Intersection of Moving Sphere and Box

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

Created: October 3, 2018

Contents

1 Introduction

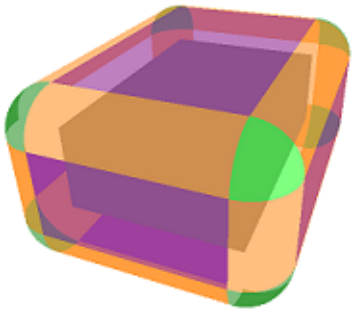
This document shows how to compute whether a sphere and box intersect, both moving with constant velocities. Both objects are considered to be solids; that is, the sphere object is the region bounded by the sphere and the box object is the region bounded by the box. Throughout the document I will continue to use the terms sphere and box for these objects. Before reading the remainder of this document, I recommend that you read Intersection of Moving Circle and Rectangle which is the 2-dimensional equivalent for intersections of circles and rectangles, both moving with constant velocities. This will give you a flavor of the ideas that apply in the 3-dimensional case but are easier to visualize in 2 dimensions.

It is possible that the sphere and box initially overlap in the sense that the common set of points is nonempty. If that set contains a single point, then the sphere and box are in contact at time zero. If they are not initially overlapping and if they will intersect at a later time, the first time of contact and the corresponding point of contact are computed. The remaining possibility is that the objects will not intersect at any time.

The sphere has center $\mathbf{C}_{\text{sph}}$, radius r and moves with constant velocity $\mathbf{V}_{\text{sph}}$. The box has center $\mathbf{C}_{\text{box}}$, unit-length axis directions $\mathbf{U}_0$, $\mathbf{U}_1$ and $\mathbf{U}_2$ that are perpendicular and extents e_0, e_1 and e_2 in those directions. The eight corners of the box are $\mathbf{C}_{\text{box}} \pm e_0\mathbf{U}_0 \pm e_1\mathbf{U}_1 \pm e_2\mathbf{U}_2$. The box moves with constant velocity $\mathbf{V}_{\text{box}}$. The problem can be converted to a canonical form for the determination of first contact time and point, if they exist. Consider motion relative to the box: the box is stationary (zero velocity) and the sphere moves with velocity $\mathbf{V}_{\text{rel}} = \mathbf{V}_{\text{sph}} - \mathbf{V}_{\text{box}}$. Now transform the box to an axis-aligned box centered at the origin and having axis directions $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$. Define $R = [\mathbf{U}_0\,\mathbf{U}_1\,\mathbf{U}_2]$, which is a rotation matrix with the specified columns. The transformation of the box to an axis-aligned box causes the sphere center to be transformed to $\mathbf{C} = R^{\mathsf{T}}(\mathbf{C}_{\text{sph}} - \mathbf{C}_{\text{box}})$. The sphere velocity is transformed to $\mathbf{V} = R^{\mathsf{T}}\mathbf{V}_{\text{rel}} = R^{\mathsf{T}}(\mathbf{V}_{\text{sph}} - \mathbf{V}_{\text{box}})$.

When the sphere and box are initially separated but will later intersect at a first contact time, they do so at a single point that is either on a box face, a box edge or at a box corner. To obtain a better geometric flavor of the problem, we can consider the equivalent problem involving Minkowski sums: the sphere is reduced to a point (its center) and the box is expanded by placing spheres of radii r at each point of the box and computing the union. Figure 1 shows the resulting expansion which is a box with rounded corners. I will refer to the solid region as the rounded box.

Figure 1. The original box is dark gray and lives inside the rounded box. The 6 faces of the rounded box are offset by r units from the original box faces, drawn in magenta. The 12 edges of the box are the axes for finite cylinders of radius r, drawn in orange. The 8 corners of the box are the origins for spheres of radius r, drawn in green.



The center travels along the ray $\mathbf{C}+t\mathbf{V}$. If this ray intersects the rounded box at a first contact time $T \geq 0$, then the sphere and box intersect at that time at the first contact point. The point $\mathbf{C}+T\mathbf{V}$ is the center of the circle at first contact, and the first contact point can be computed from it. The details of this are provided in the remainder of the document.

As in the 2-dimensional case of circle-rectangle intersections, the problem will be reduced to a canonical form where $\mathbf{C}$ occurs in the first octant. The octant is decomposed into 19 regions, each having its own logic for the sphere-box intersection query. Table 1 lists those regions.

Table 1. The canonical regions in the first octant for the sphere-box intersection queries. The vertex in the octant is $\mathbf{K} = (e_0, e_1, e_2)$ and the sphere center is $\mathbf{C} = (c_0, c_1, c_2) \geq (0, 0, 0)$. The difference is $\mathbf{\Delta} = \mathbf{C} - \mathbf{K} = (\Delta_0, \Delta_1, \Delta_2)$. The regions are listed in the order of occurrence in the pseudocode of Listing 2.

region	definition	description
R_0	$\Delta_0 \leq 0$, $\Delta_1 \leq 0$, $\Delta_2 \leq 0$	interior overlap
R_1	$0 < \Delta_0 \leq r$, $\Delta_1 \leq 0$, $\Delta_2 \leq 0$	x-face overlap
R_2	$\Delta_0 \leq 0$, $0 < \Delta_1 \leq r$, $\Delta_2 \leq 0$	y-face overlap
R_3	$0 < \Delta_0 \leq r$, $0 < \Delta_1 \leq r$, $\Delta_2 \leq 0$, $\Delta_0^2 + \Delta_1^2 \leq r^2$	xy-edge overlap
R_4	$0 < \Delta_0 \leq r$, $0 < \Delta_1 \leq r$, $\Delta_2 \leq 0$, $\Delta_0^2 + \Delta_1^2 > r^2$	xy-edge separated
R_5	$\Delta_0 \leq 0$, $\Delta_1 \leq 0$, $0 < \Delta_2 \leq r$	z-face overlap
R_6	$0 < \Delta_0 \leq r$, $\Delta_1 \leq 0$, $0 < \Delta_2 \leq r$, $\Delta_0^2 + \Delta_2^2 \leq r^2$	xz-edge overlap
R_7	$0 < \Delta_0 \leq r$, $\Delta_1 \leq 0$, $0 < \Delta_2 \leq r$, $\Delta_0^2 + \Delta_2^2 > r^2$	xz-edge separated
R_8	$\Delta_0 \leq 0$, $0 < \Delta_1 \leq r$, $0 < \Delta_2 \leq r$, $\Delta_1^2 + \Delta_2^2 \leq r^2$	yz-edge overlap
R_9	$\Delta_0 \leq 0$, $0 < \Delta_1 \leq r$, $0 < \Delta_2 \leq r$, $\Delta_1^2 + \Delta_2^2 > r^2$	yz-edge separated
R_{10}	$0 < \Delta_0 \leq r$, $0 < \Delta_1 \leq r$, $0 < \Delta_2 \leq r$, $\Delta_0^2 + \Delta_1^2 + \Delta_2^2 \leq r^2$	xyz-vertex overlap
R_{11}	$0 < \Delta_0 \leq r$, $0 < \Delta_1 \leq r$, $0 < \Delta_2 \leq r$, $\Delta_0^2 + \Delta_1^2 + \Delta_2^2 > r^2$	xyz-vertex separated
R_{12}	$r < \Delta_0$, $\Delta_1 \leq r$, $\Delta_2 \leq r$	x-face unbounded
R_{13}	$\Delta_0 \leq r$, $r < \Delta_1$, $\Delta_2 \leq r$	y-face unbounded
R_{14}	$r < \Delta_0$, $r < \Delta_1$, $\Delta_2 \leq r$	xy-edge unbounded
R_{15}	$\Delta_0 \leq r$, $\Delta_1 \leq r$, $r < \Delta_2$	z-face unbounded
R_{16}	$r < \Delta_0$, $\Delta_1 \leq r$, $r < \Delta_2$	xz-edge unbounded
R_{17}	$\Delta_0 \leq r$, $r < \Delta_1$, $r < \Delta_2$	yz-edge unbounded
R_{18}	$r < \Delta_0$, $r < \Delta_1$, $r < \Delta_2$	xyz-vertex unbounded

2 Reduction to a Canonical Form

After the transformation that maps the box to an axis-aligned box with center at the origin, we can perform an additional transformation that maps **C** to the first octant and maps **V** accordingly. Specifically, any component of **C** that is negative can be negated and the corresponding component of **V** is negated. I will use the same names for center and velocity from this point on. Listing 1 contains pseudocode for converting the sphere and box to the canonical representation that allows intersection analysis in the first octant.

Listing 1. Conversion of the sphere and box to canonical form, analysis for intersections and then conversion back to the original coordinate system.

```
struct Sphere { Vector3 center; Real radius; };
struct Box { Vector3 center; Vector3 axis[3]; Real extent[3]; };

// The returned integer is -1 if the objects initially overlap, 0 if the
// objects are initially separated but do not intersect at a later time,
// or +1 if the objects are initially separated but do intersect at a later
// time.  In the last case, firstTime and firstContact are valid quantities.
int GetFirstTimeAndContact(Sphere sphere, Vector3 sphereVelocity, Box box, Vector3 boxVelocity,
    Real& firstTime, Vector3& firstContact)
{
    // Transform the sphere and velocity to be in the coordinate system of the
    // aligned box centered at the origin.
    Vector3 cdiff = sphere.center - box.center;
    Vector3 vdiff = sphereVelocity - boxVelocity;
    Vector3 C, V;
    for (int i = 0; i < 3; ++i)
    {
        C[i] = Dot(cdiff, box.axis[i]);
        V[i] = Dot(vdiff, box.axis[i]);
    }

    // Reflect components, if necessary, to transform C to the first octant.
    // Adjust the velocity accordingly.
    int sign[3];
    for (int i = 0; i < 3; ++i)
    {
        if (C[i] >= 0)
        {
            sign[i] = 1;
        }
        else
        {
            C[i] = -C[i];
            V[i] = -V[i];
            sign[i] = -1;
        }
    }

    int intersectionType = DoQuery(box.extent, C, sphere.radius, V, firstTime, firstContact);

    if (intersectionType != 0)
    {
        // Translate back to the original coordinate system.
        for (int i = 0; i < 3; ++i)
        {
            firstContact[i] = box.center[i] + sign[i] * firstContact[i] * box.axis[i];
        }
    }
    return intersectionType;
}
```

After the canonical transformation, $\mathbf{C}$ is in one of the 19 regions listed in Table 1.

The regions whose description includes the word *overlap* are inside the rounded box; these are $\mathcal{R}_i$ for $i \in \{0, 1, 2, 3, 5, 6, 8, 10\}$. If $\mathbf{C}$ is in one of these regions, the sphere and box are initially overlapping. Generally, one expects that the intersection of sphere and box is a set of positive volume, although there is a small probability that the sphere and box are touching at a single point. Regardless, in both cases the implementation reports that the contact time is $T = 0$. The partitioning into the overlap regions allows us to select a point in the intersection, even if it is not considered a contact point.

If the initial center is outside the rounded box, the sphere might or might not intersect the box depending on the sphere velocity. The analysis is based on visibility of the rounded box boundary components from $\mathbf{C}$. For the entire rounded box, 6 of these components are rectangles, 12 of these components are line segments and 8 of these components are octants of spheres centered at the corners. The latter components are referred to as *corner sectors*.

In the next two paragraphs, whenever a rounded-box feature is identified, the sphere-box intersection algorithm involves computing the intersection of the ray $\mathbf{C} + t\mathbf{V}$ for $t \geq 0$ with those features.

The regions whose description includes the word *separated* are outside the rounded box but inside the smallest aligned box containing the rounded box. The vertex in the first octant for this bounding box is $(e_0 + r, e_1 + r, e_2 + r)$. The regions are $\mathcal{R}_i$ for $i \in \{4, 7, 9, 11\}$. The visibility information is as follows:

- If $\mathbf{C} \in \mathcal{R}_4$, the xy-edge of the original box is visible to $\mathbf{C}$. The quarter cylinder for that edge and the 2 corner sectors for the endpoints of the edge are visible to $\mathbf{C}$.
- If $\mathbf{C} \in \mathcal{R}_7$, the xz-edge of the original box is visible to $\mathbf{C}$. The quarter cylinder for that edge and the 2 corner sectors for the endpoints of the edge are visible to $\mathbf{C}$.
- If $\mathbf{C} \in \mathcal{R}_9$, the yz-edge of the original box is visible to $\mathbf{C}$. The quarter cylinder for that edge and the 2 corner sectors for the endpoints of the edge are visible to $\mathbf{C}$.
- If $\mathbf{C} \in \mathcal{R}_{11}$, the xyz-vertex ($\mathbf{K}$) of the original box is visible to $\mathbf{C}$. The corner sector for that vertex is visible to $\mathbf{C}$.

The regions whose description includes the word *unbounded* are outside the smallest aligned box containing the rounded box; these are $\mathcal{R}_i$ for $12 \leq i \leq 18$. The visibility information is as follows:

- If $\mathbf{C} \in \mathcal{R}_{12}$, the x-face of the original box is visible to $\mathbf{C}$. The x-face of the rounded box; the 4 quarter cylinders for the y- and z-edges of the x-face; and the 4 corner sectors for $(e_0, \pm e_1, \pm e_2)$ are visible to $\mathbf{C}$.
- If $\mathbf{C} \in \mathcal{R}_{13}$, the y-face of the original box is visible to $\mathbf{C}$. The y-face of the rounded box; the 4 quarter cylinders for the x- and z-edges of the y-face; and the 4 corner sectors for $(\pm e_0, e_1, \pm e_2)$ are visible to $\mathbf{C}$.
- If $\mathbf{C} \in \mathcal{R}_{15}$, the z-face of the original box is visible to $\mathbf{C}$. The z-face of the rounded box; the 4 quarter cylinders for the x- and y-edges of the z-face; and the 4 corner sectors for $(\pm e_0, \pm e_1, e_2)$ are visible to $\mathbf{C}$.
- If $\mathbf{C} \in \mathcal{R}_{14}$, the x- and y-faces and the xy-edge of the original box are visible to $\mathbf{C}$. The x- and y-faces of the rounded box; the 7 quarter cylinders for the visible edges; and the 6 corner sectors for the visible vertices are visible to $\mathbf{C}$.

- If $\mathbf{C} \in \mathcal{R}_{16}$, the x- and z-faces and the xz-edge of the original box are visible to $\mathbf{C}$. The x- and z-faces of the rounded box; the 7 quarter cylinders for the visible edges; and the 6 corner sectors for the visible vertices are visible to $\mathbf{C}$.
- If $\mathbf{C} \in \mathcal{R}_{17}$, the y- and z-faces and the yz-edge of the original box are visible to $\mathbf{C}$. The y- and z-faces of the rounded box; the 7 quarter cylinders for the visible edges; and the 6 corner sectors for the visible vertices are visible to $\mathbf{C}$.
- If $\mathbf{C} \in \mathcal{R}_{18}$, the x-, y- and z-faces and the xy-, xz- and yz-edges of the original box are visible to $\mathbf{C}$. The x-, y- and z-faces of the rounded box; the 9 quarter cylinders for the visible edges; and the 7 corner sectors for the visible vertices are visible to $\mathbf{C}$.

3 Intersection Analysis

I assume that the objects have been transformed to canonical form for intersection analysis in the first octant, as described in the previous section. The center $\mathbf{C}$ is in the first octant. Listing 2 has pseudocode for determining which of the regions $\mathcal{R}_i$ contains $\mathbf{C}$. Define $\mathbf{\Delta} = \mathbf{C} - (e_0, e_1, e_2)$.

Listing 2. Pseudocode for determining which region $\mathcal{R}_i$ contains $\mathbf{C}$.

```
int DoQuery(Vector3 extent, Vector2 3, Real r, Vector3 V, Real& firstTime, Real& firstContact)
{
    Vector3 delta = C - extent;
    if (delta[2] <= radius)
    {
        if (delta[1] <= r)
        {
            if (delta[0] <= r)
            {
                if (delta[2] <= 0)
                {
                    if (delta[1] <= 0)
                    {
                        if (delta[0] <= 0)
                        {
                            // R0: interior overlap
                        }
                        else
                        {
                            // R1: x-face overlap
                        }
                    }
                    else
                    {
                        if (delta[0] <= 0)
                        {
                            // R2: y-face overlap
                        }
                        else
                        {
                            if (delta[0] * delta[0] + delta[1] * delta[1] <= r * r)
                            {
                                // R3: xy-edge overlap
                            }
                            else
                            {
                                // R4: xy-edge separated
                            }
                        }
```

```
                }
            }
            else
            {
                if (delta[1] <= 0)
                {
                    if (delta[0] <= 0)
                    {
                        // R5: z-face overlap
                    }
                    else
                    {
                        if (delta[0] * delta[0] + delta[2] * delta[2] <= r * r)
                        {
                            // R6: xz-edge overlap
                        }
                        else
                        {
                            // R7: xz-edge separated
                        }
                    }
                }
                else
                {
                    if (delta[0] <= 0)
                    {
                        if (delta[1] * delta[1] + delta[2] * delta[2] <= r * r)
                        {
                            // R8: yz-edge overlap
                        }
                        else
                        {
                            // R9: yz-edge separated
                        }
                    }
                    else
                    {
                        if (Dot(delta, delta) <= r * r)
                        {
                            // R10: xyz-vertex overlap
                        }
                        else
                        {
                            // R11: xyz-vertex separated
                        }
                    }
                }
            }
        }
        else
        {
            // R12: x-face unbounded
        }
    }
    else
    {
        if (delta[0] <= r)
        {
            // R13: y-face unbounded
        }
        else
        {
            // R14: xy-edge unbounded
        }
    }
}
else
{
    if (delta[1] <= r)
    {
        if (delta[0] <= r)
        {
```

```
                // R15: z-face unbounded
            }
            else
            {
                // R16: xz-edge unbounded
            }
        }
        else
        {
            if (delta[0] <= r)
            {
                // R17: yz-edge unbounded
            }
            else
            {
                // R18: xyz-vertex unbounded
            }
        }
    }
}
```

3.1 Interior-Overlap Region

The sphere center is inside the original box. The intersection type is set to -1. The overlap has positive volume, so there is no first contact point. However, we choose $\mathbf{C}$ as one of the intersection points and return it as the contact point. The contact time is 0. Listing 3 contains pseudocode for this case.

Listing 3. Intersection analysis when $\mathbf{C} \in \mathcal{R}_0$.

```
int InteriorOverlap(Vector3 C, Real& contactTime, Vector3& contactPoint)
{
    contactTime = 0;
    contactPoint = C;
    return -1;
}
```

3.2 Vertex-Overlap Region

The sphere intersects the vertex $\mathbf{K}$ of the original box. The reported contact time is 0 and the reported contact point is (e_0, e_1, e_2). If $\mathbf{C}$ is exactly r units from the vertex, the intersection type is set to $+1$; otherwise, the distance from $\mathbf{C}$ to the vertex is smaller than r and the intersection type is set to -1. Listing 4 contains pseudocode for this case.

Listing 4. Intersection analysis when $\mathbf{C} \in \mathcal{R}_{10}$.

```
int VertexOverlap(Vector3 K, Real radius, Vector3 delta, Real& contactTime, Vector3& contactPoint)
{
    contactTime = 0;
    contactPoint = K;
    return (Dot(delta, delta) < radius * radius ? -1 : 1);
}
```

3.3 Edge-Overlap Regions

The sphere intersects an edge of the original box in at least one point. The reported contact time is 0 and the reported contact point is (e_0, e_1, c_2) (xy-edge), (e_0, c_1, e_2) (xz-edge) or (c_0, e_1, e_2) (yz-edge). If $\mathbf{C}$ is exactly r units from the edge, the intersection type is set to $+1$; otherwise, the distance from $\mathbf{C}$ to the edge is smaller than r and the intersection type is set to -1. Listing 5 contains pseudocode for this case.

Listing 5. Intersection analysis when $\mathbf{C}$ is in $\mathcal{R}_1$, $\mathcal{R}_2$ or $\mathcal{R}_5$. The input is (i_0, i_1, i_2) is $(0, 1, 2)$ for xy-edge overlap, $(2, 0, 1)$ for xz-edge overlap or $(1, 2, 0)$ for yz-edge overlap.

```
int EdgeOverlap(int i0, int i1, int i2, Vector3 K, Vector3 C, Real radius, Vector3 delta,
    Real& contactTime, Vector3& contactPoint)
{
    contactTime = 0;
    contactPoint[i0] = K[i0];
    contactPoint[i1] = K[i1];
    contactPoint[i2] = C[i2];
    return (delta[i0] * delta[i0] + delta[i1] * delta[i1] < radius * radius ? -1 : 1);
}
```

3.4 Face-Overlap Regions

The sphere intersects a face of the original box in at least one point. The reported contact time is 0 and the reported contact point is (e_0, c_1, c_2) (x-face), (c_0, e_1, c_2) (y-face) or (c_0, c_1, e_2) (z-face). If $\mathbf{C}$ is exactly r units from the face, the intersection type is set to $+1$; otherwise, the distance from $\mathbf{C}$ to the face is smaller than r and the intersection type is set to -1. Listing 6 contains pseudocode for this case.

Listing 6. Intersection analysis when $\mathbf{C}$ is in $\mathcal{R}_1$, $\mathcal{R}_2$ or $\mathcal{R}_5$. The input (i_0, i_1, i_2) is $(0, 1, 2)$ for x-face overlap, $(1, 2, 0)$ for y-face overlap or $(2, 0, 1)$ for z-face overlap.

```
int FaceOverlap(int i0, int i1, int i2, Vector3 K, Vector3 C, Real radius, Vector3 delta,
    Real& contactTime, Vector3& contactPoint)
{
    contactTime = 0;
    contactPoint[i0] = K[i0];
    contactPoint[i1] = C[i1];
    contactPoint[i2] = C[i2];
    return (delta[i0] < radius ? -1 : 1);
}
```

3.5 Vertex-Separated Region

The sphere and box are initially separated. The only rounded-box feature visible to $\mathbf{C}$ is the corner sector at $\mathbf{K}$. The equation for the sphere at that corner is $|\mathbf{P}-\mathbf{K}|^2 = r^2$. The set of velocity vectors $\mathbf{V}$ for which the ray $\mathbf{P}(t) = \mathbf{C} + t\mathbf{V}$ intersects the corner sector is obtained by substituting the ray into the sphere equation,

$$a_2 t^2 + 2a_1 t + a_0 = |\mathbf{V}|^2 t^2 + 2(\mathbf{V}\cdot\boldsymbol{\Delta})t + |\boldsymbol{\Delta}|^2 - r^2 = 0 \tag{1}$$

where $\boldsymbol{\Delta} = \mathbf{C} - \mathbf{K}$ and where the first equality defines the coefficients a_0, a_1 and a_2. We know that $a_0 > 0$ by definition of the vertex-separated region. If $a_2 = 0$, then $a_1 = 0$ is implied, in which case the quadratic equation reduces to $a_0 = 0$, which is not possible (no real-valued roots). Otherwise, $a_2 > 0$ and the smallest real-valued root is

$$T = \frac{-a_1 - \sqrt{a_1^2 - a_2 a_0}}{a_2} \tag{2}$$

For the ray to intersect the sphere at a time $T > 0$, it is necessary that

$$0 < a_2 = |\mathbf{V}|^2,\ \ 0 > a_1 = \mathbf{V}\cdot\boldsymbol{\Delta},\ \ 0 \le a_1^2 - a_2 a_0 = (\mathbf{V}\cdot\boldsymbol{\Delta})^2 - |\mathbf{V}|^2\left(|\boldsymbol{\Delta}|^2 - r^2\right) = r^2|\mathbf{V}|^2 - |\mathbf{V}\times\boldsymbol{\Delta}|^2 \tag{3}$$

When there is an intersection, the contact point is $\mathbf{K}$. Let $\mathbf{V} = (\nu_0, \nu_1, \nu_2)$. An early-exit test is that $\nu_0 \ge 0$ and $\nu_1 \ge 0$ and $\nu_2 \ge 0$, in which case the sphere is stationary or moving away from the box. Listing 7 contains pseudocode for this case.

Listing 7. Intersection analysis when $\mathbf{C} \in \mathcal{R}_{11}$.

```
int DoQueryRayRoundedVertex(Vector3 K, Real radius, Vector3 delta, Vector3 V,
    Real contactTime, Vector3 contactPoint)
{
    Real a1 = Dot(V, delta);
    if (a1 < 0)
    {
        // The preconditions are a0 > 0 and a2 > 0.
        Real a0 = Dot(delta, delta) - radius * radius;
        Real a2 = Dot(V, V);
        Real adiscr = a1 * a1 - a2 * a0;
        if (adiscr >= 0)
        {
            // The ray intersects the rounded vertex, so the sphere-box
            // contact point is the vertex.
            contactTime = -(a1 + sqrt(adiscr)) / a2;
            contactPoint = K;
            return 1;
        }
    }
    return 0;
}

int VertexSeparated(Vector3 K, Real radius, Vector3 delta, Vector3 V,
    Real& contactTime, Real& contactPoint)
{
    if (V[0] < 0 || V[1] < 0 || V[2] < 0)
    {
        if (DoQueryRayRoundedVertex(K, radius, delta, V, contactTime, contactPoint) == 1)
        {
            return 1;
        }
```

```
    }
    contactTime = 0;  // invalid
    contactPoint = Vector3(0, 0, 0);  // invalid
    return 0;
}
```

3.6 Edge-Separated Regions

The sphere and box are initially separated. A *rounded-box edge* consists of a quarter of a finite cylinder and two octants of a sphere; it is visible to $\mathbf{C}$. Let $\mathbf{K} = (e_0, e_1, e_2)$ and $\mathbf{\Delta} = \mathbf{C} - \mathbf{K} = (\Delta_0, \Delta_1, \Delta_2)$. Let (i_0, i_1, i_2) be a permutation of $(0, 1, 2)$ for which $0 < \Delta_{i_0} \le r$, $0 < \Delta_{i_1} \le r$, $\Delta_{i_2} \le 0$ and $\Delta_{i_0}^2 + \Delta_{i_1}^2 > r^2$. The permutation is $(0, 1, 2)$ for the xy-edge, $(2, 0, 1)$ for the xz-edge and $(1, 2, 0)$ for the yz-edge.

The finite cylinder is defined by $(p_{i_0} - e_{i_0})^2 + (p_{i_1} - e_{i_1})^2 = r^2$ with $p_{i_0} \ge e_{i_0}$, $p_{i_1} \ge e_{i_1}$ and $|p_{i_2}| \le e_{i_2}$. The octant of a sphere for $\mathbf{K}_0 = (e_{i_0}, e_{i_1}, e_{i_2})$ [permuted coordinates] is defined by $(p_{i_0} - e_{i_0})^2 + (p_{i_1} - e_{i_1})^2 + (p_{i_2} - e_{i_2})^2 = r^2$ with $p_{i_0} \ge e_{i_0}$, $p_{i_1} \ge e_{i_1}$ and $p_{i_2} \ge e_{i_2}$. The octant of a sphere for $\mathbf{K}_1 = (e_{i_0}, e_{i_1}, -e_{i_2})$ [permuted coordinates] is defined by $(p_{i_0} - e_{i_0})^2 + (p_{i_1} - e_{i_1})^2 + (p_{i_2} + e_{i_2})^2 = r^2$ with $p_{i_0} \ge e_{i_0}$, $p_{i_1} \ge e_{i_1}$ and $p_{i_2} \le -e_{i_2}$.

The set of velocity vectors $\mathbf{V}$ for which the ray $\mathbf{P}(t) = \mathbf{C} + t\mathbf{V}$ intersects the infinite cylinder coincident with the finite cylinder is obtained by substituting the ray into the cylinder equation,

$$b_2 t^2 + 2b_1 t + b_0 = \left(\nu_{i_0}^2 + \nu_{i_1}^2\right) t^2 + 2(\nu_{i_0}\Delta_{i_0} + \nu_{i_1}\Delta_{i_1})t + \left(\Delta_{i_0}^2 + \Delta_{i_1}^2 - r^2\right) = 0 \tag{4}$$

where $\mathbf{V} = (\nu_0, \nu_1, \nu_2)$, $\mathbf{\Delta} = \mathbf{C} - \mathbf{K}$ and where the first equality defines the coefficients b_0, b_1 and b_2. We know that $b_0 > 0$ by definition of the edge-separated region. If $b_2 = 0$, then $b_1 = 0$ is implied, in which case the quadratic equation reduces to $b_0 = 0$ which is not possible (no real-valued roots). Otherwise, $b_2 > 0$ and the smallest real-valued root is

$$T = \frac{-b_1 - \sqrt{b_1^2 - b_2 b_0}}{b_2} \tag{5}$$

Define $\mathbf{W} = (\nu_{i_0}, \nu_{i_1})$ and $\mathbf{D} = (\Delta_{i_0}, \Delta_{i_1})$. For the ray to intersect the infinite cylinder at a time $T > 0$, it is necessary that

$$0 < b_2 = |\mathbf{W}|^2, \;\; 0 > b_1 = \mathbf{W}\cdot\mathbf{D}, \;\; 0 \le b_1^2 - b_2 b_0 = (\mathbf{W}\cdot\mathbf{D})^2 - |\mathbf{W}|^2\left(|\mathbf{D}|^2 - r^2\right) = r^2|\mathbf{W}|^2 - \left(\mathbf{W} \times \mathbf{D}^{\perp}\right)^2 \tag{6}$$

If $b_1^2 - b_2 b_0 < 0$, the ray does not intersect the infinite cylinder, which also means it cannot intersect either corner sector. Under this condition, the sphere does not intersect the box.

When $b_1^2 - b_2 b_0 \ge 0$, we have an additional constraint that the cylinder is finite:

$$-e_{i_2} \le c_{i_2} + T\nu_{i_2} \le e_{i_2} \tag{7}$$

If T of equation (5) satisfies these inequalities, then the sphere will intersect the box at time T and the contact point is $(e_{i_0}, e_{i_1}, c_{i_2} + T\nu_{i_2})$. If T does not satisfy the inequalities, we must test whether the ray intersects one of the corner sectors.

Suppose that $c_{i_2} + T\nu_{i_2} > e_{i_2}$. We need to test the ray for intersection with the sector corresponding to $\mathbf{K}_0$. It turns out that the logic for the vertex-separated region can be used for the testing. A similar argument

applies if instead $c_{i_2} + T\nu_{i_2} < -e_{i_2}$. We need to test the ray for intersection with the sector corresponding to $\mathbf{K}_1$. The logic for the vertex-separated region can be used for the testing. The cylinder test must be performed before the sphere tests, because we can then guarantee that the ray-sphere test does not report intersection points that are on the sphere but hidden by the finite cylinder.

Listing 8 contains pseudocode for this case.

Listing 8. Intersection analysis when $\mathbf{C}$ is in $\mathcal{R}_4$, $\mathcal{R}_7$ or $\mathcal{R}_9$.

```
int DoQueryRayRoundedEdge(int i0, int i1, int i2, Vector3 K, Vector3 C, Real radius,
    Vector3 delta, Vector3 V, Real& contactTime, Vector3& contactPoint)
{
    Real b1 = V[i0] * delta[i0] + V[i1] * delta[i1];
    if (b1 < 0)
    {
        // The preconditions are b0 > 0 and b2 > 0.
        Real b0 = delta[i0] * delta[i0] + delta[i1] * delta[i1] - radius * radius;
        Real b2 = V[i0] * V[i0] + V[i1] * V[i1];
        Real bdiscr = b1 * b1 - b2 * b0;
        if (bdiscr >= 0)
        {
            Real T = -(b1 + sqrt(bdiscr)) / b2;
            Real p2 = C[i2] + T * V[i2];
            if (-K[i2] <= p2)
            {
                if (p2 <= K[i2])
                {
                    // The ray intersects the finite cylinder of the rounded edge, so the
                    // sphere-box contact point is on the corresponding box edge.
                    contactTime = T;
                    contactPoint[i0] = K[i0];
                    contactPoint[i1] = K[i1];
                    contactPoint[i2] = p2;
                    return 1;
                }
                else
                {
                    // The ray intersects the infinite cylinder but not the finite cylinder of the
                    // rounded edge.  It is possible the ray intersects the rounded vertex for K.
                    return QueryRayRoundedVertex(K, radius, delta, V, contactTime, contactPoint);
                }
            }
            else
            {
                // The ray intersects the infinite cylinder but not the finite cylinder of the
                // rounded edge.  It is possible the ray intersects the rounded vertex for K1.
                Vector3 otherK, otherDelta;
                otherK[i0] = K[i0];
                otherK[i1] = K[i1];
                otherK[i2] = -K[i2];
                otherDelta[i0] = C[i0] - otherK[i0];
                otherDelta[i1] = C[i1] - otherK[i1];
                otherDelta[i2] = C[i2] - otherK[i2];
                return QueryRayRoundedVertex(otherK, radius, otherDelta, V, contactTime, contactPoint);
            }
        }
    }
    return 0;
}

int EdgeSeparated(int i0, int i1, int i2, Vector3 K, Vector3 C, Real radius, Vector3 delta,
    Vector3 V, Real& contactTime, Vector3& contactPoint)
{
    if (V[i0] < 0 || V[i1] < 0)
    {
        if (DoQueryRayRoundedEdge(i0, i1, i2, K, C, radius, delta, V, contactTIme, contactPoint) == 1)
        {
```

```
            return 1;
        }
    }
    return 0;
}
```

3.7 Unbounded Regions

When the initial sphere center is in one of the unbounded regions, either 1, 2 or 3 rounded box faces are visible to the center. The number is 1 when $\mathbf{C}$ is in $\mathcal{R}_{12}$ (x-face), $\mathcal{R}_{13}$ (y-face) or $\mathcal{R}_{14}$ (z-face). The number is 2 when $\mathbf{C}$ is in $\mathcal{R}_{14}$ (x-face and y-face), $\mathcal{R}_{16}$ (x-face and z-face) or $\mathcal{R}_{17}$ (y-face and z-face). The number is 3 when $\mathbf{C}$ is in $\mathcal{R}_{18}$ (x-face and y-face and z-face).

The (transformed) aligned box is centered at the origin and has extents (e_0, e_1, e_2). The rounded box contains the aligned box. The smallest aligned box containing the rounded box is centered at the origin and has extents $(e_0 + r, e_1 + r, e_2 + r)$; I will call this the *super box*. If the ray $\mathbf{C} + t\mathbf{V}$ intersects the super box, it must do so at a point on one of the visible faces. The corresponding rounded box face is the candidate for ray-roundedbox intersection.

Let $(p_{i_0}, p_{i_1}, p_{i_2})$ be a permuted tuple for $\mathbf{P} = (p_0, p_1, p_2)$, the point of intersection of the ray with the super box plane $x_{i_0} = e_{i_0} + r$. Figure 2 shows the potential features on the rounded box for intersection with the ray.

Figure 2. The partitioning of the super box face $x_{i_0} = e_{i_0} + r$ to determine which rounded box features to test for intersection with the ray $\mathbf{C} + t\mathbf{V}$.

	$p_{i_1} < -e_{i_1}$	$\lvert p_{i_1}\rvert \le e_{i_1}$	$p_{i_1} > e_{i_1}$
$p_{i_2} < -e_{i_2}$	potential i_2i_0-edge intersect potential i_0i_1-edge intersect	potential i_2i_0-edge intersect	potential i_2i_0-edge intersect potential i_0i_1-edge intersect
$\lvert p_{i_2}\rvert \le e_{i_2}$	potential i_2i_0-edge intersect	certain i_0-face intersect	potential i_2i_0-edge intersect
$p_{i_2} > e_{i_2}$	potential i_2i_0-edge intersect potential i_0i_1-edge intersect	potential i_2i_0-edge intersect	potential i_2i_0-edge intersect potential i_0i_1-edge intersect

Listing 9 contains pseudocode for determining which of the 9 regions an intersection occurs (if any).

Listing 9. The query for ray-roundedface intersection testing.

```
int DoQueryRayRoundedFace(int i0, int i1, int i2, Vector3 K, Vector3 C, Real radius,
    Vector3 delta, Vector3 V, Real& contactTime, Vector3& contactPoint)
{
    Vector3 otherK, otherDelta;
    int type;

    Real T = (radius - delta[i0]) / V[i0];
    Real p1 = C[i1] + T * V[i1];
    Real p2 = C[i2] + T * V[i2];
```

```
if (p1 < -K[i1])
{
    // The ray potentially intersects the rounded (i0,i1)-edge
    // whose top-most vertex is otherK.
    otherK[i0] = K[i0];
    otherK[i1] = -K[i1];
    otherK[i2] = K[i2];
    otherDelta[i0] = C[i0] - otherK[i0];
    otherDelta[i1] = C[i1] - otherK[i1];
    otherDelta[i2] = C[i2] - otherK[i2];
    type = DoQueryRayRoundedEdge(i0, i1, i2, otherK, C, radius, otherDelta, V,
        contactTime, contactPoint);
    if (type == 0)
    {
        if (p2 < -K[i2])
        {
            // The ray potentially intersects the rounded (i2,i0)-edge
            // whose right-most vertex is otherK.
            otherK[i0] = K[i0];
            otherK[i1] = K[i1];
            otherK[i2] = -K[i2];
            otherDelta[i0] = C[i0] - otherK[i0];
            otherDelta[i1] = C[i1] - otherK[i1];
            otherDelta[i2] = C[i2] - otherK[i2];
            return DoQueryRayRoundedEdge(i2, i0, i1, otherK, C, radius, otherDelta, V,
                contactTime, contactPoint);
        }
        else if (p2 > K[i2])
        {
            // The ray potentially intersects the rounded (i2,i0)-edge
            // whose right-most vertex is K.
            return DoQueryRayRoundedEdge(i2, i0, i1, K, C, radius, delta, V,
                contactTime, contactPoint);
        }
    }
    return type;
}
else if (p1 <= K[i1])
{
    if (p2 < -K[i2])
    {
        // The ray potentially intersects the rounded (i2,i0)-edge whose
        // right-most vertex is otherK.
        otherK[i0] = K[i0];
        otherK[i1] = K[i1];
        otherK[i2] = -K[i2];
        otherDelta[i0] = C[i0] - otherK[i0];
        otherDelta[i1] = C[i1] - otherK[i1];
        otherDelta[i2] = C[i2] - otherK[i2];
        return DoQueryRayRoundedEdge(i2, i0, i1, otherK, C, radius, otherDelta, V,
            contactTime, contactPoint);
    }
    else if (p2 <= K[i2])
    {
        // The ray intersects the i0-face of the rounded box, so the
        // sphere-box contact point is on the corresponding box face.
        result.intersectionType = 1;
        result.contactTime = T;
        result.contactPoint[i0] = K[i0];
        result.contactPoint[i1] = p1;
        result.contactPoint[i2] = p2;
        return 1;
    }
    else  // p2 > K[i2]
    {
        // The ray potentially intersects the rounded (i2,i0)-edge
        // whose right-most vertex is K.
        return DoQueryRayRoundedEdge(i2, i0, i1, K, C, radius, delta, V,
            contactTime, contactPoint);
    }
}
```

```
    else // p1 > K[i1]
    {
        // The ray potentially intersects the rounded (i0,i1)-edge
        // whose top-most vertex is K.
        type = DoQueryRayRoundedEdge(i0, i1, i2, K, C, radius, delta, V,
            contactTime, contactPoint);
        if (type == 0)
        {
            if (p2 < -K[i2])
            {
                // The ray potentially intersects the rounded (i2,i0)-edge
                // whose right-most vertex is otherK.
                otherK[i0] = K[i0];
                otherK[i1] = K[i1];
                otherK[i2] = -K[i2];
                otherDelta[i0] = C[i0] - otherK[i0];
                otherDelta[i1] = C[i1] - otherK[i1];
                otherDelta[i2] = C[i2] - otherK[i2];
                return DoQueryRayRoundedEdge(i2, i0, i1, otherK, C, radius, otherDelta, V,
                    contactTime, contactPoint);
            }
            else if (p2 > K[i2])
            {
                // The ray potentially intersects the rounded (i2,i0)-edge
                // whose right-most vertex is K.
                return DoQueryRayRoundedEdge(i2, i0, i1, K, C, radius, delta, V,
                    contactTime, contactPoint);
            }
        }
        return type;
    }
    return 0;
}
```

Listing 10 contains pseudocode for determining the rounded box face to search for intersections.

Listing 10. The top-level intersection queries for the unbounded regions.

```
int VertexUnbounded(Vector3 K, Vector3 C, Real radius, Vector3 delta, Vector3 V,
    Real& contactTime, Vector3& contactPoint)
{
    if (V[0] < 0 && V[1] < 0 && V[2] < 0)
    {
        // Determine the face of the rounded box that is intersected by the ray C+T*V.
        Real T = (radius - delta[0]) / V[0];
        int j0 = 0;
        Real temp = (radius - delta[1]) / V[1];
        if (temp > T)
        {
            T = temp;
            j0 = 1;
        }
        temp = (radius - delta[2]) / V[2];
        if (temp > T)
        {
            T = temp;
            j0 = 2;
        }

        // The j0-rounded face is the candidate for intersection.
        int j1 = (j0 + 1) % 3;
        int j2 = (j1 + 1) % 3;
        if (DoQueryRayRoundedFace(j0, j1, j2, K, C, radius, delta, V, contactTime, contactPoint) == 1)
```

```
        {
            return 1;
        }
    }
    contactTime = 0;  // invalid
    contactPoint = Vector3(0, 0, 0);  // invalid
    return 0;
}

int EdgeUnbounded(int i0, int i1, int /* i2 */, Vector3 K, Vector3 C, Real radius, Vector3 delta,
    Vector3 V, Real& contactTime, Vector3& contactPoint)
{
    if (V[i0] < 0 && V[i1] < 0)
    {
        // Determine the face of the rounded box that is intersected by the ray C+T*V.
        Real T = (radius - delta[i0]) / V[i0];
        int j0 = i0;
        Real temp = (radius - delta[i1]) / V[i1];
        if (temp > T)
        {
            T = temp;
            j0 = i1;
        }

        // The j0-rounded face is the candidate for intersection.
        int j1 = (j0 + 1) % 3;
        int j2 = (j1 + 1) % 3;
        if (DoQueryRayRoundedFace(j0, j1, j2, K, C, radius, delta, V, contactTime, contactPoint) == 1)
        {
            return 1;
        }
    }
    contactTime = 0;  // invalid
    contactPoint = Vector3(0, 0, 0);  // invalid
    return 0;
}

int FaceUnbounded(int i0, int i1, int i2, Vector3 K, Vector3 C, Real radius, Vector3 delta,
    Vector3 V, Real& contactTime, Vector3& contactPoint)
{
    if (V[i0] < 0)
    {
        if (DoQueryRayRoundedFace(i0, i1, i2, K, C, radius, delta, V, contactTime, contactPoint) == 1)
        {
            return 1;
        }
    }
    contactTime = 0;  // invalid
    contactPoint = Vector3(0, 0, 0);  // invalid
    return 0;
}
```

---

4 Implementation

The GTEngine distribution has an implementation of the algorithm in files GteIntrAlignedBox3Sphere3.h and GteIntrOrientedBox3Sphere3.h, both sharing a member function DoQuery in the aligned-box case.

A sample application is GeometricTools/GTEngine/Samples/Mathematics/MovingSphereBox. For a specified stationary box, the query is performed for a moving sphere. The box and rounded box are drawn semitransparently as shown in Figure 1. The query sphere is drawn in light gray. When there is a first contact time

and point, a dark gray sphere is drawn to indicate where the light gray sphere moves to achieve first contact. The contact point is drawn as a small red sphere. The ray $\mathbf{C} + t\mathbf{V}$ is drawn as a line segment.

Figure 3 shows a couple of configurations for the intersection query.

Figure 3. Two query configurations. The contact points are displayed in the upper-left corners of the images.