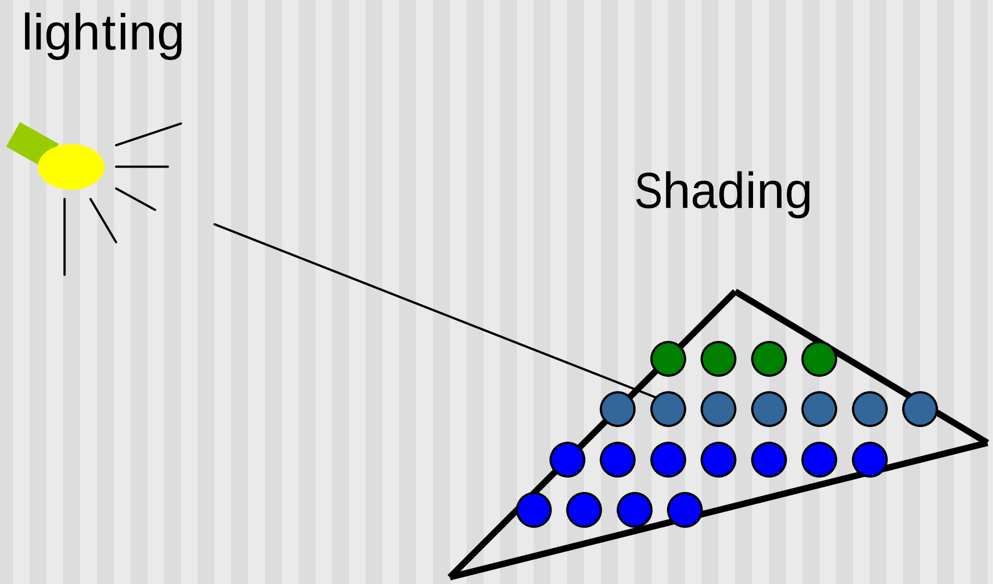
CS 4731/543: Computer Graphics
Lecture 4 (part I): Illumination and Shading

Emmanuel Agu

Illumination and Shading

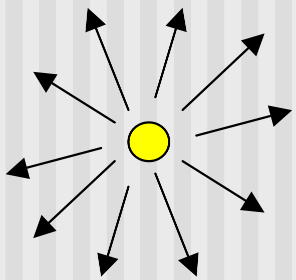
- Problem: Model light/surface points interaction to determine final color and brightness
- Apply the lighting model at a set of points across the entire surface



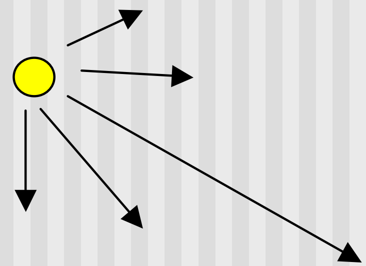
Illumination Model

- The governing principles for computing the illumination
- A illumination model usually considers:
  - Light attributes (intensity, color, position, direction, shape)
  - Object surface attributes (color, reflectivity, transparency, etc)
  - Interaction among lights and objects

Basic Light Sources

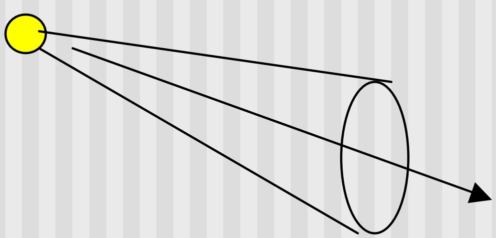


Point light

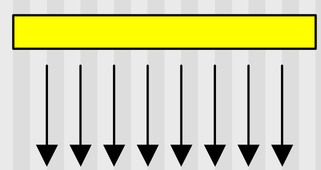


Directional light

Light intensity can be independent or dependent of the distance between object and the light source



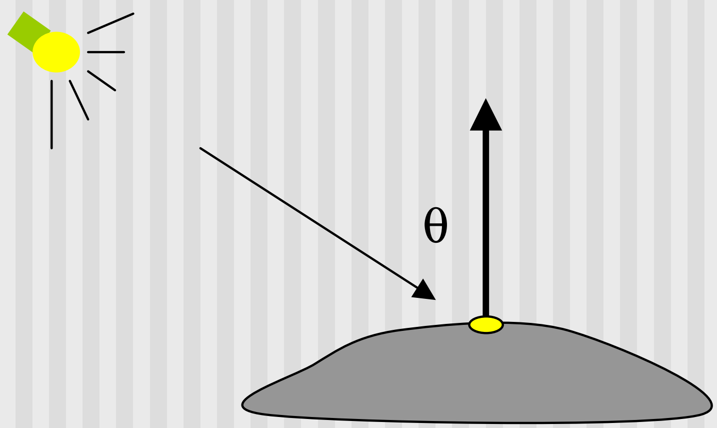
Spot light



Area light

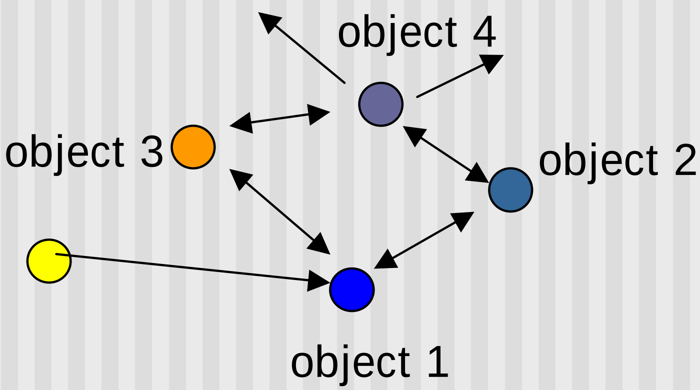
Local Illumination

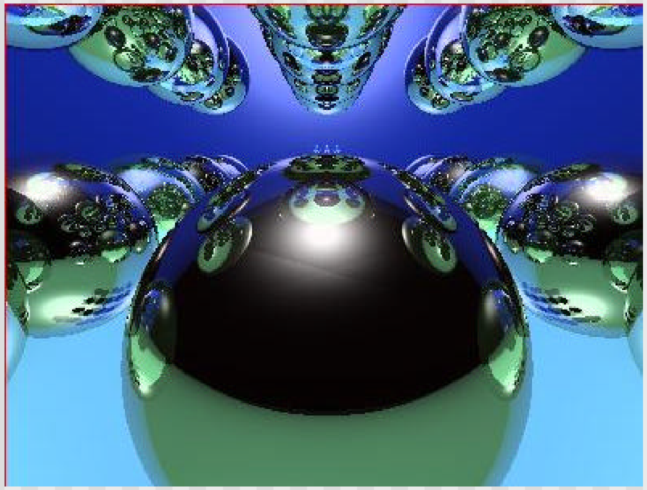
- Local illumination: only consider the light, the observer position, and the object material properties
- OpenGL does this



Global Illumination

- Global illumination: take into account the interaction of light from all the surfaces in the scene
- Example: Ray tracing





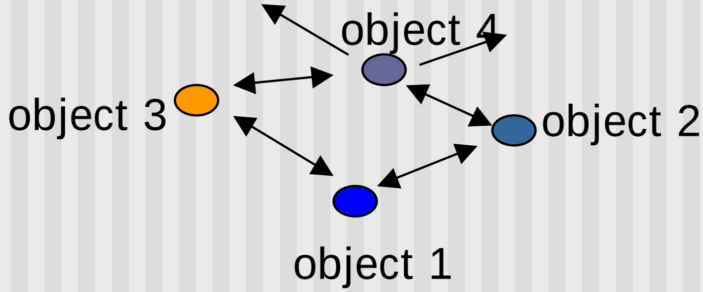
Simple Local Illumination

- The model used by OpenGL
- Consider three types of light contribution to compute the final illumination of an object
  - Ambient
  - Diffuse
  - Specular
- Final illumination of a point (vertex) =

  ambient + diffuse + specular
- Materials reflect each component differently
  - Use different material reflection coefficients, Ka, Kd, Ks

Ambient Light Contribution

- Ambient light = background light
- Light that is scattered by the environment
- **Frequently assumed to be constant**
- Very simple approximation of global illumination
- No direction: independent of light position, object orientation, observer's position or orientation



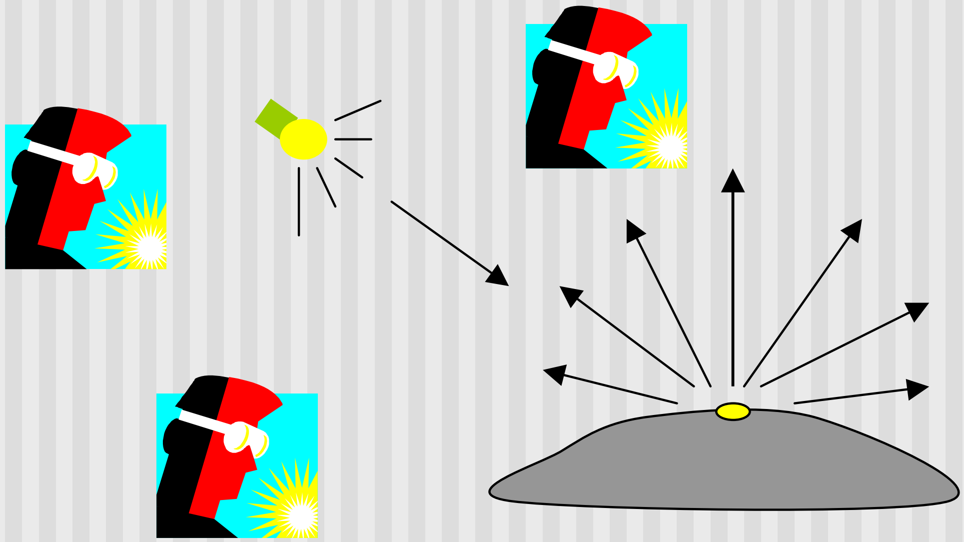
constant

Ambient = I x Ka

Ambient Light Example

Diffuse Light Contribution

- Diffuse light: The illumination that a surface receives from a light source and reflects equally in all direction

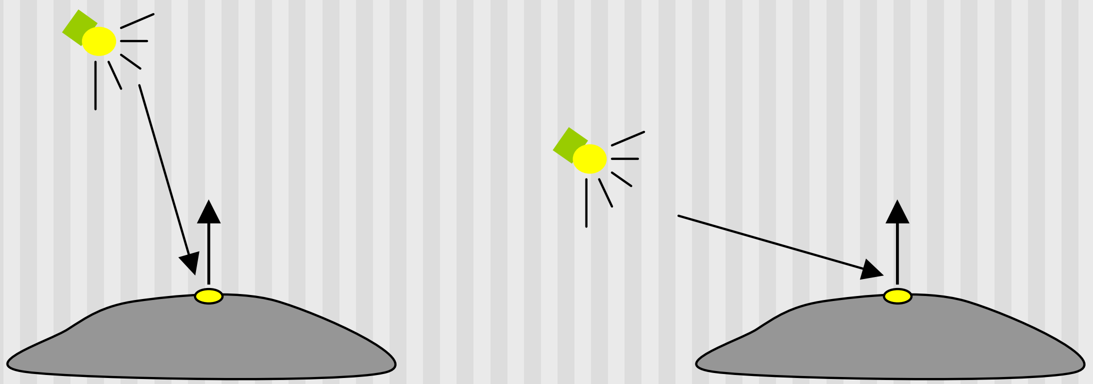


It does not matter where the eye is

Diffuse Lighting Example

Diffuse Light Calculation

- Need to decide how much light the object point receive from the light source – based on Lambert's Law



Receive more light

Receive less light

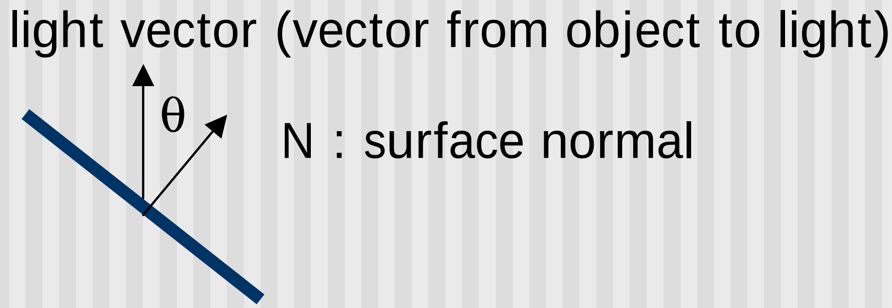
Diffuse Light Calculation

- Lambert's law: the radiant energy D that a small surface patch receives from a light source is:

$$D = I \times \cos(\theta)$$

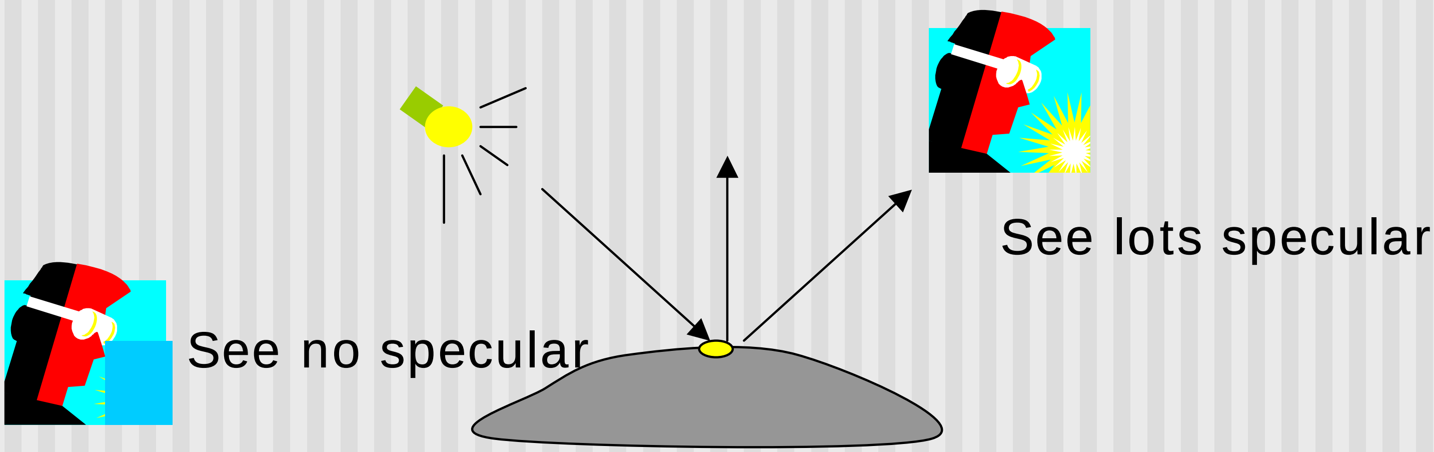
I: light intensity

θ: angle between the light vector and the surface normal



Specular light contribution

- The bright spot on the object
- The result of total reflection of the incident light in a concentrate region



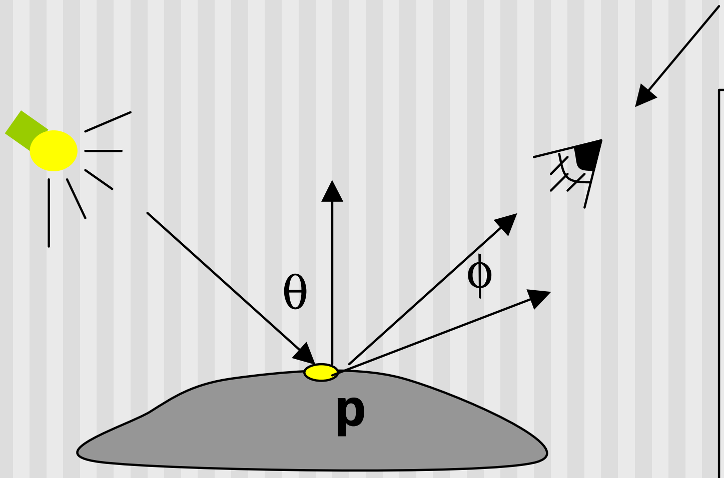
Specular light example

Specular light calculation

- How much reflection you can see depends on where you are

specular = Ks x I x $\cos^{n}(\phi)$

Only position the eye can see specular from P if object has an ideal reflection surface



But for non-perfect surface you will still see specular highlight when you move a little bit away from the ideal reflection direction

Φ is deviation of view angle from mirror direction

When ϕ is small, you see more specular highlight

Specular light calculation

- Phong lighting model

$$\text{specular} = K_s \times I \times \cos^{n}(\phi)$$

- The effect of 'n' in the phong model

n = 10

n = 90

n = 30

n = 270

Put it all together

- Illumination from a light:

  Illum = ambient + diffuse + specular

  $$= Ka \times I + Kd \times I \times (\cos\theta) + Ks \times I \times \cos^n(\phi)$$

- If there are N lights

  Total illumination for a point P = Σ (Illum)

- Some more terms to be added (in OpenGL):
  - Self emission
  - Global ambient
  - Light distance attenuation and spot light effect

Adding Color

- Sometimes light or surfaces are colored
- Treat R,G and B components separately
- i.e. can specify different RGB values for either light or material
- Illumination equation goes from:

Illum = ambient + diffuse + specular

$$= Ka \times I + Kd \times I \times (\cos\theta) + Ks \times I \times \cos^n(\phi)$$

To:

$$Illum\_r = Kar \times Ir + Kdr \times Ir \times (\cos\theta) + Ksr \times Ir \times \cos^n(\phi)$$

$$Illum\_g = Kag \times Ig + Kdg \times Ig \times (\cos\theta) + Ksg \times Ig \times \cos^n(\phi)$$

$$Illum\_b = Kab \times Ib + Kdb \times Ib \times (\cos\theta) + Ksb \times Ib \times \cos^n(\phi)$$

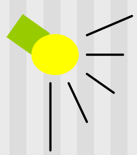
Adding Color

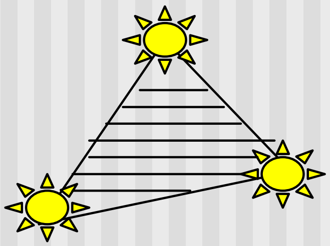
Material	Ambient Kar, Kag,kab	Diffuse Kdr, Kdg,kdb	Specular Ksr, Ksg,ksb	Exponent, n
Black plastic	0.0 0.0 0.0	0.01 0.01 0.01	0.5 0.5 0.5	32
Brass	0.329412 0.223529 0.027451	0.780392 0.568627 0.113725	0.992157 0.941176 0.807843	27.8974
Polished Silver	0.23125 0.23125 0.23125	0.2775 0.2775 0.2775	0.773911 0.773911 0.773911	89.6

Figure 8.17, Hill, courtesy of McReynolds and Blythe

Lighting in OpenGL

- Adopt Phong lighting model
  - specular + diffuse + ambient lights
  - Lighting is computed at vertices
    - Interpolate across surface (Gouraud/smooth shading)
- Setting up OpenGL Lighting:
  - Light Properties
  - Enable/Disable lighting
  - Surface material properties
  - Provide correct surface normals
  - Light model properties

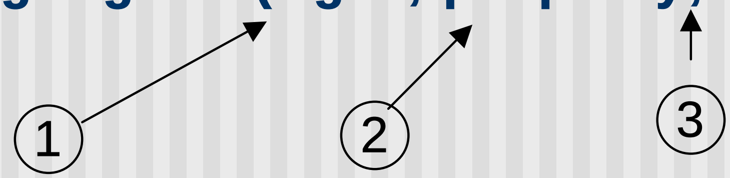




Light Properties

- Properties:
  - Colors / Position and type / attenuation

glLightfv(light, property, value)



(1) constant: specify which light you want to set the property
E.g: GL_LIGHT0, GL_LIGHT1, GL_LIGHT2 ... you can create multiple lights (OpenGL allows at least 8 lights)

(2) constant: specify which light property you want to set the value
E.g: GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION
(check the red book for more)

(3) The value you want to set to the property

Property Example

- Define colors and position a light

```
GLfloat light_ambient[] = {0.0, 0.0, 0.0, 1.0};
GLfloat  light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_position[] = {0.0, 0.0, 1.0, 1.0};

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);
```

colors

Position

What if I set Position to (0,0,1,0)?

Types of lights

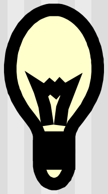
- OpenGL supports two types of lights
  - Local light (point light)
  - Infinite light (directional light)
- Determined by the light positions you provide
  - w = 0: infinite light source
  - w != 0: point light – position = (x/w, y/w, z/w)

```
GLfloat light_position[] = {x,y,z,w};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Turning on the lights

- Turn on the power (for all the lights)
  - glEnable(GL_LIGHTING);
  - glDisable(GL_LIGHTING);

- Flip each light's switch
  - glEnable(GL_LIGHT*n*) (n = 0,1,2,...)



Controlling light position

- Modelview matrix affects a light's position
- Two options:
- Option a:
  - Treat light like vertex
  - Do pushMatrix, translate, rotate, .. **glLightfv position**, popmatrix
  - Then call gluLookat
  - Light moves independently of camera
- Option b:
  - Load identity matrix in modelview matrix
  - Call glLightfv then call gluLookat
  - Light appears at the eye (like a miner's lamp)

Material Properties

- The color and surface properties of a material (dull, shiny, etc)
- How much the surface reflects the incident lights (ambient/diffuse/specular reflection coefficients)

  glMaterialfv(face, property, value)

Face: material property for which face (e.g. GL_FRONT, GL_BACK, GL_FRONT_AND_BACK)
Property: what material property you want to set (e.g. GL_AMBIENT, GL_DIFFUSE,GL_SPECULAR, GL_SHININESS, GL_EMISSION, etc)
Value: the value you can to assign to the property

Material Example

- Define ambient/diffuse/specular reflection and shininess

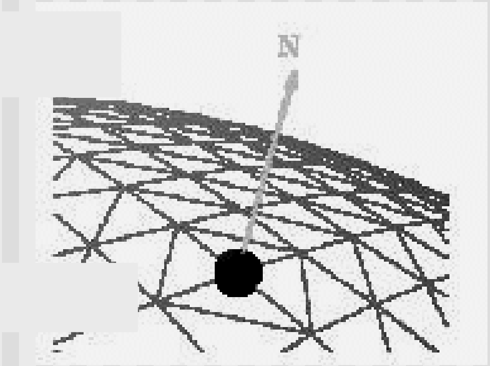
```
GLfloat mat_amb_diff[] = {1.0, 0.5, 0.8, 1.0};   ┐
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};   ┘ ← refl. coeff.
GLfloat shininess[] = {5.0}; ←   (range: dull 0 – very shiny 128)

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE,
                        mat_amb_diff);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, shininess);
```

Surface Normals

- Correct normals are essential for correct lighting
- Associate a normal to each vertex

```
glBegin(…)
    glNormal3f(x,y,z)
    glVertex3f(x,y,z)
    …
  glEnd()
```



- The normals you provide need to have a unit length
  - You can use **glEnable(GL_NORMALIZE)** to have OpenGL normalize all the normals

What about SDL?

- **Assignment:** read how to do following in SDL
  - control light sources
  - Specify material properties
  - Ambient, diffuse specular, etc
- Ref: section 5.6.4, appendix 5

References

- Hill, chapter 8