V.8
A SIMPLE METHOD FOR BOX – SPHERE INTERSECTION TESTING

James Arvo
Apollo Systems Division of Hewlett-Packard
Chelmsford, Massachusetts

Introduction

There are a number of computer graphics applications in which it is necessary to determine whether a sphere intersects an axis-aligned parallelepiped, or "box." In two dimensions this arises when rasterizing circles, which are two-dimensional spheres, into rectangular viewports, which are two-dimensional boxes. The intersection test is used to determine whether any portion of the circle is indeed visible before rasterizing. In three dimensions this operation may arise in spatial subdivision techniques that identify "voxels," or 3D boxes, pierced by the surfaces of various objects. If spheres are among the objects, or are used as bounding volumes for other objects, we need to test for 3D box-sphere intersection.

This note describes a simple method for detecting intersections of n-dimensional boxes with n-dimensional spheres taken as either surfaces or solids. Applications for $n > 3$ are not addressed here, though the algorithm works in any dimension. For greatest generality all algorithms shown below take the dimension of the space, n, as an input parameter, though typical implementations would be optimized for either two or three dimensions.

Solid Objects

Suppose we wish to determine whether a solid n-dimensional box, B, intersects a solid n-dimensional sphere with center C and radius r. Here

"solid" means that we include the interior as well as the boundary. Denote C by $(C_1, \ldots, C_n)$ and B by the closed intervals $[B_1^{min}, B_1^{max}]$, $\ldots$, $[B_n^{min}, B_n^{max}]$. We can perform this test by finding the point P on or in the box that is closest to the center of the sphere. If the distance between this point and C is no greater than r, then the box intersects the sphere. Thus, if the point $P \in R^n$ minimizes the distance function

$$\text{dist}(P) = \sqrt{(C_1 - P)^2 + \ldots + (C_n - P_n)^2}, \tag{1}$$

subject to the constraints (2)

$$B_i^{min} \le P_i \le B_i^{max} \quad \text{for } i = 1, \ldots, n,$$

then the solids intersect if and only if dist(P) $\le r$. As a simplification we will eliminate the square root by computing squared distances and comparing with r^2. It is a simple matter to find the P that minimizes the square of Eq. 1 because each term of the sum is nonnegative and can be minimized independently. If the ith coordinate of the sphere center satisfies the ith constraint, that is, if $B_i^{min} \le C_i \le B_i^{max}$, then setting $P_i = C_i$ reduces this term to zero. Otherwise, we set P_i equal to either B_i^{min} or B_i^{max} depending on which is closer. Summing the squares of the distances produces the minimum total squared distance. The algorithm in Fig. 1 uses this principle to determine whether a solid box intersects a solid sphere. The input parameters are the dimension n, box B, sphere

```
boolean function SolidBox_SolidSphere(n, B, C, r)
begin
    d_min ← 0;
    for i ← 1 . . . n do
        if C_i > B_i^max then d_min ← d_min + (C_i − B_i^max)^2;
        else
            if
        endloop; C_i < B_i^min then d_min ← d_min + (C_i − B_i^min)^2;
    if d_min ≤ r^2 then return [True]
    else return [False];

    end;
```

Figure 1. An algorithm for intersecting a solid n-dimensional box with a solid n–dimensional sphere.

center *C*, and sphere radius *r*. The function value is returned as "True" if the objects intersect and "False" otherwise.

Hollow Objects

If we wish to make either or both of the objects hollow and test only their surfaces, we can do so by regarding total inclusion of one object inside the other as nonintersection. For instance, if we wish to test whether the surface of the sphere intersects the solid box, we can add a test for whether the box is entirely contained within the sphere and regard this as nonintersection. This is shown in the algorithm of Fig. 2, in which we've added the computation of the square distance from *C* to the farthest point of *B*. If this value, denoted d_{max}, is less than r^2, the entire box is inside the sphere and there is no intersection.

The approach is different if we wish to disregard the interior of the box. Here the constraints of Eq. 2 still hold but in order for the point *P* to be on the boundary of the box we require the additional constraint that $P_i = B_i^{min}$ or $P_i = B_i^{max}$ for some *i*. In the algorithm of Fig. 1 we see that this holds unless $C_i \in (B_i^{min}, B_i^{max})$ for all *i*. In this case we need to determine whether moving *P* to the nearest face of the box places it

```
boolean function SolidBox_HollowSphere(n, B, C, r)
begin
    d_max ← 0;
    d_min ← 0;

    for i ← 1 . . . n do
        a ← (C_i − B_i^min)²
        b ← (C_i − B_i^max)²;
        d_max ← d_max + max(a, b);
        if C_i ∉ [B_i^min, B_i^max] then d_min ← d_min + min(a,b);
        endloop;

    if r² ∈ [d_min, d_max] then return [True]
    else return [False];

    end;
```

Figure 2. An algorithm for intersecting a solid *n*-dimensional box with a hollow *n*-dimensional sphere.

```
boolean function HollowBox_SolidSphere(n, B, C, r)
begin
    d_min ← 0
    d_face ← infinity
    for i ← 1 . . . n do
        t ← min((C_i − B_i^min)^2,(C_i − B_i^max)^2);
        if C_i ∈ [B_i^min, B_i^max] then d_face ← min(d_face,t);
        else begin
            d_face = 0;
            d_min ← d_min + t;
            end
        endloop;
    if d_min + d_face ≤ r^2 then return [True]
    else return [False];

    end;
```

Figure 3. An algorithm for intersecting a hollow n-dimensional box with a solid *n*-dimensional sphere.

outside the sphere. In Fig. 3 we have modified the algorithm to compute the smallest such adjustment, denoted d_{face}.We add this to the minimum distance between the box and *C*, and if the result remains less than r^2, the surface of the box intersects the solid sphere. The approaches in Fig. 2 and Fig. 3 can be combined to test for intersection between a hollow box and a hollow sphere.

Generalizing to Ellipsoids

We can easily generalize this idea to work with axis-aligned ellipsoids, that is, ellipsoids that result from scaling a sphere differently along the coordinate axes. We can specify such a ellipsoid in *n*-space by its center, $C[R^n$, and a "radius" for each axis, $a_1, \ldots, a_n$. The point $P \in R^n$ is on or in such an ellipsoid if and only if

is on or in such an ellipsoid if and only if

$$\left[\frac{C_1 - P_1}{\alpha_1}\right]^2 + \cdots + \left[\frac{C_n - P_n}{\alpha_n}\right]^2 \leq 1. \qquad (3)$$

```
boolean function SolidBox_SolidEllipsoid(n, B, C, α)
begin
    d_min ← 0;
    for i ← 1 ... n do
        if C_i < B_i^min then d_min ← d_min + [ (C_i − B_i^min) / α_i ]^2 ;
    else
        if C_i > B_i^max then d_min ← d_min + [ (C_i − B_i^max) / α_i ]^2 ;
    endloop;
    if d_min ≤ 1 then return [True]
    else return [False];
    end;
```

Figure 4. An algorithm for intersecting a solid *n*-dimensional box with a solid *n*-dimensional axis-aligned ellipsoid.

Modifying the algorithm in Fig. 1 to handle this type of ellipsoid results in the algorithm shown in Fig. 4. Here the scalar input parameter *r* has been changed to the array α. Modifications for either hollow boxes or hollow ellipsoids are analogous to those previously described.

See Appendix 2 for C Implementation (730)