CMT107 Visual Computing

V.1 Illumination Models

Xianfang Sun

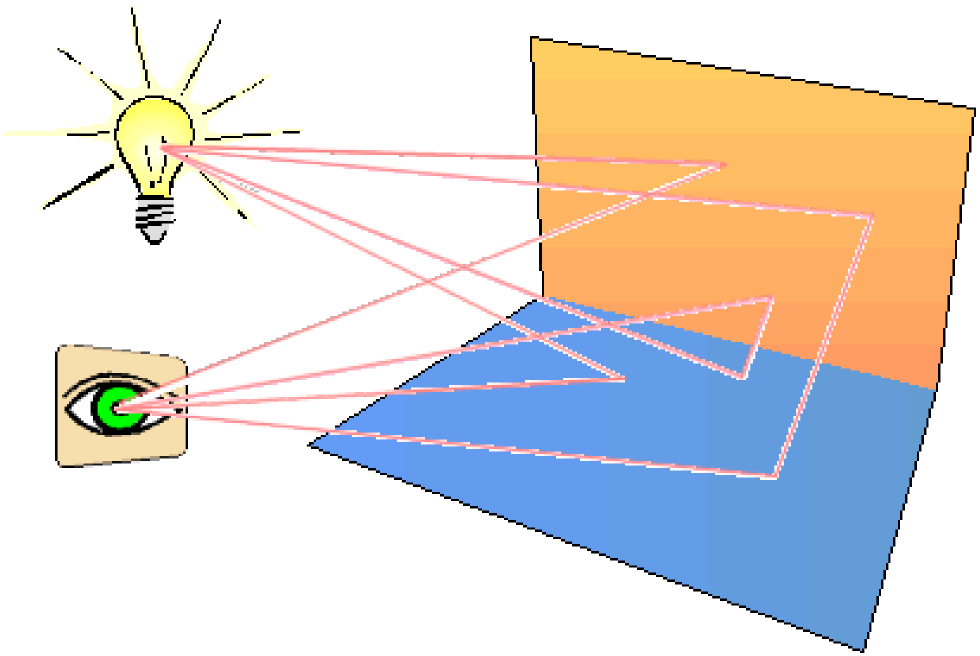
School of Computer Science & Informatics
Cardiff University

Overview

- Illumination Concepts
- Light Reflection model
  - Phong illumination model
- Light source types
- OpenGL lighting

Illumination Concepts

- *Illumination:* transport of luminous flux from light sources between points via direct and indirect paths
- *Lighting:* computing luminous intensity reflected from a specific 3D point
- *Shading:* assigning colours to a pixel
- Illumination Models: Simple approximations of light transport

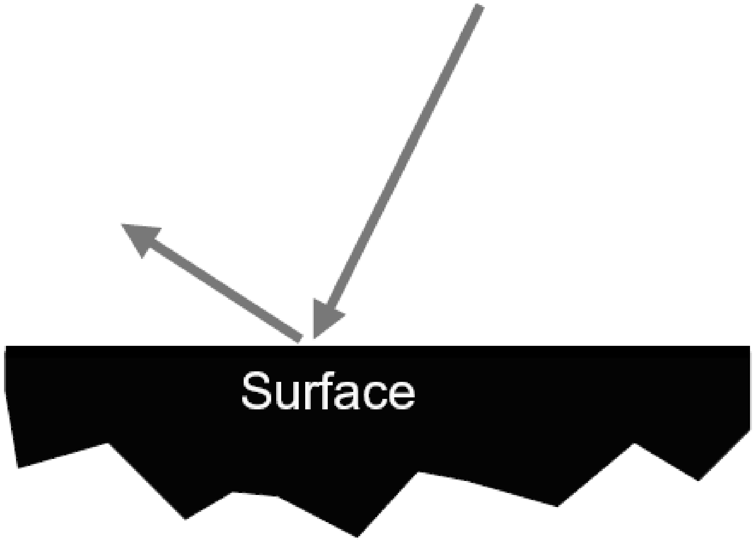


Light-Surface Interaction

- Light and surface properties determine the illumination
- Light that strikes an object is partially absorbed and partially reflected
- The amount reflected determines the colour and brightness of the object (subtractive colours)
- Reflected light is scattered depending on the smoothness and orientation of the surface

Modelling Surface Reflectance

- Compute light *reflected* by surface as *observed by viewer*
- Surface material tells *how much* of the incoming light is reflected
  - Type of light determines reflection model
- Intensity of observed light depends on *direction to light source* and *direction to viewer*



Light Reflection Types

- *Ambient* light: comes from all directions, is scattered in all directions
- *Diffuse* light: comes from one direction, is scattered in all directions
- *Specular* light: comes from one direction, reflected in preferred direction (highlights)

Ambient Reflection

- Ambient light is the same everywhere
  - Amount of reflected light of incoming intensity $\mathbb{I}_{\text{ambient,c}}$ is *independent* of direction to light source and viewer
- Intensity of reflected light observed by a viewer:

$$\mathrm{L}_{\text{ambient,c}} = \mathrm{R}_{\text{ambient,c}} \mathbb{I}_{\text{ambient,c}}$$

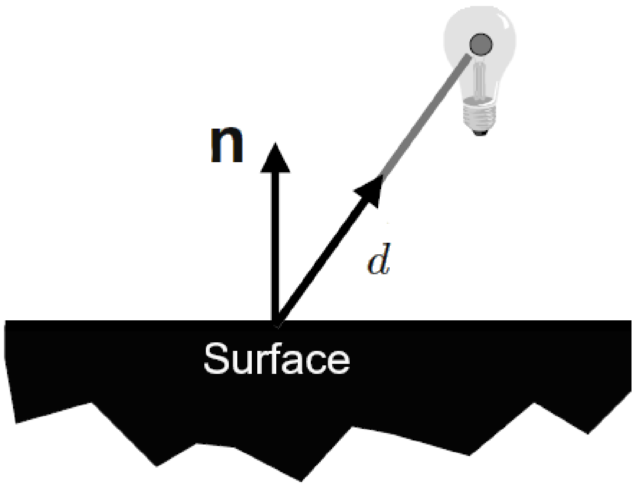
  - $\mathrm{R}_{\text{ambient,c}}$ is ambient material property for colour c (percentage of red, green or blue ambient light reflected by surface)

Diffuse Reflection

- Light is reflected in all directions
  - Amount of reflected light of incoming intensity *depends only on direction to light source*
- *Lambertian model* (use cosine law / scalar product):

$$L_{\text{diffuse},c} = R_{\text{diffuse},c}(n^t d) I_{\text{diffuse},c}$$

  - *d*: unit direction from surface point to light source
  - *n*: unit surface normal

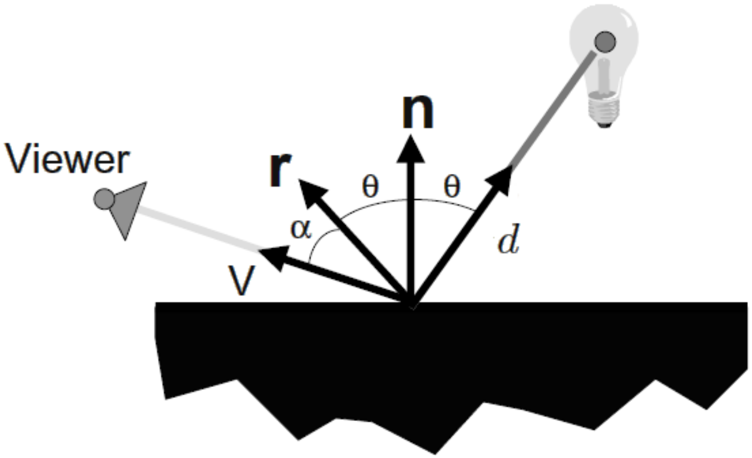


Specular Reflection

- Light is reflected preferable in *direction of perfect reflection*
  - Amount of reflected light of incoming intensity depends on direction to light source and to viewer
- Observed light intensity:

$$L_{\text{specular},c} = R_{\text{specular},c}(r^t v)^{\sigma} I_{\text{specular},c}$$

  - *r*: unit direction of perfect reflection of d
  - *v*: unit direction towards viewer position
  - σ is shininess exponent



Surface Light Emissions

- Can make surface *emit* light, not just reflect light
- Simple model:
  - Add emissive light intensities $E_{t,c}$ to light intensities for each light type t and colour c
  - Does not illuminate other surfaces
    (but can add a multiple point light sources behind surface or a directional light source for larger light emitting surfaces)

Phong Illumination Model

- Putting everything together gives the *Phong Illumination Model*
- Consider monochromatic light (e.g. red, green or blue) and a single light source:
  - Depending on light source type, at a surface point the incoming intensity of different light types is $\mathbb{I}_a, \mathbb{I}_d, \mathbb{I}_s$
  - The *intensity of reflected light* is:

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

  - *Summation* over all light sources for red, green, blue gives total intensity for all colours
- Note, Phong's illumination model is *not* physically accurate

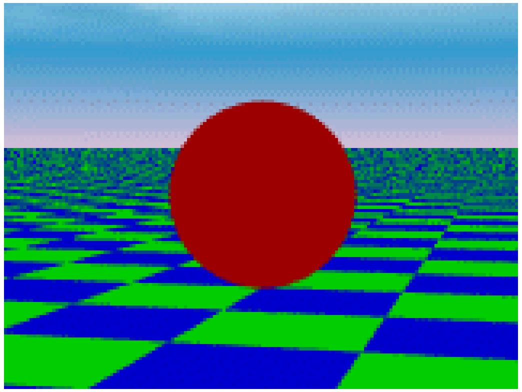
Light Source Types

- Ambient light source: light from the environment
- Directional light source: light from infinite distance in a specified direction
- Point light source: light from single point
- Spot light source: light emitted in a cone
- other light source: area light source, extended light source etc.

Ambient Light Source

- An object not directly lit is still visible
  - Caused by light reflected from other surfaces
- Modelled by a single ambient light source
  - Instead of computing surface reflections, specify *constant ambient light* for all surfaces
  - Defined solely by ambient RGB light *intensities*
- Intensity arriving at point p from an ambient light of intensity $L_{\text{ambient},c}$ and colour *c* :

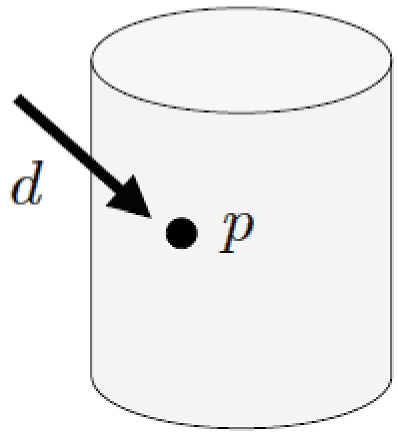
$$I_{\text{ambient}}(p, L_{\text{ambient},c}) = L_{\text{ambient},c}$$

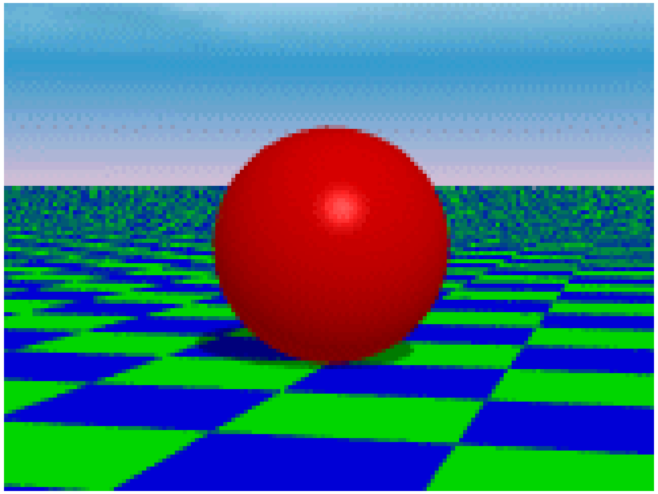


Directional Light Source

- Light from a source *infinitely far away*
  - Defined by *intensities* of emitted RGB light of all types,
  - *direction* $\boldsymbol{d}, \|\boldsymbol{d}\| = 1$ (and no position)
- Intensity arriving at point p from a directional light of intensity $L_{t,c}$:

$$I_{\mathrm{directional:d}}(p, L_{t,c}) = L_{t,c}$$

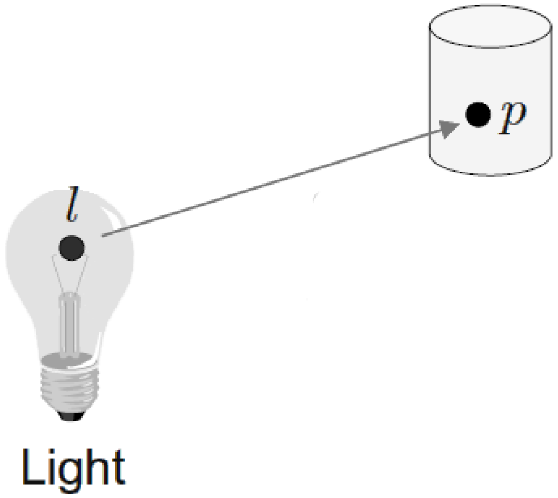


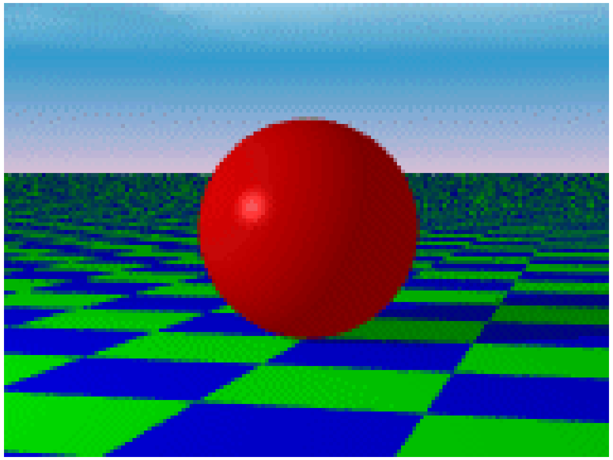


Point Light Source

- Light emitted *radially* from single point *in all directions*
  - Defined by *intensities* of emitted RGB light for all types,
  - *position* l (and no direction),
  - constant, linear and quadratic *attenuation* (k_c, k_l, k_q)
- Intensity arriving at point p from a point light of intensity L_c :

$$I_{point:l,k_c,k_l,k_q}(p, L_{t,c}) = \frac{1}{k_c + k_l\|p - l\| + k_q\|p - l\|^2} L_{t,c}$$

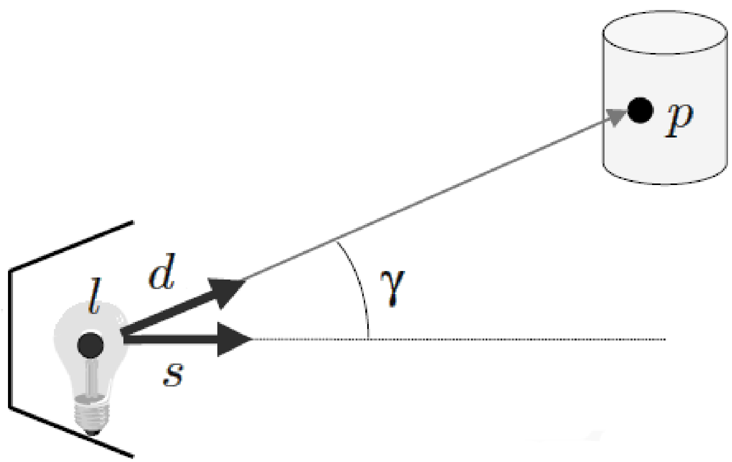


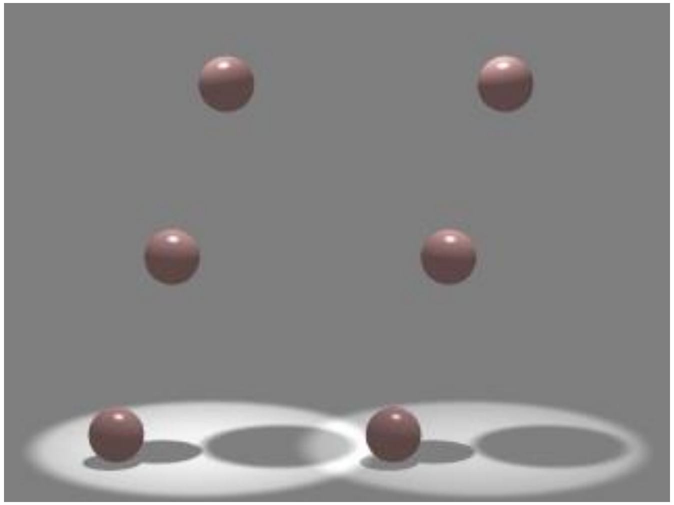


Spot Light Source

- Light emitted in a *cone*
  - Defined by *intensities* of emitted RGB light for all types,
  - *position* l, *unit cone direction* s, *spot cut-off exponent* τ,
  - constant, linear and quadratic *attenuation* (k_c, k_l, k_q)
- Intensity arriving at point p from an point light of intensity $L_{t,c}$:

$$I_{spot:l,s,\tau,k_c,k_l,k_q}(p, L_{t,c}) = \frac{(s^t((p-l)/\|p-l\|))^\tau}{k_c + k_l\|p-l\| + k_q\|p-l\|^2} L_{t,c}$$





Light Source "Visibility"

- *Angle cut-off* for spot lights:
  - If position p is outside light cone ($s^\mathsf{T}d = \cos\gamma < \cos\delta$ with $\boldsymbol{d} = (\boldsymbol{p} - \boldsymbol{l})/\|\boldsymbol{p} - \boldsymbol{l}\|$ and cone semi-angle δ), set I to 0
- Light source *behind* surface:
  - Diffuse and specular light only reflected if light source is in front of surface at p
  - Set diffuse and specular light intensities from light sources to 0 if $n^\mathsf{T}d \leq 0$
    - n: unit surface normal at p
    - d: unit direction from p to light source
  - This distinguishes between front and back of surfaces / polygons (also see two-sidedness)

OpenGL lighting

- Fixed-function pipeline version of OpenGL (old version) uses specific functions to define lighting and material properties. And lighting effects are realised inside the OpenGL pipeline
- Shader version of OpenGL (new version) needs the programmer to write code in the main program and/or the shaders to implement lighting effects
- More details in the labs …

Surface Normal Vectors

- For lighting computations OpenGL requires *normal vectors* of polygonal primitives
  - Orthogonal to surface pointing outwards
  - Used to compute reflection angle
- Normals are sent to the vertex shader together with vertex coordinates
- Normals should be unit vectors
  - The function normalize() in shaders can be used to convert a vector to a unit vector:

```
Vn = normalize(V);
```

Summary

- What is ambient, diffuse and specular light? How is the amount of reflected light for each light type computed?
- What is the Phong illumination model?
- What are ambient, directional, point and spot light sources? How is the light intensity arriving from one of these light sources at a surface point computed?
- Distinguish light reflection types and light source types.

CMT107 Visual Computing

V.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*

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

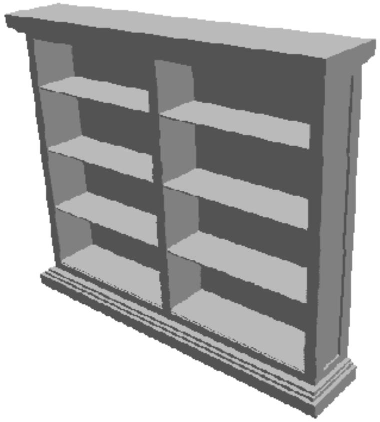
Polygon Shading

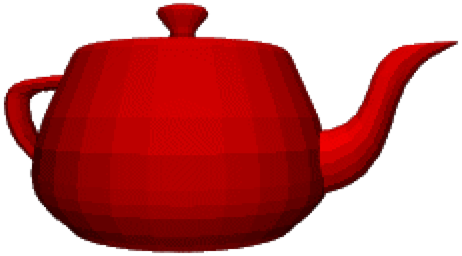
- Use Phong Illumination for *polygon shading*
  (e.g. with scan-line to set colours of pixels)
  - Need to compute *surface normals*
  - Polygon approximates 3D shape
    (normals may not be normals of actual polygon)
- Different approaches to polygon shading:
  - *Flat* shading
  - *Gouraud* shading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

Flat Shading

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

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





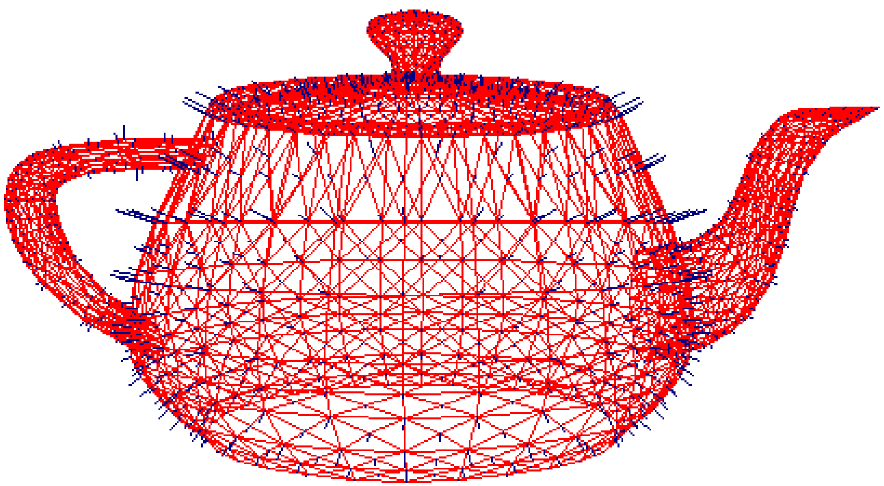
- Good for polyhedral objects, but:
  - For point light sources, *direction to light varies*
  - For specular reflections, *direction to eye varies*

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

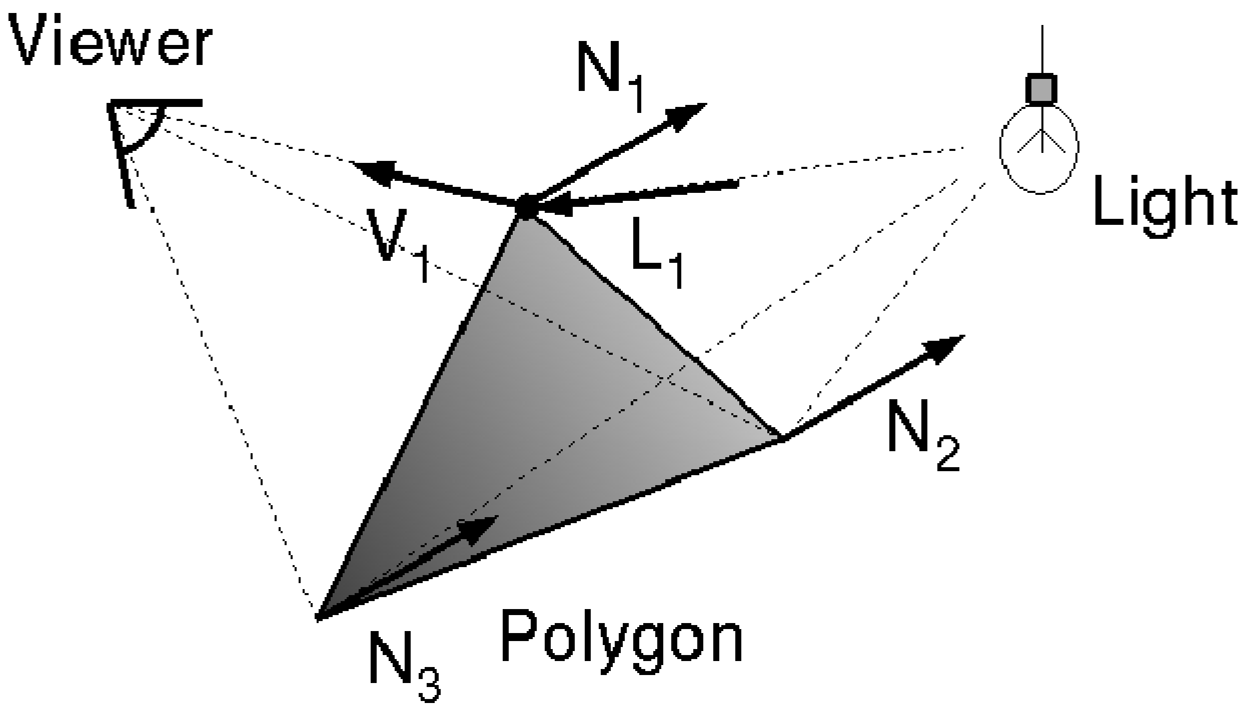
$$n_v = \sum_{l=1}^{\text{polygons}} \frac{n_l}{\|n_l\|}$$

(good if polygons approximate surface well)



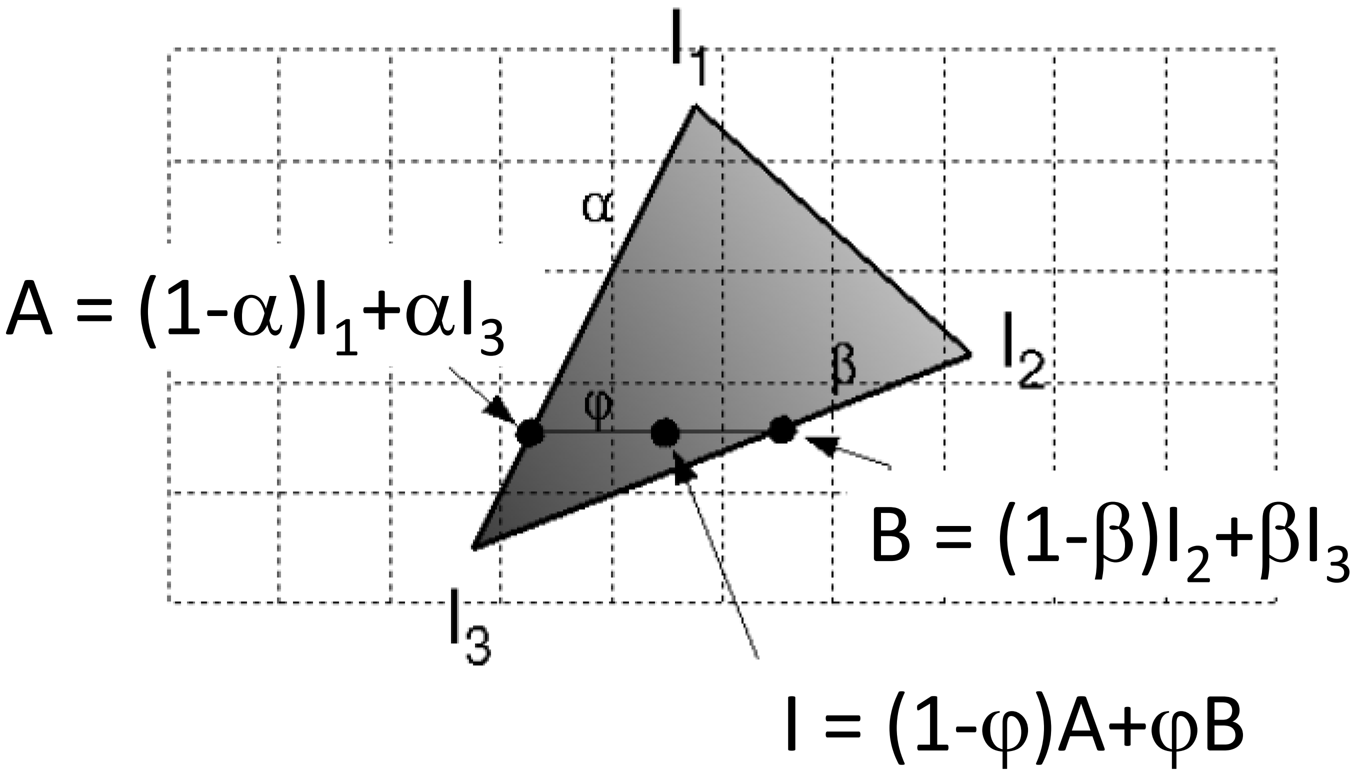
Gouraud Shading

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



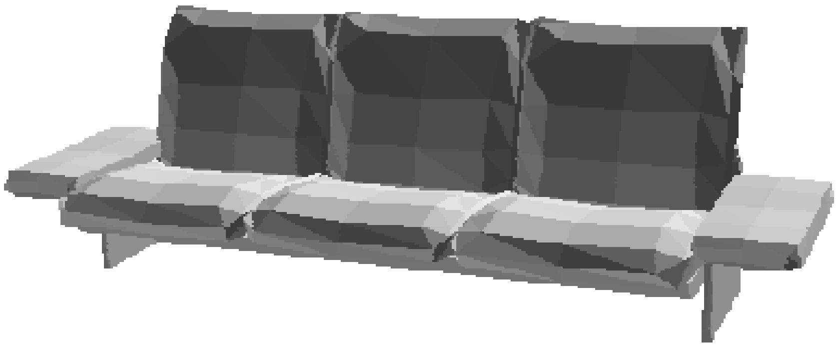
Gouraud Shading Interpolation

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

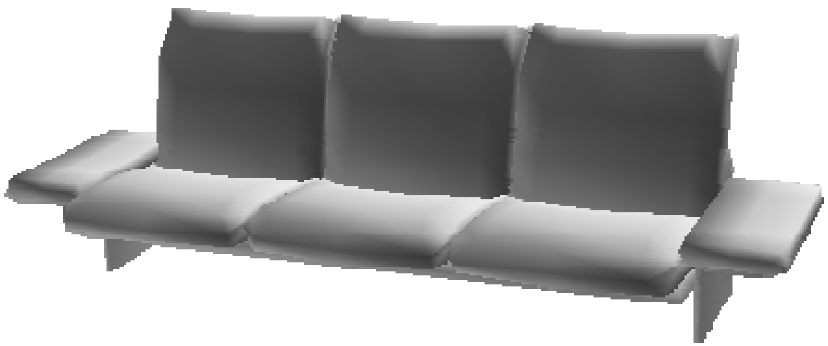


Gouraud Shading Example

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



Flat Shading



Gouraud Shading

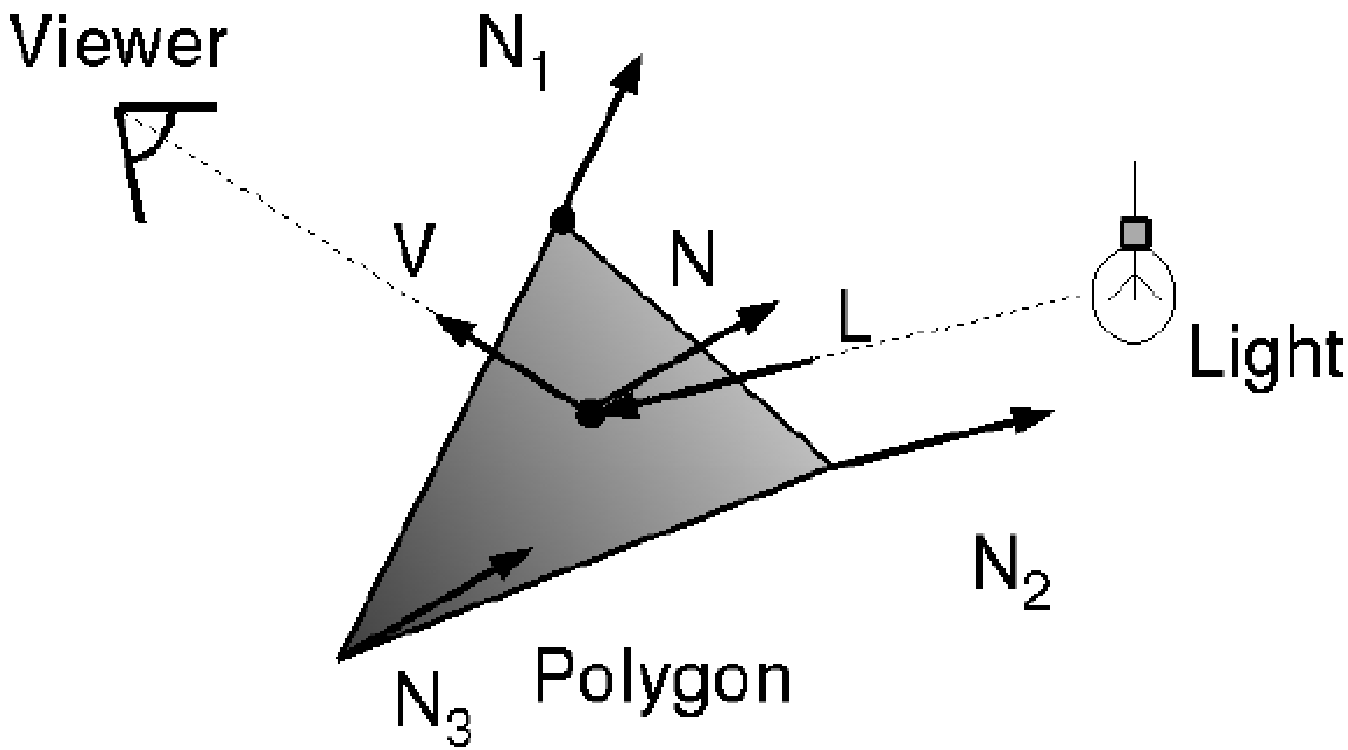
Phong Shading

- *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!

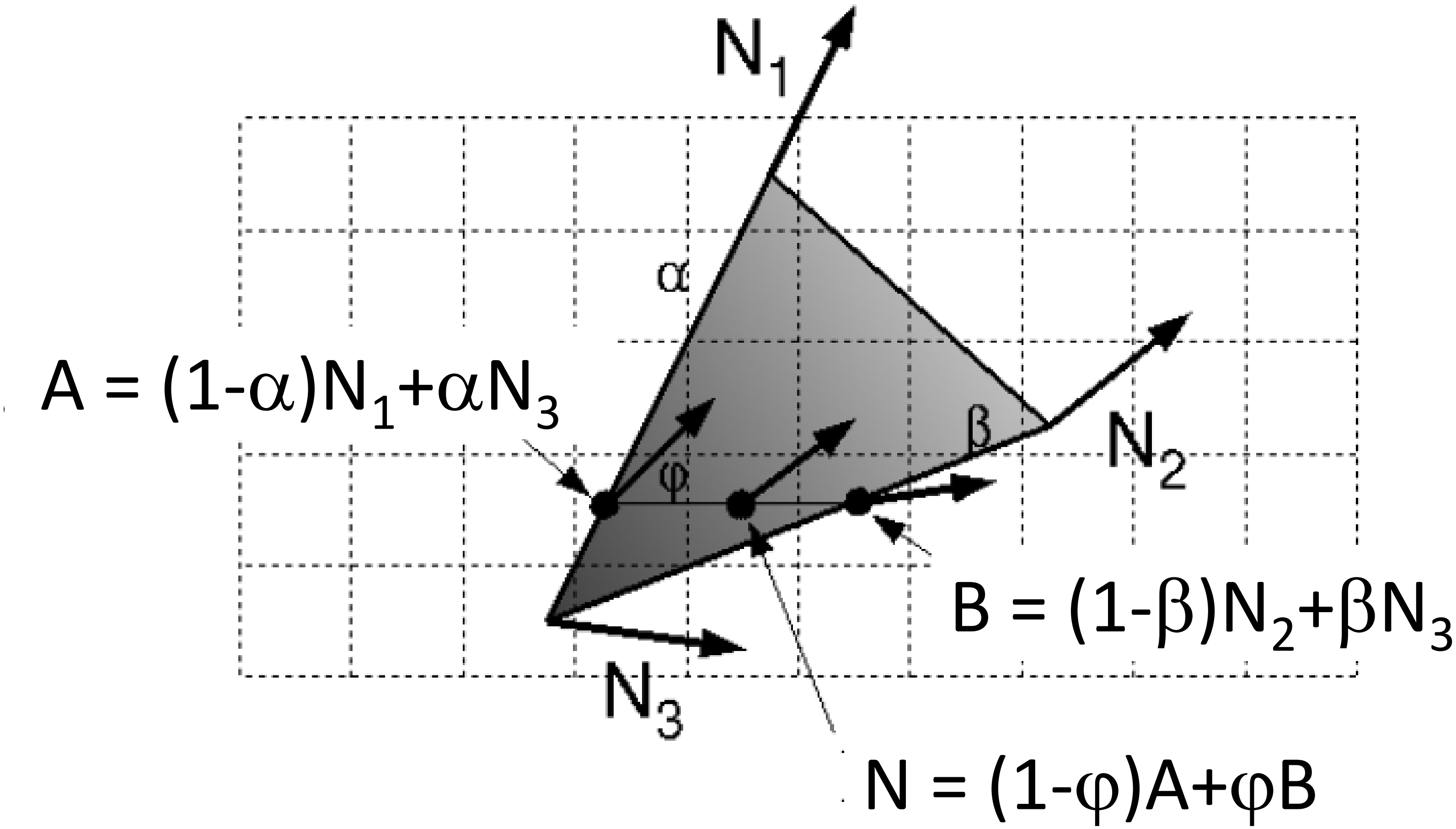
Phong Shading Interpolation

- *Bilinear interpolation* of normals from vertices



Phong Shading Interpolation

- *Bilinear interpolation* of normals from vertices



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

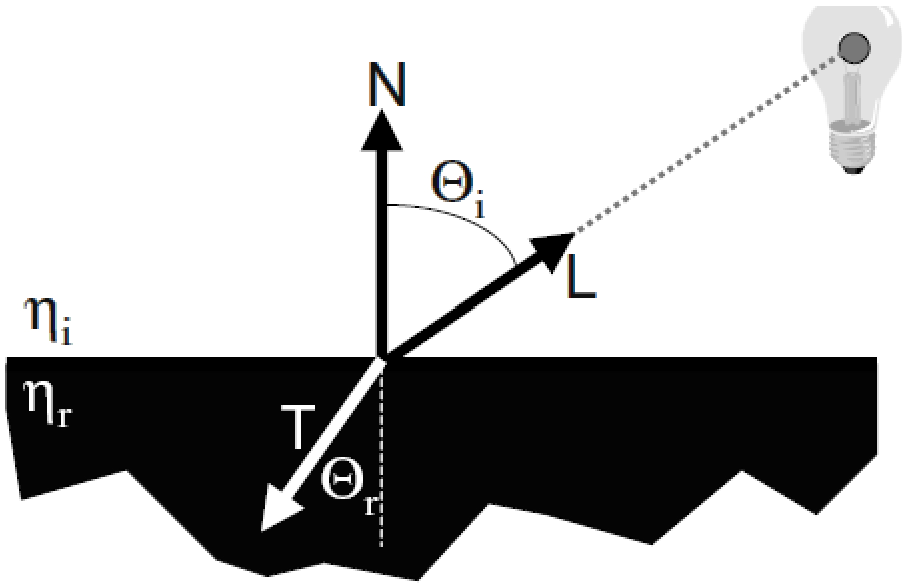
Transparency

- *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.

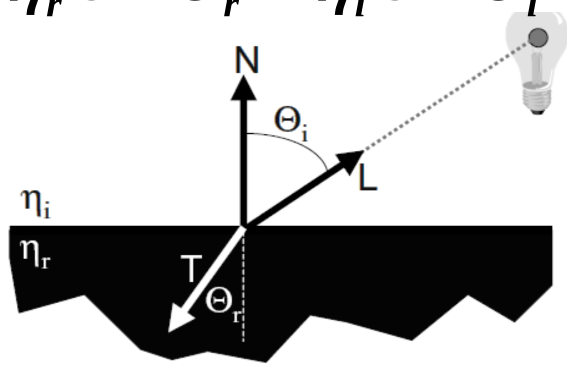
$$\mathbf{T} = -\cos\Theta_r \mathbf{N} - \sin\Theta_r S$$

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

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

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

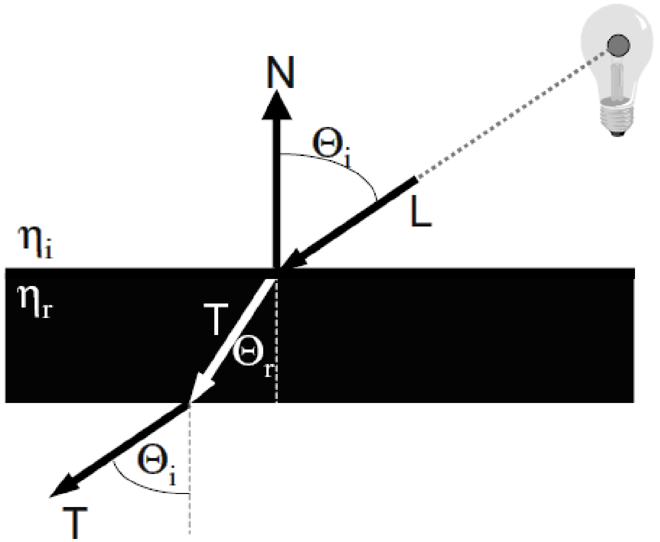
$$\boldsymbol{\eta_r} \sin\Theta_{\boldsymbol{r}} = \boldsymbol{\eta_i} \sin\Theta_{\boldsymbol{i}}$$



$$\mathbf{T} = \left(\frac{\boldsymbol{\eta_i}}{\boldsymbol{\eta_r}}\cos\Theta_{\boldsymbol{i}} - \cos\Theta_{\boldsymbol{r}}\right)\mathbf{N} - \frac{\boldsymbol{\eta_i}}{\boldsymbol{\eta_r}}\mathbf{L}$$

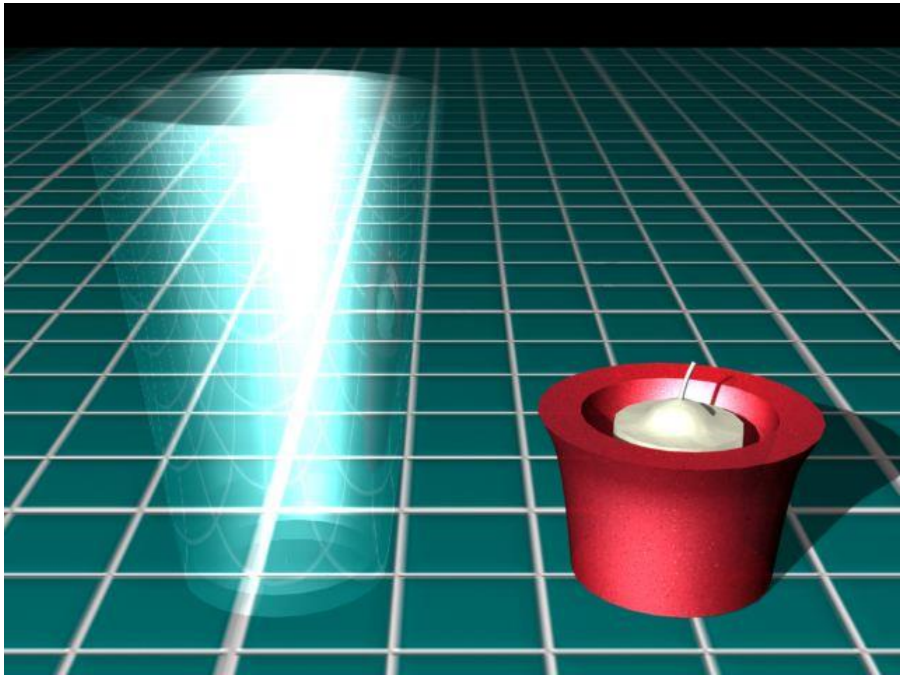
Refraction

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

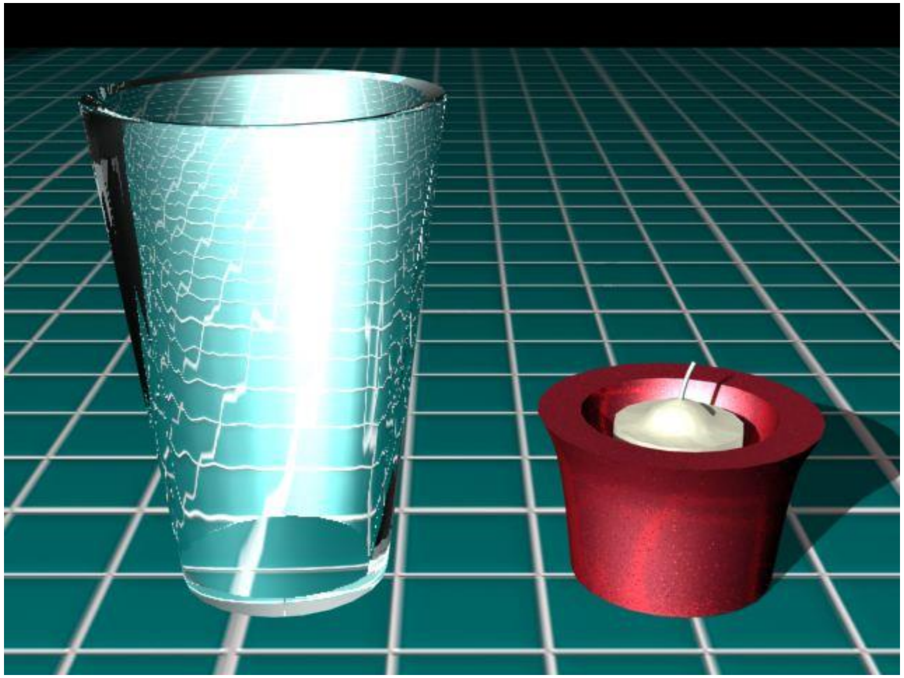


- Usually *ignore refraction*
  - Assume light travels straight through surface (good approximation for thin polygonal surfaces)

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?