◊ VII.2

Fast Polygon–Cube Intersection Testing

Daniel Green
Autodesk — Multimedia Division
San Rafael, California
daniel.green@autodesk.com

Don Hatch
Silicon Graphics, Inc.
Mountain View, California
hatch@sgi.com

◊ Overview ◊

This gem generalizes previous triangle–cube intersection methods (Voorhies 1992) to support arbitrary n-gons. Convex, concave, self-intersecting, and degenerate polygons are fully treated, and the new algorithm is more efficient and robust. The implementation uses the C vector macro library `vec.h` created by the second author (page 404).

◊ Background ◊

Efficient polygon–cube intersection testing is an important problem in computer graphics. Renderers can profit from fast polygon tests against display volumes, often avoiding the more expensive clipping operation. Likewise, bounding volume techniques can utilize such fast tests on the faces of polyhedral volumes.

The previous gem cited above gives an algorithm that tests whether a given triangle intersects the axially aligned cube of unit edge centered at the origin.

A related gem (Greene 1994) describes an efficient algorithm for testing convex polyhedra against axially aligned boxes. That algorithm works by attempting to find a plane separating the two figures. That work mentions that the intuitive approach is inefficient because of the number of possible intersection calculations. (The intuitive approach contains an intersection test of each polygon edge with each cube face, followed by intersecting each cube diagonal with the polygon body.)

◊ Description ◊

The approach presented here is a hybrid of the two previous techniques. It contains only a single intersection calculation, which is rarely performed because of the trivial tests that precede it. The rest of the calculations are of the same sort of fast inequality tests from the second work. This new implementation, however, is not restricted to convex figures.

IBM ISBN 0-12-543455-3
Macintosh ISBN 0-12-543457-X

Table 1. Intersection routines.

Function Name	File Name	Description
polygon_intersects_cube	pcube.c	low-level definitive test
fast_polygon_intersects_cube	fpcube.c	high-level wrapper test
trivial_vertex_tests	fpcube.c	trivial reject/accept test
segment_intersects_cube	pcube.c	low-level edge test
polygon_contains_point_3d	pcube.c	used internally, also generally useful

The top-level entry points of the implementation are described in Table 1. Further instructions appear within the heavily commented source code released with this book.

The previous triangle testing approach is elegant and sound, and the general approach has been retained, which proceeds from cheap trivial accept and reject tests through more expensive edge and face intersection tests. These individual tests have also been broken out into separate functions in order to allow higher-level routines to be built on top of them—such as general polyhedra and polygon mesh tests—without having to suffer redundant tests on shared vertices or edges.

The composite `fast_polygon_intersects_cube` function replaces Voorhies' original triangle–cube intersection function. It calls `trivial_vertex_tests` (essentially unchanged) and, failing classification, invokes the definitive `polygon_intersects_cube` function. This overall behavior is unchanged from the original code:

1. Trivial vertex tests.
2. If any edge intersects the cube, return TRUE.
3. If the polygon interior intersects the cube, return TRUE.
4. Return FALSE.

Step 1: Trivial Vertex Tests

The main algorithmic difference in the new point–plane tests is that in a number of places tests are no longer performed against planes that cannot possibly give useful information. For example, when the function determines that a point is located to the left of the cube, testing is not done against the cube's right face plane.

The `trivial_vertex_tests` function can be used to test an entire set of vertices for trivial rejection or acceptance. This test is useful for quickly classifying polyhedra or entire polygon meshes. Another useful application is in testing for trivial rejection of polyhedral bounding volumes against view volumes (described more fully in the next section).

The `trivial_vertex_tests` function stops testing vertices as soon as it determines that at least one vertex is to the inside of each plane. For example, suppose that at least one vertex has been found to be to the right of the left face plane, and one is found to be below the top face plane, and likewise for the other four face planes. There is then

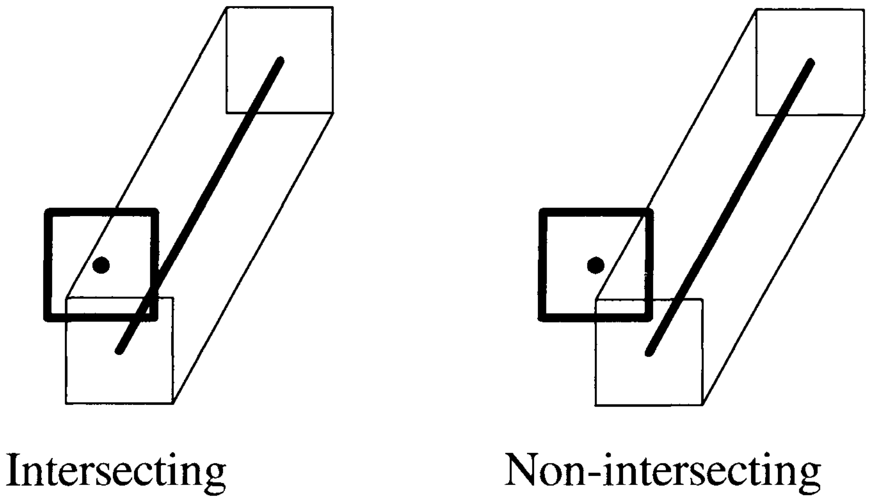

Figure 1. Two-dimensional analogue of segment–cube intersection test.

no point in classifying any of the remaining vertices against the face planes because it is impossible that the vertices *as a set* all lie outside any one of those planes.

Step 2: Segment–Cube Intersection Test

A naïve implementation of this step consists of checking whether either endpoint of the segment is inside the cube, and, if not, checking whether the segment intersects any of the six cube faces. Such an implementation, however, is inherently error-prone: It gives a false negative when, due to floating-point roundoff error, the segment slips through the "cracks" between adjacent cube faces. In fact, the original gem's implementation suffers from exactly this problem.

The approach used here is to convert this part of the problem into a different problem space: Testing whether a line segment intersects the cube is equivalent to testing whether the origin is contained within the convex solid obtained by sweeping a unit cube from (being centered at) one segment endpoint to the other. This solid is a skewed rhombic dodecahedron. The code to implement this test consists of just twelve point–plane sidedness tests, so it is computationally more efficient than the original six line–plane intersections plus six point-within-polygon tests, in addition to being more robust.

Figure 1 shows the analogous situation in two dimensions, where the swept convex solid is simply a hexagon. The line segment intersects the square if and only if the hexagon contains the center of the square.

Any intersection test can be recast in this way. In general, testing whether two objects A and B have any points in common is equivalent to testing whether the origin is contained within the composite object:

$$\{a - b \mid a \in A \mathit{and}\, b \in B\}.$$

This reasoning can be applied to the entire original polygon–cube intersection problem by recasting it as a single point-within-solid test. However, in order to handle non-convexity or tiny facets in the composite solid, the analysis required is much more complex than simple sidedness tests. For this reason, that approach was specifically rejected, although the method is appropriate for the segment–cube intersection step.

Step 3: Polygon Interior–Cube Intersection Test

Since it is now known that no vertex or edge intersects the cube, this step only needs to test whether any of the four cube diagonals intersects the interior of the polygon. This observation was utilized in the original gem's implementation. The new implementation goes a step further and uses the fact that it is sufficient to test against only the one diagonal that comes closest to being perpendicular to the plane of the polygon; if the polygon intersects any of the cube diagonals, it will intersect that one. Finding that diagonal is trivial, so this part of the implementation is up to four times as fast as the original. Omitting the intersection tests with the other three diagonals avoids another case of numerical instability in the original gem's implementation: Calculating the intersection point of the polygon's plane with a cube diagonal that is almost parallel to that plane can result in a divide-by-zero or unstable solution.

The last part of this step is to test whether the polygon contains the point that is the intersection of the polygon's plane with the chosen diagonal. This test is performed by the function `polygon_contains_point_3d`, which is made externally visible since it may be useful for other applications.

◇ Polyhedron–Cube Intersection Testing ◇

When used to test polyhedra, the functions included in this module only test for intersections with points, edges, and surfaces, not volumes. If no such intersection is reported, the volume of a polyhedron could still contain the entire unit box. That condition would then need to be checked for with an additional point-within-polyhedron test. The origin would be a natural point to check in such a test. Below is C-like pseudocode that puts all the pieces together for a fast, complete polyhedron–cube intersection test.

```
switch(trivial_vertex_tests(verts))
{
    case  1: return TRUE  /* trivial accept */
    case  0: return FALSE /* trivial reject */
    case -1: for each edge
```

```
                if(segment_intersects_cube(edge))
                    return TRUE
            for each face
                if(fast_polygon_intersects_cube(..., TRUE, TRUE))
                    return TRUE
            return polyhedron_contains_point(polyhedron, origin)
    }
```

Notice that when a box is used as a modeling-space bounding polyhedron, testing its intersection against a view volume can often be performed in either direction. In other words, not only can the box be transformed by the viewing transformation that takes the view volume to the unit cube and then tested there, but the view volume can also be transformed by the transformation that takes the bounding box to be the unit cube and the test performed there. In the latter case it is the world-space truncated pyramid of the view volume that becomes the polyhedron being tested.

◇ Conclusions ◇

A set of highly optimized intersection tests was described that support a fast polygon–cube intersection test at the highest level. The latter supports operations upon general n-gons. All intersection routines are freestanding, making them good candidates for direct replacement of related routines described previously in this series. Production versions of this gem's code may be found in the subdirectory `pcube` on the accompanying diskette and FTP mirrors. The companion library may be found in `vec.h` and is described in gem VII.7 of this volume.

◇ Bibliography ◇

(Greene 1994) Ned Greene. Detecting intersection of a rectangular solid and a convex polyhedron. In Paul S. Heckbert, editor, *Graphics Gems IV*, Chapter I.7, pages 74–82. AP Professional, Boston, 1994.

(Voorhies 1992) Douglas Voorhies. Triangle-cube intersection. In David Kirk, editor, *Graphics Gems III*, Chapter V.7, pages 236–239. AP Professional, Boston, 1992.