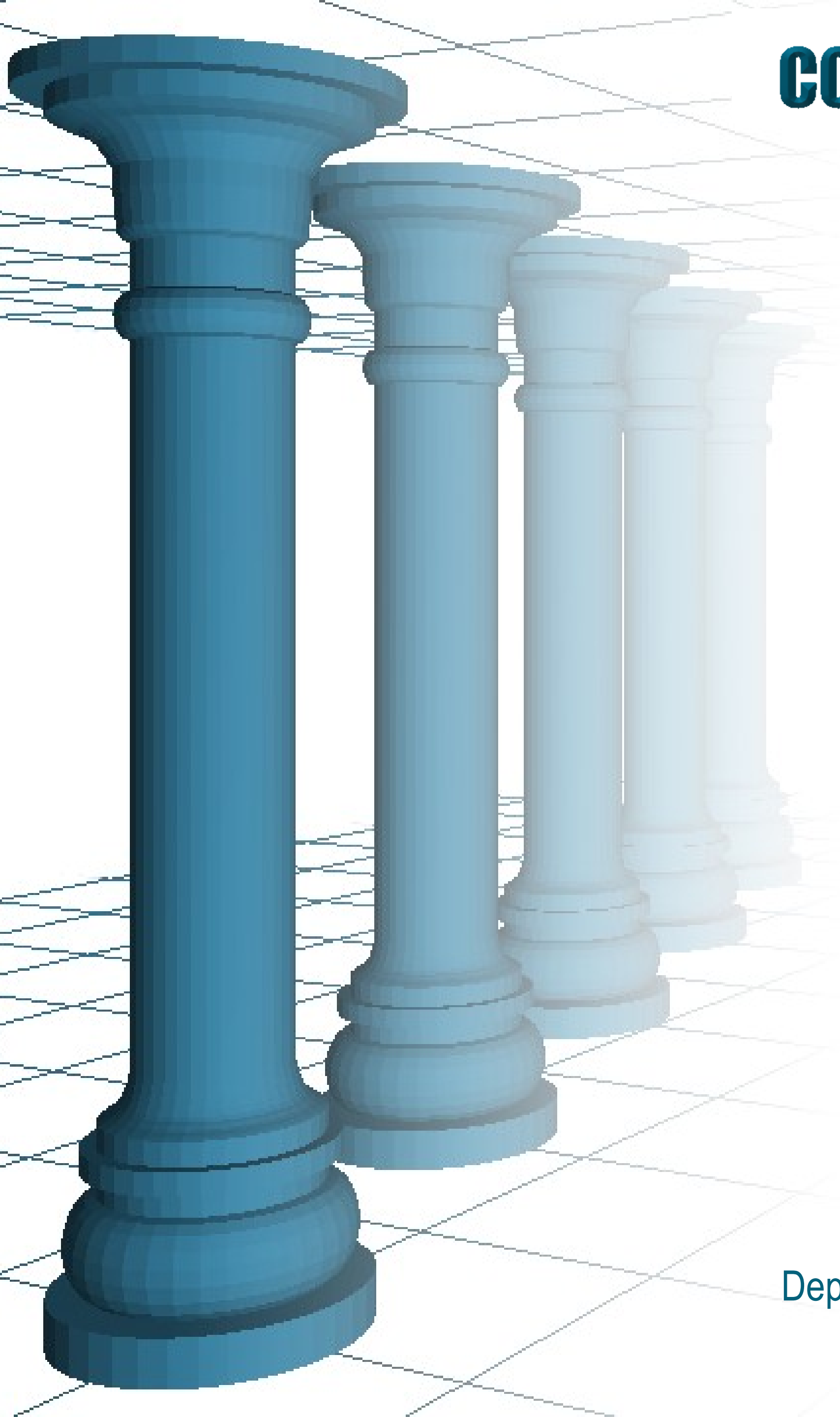
COSC363 Computer Graphics



8

Mathematics of Lighting

Vectors + Colours = Light!

R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

GLM Library

The OpenGL Mathematics (GLM) library is a convenient C++ library for developing graphics applications.

- Header only library

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
```

- Several functions and variables use similar naming convention as OpenGL and GLSL

```
glm::vec4  point(2, -3, 4, 1);
glm::mat4  viewMat = glm::lookAt(eye, look, up);
glm::mat4  rotnMat = glm::rotate(inMat, angle, axis);
```

- Particularly useful for matrix operations, lighting computations, ray tracing and OpenGL-4 shader development.

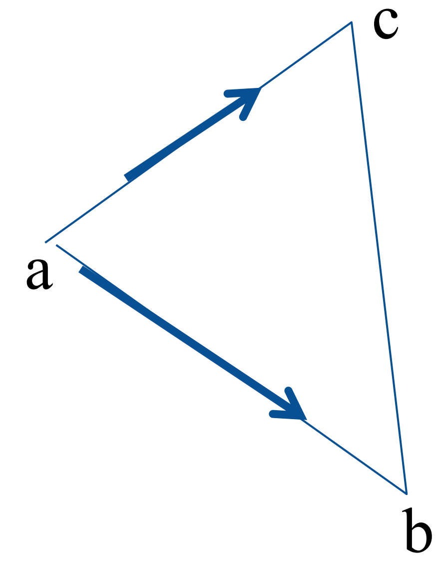
GLM Vector Operations: Example

```
glm::vec3 u, v, w;
u = glm::vec3(3, 0, 4);
v = glm::vec3(-2, -4, 3);
float u_len = glm::length(u);
cout << u_len << endl;                  //Prints 5
w = glm::cross(u, v);
cout << glm::to_string(w) << endl;      //Prints vec3(16, -17, -12)
glm::vec3 wn = glm::normalize(w);
cout << glm::to_string(wn) << endl;     //Prints vec3(0.609, -0.647, -0.457)
float dotprod = glm::dot(u, w);
cout << dotprod << endl;                //Prints 0
glm::vec4 pt(-5, 2, 9, 1);
     glBegin(GL_POINTS);
        glVertex3fv(glm::value_ptr(pt));
     glEnd();
```

Normal Vector of a Triangle

```
glm::vec3 a, b, c;
a = glm::vec3(-2, 1, 1);
b = glm::vec3(0, -5, 3);
C = glm::vec3(3, 2, -7);

glm::vec3 norm;
norm = glm::cross(b-a, c-a);
norm = glm::normalize(norm);
```



GLM Matrix Operations: Example

```
glm::mat4 im, am, bm, cm;
glm::vec4 p(2, -1, 3, 1),  q;
im = glm::mat4(1.0f);          //Identity matrix
am = glm::scale(im, glm::vec3(sx, sy, sz));
```

In Radians

```
bm = glm::rotate(am, theta, glm::vec3(0.f, 1.f, 0.f));
cm = glm::translate(bm, glm::vec3(tx, ty, tz));
q = cm * p;
```

am = I * S = S (S = scale transformation matrix)

bm = am * R = S * R (R = rotation matrix)

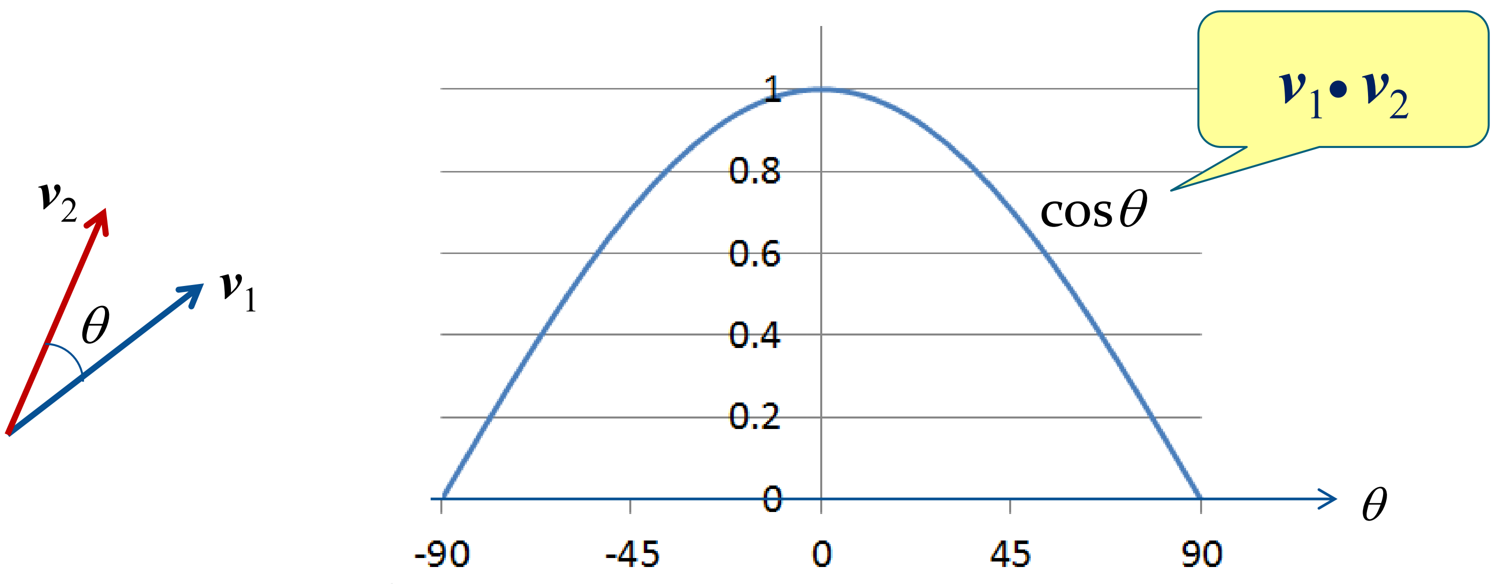
cm = bm * T = S*R*T (T = translation matrix)

q = cm * p = S*R*T*p

Cosine Variation

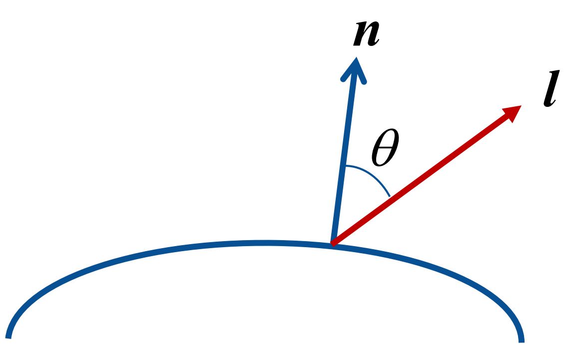
Several values in a lighting computation vary based on the cosine of the angle between two unit vectors:

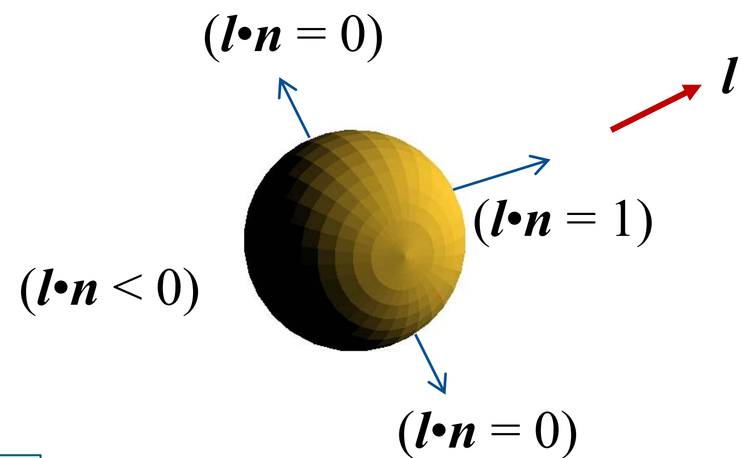
- Max value when angle = 0 (vectors parallel)
- Min value when angle = 90 degs (vectors perpendicular)



Diffuse Reflections

The diffuse reflection from a surface varies as the cosine of the angle between the **normal vector *n*** and the **light source vector *l*** (Lambert's law)



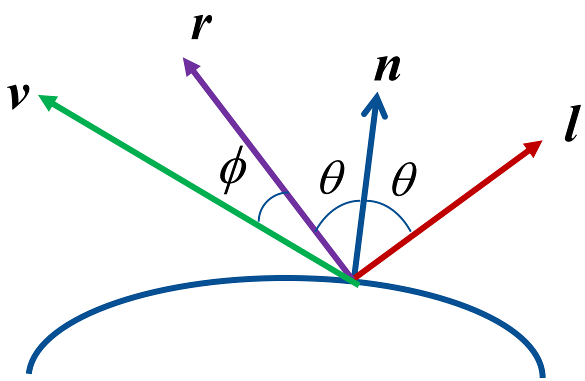


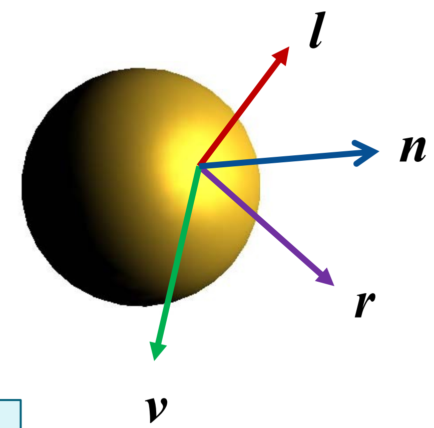
$$\boldsymbol{I}_d = L_d M_d \max(\boldsymbol{l}\bullet\boldsymbol{n}\ ,\ 0)$$

Note: $\boldsymbol{l}, \boldsymbol{n}$ must be normalized to unit vectors.
Usually, $L_d = (1, 1, 1)$. Then, $I_d = M_d \max(l.n, 0)$

Specular Reflections

The specular reflection from a surface varies as the cosine of the angle between the **reflection vector *r*** and the **view vector *v***.





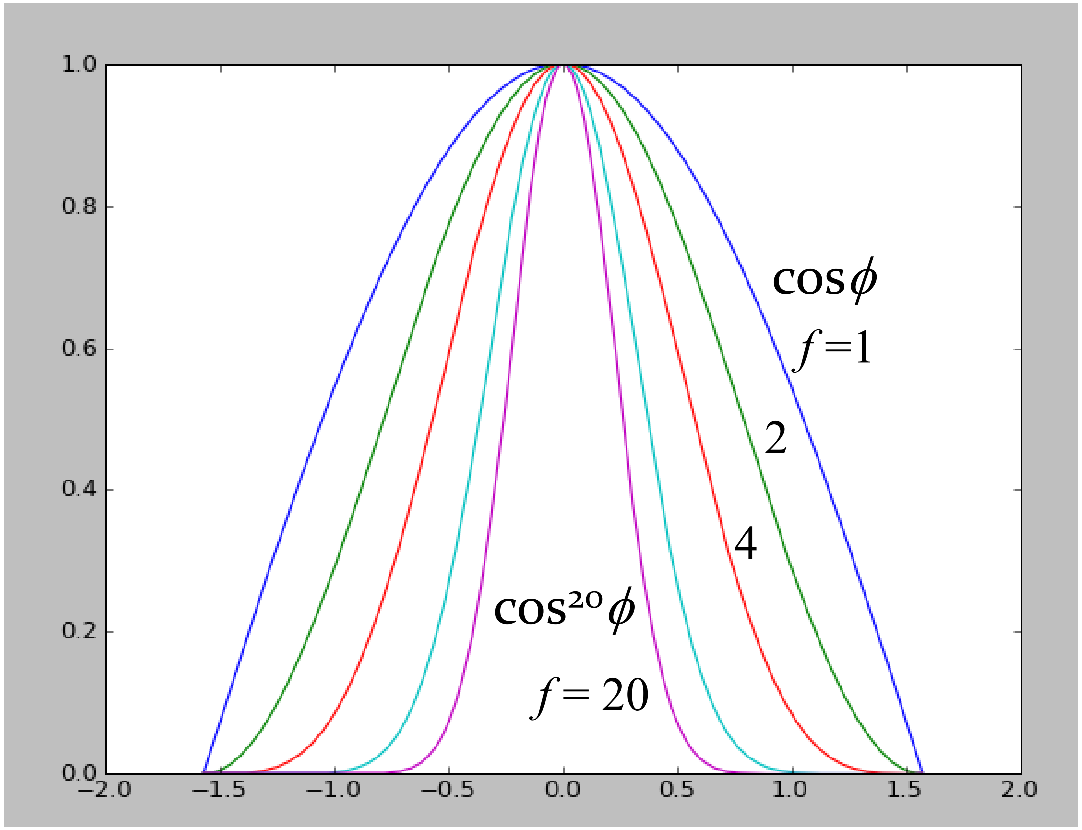
$$\boldsymbol{I_s} = L_s M_s \max(\boldsymbol{r} \bullet \boldsymbol{v} \,,\, 0)$$

We also include the Phong's constant (shininess term) f to control the diameter of the specular highlight

$$\boldsymbol{I_s} = L_s M_s \,\{\max(\boldsymbol{r} \bullet \boldsymbol{v} \,,\, 0)\}^f$$

Note: $\boldsymbol{r}, \boldsymbol{v}$ are unit vectors.

Phong's Constant

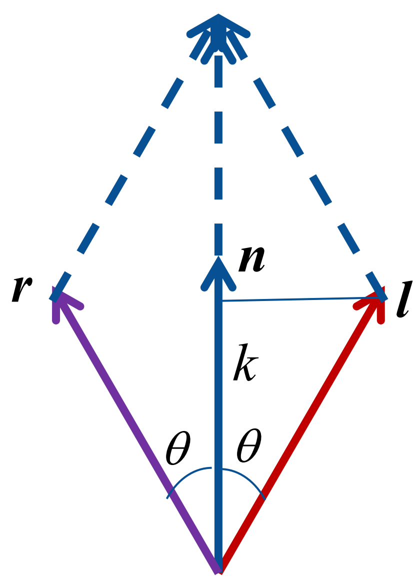


Large values of the exponent f gives highly concentrated highlights.

Computation of Reflection Vector

The computation of the specular component of lighting requires the reflection vector. It has the following properties:

- Vectors $\boldsymbol{l}$ and $\boldsymbol{v}$ make equal angles with the normal vector $\boldsymbol{n}$
- Vectors $\boldsymbol{l}$, $\boldsymbol{v}$ and $\boldsymbol{n}$ are on the same plane.



Let k be the length of projection of the vector $\boldsymbol{l}$ on the unit vector $\boldsymbol{n}$.

$k = (\boldsymbol{l} \bullet \boldsymbol{n})$ see Slide [7]-18

The projection of $\boldsymbol{r}$ on the unit vector $\boldsymbol{n}$ also has the same length k.

$$\boldsymbol{r} + \boldsymbol{l} = 2k\,\boldsymbol{n}$$

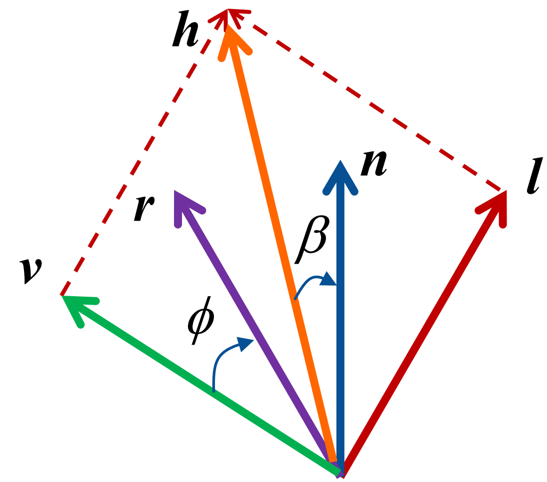
Therefore,

$$\boldsymbol{r} = 2\,(\boldsymbol{l} \bullet \boldsymbol{n})\,\boldsymbol{n} - \boldsymbol{l}$$

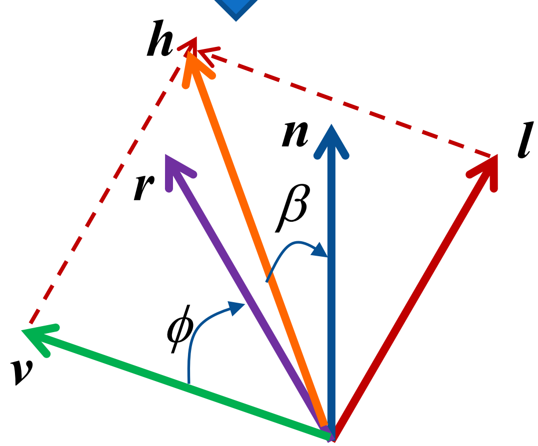
Half-way Vector

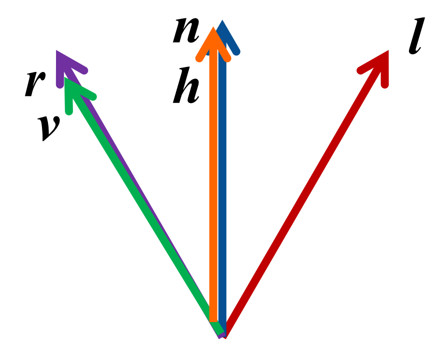
Consider the vector $\boldsymbol{h} = (\boldsymbol{l} + \boldsymbol{v})$ normalized.
Let β be the angle between $\boldsymbol{h}$ and $\boldsymbol{n}$.
We observe the following facts:

- When ϕ increases, β also increases.
- When $\phi = 0$, β also becomes 0.



Keeping $\boldsymbol{l}$, $\boldsymbol{n}$ (and $\boldsymbol{r}$) fixed, if we move $\boldsymbol{v}$ away from $\boldsymbol{r}$, then $\boldsymbol{h}$ moves away from $\boldsymbol{n}$. If $\boldsymbol{v}$ coincides with $\boldsymbol{r}$, then $\boldsymbol{h}$ coincides with $\boldsymbol{n}$.





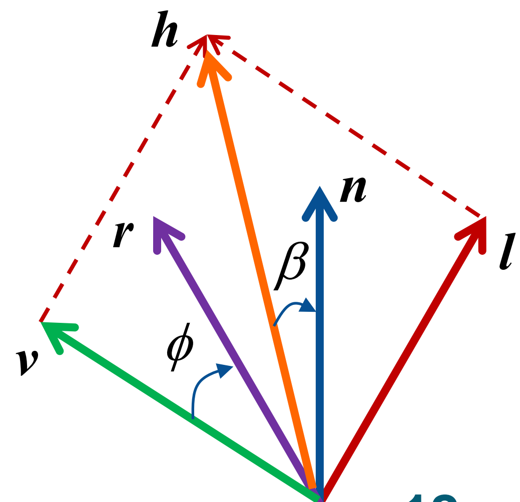
Phong-Blinn Model

- OpenGL uses an approximation of $(\boldsymbol{r} \bullet \boldsymbol{v})$ by the term $(\boldsymbol{h} \bullet \boldsymbol{n})$ in the computation of specular reflections.
- The vector $\boldsymbol{h}$ is called the **Half-way Vector**, and is computed as $\boldsymbol{h} = (\boldsymbol{l} + \boldsymbol{v})$ normalized.
- We can now rewrite the formula for specular reflection:

$$\boldsymbol{I}_s = L_s M_s \{\max(\boldsymbol{h} \bullet \boldsymbol{n}), 0\}^f$$

- The lighting equation with the above approximation is called the Phong-Blinn model.

If $\boldsymbol{l}$ is a directional source, and the view direction is constant (viewer at infinity), then $\boldsymbol{h}$ needs to be computed only once for the whole scene.



Lighting Equation: Phong-Blinn Model

Vertex Colour

$$\boldsymbol{I_P} = \underbrace{L_a M_a}_{\text{Ambient Term}} + \underbrace{L_d M_d \max(\boldsymbol{l}\bullet\boldsymbol{n}\ ,\ 0)}_{\text{Diffuse Term}} + \underbrace{L_s M_s\ \{\max(\boldsymbol{h}\bullet\boldsymbol{n}\ ,\ 0)\}^f}_{\text{Specular Term}}$$

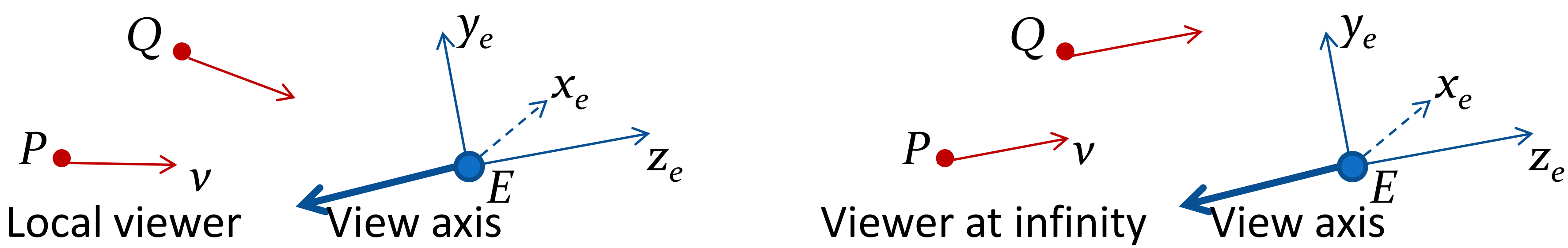
Geometrical Considerations

- The rendering speed is increased if $\boldsymbol{v}$ is made constant for all vertices (infinite viewpoint). This is the default setting.

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE);
```

- You can force the computation of the true value of $\boldsymbol{v}$ for each vertex (local viewpoint) for more realistic results:

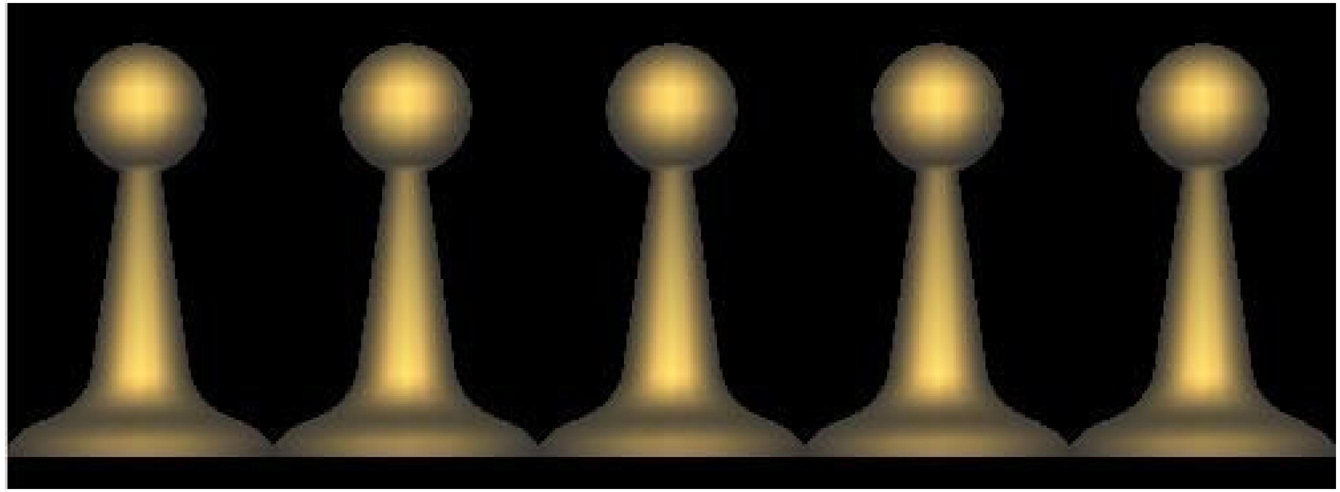
```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```



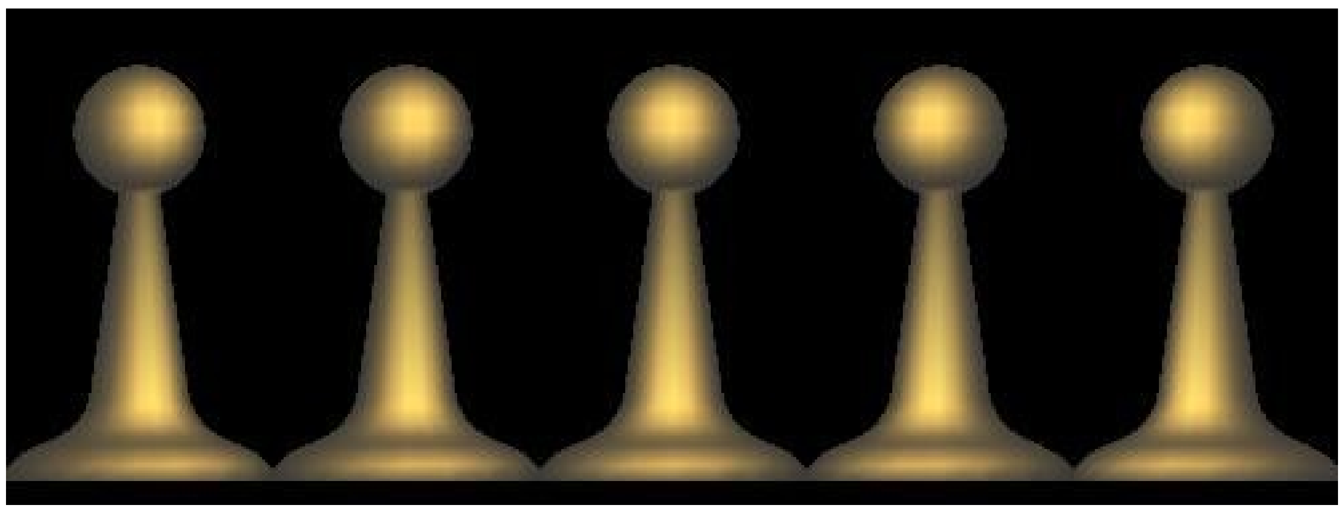
- For a light source at infinity (directional source) the vector $\boldsymbol{l}$ is constant for all vertices in the scene.

Local and Infinite Viewpoints

Infinite viewpoint



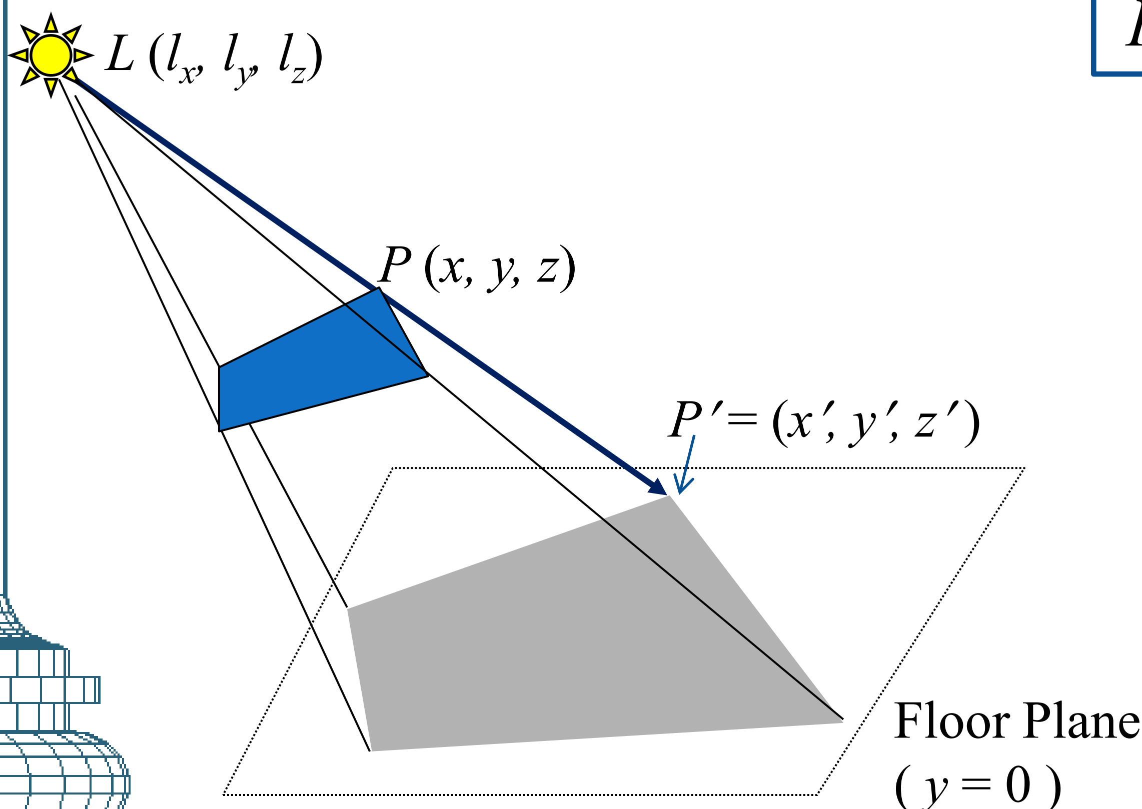
Local viewpoint



Planar Shadows

(*See also slides* [3]:16-20)

- Project each of the polygonal faces onto the floor plane, using the light source (*L*) as the centre of projection.
- Use only the ambient light to draw the projected object.



$$\boxed{P' = (1-t)L + tP, \qquad t > 1.}$$

$$x' = (1-t)l_x + tx$$

$$y' = (1-t)l_y + ty = 0$$

$$z' = (1-t)l_z + tz$$

$$\therefore t = \frac{l_y}{l_y - y}$$

Planar Shadows

- The projection P' of the vertex (x, y, z) on the floor- plane is given by the following coordinates:

$$\left.\begin{aligned} x' &= \frac{-l_x y + l_y x}{l_y - y} \\ y' &= 0 \\ z' &= \frac{-l_z y + l_y z}{l_y - y} \end{aligned}\right\} \xrightarrow{\text{Homogeneous Coordinates}} \begin{aligned} s_x &= -l_x y + l_y x \\ s_y &= 0 \\ s_z &= -l_z y + l_y z \\ w &= l_y - y \end{aligned}$$

- The above equations can be written as a transformation:

$$\begin{bmatrix} s_x \\ s_y \\ s_z \\ w \end{bmatrix} = \begin{bmatrix} l_y & -l_x & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -l_z & l_y & 0 \\ 0 & -1 & 0 & l_y \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Planar Shadows: Code

(*See also slide* [3]:20)

```
// Light source position = (lx, ly, lz)
float shadowMat[16] = { ly,0,0,0, -lx,0,-lz,-1,
                        0,0,ly,0,  0,0,0,ly };
// Draw object
 glEnable(GL_LIGHTING);
 glPushMatrix();     //Draw Actual Object
    /* Transformations */
    drawObject();
 glPopMatrix();

//  Draw shadow
 glDisable(GL_LIGHTING);
 glPushMatrix();     //Draw Shadow Object
   glMultMatrixf(shadowMat);
   /* Transformations */
   glColor4f(0.2, 0.2, 0.2, 1.0);
   drawObject();
 glPopMatrix();
```