is also the shortest path through the triangles.



The derived quantities are $\mathbf{D}_0 = (1, 0, 0)$, $\mathbf{D}_1 = (1, 1, 0)$, $\mathbf{\Delta}_0 = (-1, -1, 0)$, $\mathbf{\Delta}_1 = (-1, 1, 0)$, $a_0 = 1$, $a_1 = 2$, $b_0 = -1$, $b_1 = 0$, $c_0 = 2$, $c_1 = 2$, and $d_0 = 1$. The length function is

$$L(t_0, t_1) = \sqrt{(t_0 + 1)^2 + 1} + \sqrt{(t_1 - t_0)^2 + t_1^2} + \sqrt{(t_1 + 1)^2 + (t_1 - 1)^2} \tag{35}$$

and its first-order derivatives are

$$\begin{aligned} L_{t_0}(t_0, t_1) &= \frac{t_0+1}{\sqrt{(t_0+1)^2+1}} + \frac{t_0-t_1}{\sqrt{(t_1-t_0)^2+t_1^2}} \\ L_{t_1}(t_0, t_1) &= \frac{2t_1-t_0}{\sqrt{(t_1-t_0)^2+t_1^2}} + \frac{2t_1}{\sqrt{(t_1+1)^2+(t_1-1)^2}} \end{aligned} \tag{36}$$

The auxiliary constants in Equation (19) are $\sigma = 1$, $s_0 = 1$, $s_1 = 4$, $w_0 = 0$, and $w_1 = -2$. The coefficients of Equation (22) are $k_0 = k_1 = k_2 = k_3 = k_4 = 0$, so the equation is a tautology.

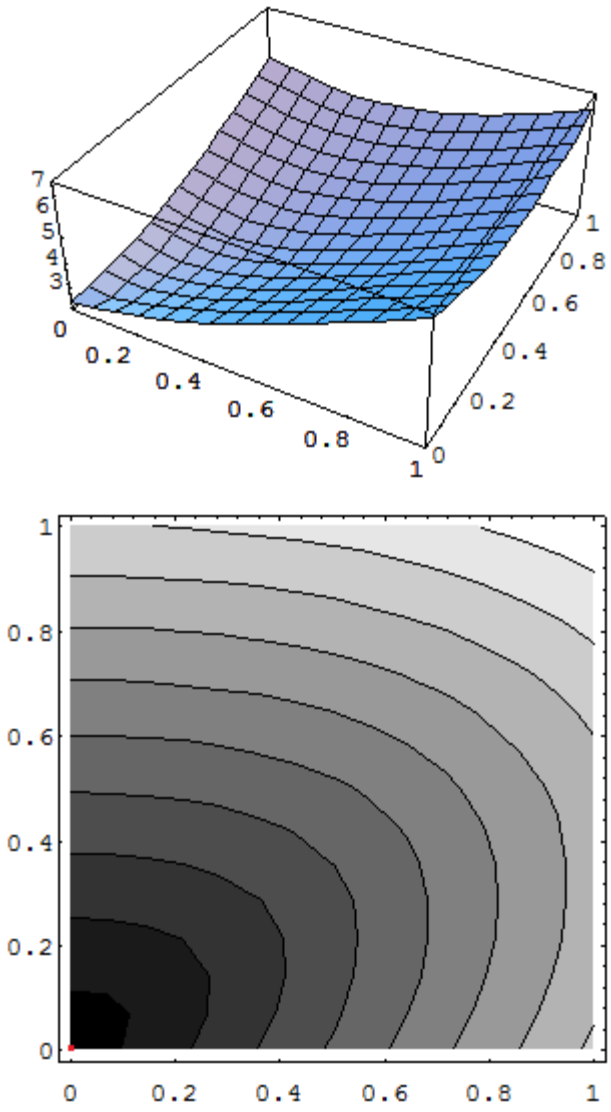
A closer analysis of the polynomials of Equation (20) shows that they factor as

$$\begin{aligned} p_0 &= t_0(1+t_1)(t_0t_1 - t_0 + 2t_1) \\ p_1 &= 2(t_0t_1 + t_0 - 2t_1)(t_0t_1 - t_0 + 2t_1) \end{aligned} \tag{37}$$

The degeneracy of Equation (21) is due to the common factor $t_0t_1 - t_0 + 2t_1$ of the two polynomials. We are interested only in roots for which $t_0 \geq 0$ and $t_1 \geq 0$. Notice that $p_0 = 0$ when $t_0 = 0$. Substituting this in the $p_1 = 0$ equation leads to $t_1 = 0$. We may ignore the p_0-root $t_1 = -1$. When the common factor is zero, namely, $t_0t_1 - t_0 + 2t_1 = 0$, there are infinitely many solutions to $p_0 = p_1 = 0$. In particular, make the change of variables $t_1 = rt_0$ for $r > 0$; then the roots are $(t_0, t_1) = ((1-2r)/r, 1-2r)$. Substituting this into the partial derivatives of Equation (36), $L_{t_0} = 0$ for $r = 1$ but $L_{t_1} \neq 0$ for any $r > 0$. That is, there is no value of r for which both partial derivatives are zero. Thus, $L(0,0)$ is already the minimum of L for $(t_0, t_1) \in [0,1]^2$.

Figure 3.10 shows a 3D plot and a contour plot of L.

Figure 3.10 Top: A 3D plot of $L(t_0, t_1)$ from Equation (30). The t_0-axis is the slanted line from the origin and goes diagonally down the page. The t_1-axis is the slanted line from the origin and goes diagonally up the page. The plot is for the domain $(t_0, t_1) \in [0, 1]^2$. Bottom: A contour plot of $L(t_0, t_1)$ on the domain $[0, 1]^2$.



The location of the global minimum is shown as a red dot, which is at the origin $(t_0, t_1) = (0, 0)$. $\bowtie$

3.4 Iterative Search for a Minimum

In the previous section, we reduced the minimization problem to computing the roots of a 4th-degree polynomial. Root finding of such polynomials can be problematic when using fixed-precision floating-point arithmetic. Moreover, imagine an n-triangle fan with origin at vertex $\mathbf{A}$ for which the first edge of the first triangle is $\langle \mathbf{P}_0, \mathbf{A} \rangle$, the last edge is $\langle \mathbf{P}_1, \mathbf{A} \rangle$, and the intermediate edges are $\langle \mathbf{B}_j, \mathbf{A} \rangle$ for $0 \le j \le n-2$. The approach of the previous section may be generalized to produce a system of polynomial equations in $n-1$ edge parameters t_i, $0 \le i \le n-2$. Elimination theory produces a single polynomial in t_0 with very large degree. Solving this is numerically ill-conditioned. It is better to avoid the conversion to polynomial equations and use instead an iterative approach to minimization.

In the three-triangle case, the parameter domain is $(t_0, t_1) \in [0,1]^2$ (a unit square). The initial value for the search may be chosen to be $(0,0)$. Based on a geometric argument, we already saw that $L(t_0, 0)$ is an increasing function for $t_0 \in [0,1]$ and $L(0, t_1)$ is an increasing function for $t_1 \in [0,1]$. It may be shown that

$$\lim_{t_0 \to 0} \frac{\partial L(t_0,0)}{\partial t_0} = \lim_{t_0 \to 0} \left(\frac{|\mathbf{D}_0|^2 t_0 - \mathbf{D}_0 \cdot \mathbf{\Delta}_0}{|t_0 \mathbf{D}_0 - \mathbf{\Delta}_0|} + \frac{|\mathbf{D}_0|^2 t_0}{|t_0 \mathbf{D}_0|} \right) = \frac{-\mathbf{D}_0 \cdot \mathbf{\Delta}_0}{|\mathbf{\Delta}_0|} + |\mathbf{D}_0| = |\mathbf{D}_0|(1 - \cos\theta_0) > 0 \quad (38)$$

where θ_0 is the angle between $\mathbf{D}_0$ and $\mathbf{\Delta}_0$. Similarly,

$$\lim_{t_1 \to 0} \frac{\partial L(0,t_1)}{\partial t_1} = |\mathbf{D}_1|(1 - \cos\theta_1) > 0 \quad (39)$$

where θ_1 is the angle between $\mathbf{D}_1$ and $\mathbf{\Delta}_1$.

Let $\mathbf{U} = (u_0, u_1)$ be a unit-length vector for which $u_0 > 0$ and $u_1 > 0$. If the first-order derivatives of L were continuous at $(0,0)$, the derivative of L at $(0,0)$ in the direction $\mathbf{U}$ would be defined by

$$\mathbf{U} \cdot \mathbf{\nabla} L(0,0) = u_0 L_{t_0}(0,0) + u_1 L_{t_1}(0,0) \quad (40)$$

where $L_{t_0}(0,0)$ and $L_{t_1}(0,0)$ are the first-order derivatives at $(0,0)$. If these derivatives are positive and the components of $\mathbf{U}$ are positive, then $\mathbf{U} \cdot \mathbf{\nabla} L(0,0)$ is positive. The conclusion would be that $L(t_0, t_1)$ increases no matter how you increase t_0 and t_1 away from the origin and into the parameter domain. The problem, though, is we already know that the first-order derivatives at $(0,0)$ are *not continuous*. We must approach the problem in a slight different manner.

3.4.1 Determining Whether a Minimum Search is Necessary

Consider $L(t_0, t_1)$ restricted to a line segment in the parameter space, say, $(t_0, t_1) = s(u_0, u_1)$ for $s \ge 0$. Define

$$F(s) = L(su_0, su_1) = |su_0 \mathbf{D}_0 - \mathbf{\Delta}_0| + s|u_0 \mathbf{D}_0 - u_1 \mathbf{D}_1| + |su_1 \mathbf{D}_1 - \mathbf{\Delta}_1| \quad (41)$$

The derivative is

$$F'(s) = \frac{u_0^2 |\mathbf{D}_0|^2 s - u_0 \mathbf{D}_0 \cdot \mathbf{\Delta}_0}{|su_0 \mathbf{D}_0 - \mathbf{\Delta}_0|} + |u_0 \mathbf{D}_0 - u_1 \mathbf{D}_1| + \frac{u_1^2 |\mathbf{D}_1|^2 s - u_1 \mathbf{D}_1 \cdot \mathbf{\Delta}_1}{|su_1 \mathbf{D}_1 - \mathbf{\Delta}_1|} \quad (42)$$

The value $F'(0)$ represents the rate of change of L at $(0,0)$ as you move into the parameter domain in the direction $\mathbf{U}$. This value is

$$F'(0) = -\frac{u_0 \mathbf{D}_0 \cdot \mathbf{\Delta}_0}{|\mathbf{\Delta}_0|} + |u_0 \mathbf{D}_0 - u_1 \mathbf{D}_1| - \frac{u_1 \mathbf{D}_1 \cdot \mathbf{\Delta}_1}{|\mathbf{\Delta}_1|} \quad (43)$$

If we can find a $\mathbf{U}$ for which $F'(0) < 0$, then L has a minimum located at some point $(t_0, t_1) \in (0, 1]^2$. Notice that this domain has removed the cases $t_0 = 0$ or $t_1 = 0$. We may search this restricted domain for the minimum point.

Now define

$$G(u_0, u_1) = F'(0) = \frac{d}{ds} L(su_0, su_1)\Big|_{s=0} \tag{44}$$

a homogeneous function because $G(\rho u_0, \rho u_1) = \rho G(u_0, u_1)$ for $\rho > 0$. Although we started out with unit-length (u_0, u_1) with positive components, it is sufficient to use the homogeneity and consider instead vectors of the form

$$(u_0, u_1) = \left(\frac{1}{|\mathbf{D}_0|}, \frac{v}{|\mathbf{D}_1|} \right) \tag{45}$$

where $v > 0$. Define $\mathbf{E}_i = \mathbf{D}_i/|\mathbf{D}_i|$ and $\boldsymbol{\xi}_i = \boldsymbol{\Delta}_i/|\boldsymbol{\Delta}_i|$ so that $\mathbf{E}_i$ and $\boldsymbol{\xi}_i$ are unit-length vectors. Define

$$H(v) = G(1/|\mathbf{D}_0|, v/|\mathbf{D}_1|) = -(\mathbf{E}_0 \cdot \boldsymbol{\xi}_0) + |\mathbf{E}_0 - v\mathbf{E}_1| - (\mathbf{E}_1 \cdot \boldsymbol{\xi}_1)v \tag{46}$$

It is easily shown that

$$H(0) = |\mathbf{E}_0| - (\mathbf{E}_0 \cdot \boldsymbol{\xi}_0) = 1 - \cos\theta_0 > 0 \tag{47}$$

where θ_0 is the angle between $\mathbf{E}_0$ and $\boldsymbol{\xi}_0$. It is also easily shown that

$$\lim_{v \to \infty} \frac{H(v)}{v} = |\mathbf{E}_1| - (\mathbf{E}_1 \cdot \boldsymbol{\xi}_1) = 1 - \cos\theta_1 > 0 \tag{48}$$

where θ_1 is the angle between $\mathbf{E}_1$ and $\boldsymbol{\xi}_1$. The question is whether there exists a $\bar{v} > 0$ for which $H(\bar{v}) < 0$. Such a $\bar{v}$ leads to a direction (u_0, u_1) for which $F'(0) < 0$.

To locate the minimum of $H(v)$ for $v > 0$, compute the first derivative and set it equal to zero,

$$H'(v) = \frac{v - \mathbf{E}_0 \cdot \mathbf{E}_1}{|\mathbf{E}_0 - v\mathbf{E}_1|} - \mathbf{E}_1 \cdot \boldsymbol{\xi}_1 = 0 \tag{49}$$

Some algebra may be used to convert this to a quadratic equation in v whose roots are

$$\bar{v} = (\mathbf{E}_0 \cdot \mathbf{E}_1) \pm (\mathbf{E}_1 \cdot \boldsymbol{\xi}_1) \sqrt{\frac{1 - (\mathbf{E}_0 \cdot \mathbf{E}_1)^2}{1 - (\mathbf{E}_1 \cdot \boldsymbol{\xi}_1)^2}} \tag{50}$$

Notice that the solution is not defined when $|\mathbf{E}_1 \cdot \boldsymbol{\xi}_1| = 1$. Geometrically, this condition implies that triangle $\langle \mathbf{B}_1, \mathbf{A}, \mathbf{P}_1 \rangle$ is degenerate; the edges $\langle \mathbf{A}, \mathbf{B} \rangle$ and $\langle \mathbf{A}, \mathbf{P}_1 \rangle$ are parallel, in the same direction or in opposite directions depending on the sign of $\mathbf{E}_1 \cdot \boldsymbol{\xi}_1$. We may assume that the input triangle mesh has no degenerate triangles, but it is possible that the mesh has needle-like triangles that could cause $|\mathbf{E}_1 \cdot \boldsymbol{\xi}_1|$ to be nearly 1, in which case a numerical implementation is ill-conditioned and must handle this properly. For example, you might avoid using the quadratic formula and instead use bisection to search for $\bar{v}$.

3.4.2 Searching for a Minimum

If $H(\bar{v}) < 0$ for a root $\bar{v} > 0$, then we have found that $F'(0) < 0$; that is, L instantaneously decreases at $(0, 0)$ in the direction associated with (u_0, u_1). We now analyze the function $F(s)$ itself, which is $L(t_0, t_1)$ restricted to the ray $s(u_0, u_1)$. It is easy to show that

$$\lim_{s \to \infty} F'(s) = u_0|\mathbf{D}_0| + |u_0\mathbf{D}_0 - u_1\mathbf{D}_1| + u_1|\mathbf{D}_1| > 0 \tag{51}$$

so for s sufficiently large, $F'(s) > 0$. The convexity of L guarantees that $F''(s) > 0$, but it is easy enough to actually compute $F''(s)$ and verify this:

$$F''(s) = \frac{u_0(a_0c_0 - b_0^2)}{|su_0\mathbf{D}_0 - \mathbf{\Delta}_0|^3} + \frac{u_1(a_1c_1 - b_1^2)}{|su_1\mathbf{D}_1 - \mathbf{\Delta}_1|^3} = \frac{u_0|\mathbf{D}_0 \times \mathbf{\Delta}_0|^2}{|su_0\mathbf{D}_0 - \mathbf{\Delta}_0|^3} + \frac{u_1|\mathbf{D}_1 \times \mathbf{\Delta}_1|^2}{|su_1\mathbf{D}_1 - \mathbf{\Delta}_1|^3} > 0 \tag{52}$$

The positivity of $F''(s)$ is guaranteed because, by assumption, we are looking for a direction (u_0, u_1) for which $u_0 > 0$ and $u_1 > 0$. The convexity guarantees that there is a value $\bar{s}$ for which $F'(\bar{s}) = 0$.

If $\bar{s}$ is large enough such that $(\bar{t}_0, \bar{t}_1) = \bar{s}(u_0, u_1)$ is outside the domain $(0, 1]^2$, compute the intersection of the ray $s(u_0, u_1)$ with the boundary of the domain, call the parameter value $\hat{s}$, and use this point for the start of the search. If $u_0 \geq u_1$, then the ray intersects the edge $t_0 = 1$, in which case $\hat{s} = 1/u_0$ and $t_1 = u_1/u_0$. If $F'(\hat{s}) > 0$, then the convexity of F ensures that $\bar{s} < \hat{s}$ and the starting point is interior to the t_0t_1-domain. If $F'(\hat{s}) \leq 0$, then start the search at $(t_0, t_1) = (1, u_1/u_0)$. Similarly, if $u_0 < u_1$, then the ray intersects the edge $t_1 = 1$, in which case $\hat{s} = 1/u_1$ and $t_0 = u_0/u_1$. If $F'(\hat{s}) > 0$, then the convexity of F ensures that $\bar{s} < \hat{s}$ and the starting point is interior to the t_0t_1-domain. If $F'(\hat{s}) \leq 0$, then start the search at $(t_0, t_1) = (u_0/u_1, 1)$.

Let the starting point be $(\bar{t}_0, \bar{t}_1)$. The search for the minimum continues by choosing a new unit-length direction $\mathbf{V} = (v_0, v_1)$ and computing the minimum of the restricted function

$$f(s) = L(\bar{t}_0 + sv_0, \bar{t}_1 + sv_1) \tag{53}$$

along the portion of the line $(\bar{t}_0 + sv_0, \bar{t}_1 + sv_1)$ that is inside the domain $(0, 1]^2$. If the minimum occurs on the boundary of the domain, then either $t_0 = 1$ or $t_1 = 1$. We already ruled out the cases $t_0 = 0$ and $t_1 = 0$, because we know that L increases along those boundary edges. Update $(\bar{t}_0, \bar{t}_1)$ to be the location of the newly found line minimum, update $\mathbf{V}$ to a new direction, redefine $f(s)$ by Equation (53), and repeat the search. The search is repeated until some convergence criterion is satisfied and we have located the minimum of $L(t_0, t_1)$ within an acceptable error tolerance.

A few strategies may be used for selecting the directions $\mathbf{V}$.

1. Coordinate-Direction Search. Alternate between $\mathbf{V} = (1, 0)$ and $\mathbf{V} = (0, 1)$.

2. Powell's Direction Set Method. Start the search with two directions $(1, 0)$ and $(0, 1)$. The next search uses an approximate conjugate direction, which is computed as the unit-length vector from the initial location to the final location after the two iterations. The process is repeated, always using the last two directions searched. No derivatives are computed, which might help to reduce the computational time.

3. Conjugate Gradient Method. If $\mathbf{V}$ is the previous direction of the search, the next direction $\mathbf{C}$ is the conjugate vector of $\mathbf{V}$ relative to the second-derivative matrix (Hessian) $H(t_0, t_1)$ of $L(t_0, t_1)$ at the current location. This vector satisfies the condition $\mathbf{C}^{\mathrm{T}}H\mathbf{U} = 0$. That is, $\mathbf{C}$ is perpendicular to $\mathbf{U}$ *with respect to the metric* H. Derivatives are computed here, so the computational time increases. The hope is that the extra costs are offset by faster convergences using a smaller number of iterations.

The goal is to obtain a fast and robust algorithm. The strategies vary in difficulty of implementation, but the more difficult ones tend to produce faster algorithms.

3.5 Numerical Implementation

Pseudocode for computing the shortest path through three triangles is shown next. The variables prefixed with an `m` are assumed to be accessible by the other functions. The ampersands in the `GetPath` function indicate that the corresponding variables are the outputs of the function.

```
void GetPath (Vector3 P0, Vector3 P1, Vector3 A, Vector3 B0, Vector3 B1,
    Vector2& tmin, Real& lmin, Vector3& M0, Vector3& M1)
{
    Vector3 mDelta0 = P0 - A;
    Vector3 mDelta1 = P1 - A;
    Vector3 mD0 = B0 - A;
    Vector3 mD1 = B1 - A;
    Real mDelta0SqrLength = Dot(mDelta0,mDelta0);
    Real mDelta1SqrLength = Dot(mDelta1,mDelta1);
    Real mD0SqrLength = Dot(mD0,mD0);
    Real mD1SqrLength = Dot(mD1,mD1);
    Real mD0DotD1 = Dot(mD0,mD1);
    Real mD0DotDelta0 = Dot(mD0,mDelta0);
    Real mD1DotDelta1 = Dot(mD1,mDelta1);

    // Compute direction mU so that mHMin = F'(0) < 0.
    Vector2 mU;
    Real mHMin;
    ComputeLargestInitialDecrease();
    if (mHMin >= 0)
    {
        // The minimum is L(0,0), so nothing to do.
        tmin = Vector2(0,0);
        lmin = sqrt(mDelta0SqrLength) + sqrt(mDelta1SqrLength);
        M0 = A; M1 = A;
        return;
    }

    // Search the ray (t0,t1) = s*(u0,u1), s >= 0, for a minimum.  The search
    // must be clamped to t0 <= 1 and t1 <= 1.
    ClampToDomain(tmin);

    // Compute the initial length and derivatives.
    lmin = L(rkTMin);
    Vector2 lder(LDer(0,tmin), LDer(1,tmin));

    // Coordinate-direction search.
    int maxIterations = <user-defined parameter>;
    Vector2 prevtmin(0,0);
    int i, j;
    for (i = 0; i < maxIterations; i++)
    {
        // Terminate when (nearly) at the global minimum.
        Real gradientEpsilon = <user-defined parameter>;
        if (Length(lder) < gradientEpsilon)
        {
            break;
        }

        // Terminate when (nearly) at the global minimum.
        Real differenceEpsilon = <user-defined parameter>;
        if (Length(tmin - prevtmin) < differenceEpsilon)
        {
            break;
        }
        prevtmin = tmin;

        for (j = 0; j < 2; j++)
        {
            if (lder[j] != 0)
            {
                Search(j,tmin,lmin,lder);
            }
```

```
        }
    }

    M0 = A + tmin[0] * mD0;
    M1 = A + tmin[1] * mD1;
}


void ComputeLargestInitialDecrease ()
{
    Real invD0Length = 1/sqrt(mD0SqrLength);
    Real invD1Length = 1/sqrt(mD1SqrLength);
    Vector3 E0 = mD0 * invD0Length;
    Vector3 E1 = mD1 * invD1Length;
    Vector3 Xi0 = mDelta0/Length(mDelta0);
    Vector3 Xi1 = mDelta1/Length(mDelta1);
    Real E0E1 = Dot(E0,E1);
    Real E0Xi0 = Dot(E0,Xi0);
    Real E1Xi1 = Dot(E1,Xi1);

    // Compute g1'(0).
    Real g1der0 = -E0E1 - E1Xi1;
    if (g1der0 >= 0)
    {
        // H cannot be negative on its domain.
        mU = Vector2(0,0);
        mHMin = 0;
        return;
    }

    // Compute the roots for the quadratic associated with g1'(r1) = 0.
    Real arg = (1 - E0E1 * E0E1)/(1 - E1Xi1 * E1Xi1);
    Real rootArg = sqrt(fArg);
    Real rootM = E0E1 - E1Xi1 * rootArg;
    Real rootP = E0E1 + E1Xi1 * rootArg;

    Real r1 = 0;
    Real absDerMin = INFINITY;
    Vector3 diff;
    Real der, absDer;
    if (rootM > 0)
    {
        diff = E0 - rootM * E1;
        der = (rootM - E0E1) - E1Xi1 * Length(diff);
        absDer = fabs(der);
        if (absDer < absDerMin)
        {
            r1 = rootM;
            absDerMin = absDer;
        }
    }
    if (rootP > 0)
    {
        diff = E0 - rootP * E1;
        der = (rootP - E0E1) - E1Xi1 * Length(diff);
        absDer = fabs(der);
        if (absDer < absDerMin)
        {
            r1 = rootP;
            absDerMin = absDer;
        }
    }

    // Compute the minimum of H = g1(r1).
    diff = E0 - r1 * E1;
    mHMin = -E0Xi0 + Length(diff) - r1 * E1Xi1;
    if (mHMin < 0)
    {
        mU[0] = invD0Length;
        mU[1] = r1 * invD1Length;
        Normalize(mU);
    }
```

```
    else
    {
        mU = Vector2(0,0);
    }
}


void ClampToDomain (Vector2& tmin)
{
    int maxIndex = 0;
    Real maxComp = mU[0];
    if (mU[1] > maxComp)
    {
        maxIndex = 1;
        maxComp = mU[1];
    }

    Real smax = 1/maxComp;
    Real der1 = FDer(smax);
    if (der1 <= 0)
    {
        for (int i = 0; i < 2; i++)
        {
            if (i != maxIndex)
            {
                tmin[i] = smax * mU[i];
            }
            else
            {
                tmin[i] = 1;
            }
        }
    }
    else
    {
        // der0 < 0 and der1 > 0, so bisect to find the root.
        Real root = GetFRoot(0,mHMin,smax,der1);
        tmin = mU * root;
    }
}


Real FDer (Real s)
{
    Vector3 tmp0 = mU[0] * mD0 - mU[1] * mD1;
    Real result = Length(tmp0);
    tmp0 = s * mU[0] * mD0 - mDelta0;
    Real tmp1 = mU[0] * (mU[0] * mD0SqrLength * s - mD0DotDelta0);
    result += tmp1/Length(tmp0);
    tmp0 = s * mU[1] * mD1 - mDelta1;
    tmp1 = mU[1] * (mU[1] * mD1SqrLength * s - mD1DotDelta1);
    result += tmp1/Length(tmp0);
    return result;
}


Real L (Vector2 t)
{
    Vector3 diff0 = mDelta0 - t[0] * mD0;
    Vector3 diffM = t[0] * mD0 - t[1] * mD1;
    Vector3 diff1 = t[1] * mD1 - mDelta1;
    return Length(diff0) + Length(diffM) + Length(diff1);
}


Real LDer (int i, Vector2 t)
{
    if (i == 0)
    {
        Vector3<Real> kDiff0 = mDelta0 - t[0] * mD0;
        Vector3<Real> kDiffM = t[0] * mD0 - t[1] * mD1;
        Real a0t0 = mD0SqrLength * t[0];
```

```
        return (a0t0 - mD0DotDelta0)/Length(diff0) + (a0t0 - mD0DotD1 * t[1])/Length(diffM);
    }
    else
    {
        Vector3 diffM = t[0] * mD0 - t[1] * mD1;
        Vector3 diff1 = t[1] * mD1 - mDelta1;
        Real a1t1 = mD1SqrLength * t[1];
        return (a1t1 - mD0DotD1 * t[0])/Length(diffM) + (a1t1 - mD1DotDelta1)/Length(diff1);
    }
}


Real GetFRoot (Real s0, Real der0, Real s1, Real der1)
{
    int numIterations = <user-defined parameter>;
    Real root = 0;
    for (int i = 0; i < numIterations; i++)
    {
        root = (s0 + s1)/2;
        Real derRoot = FDer(root);
        Real product = derRoot * der0;
        if (product < 0)
        {
            s1 = root;
            der1 = derRoot;
        }
        else if (product > 0)
        {
            s0 = root;
            der0 = derRoot;
        }
        else
        {
            break;
        }
    }
    return root;
}


void GetLRoot (int j, Vector2& t, Real s0, Real der0, Real s1, Real der1)
{
    int numIterations = <user-defined parameter>;
    for (int i = 0; i < numIterations; i++)
    {
        t[j] = (s0 + s1)/2;
        Real derRoot = LDer(j,t);
        Real product = derRoot * der0;
        if (product < 0)
        {
            s1 = t[j];
            der1 = derRoot;
        }
        else if (product > 0)
        {
            s0 = t[j];
            der0 = derRoot;
        }
        else
        {
            break;
        }
    }
}


void Search (int j, Vector2& tmin, Real& lmin, Vector2& lder)
{
    Real save, der;

    if (lder[j] > 0)
    {
```

```
            save = tmin[j];
            tmin[j] = 0;
            der = LDer(j,tmin);
            tmin[j] = save;

            if (der < (Real)0)
            {
                // Bisect [0,t[j]] to find the root.
                GetLRoot(j,tmin,0,der,tmin[j],lder[j]);
            }
            else
            {
                tmin[j] = 0;
            }
        }
        else // lder[j] < 0
        {
            save = tmin[j];
            tmin[j] = 1;
            der = LDer(j,tmin);
            tmin[j] = save;

            if (der > (Real)0)
            {
                // Bisect [t[j],1] to find the root.
                GetLRoot(j,tmin,tmin[j],lder[j],1,der);
            }
            else
            {
                tmin[j] = 1;
            }
        }

        lmin = L(tmin);
        for (int i = 0; i < 2; i++)
        {
            lder[i] = LDer(i,tmin);
        }
    }
```

4 More Than Three Triangles

The vertex $\mathbf{A}$ has 2 edges connecting it to $\mathbf{P}_0$ and $\mathbf{P}_1$. Let there be n triangles living in the region between these two edges, counterclockwise from $\langle \mathbf{A}, \mathbf{P}_0 \rangle$ to $\langle \mathbf{A}, \mathbf{P}_1 \rangle$. There are $n-1$ additional edges occurring in this region. Let the endpoints of these edges be named $\mathbf{B}_i$ for $0 \le i \le n-2$. The interior points of the edges that are used in the shortest path are

$$\mathbf{M}_i = \mathbf{A} + t_i \left(\mathbf{B}_i - \mathbf{A} \right), t_i \in [0,1] \tag{54}$$

The path length is

$$\begin{aligned} L(t_0, \ldots, t_{n-2}) &= |\mathbf{M}_0(t_0) - \mathbf{P}_0| + \textstyle\sum_{i=0}^{n-3} |\mathbf{M}_{i+1}(t_{i+1}) - \mathbf{M}_i(t_i)| + |\mathbf{P}_1 - \mathbf{M}_{n-2}(t_{n-2})| \\ &= |t_0 \mathbf{D}_0 - \mathbf{\Delta}_0| + \textstyle\sum_{i=0}^{n-3} |t_{i+1}\mathbf{D}_{i+1} - t_i \mathbf{D}_i| + |t_{n-2}\mathbf{D}_{n-2} - \mathbf{\Delta}_1| \end{aligned} \tag{55}$$

where $\mathbf{D}_i = \mathbf{B}_i - \mathbf{A}$ and $\mathbf{\Delta}_j = \mathbf{P}_j - \mathbf{A}$. Define $a_i = |\mathbf{D}_i|^2$, $b_0 = \mathbf{D}_0 \cdot \mathbf{\Delta}_0$, $b_1 = \mathbf{D}_{n-2} \cdot \mathbf{\Delta}_1$, $c_j = |\mathbf{\Delta}_j|^2$, $\ell_0 = |t_0 \mathbf{D}_0 - \mathbf{\Delta}_1|$, $\ell_1 = |t_{n-2}\mathbf{D}_{n-2} - \mathbf{\Delta}_1|$, $d_i = \mathbf{D}_i \cdot \mathbf{D}_{i+1}$, and $\lambda_i = |t_{i+1}\mathbf{D}_{i+1} - t_i \mathbf{D}_i|$.

The first-order partial derivatives are

$$\begin{aligned} L_{t_0} &= (a_0t_0 - b_0)/\ell_0 + (a_0t_0 - d_0t_1)/\lambda_0 \\ L_{t_k} &= (a_kt_k - d_{k-1}t_{k-1})/\lambda_{k-1} + (a_kt_k - d_kt_{k+1})/\lambda_k,\ \ 1 \le k \le n-3 \\ L_{t_{n-2}} &= (a_{n-2}t_{n-2} - d_{n-3}t_{n-3})/\lambda_{n-3} + (a_{n-2}t_{n-2} - b_{n-2})/\ell_1 \end{aligned} \tag{56}$$

The second-order partial derivatives are

$$\begin{aligned} L_{t_0t_0} &= (a_0c_0 - b_0^2)/\ell_0^3 + (a_0a_1 - d_0^2)t_1^2/\lambda_0^3 \\ L_{t_{k-1}t_k} &= -(a_{k-1}a_k - d_{k-1}^2)t_{k-1}t_k/\lambda_{k-1}^3,\ \ 1 \le k \le n-2 \\ L_{t_kt_k} &= (a_{k-1}a_k - d_{k-1}^2)t_{k-1}^2/\lambda_{k-1}^3 + (a_ka_{k+1} - d_k^2)t_{k+1}^2/\lambda_k^3,\ \ 1 \le k \le n-3 \\ L_{t_{n-2}t_{n-2}} &= (a_{n-2}c_1 - b_1^2)/\ell_1^3 + (a_{n-3}a_{n-2} - d_{n-3}^2)t_{n-3}^2/\lambda_{n-3}^2 \end{aligned} \tag{57}$$

Define $e_0 = (a_0c_0 - b_0^2)/\ell_0^3$, $e_1 = (a_{n-2}c_1 - b_1^2)/\ell_1^3$, and $f_k = (a_ka_{k+1} - d_k^2)/\lambda_k^3$ for $0 \le k \le n-3$; then

$$\begin{aligned} L_{t_0t_0} &= e_0 + f_0t_1^2 \\ L_{t_{k-1}t_k} &= -f_{k-1}t_{k-1}t_k,\ \ 1 \le k \le n-2 \\ L_{t_kt_k} &= f_{k-1}t_{k-1}^2 + f_kt_{k+1}^2,\ \ 1 \le k \le n-3 \\ L_{t_{n-2}t_{n-2}} &= e_1 + f_{n-3}t_{n-3}^2 \end{aligned} \tag{58}$$

4.1 Iterative Search for a Minimum

The arguments here parallel those of Section 3.4. Let $\mathbf{U} = (u_0, u_1, \ldots, u_{n-2})$ be a unit-length vector for which $u_i > 0$ for all i. Define

$$F(s) = L(su_0, \ldots, su_{n-2}) = |su_0\mathbf{D}_0 - \mathbf{\Delta}_0| + s\sum_{i=0}^{n-3}|u_i\mathbf{D}_i - u_{i+1}\mathbf{D}_{i+1}| + |su_{n-2}\mathbf{D}_{n-2} - \mathbf{\Delta}_1| \tag{59}$$

The derivative is

$$F'(s) = \frac{a_0u_0^2s - b_0u_0}{|su_0\mathbf{D}_0 - \mathbf{\Delta}_0|} + \sum_{i=0}^{n-3}|u_i\mathbf{D}_i - u_{i+1}\mathbf{D}_{i+1}| + \frac{a_{n-2}u_{n-2}^2s - b_1u_{n-2}}{|su_{n-2}\mathbf{D}_{n-2} - \mathbf{\Delta}_1|} \tag{60}$$

At $s = 0$, the derivative represents the directional derivative of L in the direction $\mathbf{U}$:

$$F'(0) = -\frac{b_0u_0}{|\mathbf{\Delta}_0|} + \sum_{i=0}^{n-3}|u_i\mathbf{D}_i - u_{i+1}\mathbf{D}_{i+1}| - \frac{b_1u_{n-2}}{|\mathbf{\Delta}_1|} \tag{61}$$

We wish to find a vector $\mathbf{U}$ for which $F'(0) < 0$, in which case we may start a search for a path of smaller length than $L(0)$. If there is no such vector, then a search is not necessary due to the convexity of L; see Section 4.2.

Define $G(\mathbf{U}) = F'(0)$. This function is homogeneous because $G(\rho\mathbf{U}) = \rho G(\mathbf{U})$. It is therefore sufficient to define $v_i = |\mathbf{D}_i|u_i$ for $1 \le i \le n-2$ and analyze

$$H(v_1, \ldots, v_{n-2}) = G\left(\frac{1}{|\mathbf{D}_0|}, \frac{v_1}{|\mathbf{D}_1|}, \ldots, \frac{v_{n-2}}{|\mathbf{D}_{n-2}|}\right) \tag{62}$$

for negativity. Specifically, we will compute the minimum for H and determine its sign.

4.1.1 The Case of Four Triangles

To illustrate the analysis, consider the case of four triangles, where

$$H(v_1, v_2) = -\mathbf{E}_0 \cdot \boldsymbol{\xi}_0 + |\mathbf{E}_0 - v_1\mathbf{E}_1| + |v_1\mathbf{E}_1 - v_2\mathbf{E}_2| - (\mathbf{E}_2 \cdot \boldsymbol{\xi}_1)v_2 \tag{63}$$

with $\mathbf{E}_i = \mathbf{D}_i/|\mathbf{D}_i|$ and $\boldsymbol{\xi}_i = \boldsymbol{\Delta}_i/|\boldsymbol{\Delta}_i|$. Define $r_1 = v_1$, $r_2 = v_2/v_1$, and

$$\begin{aligned} h_2(r_1, r_2) &= H(r_1, r_1 r_2) \\ &= (-\mathbf{E}_0 \cdot \boldsymbol{\xi}_0 + |\mathbf{E}_0 - r_1\mathbf{E}_1|) + r_1(|\mathbf{E}_1 - r_2\mathbf{E}_2| - (\mathbf{E}_2 \cdot \boldsymbol{\xi}_1)r_2) \\ &= g_1(r_1) + r_1 g_2(r_2) \end{aligned} \tag{64}$$

where

$$g_1(r_1) = -\mathbf{E}_0 \cdot \boldsymbol{\xi}_0 + |\mathbf{E}_0 - r_1\mathbf{E}_1|, \;\; g_1'(r_1) = \frac{r_1 - \mathbf{E}_0 \cdot \mathbf{E}_1}{|\mathbf{E}_0 - r_1\mathbf{E}_1|}, \;\; g_1''(r_1) = \frac{1 - (\mathbf{E}_0 \cdot \mathbf{E}_1)^2}{|\mathbf{E}_0 - r_1\mathbf{E}_1|^3} > 0 \tag{65}$$

and

$$g_2(r_2) = |\mathbf{E}_1 - r_2\mathbf{E}_2| - (\mathbf{E}_2 \cdot \boldsymbol{\xi}_1)r_2, \;\; g_2' = \frac{r_2 - \mathbf{E}_1 \cdot \mathbf{E}_2}{|\mathbf{E}_1 - r_2\mathbf{E}_2|} - \mathbf{E}_2 \cdot \boldsymbol{\xi}_1, \;\; g_2'' = \frac{1 - (\mathbf{E}_1 \cdot \mathbf{E}_2)^2}{|\mathbf{E}_1 - r_2\mathbf{E}_2|^3} > 0 \tag{66}$$

We wish to find numbers $r_1 > 0$ and $r_2 > 0$ for which $h_2(r_1, r_2) < 0$. The positivity of the second derivatives implies that g_1 and g_2 are convex. Zero-valued second derivatives are ruled out by the assumption that the triangles are nondegenerate.

The sequence of steps is long, so I have broken this down into small pieces to make it more understandable. The algorithm is *recursive in dimension*, which means that the n-parameter problem is reduced to one of similar structure for $n-1$ parameters.

Guarantee that $h_2(r_1, 0) > 0$. Consider the restricted function $h_2(r_1, 0) = g_1(r_1) + r_1$. The r_1-derivatives are $h_{2,r_1}(r_1, 0) = g_1'(r_1) + 1$ and $h_{2,r_1r_1}(r_1, 0) = g_1''(r_1) > 0$, in which case $h_2(\cdot, 0)$ is a convex function. Also, $h_2(0, 0) = 1 - \mathbf{E}_0 \cdot \boldsymbol{\xi}_0 > 0$ and $h_{2,r_1}(0, 0) = 1 - \mathbf{E}_0 \cdot \mathbf{E}_1 > 0$. The convexity of $h_2(\cdot, 0)$ guarantees that $h_2(r_1, 0) > 0$ for $r_1 \ge 0$.

Necessity of $g_2'(0) < 0$. The function $g_2(r_2)$ is convex with $g_2(0) = 1$ and $g_2'(0) = -\mathbf{E}_1 \cdot \mathbf{E}_2 - \mathbf{E}_2 \cdot \boldsymbol{\xi}_1$. If $g_2'(0) \ge 0$, then the convexity of g_2 forces $g_2'(r_2) \ge 0$ for $r_2 \ge 0$. This implies $h_{2,r_2}(r_1, r_2) = r_1 g_2'(r_2) \ge 0$ and h_2 increases in the r_2-direction no matter the choice of r_1; that is, $h_2(r_1, r_2) > 0$ for $r_1 \ge 0$ and $r_2 \ge 0$. The only chance for $h_2(r_1, r_2)$ to be negative is if $g_2'(r_2)$ attains negative values. The convexity of g_2 makes it sufficient that $g_2'(0) < 0$.

Restricting the search to a specific $\bar{r}_2$. We now know that $g_2'(0) < 0$. In the limit,

$$g_2'(\infty) = \lim_{r_2 \to \infty} g_2'(r_2) = 1 - \mathbf{E}_2 \cdot \boldsymbol{\xi}_1 > 0 \tag{67}$$

The convexity of g_2 guarantees that there is a unique finite number $\bar{r}_2 > 0$ for which $g_2'(\bar{r}_2) = 0$. This equation may be converted to a quadratic equation whose roots are

$$r_2 = (\mathbf{E}_1 \cdot \mathbf{E}_2) \pm (\mathbf{E}_2 \cdot \boldsymbol{\xi}_1)\sqrt{\frac{1-(\mathbf{E}_1 \cdot \mathbf{E}_2)^2}{1-(\mathbf{E}_2 \cdot \boldsymbol{\xi}_1)^2}} \tag{68}$$

Let $\bar{r}_2$ be the one of these two numbers for which $\bar{r}_2 > 0$ and $g_2'(\bar{r}_2) = 0$. The minimum of $h_2(r_1, r_2)$ in the r_2-direction must be $h_2(r_1, \bar{r}_2)$ since $h_{2,r_2}(r_1, \bar{r}_2) = r_1 g_2'(\bar{r}_2) = 0$. Define $h_1(r_1) = h_2(r_1, \bar{r}_2)$ and $\hat{g}_1(r_1) = h_1(r_1)$. The last definition is trivial, but sets up the notation for the case of more triangles.

Guarantee that $g_2(\bar{r}_2) > -1$. We know that $r_2 > 0$ and $\mathbf{E}_2 \cdot \boldsymbol{\xi}_1 < 1$, so

$$g_2(r_2) > |\mathbf{E}_1 - r_2\mathbf{E}_2| - r_2 = \ell_2(r_2) \tag{69}$$

where the last equality defines the lower-bound function $\ell_2(r_2)$. This function is convex with $\ell_2(0) = 1$ and $\ell_2'(0) < 0$. Observe that

$$\begin{aligned}
\lim_{r_2\to\infty} \ell_2(r_2) &= \lim_{r_2\to\infty} (|\mathbf{E}_1 - r_2\mathbf{E}_2| - r_2) \\
&= \lim_{r_2\to\infty} \left(\tfrac{(|\mathbf{E}_1 - r_2\mathbf{E}_2| - r_2)(|\mathbf{E}_1 - r_2\mathbf{E}_2| + r_2)}{|\mathbf{E}_1 - r_2\mathbf{E}_2| + r_2} \right) \\
&= \lim_{r_2\to\infty} \left(\tfrac{|\mathbf{E}_1 - r_2\mathbf{E}_2|^2 - r_2^2}{|\mathbf{E}_1 - r_2\mathbf{E}_2| + r_2} \right) \\
&= \lim_{r_2\to\infty} \left(\tfrac{|\mathbf{E}_1 - r_2\mathbf{E}_2|^2 - r_2^2}{|\mathbf{E}_1 - r_2\mathbf{E}_2| + r_2} \right) \\
&= \lim_{r_2\to\infty} \left(\tfrac{1 - 2(\mathbf{E}_1 \cdot \mathbf{E}_2) r_2}{|\mathbf{E}_1 - r_2\mathbf{E}_2| + r_2} \right) \\
&= -\mathbf{E}_1 \cdot \mathbf{E}_2 \\
&> -1
\end{aligned} \tag{70}$$

Therefore, $g(\bar{r}_2) > -1$ is guaranteed.

Guarantee that $h_1(0) > 0$. This is trivial to verify, $h_1(0) = h_2(0, \bar{r}_2) = g_1(0) = 1 - \mathbf{E}_0 \cdot \boldsymbol{\xi}_0 > 0$.

Necessity of $\hat{g}_1'(0) < 0$. The function $\hat{g}_1(r_1)$ is convex with $\hat{g}_1(0) = 1 - \mathbf{E}_0 \cdot \boldsymbol{\xi}_0 > 0$ and $\hat{g}_1'(0) = -\mathbf{E}_0 \cdot \mathbf{E}_1 + g_2(\bar{r}_2)$. If $\hat{g}_1'(0) \geq 0$, then the convexity of $\hat{g}_1$ forces $\hat{g}_1(r_1) > 0$ for $r_1 \geq 0$. This implies $h_1'(r_1) = \hat{g}_1'(r_1) \geq 0$ and h_1 increases in the r_1-direction; that is, $h_1(r_1) > 0$ for $r_1 \geq 0$. The only chance for $h_1(r_1)$ to be negative is if $\hat{g}_1'(r_1)$ attains negative values. The convexity of $\hat{g}_1$ makes it sufficient that $\hat{g}_1'(0) < 0$.

Restricting the search to a specific $\bar{r}_1$. We now know that $\hat{g}_1'(0) < 0$. In the limit,

$$\hat{g}_1'(\infty) = \lim_{r_1\to\infty} \hat{g}_1'(r_1) = 1 + g_2(\bar{r}_2) > 0 \tag{71}$$

The convexity of $\hat{g}_1$ guarantees that there is a unique finite number $\bar{r}_1 > 0$ for which $\hat{g}_1'(\bar{r}_1) = 0$. This equation may be converted to a quadratic equation whose roots are

$$r_1 = (\mathbf{E}_0 \cdot \mathbf{E}_1) \pm g_2(\bar{r}_2)\sqrt{\frac{1-(\mathbf{E}_0 \cdot \mathbf{E}_1)^2}{1-g_2(\bar{r}_2)^2}} \tag{72}$$

Let $\bar{r}_1$ be the one of these two numbers for which $\bar{r}_1 > 0$ and $\hat{g}_1'(\bar{r}_1) = 0$. The minimum of $h_1(r_1)$ must be $h_1(\bar{r}_1)$ since $h_1'(\bar{r}_1) = \hat{g}_1'(\bar{r}_1) = 0$. Thus, the minimum of $h_2(r_1, r_2)$ is the value $h_2(\bar{r}_1, \bar{r}_2)$.

4.1.2 The Case of Five Triangles

To show that the ideas extend to more triangles, consider the case of five triangles, where

$$H(v_1,v_2,v_3) = -\mathbf{E}_0\cdot\boldsymbol{\xi}_0 + |\mathbf{E}_0 - v_1\mathbf{E}_1| + |v_1\mathbf{E}_1 - v_2\mathbf{E}_2| + |v_2\mathbf{E}_2 - v_3\mathbf{E}_3| - (\mathbf{E}_3\cdot\boldsymbol{\xi}_1)v_3 \tag{73}$$

Define $r_1 = v_1$, $r_2 = v_2/v_1$, $r_3 = v_3/v_2$, and

$$\begin{aligned} h_3(r_1,r_2,r_3) &= H(r_1, r_1r_2, r_1r_2r_3) \\ &= (-\mathbf{E}_0\cdot\boldsymbol{\xi}_0 + |\mathbf{E}_0 - r_1\mathbf{E}_1|) + r_1\left(|\mathbf{E}_1 - r_2\mathbf{E}_2| + r_2\left(|\mathbf{E}_2 - r_3\mathbf{E}_3| - (\mathbf{E}_3\cdot\boldsymbol{\xi}_1)r_3\right)\right) \\ &= g_1(r_1) + r_1(g_2(r_2) + r_2g_3(r_3)) \end{aligned} \tag{74}$$

where

$$g_1(r_1) = -\mathbf{E}_0\cdot\boldsymbol{\xi}_0 + |\mathbf{E}_0 - r_1\mathbf{E}_1|,\ \ g_1'(r_1) = \frac{r_1 - \mathbf{E}_0\cdot\mathbf{E}_1}{|\mathbf{E}_0 - r_1\mathbf{E}_1|},\ \ g_1''(r_1) = \frac{1-(\mathbf{E}_0\cdot\mathbf{E}_1)^2}{|\mathbf{E}_0 - r_1\mathbf{E}_1|^3} > 0 \tag{75}$$

and

$$g_2(r_2) = |\mathbf{E}_1 - r_2\mathbf{E}_2|,\ \ g_2'(r_2) = \frac{r_2 - \mathbf{E}_1\cdot\mathbf{E}_2}{|\mathbf{E}_1 - r_2\mathbf{E}_2|},\ \ g_2''(r_2) = \frac{1-(\mathbf{E}_1\cdot\mathbf{E}_2)^2}{|\mathbf{E}_1 - r_2\mathbf{E}_2|^3} > 0 \tag{76}$$

and

$$g_3(r_3) = |\mathbf{E}_2 - r_3\mathbf{E}_3| - (\mathbf{E}_3\cdot\boldsymbol{\xi}_1)r_3,\ \ g_3'(r_3) = \frac{r_3 - \mathbf{E}_2\cdot\mathbf{E}_3}{|\mathbf{E}_2 - r_3\mathbf{E}_3|} - \mathbf{E}_3\cdot\boldsymbol{\xi}_1,\ \ g_3''(r_3) = \frac{1-(\mathbf{E}_2\cdot\mathbf{E}_3)^2}{|\mathbf{E}_2 - r_3\mathbf{E}_3|^3} > 0 \tag{77}$$

We wish to find numbers $r_1 > 0$, $r_2 > 0$, and $r_3 > 0$ for which $h_3(r_1,r_2,r_3) < 0$. The positivity of the second derivatives implies that all g_1, g_2, and g_3 are convex. Zero-valued second derivatives are ruled out by the assumption that the triangles are nondegenerate.

As in the case of four triangles, the algorithm is recursive in dimension.

Guarantee that $h_3(r_1,r_2,0) > 0$. The restricted function $h_3(r_1,0,0)$ is the same as the function $h_2(r_1,0)$ in the four-triangle case, and we found that it was positive. Thus, $h_3(r_1,0,0) > 0$ for $r_1 \ge 0$.

Consider the restricted function $h_3(r_1,r_2,0) = g_1(r_1) + r_1(g_2(r_2) + r_2)$. The r_2 derivatives are

$$h_{3,r_2}(r_1,r_2,0) = r_1(g_2'(r_2) + 1) = r_1\left(\frac{r_2 - \mathbf{E}_1\cdot\mathbf{E}_2}{|\mathbf{E}_1 - r_2\mathbf{E}_2|} + 1\right) \tag{78}$$

and

$$h_{3,r_2r_2} = r_1g_2''(r_2) = r_1\left(\frac{1-(\mathbf{E}_1\cdot\mathbf{E}_2)^2}{|\mathbf{E}_1 - r_2\mathbf{E}_2|^3}\right) > 0 \tag{79}$$

in which case $h_3(r_1,\cdot,0)$ is a convex function (in its r_2 variable). Also, $h_3(r_1,0,0) > 0$ and $h_{3,r_2}(r_1,0,0) = r_1(1 - \mathbf{E}_1\cdot\mathbf{E}_2) > 0$. The convexity of $h_3(r_1,\cdot,0)$ guarantees that $h_3(r_1,r_2,0) > 0$ for $r_1 \ge 0$ and $r_2 \ge 0$.

Necessity of $g_3'(0) < 0$. The function $g_3(r_3)$ is convex with $g_3(0) = 1$ and $g_3'(0) = -\mathbf{E}_2\cdot\mathbf{E}_3 - \mathbf{E}_3\cdot\boldsymbol{\xi}_1$. If $g_3'(0) \ge 0$, then the convexity of g_3 forces $g_3'(r_3) \ge 0$ for $r_3 \ge 0$. This implies $h_{3,r_3}(r_1,r_2,r_3) = r_1r_2g_3'(r_3) \ge 0$ and h_3 increases in the r_3-direction no matter the choices of r_1 and r_2; that is, $h_3(r_1,r_2,r_3) > 0$ for $r_1 \ge 0$ and $r_2 \ge 0$. The only chance for $h_3(r_1,r_2,r_3)$ to be negative is if $g_3'(r_3)$ attains negative values. The convexity of g_3 makes it sufficient that $g_3'(0) < 0$.

Restricting the search to a specific $\bar{r}_3$. We now know that $g_3'(0) < 0$. In the limit, notice that

$$g_3'(\infty) = \lim_{r_3 \to \infty} g_3'(r_3) = 1 - \mathbf{E}_3 \cdot \boldsymbol{\xi}_1 > 0 \tag{80}$$

The convexity of g_3, guarantees that there is a unique finite number $\bar{r}_3 > 0$ for which $g_3'(\bar{r}_3) = 0$. This equation may be converted to a quadratic equation whose roots are

$$r_3 = (\mathbf{E}_2 \cdot \mathbf{E}_3) \pm (\mathbf{E}_3 \cdot \boldsymbol{\xi}_1)\sqrt{\frac{1 - (\mathbf{E}_2 \cdot \mathbf{E}_3)^2}{1 - (\mathbf{E}_3 \cdot \boldsymbol{\xi}_1)^2}} \tag{81}$$

Let $\bar{r}_3$ be the one of these two numbers for which $\bar{r}_3 > 0$ and $g_3'(\bar{r}_3) = 0$. The minimum of $h(r_1, r_2, r_3)$ in the r_3-direction must be $h(r_1, r_2, \bar{r}_3)$ since $h_{r_3}(r_1, r_2, \bar{r}_3) = r_1 r_2 g_3'(\bar{r}_3) = 0$. Define $\hat{g}_2(r_2) = g_2(r_2) + r_2 g_3(\bar{r}_3)$ and $h_2(r_1, r_2) = h_3(r_1, r_2, \bar{r}_3) = g_1(r_1) + r_1 \hat{g}_2(r_2)$.

Guarantee that $g_3(\bar{r}_3) > -1$. We know that $r_3 > 0$ and $\mathbf{E}_3 \cdot \boldsymbol{\xi}_1 < 1$, so

$$g_3(r_3) > |\mathbf{E}_2 - r_3 \mathbf{E}_3| - r_3 = \ell_3(r_3) \tag{82}$$

where the last equality defines the lower-bound function $\ell_3(r_3)$. This function is convex with $\ell_3(0) = 1$ and $\ell_3'(0) < 0$. A construction similar to that of Equation (70) shows that $\ell_3(r_3) > -1$, in which case $g_3(\bar{r}_3) > \ell_3(\bar{r}_3) > -1$.

Guarantee that $h_2(r_1, 0) > 0$. Consider the restricted function $h_2(r_1, 0) = g_1(r_1) + r_1 \hat{g}_2(0) = g_1(r_1) + r_1$. As in the four-triangle case, $g_1(r_1) + r_1$ is positive, so $h_2(r_1, 0) > 0$ for $r_1 \geq 0$.

Necessity of $\hat{g}_2'(0) < 0$. The function $\hat{g}_2(r_2)$ is convex with $\hat{g}_2(0) = 1$ and $\hat{g}_2'(0) = -\mathbf{E}_1 \cdot \mathbf{E}_2 + g_3(\bar{r}_3)$. If $\hat{g}_2'(0) \geq 0$, then the convexity of $\hat{g}_2$ forces $\hat{g}_2'(r_2) \geq 0$ for $r_2 \geq 0$. This implies $h_{2,r_2}(r_1, r_2) = r_1 \hat{g}_2(r_2) \geq 0$ and h_2 increases in the r_2-direction no matter the choice of r_1; that is, $h_2(r_1, r_2) > 0$ for $r_1 \geq 0$ and $r_2 \geq 0$. The only chance for $h_2(r_1, r_2)$ to be negative is if $\hat{g}_2'(r_2)$ attains negative values. The convexity of $\hat{g}_2$ makes it sufficient that $\hat{g}_2'(0) < 0$.

Restricting the search to a specific $\bar{r}_2$. We now know that $\hat{g}_2'(0) < 0$. In the limit,

$$\hat{g}_2'(\infty) = \lim_{r_2 \to \infty} \hat{g}_2'(r_2) = 1 + g_3(\bar{r}_3) > 0 \tag{83}$$

The convexity of $\hat{g}_2$ guarantees that there is a unique finite number $\bar{r}_2 > 0$ for which $\hat{g}_2'(\bar{r}_2) = 0$. This equation may be converted to a quadratic equation whose roots are

$$r_2 = (\mathbf{E}_1 \cdot \mathbf{E}_2) \pm g_3(\bar{r}_3)\sqrt{\frac{1 - (\mathbf{E}_1 \cdot \mathbf{E}_2)^2}{1 - g_3(\bar{r}_3)^2}} \tag{84}$$

The facts that $g_3(0) = 1$, $g_3'(0) < 0$, and g_3 is convex imply $g_3(\bar{r}_3) < 1$. We already saw that $g_3(\bar{r}_3) > -1$. Thus, the denominator inside the square root of the equations for r_2 is not zero. Let $\bar{r}_2$ be the one of these two numbers for which $\bar{r}_2 > 0$ and $\hat{g}_2'(\bar{r}_2) = 0$. The minimum of $h_2(r_1, r_2)$ in the r_2-direction must be $h_2(r_1, \bar{r}_2)$ since $h_{2,r_2}(r_1, \bar{r}_2) = r_1 \hat{g}_2'(\bar{r}_2) = 0$. Define $h_1(r_1) = h_2(r_1, \bar{r}_2)$ and $\hat{g}_1(r_1) = h_1(r_1)$.

Guarantee that $\hat{g}_2(\bar{r}_2) > -1$. We know that $r_2 > 0$ and $g_3(\bar{r}_3) > -1$, so

$$\hat{g}_2(r_2) = |\mathbf{E}_1 - r_2 \mathbf{E}_2| + g_3(\bar{r}_3) r_2 > |\mathbf{E}_1 - r_2 \mathbf{E}_2| - r_2 = \ell_2(r_2) > -1 \tag{85}$$

The function $\ell_2(r_2)$ is the same one we analyzed in the four-triangle case. Thus, $\hat{g}_2(\bar{r}_2) > -1$.

Guarantee that $h_1(0) > 0$. This is trivial to verify, $h_1(0) = 1 - \mathbf{E}_0 \cdot \boldsymbol{\xi}_0 > 0$.

Necessity of $\hat{g}_1'(0) < 0$. The function $\hat{g}_1(r_1)$ is convex with $\hat{g}_1(0) = 1 - \mathbf{E}_0 \cdot \boldsymbol{\xi}_0 > 0$ and $\hat{g}_1'(0) = -\mathbf{E}_0 \cdot \mathbf{E}_1 + \hat{g}_2(\bar{r}_2)$. If $\hat{g}_1'(0) \geq 0$, then the convexity of $\hat{g}_1$ forces $\hat{g}_1(r_1) > 0$ for $r_1 \geq 0$. This implies $h_1'(r_1) = \hat{g}_1'(r_1) \geq 0$ and h_1 increases in the r_1-direction; that is, $h_1(r_1) > 0$ for $r_1 \geq 0$. The only chance for $h_1(r_1)$ to be negative is if $\hat{g}_1'(r_1)$ attains negative values. The convexity of $\hat{g}_1$ makes it sufficient that $\hat{g}_1'(0) < 0$.

Restricting the search to a specific $\bar{r}_1$. We now know that $\hat{g}_1'(0) < 0$. In the limit,

$$\hat{g}_1'(\infty) = \lim_{r_1 \to \infty} \hat{g}_1'(r_1) = 1 + \hat{g}_2(\bar{r}_2) > 0 \tag{86}$$

The convexity of $\hat{g}_1$ guarantees that there is a unique finite number $\bar{r}_1 > 0$ for which $\hat{g}_1'(\bar{r}_1) = 0$. This equation may be converted to a quadratic equation whose roots are

$$r_1 = (\mathbf{E}_0 \cdot \mathbf{E}_1) \pm \hat{g}_2(\bar{r}_2) \sqrt{\frac{1 - (\mathbf{E}_0 \cdot \mathbf{E}_1)^2}{1 - \hat{g}_2(\bar{r}_2)^2}} \tag{87}$$

The facts that $\hat{g}_2(0) = 1$, $\hat{g}_2'(0) < 0$, and $\hat{g}_2$ is convex imply $\hat{g}_2(\bar{r}_2) < 1$. We already saw that $\hat{g}_2(\bar{r}_2) > -1$. Thus, the denominator inside the square root of the equations for r_1 is not zero. Let $\bar{r}_1$ be the one of these two numbers for which $\bar{r}_1 > 0$ and $\hat{g}_1'(\bar{r}_1) = 0$. The minimum of $h_1(r_1)$ must be $h_1(\bar{1}_2)$ since $h_1'(\bar{r}_1) = \hat{g}_1'(\bar{r}_1) = 0$. Thus, the minimum of $h_2(r_1, r_2)$ is the value $h_2(\bar{r}_1, \bar{r}_2)$ and the minimum of $h_3(r_1, r_2, r_3)$ is $h_3(\bar{r}_1, \bar{r}_2, \bar{r}_3)$.

4.2 Analysis of Convexity

The Hessian matrix of $L(t_0, \ldots, t_{n-2})$ is symmetric and tridiagonal. This section contains a proof that it is positive definite.

For unified notation, define $f_{-1} = e_0$, $t_{-1} = 1$, $f_{n-2} = e_1$, and $t_{n-1} = 1$. Every principal submatrix of the Hessian matrix is of the form

$$S_{i,j} = \begin{bmatrix} g_i & -f_{i+1}t_{i+1}t_{i+2} & 0 & 0 & \cdots & 0 & 0 & 0 \\ -f_{i+1}t_{i+1}t_{i+2} & g_{i+1} & -f_{i+2}t_{i+2}t_{i+3} & 0 & \cdots & 0 & 0 & 0 \\ 0 & -f_{i+2}t_{i+2}t_{i+3} & g_{i+2} & -f_{i+3}t_{i+3}t_{i+4} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -f_{j-1}t_{j-1}t_j & g_{j-1} & -f_jt_jt_{j+1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & -f_jt_jt_{j+1} & g_j \end{bmatrix} \tag{88}$$

where $g_k = f_k t_k^2 + f_{k+1} t_{k+2}^2$ for $i \leq k \leq j$ and where $-1 \leq i \leq j \leq n-3$.

To compute $\det S_{i,j}$, use a cofactor expansion by the first row of the matrix to obtain

$$\det S_{i,j} = \left(f_i t_i^2 + f_{i+1} t_{i+2}^2\right) \det S_{i+1,j} + \det A_{i+1,j} \tag{89}$$

where $A_{i+1,j}$ is the submatrix of $S_{i,j}$ obtained by deleting the first row and second column. Furthermore,

$$\det S_{i,j} = f_i t_i^2 \det S_{i+1,j} + f_{i+1} t_{i+2}^2 \det S_{i+1,j} + \det A_{i+1,j} = f_i t_i^2 \det S_{i+1,j} + \det B_{i,j} \tag{90}$$

where

$$B_{i,j} = \begin{bmatrix} f_{i+1}t_{i+2}^2 & -f_{i+1}t_{i+1}t_{i+2} & 0 & 0 & \cdots & 0 & 0 & 0 \\ -f_{i+1}t_{i+1}t_{i+2} & g_{i+1} & -f_{i+2}t_{i+2}t_{i+3} & 0 & \cdots & 0 & 0 & 0 \\ 0 & -f_{i+2}t_{i+2}t_{i+3} & g_{i+2} & -f_{i+3}t_{i+3}t_{i+4} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -f_{j-1}t_{j-1}t_j & g_{j-1} & -f_jt_jt_{j+1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & -f_jt_jt_{j+1} & g_j \end{bmatrix} \tag{91}$$

The matrix is row reduced, each reduction preserving the determinant of the original matrix. Assuming the rows are named R_i through R_j, the first reduction is

$$\frac{t_{i+1}}{t_{i+2}} R_i + R_{i+1} \to R_{i+1} \tag{92}$$

which leads to

$$\left[\begin{array}{c|cccccccc} f_{i+1}t_{i+2}^2 & -f_{i+1}t_{i+1}t_{i+2} & 0 & 0 & \cdots & 0 & 0 & 0 \\ \hline 0 & f_{i+2}t_{i+3}^2 & -f_{i+2}t_{i+2}t_{i+3} & 0 & \cdots & 0 & 0 & 0 \\ 0 & -f_{i+2}t_{i+2}t_{i+3} & g_{i+2} & -f_{i+3}t_{i+3}t_{i+4} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -f_{j-1}t_{j-1}t_j & g_{j-1} & -f_j t_j t_{j+1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & -f_j t_j t_{j+1} & g_j \end{array}\right] \tag{93}$$

The lower right block is a matrix of the same structure as what we started with. Continuing with similar row reductions, we eventually arrive at

$$\det B_{i,j} = \begin{bmatrix} f_{i+1}t_{i+2}^2 & -f_{i+1}t_{i+1}t_{i+2} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & f_{i+2}t_{i+3}^2 & -f_{i+2}t_{i+2}t_{i+3} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & f_{i+3}t_{i+4}^2 & -f_{i+3}t_{i+3}t_{i+4} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & f_j t_{j+1}^2 & -f_j t_j t_{j+1} & \\ 0 & 0 & 0 & 0 & \cdots & 0 & f_{j+1}t_{j+2}^2 & \end{bmatrix} = \prod_{k=i+1}^{j+1} f_k t_{k+1}^2 \tag{94}$$

The determinants of the principal submatrices are related by the linear recurrence equation,

$$\det S_{i,j} = f_i t_i^2 \det S_{i+1,j} + \prod_{k=i+1}^{j+1} f_k t_{k+1}^2 \tag{95}$$

an equation that is valid for $i + 1 \le j$. For a fixed j, the initial value for the recurrence is

$$\det S_{j,j} = f_j t_j^2 + f_{j+1} t_{j+2}^2 > 0 \tag{96}$$

The next term is

$$\det S_{j-1,j} = f_{j-1} t_{j-1}^2 \det S_{j,j} + f_j t_{j+1}^2 f_{j+1} t_{j+2}^2 > 0 \tag{97}$$

Recursively, $\det S_{i,j} > 0$ for all $-1 \le i \le j \le n-3$, so the principal submatrices all have positive determinants and the Hessian matrix is positive definite.

4.3 The Four-Triangle Numerical Implementation

Pseudocode for computing the shortest path through four triangles is shown next. The variables prefixed with an `m` are assumed to be accessible by the other functions. The ampersands in the function definitions indicate that the corresponding variables are the outputs of the function.

```
void GetPath (Vector3 P0, Vector3 P1, Vector3 A, Vector3 B0, Vector3 B1, Vector3 B2,
    Vector3& tmin, Real& lmin, Vector3& M0, Vector3& M1, Vector3& M2)
{
    Vector3 mDelta0 = P0 - A;
    Vector3 mDelta1 = P1 - A;
```

```
    Vector3 mD0 = B0 - A;
    Vector3 mD1 = B1 - A;
    Vector3 mD2 = B2 - A;
    Real mDelta0SqrLength = Dot(mDelta0,mDelta0);
    Real mDelta1SqrLength = Dot(mDelta1,mDelta1);
    Real mD0SqrLength = Dot(mD0,mD0);
    Real mD1SqrLength = Dot(mD1,mD1);
    Real mD2SqrLength = Dot(mD2,mD2);
    Real mD0DotD1 = Dot(mD0,mD1);
    Real mD1DotD2 = Dot(mD1,mD2);
    Real mD0DotDelta0 = Dot(mD0,mDelta0);
    Real mD2DotDelta1 = Dot(mD2,mDelta1);

    // Compute direction mU so that mHMin = F'(0) < 0.
    Vector3 mU;
    Real mHMin;
    ComputeLargestInitialDecrease();
    if (mHMin >= 0)
    {
        // The minimum is L(0,0,0), so nothing to do.
        tmin = Vector3(0,0,0);
        lmin = sqrt(mDelta0SqrLength) + sqrt(mDelta1SqrLength);
        M0 = A;
        M1 = A;
        M2 = A;
        return;
    }

    // Search the ray (t0,t1,t2) = s*(u0,u1,u2), s >= 0, for a minimum.  The search
    // must be clamped to t0 <= 1, t1 <= 1, and t2 <= 1.
    ClampToDomain(tmin);

    // Compute the initial length and derivatives.
    lmin = L(rkTMin);
    Vector3 lder(LDer(0,tmin), LDer(1,tmin), LDer(2,tmin));

    // Coordinate-direction search.
    int maxIterations = <user-defined parameter>;
    Vector3 prevtmin(0,0,0);
    int i, j;
    for (i = 0; i < maxIterations; i++)
    {
        // Terminate when (nearly) at the global minimum.
        Real gradientEpsilon = <user-defined parameter>;
        if (Length(lder) < gradientEpsilon)
        {
            break;
        }

        // Terminate when (nearly) at the global minimum.
        Real differenceEpsilon = <user-defined parameter>;
        if (Length(tmin - prevtmin) < differenceEpsilon)
        {
            break;
        }
        prevtmin = tmin;

        for (j = 0; j < 3; j++)
        {
            if (lder[j] != 0)
            {
                Search(j,tmin,lmin,lder);
            }
        }
    }

    M0 = A + tmin[0] * mD0;
    M1 = A + tmin[1] * mD1;
    M2 = A + tmin[2] * mD2;
}
```

```
void ComputeLargestInitialDecrease ()
{
    Real invD0Length = 1/sqrt(mD0SqrLength);
    Real invD1Length = 1/sqrt(mD1SqrLength);
    Real invD2Length = 1/sqrt(mD2SqrLength);
    Vector3 E0 = mD0 * invD0Length;
    Vector3 E1 = mD1 * invD1Length;
    Vector3 E2 = mD2 * invD2Length;
    Vector3 Xi0 = mDelta0/Length(mDelta0);
    Vector3 Xi1 = mDelta1/Length(mDelta1);
    Real E0E1 = Dot(E0,E1);
    Real E1E2 = Dot(E1,E2);
    Real E0Xi0 = Dot(E0,Xi0);
    Real E2Xi1 = Dot(E2,Xi1);

    // Compute g2'(0).
    Real g2der0 = -E1E2 - E2Xi1;
    if (g2der0 >= 0)
    {
        // H cannot be negative on its domain.
        mU = Vector3(0,0,0);
        mHMin = 0;
        return;
    }

    // Compute the roots for the quadratic associated with g2'(r2) = 0.
    Real arg = (1 - E1E2 * E1E2)/(1 - E2Xi1 * E2Xi1);
    Real rootArg = sqrt(arg);
    Real rootM = E1E2 - E2Xi1 * rootArg;
    Real rootP = E1E2 + E2Xi1 * rootArg;

    Real r2 = 0;
    Real absDerMin = INFINITY;
    Vector3 kDiff;
    Real der, absDer;
    if (rootM > 0)
    {
        diff = E1 - rootM * E2;
        der = (rootM - E1E2) - E2Xi1 * Length(diff);
        absDer = fabs(der);
        if (absDer < absDerMin)
        {
            r2 = rootM;
            absDerMin = absDer;
        }
    }
    if (rootP > 0)
    {
        diff = E1 - rootP * E2;
        der = (rootP - E1E2) - E2Xi1 * Length(diff);
        absDer = fabs(der);
        if (absDer < absDerMin)
        {
            r2 = rootP;
            absDerMin = absDer;
        }
    }

    // Compute g2(r2).
    diff = E1 - r2 * E2;
    Real g2r2 = Length(diff) - E2Xi1 * r2;

    // Compute hat[g1]'(0).
    Real g1der0 = -E0E1 + g2r2;
    if (g1der0 >= 0)
    {
        // H cannot be negative on its domain.
        mU = Vector3(0,0,0);
        mHMin = 0;
        return;
    }
```

```
    // Compute the roots for the quadratic associated with hat[g1]'(r1) = 0.
    arg = (1 - E0E1 * E0E1)/(1 - g2r2 * g2r2);
    rootArg = sqrt(arg);
    rootM = E0E1 - g2r2 * rootArg;
    rootP = E0E1 + g2r2 * rootArg;

    Real r1 = 0;
    absDerMin = INFINITY;
    if (rootM > 0)
    {
        diff = E0 - rootM * E1;
        der = (rootM - E0E1) + g2r2 * Length(diff);
        absDer = fabs(der);
        if (absDer < absDerMin)
        {
            r1 = rootM;
            absDerMin = absDer;
        }
    }
    if (rootP > 0)
    {
        diff = E0 - rootP * E1;
        der = (rootP - E0E1) + g2r2 * Length(diff);
        absDer = fabs(der);
        if (absDer < absDerMin)
        {
            r1 = rootP;
            absDerMin = absDer;
        }
    }

    // Compute hat[g1](r1).
    diff = E0 - r1 * E1;
    Real g1r1 = -E0Xi0 + Length(diff);

    // Compute the minimum of H.
    mHMin = g1r1 + r1 * g2r2;
    if (mHMin < 0)
    {
        mU[0] = invD0Length;
        mU[1] = r1 * invD1Length;
        mU[2] = r1 * r2 * invD2Length;
        Normalize(mU);
    }
    else
    {
        mU = Vector3(0,0,0);
    }
}


void ClampToDomain (Vector3& tmin)
{
    int maxIndex = 0;
    Real maxComp = mU[0];
    if (mU[1] > maxComp)
    {
        maxIndex = 1;
        maxComp = mU[1];
    }
    if (mU[2] > maxComp)
    {
        maxIndex = 2;
        maxComp = mU[2];
    }

    Real smax = 1/maxComp;
    Real der1 = FDer(smax);
    if (der1 <= 0)
    {
        for (int i = 0; i < 3; i++)
        {
```

```
            if (i != maxIndex)
            {
                tmin[i] = smax * mU[i];
            }
            else
            {
                tmin[i] = 1;
            }
        }
    }
    else
    {
        // der0 < 0 and der1 > 0, so bisect to find the root.
        Real root = GetFRoot(0,mHMin,smax,der1);
        tmin = mU * root;
    }
}


Real FDer (Real s)
{
    Vector3 tmp0 = mU[0] * mD0 - mU[1] * mD1;
    Vector3 tmp1 = mU[1] * mD1 - mU[2] * mD2;
    Real result = Length(tmp0) + Length(tmp1);
    tmp0 = s * mU[0] * mD0 - mDelta0;
    Real tmp2 = mU[0] * (mU[0] * mD0SqrLength * s - mD0DotDelta0);
    result += tmp2/Length(tmp0);
    tmp0 = s * mU[2] * mD2 - mDelta1;
    tmp2 = mU[2] * (mU[2] * mD2SqrLength * s - mD2DotDelta1);
    result += tmp2/Length(tmp0);
    return result;
}


Real L (Vector3 t)
{
    Vector3 diff0 = mDelta0 - t[0] * mD0;
    Vector3 diffM1 = t[0] * mD0 - t[1] * mD1;
    Vector3 diffM2 = t[1] * mD1 - t[2] * mD2;
    Vector3 diff1 = t[2] * mD2 - mDelta1;
    return Length(diff0) + Length(diffM1) + Length(diffM2) + Length(diff1);
}


Real LDer (int i, Vector3 t)
{
    if (i == 0)
    {
        Vector3 diff0 = mDelta0 - t[0] * mD0;
        Vector3 diffM1 = t[0] * mD0 - t[1] * mD1;
        Real a0t0 = mD0SqrLength * t[0];
        return (a0t0 - mD0DotDelta0)/Length(diff0) + (a0t0 - mD0DotD1 * t[1])/Length(diffM1);
    }
    else if (i == 1)
    {
        Vector3 diffM1 = t[0] * mD0 - t[1] * mD1;
        Vector3 diffM2 = t[1] * mD1 - t[2] * mD2;
        Real a1t1 = mD1SqrLength * t[1];
        return (a1t1 - mD0DotD1 * t[0])/Length(diffM1) + (a1t1 - mD1DotD2 * t[2])/Length(diffM2);
    }
    else
    {
        Vector3 diffM2 = t[1] * mD1 - t[2] * mD2;
        Vector3 diff1 = t[2] * mD2 - mDelta1;
        Real a2t2 = mD2SqrLength * t[2];
        return (a2t2 - mD1DotD2 * t[1])/Length(diffM2) + (a2t2 - mD2DotDelta1)/Length(diff1);
    }
}


Real GetFRoot (Real s0, Real der0, Real s1, Real der1)
{
```

```
    int numIterations = <user-defined parameter>;
    Real root = 0;
    for (int i = 0; i < numIterations; i++)
    {
        root = (s0 + s1)/2;
        Real derRoot = FDer(root);
        Real product = derRoot * der0;
        if (product < 0)
        {
            s1 = root;
            der1 = derRoot;
        }
        else if (product > 0)
        {
            s0 = root;
            der0 = derRoot;
        }
        else
        {
            break;
        }
    }
    return root;
}


void GetLRoot (int j, Vector3& t, Real s0, Real der0, Real s1, Real der1)
{
    int numIterations = <user-defined parameter>;
    for (int i = 0; i < numIterations; i++)
    {
        t[j] = (s0 + s1)/2;
        Real derRoot = LDer(j,t);
        Real product = derRoot * der0;
        if (product < 0)
        {
            s1 = t[j];
            der1 = derRoot;
        }
        else if (product > 0)
        {
            s0 = t[j];
            der0 = derRoot;
        }
        else
        {
            break;
        }
    }
}


void Search (int j, Vector3& tmin, Real& lmin, Vector3& lder)
{
    Real save, der;

    if (lder[j] > 0)
    {
        save = tmin[j];
        tmin[j] = 0;
        der = LDer(j,tmin);
        tmin[j] = save;

        if (der < (Real)0)
        {
            // Bisect [0,t[j]] to find the root.
            GetLRoot(j,tmin,0,der,tmin[j],lder[j]);
        }
        else
        {
            tmin[j] = 0;
        }
```

```
    }
    else // lder[j] < 0
    {
        save = tmin[j];
        tmin[j] = 1;
        der = LDer(j,tmin);
        tmin[j] = save;

        if (der > (Real)0)
        {
            // Bisect [t[j],1] to find the root.
            GetLRoot(j,tmin,tmin[j],lder[j],1,der);
        }
        else
        {
            tmin[j] = 1;
        }
    }

    lmin = L(tmin);
    for (int i = 0; i < 3; i++)
    {
        lder[i] = LDer(i,tmin);
    }
}
```

4.4 The Five-Triangle Numerical Implementation

Pseudocode for computing the shortest path through five triangles is shown next. The variables prefixed with an `m` are assumed to be accessible by the other functions. The ampersands in the function definitions indicate that the corresponding variables are the outputs of the function.

```
void GetPath (Vector3 P0, Vector3 P1, Vector3 A, Vector3 B0, Vector3 B1, Vector3 B2, Vector3 B3,
    Vector4& tmin, Real& lmin, Vector3& M0, Vector3& M1, Vector3& M2, Vector3& M3)
{
    Vector3 mDelta0 = P0 - A;
    Vector3 mDelta1 = P1 - A;
    Vector3 mD0 = B0 - A;
    Vector3 mD1 = B1 - A;
    Vector3 mD2 = B2 - A;
    Vector3 mD3 = B3 - A;
    Real mDelta0SqrLength = Dot(mDelta0,mDelta0);
    Real mDelta1SqrLength = Dot(mDelta1,mDelta1);
    Real mD0SqrLength = Dot(mD0,mD0);
    Real mD1SqrLength = Dot(mD1,mD1);
    Real mD2SqrLength = Dot(mD2,mD2);
    Real mD3SqrLength = Dot(mD3,mD3);
    Real mD0DotD1 = Dot(mD0,mD1);
    Real mD1DotD2 = Dot(mD1,mD2);
    Real mD2DotD3 = Dot(mD2,mD3);
    Real mD0DotDelta0 = Dot(mD0,mDelta0);
    Real mD3DotDelta1 = Dot(mD3,mDelta1);

    // Compute direction mU so that mHMin = F'(0) < 0.
    Vector4 mU;
    Real mHMin;
    ComputeLargestInitialDecrease();
    if (mHMin >= 0)
    {
        // The minimum is L(0,0,0,0), so nothing to do.
        tmin = Vector4(0,0,0,0);
        lmin = sqrt(mDelta0SqrLength) + sqrt(mDelta1SqrLength);
        M0 = A;
        M1 = A;
        M2 = A;
        M3 = A;
```

```
        return;
    }

    // Search the ray (t0,t1,t2,t3) = s*(u0,u1,u2,u3), s >= 0, for a minimum.  The search
    // must be clamped to t0 <= 1, t1 <= 1, t2 <= 1, and t3 <= 1.
    ClampToDomain(tmin);

    // Compute the initial length and derivatives.
    lmin = L(rkTMin);
    Vector4 lder(LDer(0,tmin), LDer(1,tmin), LDer(2,tmin), LDer(3,tmin));

    // Coordinate-direction search.
    int maxIterations = <user-defined parameter>;
    Vector3 prevtmin(0,0,0,0);
    int i, j;
    for (i = 0; i < maxIterations; i++)
    {
        // Terminate when (nearly) at the global minimum.
        Real gradientEpsilon = <user-defined parameter>;
        if (Length(lder) < gradientEpsilon)
        {
            break;
        }

        // Terminate when (nearly) at the global minimum.
        Real differenceEpsilon = <user-defined parameter>;
        if (Length(tmin - prevtmin) < differenceEpsilon)
        {
            break;
        }
        prevtmin = tmin;

        for (j = 0; j < 4; j++)
        {
            if (lder[j] != 0)
            {
                Search(j,tmin,lmin,lder);
            }
        }
    }

    M0 = A + tmin[0] * mD0;
    M1 = A + tmin[1] * mD1;
    M2 = A + tmin[2] * mD2;
    M3 = A + tmin[3] * mD3;
}


void ComputeLargestInitialDecrease ()
{
    Real invD0Length = 1/sqrt(mD0SqrLength);
    Real invD1Length = 1/sqrt(mD1SqrLength);
    Real invD2Length = 1/sqrt(mD2SqrLength);
    Real invD3Length = 1/sqrt(mD3SqrLength);
    Vector3 E0 = mD0 * invD0Length;
    Vector3 E1 = mD1 * invD1Length;
    Vector3 E2 = mD2 * invD2Length;
    Vector3 E3 = mD3 * invD3Length;
    Vector3 Xi0 = mDelta0/Length(mDelta0);
    Vector3 Xi1 = mDelta1/Length(mDelta1);
    Real E0E1 = Dot(E0,E1);
    Real E1E2 = Dot(E1,E2);
    Real E2E3 = Dot(E2,E3);
    Real E0Xi0 = Dot(E0,Xi0);
    Real E3Xi1 = Dot(E3,Xi1);

    // Compute g3'(0).
    Real g3der0 = -E2E3 - E3Xi1;
    if (g3der0 >= 0)
    {
        // H cannot be negative on its domain.
        mU = Vector4(0,0,0,0);
```

```
    mHMin = 0;
    return;
}

// Compute the roots for the quadratic associated with g3'(r3) = 0.
Real arg = (1 - E2E3 * E2E3)/(1 - E3Xi1 * E3Xi1);
Real rootArg = sqrt(arg);
Real rootM = E2E3 - E3Xi1 * rootArg;
Real rootP = E2E3 + E3Xi1 * rootArg;

Real r3 = 0;
Real absDerMin = INFINITY;
Vector3 diff;
Real der, absDer;
if (rootM > 0)
{
    diff = E2 - rootM * E3;
    der = (rootM - E2E3) - E3Xi1 * Length(diff);
    absDer = fabs(der);
    if (absDer < absDerMin)
    {
        r3 = rootM;
        absDerMin = absDer;
    }
}
if (rootP > 0)
{
    diff = E2 - rootP * E3;
    der = (rootP - E2E3) - E3Xi1 * Length(diff);
    absDer = fabs(der);
    if (absDer < absDerMin)
    {
        r3 = rootP;
        absDerMin = absDer;
    }
}

// Compute g3(r3).
diff = E2 - r3 * E3;
Real g3r3 = Length(diff) - E3Xi1 * r3;

// Compute hat[g2]'(0).
Real g2der0 = -E1E2 + g3r3;
if (g2der0 >= 0)
{
    // H cannot be negative on its domain.
    mU = Vector4(0,0,0,0);
    mHMin = 0;
    return;
}

// Compute the roots for the quadratic associated with hat[g2]'(r2) = 0.
arg = (1 - E1E2 * E1E2)/(1 - g3r3 * g3r3);
rootArg = sqrt(arg);
rootM = E1E2 - g3r3 * rootArg;
rootP = E1E2 + g3r3 * rootArg;

Real r2 = 0;
absDerMin = INFINITY;
if (rootM > 0)
{
    diff = E1 - rootM * E2;
    der = (rootM - E1E2) + g3r3 * Length(diff);
    absDer = fabs(der);
    if (absDer < absDerMin)
    {
        r2 = rootM;
        absDerMin = absDer;
    }
}
if (rootP > 0)
{
```

```
    diff = E1 - rootP * E2;
    der = (rootP - E1E2) + g3r3 * Length(diff);
    absDer = fabs(der);
    if (absDer < absDerMin)
    {
        r2 = rootP;
        absDerMin = absDer;
    }
}

// Compute hat[g2](r2).
diff = E1 - r2 * E2;
Real g2r2 = Length(diff) + g3r3 * r2;

// Compute hat[g1]'(0).
Real g1der0 = -E0E1 + g2r2;
if (g1der0 >= 0)
{
    // H cannot be negative on its domain.
    mU = Vector4(0,0,0,0);
    mHMin = 0;
    return;
}

// Compute the roots for the quadratic associated with hat[g1]'(r1) = 0.
arg = (1 - E0E1 * E0E1)/(1 - g2r2 * g2r2);
rootArg = sqrt(arg);
rootM = E0E1 - g2r2 * rootArg;
rootP = E0E1 + g2r2 * rootArg;

Real r1 = 0;
absDerMin = INFINITY;
if (rootM > 0)
{
    diff = E0 - rootM * E1;
    der = (rootM - E0E1) + g2r2 * Length(diff);
    absDer = fabs(der);
    if (absDer < absDerMin)
    {
        r1 = rootM;
        absDerMin = absDer;
    }
}
if (rootP > 0)
{
    diff = E0 - rootP * E1;
    der = (rootP - E0E1) + g2r2 * Length(diff);
    absDer = fabs(der);
    if (absDer < absDerMin)
    {
        r1 = rootP;
        absDerMin = absDer;
    }
}

// Compute hat[g1](r1).
diff = E0 - r1 * E1;
Real g1r1 = -E0Xi0 + Length(diff);

// Compute the minimum of H.
mHMin = g1r1 + r1 * g2r2;
if (mHMin < (Real)0)
{
    mU[0] = invD0Length;
    mU[1] = r1 * invD1Length;
    mU[2] = r1 * r2 * invD2Length;
    mU[3] = r1 * r2 * r3 * invD3Length;
    Normalize(mU);
}
else
{
    mU = Vector4(0,0,0,0);
```

```
    }
}


void ClampToDomain (Vector4& tmin)
{
    int maxIndex = 0;
    Real maxComp = mU[0];
    if (mU[1] > maxComp)
    {
        maxIndex = 1;
        maxComp = mU[1];
    }
    if (mU[2] > maxComp)
    {
        maxIndex = 2;
        maxComp = mU[2];
    }
    if (mU[3] > maxComp)
    {
        maxIndex = 3;
        maxComp = mU[3];
    }

    Real smax = 1/maxComp;
    Real der1 = FDer(smax);
    if (der1 <= 0)
    {
        for (int i = 0; i < 4; i++)
        {
            if (i != maxIndex)
            {
                tmin[i] = smax * mU[i];
            }
            else
            {
                tmin[i] = 1;
            }
        }
    }
    else
    {
        // der0 < 0 and der1 > 0, so bisect to find the root.
        Real root = GetFRoot(0,mHMin,smax,der1);
        tmin = mU * root;
    }
}


Real FDer (Real s)
{
    Vector3 tmp0 = mU[0] * mD0 - mU[1] * mD1;
    Vector3 tmp1 = mU[1] * mD1 - mU[2] * mD2;
    Vector3 tmp2 = mU[2] * mD2 - mU[3] * mD3;
    Real result = Length(tmp0) + Length(tmp1) + Length(tmp2);
    tmp0 = s * mU[0] * mD0 - mDelta0;
    Real tmp3 = mU[0] * (mU[0] * mD0SqrLength * s - mD0DotDelta0);
    result += tmp3/Length(tmp0);
    tmp0 = s * mU[3] * mD3 - mDelta1;
    tmp3 = mU[3] * (mU[3] * mD3SqrLength * s - mD3DotDelta1);
    result += tmp3/Length(tmp0);
    return result;
}


Real L (Vector4 t)
{
    Vector3 diff0 = mDelta0 - t[0] * mD0;
    Vector3 diffM1 = t[0] * mD0 - t[1] * mD1;
    Vector3 diffM2 = t[1] * mD1 - t[2] * mD2;
    Vector3 diffM3 = t[2] * mD2 - t[3] * mD3;
    Vector3 diff1 = t[3] * mD3 - mDelta1;
```

```
    return Length(diff0) + Length(diffM1) + Length(diffM2) + Length(diffM3) + Length(diff1);
}


Real LDer (int i, Vector4 t)
{
    if (i == 0)
    {
        Vector3 diff0 = mDelta0 - t[0] * mD0;
        Vector3 diffM1 = t[0] * mD0 - t[1] * mD1;
        Real a0t0 = mD0SqrLength * t[0];
        return (a0t0 - mD0DotDelta0)/Length(diff0) + (a0t0 - mD0DotD1 * t[1])/Length(diffM1);
    }
    else if (i == 1)
    {
        Vector3 diffM1 = t[0] * mD0 - t[1] * mD1;
        Vector3 diffM2 = t[1] * mD1 - t[2] * mD2;
        Real a1t1 = mD1SqrLength * t[1];
        return (a1t1 - mD0DotD1 * t[0])/Length(diffM1) + (a1t1 - mD1DotD2 * t[2])/Length(diffM2);
    }
    else if (i == 2)
    {
        Vector3 diffM2 = t[1] * mD1 - t[2] * mD2;
        Vector3 diffM3 = t[2] * mD2 - t[3] * mD3;
        Real a2t2 = mD2SqrLength * t[2];
        return (a2t2 - mD1DotD2 * t[1])/Length(diffM2) + (a2t2 - mD2DotD3 * t[3])/Length(diffM3);
    }
    else
    {
        Vector3 diffM3 = t[2] * mD2 - t[3] * mD3;
        Vector3 diff1 = t[3] * mD3 - mDelta1;
        Real a3t3 = mD3SqrLength * t[3];
        return (a3t3 - mD2DotD3 * t[2])/Length(diffM3) + (a3t3 - mD3DotDelta1)/Length(diff1);
    }
}


Real GetFRoot (Real s0, Real der0, Real s1, Real der1)
{
    int numIterations = <user-defined parameter>;
    Real root = 0;
    for (int i = 0; i < numIterations; i++)
    {
        root = (s0 + s1)/2;
        Real derRoot = FDer(root);
        Real product = derRoot * der0;
        if (product < 0)
        {
            s1 = root;
            der1 = derRoot;
        }
        else if (product > 0)
        {
            s0 = root;
            der0 = derRoot;
        }
        else
        {
            break;
        }
    }
    return root;
}


void GetLRoot (int j, Vector4& t, Real s0, Real der0, Real s1, Real der1)
{
    int numIterations = <user-defined parameter>;
    for (int i = 0; i < numIterations; i++)
    {
        t[j] = (s0 + s1)/2;
        Real derRoot = LDer(j,t);
```

```
        Real product = derRoot * der0;
        if (product < 0)
        {
            s1 = t[j];
            der1 = derRoot;
        }
        else if (product > 0)
        {
            s0 = t[j];
            der0 = derRoot;
        }
        else
        {
            break;
        }
    }
}


void Search (int j, Vector4& tmin, Real& lmin, Vector4& lder)
{
    Real save, der;

    if (lder[j] > 0)
    {
        save = tmin[j];
        tmin[j] = 0;
        der = LDer(j,tmin);
        tmin[j] = save;

        if (der < (Real)0)
        {
            // Bisect [0,t[j]] to find the root.
            GetLRoot(j,tmin,0,der,tmin[j],lder[j]);
        }
        else
        {
            tmin[j] = 0;
        }
    }
    else // lder[j] < 0
    {
        save = tmin[j];
        tmin[j] = 1;
        der = LDer(j,tmin);
        tmin[j] = save;

        if (der > (Real)0)
        {
            // Bisect [t[j],1] to find the root.
            GetLRoot(j,tmin,tmin[j],lder[j],1,der);
        }
        else
        {
            tmin[j] = 1;
        }
    }

    lmin = L(tmin);
    for (int i = 0; i < 4; i++)
    {
        lder[i] = LDer(i,tmin);
    }
}
```

4.5 The N-Triangle Numerical Implementation

The pattern is clear in the 3-, 4-, and 5-triangle cases. The function `ComputeLargestInitialDecrease` is the most complicated one, illustrated by the following pseudocode. To avoid having to remember the values $\bar{r}_1$, $\bar{r}_2$, and so on, for use in computing the final $\mathbf{U}$ components, the $\mathbf{U}$ components are initialized to the inverse lengths of the $\mathbf{D}_i$ vectors. These components are updated each time a new $\bar{r}_i$ value is computed. The code also assumes the proper storage for $\mathbf{E}_i$, $\boldsymbol{\xi}_0$, and $\boldsymbol{\xi}_1$, and the last two of these vectors are assumed to have already been computed by the caller of the function.

```
void ComputeLargestInitialDecrease ()
{
    int N = <number of t-parameters, the number of triangles minus one>;
    int Nm1 = N - 1;
    int i0, i1;

    for (i0 = 0; i0 < N; i0++)
    {
        mU[i0] = 1/sqrt(mDSqrLength[i0]);
        mE[i0] = mD[i0] * mU[i0];
    }

    for (i0 = 0, i1 = 1; i0 < Nm1; i0++, i1++)
    {
        mEDotENext[i0] = Dot(mE[i0],mE[i1]);
    }

    Real E0Xi0 = Dot(mE[0],mXi0);
    Real ENm1Xi1 = Dot(mE[iNm1],mXi1);

    Real gr = -ENm1Xi1;
    Vector3 kDiff;
    Real r;
    int i, j;
    for (i = iNm1; i >= 1; i--)
    {
        // Compute gi'(0).
        Real Eim1Ei = mEDotENext[i-1];
        Real gder0 = -Eim1Ei + gr;
        if (gder0 >= 0)
        {
            // H cannot be negative on its domain.
            mU = VectorN::ZERO;
            mHMin = 0;
            return;
        }

        // Compute the roots for the quadratic associated with gi'(ri) = 0.
        Real arg = (1 - Eim1Ei * Eim1Ei)/(1 - gr * gr);
        Real rootArg = sqrt(fArg);
        Real rootM = Eim1Ei - gr * rootArg;
        Real rootP = Eim1Ei + gr * rootArg;

        r = 0;
        Real absDerMin = INFINITY;
        Real der, absDer;
        if (rootM > 0)
        {
            diff = mE[i-1] - rootM * mE[i];
            der = (rootM - Eim1Ei) + gr * Length(diff);
            absDer = fabs(der);
            if (absDer < absDerMin)
            {
                r = rootM;
                absDerMin = absDer;
            }
        }
        if (rootP > 0)
```

```
            {
                diff = mE[i-1] - rootM * mE[i];
                der = (rootP - Eim1Ei) + gr * Length(diff);
                absDer = abs(der);
                if (absDer < absDerMin)
                {
                    r = rootP;
                    absDerMin = absDer;
                }
            }

            // Compute gi(ri).
            diff = mE[i-1] - r * mE[i];
            gr = Length(diff) + gr * r;

            // Partial construction of U.
            for (j = Nm1; j >= i; j--)
            {
                mU[j] *= r;
            }
        }

        // Compute the minimum of H.
        mHMin = gr - E0Xi0;
        if (mHMin < 0)
        {
            Normalize(mU);
        }
        else
        {
            mU = VectorN::ZERO;
        }
    }
```

The actual code is in the files

```
GeometricTools/WildMagic4/SampleFoundation/TriangleMeshGeodesics/GeodesicSpecial.{h,cpp}
```

and handles internally the 1- and 2-triangle cases separately from the case of 3 or more triangles.

5 The General Case

The endpoints of the geodesic are $\mathbf{Q}_0$ and $\mathbf{Q}_1$. Insert these into the triangle mesh as vertices. If inserted into the interior of a triangle, subdivide to three new triangles and edges. If inserted into the interior of an edge of a triangle, subdivide two triangles to four triangles and split the edge. If already a vertex, nothing to do.

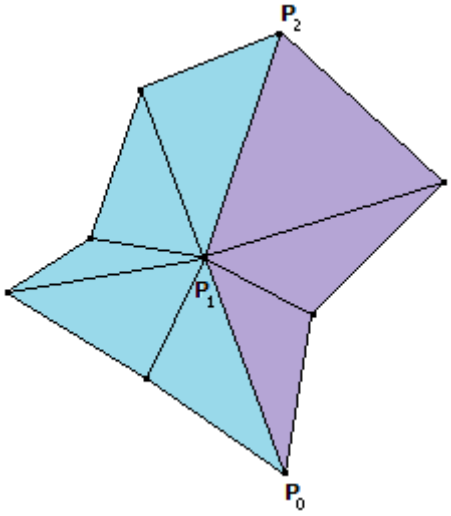
Use Dijkstra's algorithm to compute a minimum-length path of mesh edges that connect $\mathbf{Q}_0$ to $\mathbf{Q}_1$. Let this path be

$$\langle \mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n \rangle \tag{98}$$

where $\mathbf{P}_0 = \mathbf{Q}_0$ and $\mathbf{P}_n = \mathbf{Q}_1$.

Relaxation now begins. Start with $\langle \mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2 \rangle$. Figure 5.1 shows the neighborhood of edges and triangles for $\mathbf{P}_1$.

Figure 5.1 The shortest edge path between $\mathbf{P}_0$ and $\mathbf{P}_2$ is through $\mathbf{P}_1$. The neighborhood of $\mathbf{P}_1$ is shown. One relaxation is made in the collection of purple triangles, another in the collection of blue triangles.



If the subpath can be made shorter, $\mathbf{P}_1$ is removed from the current path and new vertices are added. For example, if the relaxation takes the curve into the collection of purple triangles, then two new vertices are added and the updated current path is $\langle \mathbf{P}_0, \mathbf{M}_0, \mathbf{M}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n \rangle$. The process is repeated for $\langle \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4 \rangle$, and so on, until the last triple of vertices is processed.

The third pass starts a sequence of iterations over the vertices of the current candidate for the geodesic. If the current path is

$$\langle \mathbf{P}'_0, \mathbf{P}'_1, \ldots, \mathbf{P}'_{n-1}, \mathbf{P}'_n \rangle \tag{99}$$

then each triple $\langle \mathbf{P}'_{i-1}, \mathbf{P}'_i, \mathbf{P}'_{i+1} \rangle$ is processed for $1 \le i \le n-1$. The middle vertex $\mathbf{P}'_i$ is either a mesh vertex or a point living in the interior of a mesh edge. If it is a mesh vertex, then the update is based on the ideas mentioned previously about removing the vertex from the path and replacing it by some edge-interior vertices. If it is an edge-interior point, then it is moved based on the minimization discussed for two consecutive triangles. This algorithm requires tagging the points in the current path about whether they are mesh vertices or edge-interior points.

The sequence of iterations continue until some convergence criterion is met.