Intersection of Box and Ellipsoid

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

Created: September 20, 2015

Contents

1 Introduction

This document describes how to determine whether a box and an ellipsoid overlap. This is a test-intersection query (do they overlap) and not a find-intersection query (where do they overlap). The motivation is provided by the 2D equivalent, Intersection of Rectangle and Ellipse.

The box is represented by a center $\mathbf{C}$, unit-length and orthogonal axes $\mathbf{U}_i$, and extents $e_i > 0$ for $i = 0, 1, 2$. The box is considered to be solid with points $\mathbf{X} = \mathbf{C} + \xi_0\mathbf{U}_0 + \xi_1\mathbf{U}_1 + \xi_2\mathbf{U}_2$, where $|\xi_0| \le e_0$, $|\xi_1| \le e_1$, and $|\xi_2| \le e_2$. In terms of vector-matrix algebra, $\mathbf{X} = \mathbf{C} + U\boldsymbol{\xi}$, where U is the 3×3 rotation matrix whose columns are the box axes and $\boldsymbol{\xi}$ is the 3×1 column vector whose rows are the coordinates ξ_0, ξ_1, and ξ_2. Define the eight vertices of the box to be

$$\mathbf{P}^{(i_0,i_1,i_2)} = \mathbf{C} + (2i_0 - 1)e_0\mathbf{U}_0 + (2i_1 - 1)e_1\mathbf{U}_1 + (2i_2 - 1)e_2\mathbf{U}_2$$

for i_0, i_1, and i_2 in $\{0, 1\}$.

The ellipsoid is represented by a center $\mathbf{K}$, unit-length and orthogonal axes $\mathbf{V}_i$, and axis half-lengths $a_i > 0$ for $i = 0, 1, 2$. The ellipsoid is considered to be solid with points $\mathbf{Y} = \mathbf{K} + \eta_0\mathbf{V}_0 + \eta_1\mathbf{V}_1 + \eta_2\mathbf{V}_2$, where$(\eta_0/a_0)^2 + (\eta_1/a_1)^2 + (\eta_2/a_2) = 1$. In terms of vector-matrix algebra, $\mathbf{Y} = \mathbf{K} + V\boldsymbol{\eta}$, where V is the 3×3 rotation matrix whose columns are the box axes and $\boldsymbol{\eta}$ is the 3×1 column vector whose rows are the coordinates η_0, η_1, and η_2. The ellipsoid is represented implicitly by

$$0 = Q(\mathbf{Y}) = (\mathbf{Y} - \mathbf{K})^{\mathsf{T}} M (\mathbf{Y} - \mathbf{K}) - 1 = \left|DV^{\mathsf{T}}(\mathbf{Y} - \mathbf{K})\right|^2 - 1$$

where $M = VD^2V^{\mathsf{T}}$ with $D = \text{Diagonal}(1/a_0, 1/a_1, 1/a_2)$. The solid ellipsoid (ellipsoid and region it contains) is $Q(\mathbf{Y}) \le 0$.

The data structures listed next are used in the pseudocode.

```
struct Box { Point3 center; Vector3 axis[3]; Real extent[3] };
struct Ellipsoid { Point3 center; Vector3 axis[3]; Real extent[3]; Matrix3x3 M; };
```

The matrix M is computed from the members axis[] and extent[].

2 All-Features Testing

This is a simple approach to test the box features for containment in the ellipsoid. If any box vertex is in the ellipsoid, then the box and ellipsoid overlap. If no vertex is in the ellipsoid, it is possible that a box edge intersects the ellipsoid. If no vertex is in the ellipsoid and no edge intersects the ellipsoid, it is possible that a box face intersects the ellipsoid. If no vertex, edge, or face intersects the ellipsoid, it is possible that the ellipsoid is strictly inside the box, which can be determined by testing whether the ellipsoid center is inside the box. The following discussion assumes that the tests are performed in the order mentioned, which simplifies the intersection and overlap tests.

If $\mathbf{P}$ is one of the vertices of the box, containment in the ellipsoid is characterized by $Q(\mathbf{P}) \le 0$.

If $\mathbf{P}_0$ and $\mathbf{P}_1$ are the endpoints of a box edge, the edge is represented by $\mathbf{E}(t) = \mathbf{P}_0 + (\mathbf{P}_1 - \mathbf{P}_0)t$ for $t \in [0, 1]$. When the endpoints are outside the ellipsoid, the segment-ellipsoid test-intersection query is the same as the

line-ellipsoid intersection. The line and ellipsoid intersect when the quadratic function

$$\begin{aligned} q(t) &= Q(\mathbf{E}(t)) \\ &= (\mathbf{P}_1 - \mathbf{P}_0)^{\mathsf{T}} M (\mathbf{P}_1 - \mathbf{P}_0) t^2 + 2 (\mathbf{P}_1 - \mathbf{P}_0)^{\mathsf{T}} M (\mathbf{P}_0 - \mathbf{K}) t + (\mathbf{P}_0 - \mathbf{K})^{\mathsf{T}} M (\mathbf{P}_0 - \mathbf{K}) - 1 \\ &= q_2 t^2 + 2 q_1 t + q_0 \end{aligned}$$

has real-valued roots. The last equality defines the coefficients q_i. The quadratic has real-valued roots when its discriminant is nonnegative: $q_1^2 - q_0 q_2 \geq 0$. It is possible to skip the vertex containment tests and use instead a modification of the edge-ellipsoid tests. If $q(t)$ has two real-valued roots (possibly equal), say, $t_0 \leq t_1$, then the edge intersects the solid ellipsoid when $[0,1] \cap [t_0, t_1] \neq \emptyset$ (the intersection of t-intervals is nonempty).

If $\mathbf{P}$ is a vertex of a box face with normal $\mathbf{U}$, we can can project points onto the normal line $\mathbf{K} + t\mathbf{U}$. The ellipsoid center projects to $t = 0$. The face point $\mathbf{P}$ projects to $\bar{t} = \mathbf{U} \cdot (\mathbf{P} - \mathbf{K})$. The projections of the extreme values of the ellipsoid in the direction of $\mathbf{U}$ have $t = \pm\sqrt{\mathbf{U}^{\mathsf{T}} M^{-1} \mathbf{U}}$. The plane of the box face intersects the solid ellipsoid when

$$|\bar{t}| \leq \sqrt{\mathbf{U}^{\mathsf{T}} M^{-1} \mathbf{U}} = \sqrt{\sum_{i=0}^{2} (a_i \mathbf{U} \cdot \mathbf{V}_i)^2}$$

When there is an intersection, the ellipse of intersection (possibly a degenerate point) and the face are either separated (no overlap) or the face fully contains the ellipse.

If none of the box faces overlap the solid ellipsoid, the test for the ellipsoid center contained in the box requires converting the ellipsoid center to box coordinates: $\boldsymbol{\xi} = U^{\mathsf{T}}(\mathbf{K} - \mathbf{C})$. The test for containment is $|\xi_0| \leq e_0$, $|\xi_1| \leq e_1$, and $|\xi_2| \leq e_2$.

```
bool EllipsoidContainsVertex (Ellipsoid E, Vector3 PmK)
{
    return Dot(PmK, E.M*PmK) <= 1;
}

// This function is called only when none of the box vertices are
// inside the ellipsoid.
bool EllipsoidOverlapsEdge (Ellipsoid E, Vector3 P0mK, Vector3 P1mK)
{
    Vector3 D = P1mK - P0mK, MP0mK = E.M*P0mK;
    Real q0 = Dot(P0mK, MP0mK) - 1, q1 = Dot(D, MP0mK), q2 = Dot(D, E.M*D);
    Real discr = q1*q1 - q0*q2;
    if (discr >= 0)
    {
        // The line containing P0 and P1 intersects the ellipsoid.  Determine
        // whether the segment connecting P0 and P1 intersects the ellipsoid.
        Real rootDiscr = sqrt(discr);
        Real t0 = (-q1 - rootDiscr) / q2, t1 = (-q1 + rootDiscr) / q2;
        return Overlaps([0,1], [t0,t1]);
    }
    else
    {
        return false;
    }
}

// This function is called only when none of the box edges intersect
// the ellipsoid.
bool EllipsoidOverlapsFace (Ellipsoid E, Vector3 V, Vector3 N)
{
```

```
    if (fabs(Dot(V, N)) <= sqrt(Dot(N, Inverse(E.M)*N)))
    {
        // The plane containing the box face intersects the ellipsoid.
        // Determine whether the face itself intersects the ellipsoid.
        // The ellipse of intersection (possibly degenerate point) and
        // face are either separated (no overlap) or the face fully
        // contains the ellipse.
        return Overlaps(face, ellipse);
    }
    else
    {
        return false;
    }
}

bool BoxContainsPoint (Box B, Vector3 V)
{
    return fabs(Dot(B.axis[0], V)) <= B.extent[0]
        && fabs(Dot(B.axis[1], V)) <= B.extent[1]
        && fabs(Dot(B.axis[2], V)) <= B.extent[2];
}

bool TestIntersectionAllFeatures (Box B, Ellipsoid E)
{
    // Translate so ellipsoid center is at origin.
    Vector3 CmK = B.center - E.center;

    Vector3 PmK[8];
    for (int i2 = 0, s2 = -1, j = 0; i2 < 2; ++i2, s2 += 2)
    {
        for (int i1 = 0, s1 = -1; i1 < 2; ++i1, s1 += 2)
        {
            for (int i0 = 0, s0 = -1; i0 < 2; ++i0, s0 += 2, ++j)
            {
                PmK[j] = CmK + s0*B.extent[i0]*B.axis[i0] + s1*B.extent[i1]*B.axis[i1] + s2*B.extent[i2]*B.axis[i2];
                if (EllipsoidContainsVertex(E, PmK[j])) { return true; }
            }
        }
    }

    int2 edge[12] = { (0,1), (1,3), (3,2), (2,0), (4,5), (5,7), (7,6), (6,4), (0,4), (1,5), (3,7), (2,6) };
    for (int j = 0; j < 12; ++j)
    {
        if (EllipsoidOverlapsEdge(E, PmK[edge[j][0]], PmK[edge[j][1]]) { return true; }
    }

    // Face overlap tests (point-index, axis-index).
    int2 face[6] = { (0,0), (1,0), (0,1), (2,1), (0,2), (4,2) };
    for (int j = 0; j < 6; ++j)
    {
        if (EllipsoidOverlapsFace(E, PmK[face[j][0]], B.axis[face[j][1]]) { return true; }
    }

    return BoxContainsPoint(B, -CmK);
}
```

3 Closest-Features Testing

In worst case, the all-features approach requires 8 vertex containment tests, 12 edge overlap tests, and 1 point-in-box test. This does not take into account closest-feature information, but a small modification does. First, test whether the ellipsoid center is in the box. If it is, the objects intersect. If it is not, then you will know the 1, 2, or 3 faces of the box that are visible to the ellipsoid center. In worst case, this limits you to 1 point-in-box test, 4 vertex containment tests, 3 edge overlap tests, and 3 face overlap tests.

```
bool TestOverlap1Face (Box B, Ellipsoid E, Vector3 CmK, int normal[1], int vertex[4])
{
    Vector3 V[4];
    for (int j = 0; j < 4; ++j)
    {
        int s0 = 2*(vertex[j] & 1) - 1, s1 = 2*(vertex[j] & 2) - 1, s2 = 2*(vertex[j] & 4) - 1;
        V[j] = CmK + s0*B.extent[0]*B.axis[0] + s1*B.extent[1]*B.axis[1] + s2*B.extent[2]*B.axis[2];
        if (EllipsoidContainsVertex(B, E, V[j])) { return true; }
    }

    int2 edge[4] = { (0,1), (1,2), (2,3), (3,0) };
    for (int j = 0; j < 4; ++j)
    {
        if (EllipsoidOverlapsEdge(E, V[edge[j][0]], V[edge[j][1]]) { return true; }
    }

    return EllipsoidOverlapsFace(E, V[0], B.axis[normal[0]]);
}

bool TestOverlap2Faces (Box B, Ellipsoid E, Vector3 CmK, int normal[2], int vertex[6])
{
    Vector3 V[6];
    for (int j = 0; j < 6; ++j)
    {
        int s0 = 2*(vertex[j] & 1) - 1, s1 = 2*(vertex[j] & 2) - 1, s2 = 2*(vertex[j] & 4) - 1;
        V[j] = CmK + s0*B.extent[0]*B.axis[0] + s1*B.extent[1]*B.axis[1] + s2*B.extent[2]*B.axis[2];
        if (EllipsoidContainsVertex(B, E, V[j])) { return true; }
    }

    int2 edge[7] = { (0,1), (1,2), (2,3), (3,4), (4,5), (5,0), (0,3) };
    for (int j = 0; j < 7; ++j)
    {
        if (EllipsoidOverlapsEdge(E, V[edge[j][0]], V[edge[j][1]]) { return true; }
    }

    return EllipsoidOverlapsFace(E, V[0], B.axis[normal[0]])
        || EllipsoidOverlapsFace(E, V[0], B.axis[normal[1]]);
}

bool TestOverlap3Faces (Box B, Ellipsoid E, Vector3 CmK, int normal[3], int vertex[7])
{
    Vector3 V[7];
    for (int j = 0; j < 7; ++j)
    {
        int s0 = 2*(vertex[j] & 1) - 1, s1 = 2*(vertex[j] & 2) - 1, s2 = 2*(vertex[j] & 4) - 1;
        V[j] = CmK + s0*B.extent[0]*B.axis[0] + s1*B.extent[1]*B.axis[1] + s2*B.extent[2]*B.axis[2];
        if (EllipsoidContainsVertex(B, E, V[j])) { return true; }
    }

    int2 edge[9] = { (1,2), (2,3), (3,4), (4,5), (4,5), (5,6), (0,1), (0,3), (0,5) };
    for (int j = 0; j < 9; ++j)
    {
        if (EllipsoidOverlapsEdge(E, V[edge[j][0]], V[edge[j][1]]) { return true; }
    }

    return EllipsoidOverlapsFace(E, V[0], B.axis[normal[0]])
        || EllipsoidOverlapsFace(E, V[0], B.axis[normal[1]]);
        || EllipsoidOverlapsFace(E, V[0], B.axis[normal[2]]);
}

bool ContainsPoint (Box, Ellipsoid, Vector3, int[], int[])
{
    return true;
}

TestOverlapFunction TestOverlap[4] = { ContainsPoint, TestOverlap1Face, TestOverlap2Faces, TestOverlap3Faces };
struct Query { int function; int normal[3], int vertex[7]; };
Query query[3][3][3] =
{
    {
        {
            {3, {0,1,2}, {0,1,3,2,6,4,5}},  // -e0 < x0 and -e1 < x1 and -e2 < x2
```

```
            {2, {0,1  }, {0,1,5,4,6,2  }},  // -e0 < x0 and -e1 < x1 and x2 <= e2
            {3, {0,1,2}, {4,5,1,0,2,6,7}},  // -e0 < x0 and -e1 < x1 and e2 < x2
        },
        {
            {2, {0,2  }, {0,1,3,2,6,4  }},  // -e0 < x0 and x1 <= e1 and -e2 < x2
            {1, {0    }, {0,2,6,4      }},  // -e0 < x0 and x1 <= e1 and x2 <= e2
            {2, {0,2  }, {6,2,0,4,5,7  }},  // -e0 < x0 and x1 <= e1 and e2 < x2
        },
        {
            {3, {0,1,2}, {2,6,7,3,1,0,4}},  // -e0 < x0 and e1 < x1 and -e2 < x2
            {2, {0,1  }, {2,0,4,6,7,3  }},  // -e0 < x0 and e1 < x1 and x2 <= e2
            {3, {0,1,2}, {6,7,5,4,0,2,3}},  // -e0 < x0 and e1 < x1 and e2 < x2
        }
    },
    {
        {
            {2, {1,2  }, {0,2,3,1,5,4  }},  // x0 <= e0 and -e1 < x1 and -e2 < x2
            {1, {1    }, {0,1,5,4      }},  // x0 <= e0 and -e1 < x1 and x2 <= e2
            {2, {1,2  }, {4,0,1,5,7,6  }},  // x0 <= e0 and -e1 < x1 and e2 < x2
        },
        {
            {1, {2    }, {0,1,3,2      }},  // x0 <= e0 and x1 <= e1 and -e2 < x2
            {0, {     }, {             }},  // x0 <= e0 and x1 <= e1 and x2 <= e2
            {1, {2    }, {4,5,7,6      }},  // x0 <= e0 and x1 <= e1 and e2 < x2
        },
        {
            {2, {1,2  }, {3,1,0,2,6,7  }},  // x0 <= e0 and e1 < x1 and -e2 < x2
            {1, {1    }, {3,2,6,7      }},  // x0 <= e0 and e1 < x1 and x2 <= e2
            {2, {1,2  }, {7,3,2,6,4,5  }},  // x0 <= e0 and e1 < x1 and e2 < x2
        }
    },
    {
        {
            {3, {0,1,2}, {1,3,7,5,4,0,2}},  // e0 < x0 and -e1 < x1 and -e2 < x2
            {2, {0,1  }, {5,4,0,1,3,7  }},  // e0 < x0 and -e1 < x1 and x2 <= e2
            {3, {0,1,2}, {5,7,6,4,0,1,3}},  // e0 < x0 and -e1 < x1 and e2 < x2
        },
        {
            {2, {0,2  }, {1,0,2,3,7,5  }},  // e0 < x0 and x1 <= e1 and -e2 < x2
            {1, {0    }, {1,3,7,5      }},  // e0 < x0 and x1 <= e1 and x2 <= e2
            {2, {0,2  }, {5,1,3,7,6,4  }},  // e0 < x0 and x1 <= e1 and e2 < x2
        },
        {
            {3, {0,1,2}, {3,2,0,1,5,7,6}},  // e0 < x0 and e1 < x1 and -e2 < x2
            {2, {0,1  }, {7,5,1,3,2,6  }},  // e0 < x0 and e1 < x1 and x2 <= e2
            {3, {0,1,2}, {7,6,4,5,1,3,2}},  // e0 < x0 and e1 < x1 and e2 < x2
        }
    }
};

bool TestIntersectionClosestFeatures (Box B, Ellipsoid E)
{
    // Transform the ellipsoid center to box coordinate system.
    Vector3 KmC = E.center - B.center;
    Real xi[3] = { Dot(B.axis[0], delta), Dot(B.axis[1], delta), Dot(B.axis[2], delta) };

    int select[3] =
    {
        (-R.extent[0] < xi[0] ? 0 (xi[0] <= R.extent[0] ? 1 : 2)),
        (-R.extent[1] < xi[1] ? 0 (xi[1] <= R.extent[1] ? 1 : 2)),
        (-R.extent[2] < xi[2] ? 0 (xi[2] <= R.extent[2] ? 1 : 2))
    };

    Query q = query[select[0]][select[1]][select[2]];
    return TestOverlap[q.function](B, E, -KmC, q.normal, q.s0, q.s1);
}
```

4 Conversion to Point-Parallelepiped Distance Query

The objects can be transformed so that the ellipsoid becomes a sphere and the box becomes a parallelepiped. The box and ellipsoid overlap if and only if the parallelepiped and sphere overlap. The latter query reduces to comparing the distance from sphere center to from with the sphere radius.

To compute the distance efficiently, we need to determine the 1, 2, or 3 faces of the parallelepiped that are closest to the sphere center. The analysis is similar to that in the pseudocode TestIntersectionSomeFeatures. In fact, the only difference between the two algorithms is that the current one involves distance, which uses the standard Cartesian metric. The previous algorithm is essentially the same algorithm but with the Euclidean metric imposed by the ellipsoid. It turns out that we may develop this algorithm without converting the ellipsoid to a sphere. But as was the case in Intersection of Rectangle and Ellipse, the implementation is equivalent to TestIntersectionSomeFeatures. I omit the details here, because there is no code of value from the approach.

5 Minkowski Sum of Box and Ellipsoid

A different approach involves shrinking the ellipsoid to a point and growing the box to a larger one with ellipsoidally rounded edges and corners. The latter object is the Minkowski sum of box and ellipsoid. The box and ellipsoid overlap when the ellipsoid center is in the Minkowski sum.

Translate the box center to the origin. This is equivalent to processing the box B with center $\mathbf{0}$ and the ellipsoid with center $\mathbf{K}-\mathbf{C}$. I will continue to refer to the ellipsoid center as $\mathbf{K}$. In this setting, the Minkowski sum has a bounding box B' that is centered at the origin and has the same axes as the original box. See Figure 5.1 of Intersection of Rectangle and Ellipse for the 2D equivalent. The edges of B grow to regions that are shaped by right elliptical cylinders. The vertices of B grow to regions that are shaped by ellipsoids.

We need to compute the points on the corner ellipsoids that support B'. Let $\mathbf{P}$ be a corner of B. The extreme points in a specified unit-length direction $\mathbf{N}$ on the ellipsoid defined by $Q(\mathbf{X}) = (\mathbf{X}-\mathbf{K})^{\mathsf{T}}M(\mathbf{X}-\mathbf{K})-1=0$ are points on the ellipsoid with normals parallel to $\mathbf{N}$ (two of them). Normal vectors to the ellipsoid are provided by the gradient, $\nabla Q = 2M(\mathbf{X}-\mathbf{K})$. The gradient is parallel to the specified direction when $M(\mathbf{X}-\mathbf{K}) = s\mathbf{N}$ for some scalar s. Thus, $\mathbf{X}-\mathbf{K} = sM^{-1}\mathbf{N}$. Substituting this into the quadratic equation, we can solve for $s = 1/\sqrt{\mathbf{N}^{\mathsf{T}}M^{-1}\mathbf{N}}$. The two extreme points are $\mathbf{X} = \mathbf{K} \pm M^{-1}\mathbf{N}/\sqrt{\mathbf{N}^{\mathsf{T}}M^{-1}\mathbf{N}}$. The distance from $\mathbf{K}$ to the extreme points measured along the line of direction $\mathbf{N}$ is $\ell = \sqrt{\mathbf{N}^{\mathsf{T}}M^{-1}\mathbf{N}}$.

In our application, the directions of interest are $\mathbf{U}_i$ for $i = 0, 1, 2$. The increases in the extents of the original box are $\ell_i = \sqrt{\mathbf{U}_i^{\mathsf{T}}M^{-1}\mathbf{U}_i}$ for $i = 0, 1, 2$; that is, the extents of B' are $e_i + \ell_i$ for $i = 0, 1, 2$.

If $\mathbf{K}$ is outside B', the box and ellipsoid do not overlap. If $\mathbf{K}$ is inside B', we need to determine whether $\mathbf{K}$ is outside the elliptical cylinders at the edges of B or the ellipsoids at the vertices of B. If so, the box and ellipsoid do not overlap. Sign testing will allows us to test $\mathbf{K}$ for exactly one vertex and three edges.

Let $\mathbf{P}$ be the vertex of B in the octant where all coordinates are nonnegative. Similar to Figure 5.1 of Intersection of Rectangle and Ellipse, there are three extreme points, $\mathbf{A}_i = \mathbf{P} + M^{-1}\mathbf{U}_i/\ell_i$, each an extreme in the $\mathbf{U}_i$ direction. Knowing that $\mathbf{K}$ is inside B', we may characterize the regions outside the Minkowski sum as follows. In the discussion, define $\mathbf{\Delta} = \mathbf{K} - \mathbf{P}$ and $r_i = \mathbf{V}_i \cdot \mathbf{\Delta}/a_i$. Also recall that $M = \sum_{i=0}^{2} \mathbf{V}_i\mathbf{V}_i^{\mathsf{T}}/a_i^2$.

Consider the ellipsoid centered at $\mathbf{P}$. We can represent $\mathbf{\Delta} = \sum_{i=0}^{2} z_i(\mathbf{A}_i - \mathbf{P})$ and solve for the coefficients by multiplying by M and using the definitions of the extreme points: $z_i = \ell_i \mathbf{U}_i^{\mathsf{T}} M\mathbf{\Delta}$. The point is outside the ellipsoid when $z_0 > 0$, $z_1 > 0$, $z_2 > 0$, and $\mathbf{\Delta}^{\mathsf{T}} M\mathbf{\Delta} > 1$. The first three sign tests are not affected by the ℓ_i, so an implementation does not have to formally multiply by them.

Consider the elliptical cylinder whose axis contains $\mathbf{P}$ and has direction $\mathbf{U}_2$. We can represent

$$\mathbf{\Delta} = z_0(\mathbf{A}_0 - \mathbf{P}) + z_1(\mathbf{A}_1 - \mathbf{P}) + w_2\mathbf{U}_2$$

and solve for the coefficients by multiplying by M and computing various dot products,

$$w_2 = \frac{\mathbf{U}_2^{\mathsf{T}} M\mathbf{\Delta}}{\mathbf{U}_2^{\mathsf{T}} M\mathbf{U}_2}, \quad z_0 = \ell_0 \left(\mathbf{U}_0^{\mathsf{T}} M\mathbf{\Delta} - w_2 \mathbf{U}_0^{\mathsf{T}} M\mathbf{U}_2\right), \quad z_1 = \ell_1 \left(\mathbf{U}_1^{\mathsf{T}} M\mathbf{\Delta} - w_2 \mathbf{U}_1^{\mathsf{T}} M\mathbf{U}_2\right)$$

The point is outside the elliptical cylinder when $z_0 > 0$, $z_1 > 0$, and $\mathbf{\Delta}^{\mathsf{T}}(M - \mathbf{V}_2\mathbf{V}_2^{\mathsf{T}}/a_2^2)\mathbf{\Delta} > 1$. The tests reduce to

$$-\mathbf{U}_1 \cdot M\mathbf{\Delta} \times M\mathbf{U}_2 = \mathbf{U}_0 \times \mathbf{U}_2 \cdot M\mathbf{\Delta} \times M\mathbf{U}_2 = \left(\mathbf{U}_0^{\mathsf{T}} M\mathbf{\Delta}\right)\left(\mathbf{U}_2^{\mathsf{T}} M\mathbf{U}_2\right) - \left(\mathbf{U}_0^{\mathsf{T}} M\mathbf{U}_2\right)\left(\mathbf{U}_2^{\mathsf{T}} M\mathbf{\Delta}\right) > 0$$
$$\mathbf{U}_0 \cdot M\mathbf{\Delta} \times M\mathbf{U}_2 = \mathbf{U}_1 \times \mathbf{U}_2 \cdot M\mathbf{\Delta} \times M\mathbf{U}_2 = \left(\mathbf{U}_1^{\mathsf{T}} M\mathbf{\Delta}\right)\left(\mathbf{U}_2^{\mathsf{T}} M\mathbf{U}_2\right) - \left(\mathbf{U}_1^{\mathsf{T}} M\mathbf{U}_2\right)\left(\mathbf{U}_2^{\mathsf{T}} M\mathbf{\Delta}\right) > 0$$
$$r_0^2 + r_1^2 > 1$$

Similarly, for the elliptical cylinder whose axis contains $\mathbf{P}$ and has direction $\mathbf{U}_1$, let

$$\mathbf{\Delta} = z_0(\mathbf{A}_0 - \mathbf{P}) + w_1\mathbf{U}_1 + z_2(\mathbf{A}_2 - \mathbf{P})$$

where

$$w_1 = \frac{\mathbf{U}_1^{\mathsf{T}} M\mathbf{\Delta}}{\mathbf{U}_1^{\mathsf{T}} M\mathbf{U}_1}, \quad z_0 = \ell_0 \left(\mathbf{U}_0^{\mathsf{T}} M\mathbf{\Delta} - w_1 \mathbf{U}_0^{\mathsf{T}} M\mathbf{U}_1\right), \quad z_2 = \ell_2 \left(\mathbf{U}_2^{\mathsf{T}} M\mathbf{\Delta} - w_1 \mathbf{U}_2^{\mathsf{T}} M\mathbf{U}_1\right)$$

The point is outside the elliptical cylinder when $z_0 > 0$, $z_2 > 0$, and $\mathbf{\Delta}^{\mathsf{T}}(M - \mathbf{V}_1\mathbf{V}_1^{\mathsf{T}}/a_1^2)\mathbf{\Delta} > 1$. The tests reduce to

$$\mathbf{U}_2 \cdot M\mathbf{\Delta} \times M\mathbf{U}_1 = \mathbf{U}_0 \times \mathbf{U}_1 \cdot M\mathbf{\Delta} \times M\mathbf{U}_1 = \left(\mathbf{U}_0^{\mathsf{T}} M\mathbf{\Delta}\right)\left(\mathbf{U}_1^{\mathsf{T}} M\mathbf{U}_1\right) - \left(\mathbf{U}_0^{\mathsf{T}} M\mathbf{U}_1\right)\left(\mathbf{U}_1^{\mathsf{T}} M\mathbf{\Delta}\right) > 0$$
$$-\mathbf{U}_0 \cdot M\mathbf{\Delta} \times M\mathbf{U}_1 = \mathbf{U}_2 \times \mathbf{U}_1 \cdot M\mathbf{\Delta} \times M\mathbf{U}_1 = \left(\mathbf{U}_2^{\mathsf{T}} M\mathbf{\Delta}\right)\left(\mathbf{U}_1^{\mathsf{T}} M\mathbf{U}_1\right) - \left(\mathbf{U}_2^{\mathsf{T}} M\mathbf{U}_1\right)\left(\mathbf{U}_1^{\mathsf{T}} M\mathbf{\Delta}\right) > 0$$
$$r_0^2 + r_2^2 > 1$$

Finally, for the elliptical cylinder whose axis contains $\mathbf{P}$ and has direction $\mathbf{U}_0$, let

$$\mathbf{\Delta} = w_0\mathbf{U}_1 + z_1(\mathbf{A}_1 - \mathbf{P}) + z_2(\mathbf{A}_2 - \mathbf{P})$$

where

$$w_0 = \frac{\mathbf{U}_0^{\mathsf{T}} M\mathbf{\Delta}}{\mathbf{U}_0^{\mathsf{T}} M\mathbf{U}_0}, \quad z_1 = \ell_1 \left(\mathbf{U}_1^{\mathsf{T}} M\mathbf{\Delta} - w_0 \mathbf{U}_1^{\mathsf{T}} M\mathbf{U}_0\right), \quad z_2 = \ell_2 \left(\mathbf{U}_2^{\mathsf{T}} M\mathbf{\Delta} - w_0 \mathbf{U}_2^{\mathsf{T}} M\mathbf{U}_0\right)$$

The point is outside the elliptical cylinder when $z_1 > 0$, $z_2 > 0$, and $\mathbf{\Delta}^{\mathsf{T}}(M - \mathbf{V}_0\mathbf{V}_0^{\mathsf{T}}/a_0^2)\mathbf{\Delta} > 1$. The tests reduce to

$$-\mathbf{U}_2 \cdot M\mathbf{\Delta} \times M\mathbf{U}_0 = \mathbf{U}_1 \times \mathbf{U}_0 \cdot M\mathbf{\Delta} \times M\mathbf{U}_0 = \left(\mathbf{U}_1^{\mathsf{T}} M\mathbf{\Delta}\right)\left(\mathbf{U}_0^{\mathsf{T}} M\mathbf{U}_0\right) - \left(\mathbf{U}_1^{\mathsf{T}} M\mathbf{U}_0\right)\left(\mathbf{U}_0^{\mathsf{T}} M\mathbf{\Delta}\right) > 0$$
$$\mathbf{U}_1 \cdot M\mathbf{\Delta} \times M\mathbf{U}_0 = \mathbf{U}_2 \times \mathbf{U}_0 \cdot M\mathbf{\Delta} \times M\mathbf{U}_0 = \left(\mathbf{U}_2^{\mathsf{T}} M\mathbf{\Delta}\right)\left(\mathbf{U}_0^{\mathsf{T}} M\mathbf{U}_0\right) - \left(\mathbf{U}_2^{\mathsf{T}} M\mathbf{U}_0\right)\left(\mathbf{U}_0^{\mathsf{T}} M\mathbf{\Delta}\right) > 0$$
$$r_1^2 + r_2^2 > 1$$

Pseudocode for the algorithm is listed next. In practice, the Ellipsoid data structure stores an orientation matrix V rather than M. The computation of ℓ_i becomes

$$\ell_i^2 = \mathbf{U}_i^\mathsf{T} M^{-1} \mathbf{U}_i = \mathbf{U}_i^\mathsf{T} V D^{-2} V^\mathsf{T} \mathbf{U}_i = (a_0 \mathbf{U}_i \cdot \mathbf{V}_0)^2 + (a_1 \mathbf{U}_i \cdot \mathbf{V}_1)^2 + (a_2 \mathbf{U}_i \cdot \mathbf{V}_2)^2$$

so you do not need to apply a general inversion algorithm to M.

```
bool TestIntersectionMinkowski (Box B, Ellipsoid E)
{
    // Compute the increase in extents for B'.
    Real L[3];
    for (int i = 0; i < 3; ++i)
    {
        sqrt(Dot(B.axis[i], Inverse(E.M)*B.axis[i])),
    };

    // Transform the ellipsoid center to box coordinate system.
    Vector3 KmC = E.center - B.center;
    Real xi[3] = { Dot(B.axis[0], KmC), Dot(B.axis[1], KmC), Dot(B.axis[2], KmC) };

    if (fabs(xi[0]) > B.extent[0] + L[0] || fabs(xi[1]) > B.extent[1] + L[1] || fabs(xi[2]) > B.extent[2] + L[2])
    {
        // K is outside the box B'.
        return false;
    }

    Real s[3] = { (xi[0] >= 0 ? 1 : -1), (xi[1] >= 0 ? 1 : -1), (xi[2] >= 0 ? 1 : -1) };
    Vector3 PmC = s[0]*B.extent[0]*B.axis[0] + s[1]*B.extent[1]*B.axis[1] + s[2]*B.extent[2]*B.axis[2];
    Vector3 Delta = KmC - PmC, MDelta = E.M*Delta;
    Real rsqr[3];
    for (int i = 0; i < 3; ++i)
    {
        Real r = Dot(E.axis[i], Delta)/E.extent[i];
        rsqr[i] = r*r;
    }
    Real UMD[3] = { Dot(B.axis[0], MDelta), Dot(B.axis[1], MDelta), Dot(B.axis[2], MDelta) };
    Matrix3x3 UMU;  // Only the 6 distinct entries are computed.
    for (int row = 0; row < 3; ++row)
    {
        Vector3 product = E.M*B.axis[row];
        for (int col = row; col < 3; ++col)
        {
            UMU[row][col] = Dot(B.axis[col], product);
        }
    }

    if (s[0]*(UMD[0]*UMU[2][2] - UMU[0][2]*UMD[2]) > 0
    &&  s[1]*(UMD[1]*UMU[2][2] - UMU[1][2]*UMD[2]) > 0
    &&  rsqr[0] + rsqr[1] > 1)
    {
        // K is outside the elliptical cylinder <P,U2>.
        return false;
    }

    if (s[0]*(UMD[0]*UMU[1][1] - UMU[0][1]*UMD[1]) > 0
    &&  s[2]*(UMD[2]*UMU[1][1] - UMU[1][2]*UMD[1]) > 0
    &&  rsqr[0] + rsqr[2] > 1)
    {
        // K is outside the elliptical cylinder <P,U1>.
        return false;
    }

    if (s[1]*(UMD[1]*UMU[0][0] - UMU[0][1]*UMD[0]) > 0
    &&  s[2]*(UMD[2]*UMU[0][0] - UMU[0][2]*UMD[0]) > 0
    &&  rsqr[1] + rsqr[2] > 1)
    {
        // K is outside the elliptical cylinder <P,U0>.
        return false;
    }

    if (s[0]*UMD[0] > 0 && s[1]*UMD[1] > 0 && s[2]*UMD[2] > 0 && rsqr[0] + rsqr[1] + rsqr[2] > 1)
    {
        // K is outside the ellipsoid at P.
        return false;
    }

    return true;
}
```