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 their properties
- Understand three rendering methods and compare their pros and cons
- Able to program lighting and shading using OpenGL

Lighting and Shading Models

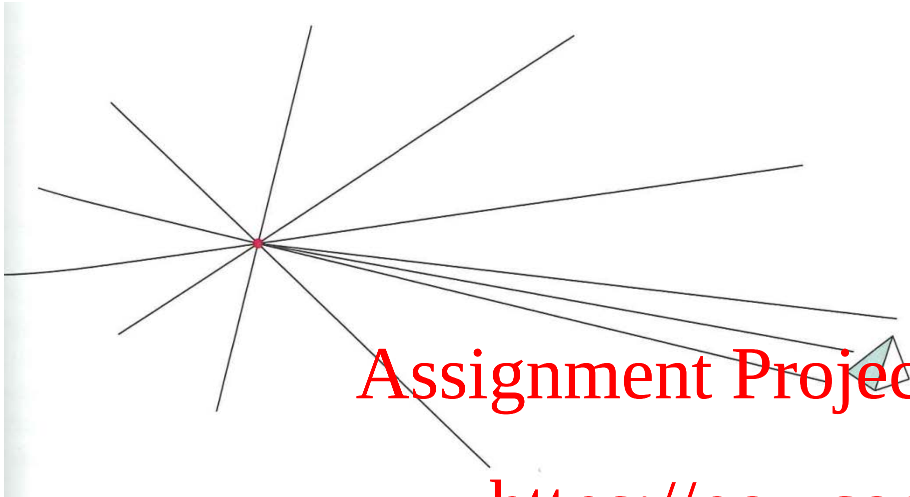
- Calculate intensity and colour of light that we should see at a given point of a scene
- Ultimate aim : *Photorealism*

- Lighting /Illumination models
  - models lighting from light sources and the environment

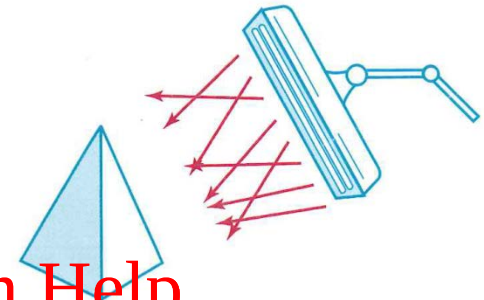
- Shading models
  - models how lights are processed (reflected, absorbed, refracted etc) by the objects and the atmosphere

Light sources

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

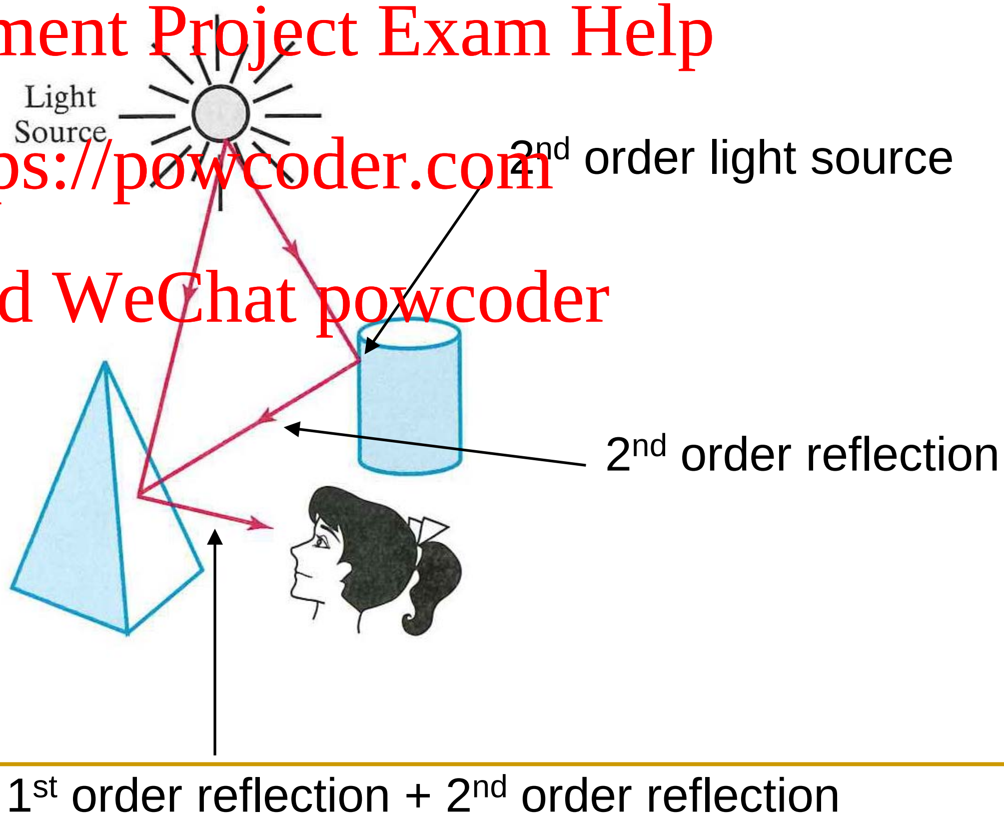


Point Source



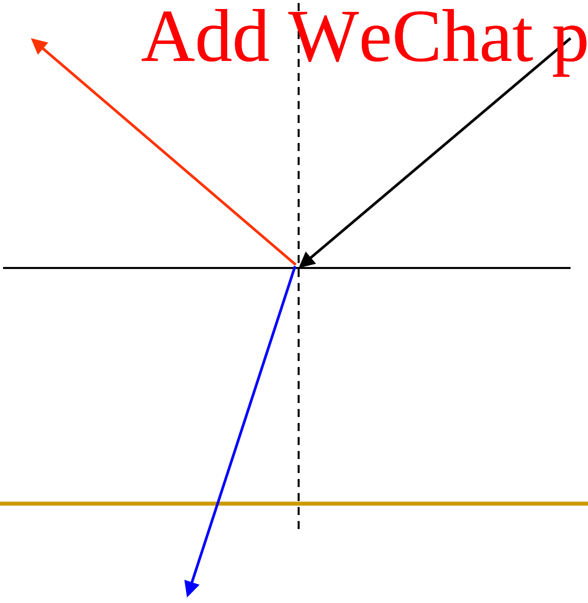
Distributed Source

- Realistic lighting is higher order and complicated



Shading

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



Object properties

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

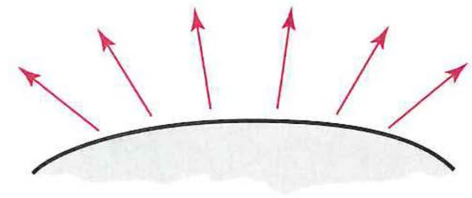
- The amount of light reflected depends on material.

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

- Let restrict discussion to opaque object at present

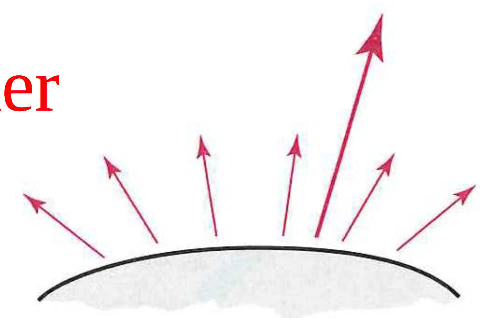
Types of Reflection

- Ambient reflection
  - Average signal from the background
  - Non-directional



Diffuse reflections from a surface.

- Diffuse reflection
  - Rough, dull, matte surfaces
  - scatter light equally in all directions



Specular reflection superimposed on diffuse reflection vectors.

- Specular reflection
  - Smooth, shiny, mirror like surfaces
  - reflect light more in one direction

Ambient reflection

$$I_{ambdiff} = k_a I_a$$

ka ambient reflection coefficent, $0 \le k_a \le 1$

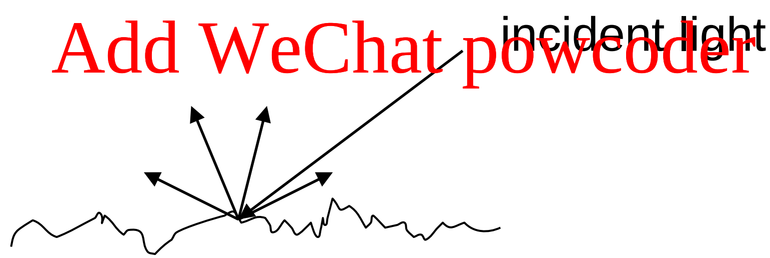
I_a incident ambient light

- 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 the viewing direction



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

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

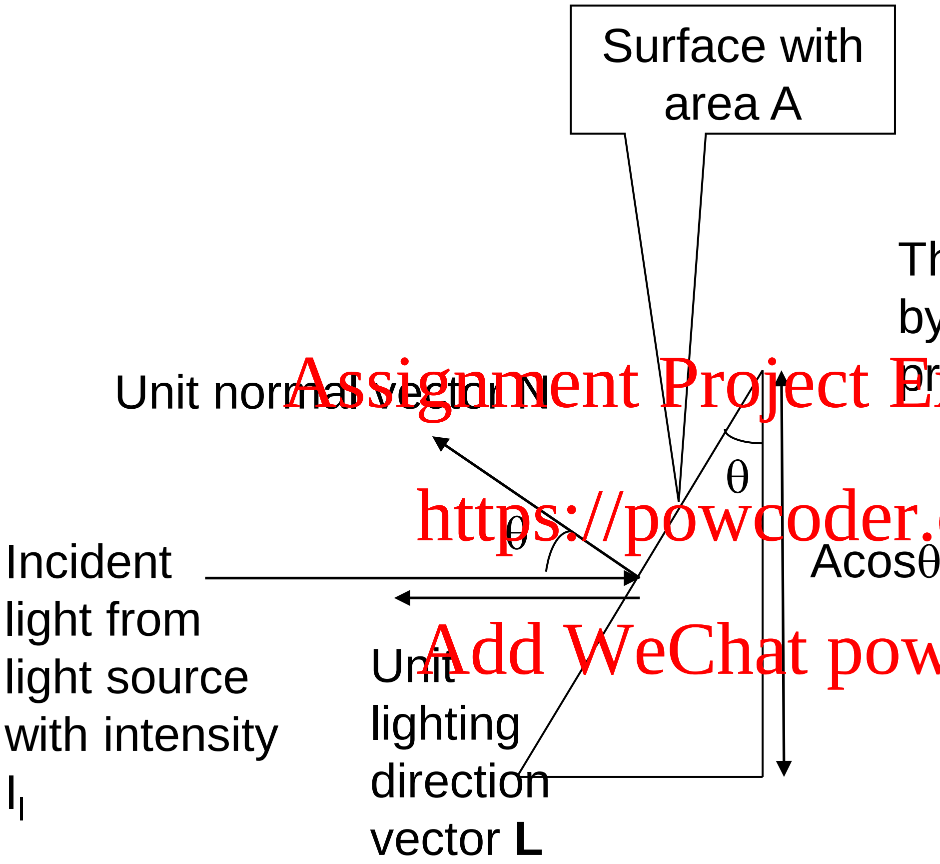
k_d diffuse reflection coefficient, $0 \le k_d \le 1$

I_l Incident light intensity

N unit normal of the surface

L unit light direction vector

- **N·L** models the projected area

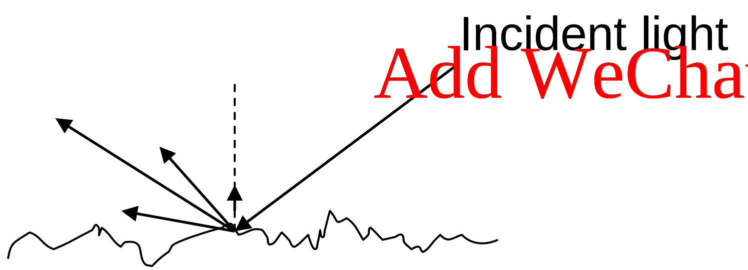


The total amount of light received by the surface with area A is proportional to Acosθ = A (**N•L**)

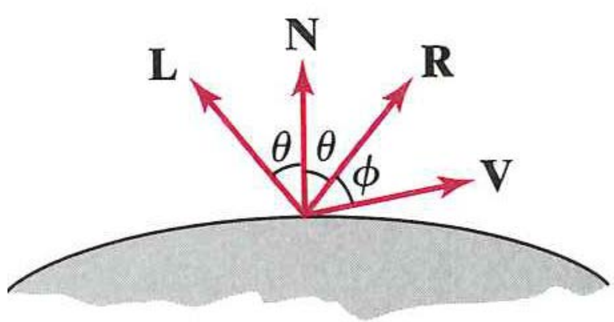
Specular reflection

- 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 mirror)

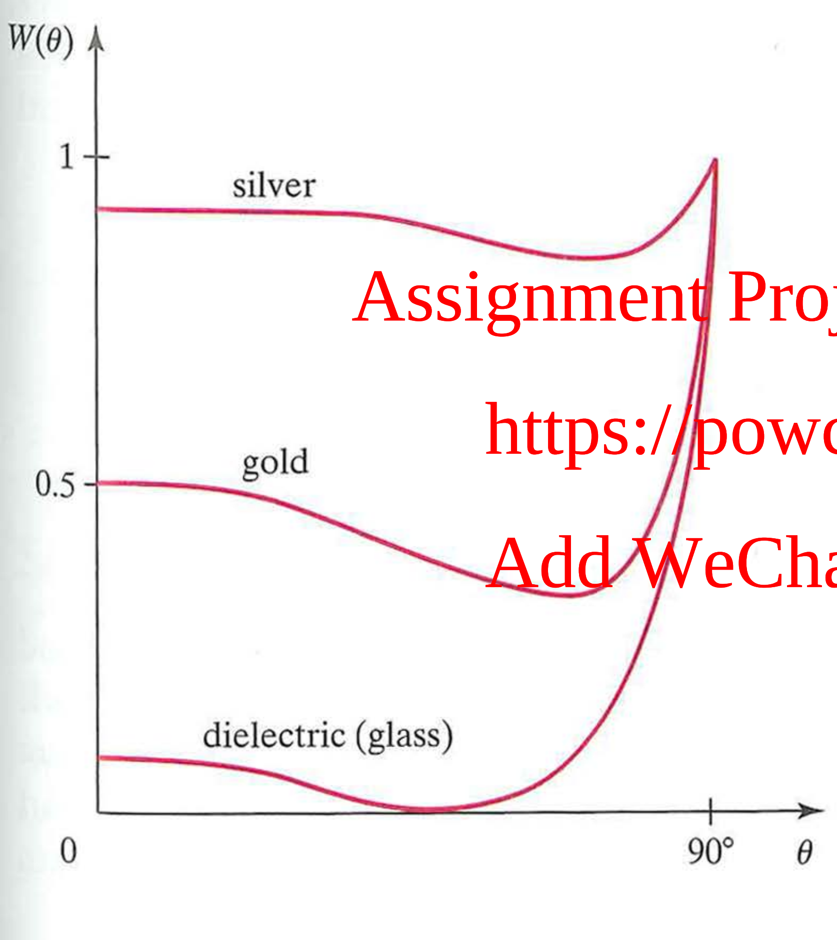
R unit specular reflection direction vector

R = (2**N**·**L**)**N** – **L**

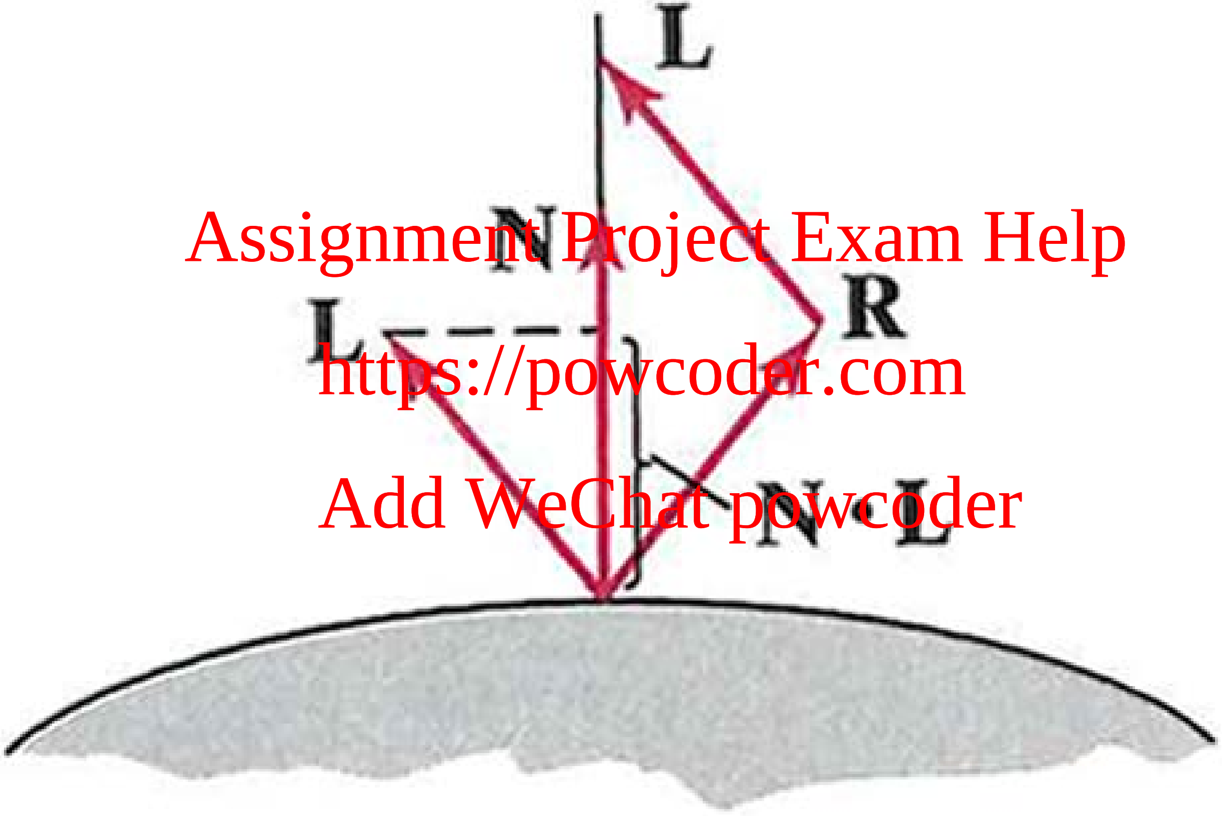
V unit viewing direction vector

cos(ϕ) = **R**·**V** $0 \le \phi \le \pi/2$

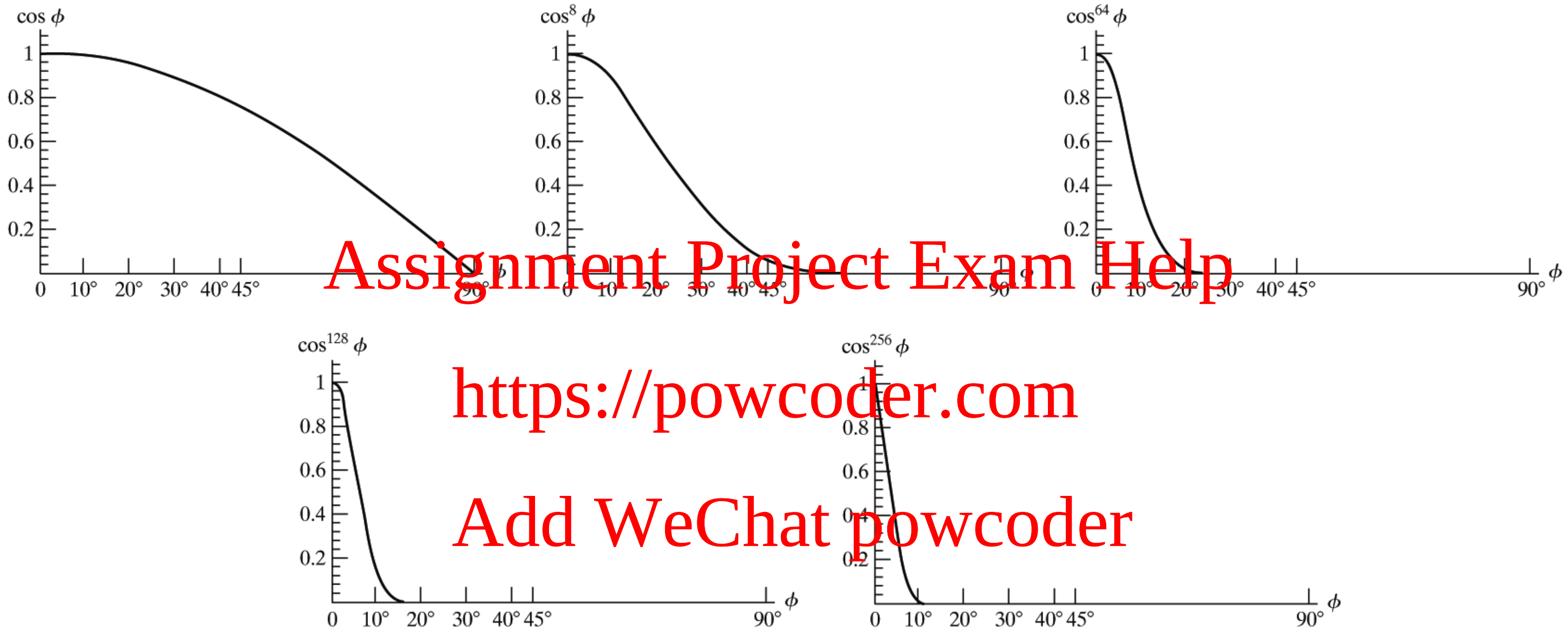
n_s specular reflection exponent, $n_s = \infty$ for perfect mirror



Approximate variation of the specular-reflection coefficient for different materials, as a function of the angle of incidence.



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



Plots of $\cos^{n_s} \phi$ using five different values for the specular exponent n_s.

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

$$I = k_a I_a + \sum_{i=1}^{n} I_{li} [k_d (\mathbf{N} \cdot \mathbf{L}_i) + W(\theta_i)(\mathbf{V} \cdot \mathbf{R}_i)^{n_s}]$$

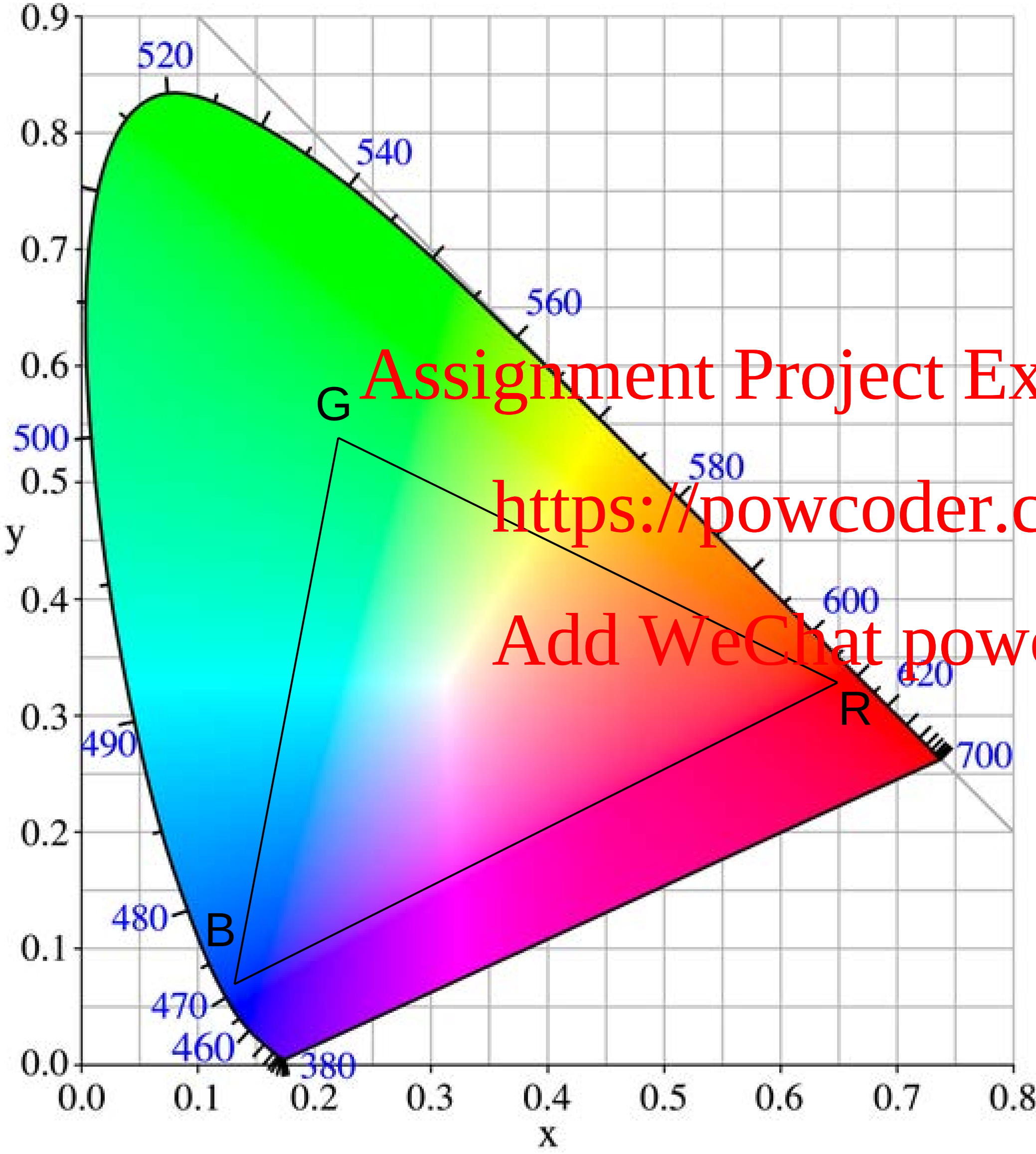
Colour model

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

$$I_R = k_{aR} I_{aR} + \sum_{i=1}^{n} I_{lRi} [k_{dR} (\mathbf{N} \cdot \mathbf{L}_i) + W_R(\theta_i)(\mathbf{V} \cdot \mathbf{R}_i)^{n_{sR}}]$$

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

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



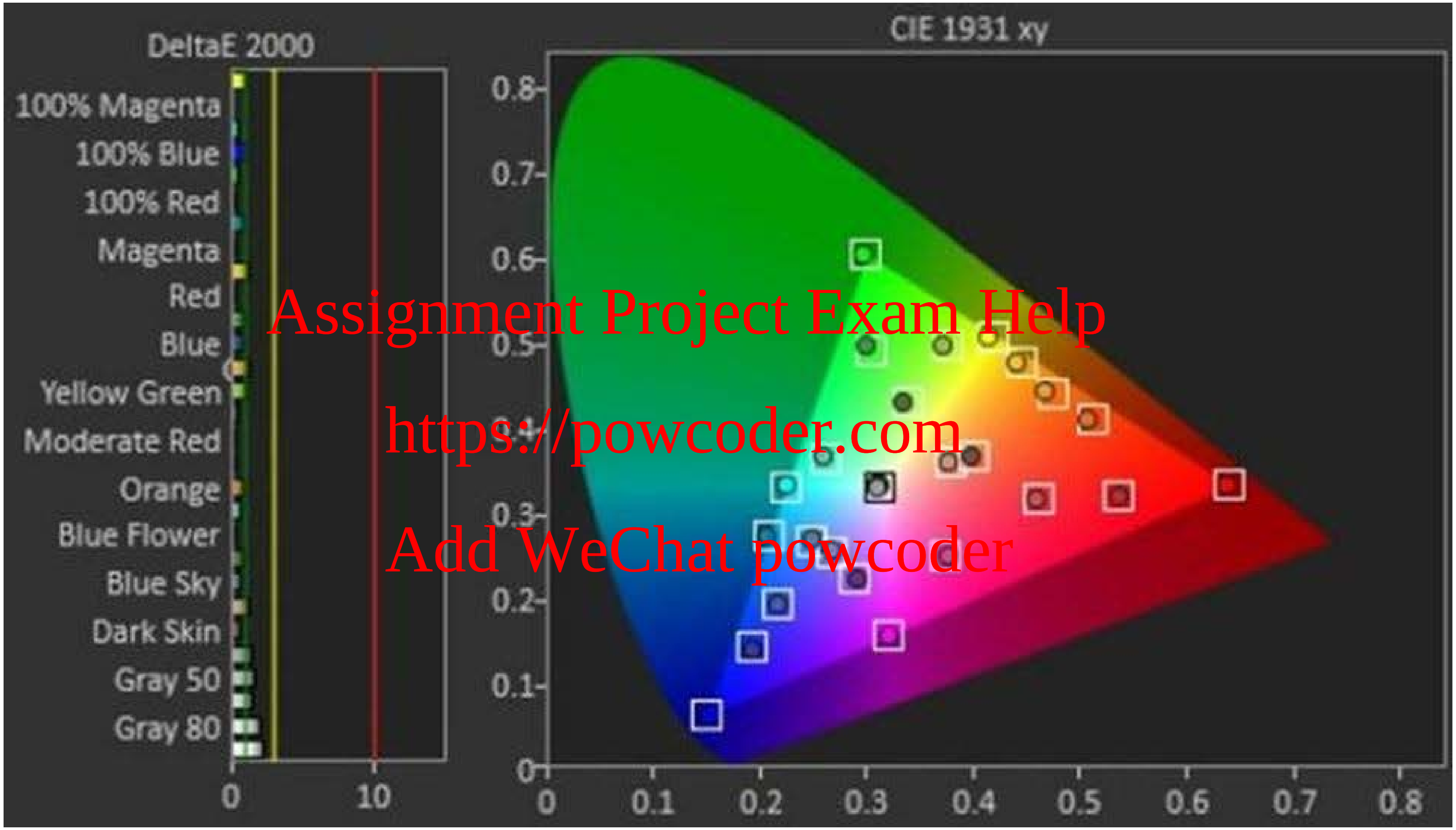
Note:

Only colours in the triangle is displayable.

Some naturally occurring colours outside the triangle cannot be displayed!

Quattron technology uses 4 primary Colours RYGB that extends the displayble colours

CIE chromaticity diagram
-Represent all possible colours seeable by humans


DeltaE 2000
100% Magenta
100% Blue
100% Red
Magenta
Red
Blue
Yellow Green
Moderate Red
Orange
Blue Flower
Blue Sky
Dark Skin
Gray 50
Gray 80
0
10
CIE 1931 xy
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

LG-32UD59-B

Shading Models /Rendering Models

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

- Three common ways to shade the polygons:
    - Flat Shading
    - Gouraud Shading
    - Phong Shading

Increasing realism

Increasing 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 - ugly!
- Human vision is subject to “Mach band effect” – intensity discontinuities are accentuated. This amplifies the edges of the polygons, which is undesirable

(a) (b) (c)

A polygon mesh approximation of an object (a) is displayed using flat surface rendering in (b) and using Gouraud surface rendering in (c).

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 eliminates the intensity discontinuities that occur in flat shading
- Slower (disadv.)
- Smooth out specular highlights (disadv.)

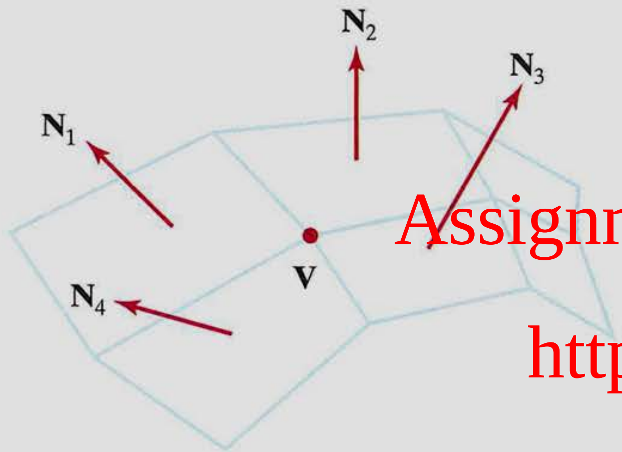
- Step 1 : Determine the average unit normal vector at each polygon vertex

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

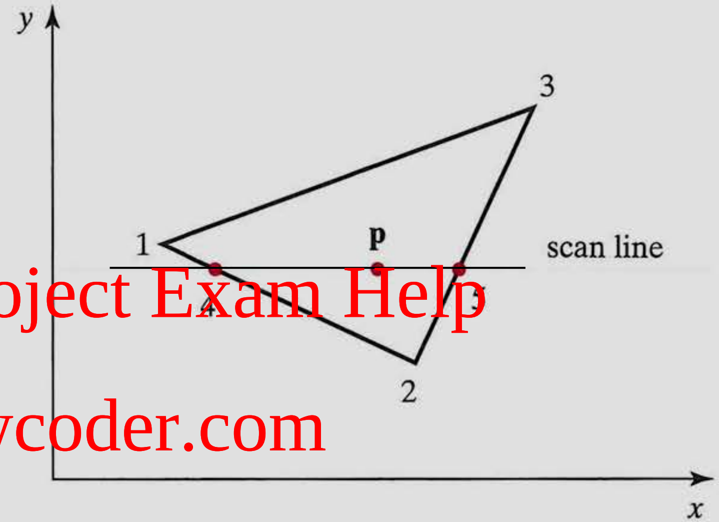
(each $\mathbf{N}_k$ is a unit vector, $\mathbf{N}_v$ is a unit vector by def.)

- Step 2 : Apply an illumination model to each vertex to calculate the vertex intensity

- Step 3 : linearly interpolate the vertex intensities over the surface of the polygon



The normal vector at vertex **V** is calculated as the average of the surface normals for each polygon sharing that vertex.



For Gouraud surface rendering, the intensity at point 4 is linearly interpolated from the intensities at vertices 1 and 2. The intensity at point 5 is linearly interpolated from intensities at vertices 2 and 3. An interior point **p** is then assigned an intensity value that is linearly interpolated from intensities at positions 4 and 5.

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 of the polygon : linearly interpolate across the scan line

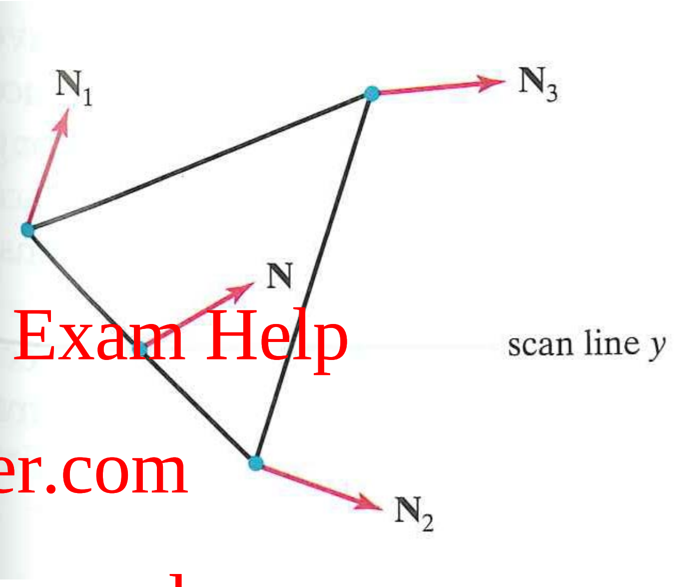
$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

Phong shading

- Similar to Gouraud shading, but interpolates **normal vectors** instead.
- Captures specular highlights
- Highest realism
- Slowest (disadv.)

- 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

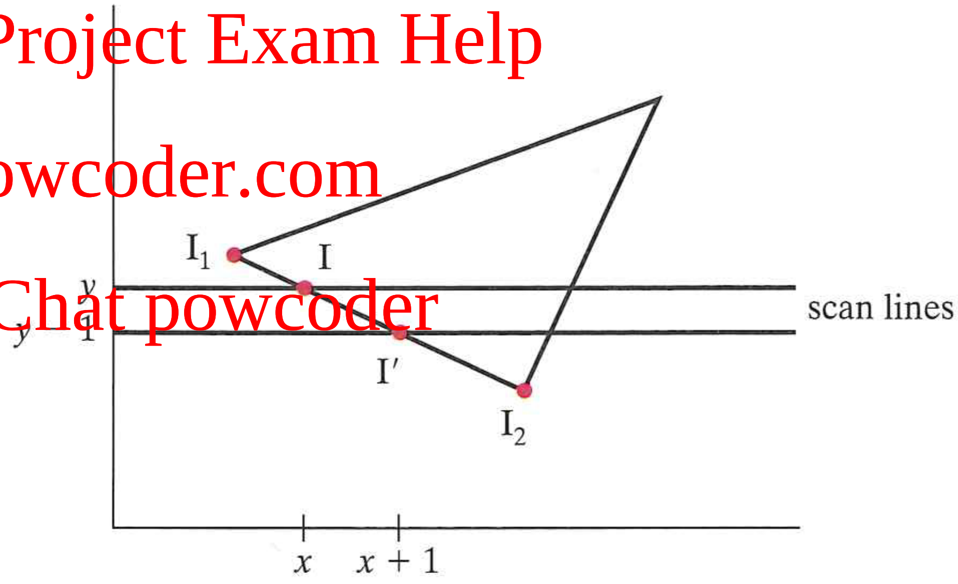
Incremental form

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

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

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
                                      //  source; the last entry is 1.0
GLfloat  light2PosType [ ] = {0.0, 1.0, 0.0, 0.0};    // light
                                      // direction; the last 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 blackColor [ ] = {0.0, 0.0, 0.0, 1.0};
GLfloat whiteColor [ ] = {1.0, 1.0, 1.0, 1.0};
```



```
glLightfv (GL_LIGHT3, GL_AMBIENT, blackColor);
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 [ ] = {1.0, 1.0, 1.0, 1.0}; // $W_R(\theta)$ = 1.0, …

glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT_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);

surfRenderingMethod = *GL_FLAT* Flat shading

= *GL_SMOOTH* 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 (normalVector1); // normal vector at vertex1 calculated
                                   // by average 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. 17.10 for different shading method
- Text: Ch. 17.11 for OpenGL commands
- Demo: Run lightposition.exe and lightmaterial.exe in TUTORS program
- Quattron technology: http://en.wikipedia.org/wiki/Quattron