

OpenGL works in two modes

- 1) Color is specified by user (`glColor*`)
- 2) Color is calculated from the illumination model
i.e., lights and material properties

The first mode is set by `glDisable(GL_LIGHTING)`

The second mode is set by `glEnable(GL_LIGHTING)`

Color is calculated *per vertex*

In OpenGL 2.0 the color is calculated also per pixel

Principle

- 1) Define material properties
- 2) Define normal vectors at the vertex
- 3) Put lights in the 3D space

that's it

ad 1) `glMaterial*` makes it

ad 2) `glNormal*` makes it

ad 3) `glLight*` makes it



Color is calculated per vertex

How is the color calculated between vertices?

There are two ways

- 1) Constant shading (flat shading)
Color is constant, given by the last color provided
- 2) Color is interpolated (Gouraud shading) [guroood]

Changing the shading mode

```
void glShadeModel(GLenum mode)
```

where

`mode` is either `GL_FLAT` or `GL_SMOOTH`

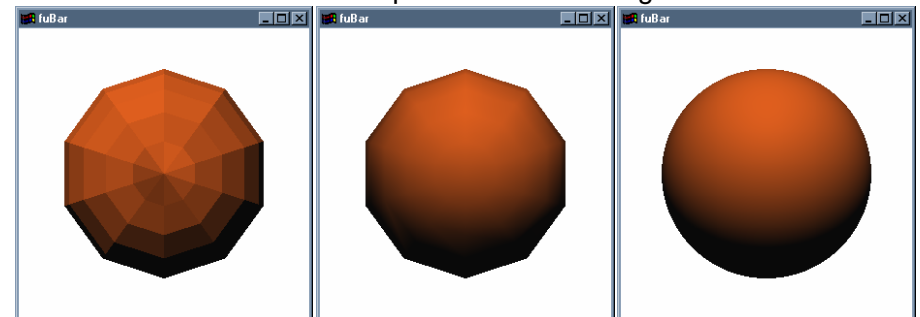


Better illumination can be achieved by increasing the object resolution

flat

interpolated

high resolution



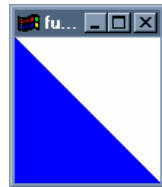
low resolution causes silhouette edges artifacts



Example:

```
glShadeModel(GL_SMOOTH);
glBegin(GL_TRIANGLES);
    glColor3f(1,0,0);glVertex3f(-1,-1,0);
    glColor3f(0,1,0);glVertex3f(1,-1,0);
    glColor3f(0,0,1);glVertex3f(-1,1,0);
glEnd();
```

last color is the entire object color in the GL_FLAT mode



(C) Bedrich Benes

5



Principle

- Each vertex is always illuminated by all lights
i.e., there are *no shadows*
- Color is summed and clamped to the maximal intensity
(1.f or 255 per channel)
- OpenGL uses a modified version of
Phong illumination model
- *Ad hoc* physically incorrect
- Fast and visually plausible results
- Lights are point lights
- Lights are invisible (!)

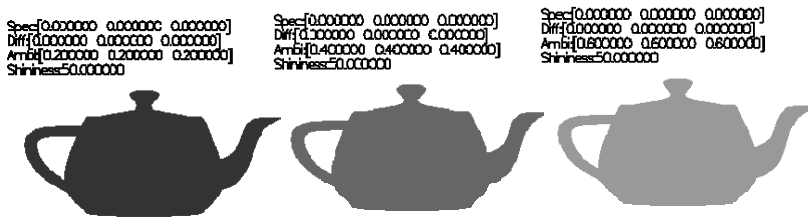
(C) Bedrich Benes

6



• Ambient light

is everywhere, does not have direction, does not change



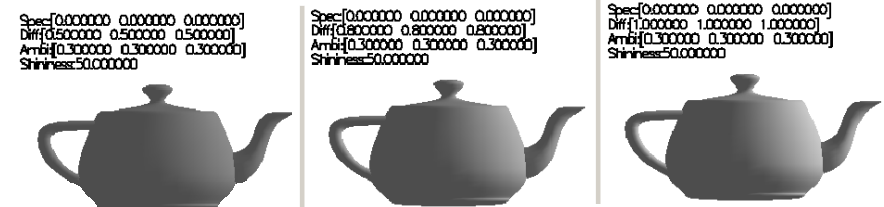
(C) Bedrich Benes

7



• Diffuse light

depends on the direction of the light and the normal vector
changes with the cosine of the angle



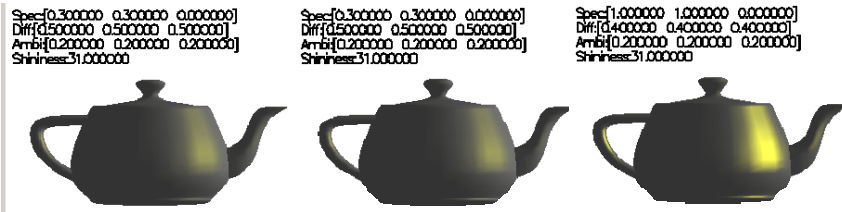
(C) Bedrich Benes

8



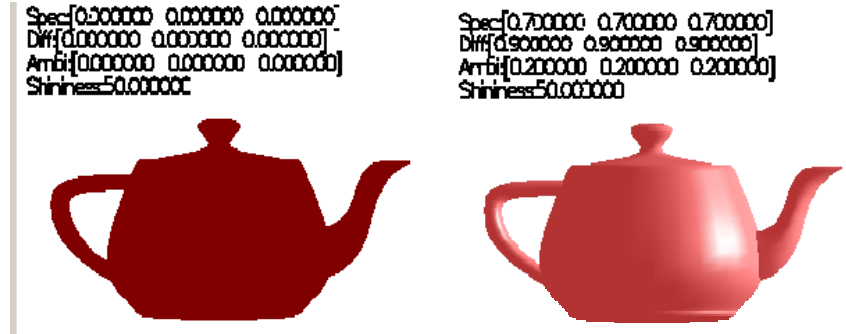
- *Specular light*

depends on the direction of the light, the normal vector, and the position of the viewer



- *Emmision light*

is self illumination of an object (fluorescence)
(used in games for arrows, switches, and so on)

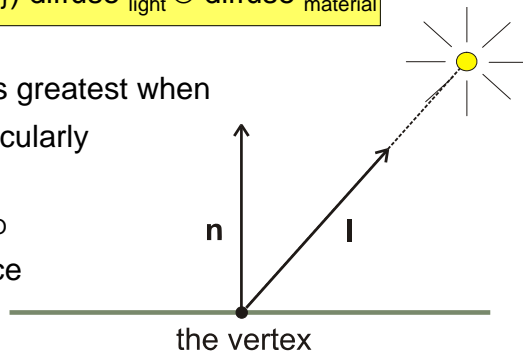


- *Equations*

$$\text{ambient} = \text{ambient}_{\text{light}} \otimes \text{ambient}_{\text{material}}$$

$$\text{diffuse} = (\max\{\mathbf{l} \cdot \mathbf{n}, 0\}) \text{diffuse}_{\text{light}} \otimes \text{diffuse}_{\text{material}}$$

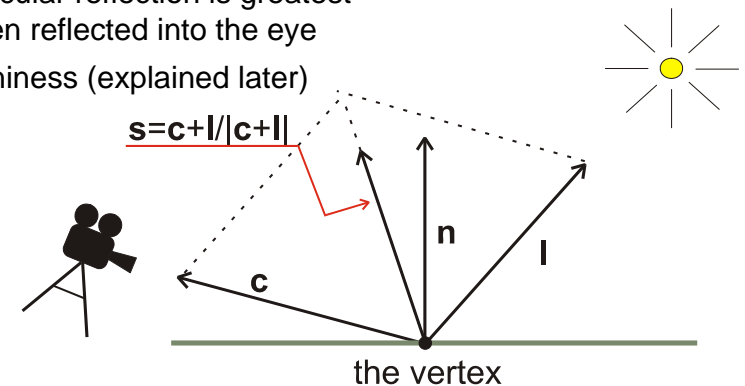
- diffuse reflection is greatest when light falls perpendicularly onto the surface
- smallest when 90° or down the surface



- *Equations*

$$\text{specular} = (\max\{\mathbf{s} \cdot \mathbf{n}, 0\})^{\text{shininess}} \text{specular}_{\text{light}} \otimes \text{specular}_{\text{material}}$$

- specular reflection is greatest when reflected into the eye
- shininess (explained later)





- Equations Putting this all together

Computed for red, green, and blue component separately
Emission, ambient, diffuse, and specular is summed.

Clamped to the range [0,1]

$color = emission + global\ ambient_{light} \otimes ambient_{material} + \sum light$

$light = attenuation * spot * (ambient + diffuse + specular)$

$attenuation = 1 / (const + d * linear + d^2 * quadratic)$



OpenGL supports at least 8 lights

`GL_LIGHT0, ..., GL_LIGHT7`

a light is enabled (turned on) by

`glEnable(GL_LIGHTn)`

and disabled by

`glDisable(GL_LIGHTn)`



Light properties are set by one procedure

`void glLight{if}[v](GLenum light, GLenum p, TYPE val)`

light is `GL_LIGHTn`
p is one of the possible parameters
val is its value (vector, scalar)



param	meaning	default
<code>GL_AMBIENT</code>	ambient light	(0,0,0,1)
<code>GL_DIFFUSE</code>	diffuse light	(1,1,1,1)
<code>GL_SPECULAR</code>	specular light	(1,1,1,1)
<code>GL_POSITION</code>	position of the light	[0,0,1,0]

note:

position [x,y,z,1] local light
position [x,y,z,0] infinite light, i.e., directional light

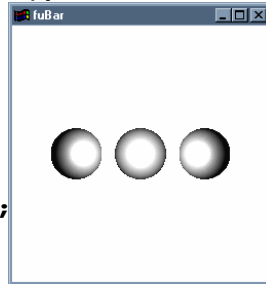


Example:

white spheres lit by different lights

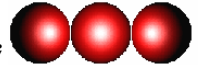
```
GLfloat light_position[]={0,0,-4,1.0}; //w=0:infinite
GLfloat light_color[]={1,1,1,1};
```

```
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_color);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glTranslatef(0,0,-10);
glutSolidSphere(0.2, 100, 100); //middle
glLoadIdentity(); glTranslatef(0.5,0,-5);
glutSolidSphere(0.2, 100, 100); //left
glLoadIdentity(); glTranslatef(-0.5,0,-5);
glutSolidSphere(0.2, 100, 100); //right
```

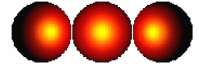


Example:

```
GLfloat specular[] = {1.f,1.f,1.f,1.f};
GLfloat diffuse[] = {1.f,0.f,0.f,1.f};
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
```



```
GLfloat specular[]={1.f,1.f,0.f,1.f};
GLfloat diffuse[]={1.f,0.f,0.f,1.f};
```



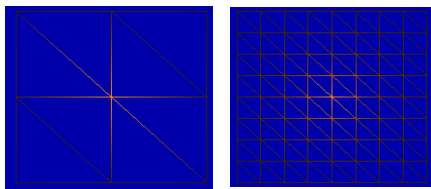
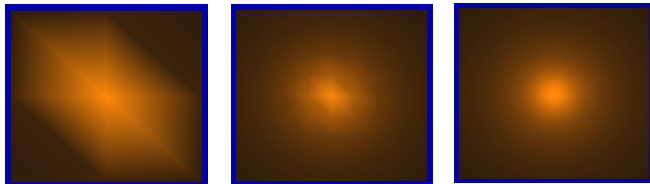
```
GLfloat specular[]={0.f,0.f,1.f,1.f};
GLfloat diffuse[]={0.5,0.5,0.5, 1.f};
```



```
GLfloat specular[] = {1.f,0.f,0.f,1.f};
GLfloat diffuse[] = {1.f,1.f,1.f,1.f};
```



The geometric level of detail is important for illumination

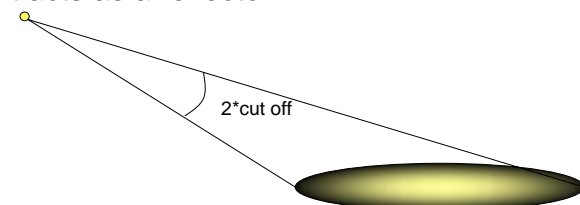


param	meaning	default
GL_SPOT_DIRECTION	direction	(0,0,-1)
GL_SPOT_CUTOFF	the cutoff	180.0

note:

GL_SPOT_CUTOFF set to 180° sets omnidirectional light

Spot light acts as a reflector

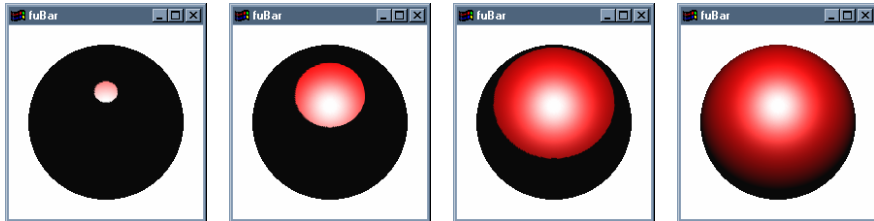




Example:

changing the GL_SPOT_CUTOFF

```
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF, 7.0);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_dir);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```



(C) Bedrich Benes

21

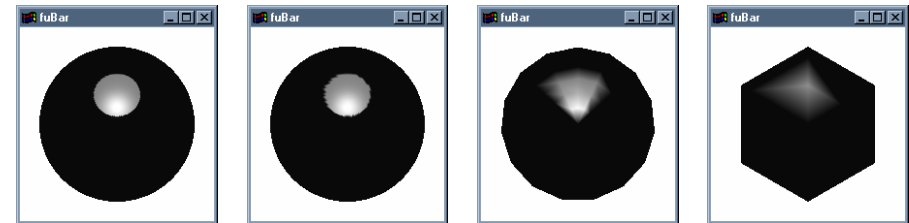


Problems

OpenGL calculates illumination per vertex

It is quite difficult to get precise boundary of the spot light

We need a high geometric precision



(C) Bedrich Benes

22



Problems

How can we calculate the GL_SPOT_DIRECTION?

Easily...

We know the position of the light

$P = [x_P, y_P, z_P]$

and the position (target) we want to illuminate

$T = [x_T, y_T, z_T]$

so it must be the vector

$D = T - P$

(C) Bedrich Benes

23



The light emitted by a non-parallel light source attenuates with the square of the distance

OpenGL provides the general equation

$$a = \frac{1}{C + dL + d^2Q}$$

where:

d is the distance between the vertex and the light source,

C is the constant attenuation,

L is the linear attenuation, and

Q is the quadratic attenuation

(C) Bedrich Benes

24



param	meaning	default
GL_CONSTANT_ATTENUATION	C	1
GL_LINEAR_ATTENUATION	L	0
GL_QUADRATIC_ATTENUATION	Q	0

note:

By default, there is NO attenuation



Global ambient

```
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,color);
```

where:

color is the color of the global ambient

Note:

All color contributions are summed and clamped to 1.f



Local vs. infinite viewpoint

viewpoint location affects specular highlights

if the viewer is infinite it is much more easier (faster)

```
glLightModelf(GL_LIGHT_MODEL_LOCAL_VIEWER,GL_TRUE)
```

calculates specular reflections as if the viewer is infinite

i.e., the viewing vector is *constant* for each vertex,

i.e. , it is faster....



Front vs. front and back faces illumination

by default only front faces are lit

```
glLightModelf(GL_LIGHT_MODEL_TWO_SIDE,GL_TRUE)
```

turns on both faces illumination calculation



```
void glMaterial{if}[v](GLenum face, GLenum p, TYPE val)
```

face is GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK

p is one of the possible parameters (see below)

val is its value

defines the portion of light reflected by the material

the light can be reflected in three typical ways

Ambient - to all sides equally

Diffuse - to all sides equally, but the intensity depends on the normal vector and light position

Specular - depends on the above and the viewer position



param	meaning	default
GL_AMBIENT	ambient reflection	(0.2,0.2,0.2,1)
GL_DIFFUSE	diffuse reflection	(0.8,0.8,0.8,1)
GL_AMBIENT_AND_DIFFUSE		
GL_SPECULAR	specular reflection	(0,0,0,1)
GL_SHININESS	exponent	0.0
GL_EMISSION	self illumination	(0,0,0,1)



Material	GL AMBIENT	GL DIFFUSE	GL SPECULAR	GL SHININESS
Brass	0.329412 0.223529 0.027451 1.0	0.780392 0.568627 0.113725 1.0	0.992157 0.941176 0.807843 1.0	27.8974
Bronze	0.2125 0.1275 0.054 1.0	0.714 0.4284 0.18144 1.0	0.393548 0.271906 0.166721 1.0	25.6
Polished Bronze	0.25 0.148 0.06475 1.0	0.4 0.2368 0.1036 1.0	0.774597 0.458561 0.200621 1.0	76.8
Chrome	0.25 0.25 0.25 1.0	0.4 0.4 0.4 1.0	0.774597 0.774597 0.774597 1.0	76.8
Copper	0.19125 0.0735 0.0225 1.0	0.7038 0.27048 0.0828 1.0	0.256777 0.137622 0.086014 1.0	12.8

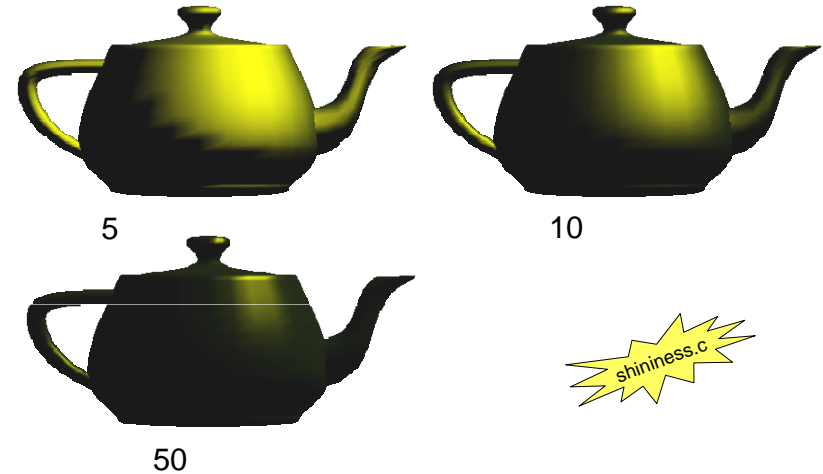




Material	GL AMBIENT	GL DIFFUSE	GL SPECULAR	GL SHININESS
Polished Copper	0.2295 0.08825 0.0275 1.0	0.5508 0.2118 0.066 1.0	0.580594 0.223257 0.0695701 1.0	51.2
Gold	0.24725 0.1995 0.0745 1.0	0.75164 0.60648 0.22648 1.0	0.628281 0.555802 0.366065 1.0	51.2
Polished Gold	0.24725 0.2245 0.0645 1.0	0.34615 0.3143 0.0903 1.0	0.797357 0.723991 0.208006 1.0	83.2
Pewter	0.105882 0.058824 0.113725 1.0	0.427451 0.470588 0.541176 1.0	0.333333 0.333333 0.521569 1.0	9.84615

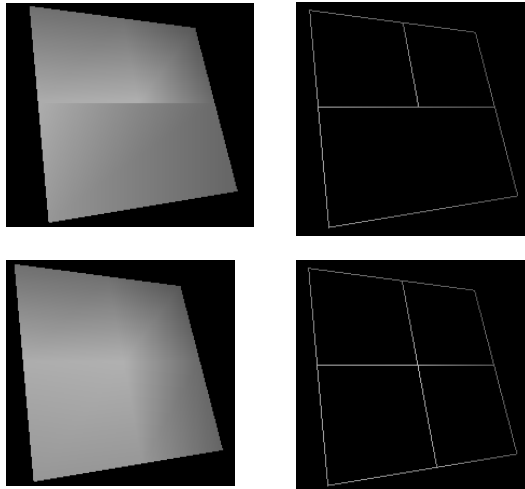


The shininess coefficient



The T-vertex and lighting

T-vertex causes
illumination artifacts



The normal vector is *critical*
it is defined for each vertex by the call of
`glNormal*()`
so the classical per-vertex definition is

```
glNormal*()
glMaterial*()
glVertex*()
```



The normal vector must be normalized
This can be done by OpenGL automatically by

```
glEnable(GL_AUTO_NORMAL)
```

or by yourself...

```
void Normalize(GLdouble *v){
    static GLdouble size;
    size = v[0]*v[0]+v[1]*v[1]+v[2]*v[2];
    size = sqrt(size);
    v[0]/=size;
    v[1]/=size;
    v[2]/=size;
}
```



How the normal vector can be obtained?

1) Analytically

if we know it from the surface definition

e.g., Sphere - each vertex v has normal vector $n=v-o$

where o is the origin

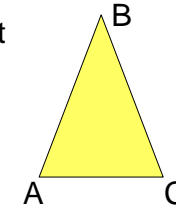
2) By local differences

and the vector product

$v_1=A-B$

$v_2=A-C$

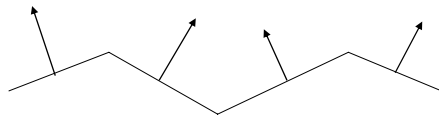
$n=v_1 \times v_2$



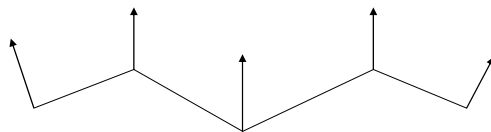
How the normal vector can be obtained?

In many cases it is good to average the neighboring normals

a) normals per face



b) normals per vertex (average of the case a))



Example:

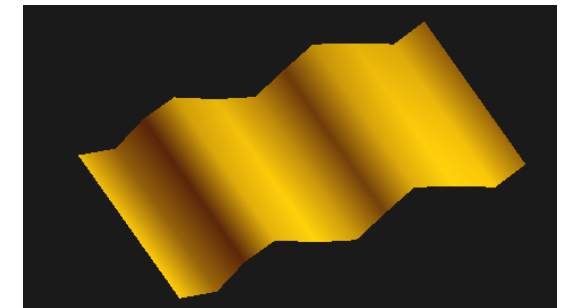
analytically obtained normals

silhouette edges are

really rough,

but the inner part

looks alright





Call of `glMaterial()` is expensive
in a case of extensive changes of material properties use

```
void glColorMaterial(GLenum face, GLenum mode)
```

where

`face` is `GL_FRONT`, `GL_BACK`, or `GL_FRONT_AND_BACK`

`mode` is `GL_AMBIENT`, `GL_DIFFUSE` etc.

use if one property varies for many vertices
enabled it `glEnable(GL_COLOR_MATERIAL);`
and use `glColor*`



Example:

```
glColorMaterial(GL_FRONT, GL_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);
glColor3fv(color1);
Render1();
glColor3fv(color2);
Render2();
glColor3fv(color3);
Render3();
glDisable(GL_COLOR_MATERIAL);
```



- Jackie Neider, Tom Davis, Mason Woo
OpenGL Programming Guide,
Addison-Wesley Publication Company
ON LINE at <http://www.opengl.org.ru/docs/>
- www.opengl.org/developers/code/tutorials.html
- SIGGRAPH 2001
An Interactive Introduction To OpenGL Programming
www.opengl.org/developers/code/s2001/index.html
- SIGGRAPH '99
Lighting and Shading Techniques for Interactive Applications
www.opengl.org/developers/code/sig99/index.html