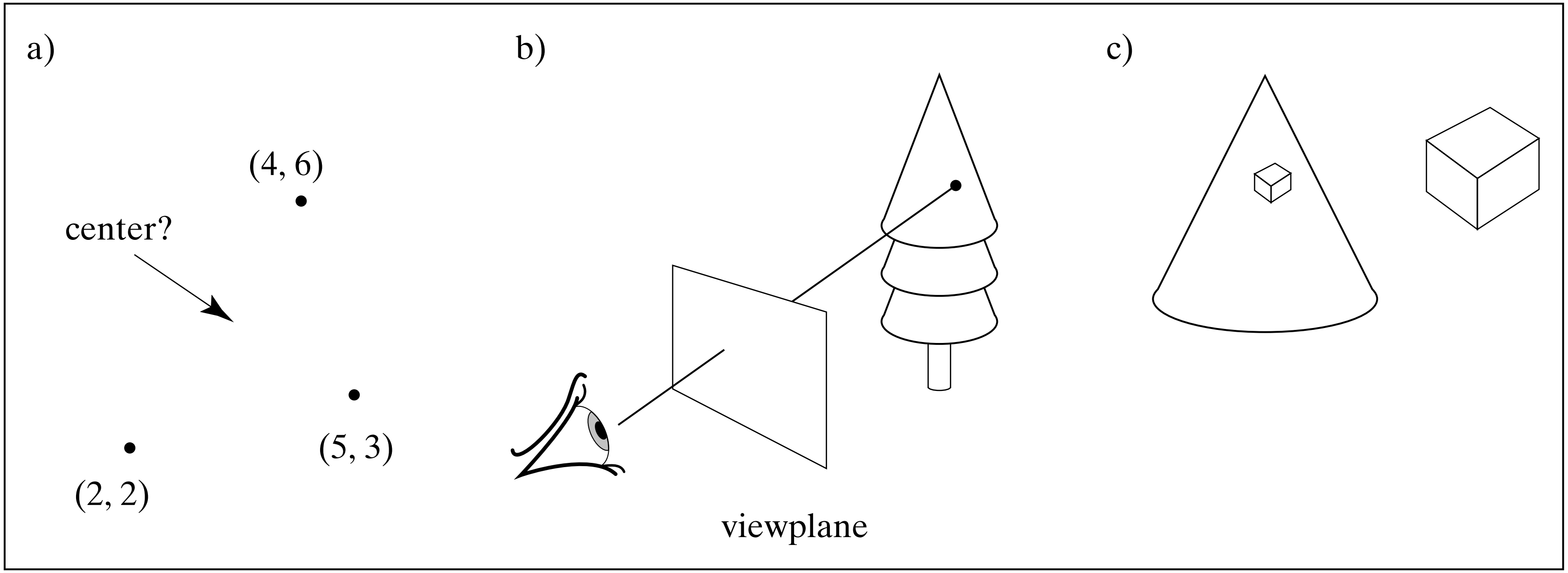
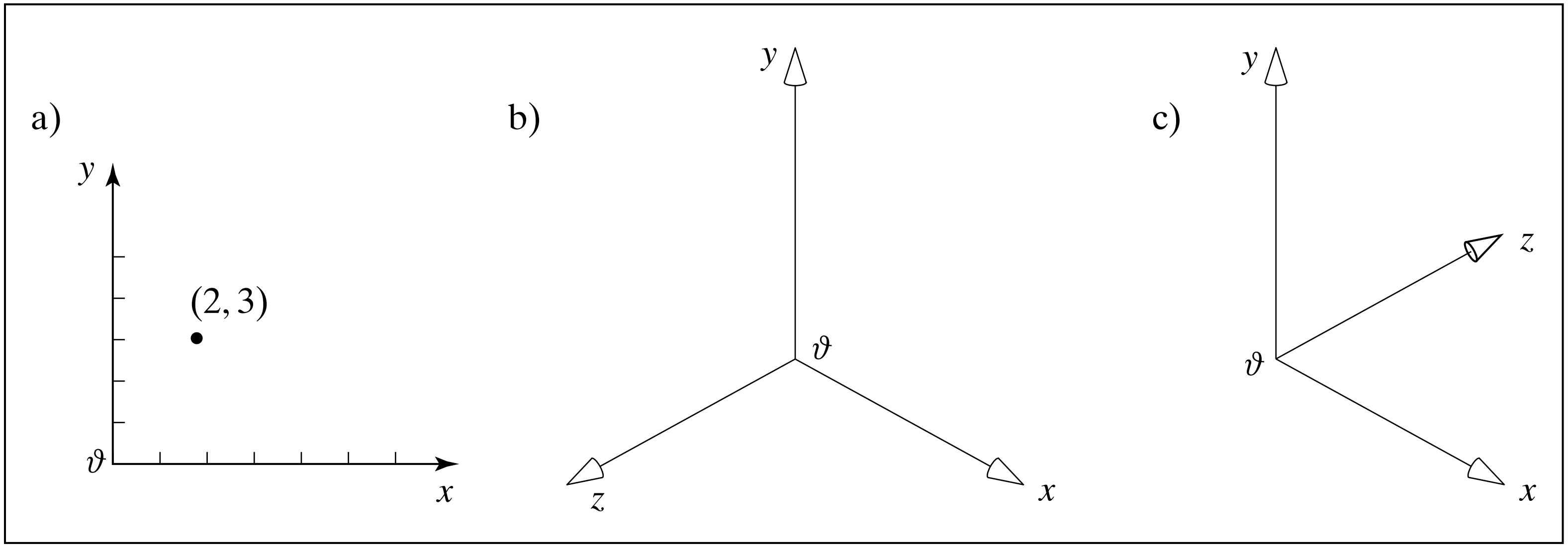


FIGURE 4.1 Three sample geometric problems that yield readily to vector analysis.

FIGURE 4.2 The familiar two- and three-dimensional coordinate systems.



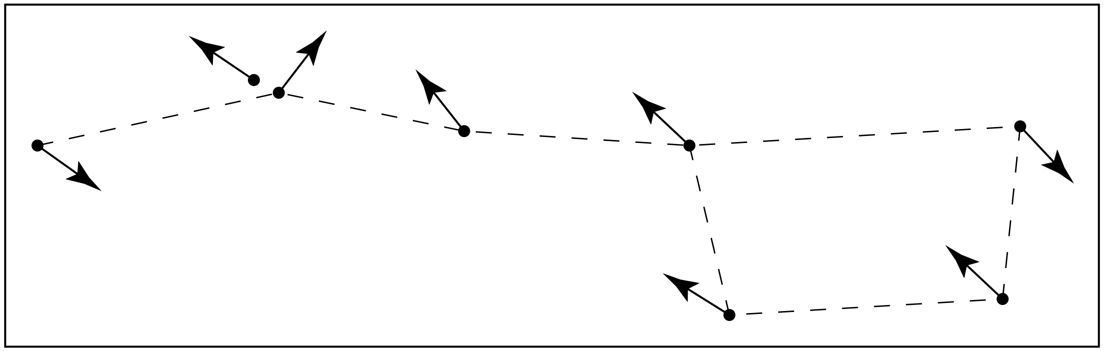
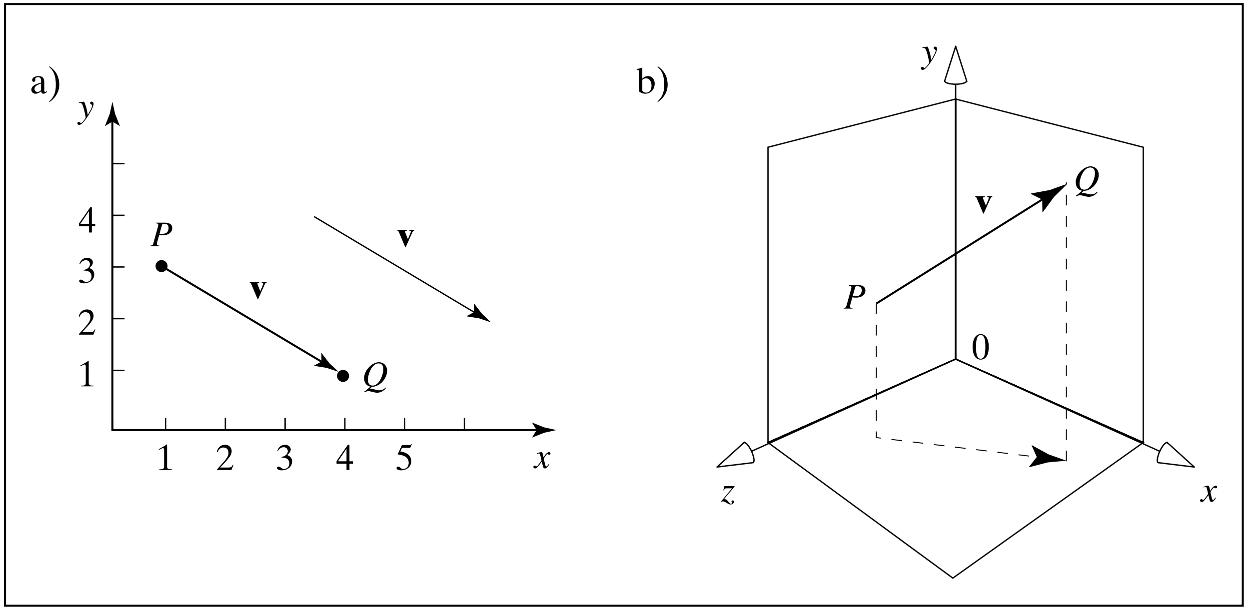


FIGURE 4.3 The Big Dipper now and in AD 50,000.

FIGURE 4.4 A vector as a displacement.



Prentice Hall
from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

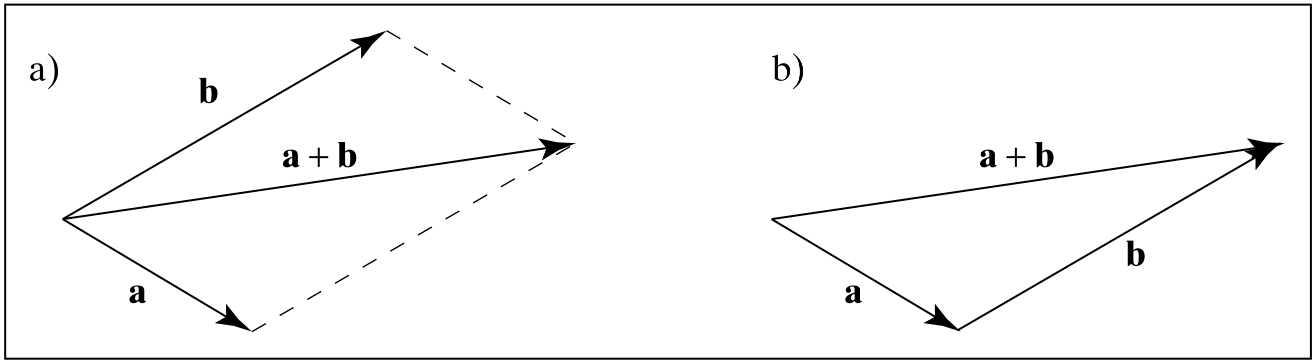


FIGURE 4.5 The sum of two vectors.

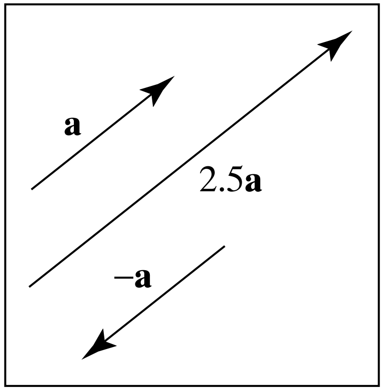


FIGURE 4.6 Scaling a vector.

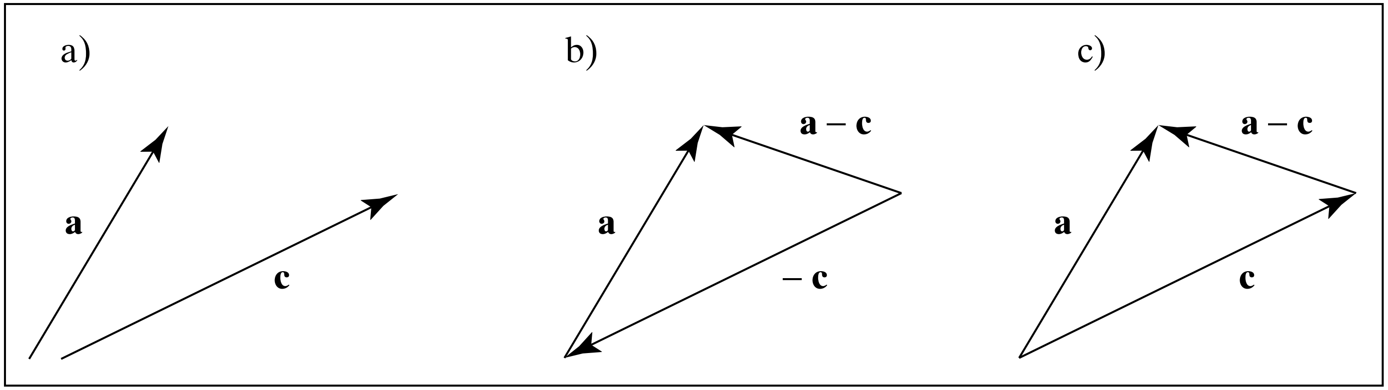


FIGURE 4.7 Subtracting vectors.

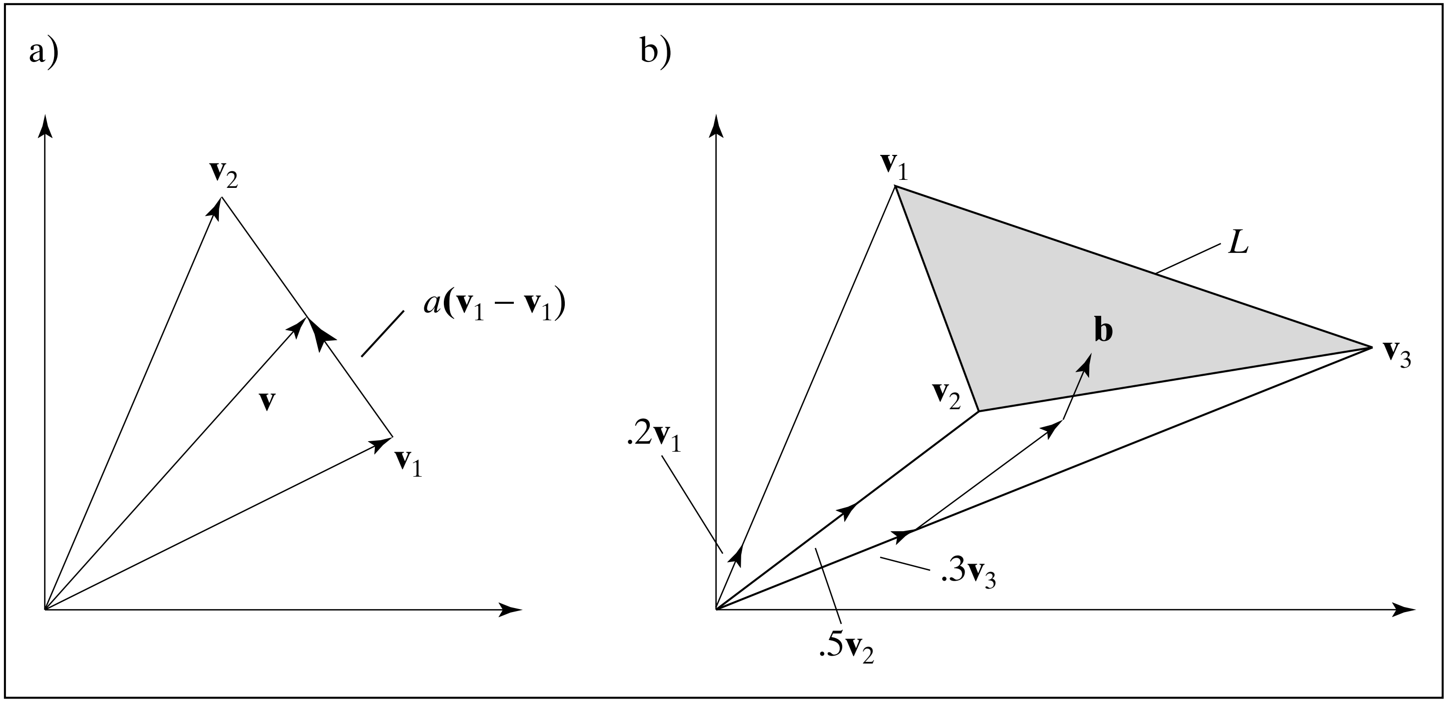


FIGURE 4.8 The set of vectors representable by convex combinations.

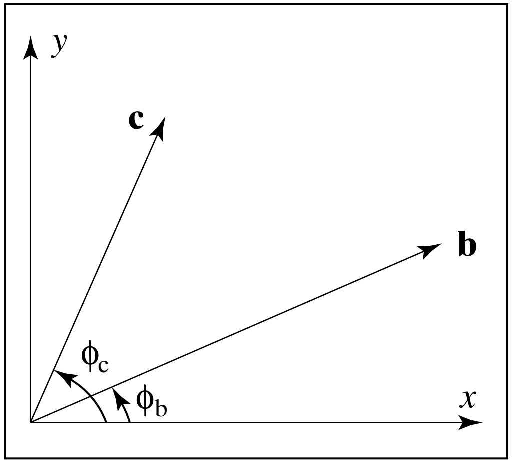


FIGURE 4.9 Finding the angle between two vectors.

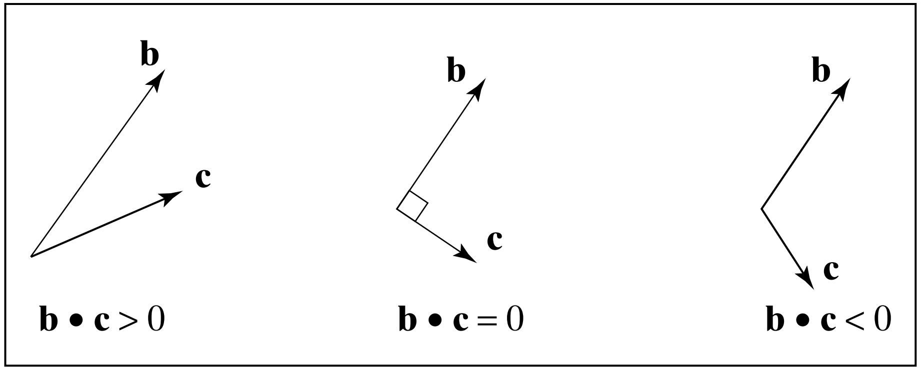


FIGURE 4.10 The sign of the dot product.

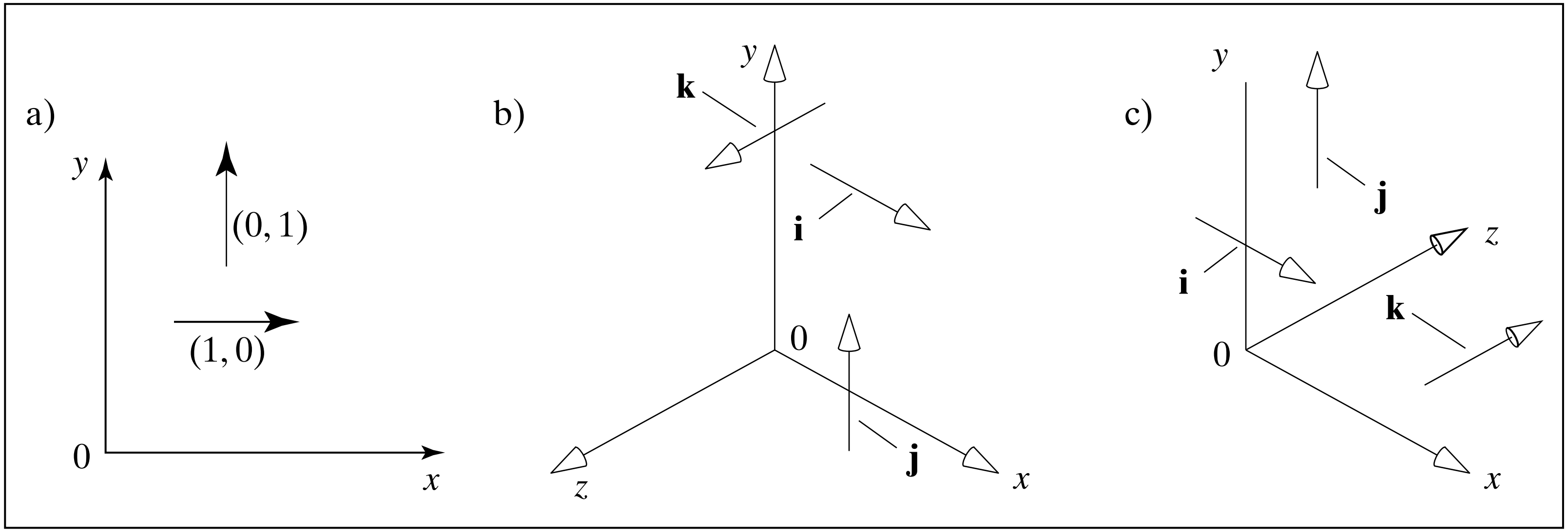
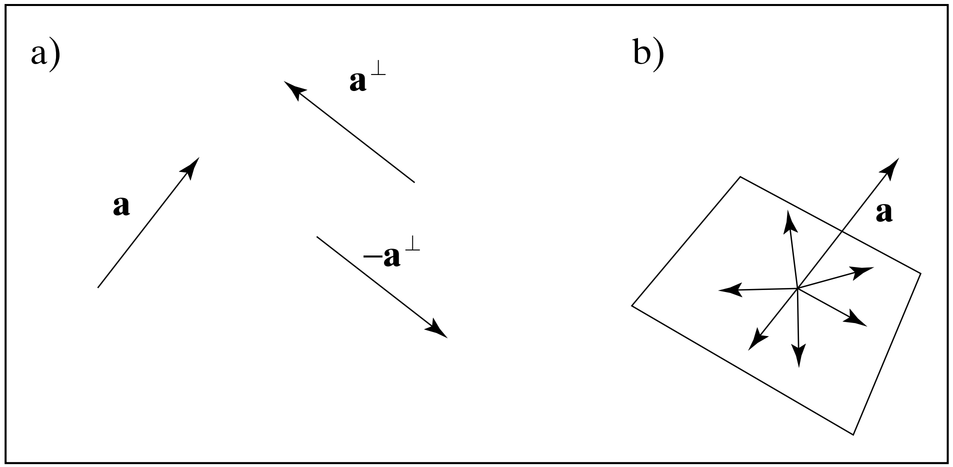


FIGURE 4.11 The standard unit vectors.

FIGURE 4.12 The vector $\mathbf{a}^{\perp}$ perpendicular to **a**.



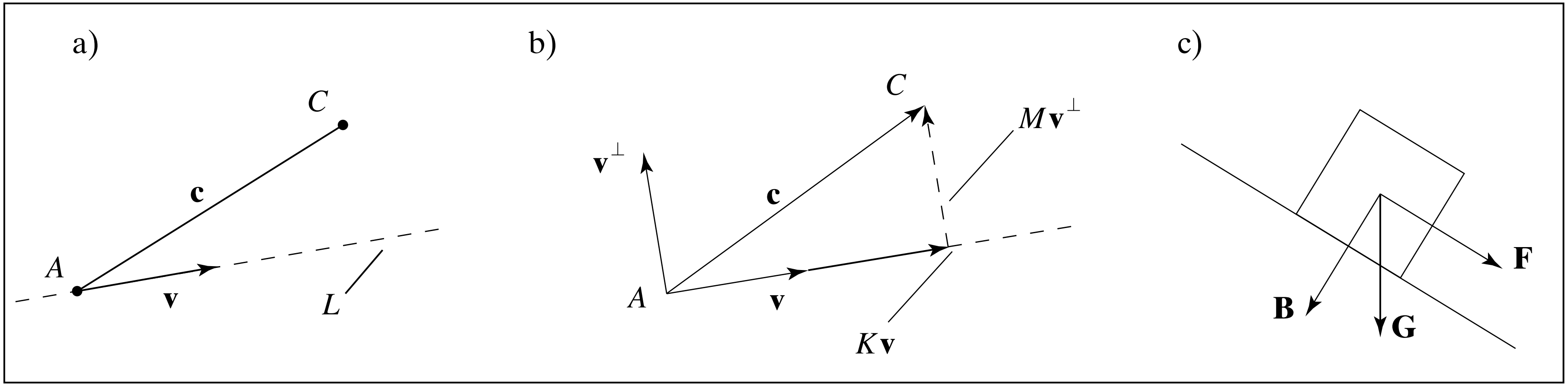


FIGURE 4.13 Resolving a vector into two orthogonal vectors.

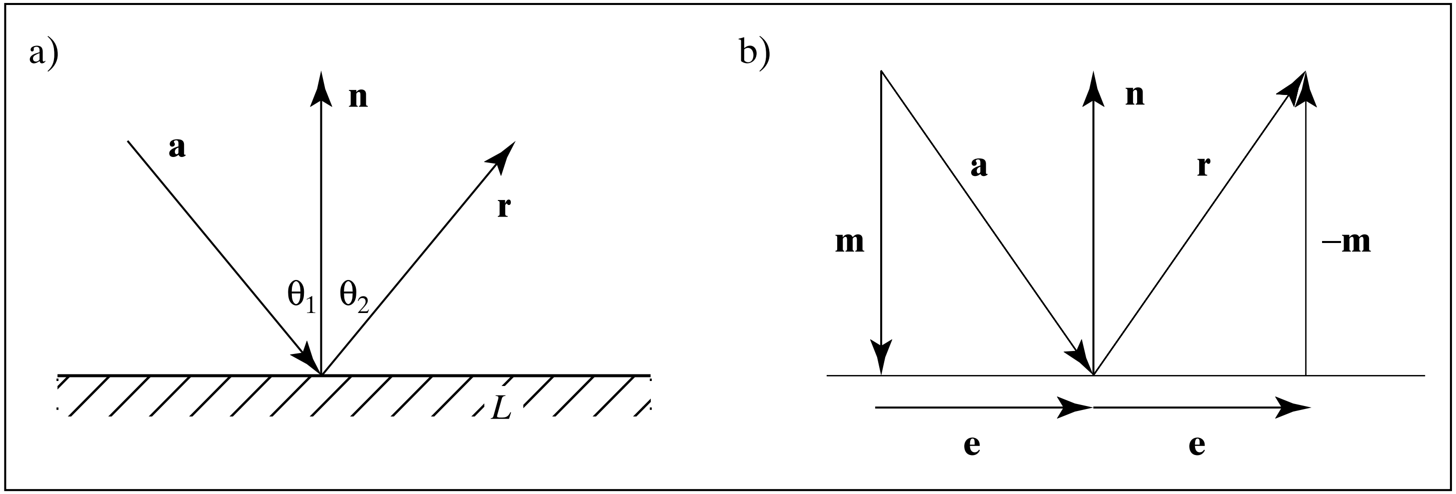
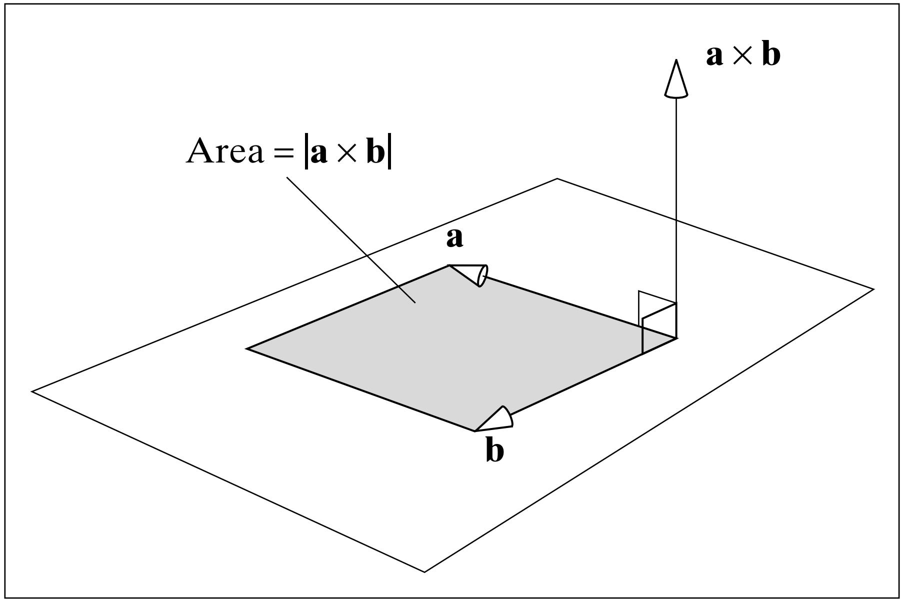


FIGURE 4.14 Reflection of a ray from a surface.

FIGURE 4.15 Interpretation of the cross product.



from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

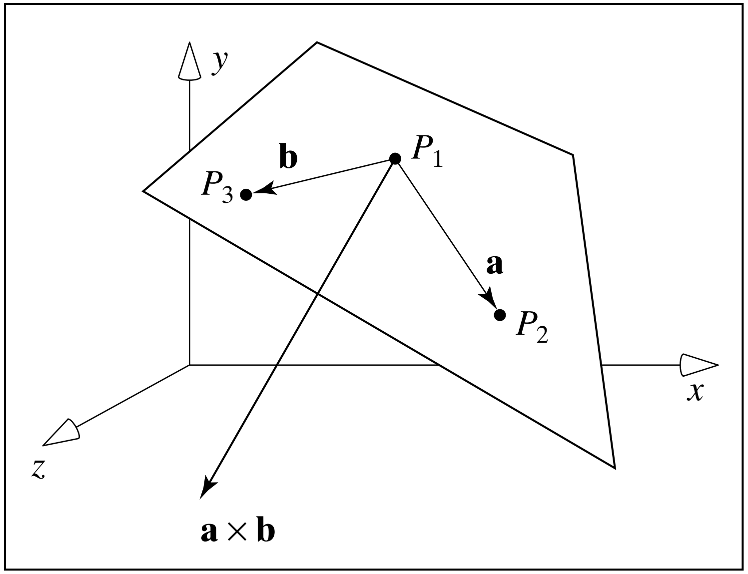
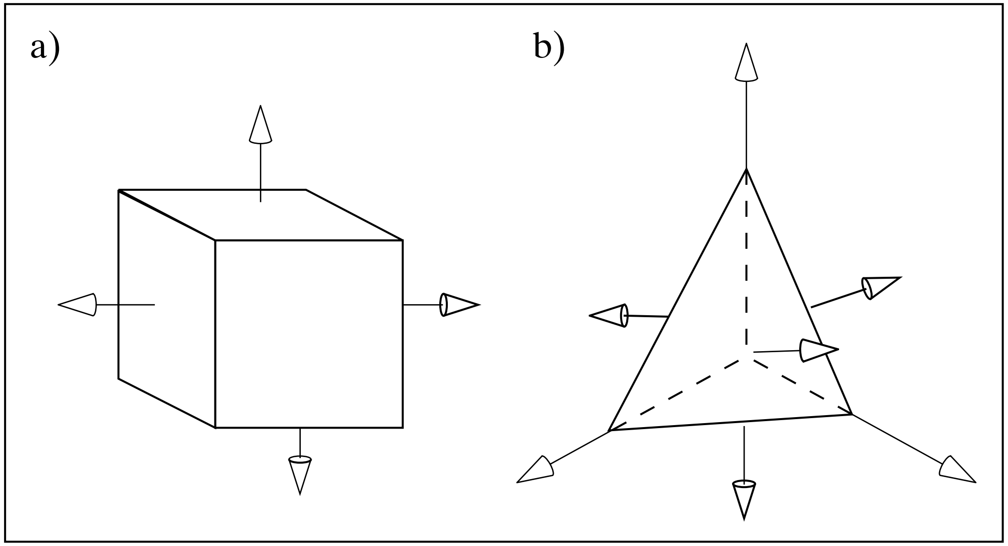


FIGURE 4.16 Finding the plane through three given points.

FIGURE 4.17 Finding the normal vectors to faces.



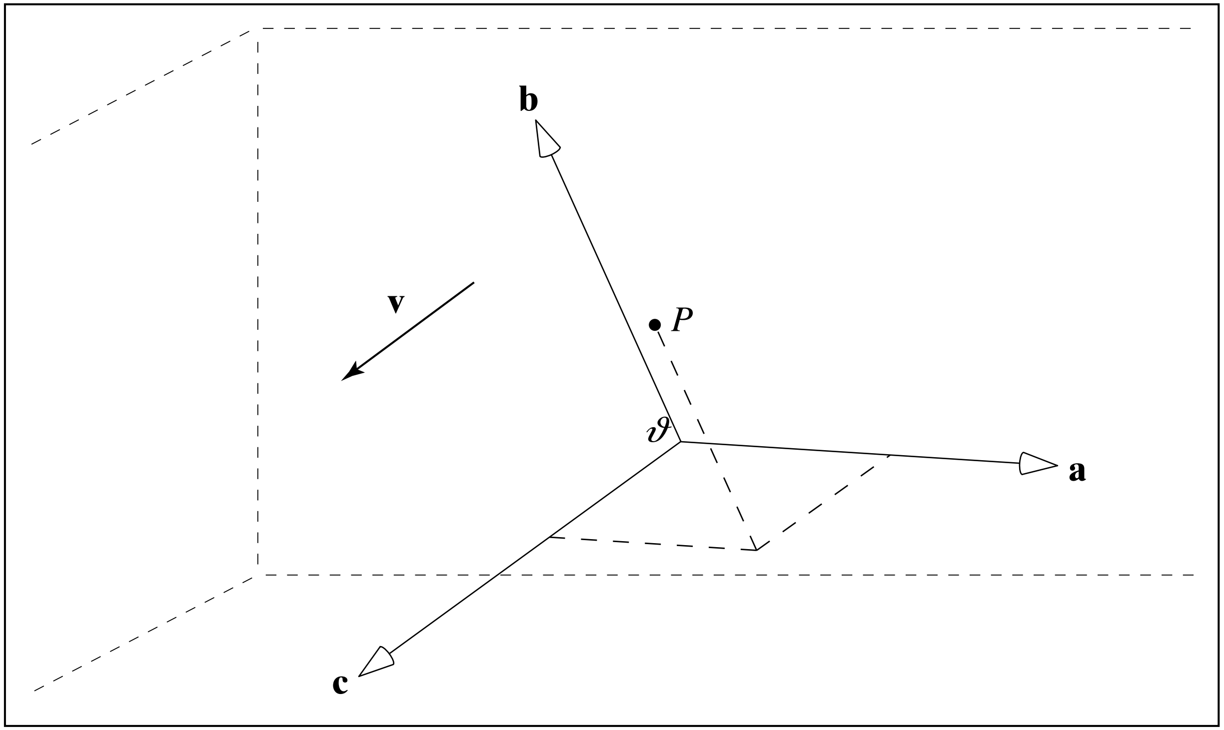
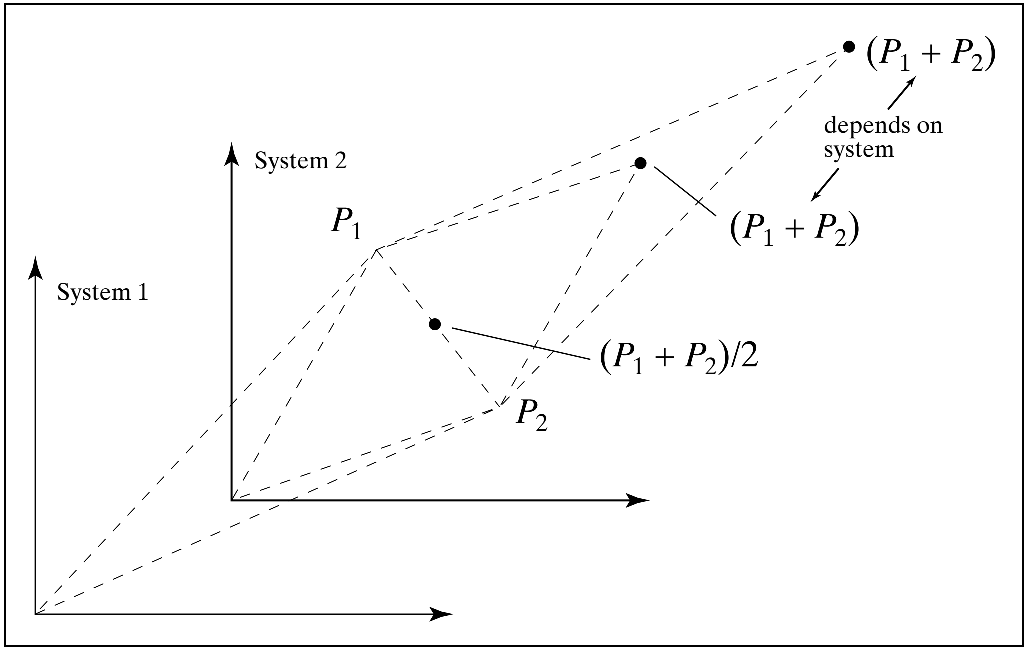


FIGURE 4.18 A coordinate frame positioned in "the world."

FIGURE 4.19 Adding points is not a valid operation.



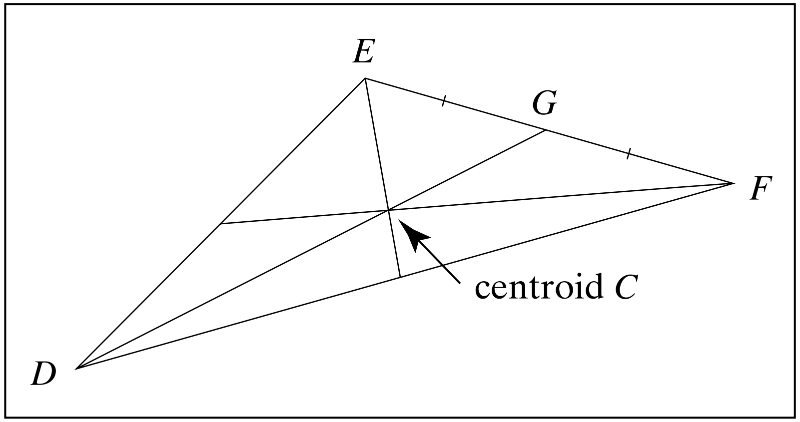


FIGURE 4.20 The centroid of a triangle as an affine combination.

```
float lerp(float a, float b, float t)
{
      return a + (b - a) * t;  // return a float
}
```

FIGURE 4.21 Linear interpolation effected by `lerp()`.

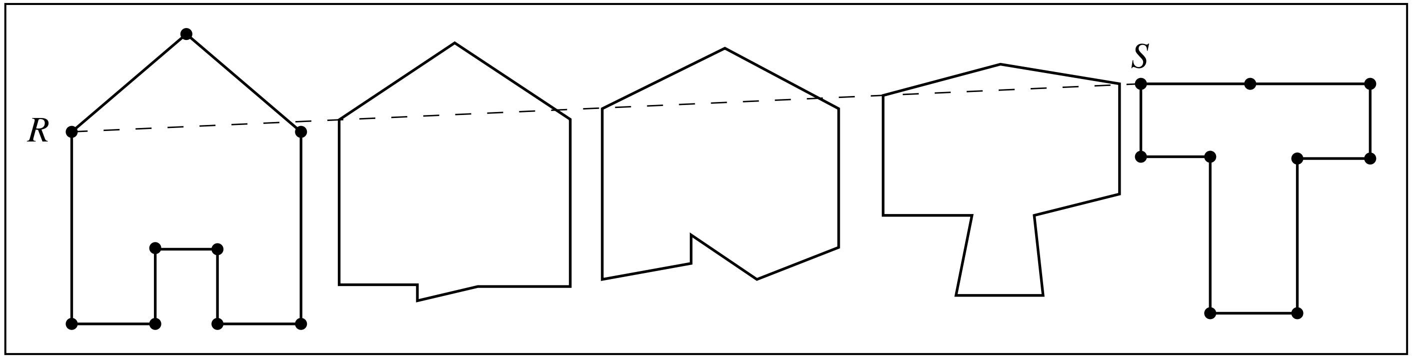


FIGURE 4.22 Tweening a *T* into a house.

```
void Canvas:: drawTween(Point2 A[], Point2 B[], int n, float t)
{   // draw the tween at time t between polylines A and B
  for(int i = 0; i < n; i++)
  {
     Point2 P;
     P = Tween(A[i], B[i],t);
     if(i == 0)  moveTo(P.x, P.y);
     else   lineTo(P.x, P.y);
  }
}
```

FIGURE 4.23 Tweening two polylines.

FIGURE 4.24 From man to woman. (Courtesy of Marc Infield.)

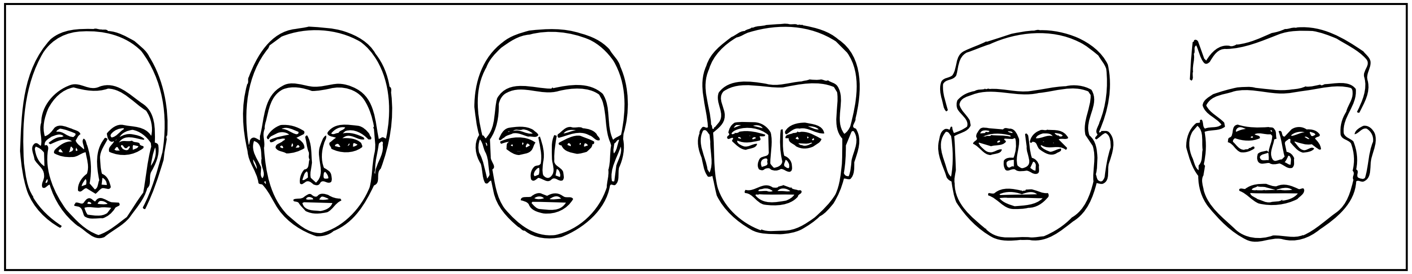


FIGURE 4.25 Face caricature: Tweening and extrapolation. (Courtesy of Susan Brennan.)

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

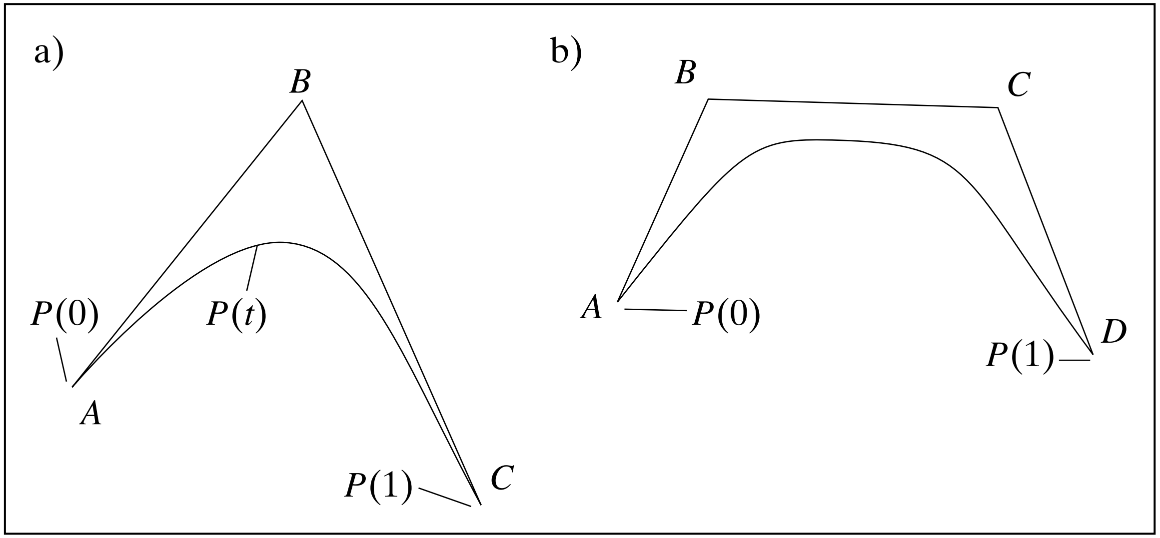


FIGURE 4.26 Bezier curves as tweening.

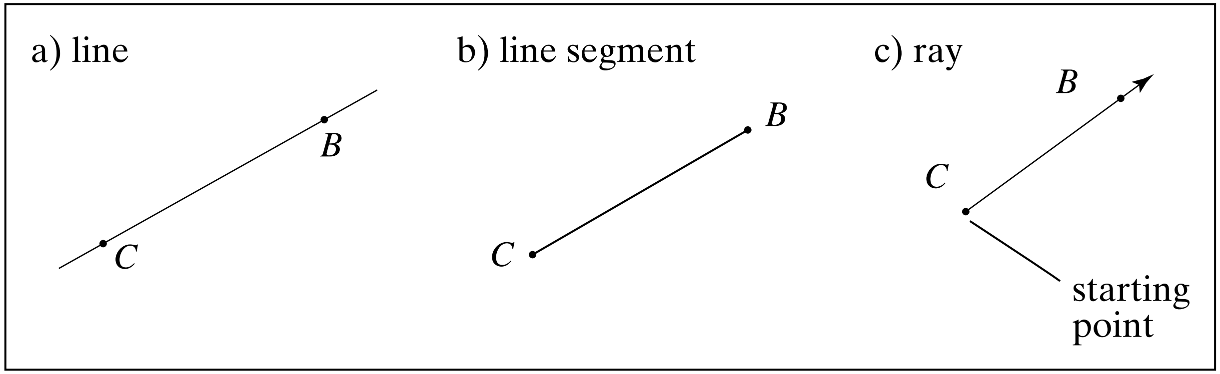
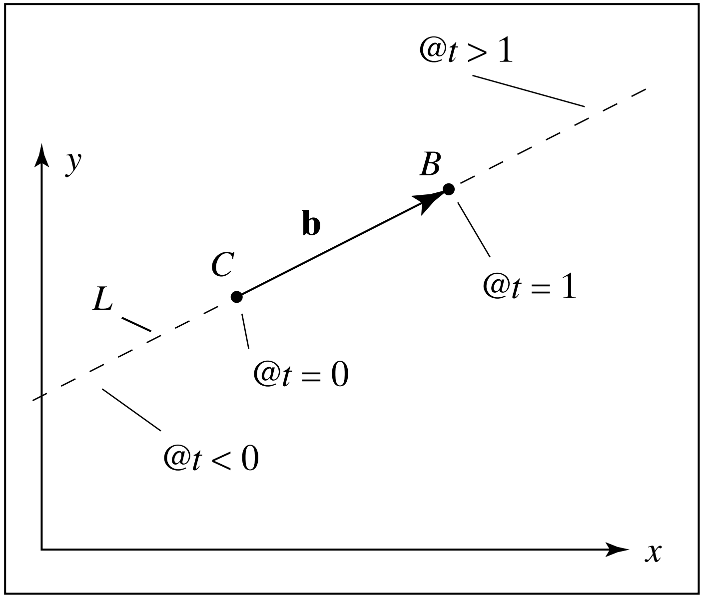


FIGURE 4.27 Lines, segments, and rays.

FIGURE 4.28 Parametric representation $L(t)$ of a line.



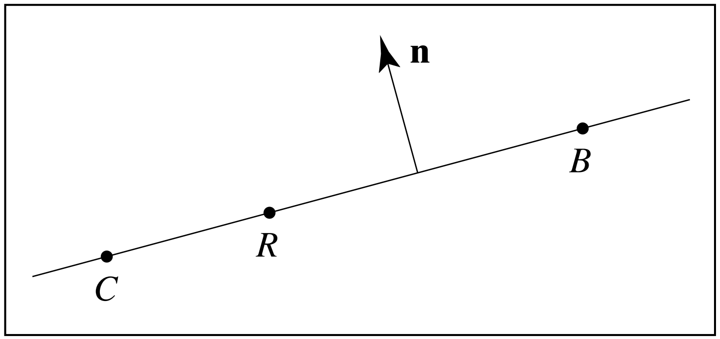


FIGURE 4.29 Finding the point normal form for a line.

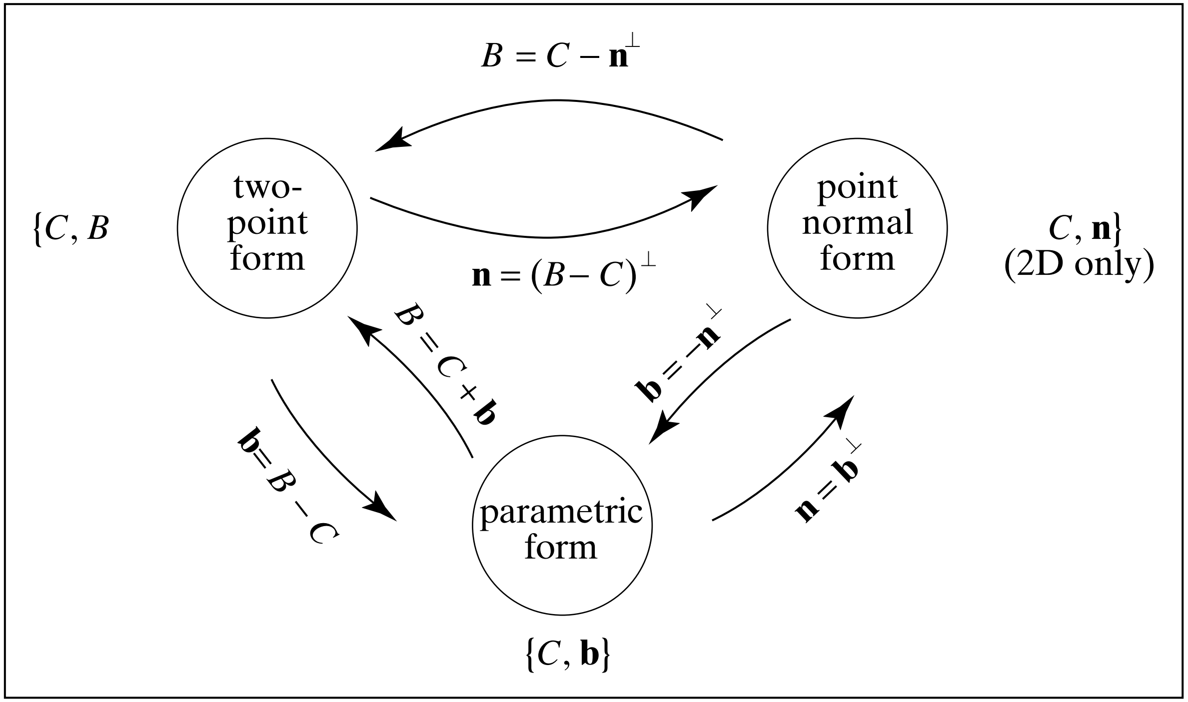


FIGURE 4.30 Moving between representations of a line.

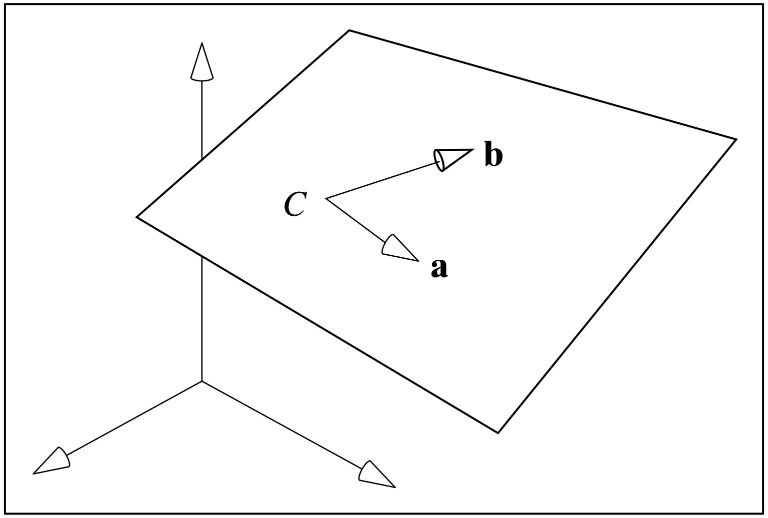


FIGURE 4.31 Defining a plane parametrically.

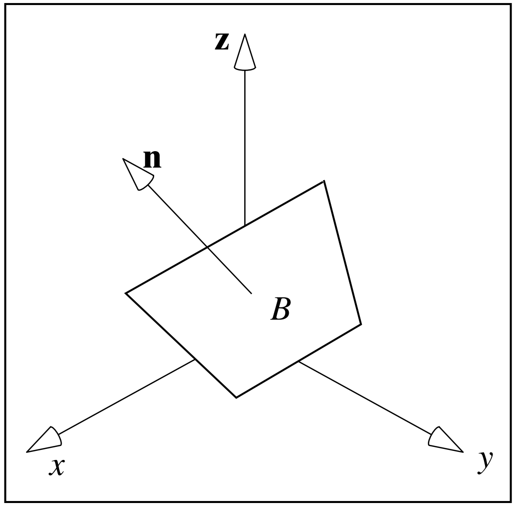


FIGURE 4.32 Determining the equation of a plane.

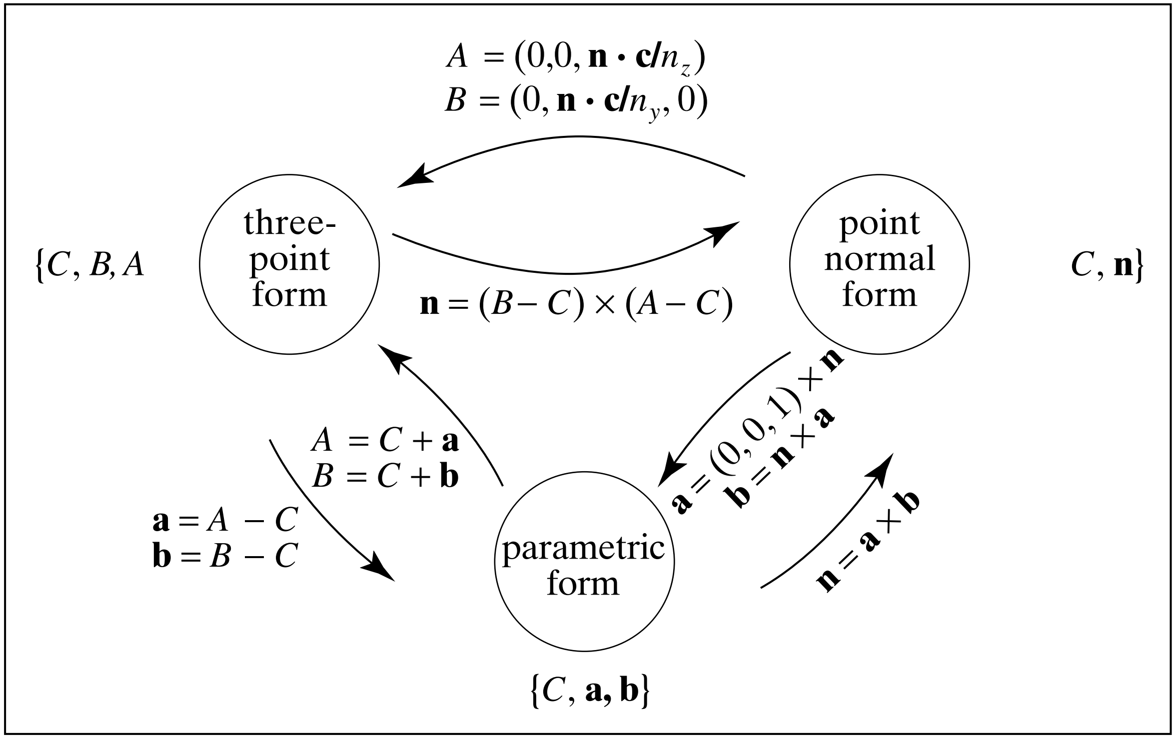


FIGURE 4.33 Moving between representations of a plane.

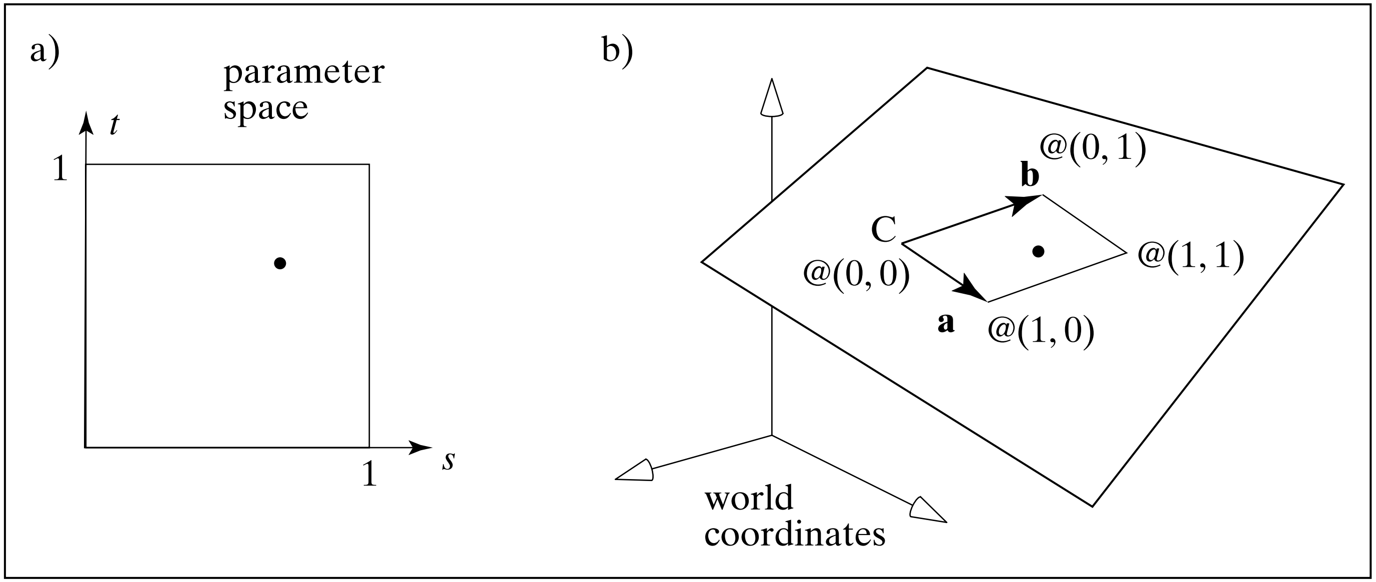


FIGURE 4.34 Mapping between two spaces to define a planar patch.

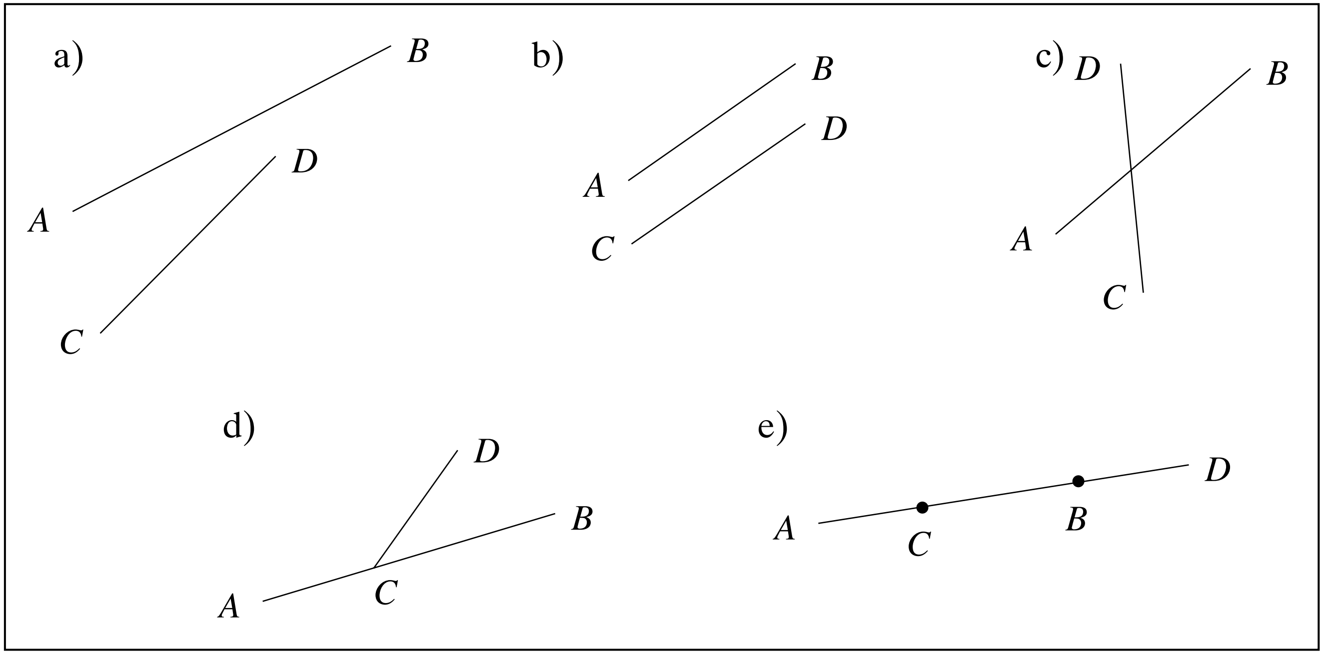
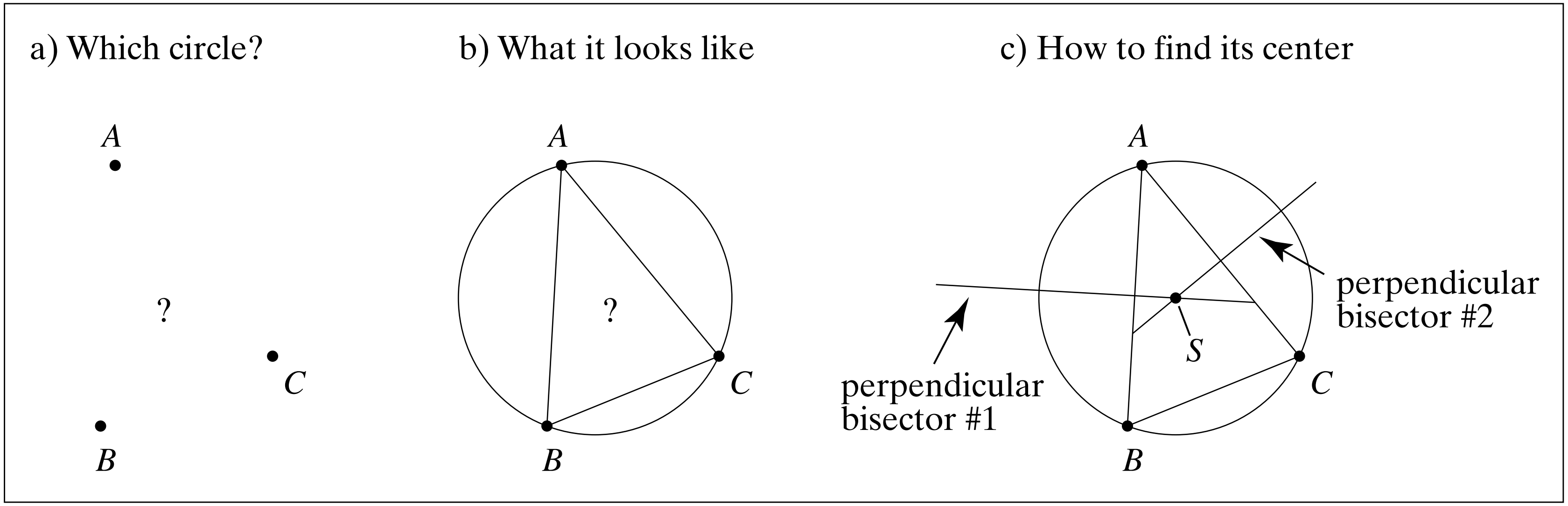


FIGURE 4.35 Many cases for two line segments.

FIGURE 4.36 Finding the excircle.



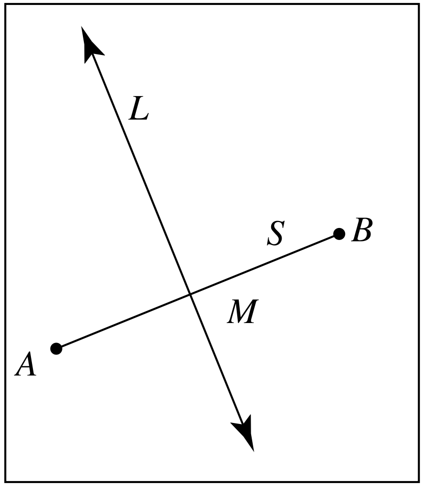


FIGURE 4.37 The perpendicular bisector of a segment.

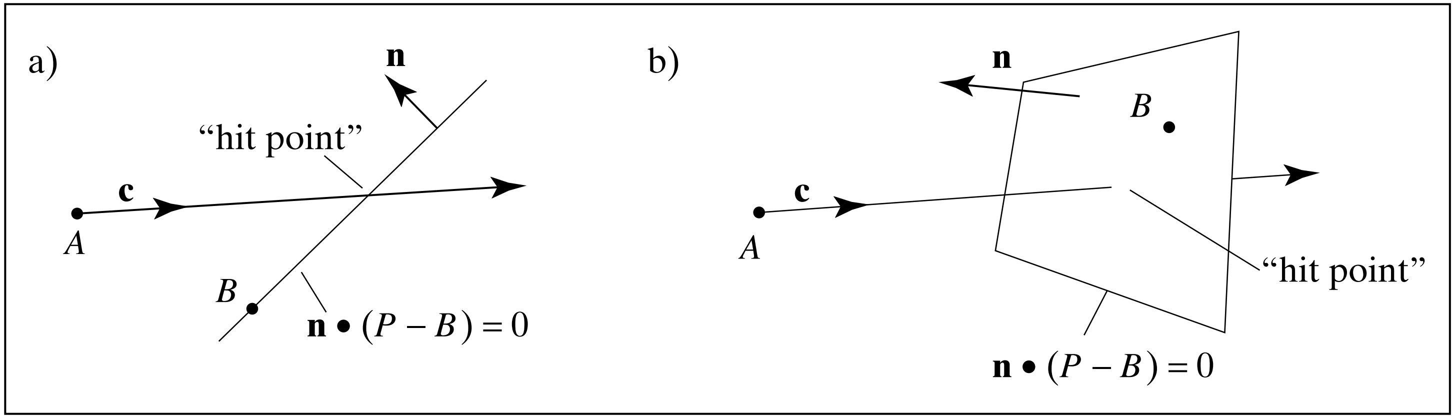


FIGURE 4.38 Where does a ray hit a line or a plane?

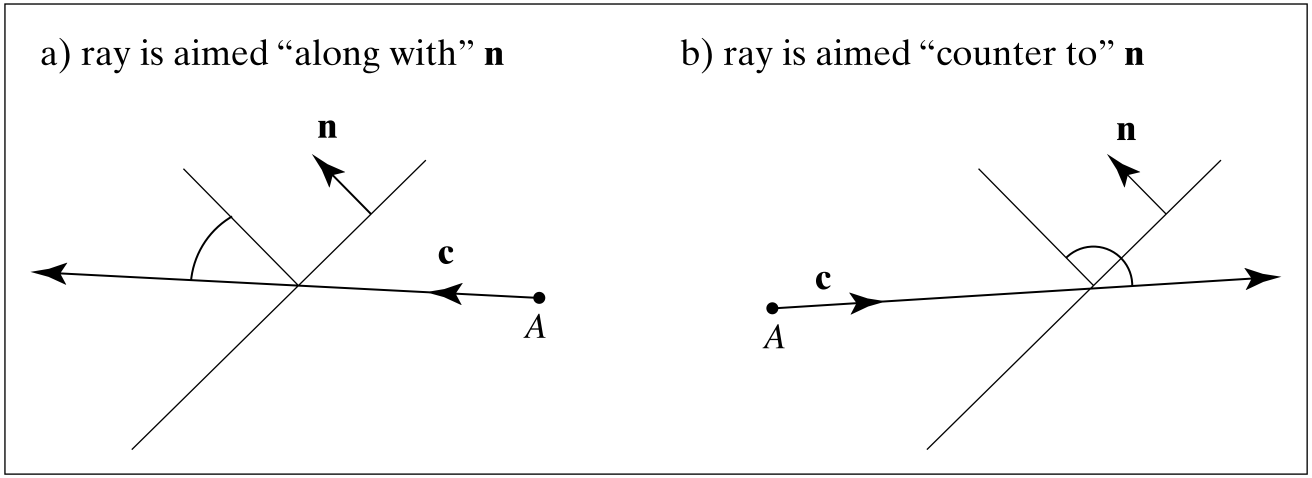


FIGURE 4.39 The direction of the ray is “along” or “against” **n**.

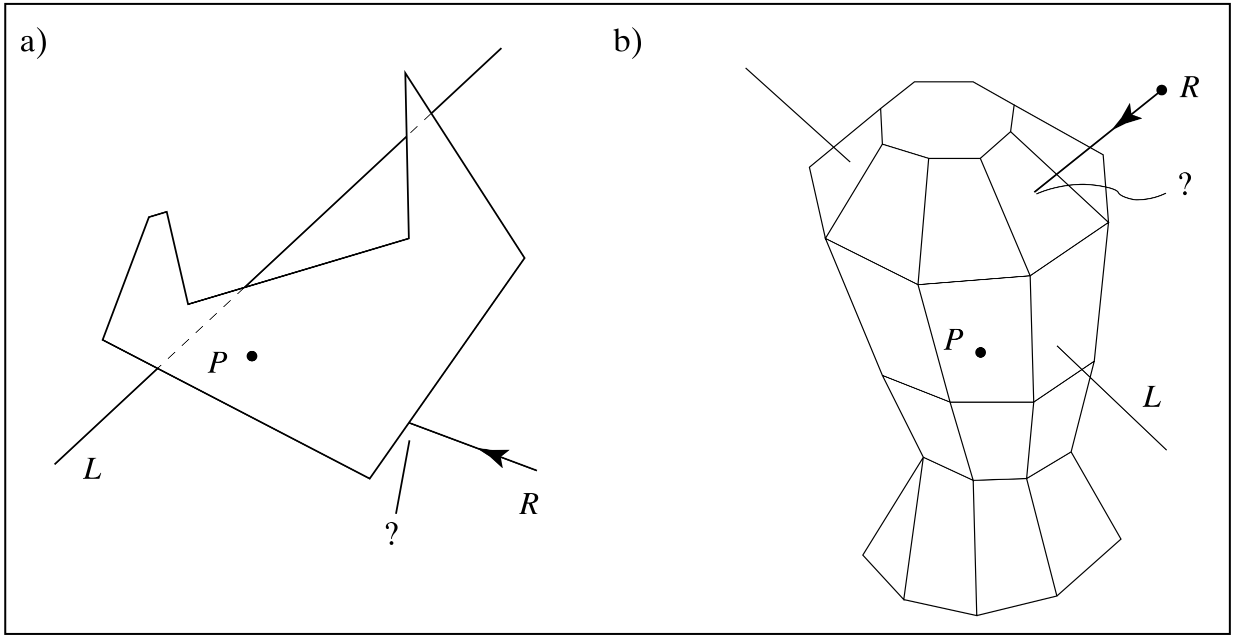


FIGURE 4.40 Intersection problems involving a line and a polygonal object.

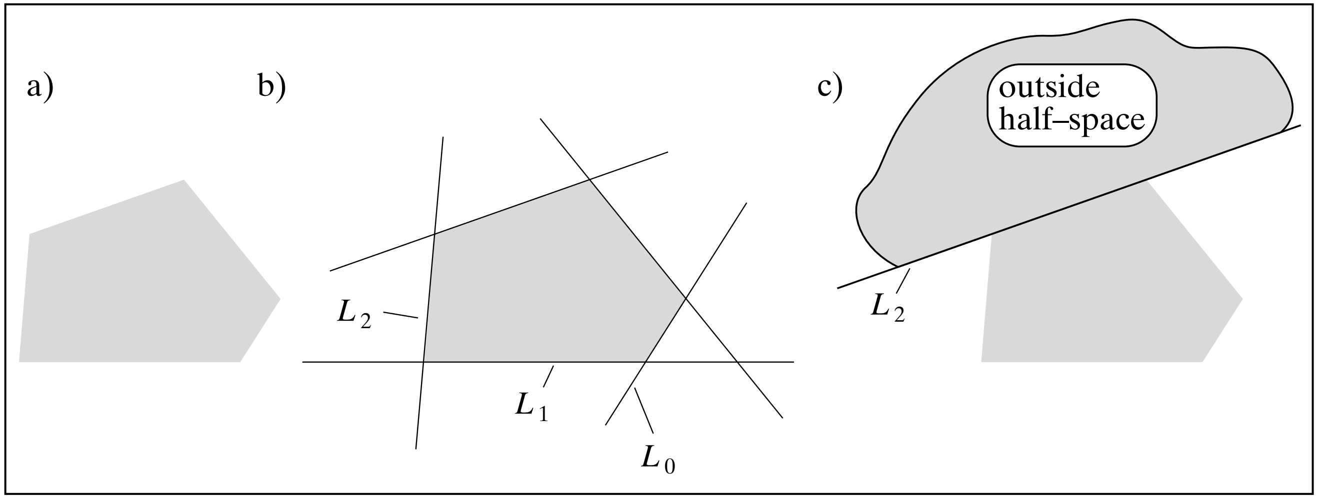


FIGURE 4.41 Convex polygons and polyhedra.

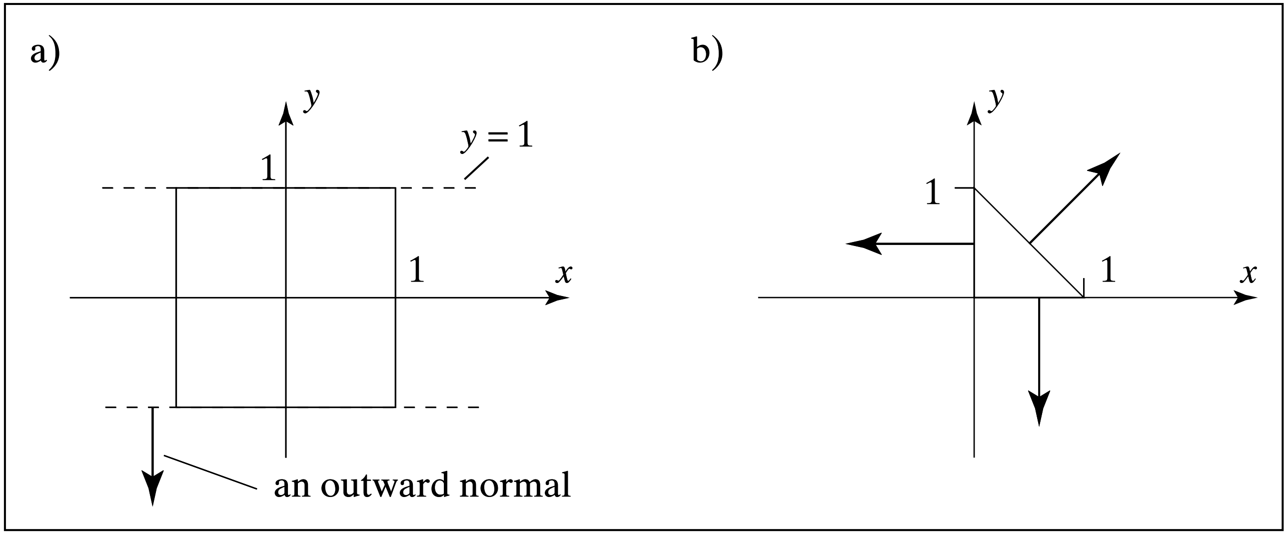
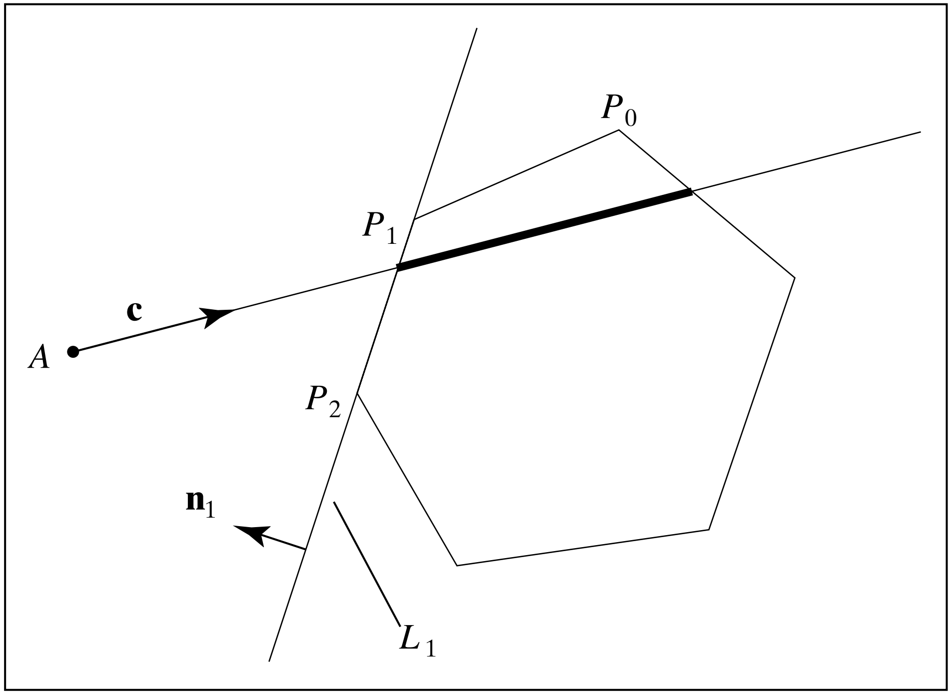


FIGURE 4.42 Examples of convex polygons.

FIGURE 4.43 Ray $A + \mathbf{c}t$ intersecting a convex polygon.



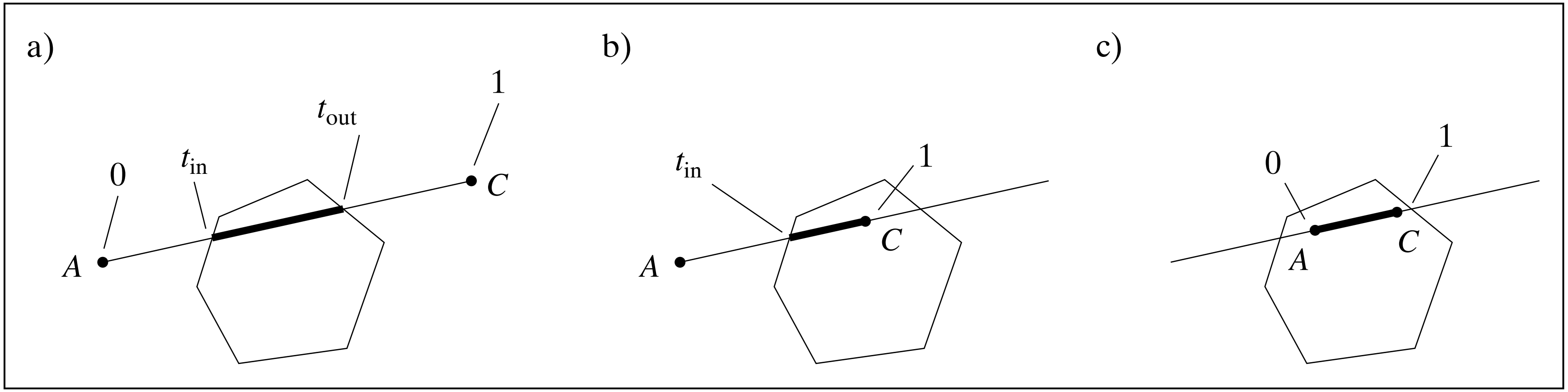


FIGURE 4.44 A segment clipped by a polygon.

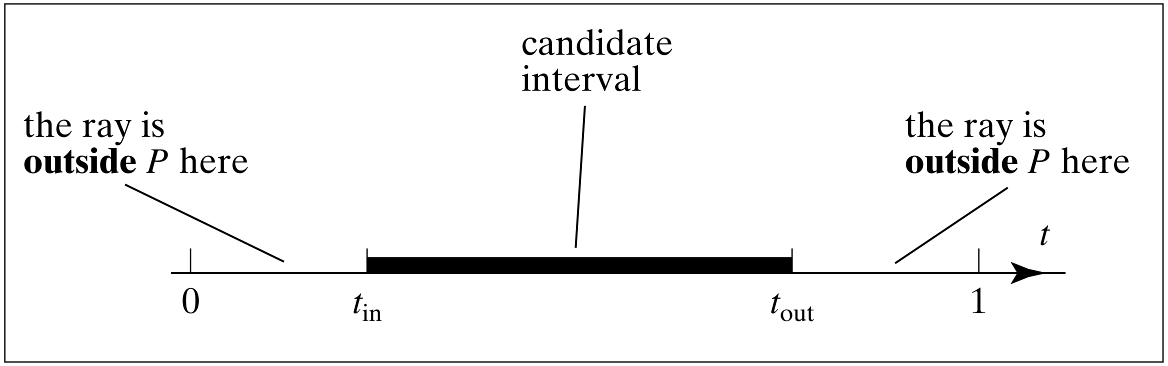


FIGURE 4.45 The candidate interval for a hit.

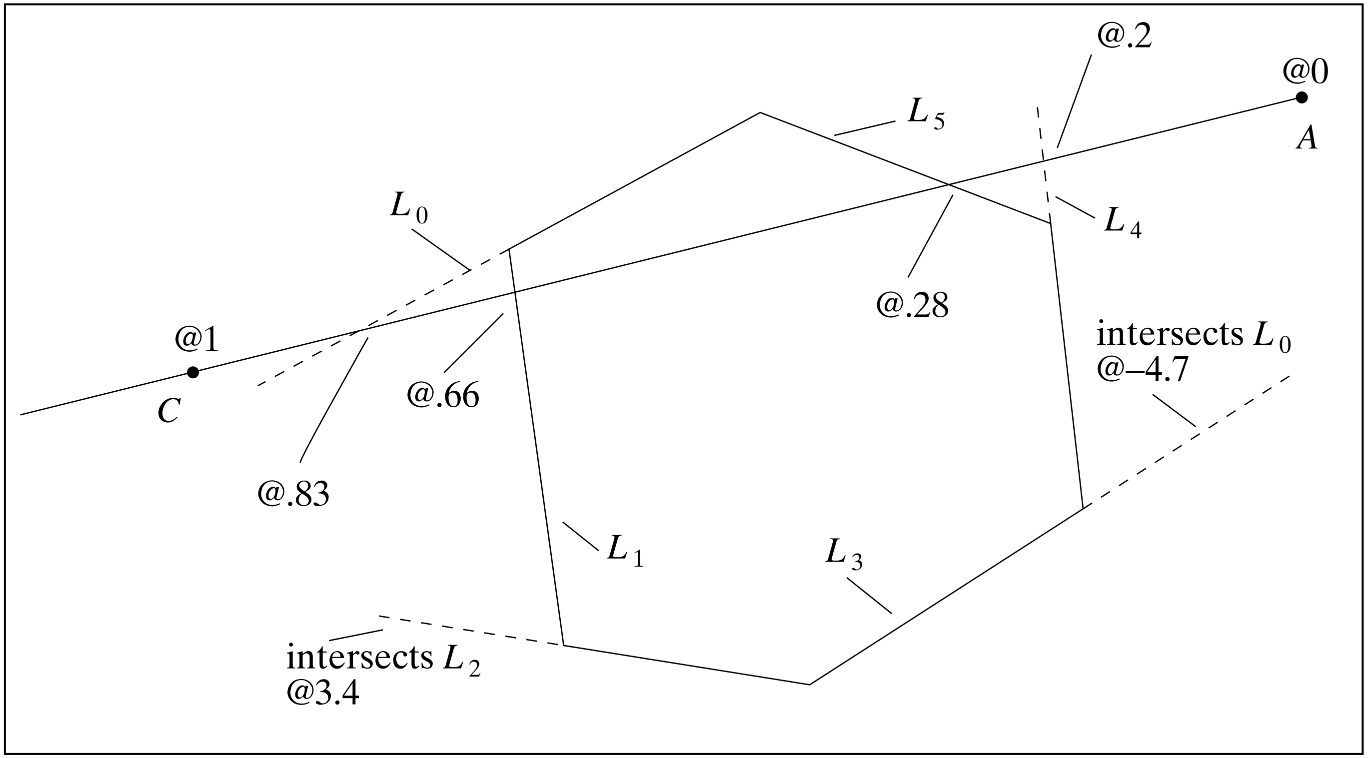


FIGURE 4.46 Testing when a ray lies inside a convex polygon.

Line test	t_{in}	t_{out}
0	0	0.83
1	0	0.66
2	0	0.66
3	0	0.66
4	0.2	0.66
5	0.28	0.66

FIGURE 4.47 Updates on the values of t_{in} and t_{out}.

```
int CyrusBeckClip(LineSegment& seg, LineList L)
{
 double numer, denom; // used to find hit time for each line
 double tIn = 0.0, tOut = 1.0;
 Vector2 c, tmp;
 form vector:  c = seg.second - seg .first
 for(int i = 0; i < L.num; i++) // chop at each bounding line
 {
    form vector tmp = L.line[i].pt - first
    numer = dot(L.line[i].norm, tmp);
    denom = dot(L.line[i].norm, c);
    if(!chopCI(tIn, tOut numer, denom,)) return 0; // early out
 }
 // adjust the endpoints of the segment; do second one 1st.
 if (tOut < 1.0 ) // second endpoint was altered
 {
     seg.second.x  = seg.first.x + c.x * tOut;
     seg.second.y  = seg.first.y + c.y * tOut;
 }
 if (tIn > 0.0)  // first endpoint was altered
 {
     seg.first.x  = seg.first.x + c.x * tIn;
     seg.first.y  = seg.first.y + c.y * tIn;
}
     return 1; // some segment survives
}
```

FIGURE 4.48 Pseudocode for Cyrus–Beck clipper for a convex polygon, 2D case.

```
int chopCI(double& tIn, double& tOut, double numer, double denom)
{
  double tHit;
  if(denom < 0)      // ray is entering
  {
     tHit = numer / denom;
     if(tHit > tOut) return 0;       // early out
     else if(tHit > tIn) tIn = tHit; // take larger t
  }
  else if(denom > 0)          // ray is exiting
  {
     tHit = numer / denom;
     if(tHit < tIn) return 0;      // early out
     if(tHit < tout) tOut = tHit; // take smaller t
  }
  else              // denom is 0: ray is parallel
  if(numer <= 0) return 0;   // missed the line

  return 1; // CI is still non-empty
}
```

FIGURE 4.49 Clipping against a single bounding line.

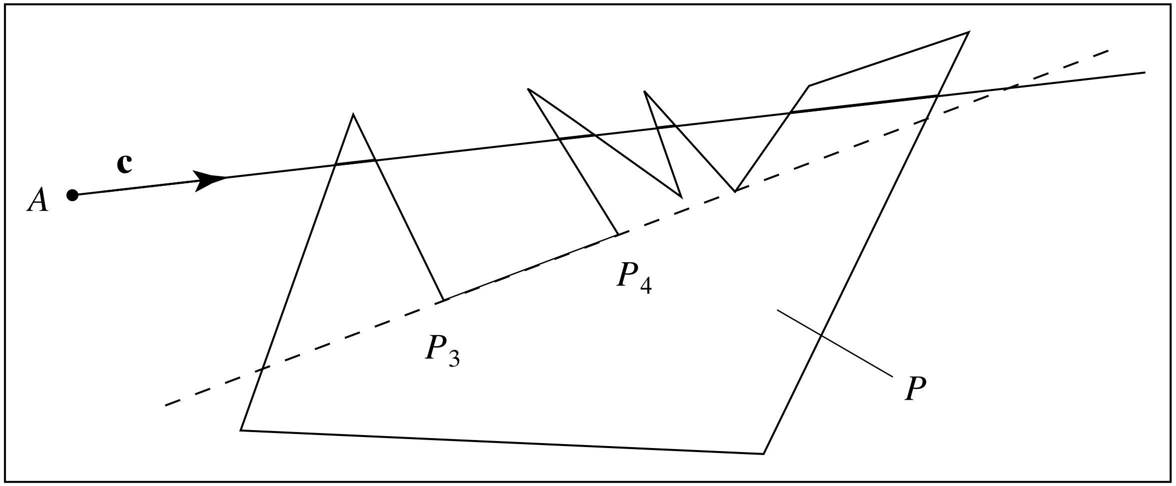


FIGURE 4.50 Where is a ray inside an arbitrary polygon P?

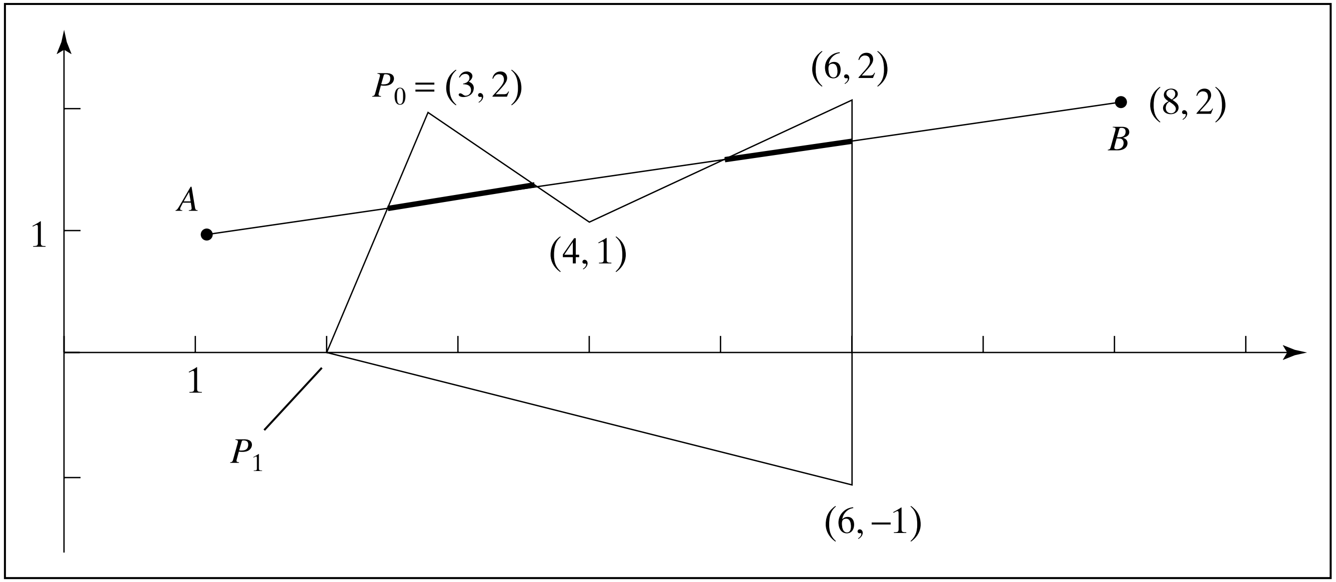
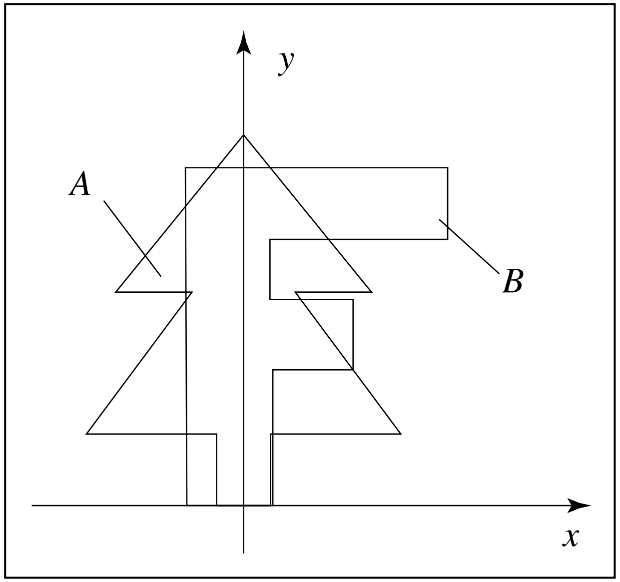


FIGURE 4.51 Clipping a line against a polygon.

FIGURE 4.52 Tweening two polylines.



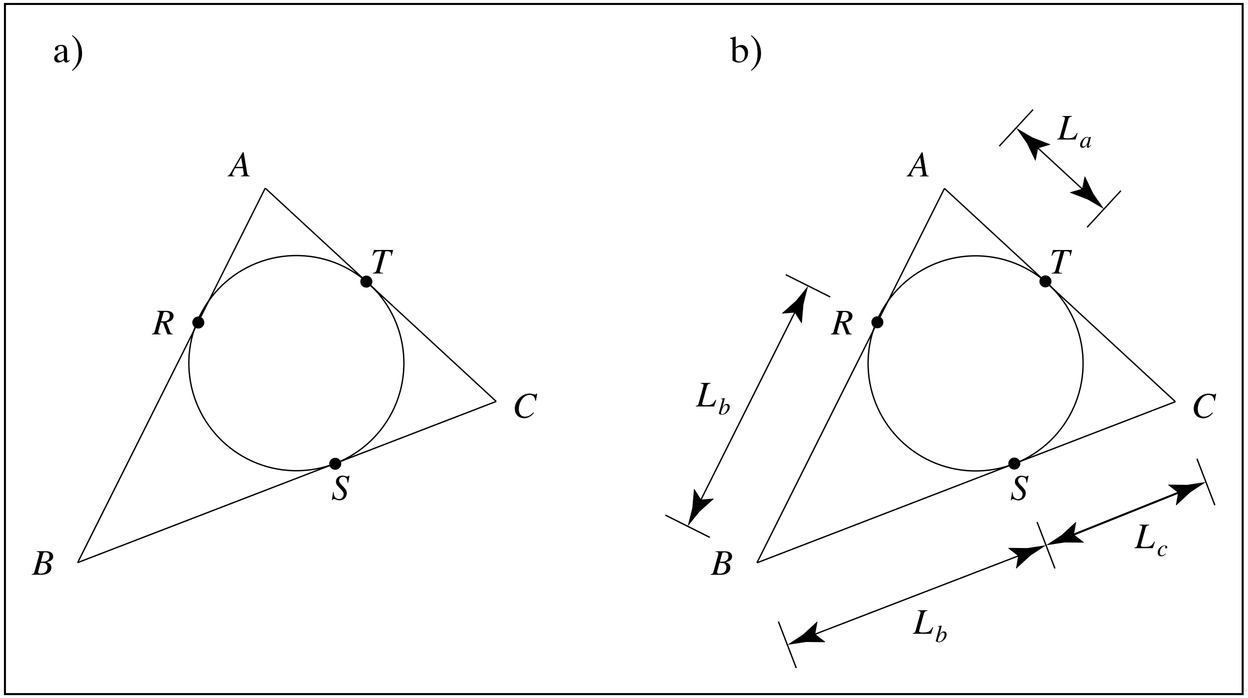


FIGURE 4.53 The inscribed circle of *ABC* is the excircle of *RST* .

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

FIGURE 4.54 The nine-point circle.

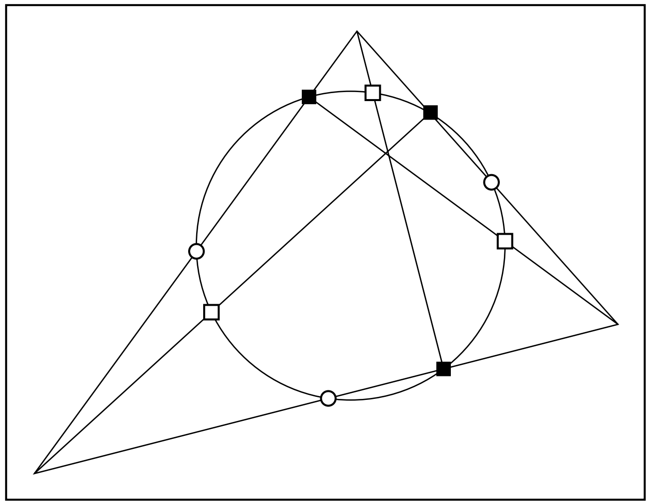
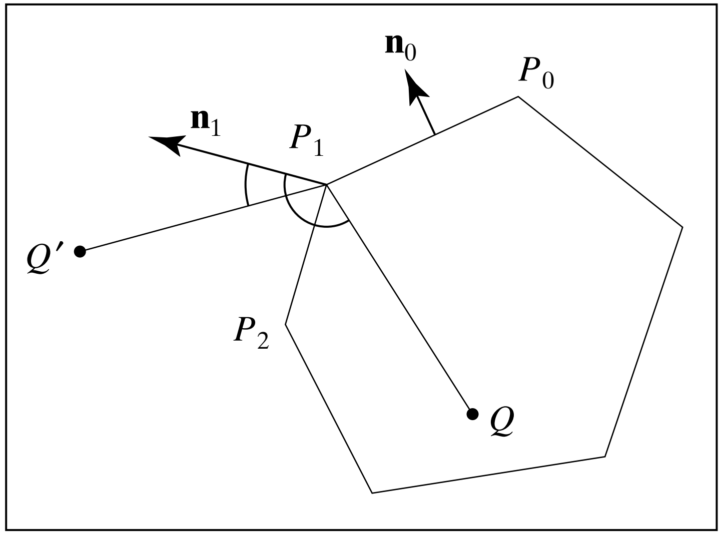


FIGURE 4.55 Is point Q inside polygon P?



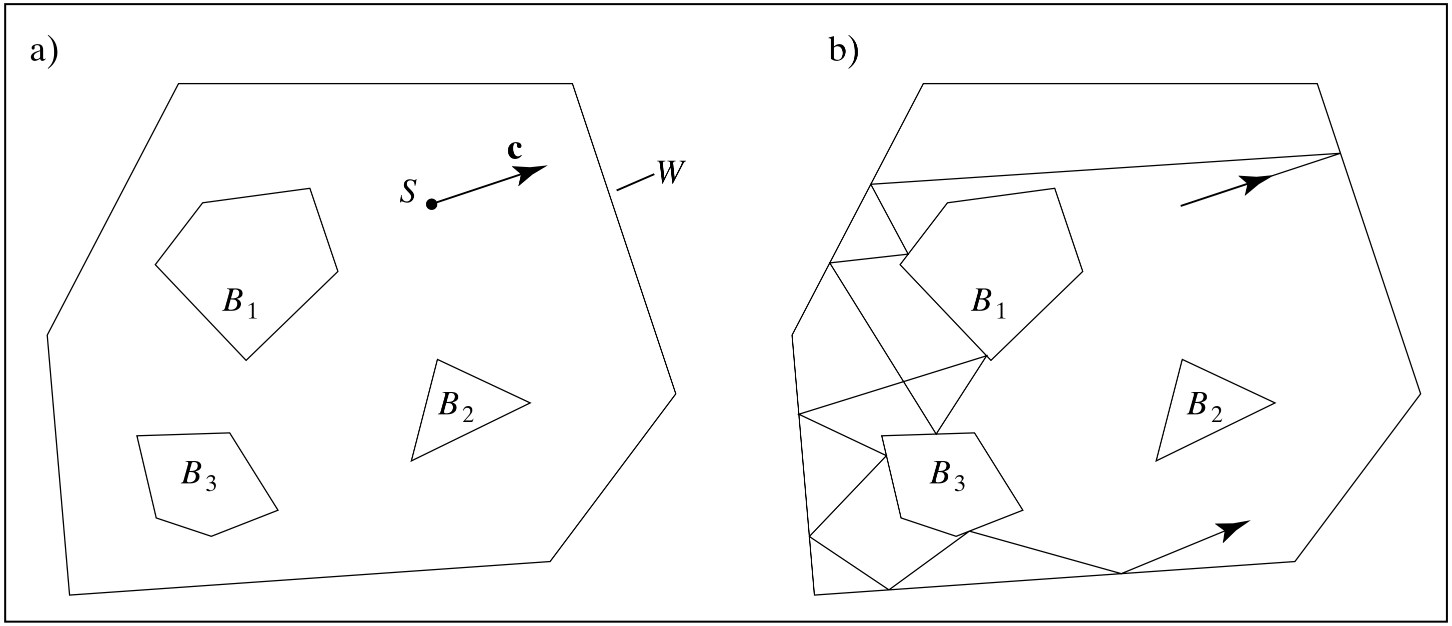


FIGURE 4.56 A 2D ray-tracing experiment.

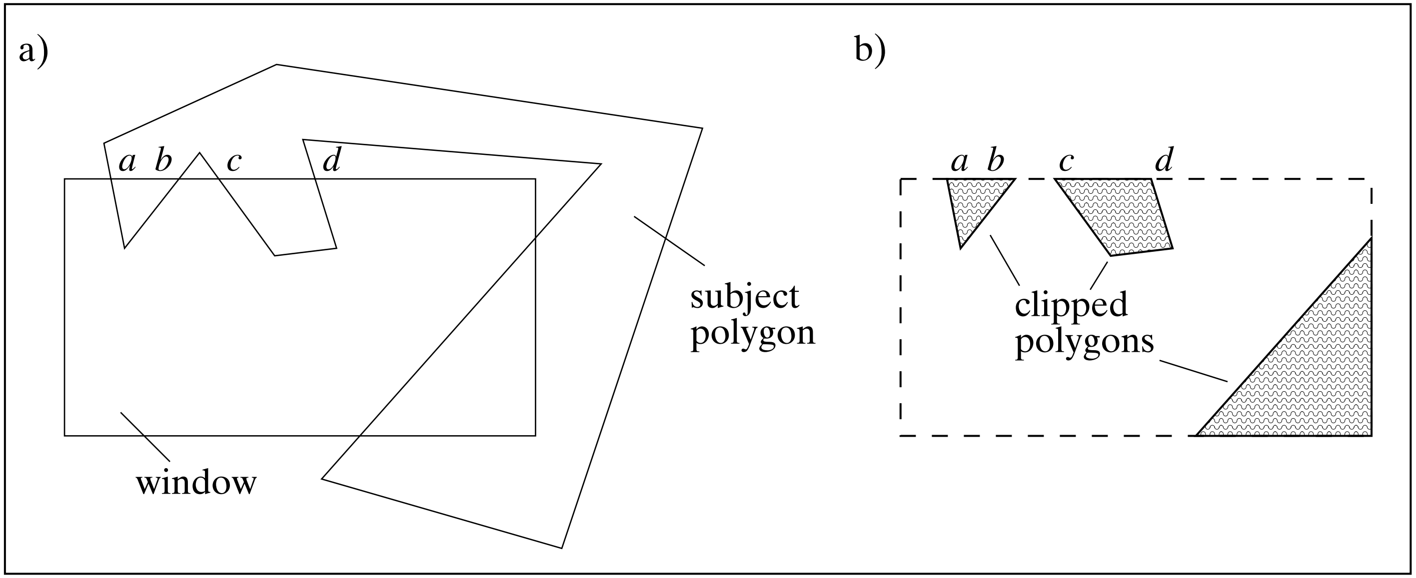


FIGURE 4.57 Clipping a polygon against a polygon.

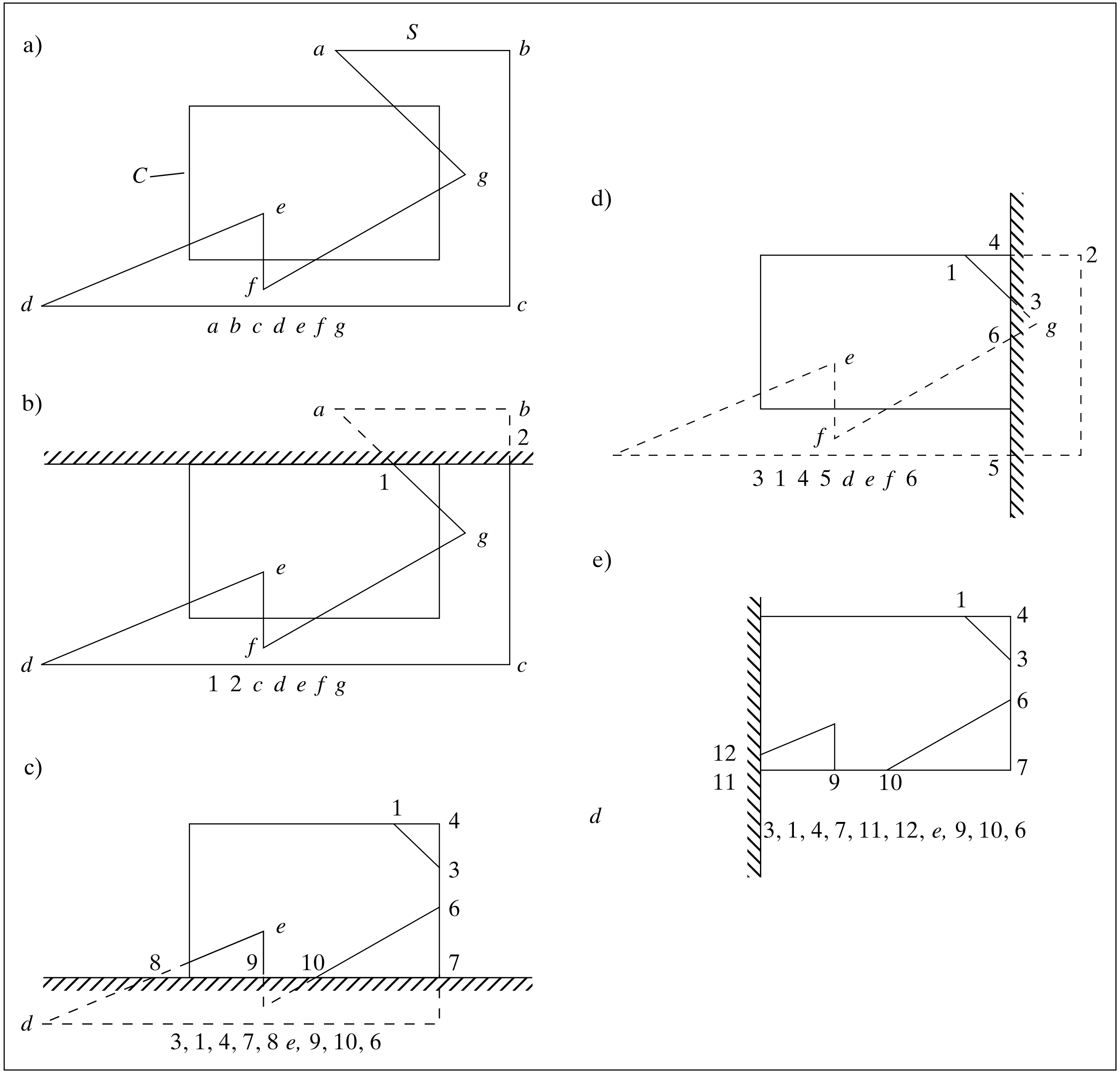
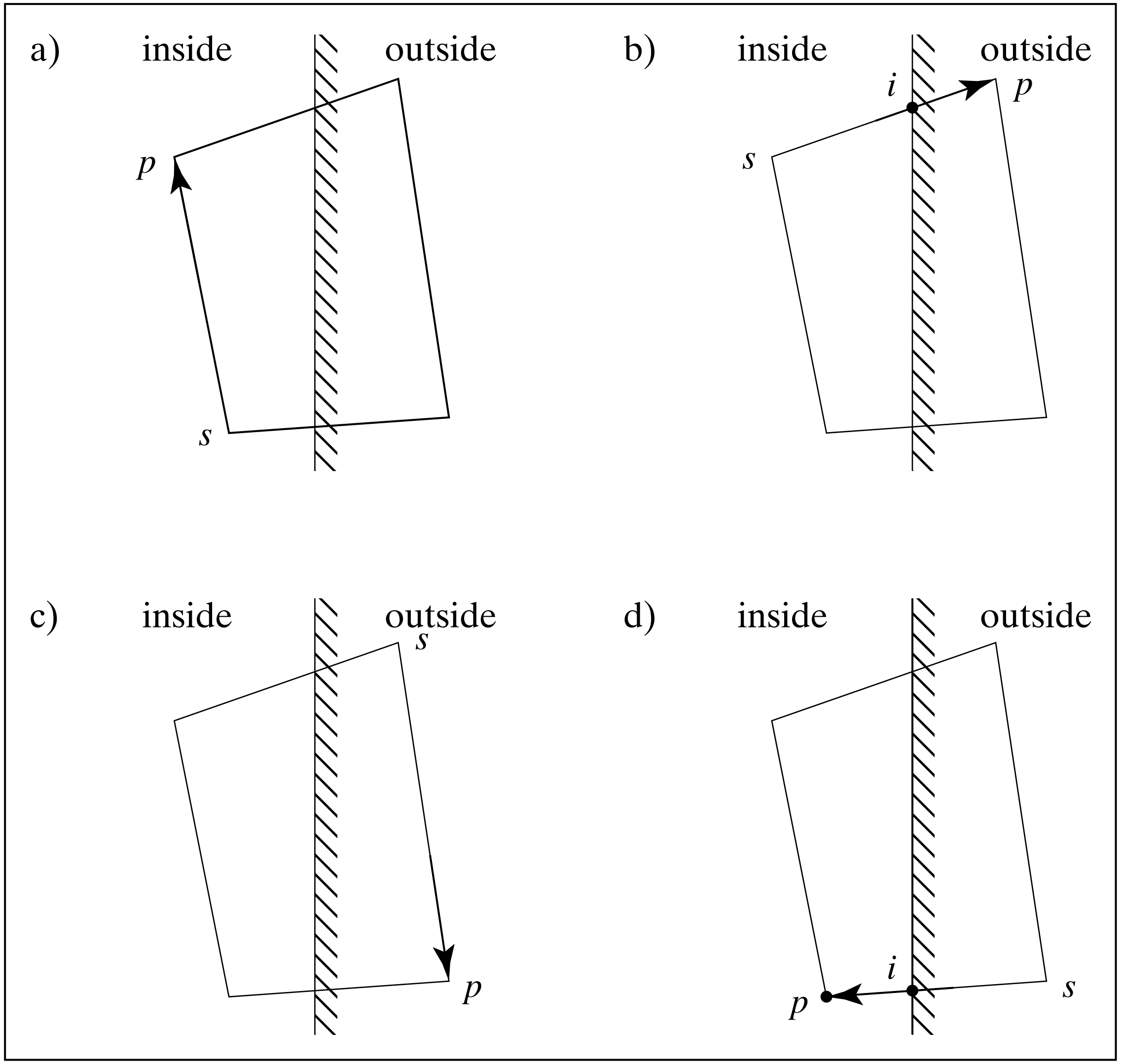


FIGURE 4.58
Sutherland–Hodgman polygon clipping.

FIGURE 4.59 Four cases for each edge of *S*.



from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

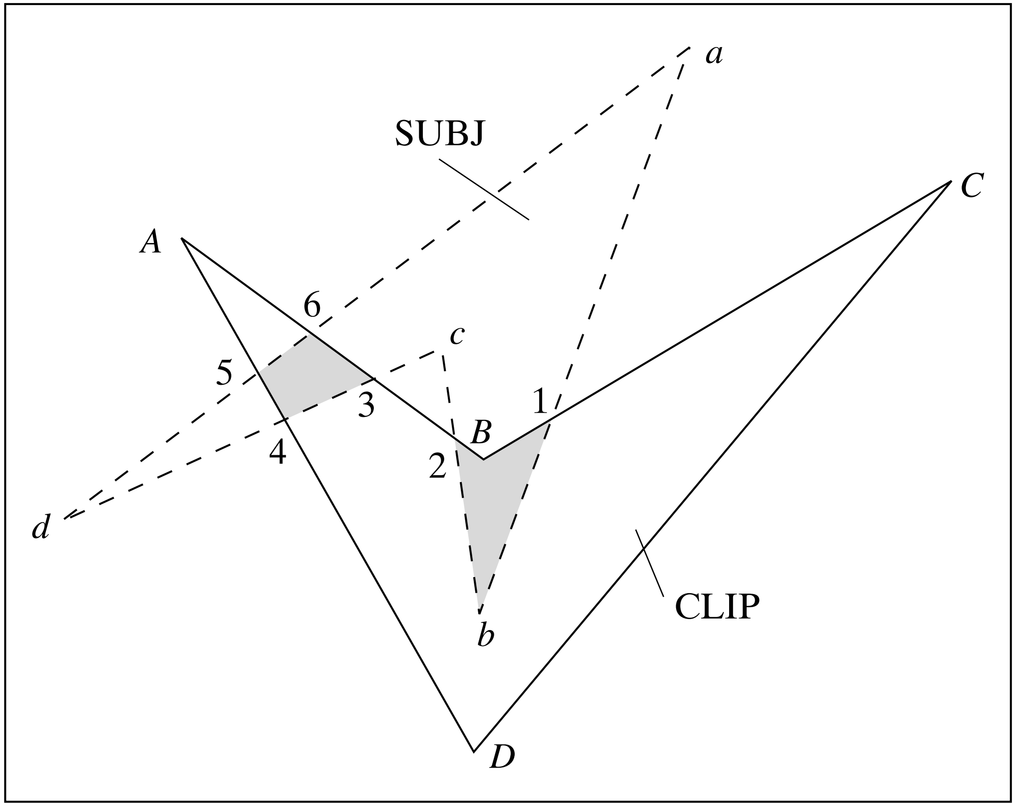


FIGURE 4.60 Weiler–Atherton clipping.

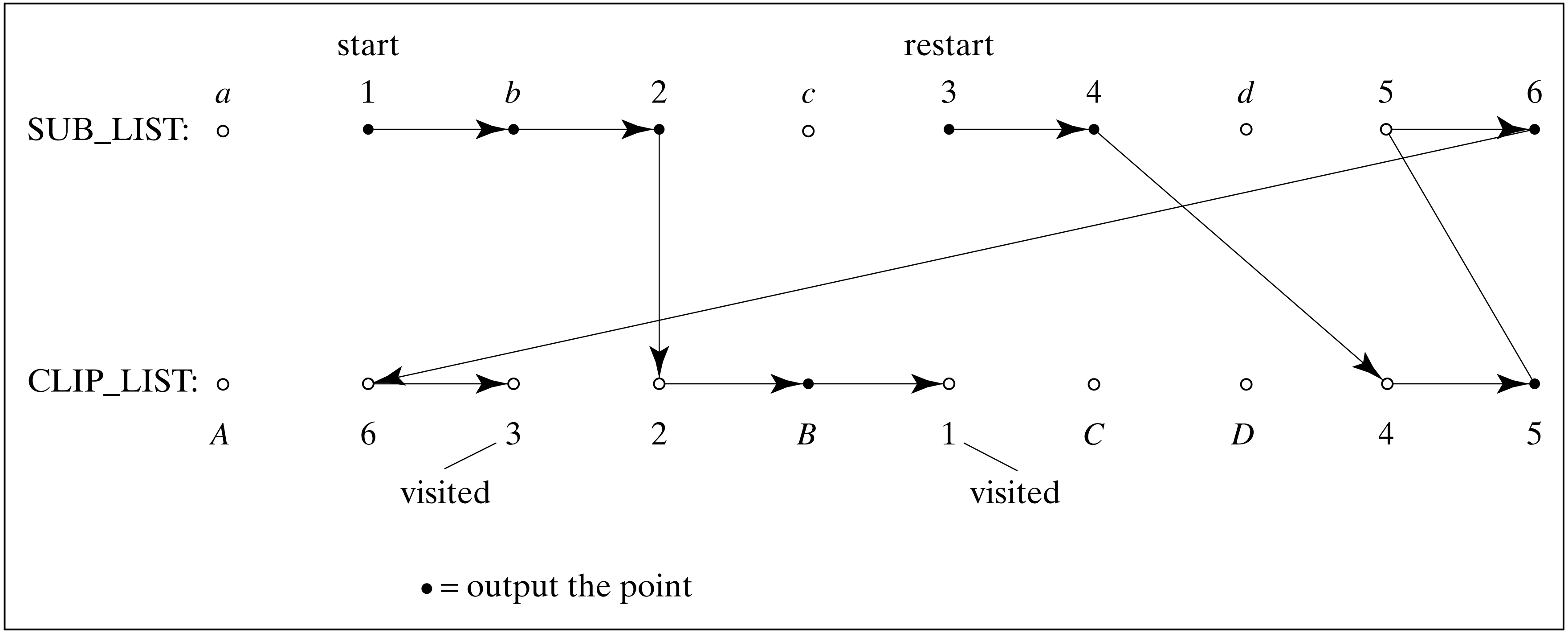
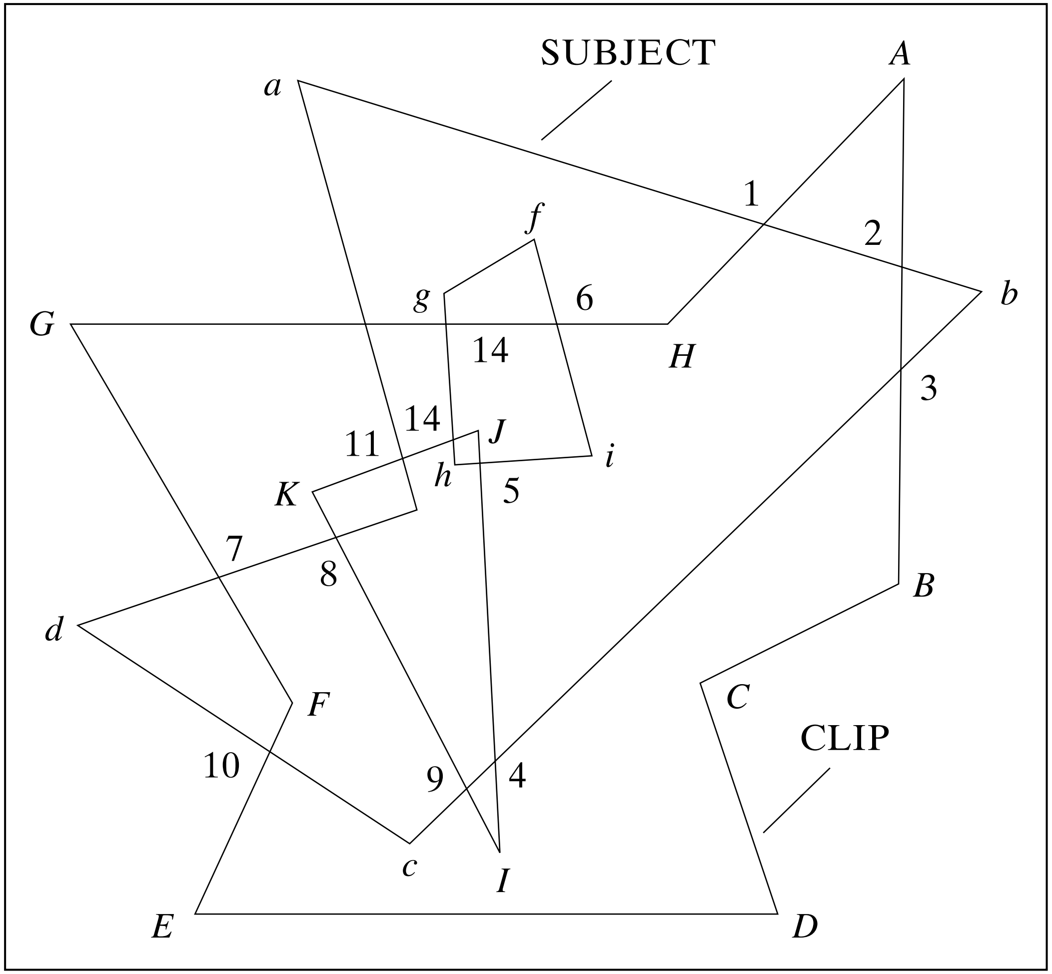


FIGURE 4.61 Applying the Weiler–Atherton method.

FIGURE 4.62 Weiler–Atherton clipping: polygons with holes.



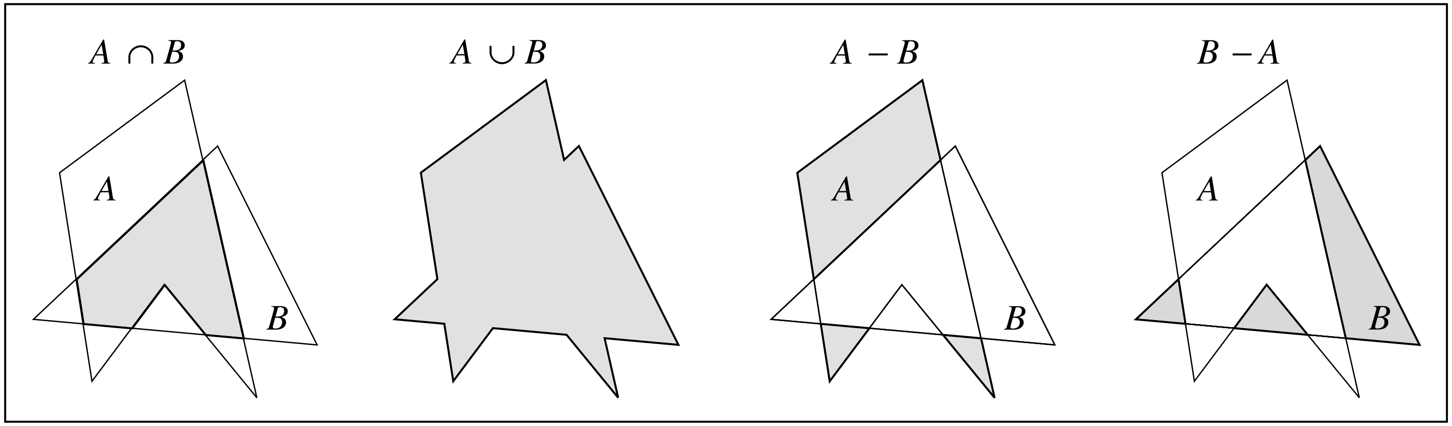


FIGURE 4.63 Polygons formed by Boolean operations on polygons.

from **Computer Graphics Using OpenGL, 2e**, by F. S. Hill

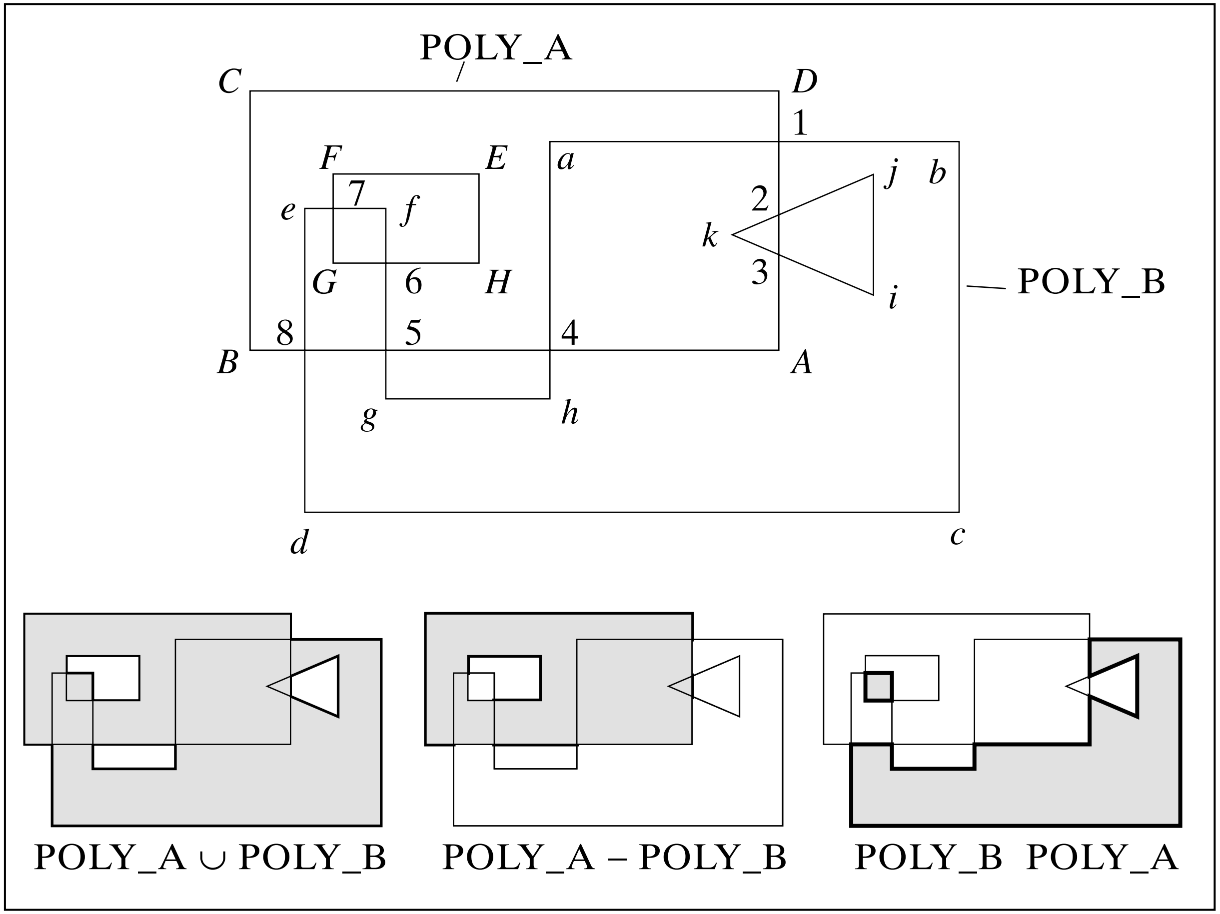


FIGURE 4.64 Forming the union and difference of two polygons.