Intersection of a Sphere and a Cone

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

Created: March 8, 2002
Last Modified: March 2, 2008

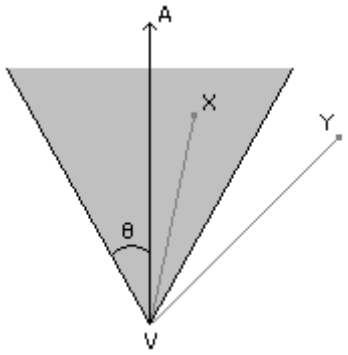
Contents

1 Discussion

A sphere with center $\mathbf{C}$ and radius $R > 0$ is defined by the set of points $\mathbf{X}$ satisfying $|\mathbf{X} - \mathbf{C}| = R$. The solid sphere is the sphere plus the region it bounds, specified as $|\mathbf{X} - \mathbf{C}| \le R$.

A single-sided cone with vertex $\mathbf{V}$, axis ray with origin at $\mathbf{V}$ and unit-length direction $\mathbf{A}$, and cone angle $\theta \in (0, \pi/2)$ is defined by the set of points $\mathbf{X}$ such that vector $\mathbf{X} - \mathbf{V}$ forms an angle θ with $\mathbf{A}$. The algebraic condition is $\mathbf{A} \cdot (\mathbf{X} - \mathbf{V}) = |\mathbf{X} - \mathbf{V}| \cos(\theta)$. The solid cone is the cone plus the region it bounds, specified as $\mathbf{A} \cdot (\mathbf{X} - \mathbf{V}) \ge |\mathbf{X} - \mathbf{V}| \cos(\theta)$. Figure 1.1 shows such a cone.

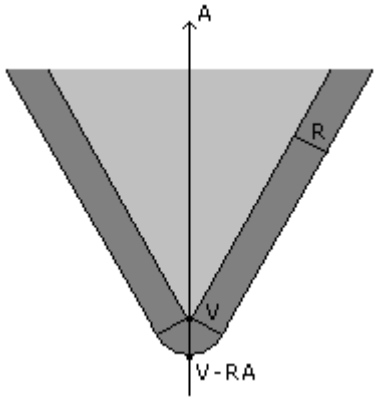
Figure 1.1 A 2D view of a single-sided code. $\mathbf{X}$ is inside the cone. $\mathbf{Y}$ is outside the cone.



Because of the constraint on θ, both $\cos(\theta) > 0$ and $\sin(\theta) > 0$.

The problem of determining if a sphere intersects a cone is equivalent to determining if $\mathbf{C}$ is inside a region formed by offsetting the cone boundary by a distance R. Figure 1.2 shows the offset region.

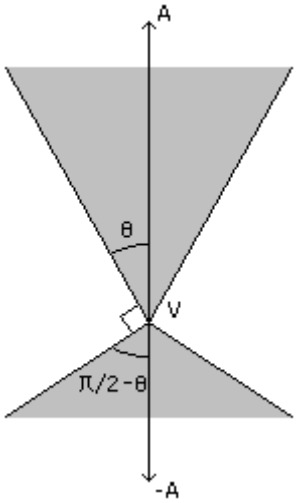
Figure 1.2 The cone of Figure 1.1 offset by a distance R.



The offset region is nearly a cone, but the bottom of the cone is a sector of a sphere. The sphere intersects the cone of Figure 1.1 if and only if **C** is inside either the light gray or dark gray region shown in Figure 1.2.

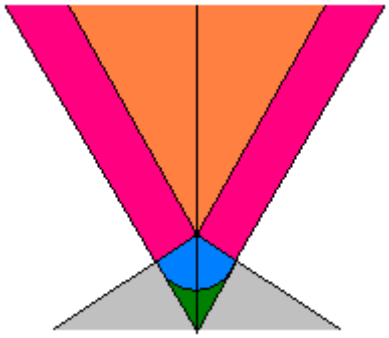
The complementary cone of the one in Figure 1.1 has the same vertex **V**, the axis direction $-\mathbf{A}$, and angle $\pi/2 - \theta$. Figure 1.3 shows the original cone and its complementary cone.

Figure 1.3 The complementary cone of Figure 1.1.



The solid complimentary cone is defined by $-\mathbf{A} \cdot (\mathbf{X} - \mathbf{V}) \geq |\mathbf{X} - \mathbf{V}| \sin(\theta)$. The cone containing the offset region of Figure 1.2 has the same axis direction and angle, but some trigonometry will show that its vertex is $\mathbf{U} = \mathbf{V} - (R/\sin(\theta))\mathbf{A}$. Figure 1.4 shows the three cones and the offset region.

Figure 1.4 The three cones and the offset region of the first cone.



The original solid cone K is the orange region and is defined by $\mathbf{A} \cdot (\mathbf{X} - \mathbf{V}) \geq |\mathbf{X} - \mathbf{V}| \cos(\theta)$. The solid

complementary cone K' is the union of the grey, blue, and green regions. It is defined by $-\mathbf{A} \cdot (\mathbf{X} - \mathbf{V}) \geq |\mathbf{X} - \mathbf{V}| \sin(\theta)$. The extended solid cone K'' containing the offset region for K is the union of the red, orange, blue, and green regions. It is defined by $\mathbf{A} \cdot (\mathbf{X} - \mathbf{U}) \geq |\mathbf{X} - \mathbf{U}| \cos(\theta)$. Any point $\mathbf{X}$ in the blue region satisfies $|\mathbf{X} - \mathbf{V}| \leq R$.

The conditions for when the sphere will or will not intersect K are as follows.

1. $\mathbf{C}$ is outside of K''. The center is in either the white or the gray regions in Figure 4. The sphere does not intersect K.
2. $\mathbf{C}$ is inside K'' and outside K'. The center is in either the red or the orange rgions in Figure 4. The sphere intersects K.
3. $\mathbf{C}$ is inside K'' and inside K'. The center is in either the blue or the green regions in Figure 4. The sphere intersects K if and only if $|\mathbf{C} - \mathbf{V}| <= R$.

The pseudocode is shown below. It is assumed that $\cos(\theta)$ and $\sin(\theta)$ have been precomputed for the cone.

```
bool SphereIntersectsCone (Sphere S, Cone K)
{
    U = K.vertex - (Sphere.radius/K.sin)*K.axis;
    D = S.center - U;
    if ( Dot(K.axis,D) >= Length(D)*K.cos )
    {
        // center is inside K''
        D = S.center - K.vertex;
        if ( -Dot(K.axis,D) >= Length(D)*K.sin )
        {
            // center is inside K'' and inside K'
            return Length(D) <= S.radius;
        }
        else
        {
            // center is inside K'' and outside K'
            return true;
        }
    }
    else
    {
        // center is outside K''
        return false;
    }
}
```

In a practical situation where you have to make many calls to this function, the square root calculations and the division by $\sin(\theta)$ are expensive. A typical situation is that the cone corresponds to a spot light and a sphere is a bounding volume for an object. If the sphere intersects the cone, then the object must be lit by the renderer. The pseudocode can be rewritten to avoid the expensive calculations.

First, let us remove the square root calls. The idea is that we have tests of the form $\alpha \geq \beta$ where $\beta > 0$ and requires a square root calculation. The test $\alpha \geq \beta > 0$ is equivalent to: $\alpha > 0$ and $\alpha^2 \geq \beta^2$.

```
bool SphereIntersectsCone (Sphere S, Cone K)
{
    U = K.vertex - (Sphere.radius/K.sin)*K.axis;
    D = S.center - U;
    dsqr = Dot(D,D)
    e = Dot(K.axis,D);
    if ( e > 0 and e*e >= dsqr*(K.cos*K.cos) )
    {
        D = S.center - K.vertex;
        dsqr = Dot(D,D);
        e = -Dot(K.axis,D);
        if ( e > 0 and e*e >= dsqr*(K.sin*K.sin) )
            return dsqr <= S.radius*S.radius;
        else
            return true;
    }
    return false;
}
```

Second, the simplest way to remove the division by $\sin(\theta)$ is to just precompute the reciprocal for the cone. We can also precompute the squares of the sine and cosine terms. We can also precompute the squared radius for the sphere.

```
bool SphereIntersectsCone (Sphere S, Cone K)
{
    U = K.vertex - (Sphere.radius*K.sinReciprocal)*K.axis;
    D = S.center - U;
    dsqr = Dot(D,D);
    e = Dot(K.axis,D);
    if ( e > 0 and e*e >= dsqr*K.cosSqr )
    {
        D = S.center - K.vertex;
        dsqr = Dot(D,D);
        e = -Dot(K.axis,D);
        if ( e > 0 and e*e >= dsqr*K.sinSqr )
            return dsqr <= S.radiusSqr;
        else
            return true;
    }
    return false;
}
```