further, the separating-axis test was suggested for the collision detection of oriented bounding boxes by [Larcombe95].

5.2.1.1 Robustness of the Separating-axis Test

A potential problem with the separating-axis test is robustness in the case of a separating axis being formed by the cross product of an edge from each object. When these two edges become parallel, the result is the zero vector and all projections onto this axis, and sums of these projections, are therefore zero. Thus, if the test is not carefully crafted, a zero vector may incorrectly be interpreted as a separating axis. Due to the use of floating-point arithmetic, this problem may occur even in the case of a near-zero vector for two near-parallel edges. In fact, the robustness problem of the separating-axis test was encountered in Section 4.4.2 in the context of the OBB-OBB intersection test, and a solution for the problem in that particular context can be found there.

When possible, it is best to analyze the robustness problem in the context in which it will occur. A generic solution to the problem is to test if the resulting cross-product vector is a (near) zero vector, and if so attempt to deal with the problem either by producing another axis that is perpendicular to the two vectors or by ignoring the axis if separation on the axis can be ruled out.

The following code fragment outlines how a more robust separating-axis test for edges *AB* and *CD* could be implemented.

```
// Compute a tentative separating axis for ab and cd
Vector m = Cross(ab, cd);
if (!IsZeroVector(m)) {
    // Edges ab and cd not parallel, continue with m as a potential separating axis
    ...
} else {
    // Edges ab and cd must be (near) parallel, and therefore lie in some plane P.
    // Thus, as a separating axis try an axis perpendicular to ab and lying in P
    Vector n = Cross(ab, c - a);
    m = Cross(ab, n);
    if (!IsZeroVector(m)) {
        // Continue with m as a potential separating axis
        ...
    }
    // ab and ac are parallel too, so edges must be on a line. Ignore testing
    // the axis for this combination of edges as it won't be a separating axis.
    // (Alternatively, test if edges overlap on this line, in which case the
    // objects are overlapping.)
    ...
}
```

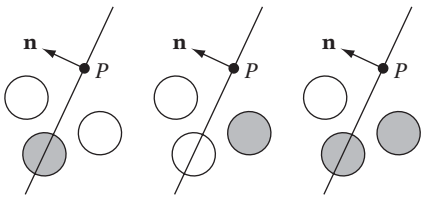

Figure 5.15 Illustrating the three sphere-plane tests. (a) Spheres intersecting the plane. (b) Spheres fully behind the plane. (c) Spheres intersecting the negative halfspace of the plane. Spheres testing true are shown in gray.

The **`IsZeroVector()`** function tests if its argument is a vector with a magnitude sufficiently close to zero (according to some tolerance value; see Section 11.3.1). Widening the tolerance intervals and treating near-parallel edges as parallel may result in near-intersections being interpreted as intersections. Overall, this is much more attractive than the alternative: two intersecting objects falsely reported as nonintersecting due to the projection onto a zero-vector separating axis, for example.

A related source of robustness errors is when the vectors used in the cross product have a large magnitude, which may result in additional loss of precision in the calculations involving the cross product. If a bound on the magnitude of the input vectors it not known, it is prudent to normalize them before computing the cross product to maintain precision.

5.2.2 Testing Sphere Against Plane

It is possible to test a sphere against a plane in several ways. This section describes three such tests: testing if the sphere intersects the plane, if the sphere lies fully behind the plane, and if the sphere intersects the negative halfspace of the plane. Figure 5.15 illustrates these three scenarios.

Let a sphere S be specified by a center position C and a radius r, and let a plane π be specified by $(\mathbf{n} \cdot X) = d$, where $\mathbf{n}$ is a unit vector; that is, $\|\mathbf{n}\| = 1$. To determine if the sphere is intersected by the plane, the plane equation can be evaluated for the sphere center. Because $\mathbf{n}$ is unit, the resulting value corresponds to the signed distance of the sphere center from the plane. If the absolute value of the distance is within the sphere radius, the plane intersects the sphere:

```
// Determine whether plane p intersects sphere s
int TestSpherePlane(Sphere s, Plane p)
{
   // For a normalized plane (|p.n| = 1), evaluating the plane equation
```