Computing normal and reflection vectors

• Normal vectors

- Plane normal

  * Equation of plane: $ax+by+cz+d=0$

  * Normal: (a,b,c)

- Sphere normal

  * Equation of sphere: $f(x, y, z) = x^2 + y^2 + z^2 - 1 = 0$

  * Normal: $\begin{bmatrix} f'_x \\ f'_y \\ f'_z \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \\ 2z \end{bmatrix}$

- In general, you need to use *cross products* to compute normals

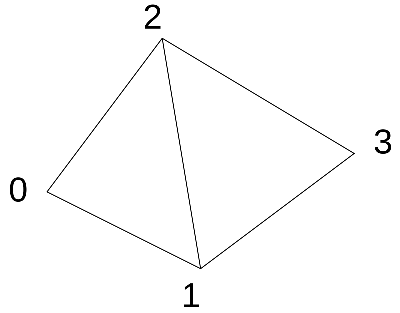
• Specifying normals in OpenGL

- OpenGL associates normals with vertices than surfaces

- Normals are associated with vertices using the statement

glNormal3f(nx, ny, nz)

- Warning: normal will be associated with all the vertices specified after the call to *glNormal()* (it acts like a state variable)



- Example1: provide a single normal for each triangle

```
glBegin(GL_TRIANGLES):
  glNormal3f(u0, v0, w0); /* normal for triangle 012 */
  glVertex3f(x0, y0, z0);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);

  glNormal3f(u1, v1, w1); /* normal for triangle 132 */
  glVertex3f(x1, y1, z1);
  glVertex3f(x3, y3, z3);
  glVertex3f(x2, y2, z2);
glEnd();
```

- Example2: provide a single normal for each vertex

```
glBegin(GL_TRIANGLES):
  glNormal3f(u0, v0, w0); /* normal for triangle 012 */
  glVertex3f(x0, y0, z0);
  glNormal3f(ua, va, wa); /* average normal */
  glVertex3f(x1, y1, z1);
  glNormal3f(ua, va, wa); /* average normal */
  glVertex3f(x2, y2, z2);

  glNormal3f(ua, va, wa); /* average normal */
  glVertex3f(x1, y1, z1);
  glNormal3f(u1, v1, w1); /* normal for triangle 132 */
  glVertex3f(x3, y3, z3);
  glNormal3f(ua, va, wa); /* average normal */
  glVertex3f(x2, y2, z2);
glEnd();
```

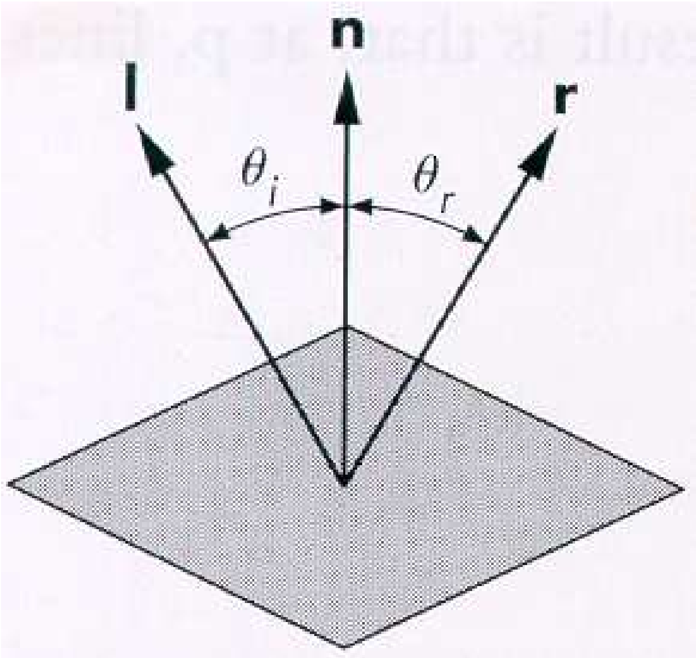
- Warning: Each normal must be normalized !!

- OpenGL will normalize the vectors for you, if you include the following statement in your initialization code:

glEnable(GL_NORMALIZE);

• Reflection vectors

- $cos(\theta_i)$=$cos(\theta_r)$
- $cos(\theta_i) = l.\, n$, $cos(\theta_r) = r.\, n$,
- $r = \alpha l + \beta n$ (r is linearly dependend on n and l)
- Additional constraint: $\|r\| = 1$
- Solving for r: $r = 2(l.\, n)n - l$



Polygonal Shading

- Amount of work required for shading flat polygons can be reduced

- Curved surfaces are usually decomposed into many small flat polygons

• Flat Shading

- Assumptions

  * Vector **n** is constant (true for completely flat polygons)

  * Vector **v** is constant (true if the viewer is distant)

  * Vector **l** is constant (true if the light source is distant)

- If **n**, **v**, and **l** are constant, shading calculations needs to be carried out **once**

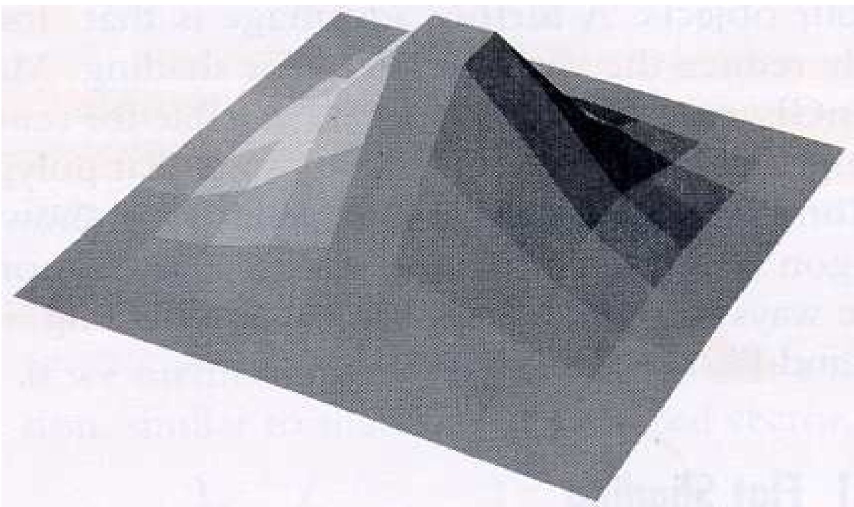
• Flat shading in OpenGL

glShadeModel(GL_FLAT)

(OpenGL uses the normal associated with the first vertex of a single polygon for the shading calculation)

• Problems with flat shading

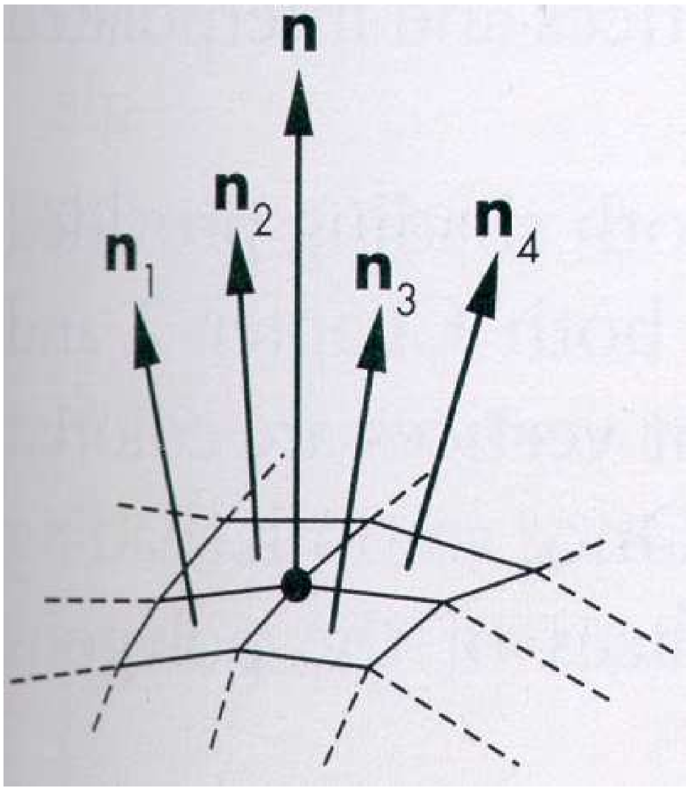
- Differences in shading between adjacent polygons



• Interpolative (Gouraud) shading

- For each vertex, considered the normalized average of the normals of the polygons that share the vertex

$$\frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|}$$



• Smooth shading in OpenGL

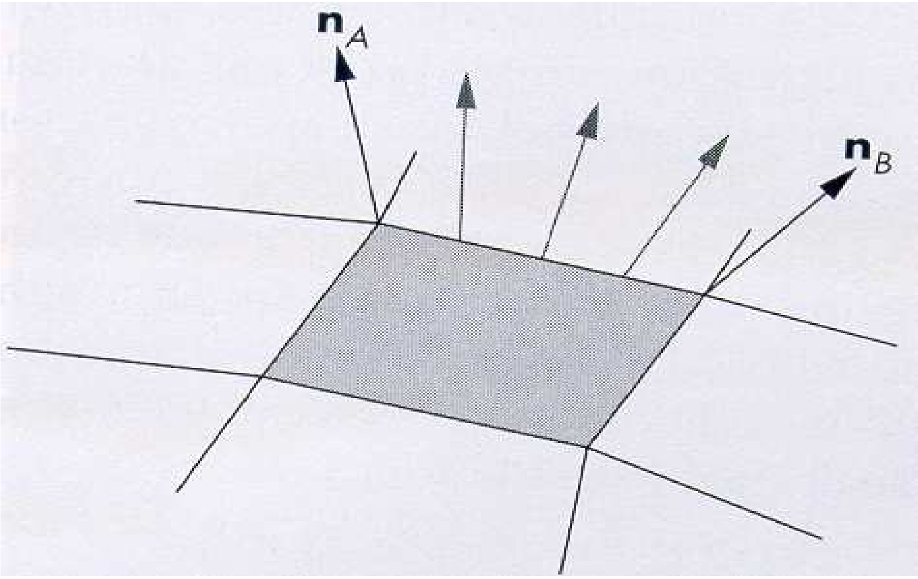
glShadeModel(GL_SMOOTH)

(needs a data structure for storing the neighbor vertices of a vertex)

• Phong polygonal shading

- Interpolate normals along an edge

$$n(\alpha) = (1-\alpha)n_A + \alpha n_B$$



- Interpolate normals at interior points

$$n(\alpha, \beta) = (1-\beta)n_C + \beta n_D$$