Normals

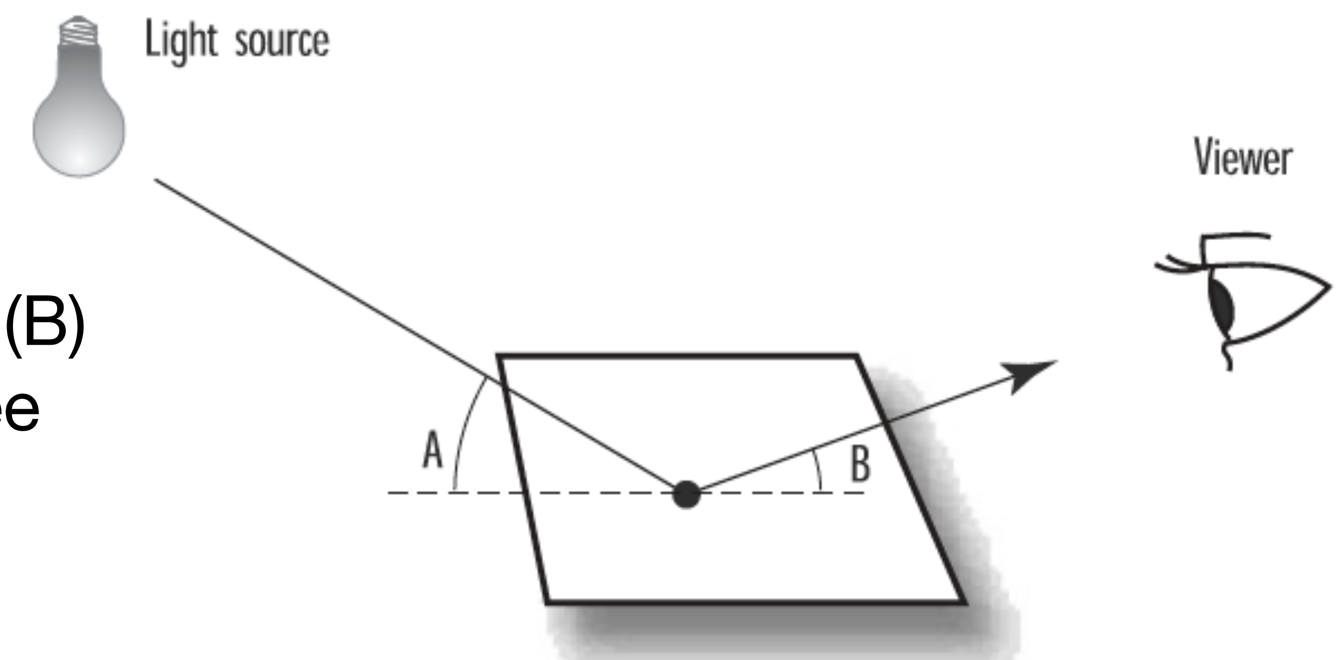
OpenGL

Which Way is Up?

- When you specify a light source, tell OpenGL where it is and in which direction it's shining.
- Often, the light source shines in all directions, but it can be directional.
- Either way, for any object, the rays of light from any source (other than a pure ambient source) strike the surface of the polygons that make up the object at an angle.
- In the case of a directional light, the surfaces of all polygons might not necessarily be illuminated.
- To calculate the shading effects across the surface of the polygons, OpenGL must be able to calculate the angle.

Angles

- A polygon (a square) is being struck by a ray of light from some source.
- The ray makes an angle (A) with the plane as it strikes the surface.
- The light is then reflected at an angle (B) toward the viewer (or you wouldn't see it).
- These angles are used by OpenGL in conjunction with the lighting and material properties to calculate the apparent color of that location

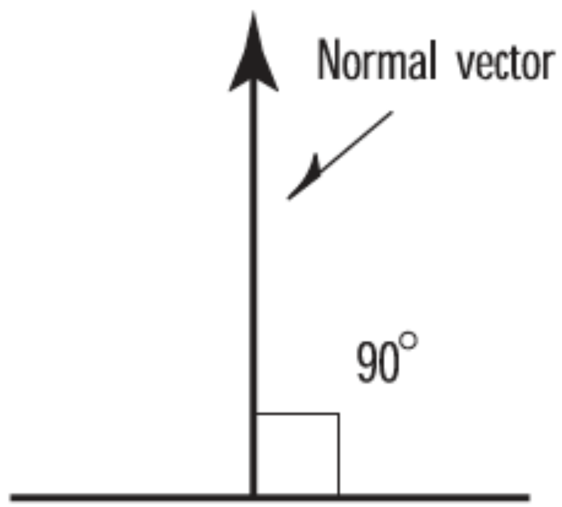


Calculating the Angles?

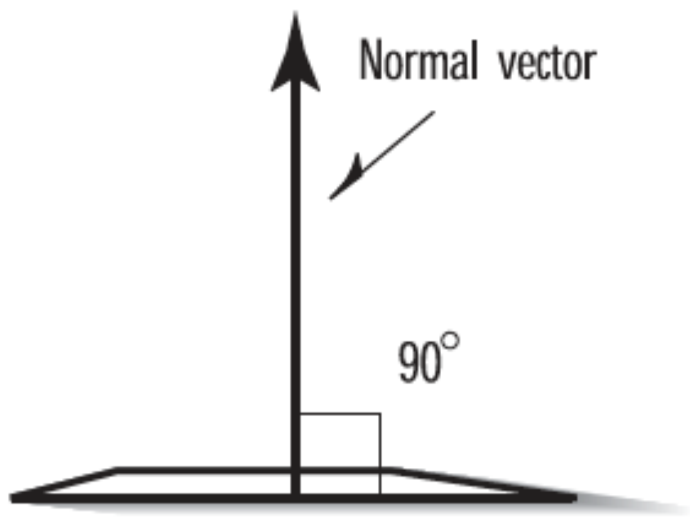
- From a programming standpoint, these lighting calculations present a slight conceptual difficulty.
- Each polygon is created as a set of vertices, and each vertex is then struck by a ray of light at some angle.
- How to calculate the angle between a point and a line (the ray of light)?
- Can’t geometrically find the angle between a single point and a line in 3D space because there are an infinite number of possibilities.
- Therefore, you must associate with each vertex some piece of information that denotes a direction upward from the vertex and away from the surface of the primitive.

Surface Normals

- A line from the vertex in the upward direction starts in some imaginary plane at a right angle.
- This line is called a normal vector.
- The imaginary plane is the surface of the polygon



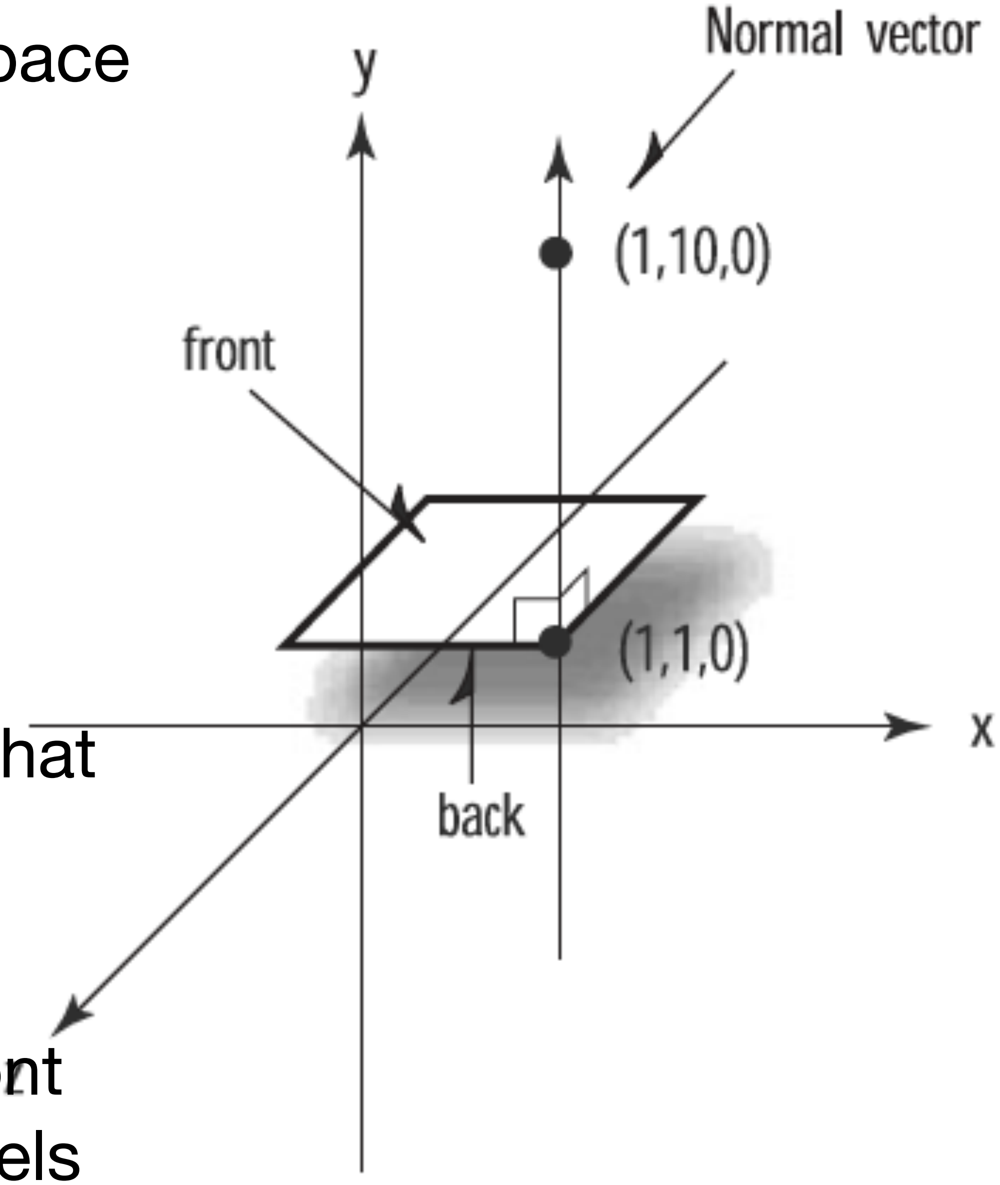
A 2D normal vector



A 3D normal vector

Specifying Normals

- Eg a plane floating above the xz plane in 3D space
- The line through the vertex (1,1,0) that is perpendicular to the plane.
- Select a point on this line, say (1,10,0), the line from the first point (1,1,0) to the second point (1,10,0) is our normal vector.
- The second point specified actually indicates that the direction from the vertex is up in the y direction.
- This convention is also used to indicate the front and back sides of polygons, as the vector travels up and away from the front surface.



Normal Vector

- This second point is the number of units in the x, y, and z directions for some point on the normal vector away from the vertex.

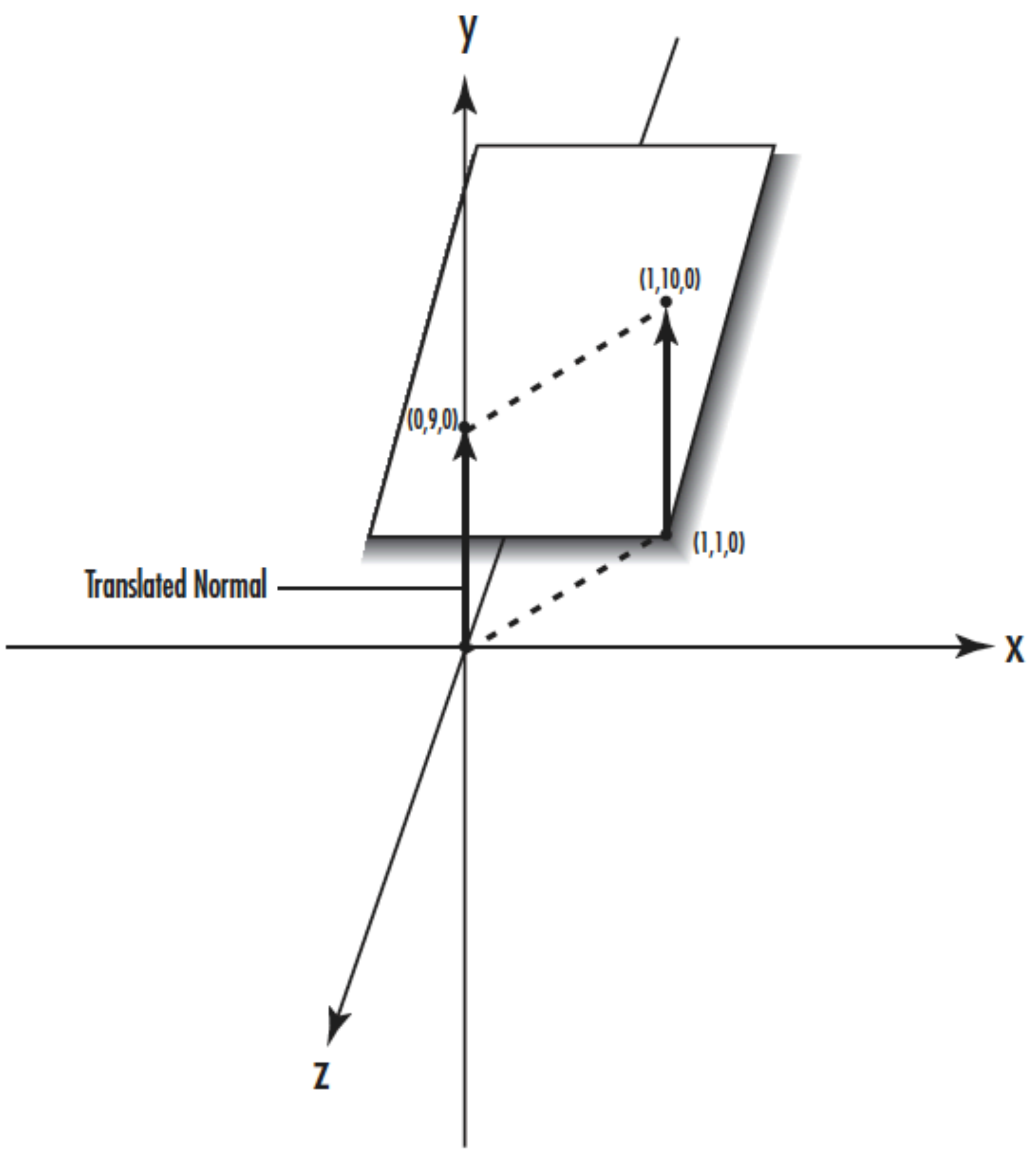
- Rather than specify two points foreach normal vector, we can subtract the vertex from the second point on the normal, yielding a single coordinate triplet that indicates the x, y, and z steps away from the vertex.

- For our example, this is(1,10,0) – (1,1,0) = (1 – 1, 10 – 1, 0) = (0,9,0)

Normalised

- If the vertex were translated to the origin, the point specified by subtracting the two original points would still specify the direction pointing away and at a 90°angle from the surface.

- The vector is a directional quantity that tells OpenGL which direction the vertices (or polygon) face



Specifying Normals to OpenGL

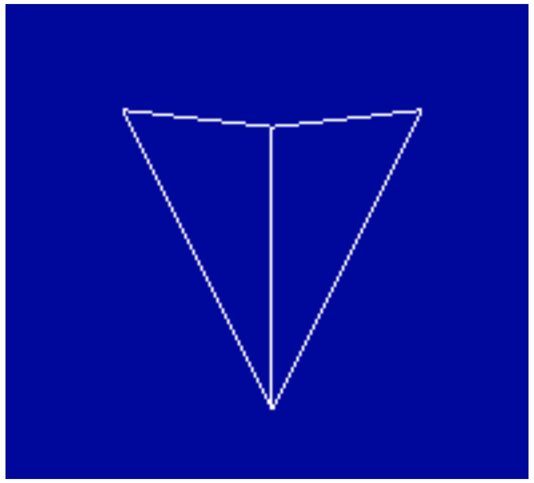
```
Vector3  noseCone[][3] =
{ { Vector3 (  0.0,  0.0,   6.0),
    Vector3 ( -1.5,  0.0,   3.0),
    Vector3 (  1.5,  0.0,   3.0)   },
  { Vector3 (  1.5,  0.0,   3.0),
    Vector3 (  0.0,  1.5 ,  3.0),
    Vector3 (  0.0,  0.0,   6.0)   },
  { Vector3 (  0.0,  0.0,   6.0),
    Vector3 (  0.0,  1.5,   3.0),
    Vector3 ( -1.5,  0.0,   3.0)   }
};
```

- The function glNormal3f takes the coordinate triplet that specifies a normal vector pointing in the direction perpendicular to the surface of this triangle.

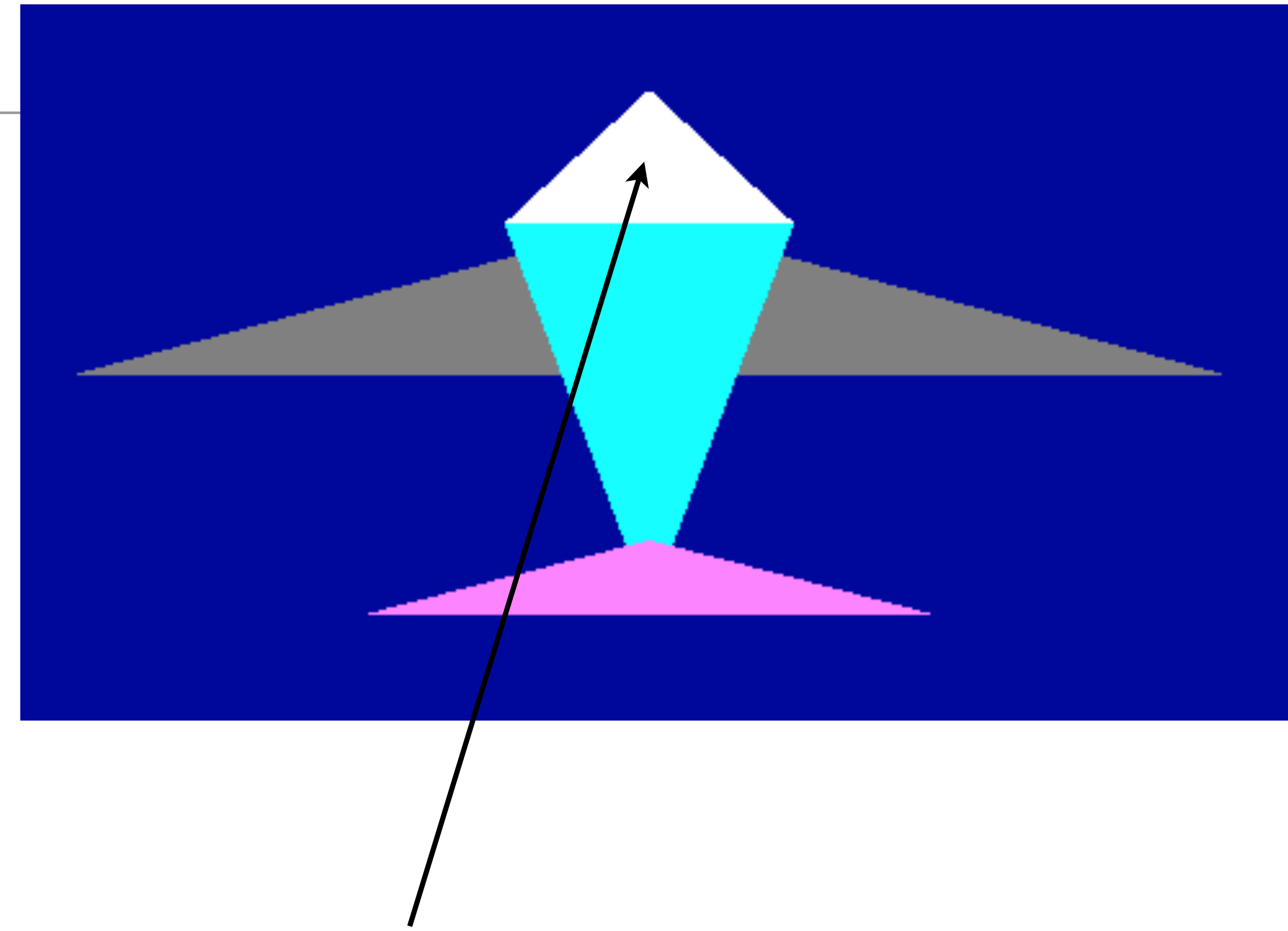
- Here, the normals for all three vertices have the same direction, which is down the negative y-axis.

- A simple example because the triangle is lying flat in the xz plane, and it actually represents part of the nose cone of our model jet.

```
glBegin( GL_TRIANGLES);
  glNormal3f(0.0f, -1.0f, 0.0f);
  glVertex3f(0.0f, 0.0f, 60.0f);
  glVertex3f(-15.0f, 0.0f, 30.0f);
  glVertex3f(15.0f, 0.0f, 30.0f);
glEnd();
```



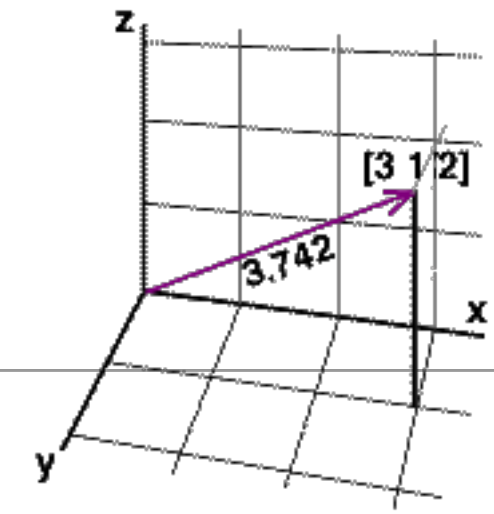
Recap: Winding



- Take special note of the order of the vertices in the jet's triangle.
- If you view this triangle being drawn from the direction in which the normal vector points, the corners appear counter clockwise around the triangle.
- This is called polygon winding.
- By default, the front of a polygon is defined as the side from which the vertices appear to be wound in a counterclockwise fashion.

```
glBegin( GL_TRIANGLES);
  glNormal3f(0.0f, -1.0f, 0.0f);
  glVertex3f(0.0f, 0.0f, 60.0f);
  glVertex3f(-15.0f, 0.0f, 30.0f);
  glVertex3f(15.0f, 0.0f, 30.0f);
glEnd();
```

Unit Normals



- A unit normal is just a normal vector that has a length of 1.

```
V[3 1 2]
    x = 3,
    y = 1,
    z = 2,
```

- All surface normals must eventually be converted to unit normals.

- Normalization:

```
length = sqrt((ax * ax) + (ay * ay) + (az * az))
length = sqrt(9 + 1 + 4) = 3.742
```

  - Calculate length: square each component, add them together, and take the square root.

```
x = 3.0 / 3.742 = 0.802
y = 1.0 / 3.742 = 0.267
z = 2.0 / 3.742 = 0.534
```

  - Divide each component of the normal by the length

```
V[0.8, 0.27. 0.534]
```

OpenGL Normalize Computation

- Instruct OpenGL to convert your normals to unit normals automatically, by enabling normalization with glEnable and a parameter of GL_NORMALIZE:;

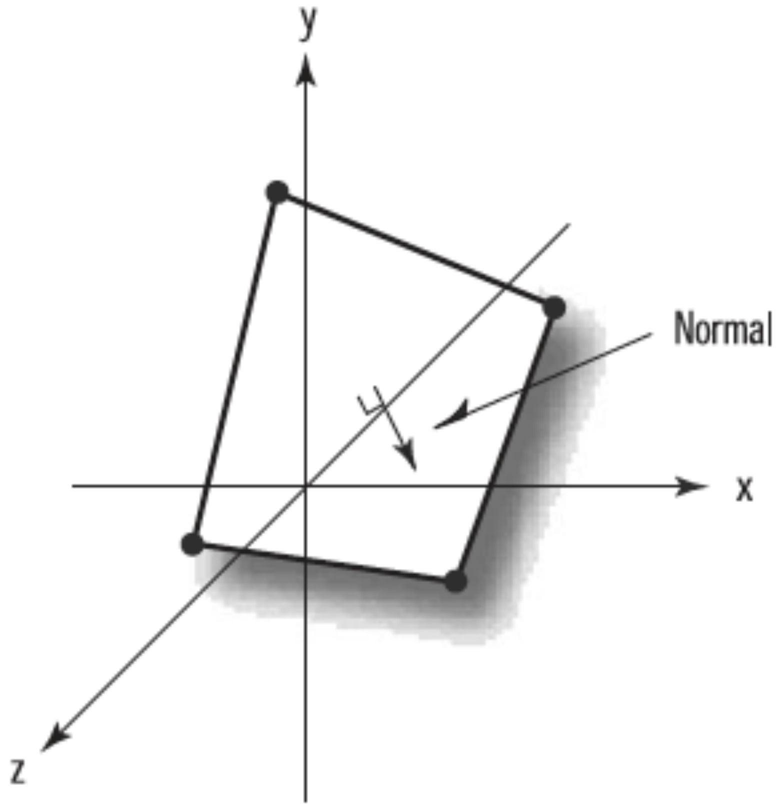
```
glEnable(GL_NORMALIZE);
```

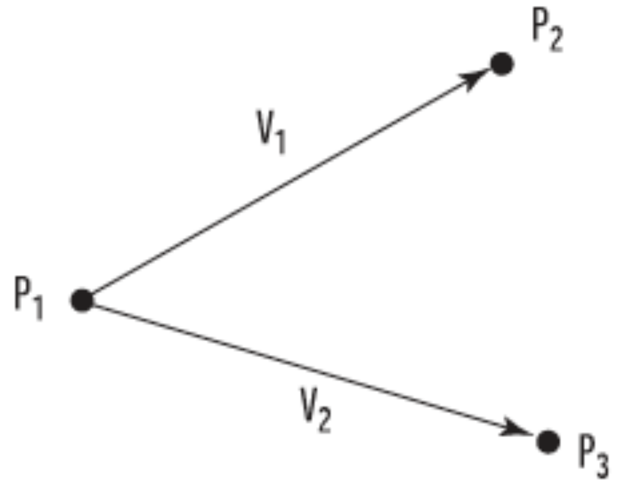
- This approach does, however, have performance penalties on some implementations.
- May be better to calculate your normals ahead of time as unit normals instead of relying on OpenGL to perform this task.
- If applying scaling during a transformation, may need to rescale the normals to keep lighting effects consistent.

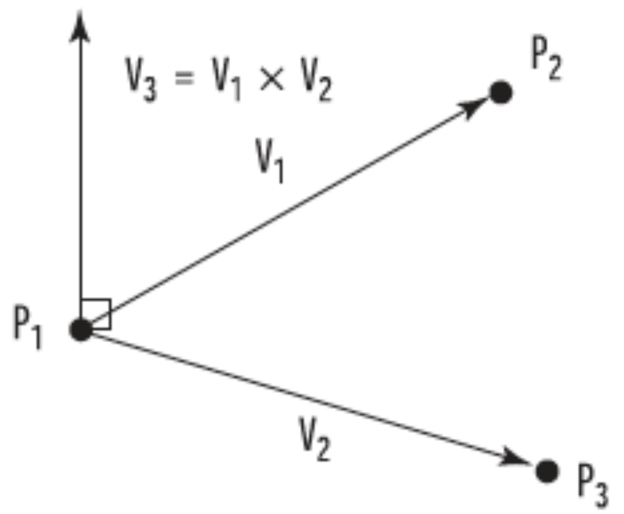
```
glEnable(GL_RESCALE_NORMALS);
```

Finding a Normal

- Take three points that lie in the plane of the polygon (P1, P2 and P3).
- Define two vectors: V1 from P1 to P2, and V2 from P1 to P3.
- Two vectors in three-dimensional space define a plane, so the cross product of V1 and V2 yields a vector is perpendicular to that plane - the Normal.







findNormal()

```
Vector3 findNormal(const Vector3& point1, const Vector3& point2, const Vector3& point3)
{
  Vector3 v1, v2;

  // Calculate two vectors from the three points. Assumes counter clockwise winding
  v1.X = point1.X - point2.X;
  v1.Y = point1.Y - point2.Y;
  v1.Z = point1.Z - point2.Z;

  v2.X = point2.X - point3.Z;
  v2.Y = point2.Y - point3.Y;
  v2.Z = point2.Z - point3.Z;

  // Take the cross product of the two vectors to get he normal vector.
  Vector3 result;
  result.X =  v1.Y * v2.Z - v2.Y * v1.Z;
  result.Y = -v1.X * v2.Z + v2.X * v1.Z;
  result.Z =  v1.X * v2.Y - v2.X * v1.Y;
  return result;
}
```

Generate Normals

- Compute the normal and send to pipeline in advance of the vertices.

```
void render (Vector3 vectors[][3], int size)
{
  for (int i=0; i<size; i++)
  {
    glBegin(GL_TRIANGLES);
      Vector3 normal = findNormal( vectors[i][0], vectors[i][1], vectors[i][2]);
      glNormal3f(normal.X, normal.Y, normal.Z);
      vectors[i][0].render();
      vectors[i][1].render();
      vectors[i][2].render();
    glEnd();
  }
}
```