I.12

FAST CIRCLE–RECTANGLE INTERSECTION CHECKING

Clifford A. Shaffer
Virginia Tech
Blacksburg, Virginia

If you do a lot of graphics or spatial data programming, sooner or later you will want to know if a circle and a rectangle intersect, or if a sphere and a box intersect. This is even more likely if you use quadtree or octree methods. (For example, you may want to find all nodes of the tree within a certain Euclidean distance of a point). Unfortunately, this problem is not as easy to solve as it appears. The first approach that normally comes to mind is to check if any corner of the rectangle falls within the circle (using a simple distance check). Unfortunately, this approach will sometimes give false negative results. There are three anomalous cases to watch out for. First, while no corner of the rectangle may be in the circle, a chord of the circle may overlap one edge of the rectangle (see Fig. 1). Second, the rectangle might fall inside a bounding box placed around the circle, but still be outside the circle (see Fig. 2). Third, the circle might lie entirely inside the rectangle (see Fig. 3).

A fast algorithm is presented, for determining if a circle and a rectangle intersect. The 3D case can easily be derived from the 2D case; although it is a little longer, it requires only slightly more execution time. The 2D version of this algorithm requires at most five comparisons (all but one test against 0), three multiplies, five add/subtracts (four of which are for normalization) and one absolute-value function. It basically works by determining where the rectangle falls with respect to the center of the circle. There are nine possibilities in 2D (27 in 3D): the rectangle can be entirely to the NW, NE, SW, or SE of the circle's centerpoint (four cases), directly N, E, S, or W of the circle's centerpoint (four cases) or in the center (that is, containing the circle's centerpoint). The algorithm

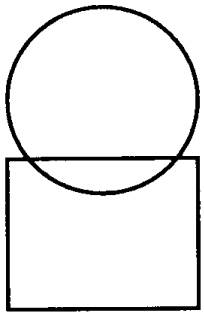

Figure 1.

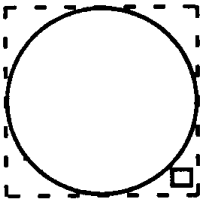

Figure 2.

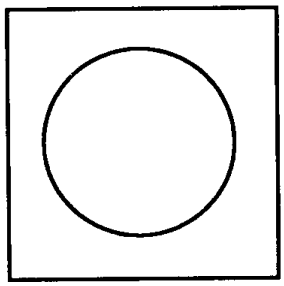

Figure 3

enumerates these cases and determines the distance between the single closest point on the border of the rectangle and the center of the circle.

```
boolean Check_Intersect(R, C, Rad)

Return TRUE iff rectangle R intersects circle with centerpoint C and radius Rad.
begin Comments assume origin is at lower left corner
      Translate coordinates, placing C at the origin
      R.max ← R.max – C; R.min ← R.min – C;
      if (R.max.x < 0) R to left of circle center
           then if (R.max.y < 0) R in lower left corner
                     then return (R.max.x² + R.max.y² < Rad²);
                     else if (R.min.y > 0) R in upper left corner
                               then return (R.max.x² + R.min.y² < Rad²);
                               else R due West of circle
                                         return (|R.max.x|< Rad):
```

```
else if (R.min.x > 0) R to right of circle center
        then if (R.max.y < 0) R in lower right corner
                then return (R.min.x² + R.max.y² < Rad²);
                else if (R min.y > 0) R in upper right corner
                        then return (R.min.x² + R.min.y² < Rad²);
                        else R due EAST of circle
                                return (R.min.x < Rad)
        else R on circle vertical centerline
         if (R.max.y < 0) R due South of circle
                then return (|R.max.y| < Rad);
                else if (R.min.y > 0) R due North of circle
                        then return (R.min.y < Rad);
                        else R contains circle centerpoint
                                return (TRUE);
end; Check_intersect
```

See also Fast Ray-Box Intersection (395); Spheres-to-Voxels Conversion (327); A Simple Method for Box-Sphere Intersection Testing (335); Ray Tracing (383)

See Appendix 2 for C Implementation (656)