# Lighting and Rasterization - Shading

# Intended Learning Outcomes

- Classify different types of light sources
- Understand the image formation process
- Mathematically model three types of reflection and understand the
- Understand thr nd compare their pros and cons
- Able to program lighting and ng OpenGL

# Lighting and Shading Models

- Calculate intensity and colour of light that we should see at a given point of a scene
- Ultimate aim : *Photorealism*



- Lighting /Illumi



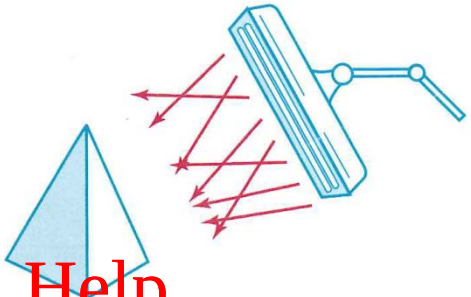
  - models lighting from light s the environment



- Shading models
  - models how lights are processed (reflected, absorbed, refracted etc) by the objects and the atmosphere

# Light sources

- Ambient source
  - models background light
- Point source
  - for small nearby light sources
- Distributed sou
  - for large nea
  - models by a collection of p s
- Lighting direction
  - (e.g. sun) - for distant light sources



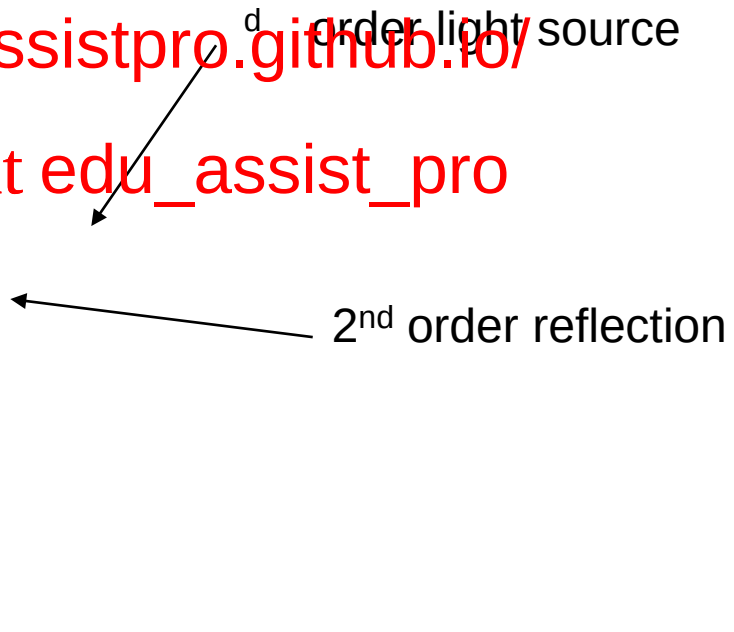


Point Sour

Distributed Source

- Realistic lighting is higher order and complicated

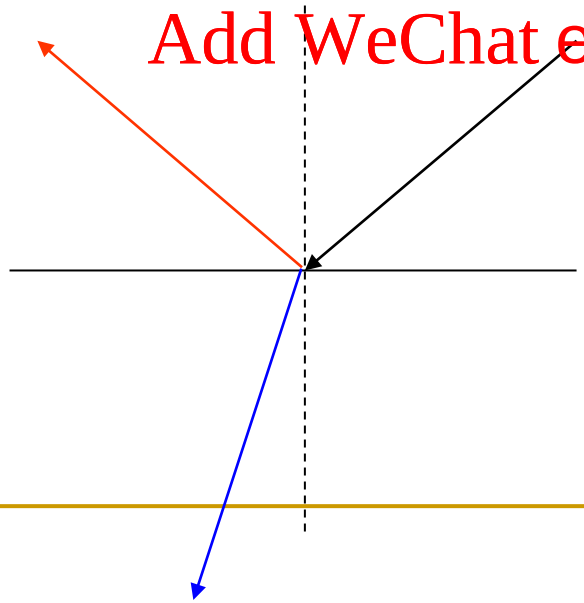




# Shading

- When light is incident on an object
  - part is reflected
  - part is absorbed
  - part is refrac





# Object properties

- *Opaque* object only reflect and absorb light
- *Transparent* object only refract and absorb light
- *Semi-transparent* object reflect, refract and absorb light



- The amount of [illegible] on material.





  - Shiny material : reflect mos t
  - Dull material : absorb most of the light

- Let restrict discussion to opaque object at present

# Types of Reflection

- Ambient reflection
  - Average signal from the background
  - Non-directional
- Diffuse reflect
  - Rough, dull, matte surfaces
  - scatter light equally in all dire
- Specular reflection
  - Smooth, shiny, mirror like surfaces
  - reflect light more in one direction

# Ambient reflection

$$I_{ambdiff} = k_a I_a$$



ka ambient r $k_a \leq 1$

$I_a$ incident a





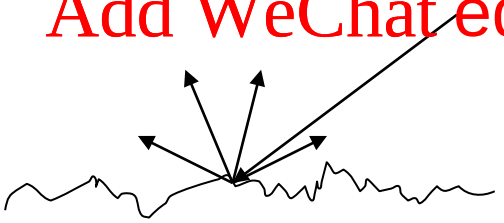
- Can be interpreted as the average value of diffuse reflection from numerous light sources in the background

# Diffuse Reflection

- Consider a point light source or lighting direction

- *Lambertian surfaces* : Reflections from the surface are scattered with equal intensity in all directions, independent of

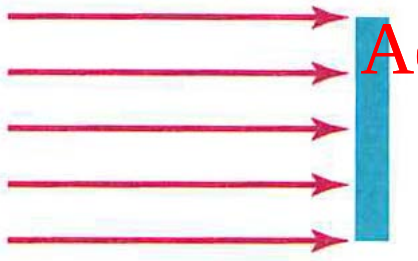




Diffuse (Lambertian)
Surface (Rough, dull
e.g. wood)

- Amount of incident light received by the surface is proportional to the projected area of the surface in the lighting direction





$$I_{l,diff} = k_d I_l (\mathbf{N} \cdot \mathbf{L})$$

$k_d$ diffuse reflection coefficient, $0 \le k_d \le 1$

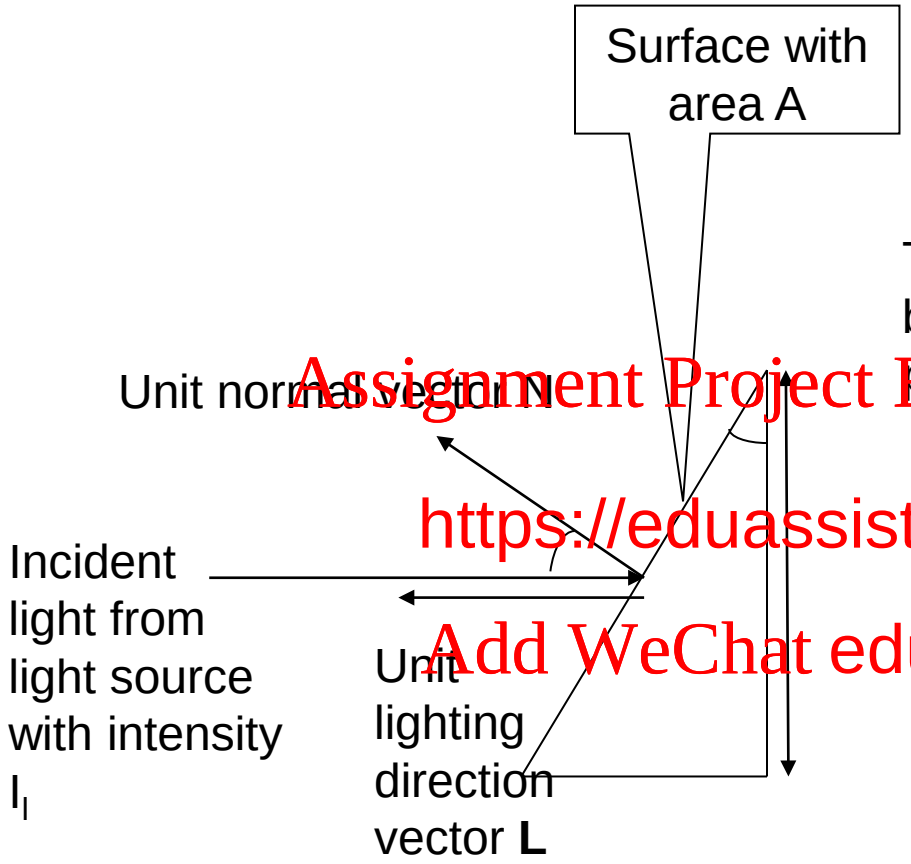
$I_l$ Incident lig

**N** unit normal

**L** unit light direction vector



- **N**·**L** models the projected area

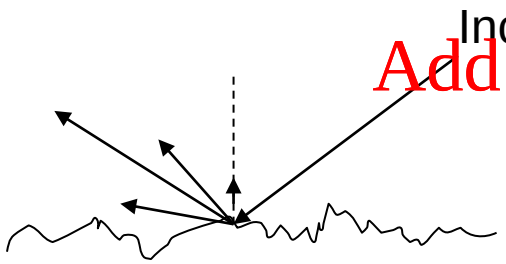



The total amount of light received by the surface with area A is proportional to $A\cos\theta = A$ (**N**•**L**)

# Specular reflection

- Consider a point light source or lighting direction.
- Ideal specular surface = perfect mirror: light is only reflected in the direction of R
- Non-ideal reflector: some light are scattered around R






Specular
Surface (Shiny e.g. mirror, gold silver, glass)

L Incident light direction

$$I_{l,spec} = W(\theta) I_l \cos^{n_s} \phi$$

W(θ) specular reflection coefficent, $0 \le W(\theta) \le 1$
sometimes W(θ) is assumed to be a constant $k_s$

**N** bisects **L** and **R** (incident angle = reflection angle in a perfect mirr

**R** unit specul r



**R** = (2**N**·**L**)**N** – **L**

**V** unit viewing direction vector

$\cos(\phi) = \mathbf{R} \cdot \mathbf{V}$ $\quad 0 \le \phi \le \pi/2$

$n_s$ specular reflection exponent, $n_s = \infty$ for perfect mirror

$$\mathbf{R} = (2\mathbf{N}\cdot\mathbf{L})\mathbf{N} - \mathbf{L}$$

# General Model with n light sources with ambient, diffuse and specular terms

$$I = k_a I_a + \sum_{i=1}^{n} [\ldots (\theta_i)(\mathbf{V} \cdot \mathbf{R}_i)^{n_s}]$$

# Colour model

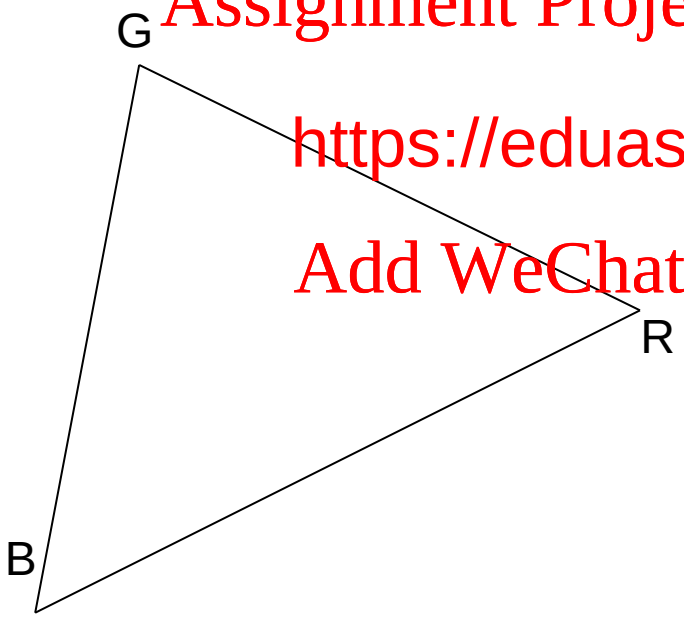
- Each light source is a vector with Red, Green, Blue component ($I_{lR}$, $I_{lG}$, $I_{lB}$)
- Calculates each component separately



$$I_R = k_{aR} I_{aR} + \sum_{i=1}^{n} I_{lRi} \; k_{dR} \qquad )^{n_{sR}} ]$$

$$I_G = k_{aG} I_{aG} + \sum_{i=1}^{n} I_{lGi} [k_{dG} (\mathbf{N} \cdot \mathbf{L}_i) + W_G(\theta_i)(\mathbf{V} \cdot \mathbf{R}_i)^{n_{sG}}]$$

$$I_B = k_{aB} I_{aB} + \sum_{i=1}^{n} I_{lBi} [k_{dB} (\mathbf{N} \cdot \mathbf{L}_i) + W_B(\theta_i)(\mathbf{V} \cdot \mathbf{R}_i)^{n_{sB}}]$$




Note:

Only colours in the triangle is displayable.

Some naturally occurring colours outside the triangle cannot be displayed!

Quattron technology uses rimary lours RYGB that extends the displayble colours

CIE chromaticity diagram
-Represent all possible colours seeable by humans

LG-32UD59-B

# Shading Models /Rendering Models

- Input : Object tessellated into polygons (standard graphics object)

- Three common ways to shade the polygons:



  - Flat Shadin
  - Gouraud Shading
  - Phong Shading

ng realism

computational cost

# Flat shading

- A single intensity is calculated for the polygon. All points of the polygon are then displayed with the same intensity value
- Fast (Adv.)
- Faceted look -
- Human vision is subject to "M ffect" – intensity discontinuities are accentuate plifies the edges of the polygons, which is undesirable

# Gouraud shading

- Linearly interpolate **intensity values** across each polygon
- Intensities for each polygon are matched with the values of adjacent polygons along the common edges
- Interpolation el scontinuities that occur in flat sh
- Slower (disadv.)
- Smooth out specular highlights (disadv.)

- Step 1 : Determine the average unit normal vector at each polygon vertex

$$\mathbf{N}_v = \frac{\sum_{k=1}^{n} \mathbf{N}_k}{\left| \sum_{k=1}^{n} \mathbf{N}_k \right|}$$

(each $\mathbf{N}_k$ is a unit vector, $\mathbf{N}_v$ is a unit vector by def.)



- Step 2 : Apply each vertex to calculate the v



- Step 3 : linearly interpolate th ensities over the surface of the polygon

# Linear Interpolation

- Points lying on an edge of the polygon : linearly interpolate between two endpoints

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$



- interior points interpolate across the scan line





$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

# Phong shading

- Similar to Gouraud shading, but interpolates **normal vectors** instead.
- Captures specular highlights
- Highest realis
- Slowest (disad

- Step 1 : determine the average unit normal vector at each polygon vertex

$$\mathbf{N} = \frac{y - y_2}{y_1 - y_2}\mathbf{N}_1 + \frac{y_1 - y}{y_1 - y_2}\mathbf{N}_2$$



- Step 2 : linearly interpolate the vertex normals over the surface of the polygon
- Step 3 : apply an illumination model to calculate pixel intensities of each surface point

# Incremental form

- Linear interpolation equation is expressed in incremental form to save computation:

$$I(y) = I_1 + \frac{I_2 - I_1}{y_1 -}$$



one scan line down

$$I(y-1) = I(y) + \frac{I_2 - I_1}{y_1 - y_2}$$

# OpenGL Functions : Lighting

```
glEnable (GL_LIGHTING);  // activate lighting routines

glLight* (lightName, lightProperty, propertyValue);

GLfloat  light1PosType [] = {2.0, 0.0, 3.0, 1.0};    // point
                                                     st entry is 1.0
GLfloat  light2Po                                    0.0};   // light
                    // direc                            t entry is 0.0

glLightfv (GL_LIGHT1, GL_POSITION, light1PosType);  // v
    for vector
glEnable (GL_LIGHT1);

glLightfv (GL_LIGHT2, GL_POSITION, light2PosType);
glEnable (GL_LIGHT2);
```

# Light source colour

- (R, G, B, A)  A stands for alpha value



```
GLfloat blackCol                                    };
GLfloat whiteCol                                    };
```





```
glLightfv (GL_LIGHT3, GL_AMB                    kColor);
glLightfv (GL_LIGHT3, GL_DIFFUSE, whiteColor);
glLightfv (GL_LIGHT3, GL_SPECULAR, whiteColor);
```

# Surface Property

*glMaterial* (surfFace, surfProperty, propertyValue);*

*diffuseCoeff [ ] = {0.2, 0.4, 0.9, 1.0}; // kdR = 0.2, kdG = 0.4, kdB = 0.9*

*specularCoeff [ ] = {1 1.0, …*

*glMaterialfv (GL_FRONT_AND_BACK T_AND_DIFFUSE, diffuseCoeff );*

*glMaterialfv (GL_FRONT_AND_BACK, GL_SPECULAR, specularCoeff);*

*glMaterialf (GL_FRONT_AND_BACK, GL_SHININESS, 25.0 ); // $n_s$ = 25*

# Surface Rendering

- FLAT and Gouraud Shading

  glShadeModel (surfRenderingMethod);



  surfRendering Flat shading



  Gouraud



- Calculating normals

  *glNormal3* (Nx, Ny, Nz);*

# Gouraud shade a triangle

```
glEnable (GL_NORMALIZE); // convert all normal vectors to unit vector
glLightModeli (GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
                    // set correct V for specular calculations

glBegin (GL_TRIANGLES);
    glNormal3fv (n                                  tor at vertex1 calculated
                                                    e unit normal vector
    glVertex3fv  (vertex1);
    glNormal3fv (normalVector2);
    glVertex3fv  (vertex2);
    glNormal3fv (normalVector3);
    glVertex3fv  (vertex3);
glEnd ( );
```

# References

- Text: Ch. 17.1-17.3 for lighting and shading equations
- Text: Ch. 19.3 – 19.4 for CIE chromaticity diagram and RGB model
- Text: Ch. 17.1 ethod
- Text: Ch. 17.1 ds
- Demo: Run lightposition.exe tetrad.exe in TUTORS program
- Quattron technology: http://en.wikipedia.org/wiki/Quattron