The Area of Intersecting Ellipses

David Eberly, Geometric Tools, Redmond WA 98052
https://www.geometrictools.com/

Created: September 4, 2008
Last Modified: November 26, 2016

Contents

1 Introduction

This document describes an algorithm for computing the area of intersection of two ellipses. The formulas are in closed form, thus providing the *exact* area in terms of real-valued arithmetic. Naturally, the computer evaluation of the trigonometric functions in the formulas has some numerical round-off errors, but the formulas allow you to avoid (1) approximating the ellipses by convex polygons, (2) using the intersection of convex polygons as an approximation to the intersection of ellipses, and (3) using the area of intersection of convex polygons as an approximation to the area of intersection of ellipses.

The algorithm has two main aspects: Computing the points of intersection of the ellipses and computing the area bounded by a line and an elliptical arc. Computing the intersection points is described in lengthy detail in Intersection of Ellipses.

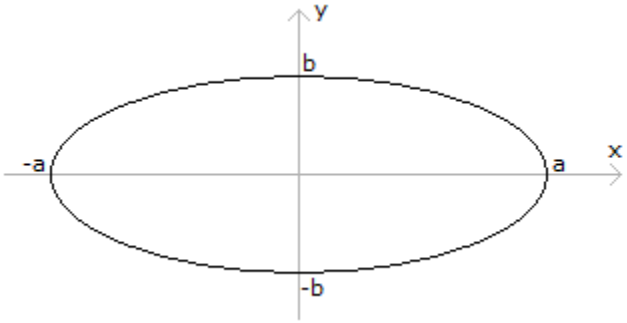
2 Area of an Ellipse

An axis-aligned ellipse centered at the origin is

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1 \tag{1}$$

where I assume that $a > b$, in which case the major axis is along the x-axis. Figure 1 shows such an ellipse.

Figure 1. An axis-aligned ellipse centered at the origin with $a > b$.



The area bounded by the ellipse is πab. Using the methods of calculus, the area A is four times that of the area in the first quadrant,

$$A = 4\int_0^a y\,dx = 4\int_0^a b\sqrt{1-(x/a)^2}\,dx \tag{2}$$

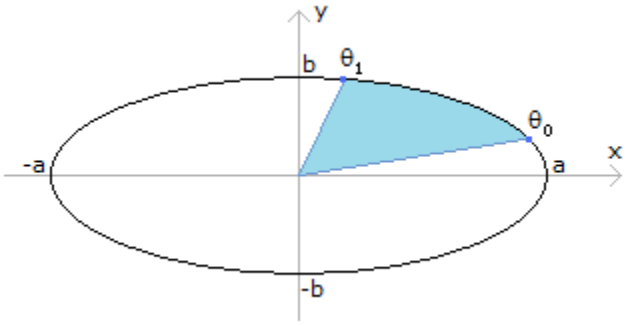
The integral may be computed using the change of variables $x = a\cos\phi$ for $0 \le \phi \le \pi/2$. The differential is $dx = -a\sin\phi\, d\phi$ and the area is

$$\begin{aligned} A &= 4\int_0^a b\sqrt{1-(x/a)^2}\, dx \\ &= 4b\int_{\pi/2}^0 \sin\phi(-a\sin\phi\, d\phi) \\ &= 4ab\int_0^{\pi/2} \sin^2\phi\, d\phi \\ &= 2ab\int_0^{\pi/2} (1-\cos(2\phi))\ d\phi \\ &= 2ab\left(\phi - \tfrac{1}{2}\sin(2\phi)\right)\Big|_0^{\pi/2} \\ &= 2ab\left[\left(\tfrac{\pi}{2} - \tfrac{1}{2}\sin(\pi)\right) - \left(0 - \tfrac{1}{2}\sin(0)\right)\right] \\ &= \pi ab \end{aligned} \tag{3}$$

3 Area of an Elliptical Sector

An *elliptical arc* is a portion of the ellipse bounded by two points on the ellipse. The arc is delimited by angles θ_0 and θ_1 with $\theta_0 < \theta_1$. An *elliptical sector* is the region bounded by an elliptical arc and the line segments containing the origin and the endpoints of the arc. Figure 2 shows an elliptical arc and the corresponding elliptical sector.

Figure 2. An elliptical arc and its corresponding elliptical sector.



The polar-coordinate representation of the arc is obtained by substituting $x = r\cos\theta$ and $y = r\sin\theta$ into Equation (1) and solving for r^2,

$$r^2 = \frac{a^2b^2}{b^2\cos^2\theta + a^2\sin^2\theta} \tag{4}$$

The area of the sector is

$$A(\theta_0,\theta_1) = \int_{\theta_0}^{\theta_1} \frac{1}{2}r^2\, d\theta = \int_{\theta_0}^{\theta_1} \frac{\left(a^2b^2/2\right)\, d\theta}{b^2\cos^2\theta + a^2\sin^2\theta} \tag{5}$$

An antiderivative of the integrand is

$$F(\theta) = \frac{ab}{2}\left[\theta - \text{Tan}^{-1}\left(\frac{(b-a)\sin 2\theta}{(b+a)+(b-a)\cos 2\theta}\right)\right] \tag{6}$$

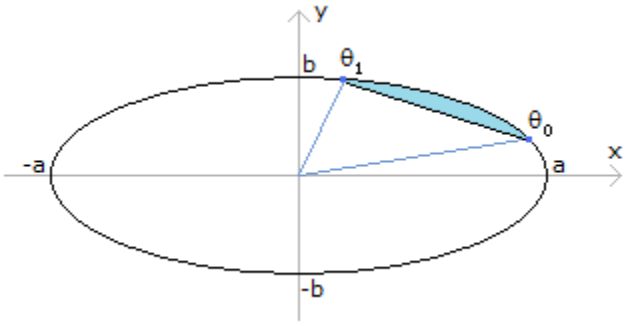
where $\text{Tan}^{-1}(z)$ is the principal branch of the inverse tangent function whose range is $(-\pi/2, \pi/2)$. The area of the elliptical sector is therefore

$$A(\theta_0, \theta_1) = F(\theta_1) - F(\theta_0) \tag{7}$$

4 Area Bounded by a Line Segment and an Elliptical Arc

Figure 3 shows the region bounded by an elliptical arc and the line segment connecting the arc's endpoints.

Figure 3. The region bounded by an elliptical arc and the line segment connecting the arc's endpoints.



The area of this region is the area of the elliptical sector minus the area of the triangle whose vertices are the origin, $(0, 0)$, and the arc endpoints $(x_0, y_0) = (r_0 \cos\theta_0, r_0 \sin\theta_0)$ and $(x_1, y_1) = (r_1 \cos\theta_1, r_1 \sin\theta_1)$, where θ_i are the polar angles to the points and where r_i are determined using Equation (4). The triangle area is

$$\frac{1}{2}|x_1 y_0 - x_0 y_1| = \frac{r_0 r_1}{2}|\cos\theta_1 \sin\theta_0 - \cos\theta_0 \sin\theta_1| = \frac{r_0 r_1}{2}|\sin(\theta_1 - \theta_0)| \tag{8}$$

In an implementation it is sufficient to use $|x_1 y_0 - x_0 y_1|/2$ rather than compute the right-hand side of Equation (8).

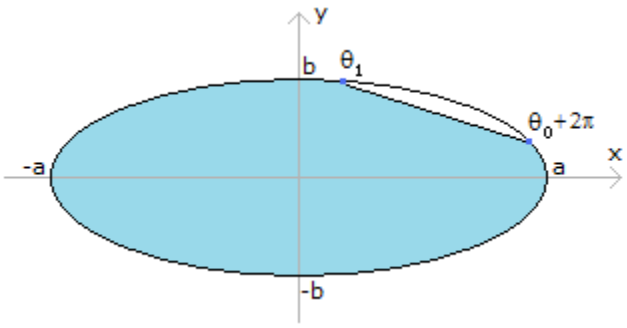
If $\alpha(\theta_0, \theta_1)$ denotes the area of the aforementioned region, then

$$\alpha(\theta_0, \theta_1) = A(\theta_0, \theta_1) - \frac{1}{2}|x_1 y_0 - x_0 y_1| \tag{9}$$

where $A(\theta_0, \theta_1)$ is the area of the sector as defined in Equation (7) and the other term on the right-hand side is the area of the triangle.

The region of interest could be that bounded by the line segment and the elliptical arc that is spanned counterclockwise from θ_1 to $\theta_0 + 2\pi$. Figure 4 illustrates.

Figure 4. The other region bounded by an elliptical arc and the line segment connecting the arc's endpoints.



The area of this region is

$$\alpha(\theta_1, \theta_0 + 2\pi) = \pi ab - \alpha(\theta_0, \theta_1) \tag{10}$$

The bounded region has the area of the ellipse minus the area of the smaller region bounded by the line segment and elliptical arc. In Equation (9), the origin is outside the bounded region. In Equation (10), the origin is inside the bounded region.

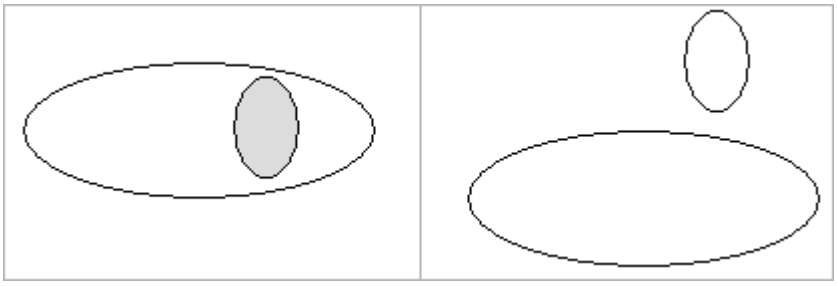
5 Area of Intersecting Ellipses

We need to compute the points of intersection between two ellipses, including classification whether the intersections are transverse or tangential. A robust algorithm for this is described in Intersection of Ellipses, and the implementation is in the file GteIntrEllipse2Ellipse2.h. In the following discussion, the ellipses are named E_0 and E_1. The subsections describe each geometric configuration that can arise.

5.1 No Intersection Points

One ellipse is contained strictly in the other, or the ellipses (as solids) are separated. Figure 5 illustrates.

Figure 5. Left: One ellipse is contained by the other. Right: The ellipses are separated.



In the case of containment, the area of intersection is the area of the smaller ellipse. In the case of separation, the area of intersection is zero. The logic is shown in Listing 1.

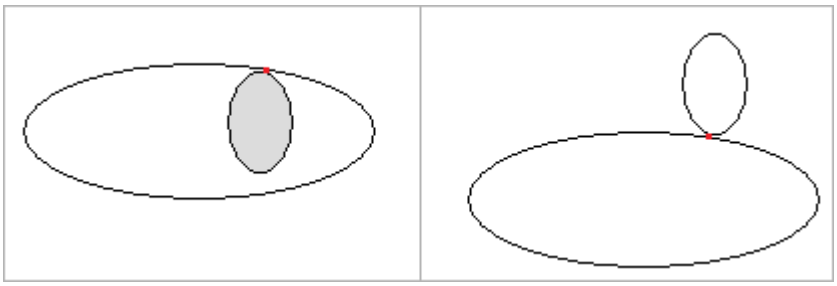
Listing 1. The area computation when the ellipses are separated, one contains the other, or one is outside the other but just touching the other.

```
Real AreaCS(Ellipse E0, Ellipse E1, int numPoints)
{
    if (numPoints <= 1)
    {
        Vector2<Real> diff = E0.center - E1.center;
        Real qform0 = Dot(diff, E0.M * diff);
        Real qform1 = Dot(diff, E1.M * diff);
        if (qform0 > 1 && qform1 > 1)
        {
            // Each ellipse center is outside the other ellipse, so the
            // ellipses are separated (numPoints == 0) or outside each
            // other and just touching (numPoints == 1).
            return 0;
        }
        else
        {
            // One ellipse is inside the other.  Determine this indirectly by
            // comparing areas.
            Real product0 = E0.extent[0] * E0.extent[1];
            Real product1 = E1.extent[0] * E1.extent[1];
            if (product0 < product1)
            {
                // E1 contains E0
                return pi * product0;
            }
            else
            {
                return pi * product1;
            }
        }
    }
    else  // In our implementation, we set numPoints to INT_MAX for this case.
    {
        // The ellipses are the same.
        return pi * E0.extent[0] * E0.extent[1];
    }
}
```

5.2 One Intersection Point

One ellipse is contained in the other but the two ellipses are tangent at the point of intersection, or the ellipses (as solids) are separated except for a single point of tangency. Figure 6 illustrates.

Figure 6. Left: One ellipse is contained by the other but they are tangent at a single point. Right: The ellipses are separated except for a single point of tangency. The intersection point is drawn as a red dot.

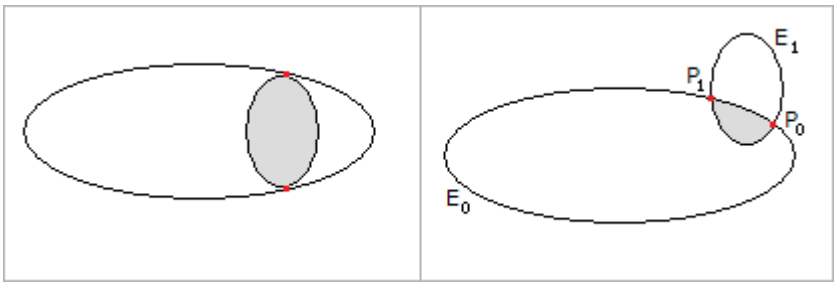


The area of intersection is computed using the function AreaCS of Listing 1.

5.3 Two Intersection Points

Two possible configurations are possible. Either one ellipse is contained in the other but the two ellipses are tangent at the point of intersection, or the ellipses intersect at two distinct points. Figure 7 illustrates.

Figure 7. Left: One ellipse is contained by the other but they are tangent at two points. Right: The ellipses intersect at two distinct points. The intersection points are drawn as red dots.



For the case shown in the left of the figure, the area of intersection is computed using the function AreaCS because one ellipse is contained in the other.

The other case is shown in the right of the figure. The region of intersection is bounded by two elliptical arcs, one from each ellipse. We must compute the areas bounded by the chord connecting the arc endpoints,

one for ellipse E_0 and one for ellipse E_1. Observe that the chord partitions an ellipse into two subsets. We need to decide which subset is the one for which we need to compute the area. This is determined by the ordering of the two points of intersection. In Figure 7, notice that at $\mathbf{P}_0$, the tangent vector pointing in the direction of the counterclockwise traversal of E_0 can be used to decide that we compute the area between the ordered chord $\langle \mathbf{P}_0, \mathbf{P}_1 \rangle$ and the E_0-arc above it.

Ellipse E_1 has the opposite property. The tangent pointing in the direction of the counterclockwise traversal of E_1 points outside E_1, so the ordered chord is $\langle \mathbf{P}_1, \mathbf{P}_0 \rangle$ and we compute the area between it and the E_1-arc below it. The ordering therefore relies solely on computing the sign of the dot-perp of tangents, or equivalently computing the sign of the dot-perp of normals (which are provided by the gradients of the quadratic polynomials defining the ellipses).

Listing 2 is the pseudocode for computing the area for the right-most configuration of Figure 7.

Listing 2. Pseudocode for computing the area of intersection when there are two distinct points of intersection. The code is shared by the 3-point intersection case, so the last two parameters are indices of the proper 2 points to use in the array of intersection points.

```
Real Area2(Ellipse E0, Ellipse E1, int i0, int i1, Vector2<Real> points[4])
{
    // The endpoints of the chord.
    Vector2<Real> P0 = points[i0], P1 = points[i1];

    // Compute locations relative to the ellipses.
    Vector2<Real> P0mC0 = P0 - E0.center, P0mC1 = P0 - E1.center;
    Vector2<Real> P1mC0 = P1 - E0.center, P1mC1 = P1 - E1.center;

    // Compute the ellipse normal vectors at endpoint P0.  This is sufficient
    // information to determine chord endpoint order.  The matrices M are
    // from the standard form, (X-C)^T * M * (X-C) = 1.
    Vector2<Real> N0 = E0.M * P0mC0, N1 = E1.M * P0mC1;
    Real dotperp = DotPerp(N1, N0);

    // Choose the endpoint order for the chord region associated with E0.
    if (dotperp > 0)
    {
        // The chord order for E0 is <P0,P1> and for E1 is <P1,P0>.
        return
            ComputeAreaChordRegion(E0, P0mC0, P1mC0) +
            ComputeAreaChordRegion(E1, P1mC1, P0mC1);
    }
    else
    {
        // The chord order for E0 is <P1,P0> and for E1 is <P0,P1>.
        return
            ComputeAreaChordRegion(E0, P1mC0, P0mC0) +
            ComputeAreaChordRegion(E1, P0mC1, P1mC1);
    }
}
```

The function ComputeAreaChordRegion is an implementation of the algorithm described in Section 4. Listing 3 is pseudocode for this method.

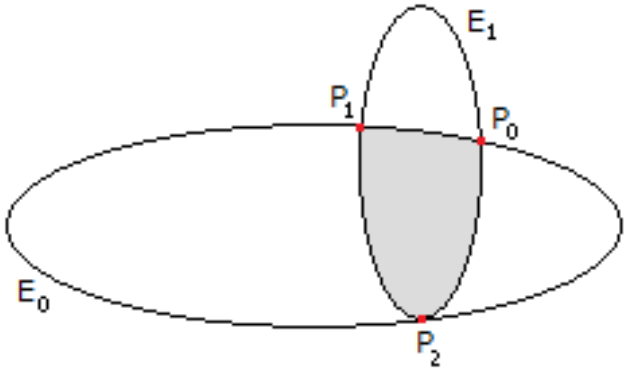
Listing 3. Pseudocode for computing the area bounded by an elliptical arc and the chord that connects the endpoints of the arc.

```
Real ComputeAreaChordRegion(Ellipse E, Vector2<Real> P0mC, Vector2<Real> P1mC)
{
    // Compute polar coordinates for P0 and P1 on the ellipse.
    Real x0 = Dot(E.axis[0], P0mC);
    Real y0 = Dot(E.axis[1], P0mC);
    Real theta0 = atan2(y0, x0);
    Real x1 = Dot(E.axis[0], P1mC);
    Real y1 = Dot(E.axis[1], P1mC);
    Real theta1 = atan2(y1, x1);

    // The arc straddles the atan2 discontinuity on the negative x-axis.  Wrap
    // the second angle to be larger than the first angle.
    if (theta1 < theta0)
    {
        theta1 += twoPi;
    }

    // Compute the area portion of the sector due to the triangle.
    Real triArea = fabs(DotPerp(P0mC, P1mC)) / 2;

    // Compute the chord region area.
    Real dtheta = theta1 - theta0;
    Real F0, F1, sectorArea;
    if (dtheta <= pi)
    {
        // Use the area formula directly.
        //   area(theta0,theta1) = F(theta1) - F(theta0) - area(triangle)
        F0 = ComputeIntegral(E, theta0);
        F1 = ComputeIntegral(E, theta1);
        sectorArea = F1 - F0;
        return sectorArea - triArea;
    }
    else
    {
        // The angle of the elliptical sector is larger than pi radians.
        // Use the area formula
        //   area(theta0,theta1) = pi*a*b - area(theta1,theta0)
        theta0 += twoPi;  // ensure theta0 > theta1
        F0 = ComputeIntegral(E, theta0);
        F1 = ComputeIntegral(E, theta1);
        sectorArea = F0 - F1;
        return pi * E.extent[0] * E.extent[1] - (sectorArea - triArea);
    }
}

Real ComputeIntegral(Ellipse E, Real theta)
{
    Real twoTheta = 2 * theta, sn = sin(twoTheta), cs = cos(twoTheta);
    Real halfAB = E.extent[0] * E.extent[1] / 2;
    Real BpA = E.extent[1] + E.extent[0];
    Real BmA = E.extent[1] - E.extent[0];
    Real arg = BmA * sn / (BpA + BmA * cs);
    return halfAB * (theta - atan(arg));
}
```

5.4 Three Intersection Points

The two ellipses intersect tangentially at one point and transversely at two points. Figure 8 illustrates.

Figure 8. Ellipses that intersect in 3 points.

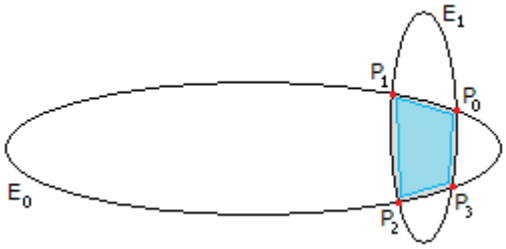


The point of tangency is irrelevant here. The region of intersection is still bounded by two elliptical arcs, one from each ellipse, so the algorithm for 2 intersection points still applies. In the figure, $\mathbf{P}_2$ is not required in the area calculation. However, it is required to know which 2 of the 3 intersections are the endpoints of the chord of interest.

5.5 Four Intersection Points

The two ellipses intersect transversely at four points. Figure 9 illustrates.

Figure 9. Ellipses that intersect in 4 points.



The region of intersection is the union of a convex quadrilateral and four chord regions, each region bounded by an elliptical arc and the line segment connecting the endpoints of the arc.

The key observations are (1) we need an ordering of the quadrilateral vertices for each ellipse [the code uses counterclockwise] and (2) to compute the areas is that we need to know which elliptical arc to use for an ordered chord. The ordering is obtained by computing polar coordinates for the points relative to the ellipse. The dot-perp concept used for the function Area2 allows us to identify the correct arc to use.

Pseudocode for the function that computes the area of intersection is shown in Listing 4.

Listing 4. Pseudocode for computing the area of intersection of ellipses when there are 4 points of intersection.

```
Real Area4(Ellipse E0, Ellipse E1, Vector2<Real> points[4])
{
    // Select a counterclockwise ordering of the points of intersection.  Use
    // the polar coordinates for E0 to do this.  The multimap is used in the
    // event that computing the intersections involved numerical rounding
    // errors that lead to a duplicate intersection, even though the
    // intersections are all labeled as transverse.  [See the comment in the
    // GTEngine function SpecialIntersection.]
    multimap<Real, int> ordering;
    int i;
    for (i = 0; i < 4; ++i)
    {
        Vector2<Real> PmC = points[i] - E0.center;
        Real x = Dot(E0.axis[0], PmC);
        Real y = Dot(E0.axis[1], PmC);
        Real theta = atan2(y, x);
        ordering.insert(make_pair(theta, i));
    }

    array<int, 4> permute;
    i = 0;
    for_each (element in ordering)
    {
        permute[i++] = element.second;
    }

    // Start with the area of the convex quadrilateral.
    Vector2<Real> diag20 = points[permute[2]] - points[permute[0]];
    Vector2<Real> diag31 = points[permute[3]] - points[permute[1]];
    Real area = fabs(DotPerp(diag20, diag31)) / 2;

    // Visit each pair of consecutive points.  The selection of ellipse for
    // the chord-region area calculation uses the "most counterclockwise"
    // tangent vector.
    for (int i0 = 3, i1 = 0; i1 < 4; i0 = i1++)
    {
        // Get a pair of consecutive points.
        Vector2<Real> P0 = points[permute[i0]];
        Vector2<Real> P1 = points[permute[i1]];

        // Compute locations relative to the ellipses.
        Vector2<Real> P0mC0 = P0 - E0.center, P0mC1 = P0 - E1.center;
        Vector2<Real> P1mC0 = P1 - E0.center, P1mC1 = P1 - E1.center;

        // Compute the ellipse normal vectors at endpoint P0.  The matrices
        // M are from the standard form, (X-C)^T * M * (X-C) = 1.
        Vector2<Real> N0 = E0.M * P0mC0, N1 = E1.M * P0mC1;
        Real dotperp = DotPerp(N1, N0);
        if (dotperp > 0)
        {
            // The chord goes with ellipse E0.
            area += ComputeAreaChordRegion(E0, P0mC0, P1mC0);
        }
        else
        {
            // The chord goes with ellipse E1.
            area += ComputeAreaChordRegion(E1, P0mC1, P1mC1);
        }
    }
    return area;
}
```

5.6 An Implementation

The general logic for the area computation is shown in Listing 5 and is based on the discussions in the previous subsections.

Listing 5. The top-level dispatch function for computing the area of intersection of two ellipses. The points of intersection are computed first, including whether each is transverse or tangential. The dispatcher calls the correct function for each geometrically distinct case.

```
Real AreaOfIntersection(Ellipse E0, Ellipse E1)
{
    // Each pair (points[i], isTransverse[i]) stores the point of intersection
    // and a Boolean valuethat is 'true' when the intersection is transverse
    // but 'false' when the intersection is tangential.
    Vector2<Real> points[4];
    bool isTransverse[4];
    int numPoints;
    FindIntersections(E0, E1, numPoints, points, isTransverse);

    if (numPoints > 0)
    {
        if (numPoints == 1)
        {
            // Containment or separation.
            return AreaCS(E0, E1, numPoints);
        }
        else if (numPoints == 2)
        {
            if (isTransverse[0])
            {
                // Both intersection points are transverse.
                return Area2(E0, E1, 0, 1, points);
            }
            else
            {
                // Both intersection points are tangential, so one ellipse
                // is contained in the other.
                return AreaCS(E0, E1, numPoints);
            }
        }
        else if (numPoints == 3)
        {
            // The tangential intersection is irrelevant in the area computation.
            if (!isTransverse[0])
            {
                return Area2(E0, E1, 1, 2, points);
            }
            else if (!isTransverse[1])
            {
                return Area2(E0, E1, 2, 0, points);
            }
            else  // isTransverse[2] == false
            {
                return Area2(E0, E1, 0, 1, points);
            }
        }
        else  // numPoints == 4
        {
            return Area4(E0, E1, points);
        }
    }
    else
    {
        // Containment, separation, or same ellipse.
        return AreaCS(E0, E1, numPoints);
    }
```

```
}
```
