CMT107 Visual Computing

IV.2 Polygon Shading

Xianfang Sun

School of Computer Science & Informatics
Cardiff University

Overview

- Shading polygons
  - Flat shading
  - Gouraud shading
  - Phong shading
- Special effects
  - Transparency
  - Refraction
  - Atmospheric effects
- OpenGL Shading

Shading

- The colour of 3D objects is not the same everywhere
  - An object drawn in a single colour appears flat
  - Light-material interactions cause each point to have a different colour or *shade* in 3D
- *Global* shading requires to calculate all reflections between all objects
  - In general this is not computable
- We use a simplified *local* model: *Phong illumination*

$$R_a \mathbb{I}_a + R_d (n^t d) \mathbb{I}_d + R_s (r^t v)^\sigma \mathbb{I}_s$$

Polygon Shading

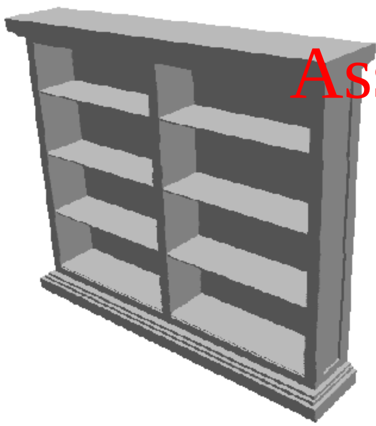
- Use Phong Illumination for *polygon shading* (e.g. with scan-line to set colours of pixels)
  - Need to compute *surface normals*
  - Polygon approximates 3D shape (normals may not be normals of actual polygon)

- Different approaches to polygon shading:
  - *Flat* shading
  - *Gouraud* shading
  - *Phong* shading

Flat Shading

- *One illumination calculation* per polygon
  - Each pixel is assigned the same colour
  - Usually computed for centroid of polygon:

$$\text{centroid} = \frac{1}{\text{vertices}} \sum_{l=1}^{\text{vertices}} p_l$$



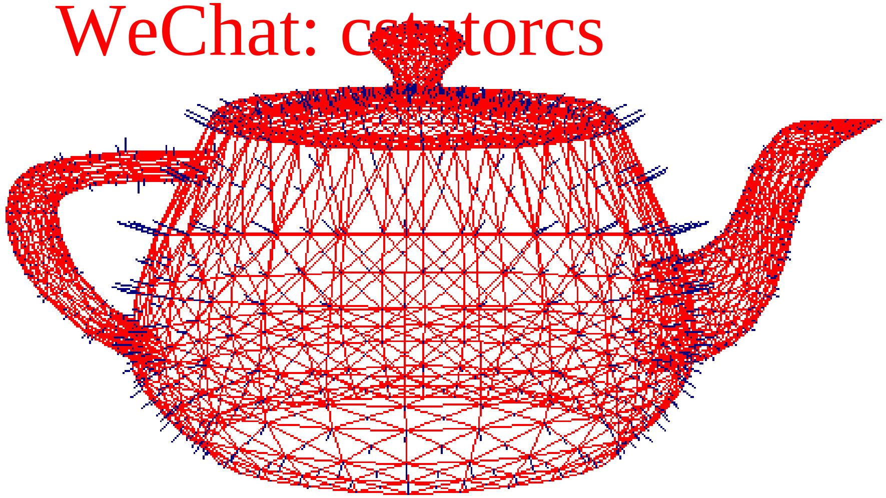
- Good for polyhedral objects, but:
  - For point light sources, *direction to light varies*
  - For specular reflections, *direction to eye varies*

Vertex Normals

- Introduce *surface normals* for each vertex
  - Usually *different* from polygon normal
  - Either *exact* normals of surface
  - Or *average* of normals of polygons meeting at a vertex

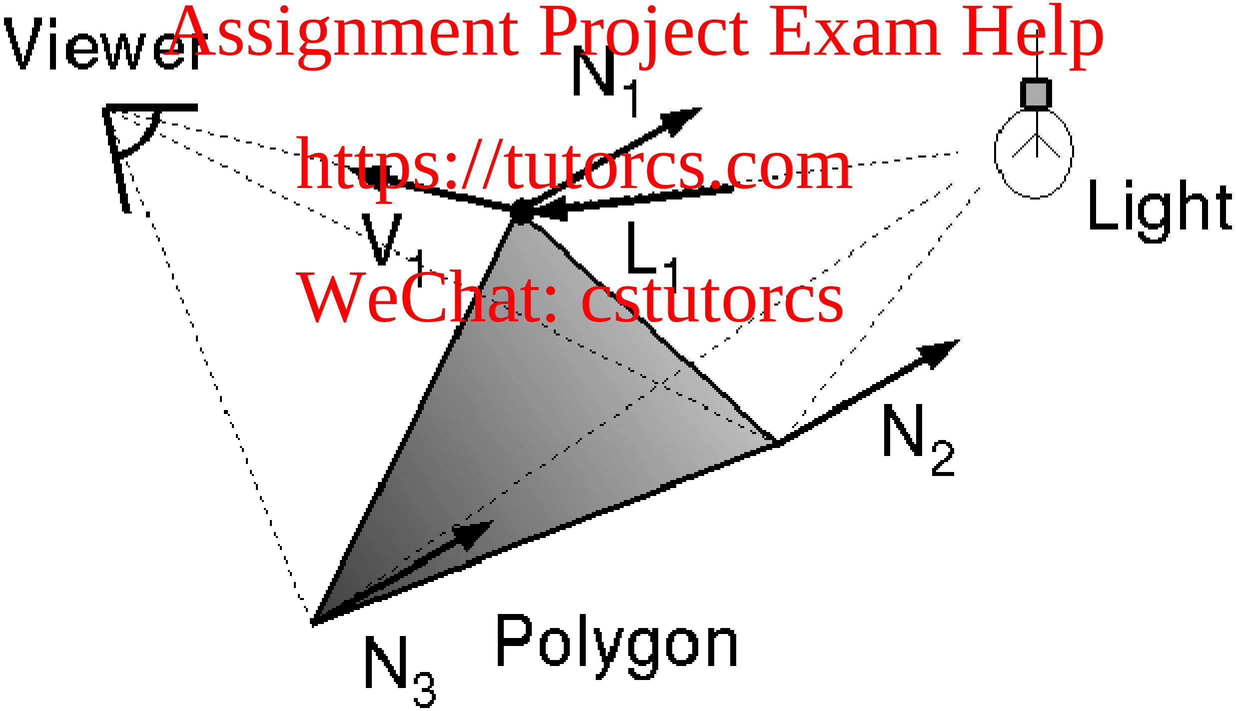
$$n_v = \sum_{l=1}^{\text{polygons}} \frac{n_l}{||n_l||}$$

(good if polygons approximate surface well)



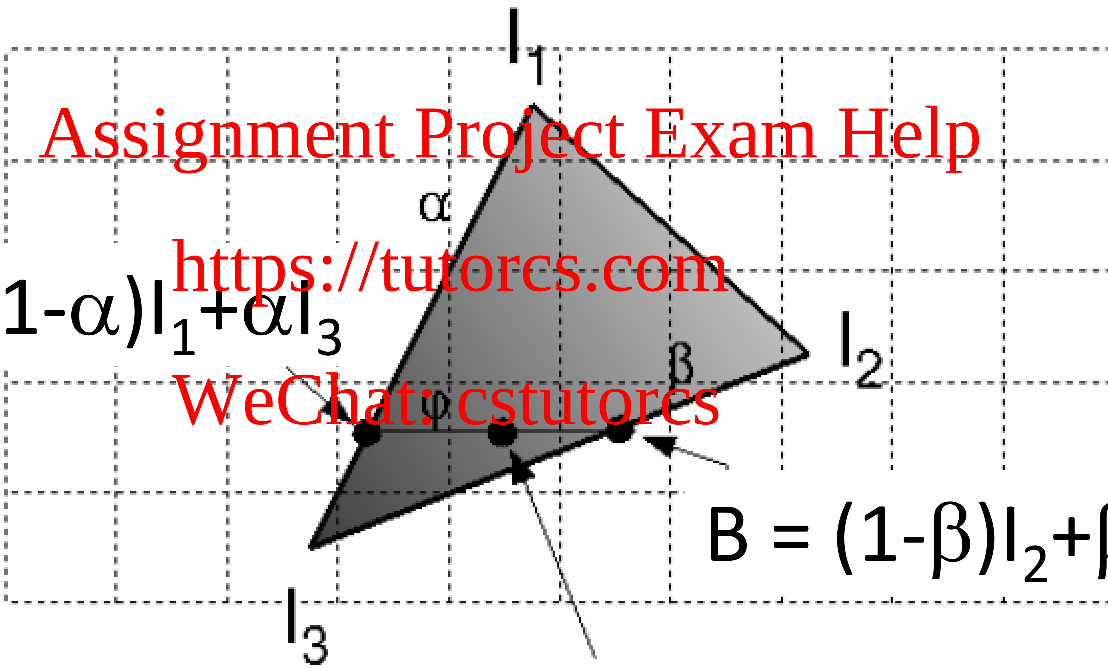
Gouraud Shading

- Compute illumination for vertices of polygon
  - Use vertex normals
  - *Linearly interpolate colours* between vertices



Gouraud Shading Interpolation

- *Bilinearly interpolate* colours between vertices down and across scan lines



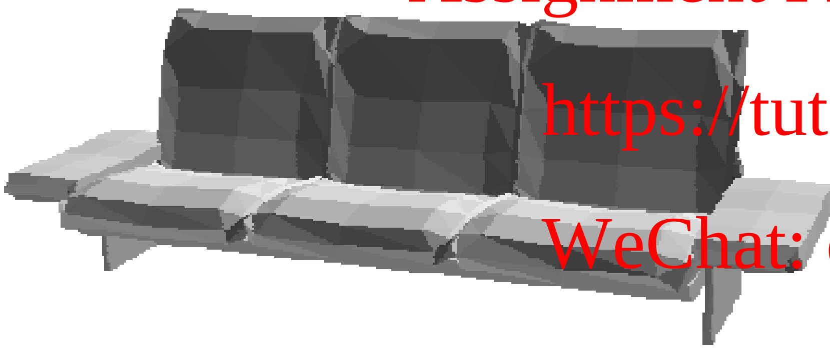
$A = (1-\alpha)I_1+\alpha I_3$

$B = (1-\beta)I_2+\beta I_3$

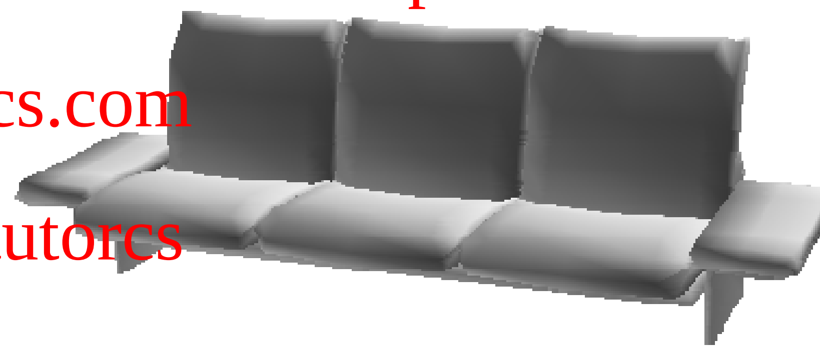
$I = (1-\varphi)A+\varphi B$

Gouraud Shading Example

- Creates *smoothly* shaded polygonal mesh
- *Artefacts* still visible
- Need a *fine mesh* to capture subtle lighting effects



Flat Shading



Gouraud Shading

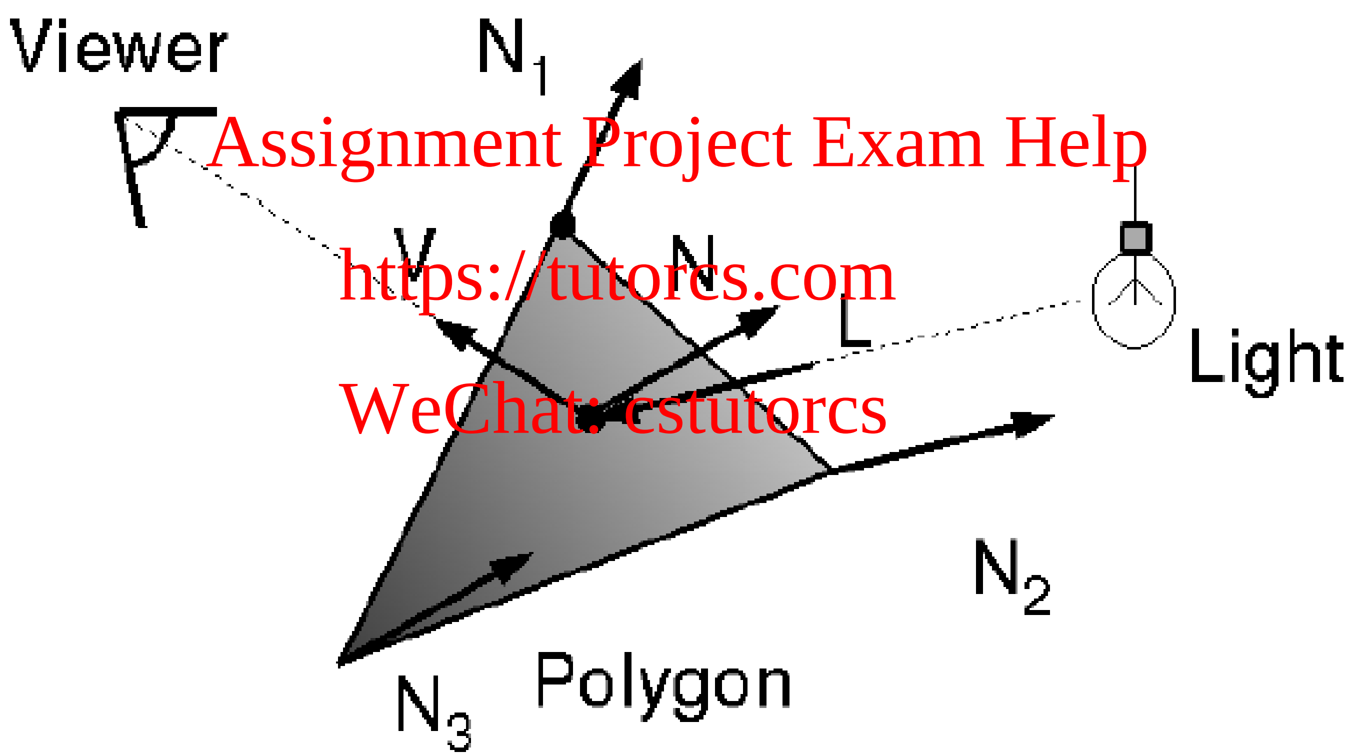
Phong Shading

- *One lighting calculation per pixel*
  - *Linearly interpolate vertex normals* across polygon

- *Very smooth* appearance, but *artefacts along silhouettes*
- Do not confuse with Phong illumination model!

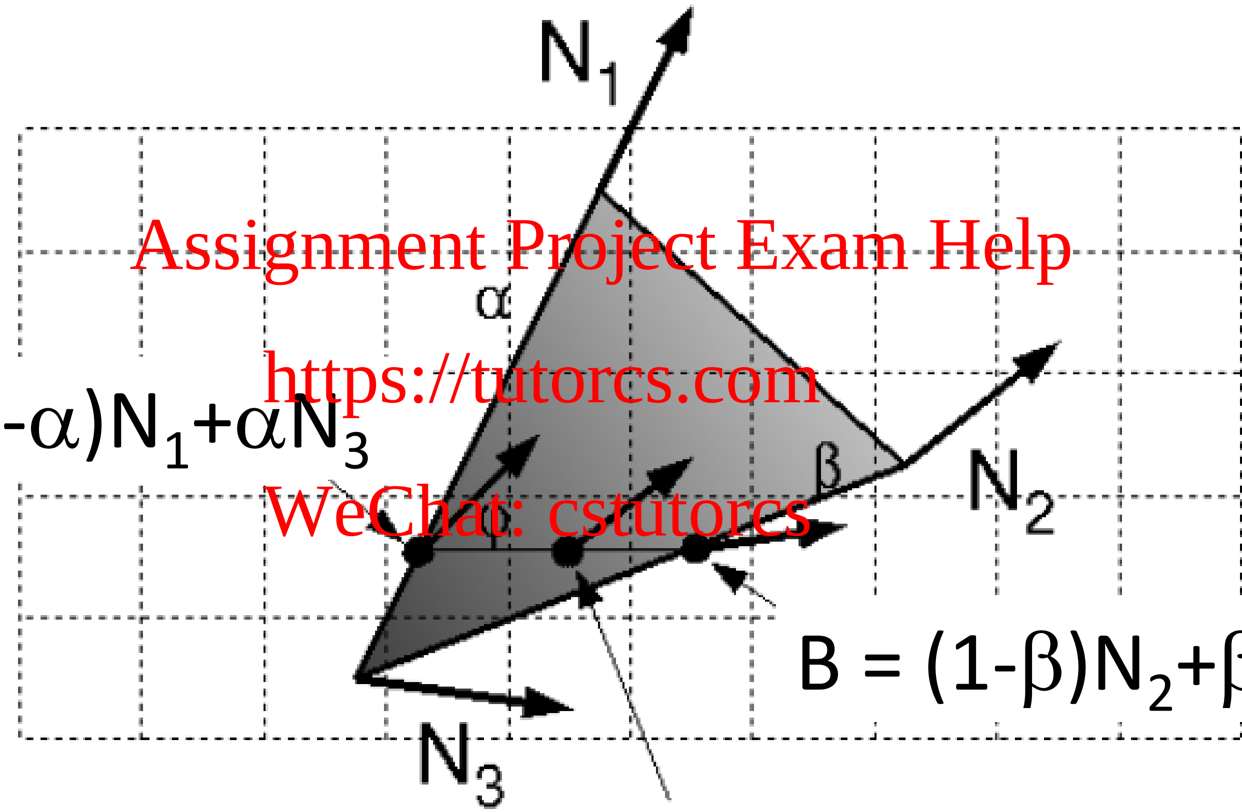
Phong Shading Interpolation

- *Bilinear interpolation* of normals from vertices



Phong Shading Interpolation

- *Bilinear interpolation* of normals from vertices



$A = (1-\alpha)N_1+\alpha N_3$

$B = (1-\beta)N_2+\beta N_3$

$N = (1-\varphi)A+\varphi B$

Shading Notes

- Be careful when transforming surface normals
  - Normals are not points, but a surface property
  - Point transformations are different from normal transformations
    (point transformation A becomes $(A^{-1})^t$ for normals)
- Advanced shaders implemented on GPU in OpenGL SL

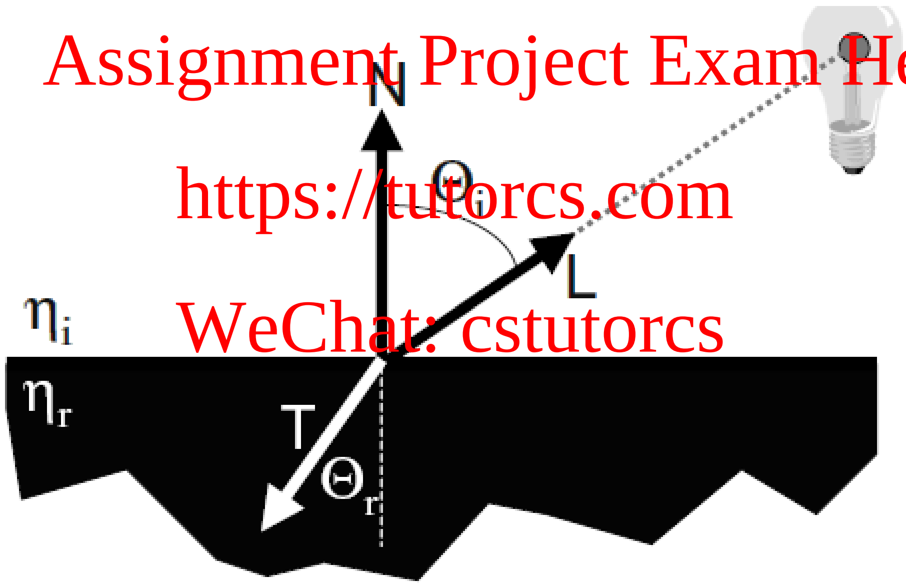
Transparency

- *Opacity coefficient* k tells how much light is blocked:
- $$I = kI_{reflected} + (1 - k)I_{transmitted}$$
  - $k \in [0,1]$: 0 for transparent surface, 1 for opaque surface
  - $I_{reflected}$ is intensity of reflected light
  - $I_{transmitted}$ is intensity of transmitted light from behind the surface
- Requires *expansion of visible surface detection* to access polygons further behind
  - Use *A buffer*

Snell's Law

- *Refraction* direction required for physically correct transparency computation
- *Snell's law*

$$\eta_r \sin\Theta_r = \eta_i \sin\Theta_i$$



$$\mathbf{T} = \left(\frac{\eta_i}{\eta_r} \cos\Theta_i - \cos\Theta_r \right) \mathbf{N} - \frac{\eta_i}{\eta_r} \mathbf{L}$$

Snell's Law

Vector decomposition:

$$\mathbf{L} = \cos\Theta_i \mathbf{N} + \sin\Theta_i S \qquad S = \frac{1}{\sin\Theta_i}\left(\mathbf{L} - \cos\Theta_i \mathbf{N}\right)$$

where S is a vector on the horizontal direction.

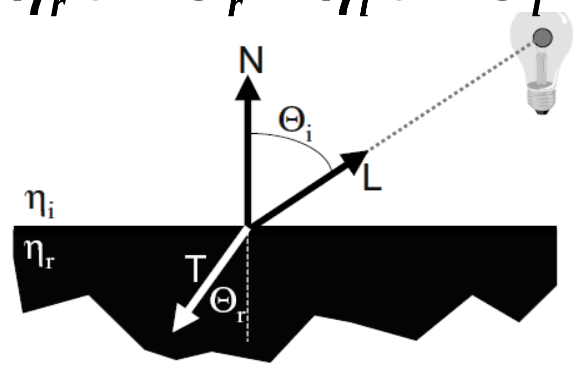
$$\mathbf{T} = -\cos\Theta_r \mathbf{N} - \sin\Theta_r S$$

$$\mathbf{T} = -\cos\Theta_r \mathbf{N} - \frac{\sin\Theta_r}{\sin\Theta_i}\left(\mathbf{L} - \cos\Theta_i \mathbf{N}\right)$$

$$= -\cos\Theta_r \mathbf{N} - \frac{\eta_i}{\eta_r}\left(\mathbf{L} - \cos\Theta_i \mathbf{N}\right)$$

$$= \left(\frac{\eta_i}{\eta_r}\cos\Theta_i - \cos\Theta_r\right)\mathbf{N} - \frac{\eta_i}{\eta_r}\mathbf{L}$$

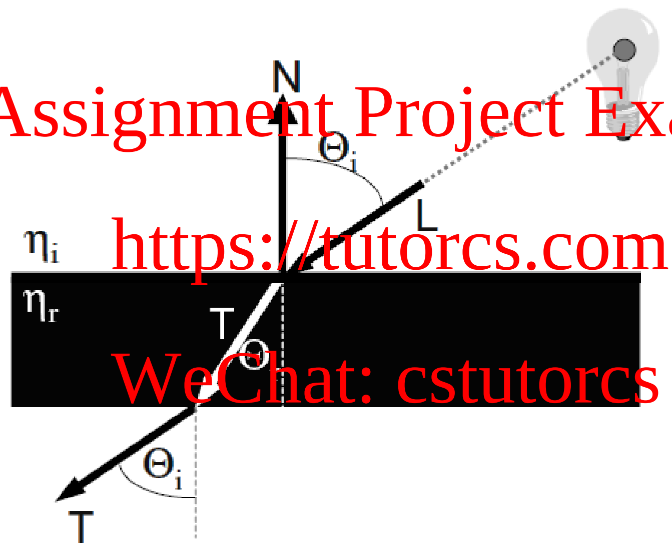
$$\boldsymbol{\eta_r} \sin\Theta_r = \boldsymbol{\eta_i} \sin\Theta_i$$



$$\mathbf{T} = \left(\frac{\eta_i}{\eta_r}\cos\Theta_i - \cos\Theta_r\right)\mathbf{N} - \frac{\eta_i}{\eta_r}\mathbf{L}$$

Refraction

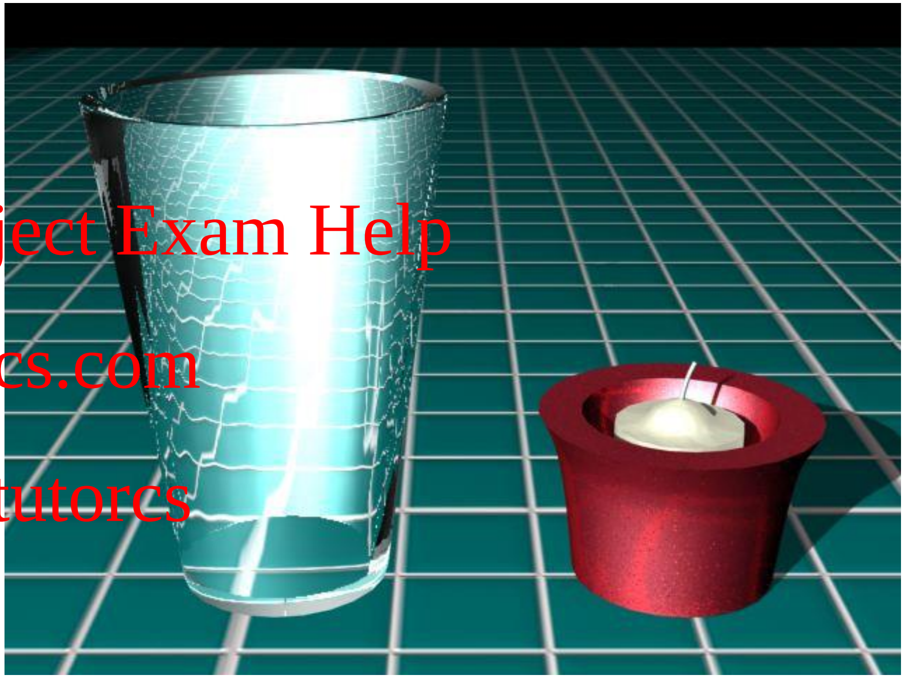
- Refraction of light through glass
  - Emerging refracted ray travels along a path parallel to incoming light ray



- Usually *ignore refraction*
  - Assume light travels straight through surface (good approximation for thin polygonal surfaces)

Refraction Example

No Refraction



With Refraction

Atmospheric Effects

- Similar to transparency, modify light intensities for fog, smoke, etc.

$$I = f_{atmo}(d)I_{object} + (1 - f_{atmo}(d))I_{atmo}$$

  - I_{object} is intensity from visible object
  - I_{atmo} is intensity for atmospheric effect
  - $f_{atmo}(d)$ is function modelling atmospheric effect depending on distance d from viewer, e.g.:

$$f_{atmo,1}(d) = e^{-cd}$$

$$f_{atmo,2}(d) = e^{-(cd)^2}$$

$$f_{atmo,3}(d) = (End-d)/(End-Start)$$

OpenGL Shading

- Fixed-function pipeline version of OpenGL uses specific functions to realise flat and Gouraud shading, transparency, and fog effect
- Shader version of OpenGL needs the programmer to write code in the main program and/or the shaders to implement these effects
- More details in the labs …

Summary

- How does flat, Gouraud, Phong shading for polygons work? What are the differences / similarities between the different shading algorithms?
- Why do we need explicit surface normals for vertices?
- How can we add transparency and atmospheric effects to our lighting computations?
- What is refraction / Snell's law?
- Why is refraction usually ignored?