Master of intelligent systems

Image synthesis

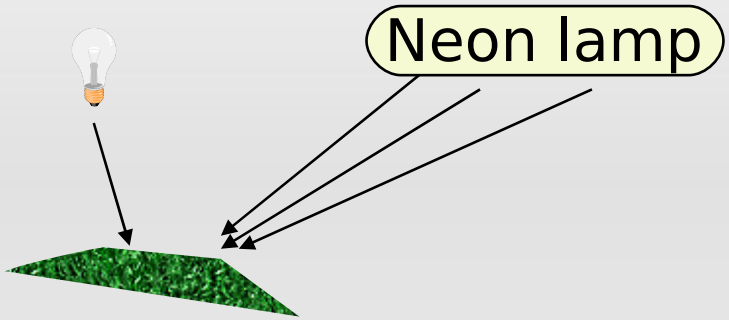
Lighting and shading models

Lighting and shading models

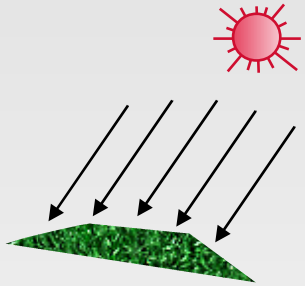
- Illumination
  - Transport of the direct or indirect photon flux from the light sources
- Lighting
  - Computation of the light intensity at some point of the scene
- Shading
  - Using the shading model to get the color of any pixel

Light sources

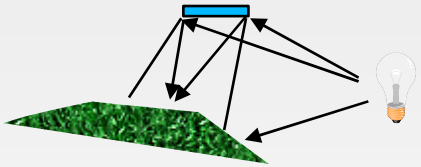
- Point light or area light



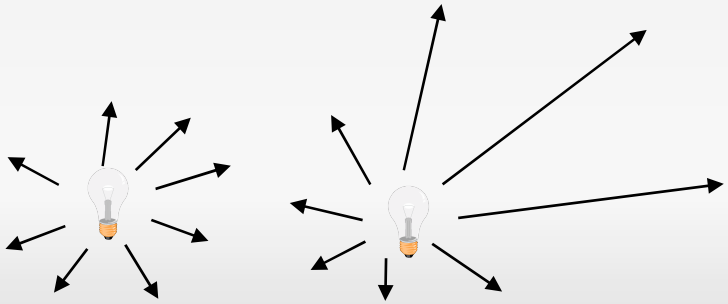
- Directional light



- Direct / Indirect illumination



- Isotropic / Anisotropic distribution

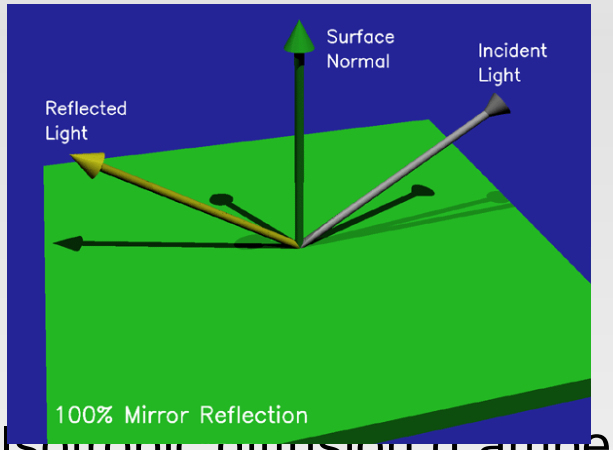


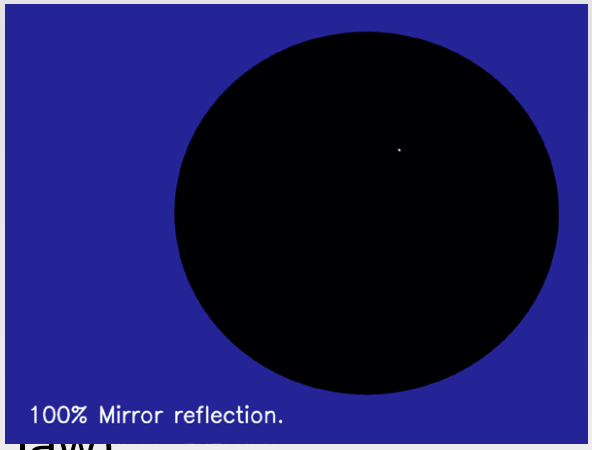
Lighting models

- Light reflexion
  - Composed of 3 terms :
    - Perfectly specular term (mirror)
    - Lambertian diffusion (isotropic diffusion)
    - Oriented imperfect specular term (polished surfaces)

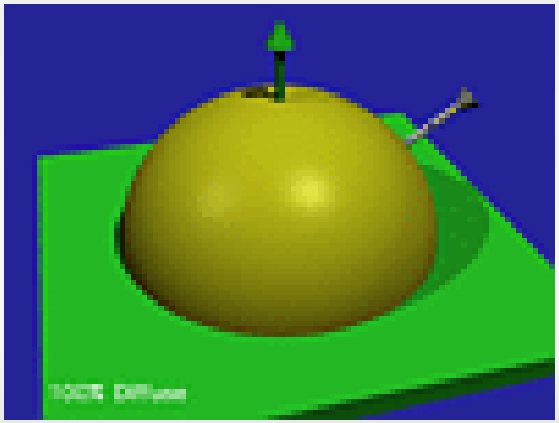
Lighting models

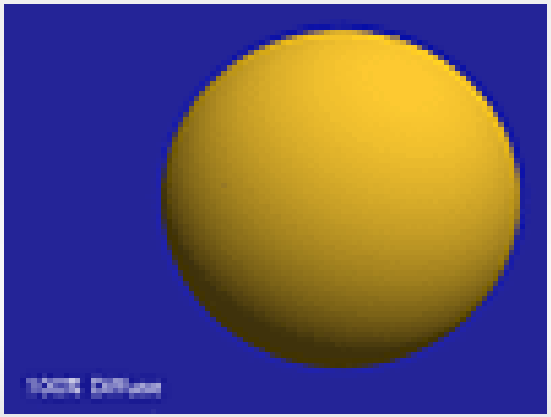
- Perfect mirror (Descartes' law)

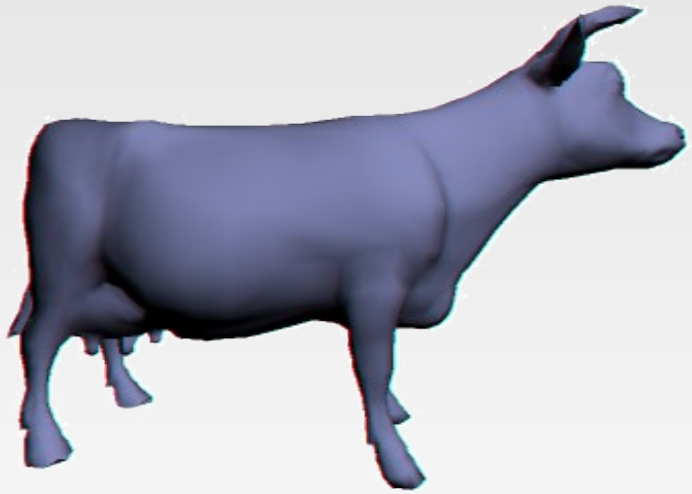




- Isotropic diffusion (Lambert 's law)

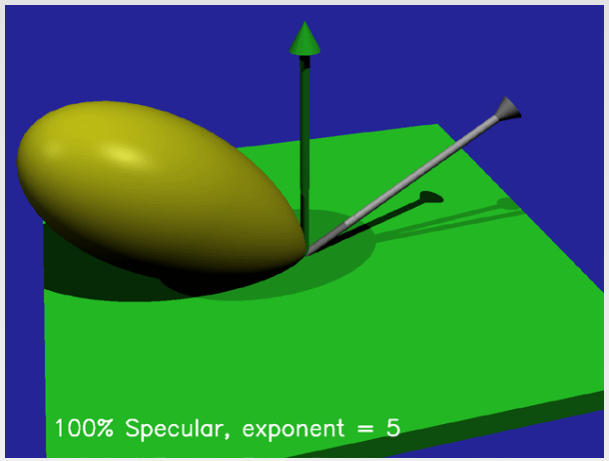


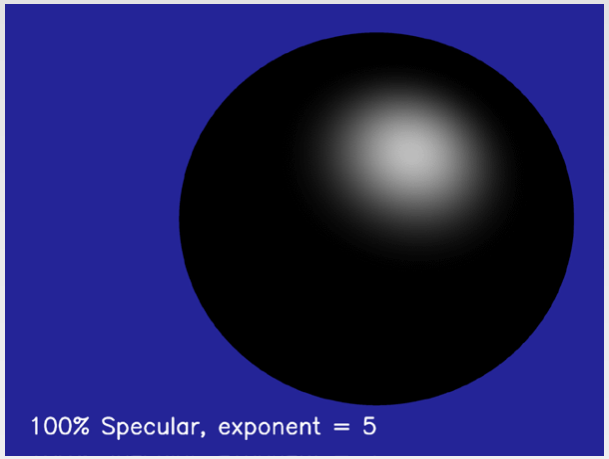


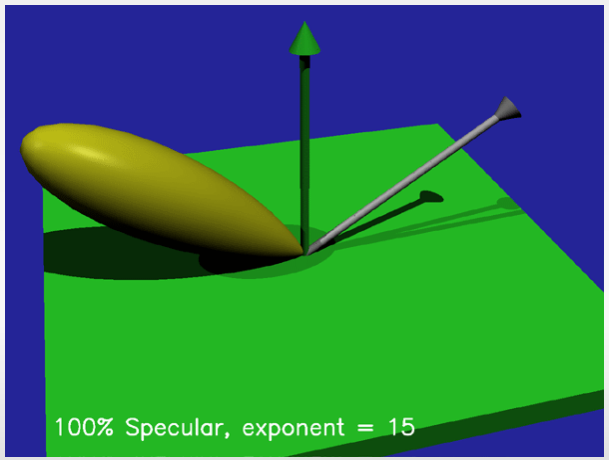


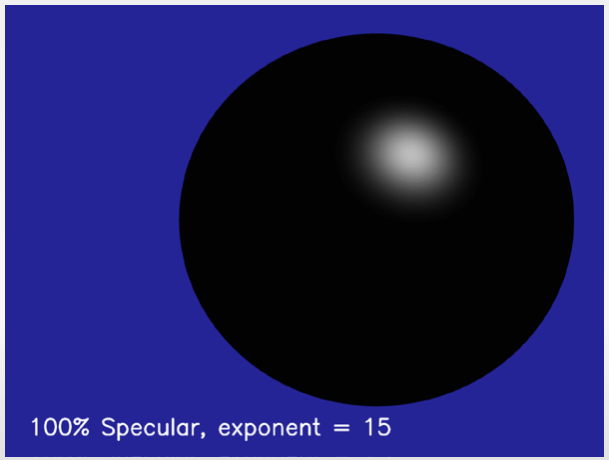
Lighting models

- Imperfect specular reflector



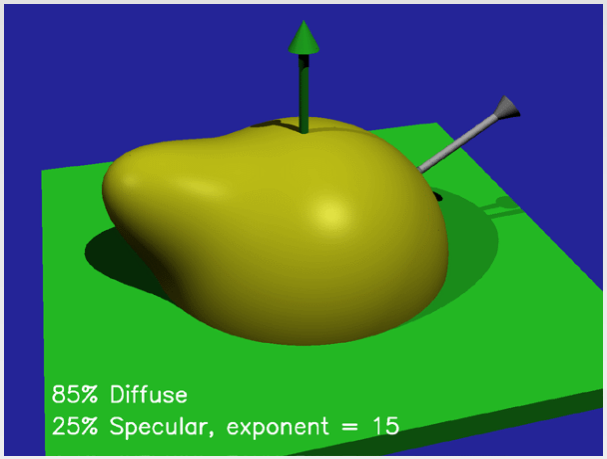


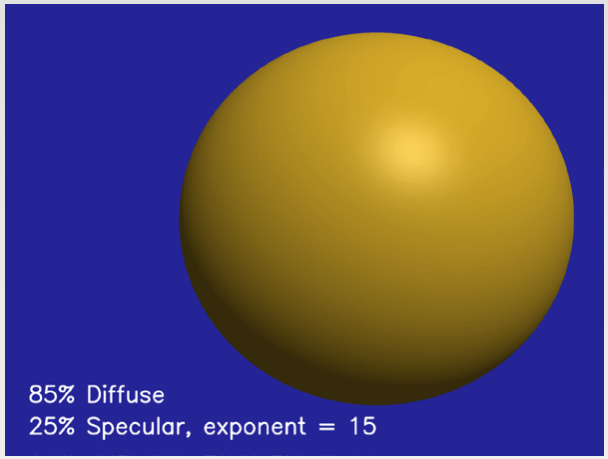


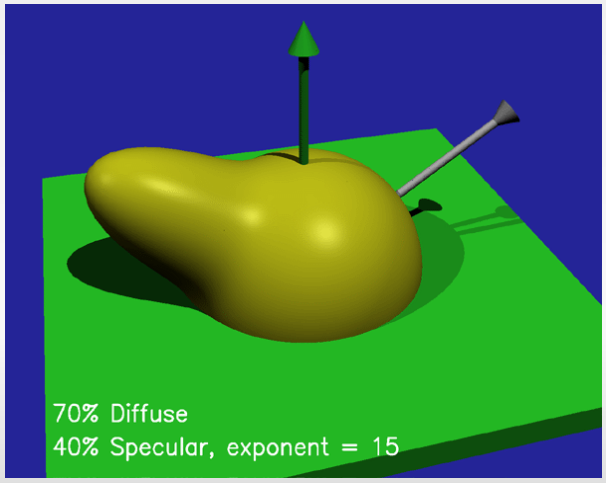


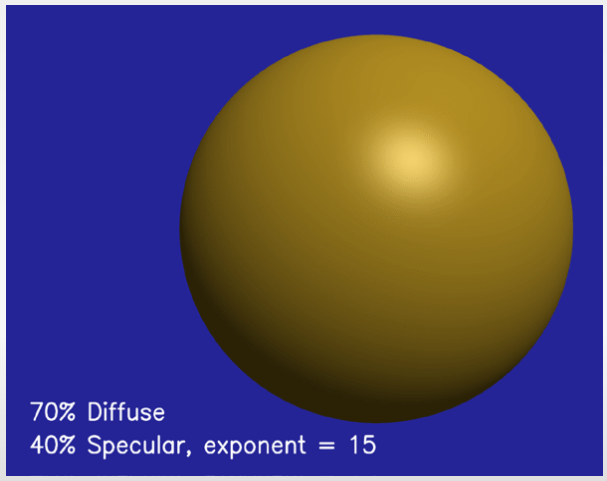
Lighting models

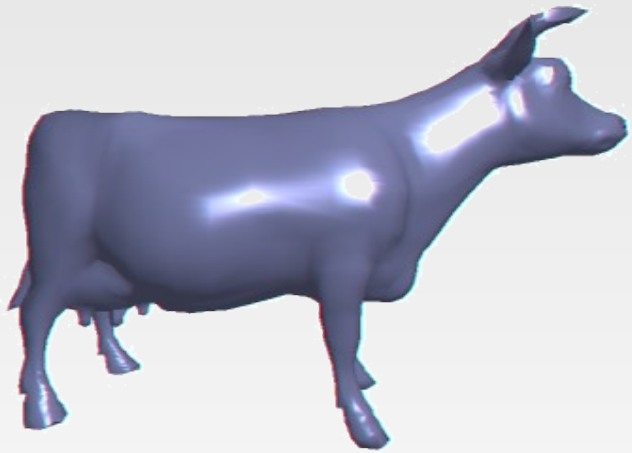
- Mixing specular and diffuse reflexion





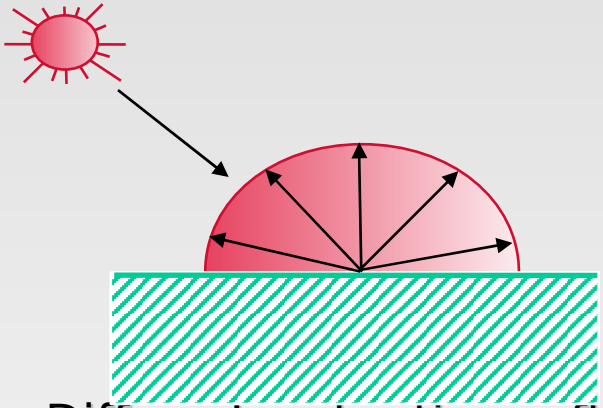






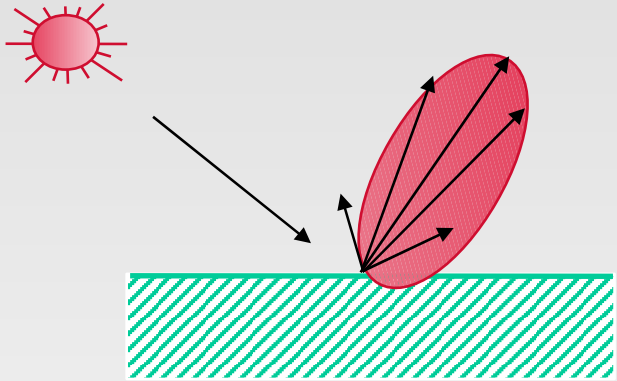
Lighting models

- Light reflexion



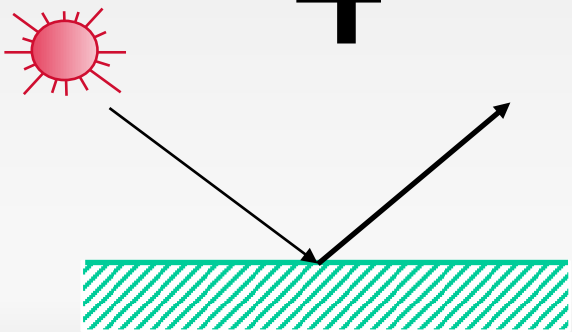
Diffuse Lambertian reflexion

+



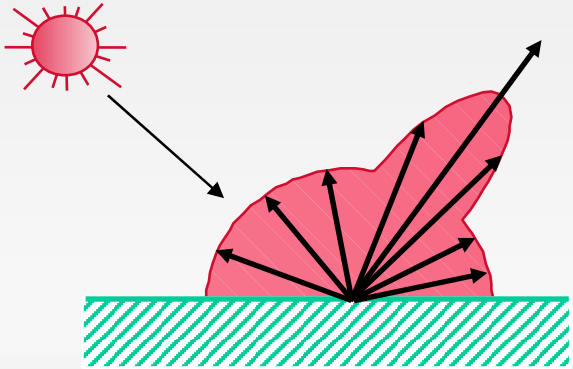
Imperfect specular reflexion

+



Perfect specular reflexion

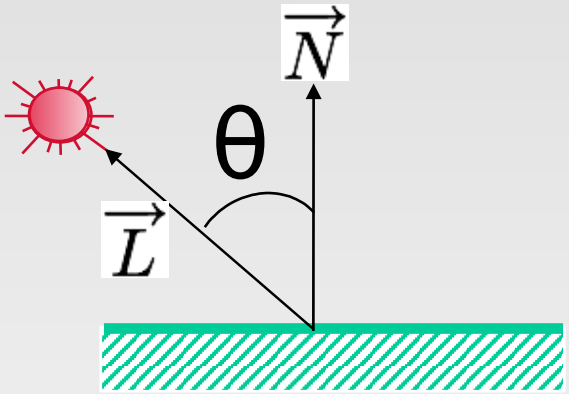
=



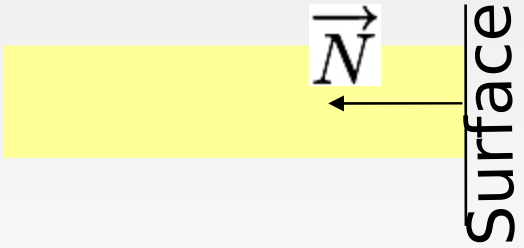
Final reflexion function

Lighting models

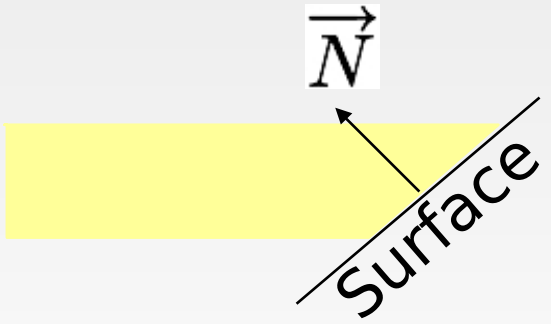
- Lambert's law



$$I = I_l.K_d.cos(\theta) \quad \text{with} \quad 0 \leq \theta \leq \pi/2 \quad \text{and} \quad 0 \leq k_d \leq 1$$



Maximum light received per surface unit

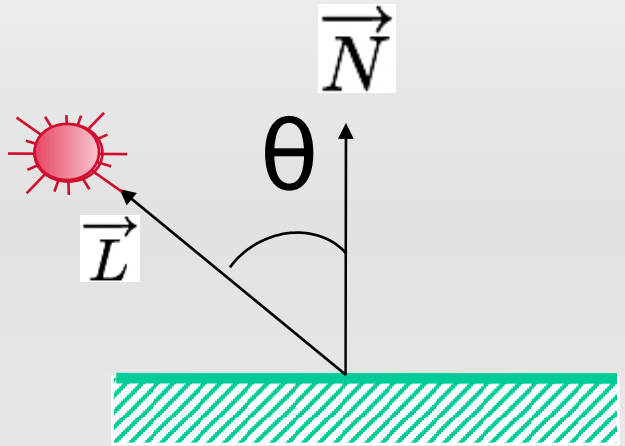


Less light received per surface unit

Lighting models

- Lambert's law
  - Taking ambient lighting into account :

$$I = I_a.k_a + I_l.k_d.cos(\theta)$$



  - Taking attenuation by distance into account :

$$I = I_a.k_a + I_l.k_d.cos(\theta).fatt$$

  - Taking colors into account :

$$I_c = I_{ac}.k_{ac} + I_{lc}.k_{dc}.cos(\theta).fatt$$

$$I_m = I_{am}.k_{am} + I_{lm}.k_{dm}.cos(\theta).fatt$$

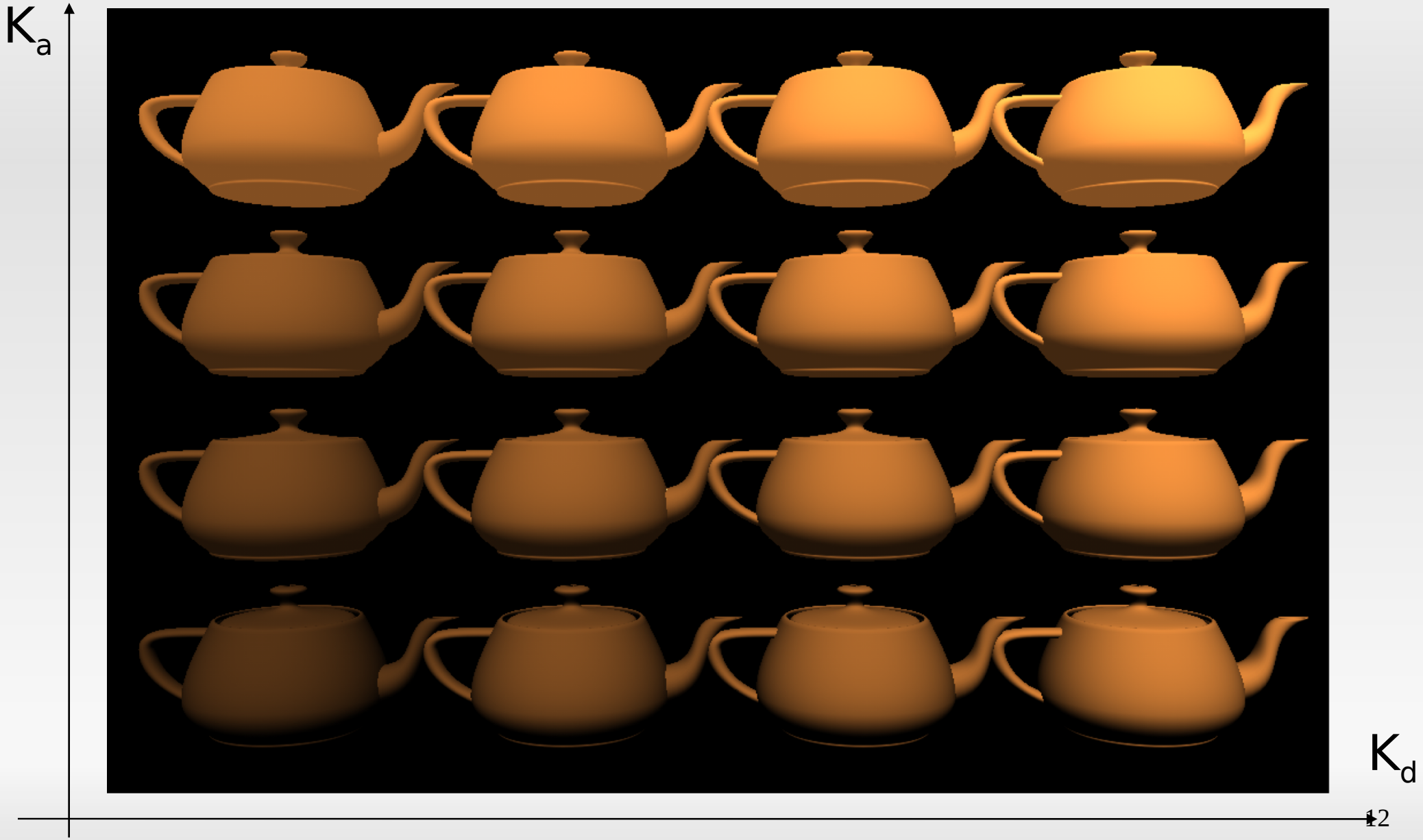
$$I_j = I_{aj}k_{aj} + I_{lj}k_{dj}\cos\theta\,.\,fatt$$

Lighting models

∀ Effects of increasing k_a

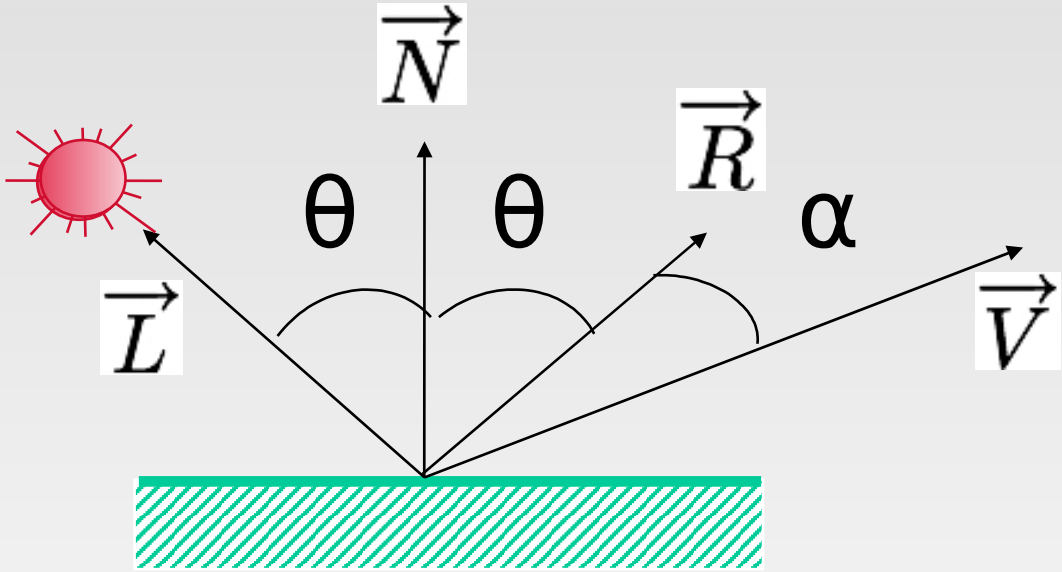
∀ Effects of increasing k_d $(k_a = 0)$

Lighting models



Lighting models

- Phong lighting model:

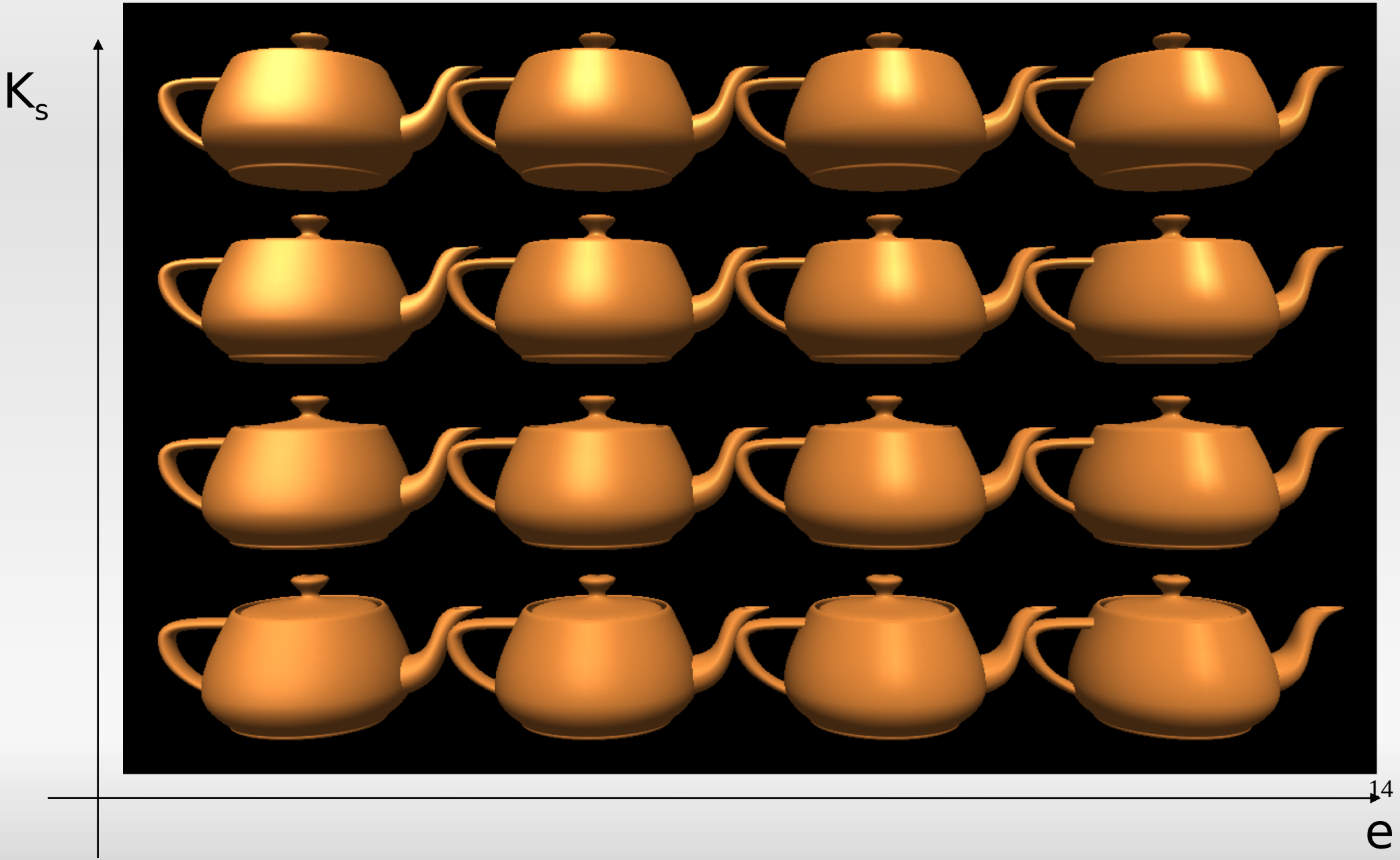


$$I = I_a . k_a + I_l . (k_d . \cos(\theta) + k_s . \cos(\alpha)^e)$$

If vectors are normalized:

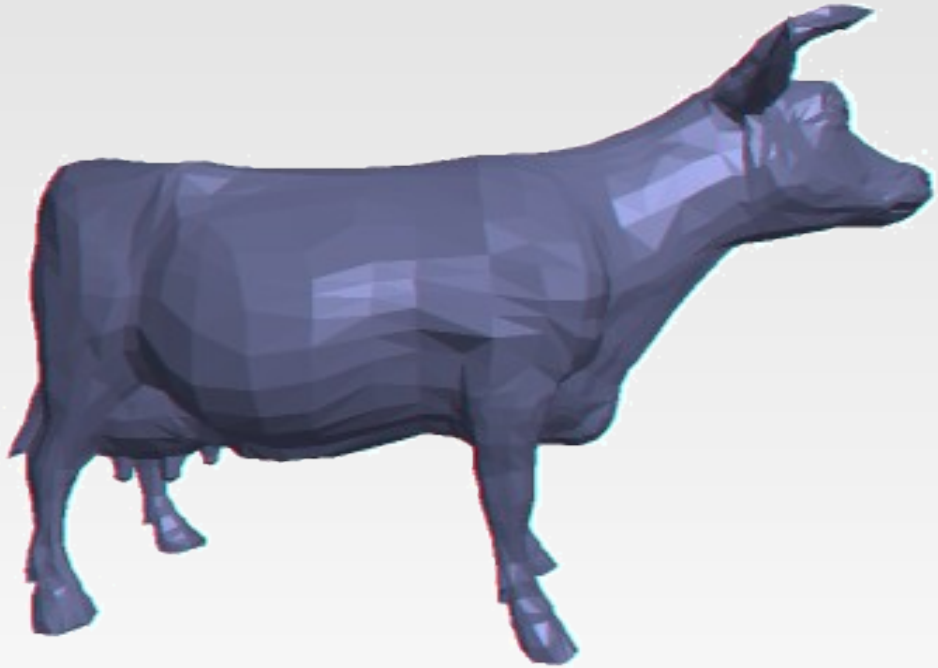
$$I = I_a . k_a + I_l . (k_d . (\vec{L} . \vec{N}) + k_s . (\vec{R} . \vec{V})^e)$$

Lighting models



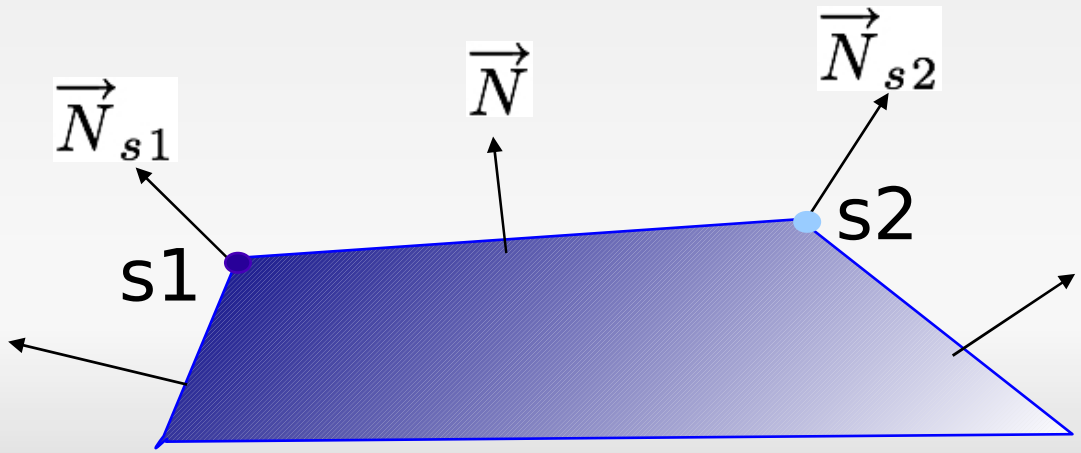
Shading models

- Flat shading
  - One illumination value computed per polygon



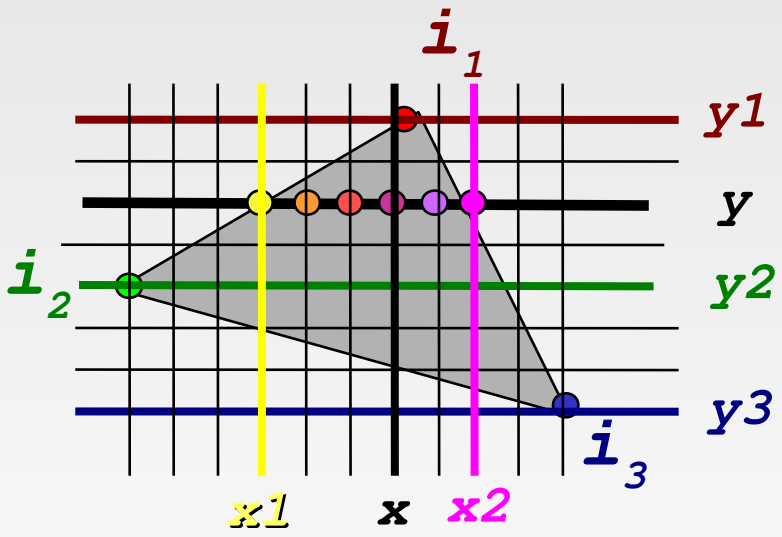
Shading models

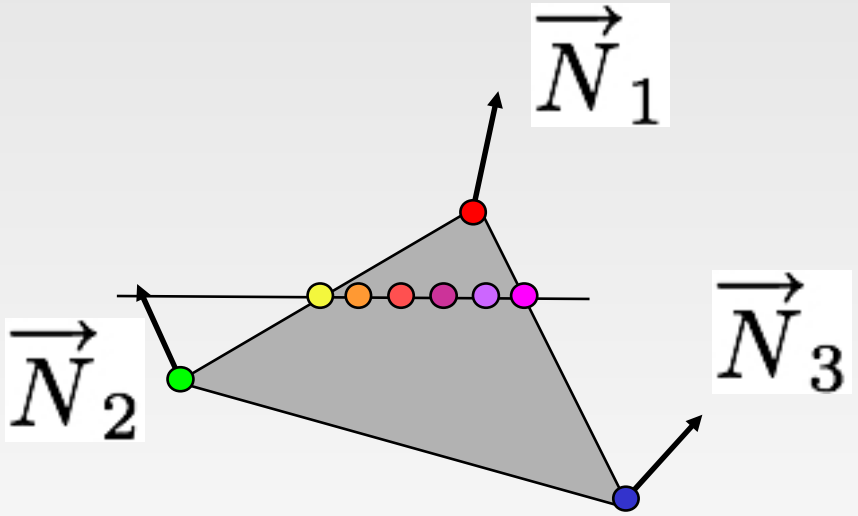
- Gouraud shading
  - Per vertex lighting computation
  - Normals at the vertices are computed by interpolating normals of the polygons



Shading models

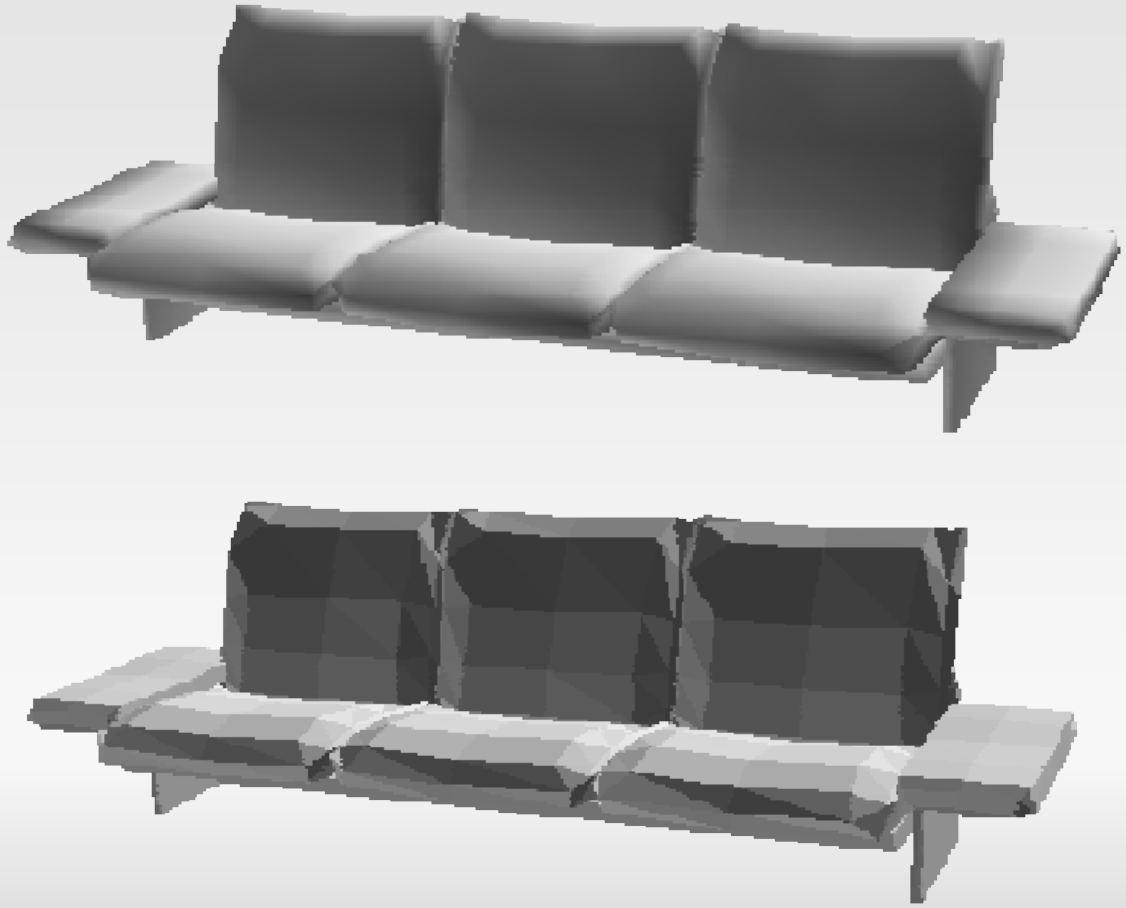
- Gouraud shading
  - Colors are bilinearly interpolated from the illumination computed at the vertices





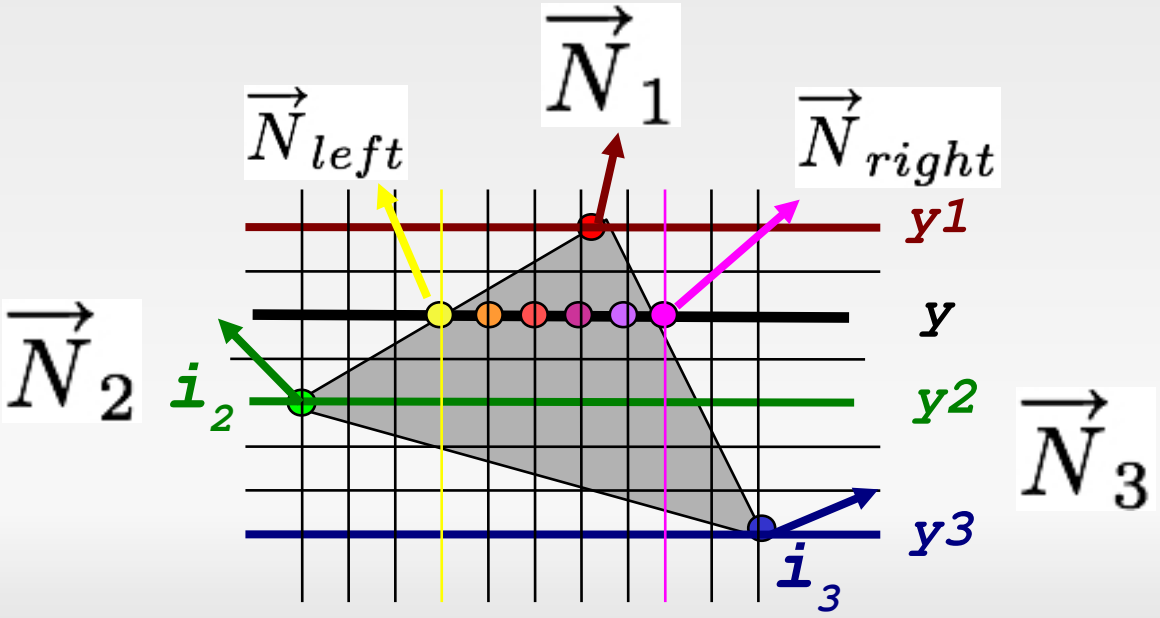
Shading models

- Gouraud shading vs flat shading



Shading models

- Phong shading model (not Phong lighting model...)
  - Normals are bilinearly interpolated from the normals computed at the vertices
  - Lighting model applied at each point of the polygon



Light sources in OpenGL

- Creating light sources
  - Properties
    - type (ambiant, diffuse, specular, spot light, etc.)
    - color
    - position
    - direction
    - etc.
  - void glLight{if} (GLenum light, GLenum pname, TYPE param);
    void glLight{if}v (GLenum light, GLenum pname, TYPE *param);
    - light : ID of the light to set : GL_LIGHT0, ….. GL_LIGHT7
    - pname : ID of the parameter
    - param : input data (value of the parameter)

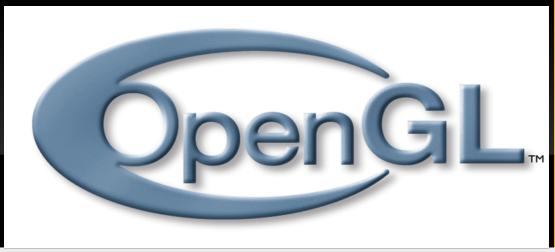
Light sources in OpenGL

- Activation
  - glEnable(GL_LIGHTING); // activates lighting
  - glDisable(GL_LIGHTING); // deactivates lighting

  - glEnable(GL_LIGHTi); // switches on light #i
  - glDisable(GL_LIGHTi); // switches off light #i

  - Getting the maximum number of lights available (at least 8):
    - glGetIntegerv(GL_MAX_LIGHTS, &nbLights)

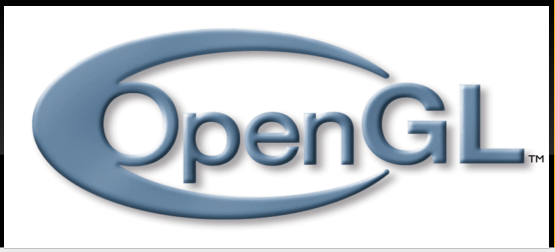
Light sources in



- Example:
  - GLfloat lumAmbiant [ ] = { 0.0, 0.0, 0.0, 1.0}; //{Black, 1.0};
  - GLfloat lumDiffuse [ ] = { 1.0, 1.0, 1.0, 1.0}; // {White, 1.0};
  - GLfloat lumSpecular [ ] = { 1.0, 1.0, 1.0, 1.0}; // {White, 1.0};
  - GLfloat lumPosition [ ] = { 1.0, 1.0, 1.0, 1.0};

  - glLightfv (GL_LIGHT0, GL_AMBIENT, lumAmbiant);
  - glLightfv (GL_LIGHT0, GL_DIFFUSE, lumDiffuse);
  - glLightfv (GL_LIGHT0, GL_SPECULAR, lumSpecular);
  - glLightfv (GL_LIGHT0, GL_POSITION, lumPosition);
  - glEnable (GL_LIGHT0); // switch the light on

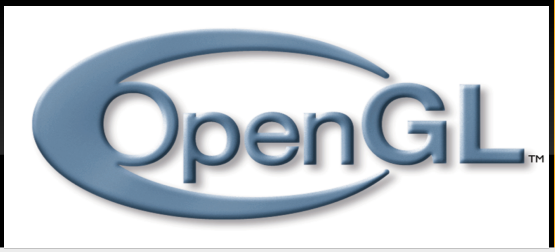
Light sources in



- Setting the position of a light:
  - In homogeneous coordinates
  - GLfloat lumPosition[ ]={ 1.0, 1.0, 1.0, 1.0}; // point light
  - GLfloat lumPosition[ ]={ 1.0, 1.0, 1.0, 0.0}; // directional light
  - glLightv (GL_LIGHT0, GL_POSITION, lumPosition);

- Setting the attenuation by distance factor
  - glLightf (GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
  - glLightf (GL_ LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
  - glLightf (GL_ LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);

Light sources in



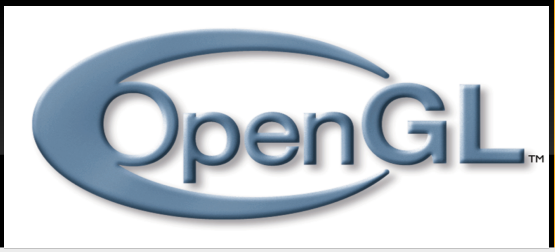
- Default values :
  - GLfloat ambient[ ] = {0.0, 0.0, 0.0, 1.0};
  - GLfloat diffuse[ ] = {1.0, 1.0, 1.0, 1.0};
  - GLfloat specular[ ] = {1.0, 1.0, 1.0, 1.0};
  - GLfloat light0Position [ ] = {1.0, 1.0, 1.0, 0.0};

  - GLfloat specular [ ] = {0.8, 0.8, 0.8, 1.0};
  - GLubyte shinyObj = 128;

Light sources in OpenGL

- Setting the components of the lighting model:
  - Intensity of the global ambient lighting
  - Lighting of front and / or back faces
    - glLightModel{if} (GLenum pname, TYPE param);
    - glLightModel{if}v (GLenum pname, TYPE *param);
- Setting the shading model:
  - glShadeModel(GLenum mode);
    - mode : either GL_SMOOTH (Gouraud) or GL_FLAT

Light sources in



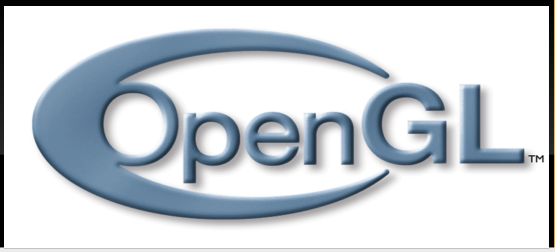
- Setting the material components
  - void glMaterial{if} (GLenum face, GLenum pname, TYPE param);
  - void glMaterial{if}v (GLenum face, GLenum pname, TYPE *param);

  - face : GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
  - pname : GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_SHININESS, GL_AMBIENT_AND_DIFFUSE, GL_COLOR_INDEXES

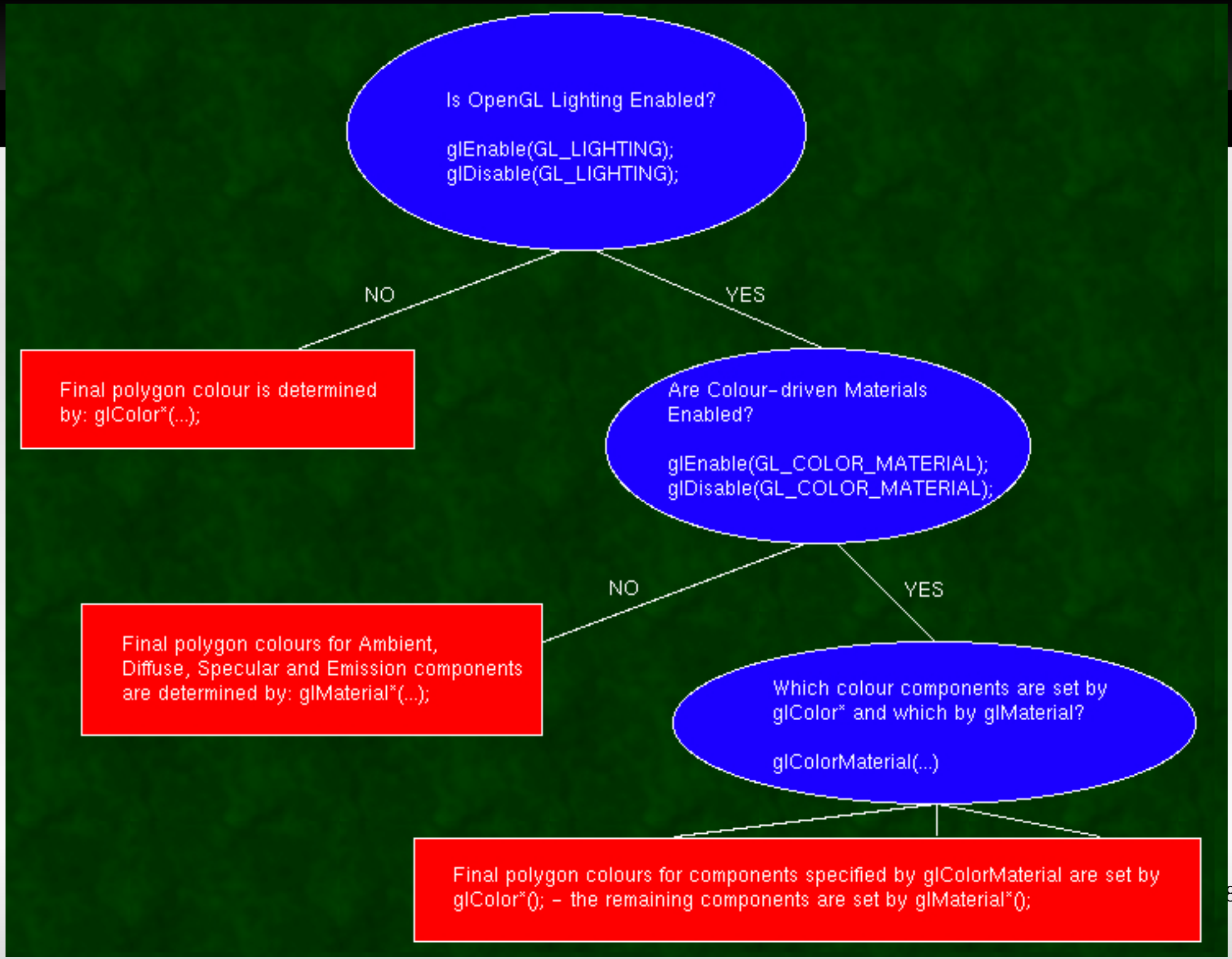
  - params : value of the parameter

  - Exemples :
    - glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
    - glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuse);
    - glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);

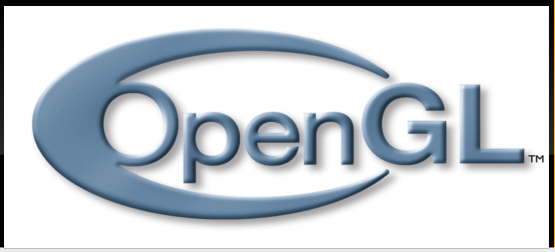
Light sources in



- Mode COLOR_MATERIAL : uses values defined in glColor(...) to set properties of the material
  - glEnable(GL_COLOR_MATERIAL);
  - glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
    - glColor() controls diffuse and ambient components of the material
  - glColorMaterial(GL_FRONT_AND_BACK, GL_SPECULAR);
    - glColor3f() controls the specular component


Is OpenGL Lighting Enabled?
glEnable(GL_LIGHTING);
glDisable(GL_LIGHTING);
NO
YES
Final polygon colour is determined by: glColor*(...);
Are Colour-driven Materials Enabled?
glEnable(GL_COLOR_MATERIAL);
glDisable(GL_COLOR_MATERIAL);
NO
YES
Final polygon colours for Ambient, Diffuse, Specular and Emission components are determined by: glMaterial*(...);
Which colour components are set by glColor* and which by glMaterial?
glColorMaterial(...)
Final polygon colours for components specified by glColorMaterial are set by glColor*(); - the remaining components are set by glMaterial*();

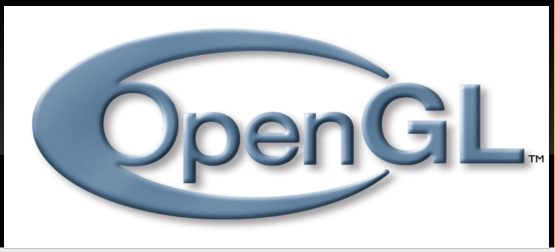
Light sources in



- Position of a light source
  - glLightfv(GL_LIGHT0, GL_POSITION, position);
  - 'position' is multiplied by the current modelview matrix
  - for a static light source:

```
glMatrixMode ( GL_PROJECTION );
glLoadIdentity();
glPersective ( …);
glMatrixMode ( GL_MODELVIEW )
glLoadIdentify ();
...
// right after gluLookAt(…);
glLightfv ( LG_LIGHT0, GL_POSITION, position );
```

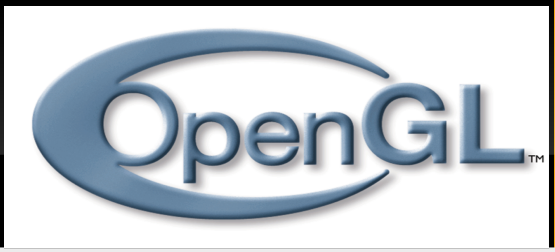
Light sources in



- Position of a light source
  - for a moving light source:

```
void display ( void ) {
    GLfloat lumPosition[] = {0.0, 0.0, 1.5, 1.0};
    …
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
      gluLookAt ( …);
      glPushMatrix (); //stores the current modelview matrix
      glRotatef ( spin, axeX, axeY, axeZ);
      glLightfv ( GL_LIGHT0, GL_POSITION, lumPosition);
      …
      glPopMatrix (); //restores the modelview matrix
      …
}
```

Light sources in



- Setting normals at the vertices:
  - void glNormal3{b d f i s}v(const GLbyte *v);
    void glNormal3{b d f i s}(TYPE nx, TYPE ny, TYPE nz);
    - Sets the normal for the next vertices
    - glBegin(GL_TRIANGLES)
      glNormal3f(0.f, 0.f, 1.f);
      glVertex3f(1.f, 0.f, 0.f); // has normal (0, 0, 1)
      glNormal3f(0.f, 1.f, 1.f);
      glVertex3f(0.f, 1.f, 0.f); // has normal (0, 1, 1)
      glVertex3f(1.f, 1.f, 0.f); // has normal (0, 1, 1)
      glEnd();
- Forcing unit normals
    - glEnable(GL_NORMALIZE);

Tutorial: lighting and transformations

- Write an application showing a red teapot centered at (0, 0, 0). The camera uses a perspective projection and is located at (0,0,-10)
  - gluPerspective(fovy, aspect_ratio, zNear, zFar);
  - gluLookAt(eyeX, eyeY, eyeZ, ctrX, ctrY, ctrY, upX, upY, upZ);
- Activate lighting (glEnable(GL_LIGHTING)) right before glutMainLoop(). What do you see? Activate Light #0, then enable depth test. Do not forget to clear the depth buffer at the beginning of the display function.
- Change properties of light #0: set its position to (-5, 5,-5), its ambient component to (0.2, 0.1, 0) and its diffuse component to (1, 0.5, 0) (orange)
- Make the light rotate around the teapot (around the y axis)
- Add another light: position (5,0,-5); diffuse (1,1,1); ambient (0,0,0);
- Change the material of the teapot: diffuse(0,0,1); ambient (0,0,1);