Intersection of an Oriented Box and a Cone

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

Created: January 4, 2015

Contents

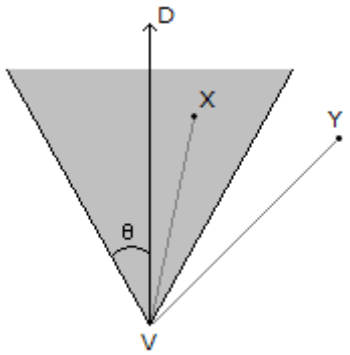
1 Introduction

A *infinite, single-sided, solid cone* has a vertex $\mathbf{V}$, an axis ray whose origin is $\mathbf{V}$ and unit-length direction is $\mathbf{D}$, and an acute cone angle $\theta \in (0, \pi/2)$. A point $\mathbf{X}$ is inside the cone when $\mathbf{X} = \mathbf{V}$ or when the angle between $\mathbf{D}$ and $\mathbf{X} - \mathbf{V}$ is in $[0, \theta]$. Algebraically, the containment is defined by

$$F(\mathbf{X}) = \mathbf{D} \cdot \frac{(\mathbf{X} - \mathbf{V})}{|\mathbf{X} - \mathbf{V}|} - \cos(\theta) \geq 0 \tag{1}$$

where the left-most equality defines the function F. The point is on the cone when it is the vertex or when the inequality of Equation (1) is replaced by an equality. Figure 1 shows a 2D cone, which is sufficient to illustrate the quantites in 3D.

Figure 1. A 2D view of a single-sided cone. $\mathbf{X}$ is inside the cone and $\mathbf{Y}$ is outside the cone.



Because of the constraint on θ, both $\cos(\theta)$ and $\sin(\theta)$ are positive. The single-sided cone is *finite* when you specify a maximum height h measured along the cone axis, in which case $\mathbf{D} \cdot (\mathbf{X} - \mathbf{V}) \leq h$. You may think of the infinite single-sided code having height $h = +\infty$. The *supporting plane* for the single-sided cone is $\mathbf{D} \cdot (\mathbf{X} - \mathbf{V}) = 0$. Any candidates for containment in the cone must satisfy the conservative condition $\mathbf{D} \cdot (\mathbf{X} - \mathbf{V}) \geq 0$.

An *oriented box* has a center $\mathbf{C}$, unit-length axis directions $\mathbf{U}_i$ that form a right-handed orthonormal set, and extents $e_i > 0$ that measure half the edge length in each dimension i with $0 \leq i \leq 2$. A point $\mathbf{P}$ in the box may be written in the form

$$\mathbf{P} = \mathbf{C} + \sum_{i=0}^{2} x_i \mathbf{U}_i \tag{2}$$

where $x_i = \mathbf{U}_i \cdot (\mathbf{P} - \mathbf{C})$ and $|x_i| \leq e_i$. In our implementation, the box corners are

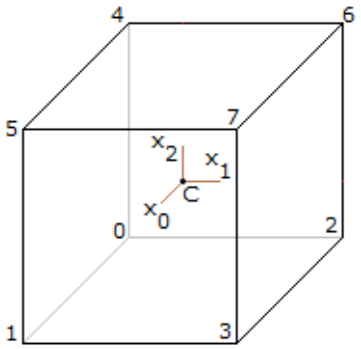
$$\mathbf{K}_j = \mathbf{C} + \sum_{i=0}^{2} \sigma_{ij} e_i \mathbf{U}_i \tag{3}$$

for $0 \leq j < 8$ and for

$$\sigma_{ij} = \begin{cases} +1, & j\&(1 \ll i) \neq 0 \\ -1, & j\&(1 \ll i) = 0 \end{cases} \tag{4}$$

That is, σ_{ij} is 1 when bit i of j is 1; it is -1 when bit i of j is 0. Figure 2 shows a 3D box in the coordinate system of the x_i.

Figure 2. A 3D box in the coordinate system of the x_i. The j-indices of the corners $\mathbf{K}_j$ are shown in the figure.



A set S is said to be *convex* if for any pair of points in S, the segment connecting them is in S. That is, if $\mathbf{X}_0 \in S$ and $\mathbf{X}_1 \in S$, then $(1-t)\mathbf{X}_0 + t\mathbf{X}_1 \in S$ for all $t \in [0,1]$. In 3-dimensional space, we can classify a convex object as *linear*, *planar*, or *volumetric*. The linear convex objects are lines, rays, and segments. A planar convex object lives in a plane and has intrinsic dimension 2; for example, a plane is convex (and infinite) and a triangle is a planar convex object. A volumetric object has intrinsic dimension 3; for example, a sphere, a cone, and an oriented box are all volumetric convex objects when viewed as solids.

2 Test-Intersection Query by Projection

We want to formulate *test-intersection* queries that determine whether a single-sided cone and a convex object overlap. The typical computer graphics application is lighting of objects by a spot light that is modeled as a cone. An infinite cone is used when there is no attenuation of the light; otherwise, a finite cone is used. Because the lighting calculations are expensive, a broad-phase culling is applied first to eliminate objects that are not visible to the spot light. Each object has an easy-to-update convex bounding volume. The broad-phase culling rejects objects whose bounding volumes are not intersected by the infinite cone.

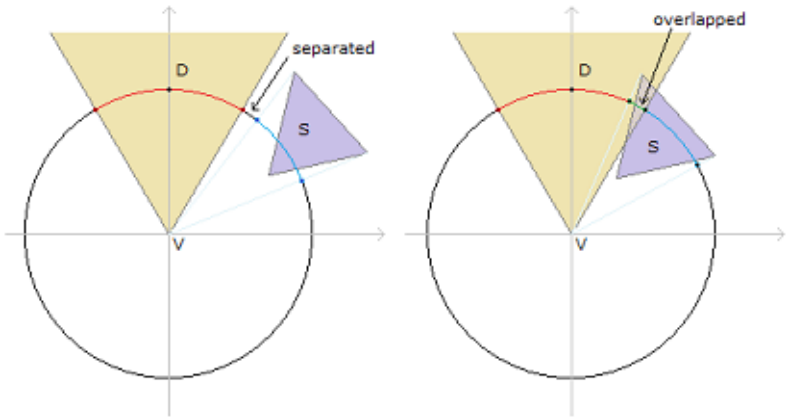
The abstract problem reduces to the following. The left-hand side of Equation (1) is a dot product of two unit-length vectors, which is $\cos(\phi)$ where ϕ is the angle between the two vectors. In the context of the cone definition, the comparison $\cos(\phi) \geq \cos(\theta) > 0$ is equivalent to $0 < \phi \leq \theta$. The normalization of $\mathbf{X} - \mathbf{V}$ is a projection of the difference onto a unit sphere. The problem is formulated in Euclidean geometry, but the projection converts the problem to spherical geometry. The dot product of two points on the unit sphere is the great-circle distance between the two points, which is the angle between the points.

If S is a convex set of points that represent the object, project those points onto the unit sphere whose center is $\mathbf{V}$. The solid cone itself projects onto a spherical disk whose spherical radius is θ. The cone and convex object intersect when the two projections overlap. To determine there is overlap, we need only find one point in the overlap set. Equivalently as a spherical distance problem, we need only find one point in the

projection of the convex object that is within a distance θ of the spherical point corresponding to the cone axis direction. This is a fancy way of stating that we need to find $\mathbf{X} \in S$ whose projection $(\mathbf{X}-\mathbf{V})/|\mathbf{X}-\mathbf{V}|$ forms an angle with $\mathbf{D}$ by an angle no larger than θ; however, the formulation as a projection allows us to derive an algorithm to locate the point $\mathbf{A}$ whose projection $(\mathbf{A}-\mathbf{V})/|\mathbf{A}-\mathbf{V}|$ is closest to $\mathbf{D}$. This amounts to computing $\mathbf{A}$ that maximizes the dot product of the projection with $\mathbf{D}$.

Figure 3 illustrates separation and overlap of the projections in 2D.

Figure 3. The projection of cone and convex object onto the unit circle centered at $\mathbf{V}$. The left image shows that the circular arcs of projection are separated, so the cone and convex object do not intersect. The right image shows that the circular arcs of projection are overlapped, so the cone and convex object do intersect.



An important observation is that the candidate point closest to $\mathbf{D}$ is generated by a point whose projection is on the boundary of the projected object. This allows us to be efficient and search only the projection boundary points for a point that is inside the cone.

3 Intersection with an Oriented Box

The point $\mathbf{P}$ that maximimizes F is either at a corner of the box or at an interior point of an edge of the box. If the maximum point is on an edge and $\mathbf{E}$ is the edge direction, then the point must solve $\mathbf{E} \cdot \nabla F(\mathbf{P}) = 0$.

Based on the projection idea, we need only analyze the boundary polygon of projection of the box. The polygon type is determined by the location of the cone vertex $\mathbf{V}$ relative to the box. The polygon is generated by 3 faces (a spherical hexagon), 2 faces (a spherical hexagon with two pair of cospherical arcs), or 1 face (a spherical rectangle). To simplify, we can represent the cone vertex in the box coordinate system,

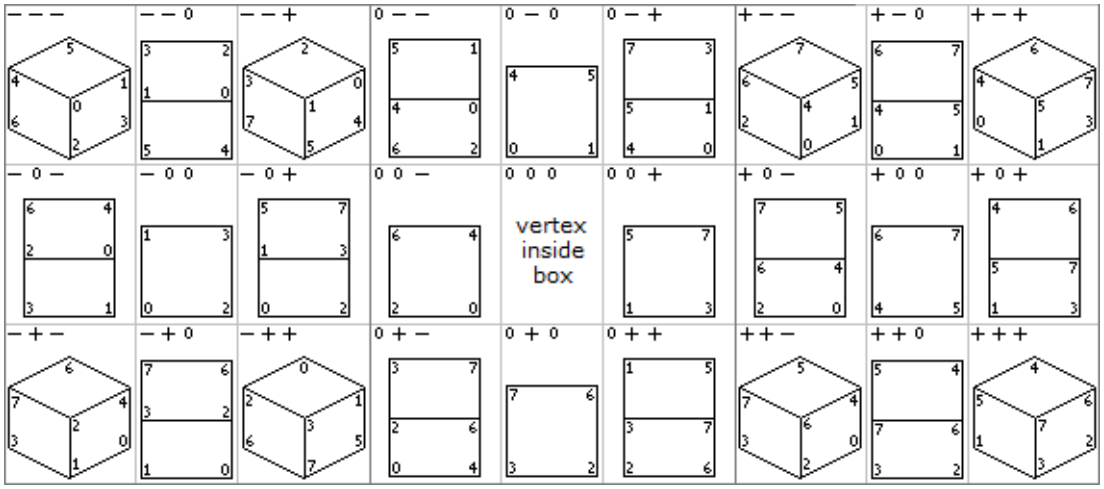
$$\mathbf{V} = \mathbf{C} + \sum_{i=0}^{2} z_i \mathbf{U}_i \tag{5}$$

where $z_i = \mathbf{U}_i \cdot (\mathbf{V} - \mathbf{C})$. The polygon type is determined based on whether $z_i < -e_i$, $|z_i| \le e_i$, or $z_i > e_i$.

For example, if $z_0 > e_0$, $z_1 > e_1$, and $z_2 > e_2$, the cone vertex can see the 3 faces that share the corner $\mathbf{K}_7$. If $z_0 > e_0$, $z_1 > e_1$, and $|z_2| \leq e_2$, the cone vertex can see the 2 faces that share the edge $\langle \mathbf{K}_3, \mathbf{K}_7 \rangle$. If $z_0 > e_0$, $|z_1| \leq e_1$, and $|z_2| \leq e_2$, the cone vertex can see the face $\langle \mathbf{K}_1, \mathbf{K}_3, \mathbf{K}_7, \mathbf{K}_5 \rangle$.

The implementation includes a lookup table to store the polygon indices into the box corner array in order to avoid computing the topology at run time. The indices are stored in counterclockwise order, which is useful to determine when the cone ray intersects a visible face. Figure 4 shows all the possibilities.

Figure 4. The lookup table for the spherical polygons of projection. The i-th sign corresponds to z_i and is $-$ when $z_i < -e_i$, $+$ when $z_i > e_i$, or 0 when $|z_i| \leq e_i$.



The order of geometric processing is arranged as follows, each test increasing in the computational time required for that stage.

1. Cull boxes below the supporting plane $\mathbf{D} \cdot (\mathbf{P} - \mathbf{V}) = 0$ of the cone. Boxes that just touch $\mathbf{V}$ are culled even though $\mathbf{V}$ is a point of intersection. If the cone has finite height, also cull boxes outside the plane $\mathbf{D} \cdot (\mathbf{P} - \mathbf{V}) = h$. Boxes that just touch the plane are culled even though there might be points of intersection.

2. Cull boxes that contain $\mathbf{V}$.

3. Report an intersection when a box corner corresponding to a spherical polygon vertex is inside the cone. Geometrically, that vertex is inside the spherical cap of projection of the cone onto the unit sphere centered at $\mathbf{V}$. Keep track of the corner that maximizes F for all processed corners.

4. Let $\mathbf{K}_{j_{\max}}$ be the corner that generates the maximum of F for all corners. This corner is shared by two box edges that project to spherical polygon edges. If one of the two edges sharing $\mathbf{K}_{j_{\max}}$ has an interior point for which the directional derivative of F has a local maximum, report an intersection when the interior point is inside the cone. If that point is not inside the cone, then the box and cone intersect if and only if $\mathbf{D}$ is inside the spherical polygon.

If neither edge sharing $\mathbf{K}_{j_{\max}}$ has a local maximum, then the box and cone do not intersect.

An implementation is in the files

GeometricTools/GTEngine/Include/GteIntrAlignedBox3Cone3.h
GeometricTools/GTEngine/Include/GteIntrOrientedBox3Cone3.h

The aligned-box query defines the polygon lookup table. The oriented-box query derives from the other query to share the lookup information. Steps 1 and 2 are implemented for each type of box, but after a polygon is looked up, the aligned-box has an internal query that is passed relevant information, thereby allowing the internal query to be shared by aligned and oriented boxes.

The implementation also stores computed values that are reused. It is also designed to avoid square roots and division in order to obtain high-performance execution. This design also supports exact arithmetic if you need more accuracy in exchange for computing time. The comparison $F(\mathbf{X}) \geq 0$ for $\mathbf{D} \cdot (\mathbf{X} - \mathbf{V}) > 0$ is reformulated as

$$(\mathbf{D} \cdot (\mathbf{X} - \mathbf{V}))^2 - \cos^2(\theta)|\mathbf{X} - \mathbf{V}|^2 > 0 \tag{6}$$

The comparisons for updating the maximum value of F at the corners is $F(\mathbf{K}_1) > F(\mathbf{K}_0)$ with $\mathbf{D}\cdot(\mathbf{K}_i-\mathbf{V}) \geq 0$ for both i. It is reformulated as

$$(\mathbf{D} \cdot (\mathbf{K}_1 - \mathbf{V}))^2|\mathbf{K}_0 - \mathbf{V}|^2 > (\mathbf{D} \cdot (\mathbf{K}_0 - \mathbf{V}))^2|\mathbf{K}_1 - \mathbf{V}|^2 \tag{7}$$

The computation of a local maximum of F on an edge produces a point of the form $\mathbf{M}$ for which one component requires a division. For example, $\mathbf{M} - \mathbf{V} = (n/d, y, z)$, where y and z are already computed numbers. It turns out that at the local maximum, it must be that $\mathbf{D} \cdot (\mathbf{M} - \mathbf{V}) \geq 0$. The reformulated sign test for $F(\mathbf{M})$ works whether we use $\mathbf{M} - \mathbf{V}$ or $d(\mathbf{M} - \mathbf{V})$.

3.1 Culling Against Planes

Step 1 involves projecting the box onto the line containing the cone ray. The parameterized line is $\mathbf{V} + t\mathbf{D}$ for real-valued t. The t-interval of projection is $[c - r, c + r]$ where

$$c = \mathbf{D} \cdot (\mathbf{C} - \mathbf{V}), \quad r = \sum_{i=0}^{2} e_i|\mathbf{D} \cdot \mathbf{U}_i| \tag{8}$$

The box is below the support plane when $c + r \leq 0$. When the cone has finite height h, the box is above the capping plane when $c - r \geq h$.

A convervative approach for culling may be used as an attempt to amortize the cost of culling, hoping to spend cycles up front in order to cull boxes quickly using other planes. The trade-off is that if additional tests do not cull a box, the worst-case behavior becomes more expensive. The additional planes are of the form $\mathbf{N} \cdot (\mathbf{X} - \mathbf{V}) = 0$, where $\mathbf{D} \cdot \mathbf{N} = \sin(\theta)$. The planes are tangent to the cone and contain the cone vertex $\mathbf{V}$. The box is projected onto the normal line $\mathbf{V} + s\mathbf{N}$. The s-interval of projection is $[c - r, c + r]$, where $c = \mathbf{N} \cdot (\mathbf{C} - \mathbf{V})$ and $r = \sum_{i=0}^{2} e_i|\mathbf{N} \cdot \mathbf{U}_i|$. The box is on the side of the plane opposite the cone whenever $c + r \leq 0$.

3.2 Cone-Vertex-in-Box Test and Polygon Lookup

The prelude to Step 2 is to compute $-z_i = \mathbf{U}_i \cdot (\mathbf{C} - \mathbf{V})$. The indexing scheme for the table lookup of Figure 4 is

```
int index[3];
for (int i = 0; i < 3; ++i)
{
    Real zMinus = Dot(U[i], C - V);
    if (zMinus < -e[i])
    {
        index[i] = 2;
    }
    else if (zMinus > e[i])
    {
        index[i] = 0;
    }
    else
    {
        index[i] = 1;
    }
}
int lookup = index[0] + 3 * index[1] + 9 * index[2];
if (lookup == 13)  // index[i] = 0 for all i
{
    // Cone vertex is inside box, so cone and box intersect.
}
else
{
    Polygon polygon = PolygonTable[lookup];  // Figure 3.1
    // Proceed to Step 3.
}
```

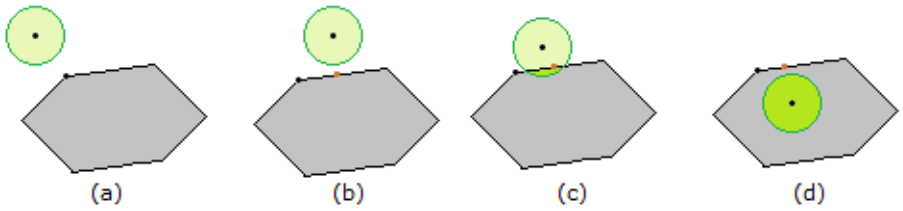
3.3 Box-Corner-in-Cone Test

In Step 3, we compute $F(\mathbf{K}_{i_j})$ for the spherical polygon vertices (all box corners), where the i_j are in a subset of $\{0, \ldots, 7\}$. As soon as we find one for which F is positive, we know that the cone and box intersect. Otherwise, all box corners are outside the cone. We keep track of the index $j_{\max}$ for which $F(\mathbf{K}_{j_{\max}})$ is the maximum of all F-values at the polygon vertices.

3.4 Box-Edge-Intersects-Cone Test

In Step 4, we know that $\mathbf{K}_{j_{\max}}$ is the box corner that forms the smallest dot product (largest F) with $\mathbf{D}$ among all corners. It is possible that an edge sharing that corner has a point with even smaller dot product (larger F). In this case, that point is either inside the cone (and we have a box-cone intersection) or outside (the box and cone are separated). Figure 5 illustrates the various configurations.

Figure 5. The green disk is the spherical disk of projection of the cone with center $\mathbf{D}$. The spherical polygon is gray. Its vertex corresponding to the box corner of maximum F is drawn as a black dot. The edge points that have larger F than the vertex are drawn as orange dots. (a) The box corner has maximum $F$and is outside the disk, so the box and cone do not intersect. (b) An edge point has maximum F and is outside the disk, so the box and cone do not intersect. (c) An edge point has maximum F and is inside the disk, so the box and cone intersect. (d) An edge point has F larger than that of the corner, is not inside the disk, but $\mathbf{D}$ is inside the polygon; the box and cone intersect.



Let $\mathbf{K}_{j_{\text{next}}}$ be the corner corresponding to the spherical polygon vertex that is the next counterclockwise vertex from $\mathbf{K}_{j_{\max}}$. By design we know that $F(\mathbf{K}_{j_{\text{next}}}) \le F(\mathbf{K}_{j_{\max}})$. If the derivative of F at $\mathbf{K}_{j_{\max}}$ and in the direction of the edge is positive, we are guaranteed the edge has an interior point at which F attains a local maximum; otherwise, if the derivative were to remain positive along the edge, F increases along the edge and the F-value at $\mathbf{K}_{j_{\text{next}}}$ would be forced to be larger than that of $\mathbf{K}_{j_{\max}}$, which is a contradiction. We must compute the point at which the local maximum occurs and evaluate F at it to determine whether the point is inside or outside the cone.

We need to compute the gradient of F, which is

$$\nabla F(\mathbf{X}) = \frac{|\mathbf{X}-\mathbf{V}|^2\,\mathbf{D} - (\mathbf{D}\cdot(\mathbf{X}-\mathbf{V}))(\mathbf{X}-\mathbf{V})}{|\mathbf{X}-\mathbf{V}|^3} \tag{9}$$

To illustrate, suppose the edge direction is $\mathbf{U}_0$ and

$$\mathbf{K}_{j_{\max}} - \mathbf{V} = \mathbf{C} - \mathbf{V} - e_0\mathbf{U}_0 + x_1\mathbf{U}_1 + x_2\mathbf{U}_2,\;\; \mathbf{K}_{j_{\text{next}}} - \mathbf{V} = \mathbf{C} - \mathbf{V} + e_0\mathbf{U}_0 + x_1\mathbf{U}_1 + x_2\mathbf{U}_2 \tag{10}$$

The directional derivative of F at the first corner is $\mathbf{U}_0 \cdot \nabla F(\mathbf{K}_{j_{\max}})$. We care only about its sign, so the directional derivative is positive at the corner when

$$|\mathbf{K}_{j_{\max}} - \mathbf{V}|^2\,(\mathbf{U}_0\cdot\mathbf{D}) - (\mathbf{D}\cdot(\mathbf{K}_{j_{\max}} - \mathbf{V}))(\mathbf{K}_{j_{\max}} - \mathbf{V}) > 0 \tag{11}$$

In the implementation, when we reach this test we will already have computed the vector difference, the squared length, and the dot products, so the cost is minimal—two multiplications and a subtraction.

When the directional derivative is positive, we need to compute the interior edge point $\mathbf{M}$ at which the derivative is zero. Let $\mathbf{C}-\mathbf{V} = \sum_{i=0}^{2} y_i\mathbf{U}_i$, so $y_i = -z_i$ (defined previously). Let $\mathbf{M} = \mathbf{C} + \sum_{i=0}^{2} x_i\mathbf{U}_i$ for some $x_0 \in (-e_0, e_0)$ and where x_1 and x_2 are the same values as those for $\mathbf{K}_{j_{\max}}$. In box-axis coordinates, $\mathbf{M}-\mathbf{V}$ is $(x_0+y_0, x_1+y_1, x_2+y_2)$. Define $d_i = \mathbf{U}_i\cdot\mathbf{D}$. Setting the numerator of the directional derivative to zero at $\mathbf{M}$,

$$d_0((x_0+y_0)^2 + (x_1+y_1)^2 + (x_2+y_2)^2) - (d_0(x_0+y_0) + d_1(x_1+y_1) + d_2(x_2+y_2))(x_0+y_0) = 0 \tag{12}$$

which has solution

$$x_0 + y_0 = \frac{d_0((x_1 + y_1)^2 + (x_2 + y_2)^2)}{d_1(x_1 + y_1) + d_2(x_2 + y_2)} \tag{13}$$

We do not need to solve explicitly for x_0, because the point-in-cone test involves manipulating $\mathbf{M} - \mathbf{V}$ whose components in box-axis coordinates are $x_i + y_i$.

The reformulated sign test for $F(\mathbf{M})$ depends on $\mathbf{D} \cdot (\mathbf{M} - \mathbf{V}) \geq 0$, which we do not know *a priori*. As it turns out, this is true. The numerator of the directional derivative is linear in x_0, is positive at $-e_0$, and the local maximum occurs when the derivative is zero. We also know that $\mathbf{D} \cdot (\mathbf{X} - \mathbf{V})$ is linear in x_0 and is positive at $-e_0$. This latter fact is known by the construction of $\mathbf{K}_{j_{\max}}$ occurring only for points above the supporting plane where $\mathbf{D} \cdot (\mathbf{X} - \mathbf{V}) \geq 0$. If there is a point $\mathbf{A}$ on the edge for which $\mathbf{D} \cdot (\mathbf{A} - \mathbf{V}) = 0$, the directional derivative of F at that point is

$$\mathbf{U}_0 \cdot \nabla F(\mathbf{A}) = (\mathbf{U}_0 \cdot \mathbf{D})/|\mathbf{A} - \mathbf{V}| \tag{14}$$

The directional derivative of $\mathbf{D} \cdot (\mathbf{X} - \mathbf{V})$ along the edge is $\mathbf{U}_0 \cdot \mathbf{D}$. We know that the dot product was positive at $\mathbf{K}_{j_{\max}}$ and decreased to zero at $\mathbf{A}$, which implies $\mathbf{U}_0 \cdot \mathbf{D} < 0$. Thus, the expression in Equation (14) is negative which in turn implies that $\mathbf{U}_0 \cdot \nabla F(\mathbf{X})$ must become zero *before* $\mathbf{D} \cdot (\mathbf{X} - \mathbf{V})$ becomes zero. That is, $\mathbf{U}_0 \cdot \nabla F(\mathbf{M}) = 0$ and $\mathbf{D} \cdot (\mathbf{M} - \mathbf{V}) > 0$.

We now know that we can safely test the sign of $F(\mathbf{M})$ using the reformulated test. If $F(\mathbf{M}) > 0$, we know $\mathbf{M}$ is inside the cone, so the box and cone intersect. If $F(\mathbf{M}) \leq 0$, it is still possible that the box and cone intersect. This hinges on whether $\mathbf{D}$ is inside or outside the spherical polygon; see Figure 5 (b)(d). It is sufficient to apply a visibility test that is common for point-in-polygon tests, although we require a 3D test for the point-in-spherical-polygon test. The point $\mathbf{D}$ is on the inside of the edge from $\mathbf{K}_{j_{\max}}$ to $\mathbf{K}_{j_{\text{next}}}$ when

$$\mathbf{D} \cdot (\mathbf{K}_{j_{\max}} - \mathbf{V}) \times (\mathbf{K}_{j_{\text{next}}} - \mathbf{V}) \leq 0 \tag{15}$$

which is effectively a test of sign of a determinant.

To illustrate, consider the example of Equation (10). The determinant test reduces to

$$\mathbf{D} \cdot (\mathbf{K}_{j_{\max}} - \mathbf{V}) \times (\mathbf{K}_{j_{\text{next}}} - \mathbf{V}) = 2e_0 \left((\mathbf{D} \cdot \mathbf{U}_1)(\mathbf{U}_2 \cdot (\mathbf{K}_{i_j} - \mathbf{V})) - (\mathbf{D} \cdot \mathbf{U}_2)(\mathbf{U}_1 \cdot (\mathbf{K}_{i_j} - \mathbf{V}))\right) \tag{16}$$

The four dot-product quantities in Equation (16) are computed in Step 3, so they are available for later use; in particular, the determinant sign test uses them and the test is relatively inexpensive. Similarly, for corners $\mathbf{K}_{j_{\max}} = (x_0, -e_1, x_2)$ and $\mathbf{K}_{j_{\text{next}}} = (x_0, +e_1, x_2)$ listed as box-axis coordinates,

$$\mathbf{D} \cdot (\mathbf{K}_{j_{\max}} - \mathbf{V}) \times (\mathbf{K}_{j_{\text{next}}} - \mathbf{V}) = 2e_1 \left((\mathbf{D} \cdot \mathbf{U}_2)(\mathbf{U}_0 \cdot (\mathbf{K}_{i_j} - \mathbf{V})) - (\mathbf{D} \cdot \mathbf{U}_0)(\mathbf{U}_2 \cdot (\mathbf{K}_{i_j} - \mathbf{V}))\right) \tag{17}$$

and for corners $\mathbf{K}_{j_{\max}} = (x_0, x_1, -e_2)$ and $\mathbf{K}_{j_{\text{next}}} = (x_0, x_1, +e_2)$ listed as box coordinates,

$$\mathbf{D} \cdot (\mathbf{K}_{j_{\max}} - \mathbf{V}) \times (\mathbf{K}_{j_{\text{next}}} - \mathbf{V}) = 2e_2 \left((\mathbf{D} \cdot \mathbf{U}_0)(\mathbf{U}_1 \cdot (\mathbf{K}_{i_j} - \mathbf{V})) - (\mathbf{D} \cdot \mathbf{U}_1)(\mathbf{U}_0 \cdot (\mathbf{K}_{i_j} - \mathbf{V}))\right) \tag{18}$$

The examples provided here are for an edge whose variable increases from $-e_i$ to $+e_i$. The polygon edges are listed in counterclockwise ordered when viewed from the cone vertex, so it is possible that the edge variable decreases from $+e_i$ to $-e_i$. In this case, we need to analyze various quantities multiplied by a negative sign. The box-corner indexing scheme allows us to determine from the polygon's edge indices which of Equations (16)-(18) to use and in which direction the edge is traversed. Define the index difference $\delta = i_{j+1} - i_j$. The direction of traversal is represented by $\sigma = \text{Sign}(\delta)$. The component index, the subscript of x_k, is $k = |\delta/2|$ (where the division by 2 is an integer division) because $|\delta| \in \{1, 2, 4\}$. For example, pseudocode for the determinant sign computation is

```
// Polygon edge indices are j0, j1.  The corners are K[j0] and K[j1].  The
// vector from cone vertex to corner K[m] is KmV[m] = K[m]-V.  The relevant
// dot products Dot(U[*],KmV[*]) are initialized in the corner-in-cone tests.
// The dot products Dot(D,U[*]) are initialized in the plane culling tests.
Real UdKmV[8], DdU[3];
int delta = j1 - j0;
int sigma = (delta > 0 ? 1 : -1);
int k0 = |delta| / 2, k1 = (k0 + 1) % 3, k2 = (k0 + 2) % 3;
Real det = sigma * (DdU[k1] * UdPmV[j0][k2] - DdU[k2] * UdPmV[j0][k1]);
```

For a large number of queries, the modulus operations for k1 and k2 can incur some cost, so in our implementation we use an array lookup to avoid the modulus,

```
int mod3[4] = { 0, 1, 2, 0 };
int k0 = number in { 0, 1, 2 };
int k1 = mod3[k0 + 1];  // (k0 + 1) % 3
int k2 = mod3[k1 + 1];  // (k1 + 1) % 3
```

If the directional derivative numerator of Equation (11) is instead nonpositive, we repeat this algorithm using the other edge whose opposite vertex corresponds to a corner $\mathbf{K}_{j\text{prev}}$.

If neither edge has a local maximum in F, we know that the box and cone are separated because $\mathbf{K}_{j_{\max}}$ is the box point that maximizes F and this point was already determined not to be inside the cone.