Lecture 4: Algorithms for 3D volumes

In building graphical scenes from a number of sub-objects, it is frequently necessary to discover how one object interferes with another. We may wish to do this to remove hidden lines from parts of a wireframe representation, or alternatively we may wish simply to check for collisions between objects, which, in the design of chemical plant piping for example, would indicate errors in the design. This study normally involves two fundamental tests: *containment*, which checks to see if a point is inside an object and *clipping*, which determines where a line (or polygon) intersects an object.

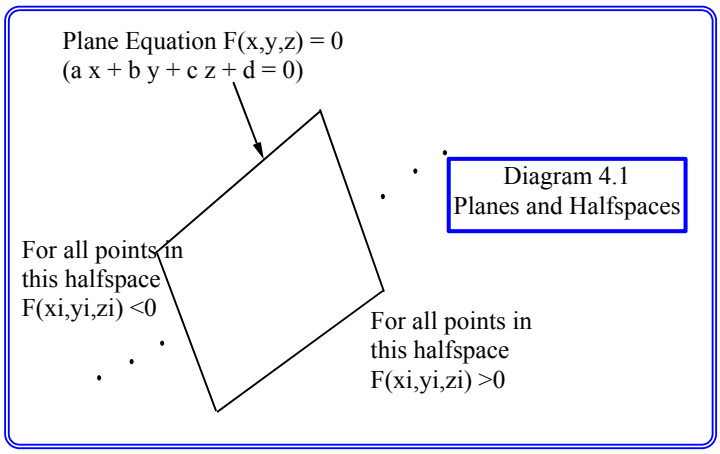


Diagram 4.1
Planes and Halfspaces

Containment within a convex volume

A simple definition of a convex volume is one where the line joining any two points on the surface is completely contained within the volume. An object produced by the intersection of infinite planes is always convex. The convexity of an object bounded by planar facets can be tested by the following simple algorithm (see Diagram 4.1):

```
convex:=true;
for each planar facet do
begin
{ find the functional equation of the facet plane f(x,y,z)=0 }
for all other vertices not belonging to this facet do
    if sign(f(xi,yi,zi))<>sign(previous vertex)
    then  convex:=false;
end
```

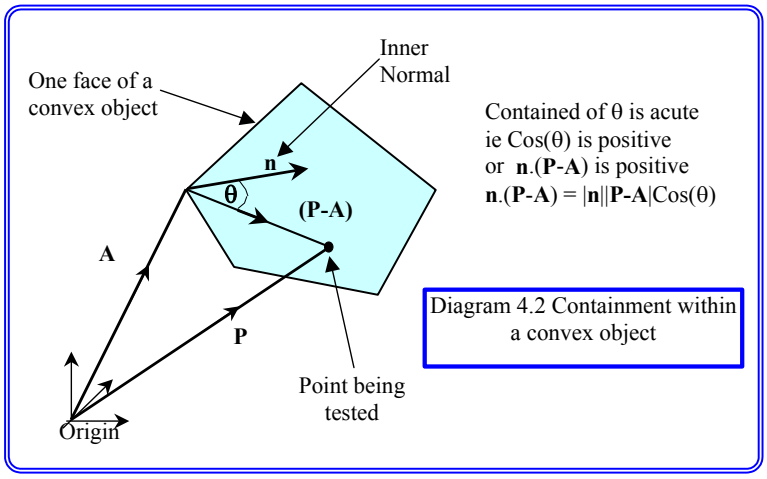


Diagram 4.2 Containment within a convex object

This is not a particularly efficient algorithm but it works for all cases. For convex objects there is an easy test to determine whether a point **P** is contained within that volume. Consider a line drawn from the point under test to any point on any surface of the convex volume as shown in Diagram 4.2. If the point is inside the volume then the angle made by the line to the surface inward normal is always acute (or zero).

$$\mathbf{n}.(\mathbf{P} - \mathbf{A}) >= 0$$

A is a point on the surface, **n** is the inner normal of the surface. As shown in Diagram 4.3, we do not necessarily know the direction of the surface normal. To find the normal to any surface we take the cross product of two edges, and to determine its sign we test it with a vertex of an adjoining surface as shown in

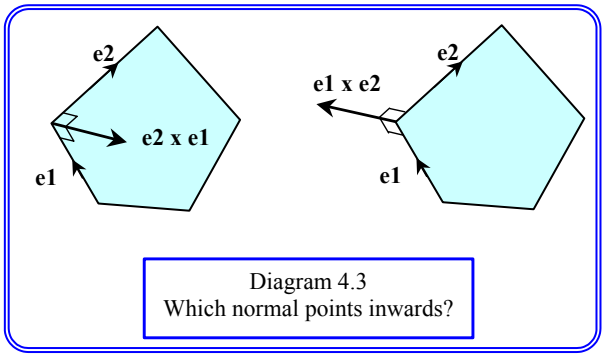


Diagram 4.3
Which normal points inwards?

Diagram 4.4. The inner surface normal is a useful quantity to store in the data structure describing the object.

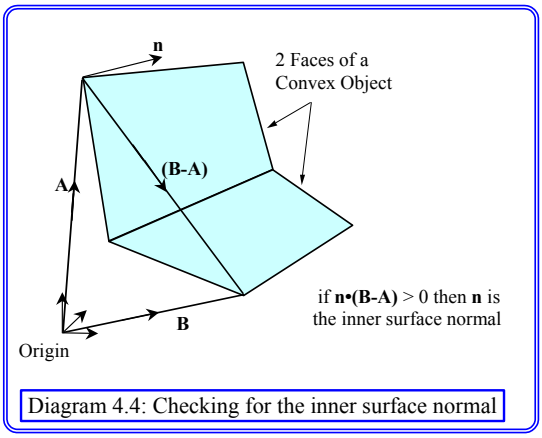


Diagram 4.4: Checking for the inner surface normal

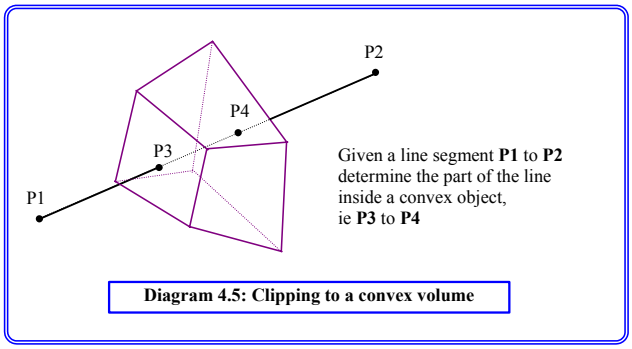


Diagram 4.5: Clipping to a convex volume

Clipping to a convex volume (Cyrus Beck Algorithm)

This algorithm is a simple extension of the containment algorithm given above. Given a line joining **P1** and **P2** and a convex volume, we wish to find the part of that line that lies completely within the volume as shown in Diagram 4.5. For each surface, we take a point **A** on the surface, which could simply be one of the vertices, and examine the signs of:

n.(P1 - A) and **n.(P2 - A)**

where **n** is is the inner surface normal. The following cases are possible:

Both signs -ve or zero: The line is completely outside the body and the algorithm terminates returning null.

Both signs +ve or zero: The line could be inside the body, but it is not clipped by this plane.

Both signs zero: The line lies on the plane. Depending on the desired result, either finish or proceed to the next surface.

One sign negative, one sign positive:

Find the intersection of the line and the plane by solving:

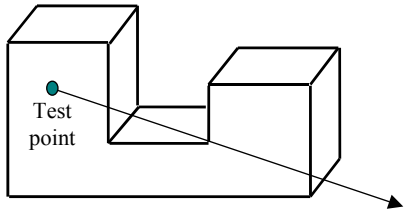
$\mathbf{n}.(\mu\mathbf{P2} + (1-\mu)\mathbf{P1} - \mathbf{A}) = 0$.

Replace the point that yielded the negative sign by the intersection point. When all surfaces have been considered, the remaining line span is the required result.

Containment within a non-convex volume

If the volume is concave, the best test for containment is the piercing test. We take a ray, for simplicity parallel to one of the axes, and count the number of intersections it makes leaving the body. If that is odd, the point is contained, if even then it is outside. (Diagram 4.6)

Suppose the point under test is: **P1** = [x1,y1,z1] and we choose: **P2** = [x1+1,y1,z1]
then the line: $\mu\mathbf{P2} + (1-\mu)\mathbf{P1}$ is a ray parallel to the x axis.



A ray is projected in any direction.

If the number of intersections with the object is odd, then the test point is inside

Diagram 4.6
The Ray Intersection test

We test this ray against the plane of each surface of the body, finding the intersection between the plane containing the face and the ray. This calculation is very simple for the ray parallel to the x direction since the y and z components in the equations are known constants. If the intersection point is such that $\mu<0$ we ignore it since we are only interested in intersections in one direction which we choose to be positive μ.

If the intersection point does have a positive μ, then we do a further test to find if it is contained within the edges bounding that face. This is essentially a two dimensional containment test and it is done by the same method. This time however, we need to choose a ray that lies on the plane. One simple way of doing that is to take a point on the boundary, for example midway between two vertices **Pb** = (**Vj**+**Vk**)/2 (see Diagram 4.7). This gives us a line equation:

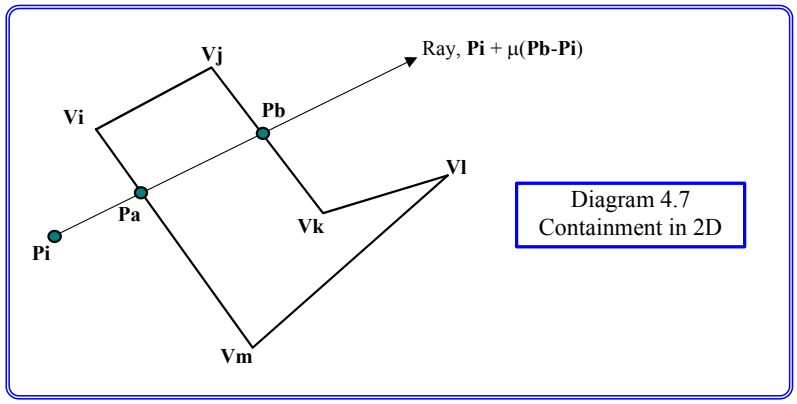


Diagram 4.7
Containment in 2D

P = **Pi** + μ(**Pb-Pi**)

each line of the face has equation

P= **Vi** + ν(**Vj-Vi**)

and we solve for the intersection point values of μ and ν. Note that we have a redundant system of equations (3 equations and two unknowns) since the plane of the face is not the x-y plane. In practice we the problem can be solved in orthographic projection but we may need to choose the correct pair of variables. If at the solution point $\mu>0$ and $0<=\nu<1$ we add 1 to the intersection count. Note that there is a problem when the intersection goes through a vertex as to how many intersections are calculated. When all lines have been tested, if the count is even then the point is contained in this polygon, and we add one to our 3D count of intersections.

Clipping to a Concave Volume

Clipping can be easily achieved by the same basic methods used for the containment test. We need to find every intersection of the line to be clipped with the volume. Thus the line to be clipped is intersected with the plane of each face, and a valid intersection is recorded if the intersection found is between the end points of the line and within the polygon bounding the face. This process divides the line into one or more segments. Now it follows that the segments on the line alternate between being inside and outside the concave volume. If we test a point on the first segment for containment, we can then deduce which parts of the line are inside and which are outside, and we can clip the line accodringly.

Breaking a Concave volumes into several Convex Volumes

Algorithms for concave objects are in general difficult, and contain many exceptional cases which need to be dealt with separately. Thus, it is often simpler to break a concave object into a set of convex ones, and perform a simpler algorithm on each of the resultant parts.

To do this we need to consider the plane of each face in turn. Now if **n** is the normal vector of that plane we know that the sign of **n.**(**P** - **A**) will determine which side of the plane the point P lies. We therefore test each vertex of the polyhedron. If they all have the same sign, or are zero, then the polyhedron is convex with respect to that plane as shown in figure 4.8. If not we break the object into convex parts.

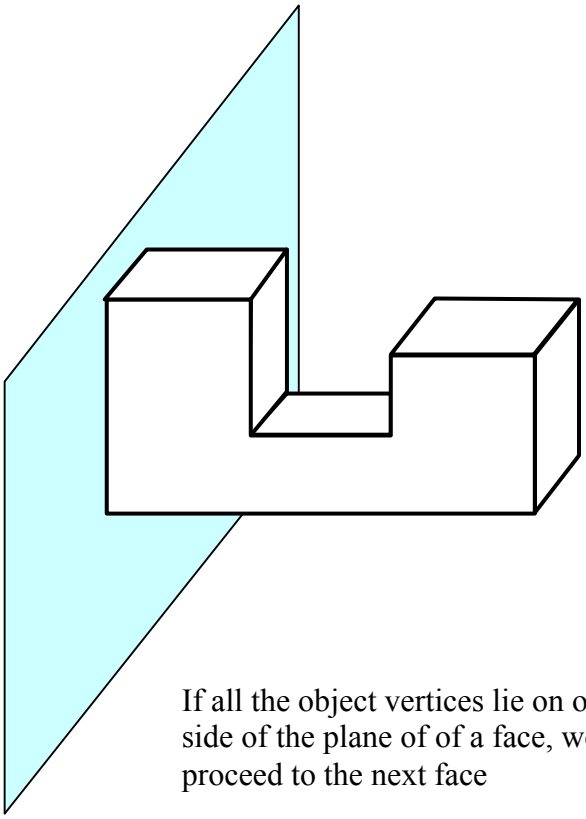
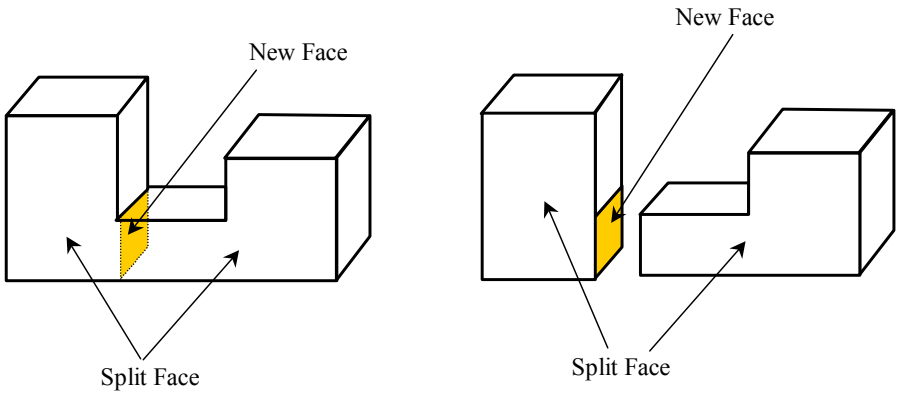


Diagram 4.8: Testing the faces of a concave volume

The first step is to determine how many sub-objects the original object has been divided into. This can be done by checking the connectivity of the vertices. For each connected set of vertices on one side of the cutting plane, it is then necessary to find all the edges that project from them that intersect with the cutting plane. Finally, it is necessary to determine how these points are arranged to form the closed polygon(s) on the cutting plane which will be bounding face(s) of the sub-object.

Use can be made of the fact that the new boundaries found will be the same for objects on both sides of the cutting plane. Once an object has been divided into two or more sub-objects the algorithm is re-entered with those sub-objects, and proceeds until all sub-objects are found to be convex. The implementation details are complex, and depend strongly on the data structure chosen to represent the object.



Repeat on all concave sub parts