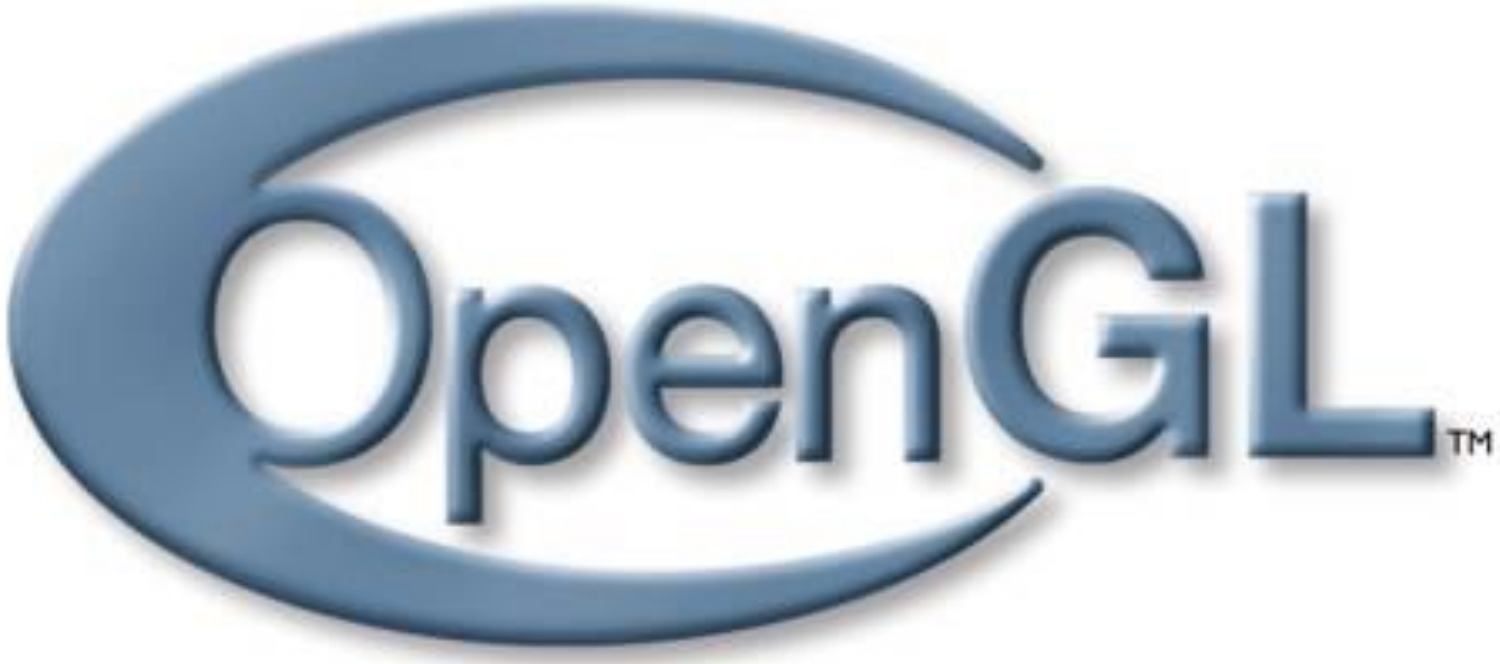


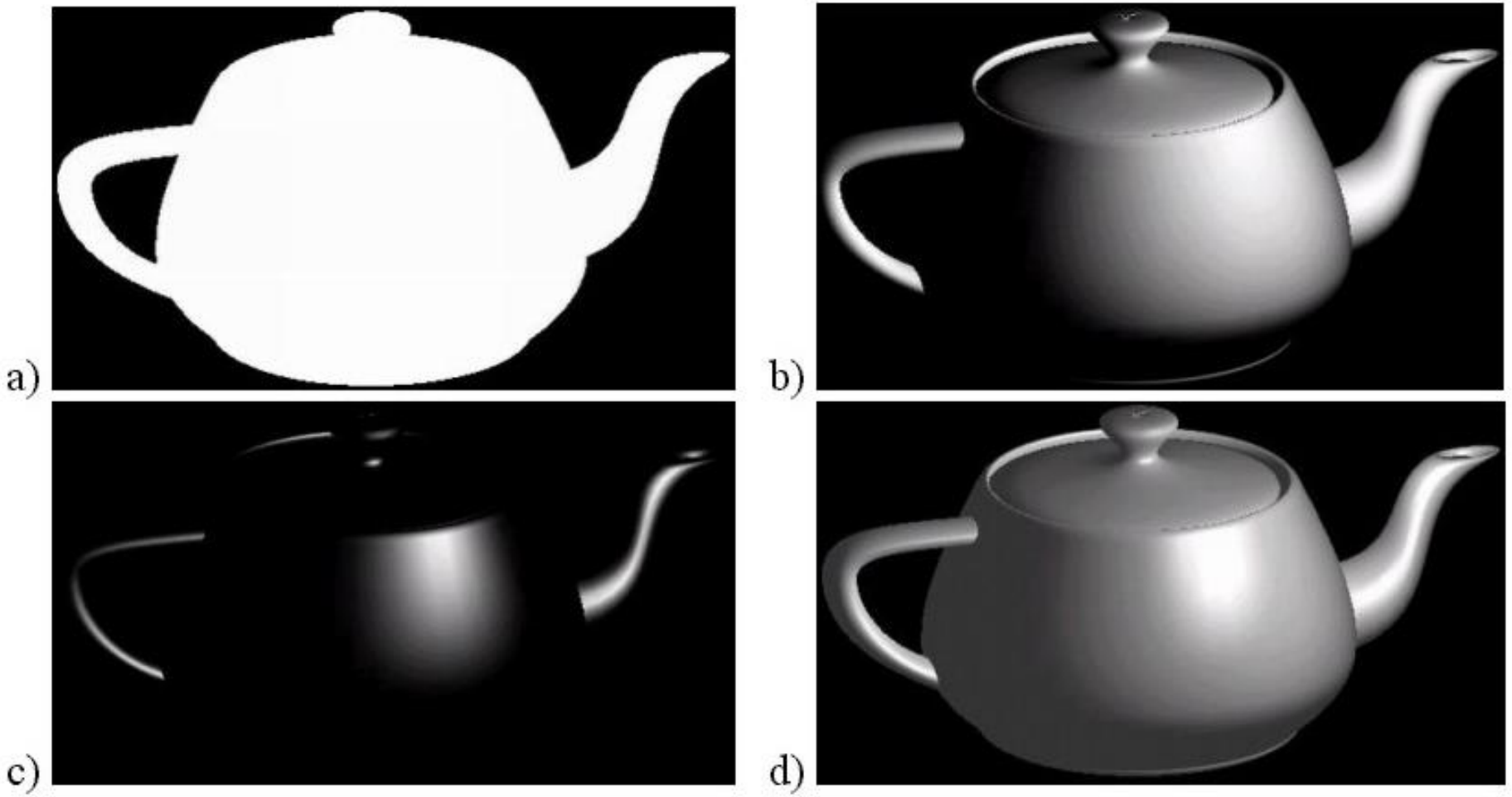
part 7

Martin Samuelčík
http://www.sccg.sk/~samuelcik
Room I4

Lighting

- Bringing lights into virtual scene
- Simulating and approximating physics laws
- More realistic rendering, better visual description, perception of surfaces
- Several models of approximation
- Light source = emitting of energy into scene
- Material = description how light interferes with surface of object

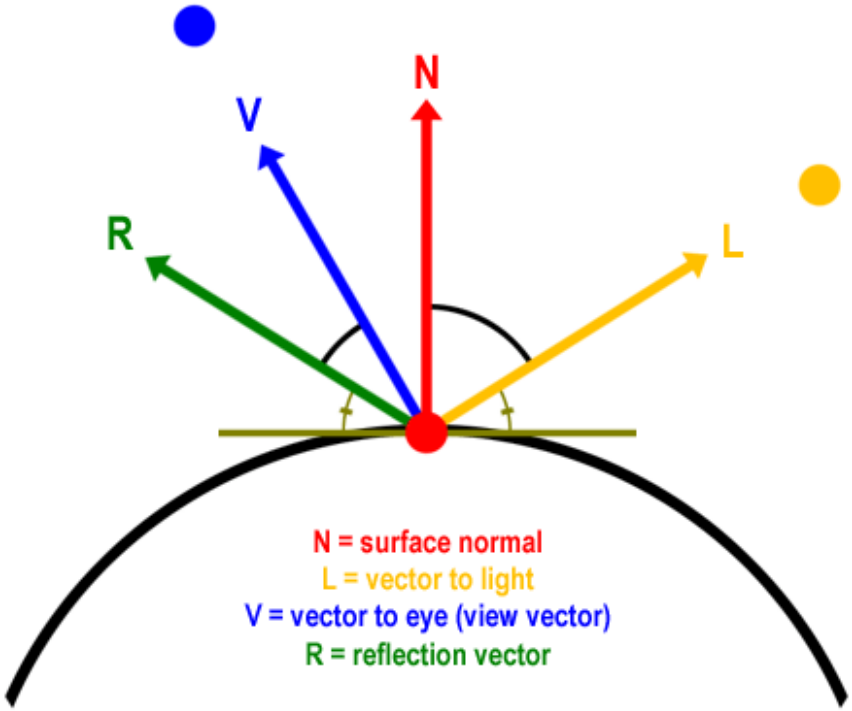
Lighting

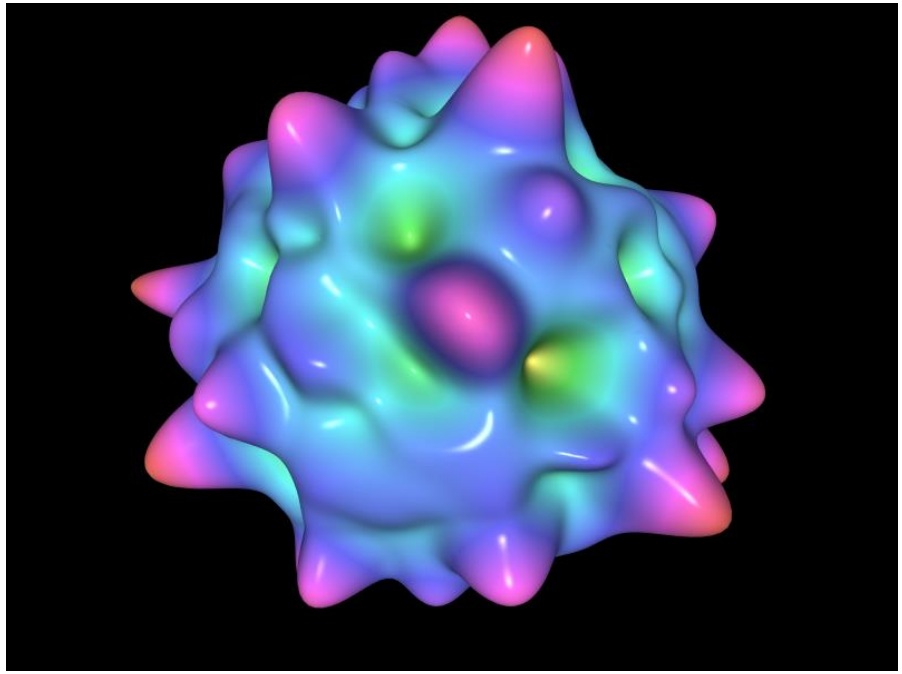


Phong lighting model

- Most used approximation of lighting in computer graphics
- Computation of final color at point on surface of object
- Inputs for computation
  - Lights – emitting energy that hits the point, given by position, direction
  - Material – how the energy is absorbed ans scattered at point
  - Normal – local description of geometry at point

Phong lighting model





OpenGL®

Phong lighting model

- Local model – only direct influence of lights is computed, no bounces of light
- Model consists of 3 basic components, each light and material has these 3 components
- Component of light or material is one RGB color
- When combining contributions from light sources
  - Summation of contributions
  - $(R_1+R_2,G_1+G_2,B_1+B_2)$
  - Sum can't exceed 1
- When combining incoming light and material at point on surface
  - $(R_L*R_M,G_L*G_M,B_L*B_M)$

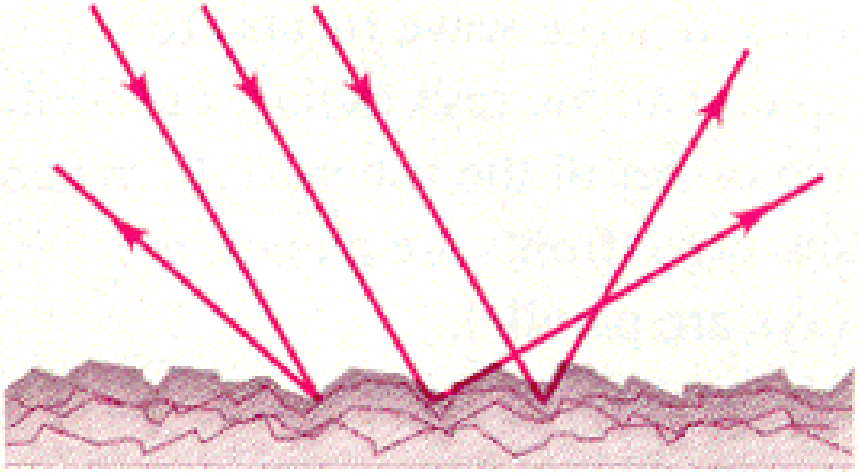
Ambient component

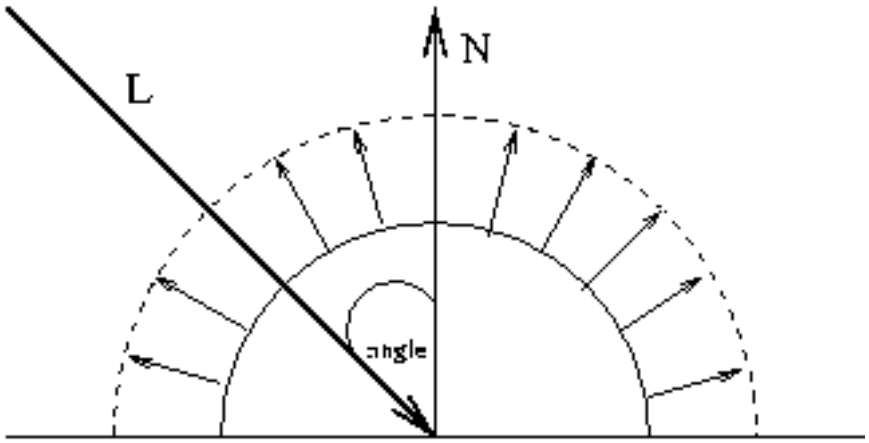
- Simulating light scattered by environment multiple times
- Constant ambient material color over whole surface
- Does not simulate occlusion of near geometry – no global solution
- Local and global ambient components
- No contribution of light position or direction
- No contribution of surface shape

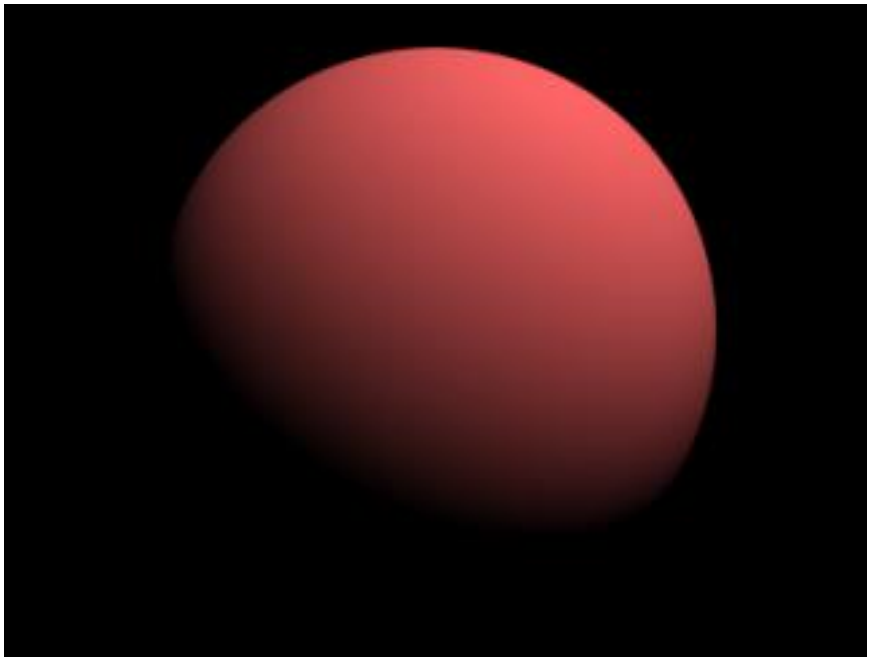
Diffuse component

- Simulating scattering of light on micro facets in the close neighborhood at point on surface
- Lights scatters in all directions – no dependency on view of point
- Statistically there is more micro facets parallel to surface then perpendicular to surface – intensity of diffuse component is based on angle between surface and incoming light direction

Diffuse component





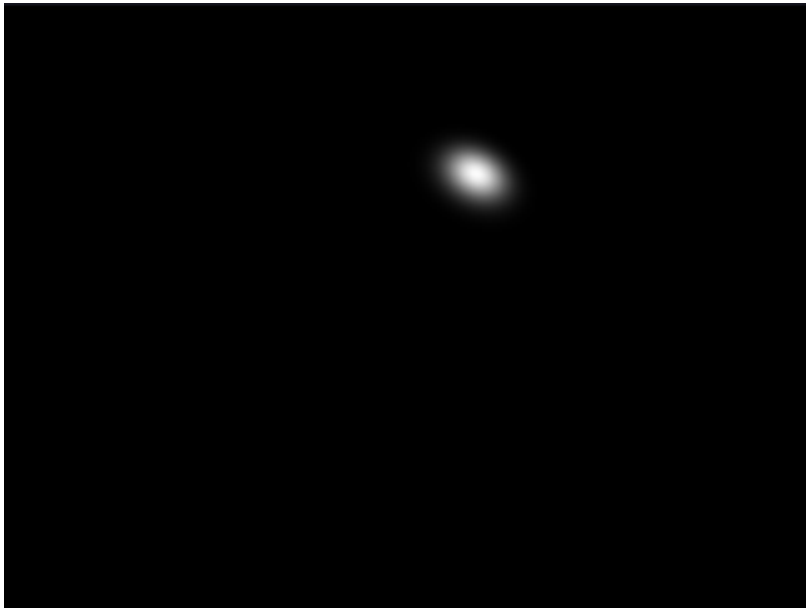


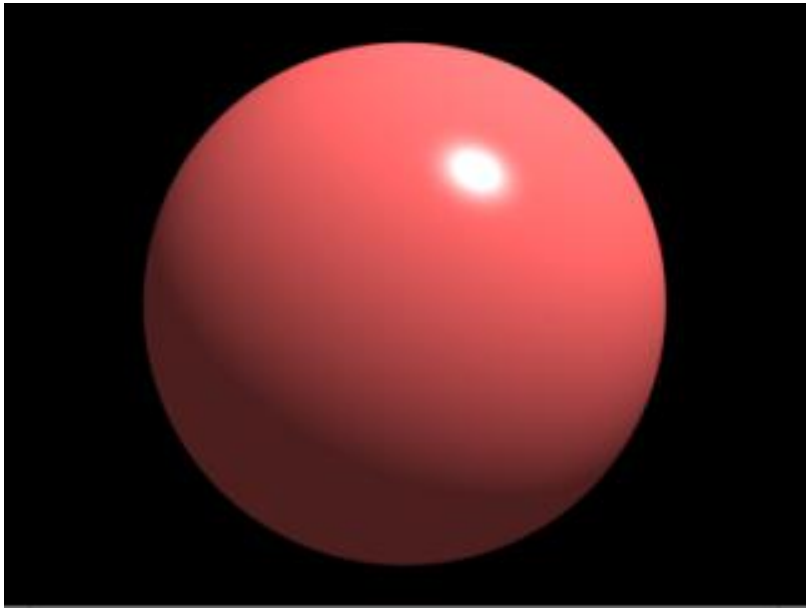
OpenGL®

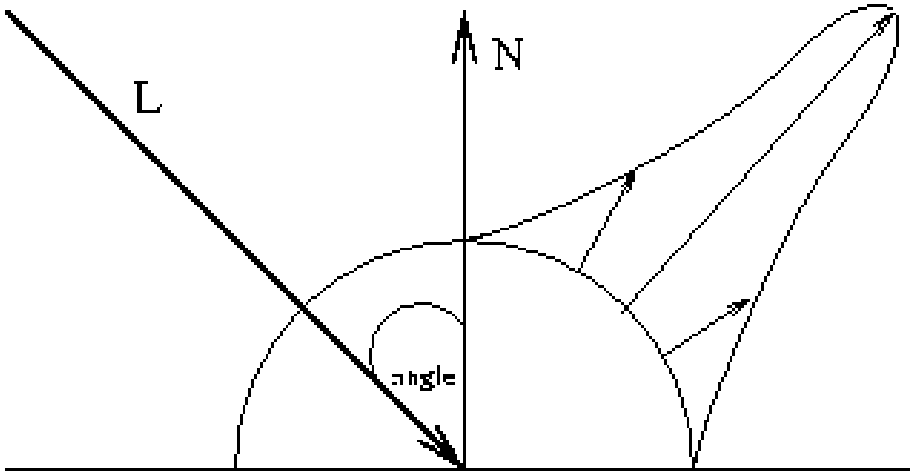
Specular component

- Simulating ideal reflection of light at point on surface
- Light is reflected based on physical law of reflection – angle of approaching ray and surface is equal to angle of reflected ray and surface
- Component depends on position of viewer – if viewer lies at reflected view, the component has maximal intensity

Specular component







Emission

- Objects can emit energy
- Simple addition of colors to computation
- Only in material as fourth component, no light component
- It is not treated as another light source, does not affect other objects

Lights

- Only point and directional lights in Phong lighting model
- Attenuation - decreasing energy by distance of light and point on surface
- Point lights
  - Omni light – emitting energy from point in all direction, given by position
  - Reflectors – emitting energy from point in cone-like shape, given by position, direction, cone angle and cone attenuation
- Directional lights
  - Emitting energy from point at infinity
  - Given by direction

Phong computation

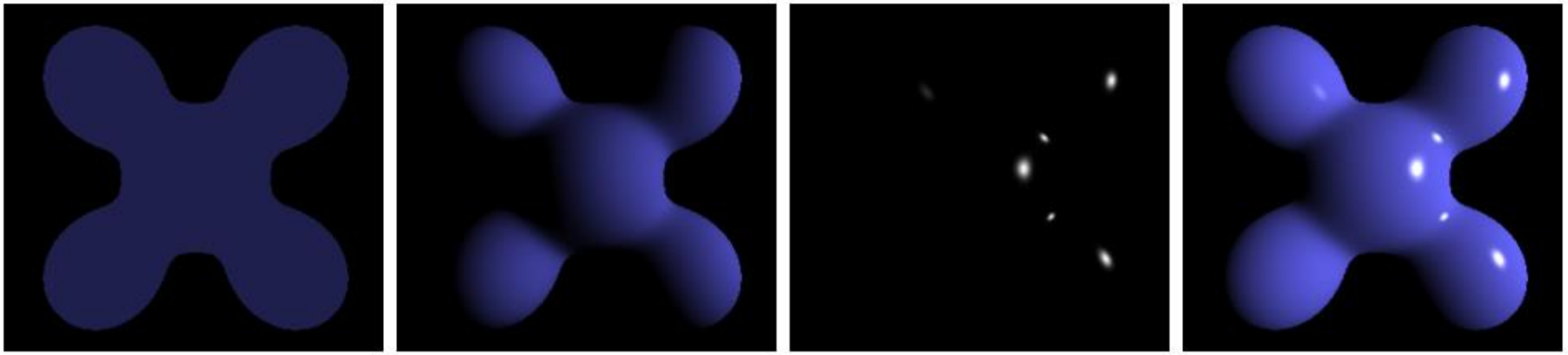
$$final\_color = emission_{material} + ambient_{light\_model} * ambient_{material} +$$

$$\sum_{i=0}^{n-1} \left(\frac{1}{k_c + k_l * d + k_q * d^2}\right) * spotlighteffect_i *$$

$$[ambient_{light} * ambient_{material} + (\max(L.N, 0)) * diffuse_{light} * diffuse_{material} +$$

$$(\max(S.V, 0))^{shininess} * specular_{light} * specular_{material}]_i$$

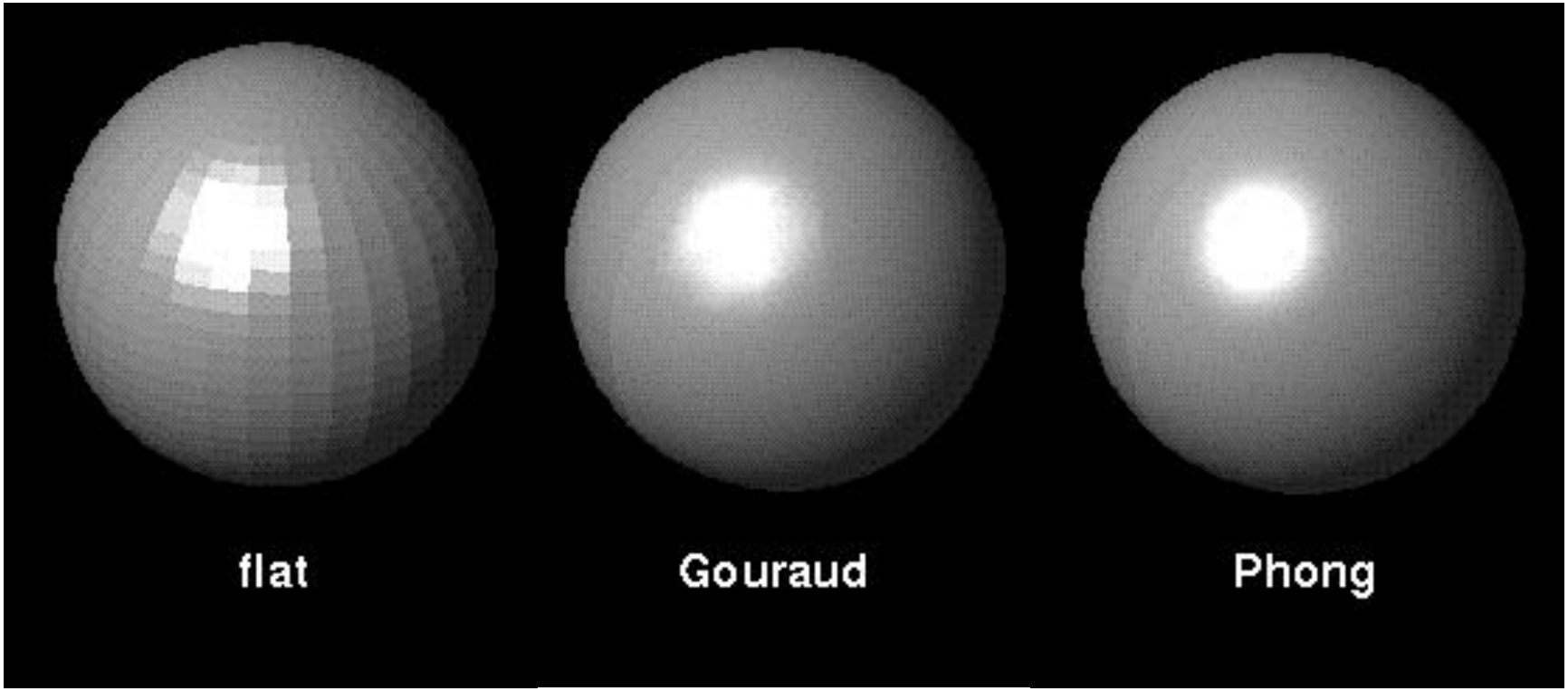
d-distance of point and light
L-vector from point to light
N-normal of surface at point
V-vector from point to camera
S-reflected vector L around normal N

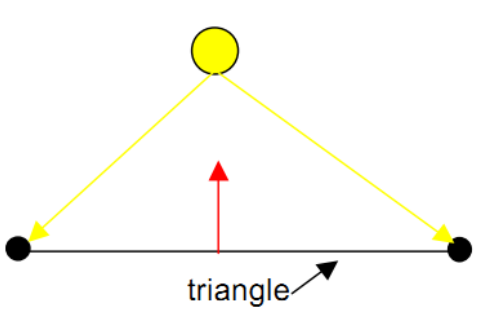


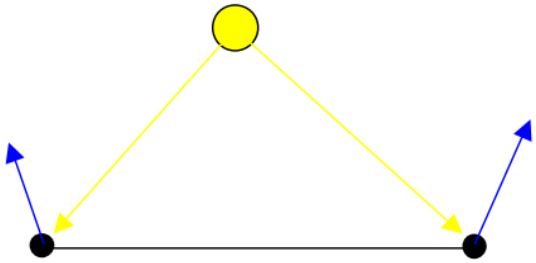
Shading

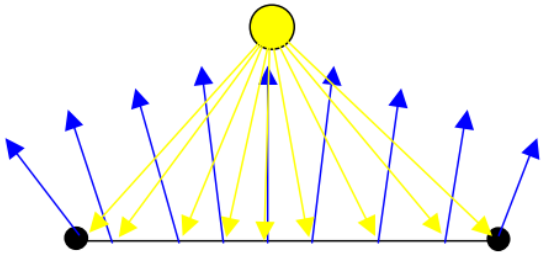
- Phong lighting model computes final color for points on surface (inside polygon), how to compute colors for whole rendered polygon?
  - **Flat shading** – illumination is computed for only one point of polygon (one vertex, barycenter), and computed color is copied to each pixel of polygon
  - **Gourard (per-vertex) shading** – illumination is computed for each vertex of polygon, for other pixels, color is interpolated using colors from vertices
  - **Phong shading (per-pixel)** – illumination is computed for each pixel separately, normal in pixel computation is computaed using interpolation of vertex normals

Shading









OpenGL lighting

- OpenGL 1.0 fixed function pipeline implementing Phong illumination model with flat and Gourard shading
  - Flat – glShadeModel(GL_FLAT)
  - Gourard – glShadeModel(GL_SMOOTH)
- Computing per-vertex lighting, computed color replaces current color of vertex
- Preparing OpenGL lighting
  - Assign normals to vertices
  - Create and initialize lights
  - Select a lighting model
  - Define materials for objects

OpenGL lights

- Defined 8 lights
- All disabled by default
- **void glLight{if}[v](GLenum light, GLenum pname, TYPE param)**
  - Specifying properties of light
  - light – GL_LIGHT0,…,GL_LIGHT7
  - pname – name of property
  - param – value of property

OpenGL lights

Parameter name	Default value	Meaning
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	ambient RGBA intensity of light
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	diffuse RGBA intensity of light
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	specular RGBA intensity of light
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(*x, y, z, w*) position of light, in object
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	(*x, y, z*) direction of spotlight
GL_SPOT_EXPONENT	0.0	spotlight exponent, attenuation from axis
GL_SPOT_CUTOFF	180.0	spotlight cutoff angle
GL_CONSTANT_ATTENUATION	1.0	constant attenuation factor, k_c
GL_LINEAR_ATTENUATION	0.0	linear attenuation factor, k_l
GL_QUADRATIC_ATTENUATION	0.0	quadratic attenuation factor, k_q

Setting position

- Point light = real position
  - Fourth coordinate, w <> 0
- Directional light = position at infinity
  - Fourth coordinate, w = 0
- When setting position, given light position is transformed by current modelview matrix and stored in camera coordinates, same for spotlight direction
- One can define lights that are at camera position, that are rotating around object or that are at fixed position

OpenGL lighting

- Enabling/disabling lights
  - glEnable(GL_LIGHTi), glEnable(GL_LIGHTi), i=0,1,…,7
- Enabling/disabling lighting
  - glEnable(GL_LIGHTING), glDisable(GL_LIGHTING)
- **void glLightModel{if}[v](GLenum pname, TYPE param)**

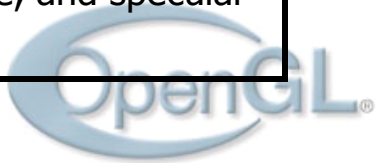
pname	Def value	Meaning
GL_LIGHT_MODEL_AMBIENT	(0.2,0.2,0.2,1.0)	Ambient RGBA intensity of the entire scene
GL_LIGHT_MODEL_LOCAL_VIEWER	GL_FALSE	How specular reflection angles are computed
LIGHT_MODEL_TWO_SIDE	GL_FALSE	Choose between one-sided or two-sided lighting

OpenGL materials

- Current material is state variable, separately for front and back face
- When rendering primitive, current material is attached to that primitive
- Changing current material
- **void glMaterial{if}[v](GLenum face, GLenum pname, TYPE param)**
  - face – GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
  - pname – identifier of property
  - pram – value of parameter

OpenGL materials

Parameter name	Default value	Meaning
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	ambient color of material
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	diffuse color of material
GL_AMBIENT_AND_DIFFUSE		ambient and diffuse color of material
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	specular color of material
GL_SHININESS	0.0	specular exponent
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	emissive color of material
GL_COLOR_INDEXES	(0,1,1)	ambient, diffuse, and specular color indices



Color material

- When enabled, current color is also copied to specified component of current material
- **void glColorMaterial(GLenum face, GLenum mode)**
  - face – GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
  - mode - GL_EMISSION, GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_AMBIENT_AND_DIFFUSE
- Enable/disable color material
  - By default, it is disabled
  - glEnable(GL_COLOR_MATERIAL)
  - glDisable(GL_COLOR_MATERIAL)

Example

```
GLfloat light_ambient[] = { 0.0f, 0.0f, 0.0f, 1.0f };
GLfloat light_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
GLfloat light_position[] = { 1.0f, 1.0f, 1.0f, 0.0f };
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
glEnable(GL_LIGHT0);

glEnable(GL_LIGHTING);
GLfloat mat_ambient[] = { 0.5f, 0.5f, 0.5f, 1.0f };
GLfloat mat_diffuse[] = { 1.0f, 0.5f, 0.0f, 1.0f };
GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, 25.0f);
renderCube();

glColorMaterial(GL_FRONT, GL_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);
glColor3f(0.0f, 0.0f, 1.0f);
renderCube();
glDisable(GL_COLOR_MATERIAL);
```

Shaders & lighting

- Flat & Gourard shading – computation of lighting in vertex shader
- Phong shading - computation of lighting in fragment shader
- All computations with vector must be in same coordinate space
- For Phong lighting computations, we need in shaders current material, normal and parameters of lights
  - Lights, materials – uniforms
  - Per-vertex normal – attribute
  - For Phong shading, we need normal, view and light vectors in fragment shader – varying

GLSL per-vertex lighting

```
// VERTEX SHADER
void main(void)
{
   vec4 V_eye = gl_ModelViewMatrix * gl_Vertex;
   vec4 L_eye = gl_LightSource[0].position - V_eye;
   vec4 N_eye = vec4(gl_NormalMatrix * gl_Normal, 1.0);
   V_eye = -V_eye;

   float diffuse = clamp(dot(L_eye, N_eye), 0.0, 1.0);
   vec4 R_eye = reflect(N_eye, L_eye);
   float specular = pow(clamp(dot(R_eye, V_eye), 0.0, 1.0), 16.0);

   vec4 color = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;
   color += diffuse * gl_LightSource[0].diffuse * gl_FrontMaterial.diffuse;
   color += specular * gl_LightSource[0].specular * gl_FrontMaterial.specular;

   gl_FrontColor = color;
   gl_BackColor = color;
   gl_TexCoord[0] = gl_MultiTexCoord0;
   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}

// FRAGMENT SHADER
uniform sampler2D tex;

void main(void)
{
   gl_FragColor = gl_Color * texture2D(tex, gl_TexCoord[0].st);
}
```

GLSL per-pixel lighting

```
// VERTEX SHADER
varying vec4 V_eye;
varying vec4 L_eye;
varying vec4 N_eye;

void main(void)
{
   V_eye = gl_ModelViewMatrix * gl_Vertex;
   L_eye = gl_LightSource[0].position - V_eye;
   N_eye = vec4(gl_NormalMatrix * gl_Normal, 1.0);
   V_eye = -V_eye;

   gl_TexCoord[0] = gl_MultiTexCoord0;
   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

```
// FRAGMENT SHADER
varying vec4 V_eye;
varying vec4 L_eye;
varying vec4 N_eye;
uniform sampler2D tex;

void main(void)
{
   vec4 V = normalize(V_eye);
   vec4 L = normalize(L_eye);
   vec4 N = normalize(N_eye);

   float diffuse = clamp(dot(L, N), 0.0, 1.0);
   vec4 R = reflect(N, L);
   float specular = pow(clamp(dot(R, V), 0.0, 1.0), 16.0);

   vec4 color = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;
   color += diffuse * gl_LightSource[0].diffuse * texture2D(tex, gl_TexCoord[0].st);
   color += specular * gl_LightSource[0].specular * gl_FrontMaterial.specular;

   gl_FragColor = color;
}
```

The End!

Questions?