Shading

Reading: Angel Ch.6

What is "Shading"?

So far we have built 3D models with polygons and rendered them so that each polygon has a uniform colour:

- results in a 'flat' 2D appearance rather than 3D
- implicitly assumed that the surface is lit such that to the viewer it appears uniform

'**Shading**' gives the surface its 3D appearance

- under natural illumination surfaces give a variation in colour '**shading**' across the surface
- the amount of reflected light varies depends on:
    - the angle between the surface and the viewer
    - angle between the illumination source and surface
    - surface material properties (colour/roughness…)

Shading is essential to generate realistic images of 3D scenes

Realistic Shading

The goal is to render scenes that appear as realistic as photographs of real scenes

This requires simulation of the physical processes of image formation

- accurate modeling of physics results in highly realistic images
- accurate modeling is computationally expensive (not real-time)
- to achieve a real-time graphics pipeline performance we must compromise between physical accuracy and computational cost

To model shading requires:

(1) Model of light source
(2) Model of surface reflection

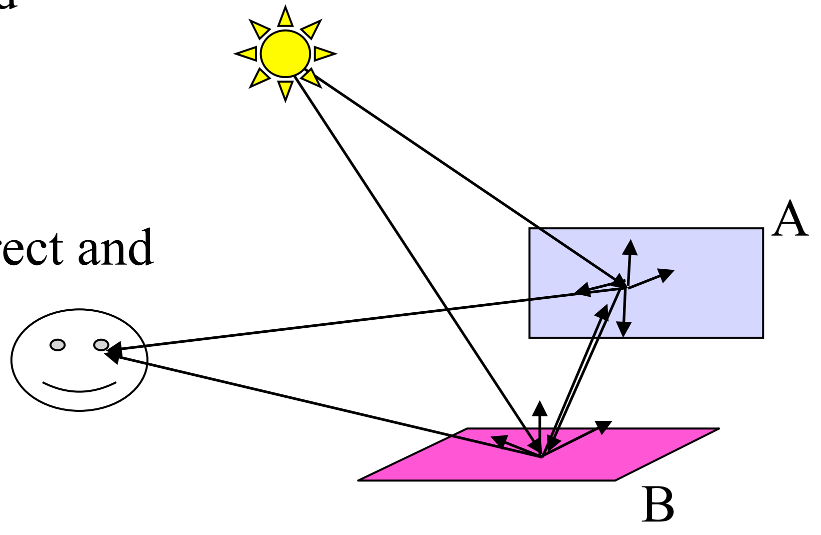
Physically accurate modelling of shading requires a global analysis of the scene and illumination to account for surface reflection between surfaces/shadowing/ transparency etc.

Fast shading calculation considers only local analysis based on:

- material properties/surface geometry/light source position & properties

Physics of Image Formation

Consider the simple scene:
- light rays hit surface A and are scattered according to the material properties
- some of light from A hits B
- some of light from B hits A
- observed colour is a combination of direct and reflected illumination



Recursive process
- Inter-reflection between surfaces causes colour bleeding
  ie a white surface next to a red surface appears red
- The observed colour is a result of multiple interactions among light sources and reflective surfaces

Requires global solution
- integrating all rays of light from all light sources and surface inter-reflections based on material properties

Rendering Equation

Mathematically we can model this process based on the global energy as an integral equation the **'rendering equation'**

- integrates over all light rays in a scene
- compute the surface colour 'shading' at every point in the scene
- can not be solved in general (even numerically)

Various approximations:

- **Ray tracing** - trace rays of light from the viewer to the scene
  - realistic rendering for shiny surfaces
- **Radiosity** - compute inter-reflection between diffuse surfaces
  - realistic for scenes with diffuse (matt) surfaces such a rooms

Both approaches require assumptions which approximate the surface/illumination properties

Both are computationally expensive & cannot be used in a real-time rendering pipeline for computer generated imagery

Focus on a simpler rendering approximation the **'Phong model'**

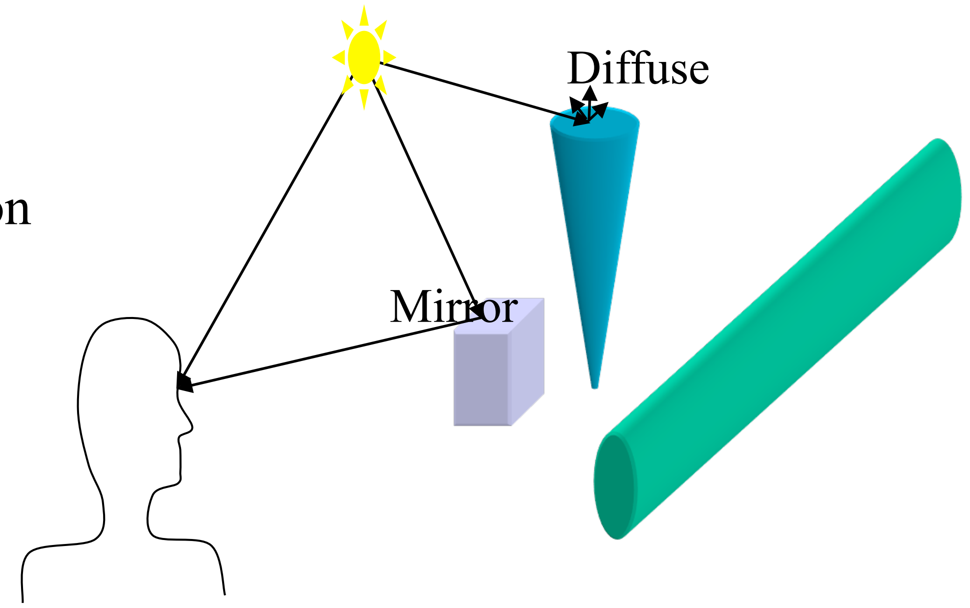
- shading for real-time pipeline graphics

Phong Model

Local computation of surface shading based on
- illumination properties
- material properties
- surface shape

Approach:
- Trace rays of light from the illumination source to the object surface
- Only consider a single interaction between light source & surface
(no inter-reflection between surfaces)

Model:
(1) Light source
(2) Light-surface interaction



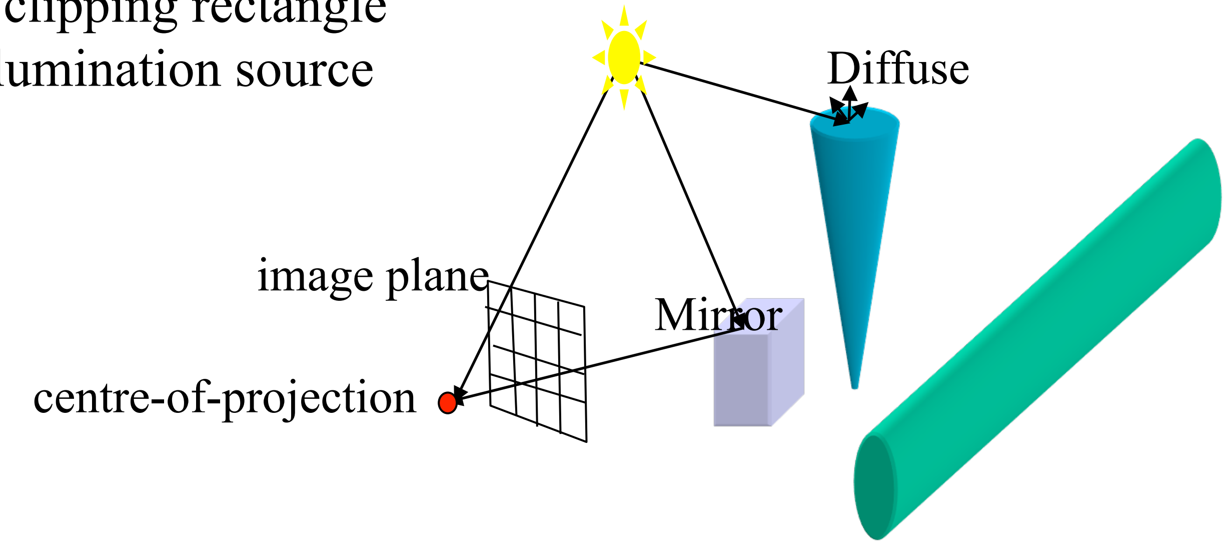
CG Image Plane Approximation

In computer graphics we approximate the viewer by the projection plane
- discretize projection ‘image’ plane as a grid of **‘pixels’**

Problem is to compute the colour observed at each discrete pixel

Consider only rays that leave the source and reach the viewer
- direct from source
- single surface reflection

these are the rays which reach the **centre-of-projection**
after passing through the clipping rectangle
- most rays leaving the illumination source
  do not contribute



Light Sources

Two fundamental processes
- self-emission (due to an internal energy source)
- reflection

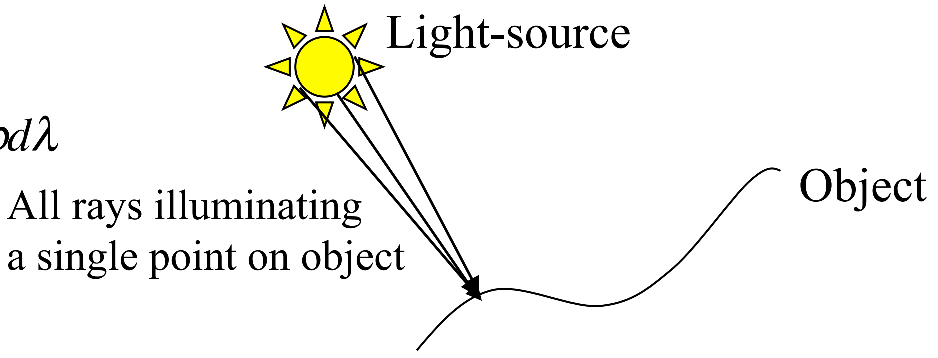
Simple light sources (only self-emission)
- object with a surface
- each point on surface *(x,y,z)* emits light
- light emitted at a point is a function of angle (θ,ϕ) and wavelength λ

General light source illumination function: $I(x,y,z,\theta,\phi,\lambda)$
- six parameter function

Total Illumination = Integral over the surface of the light source for all angles & all wavelenths

$$I_T = \int_x \int_y \int_z \int_\phi \int_\varphi \int_\lambda I(x,y,z,\phi,\varphi,\lambda)\,dx\,dy\,dz\,d\phi\,d\varphi\,d\lambda$$

prohibitively complex/expensive



Light Sources II

For a distributed light source (such as a light bulb) solving the integral of 6 parameters is difficult and computationally expensive

Simplified by:

- approximating distributed light source as a polygonal surface
- approximation by a set of point sources

can approximate a light source of arbitrary complexity at increased computation cost

4 Basic light source types used in CG:

(1) Ambient - equal light in all directions
(2) Point source - light from a single point in all directions
(3) Spot light - point source with limited range of directions
(4) Distant source - all light rays are parallel

- with this combination of lights we can approximate most physical lighting conditions
- consider modeling of each in detail

Light Colour

In addition to light intensity we must model the emitted light colour
- the amount of light emitted at different frequencies varies $I(\lambda)$

In practice we can approximate the colour with a 3 component (r,g,b) colour model
- human vision system is based on three-colour theory which says that we percieve three primary colours (red,green,blue) rather than a full colour distribution
- therefore, we can generate images with a realistic appearance using a 3 colour model (most display devices ie CRT use 3 components)
- each component has an associated distribution over wavelength

Describe the light source colour with a 3 component intensity or '**luminance**' function:

$$I = [I_r, I_g, I_b]$$

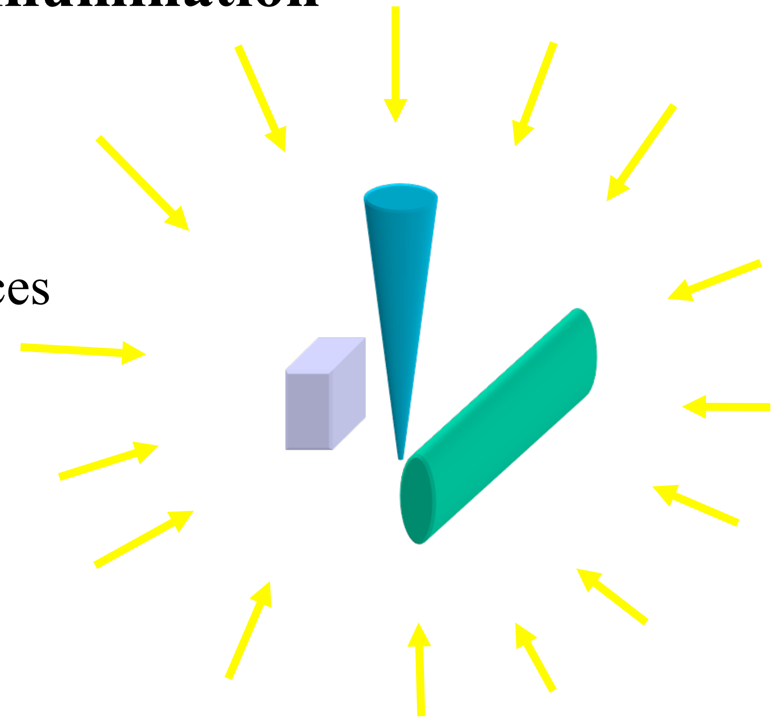
The intensity of the red, green and blue components are assumed to be independent

Treat each colour component independently in computing the surface 'shading' through illumination and surface reflectance

Ambient Illumination

Uniform illumination in all directions
 ie sun on a cloudy day

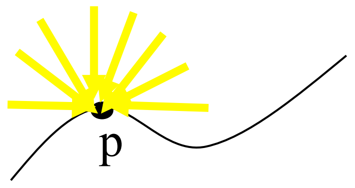
Could model as many distributed sources added together - very expensive



Model by uniform illumination of a surface point by 'ambient light' intensity:

$$I_a = [I_{ar}, I_{ag}, I_{ab}]$$

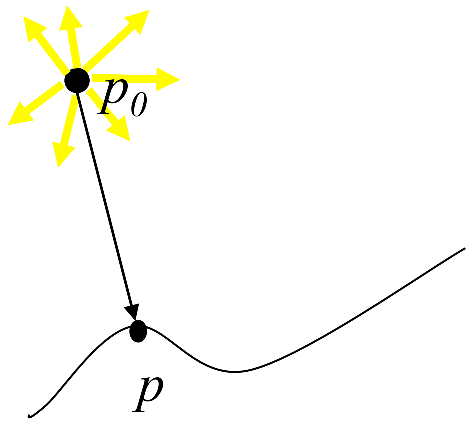
Every point receives the same illumination I_a
- each surface point can reflect this differently according to material properties



Point Source

Emits light equally in all directions
from position p_0

$$I(p_0) = [I_r(p_0), I_g(p_0)\ I_b(p_0)\ ]$$



Intensity of illumination received at surface point p
due to point source at p_0 is given by
inverse square law:

$$\boxed{I(p, p_0) = \frac{1}{\| p - p_0 \|^2} I(p_0)}$$

A point source produces high-contrast scenes: objects appear either bright due to direct illumination or dark due to no illumination (hard shadows)

- In real-scenes light sources have a finite size and produce soft-shadows
- combination of ambient & point source gives soft shadows

In practice we replace inverse square term with an approximation:
a,b,c are constant parameters chosen
to soften illumination

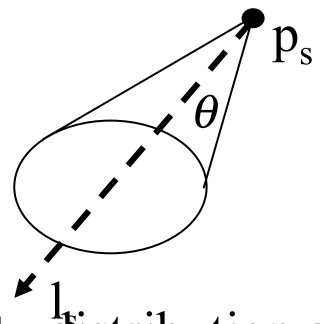
$$\boxed{\begin{aligned} I(p, p_0) &= \frac{1}{(a + bd + cd^2)} I(p_0) \\ d &= \| p - p_0 \| \end{aligned}}$$

for *d* large illumination is constant

Spot Lights

Point source with a narrow range of angles through which light is emitted

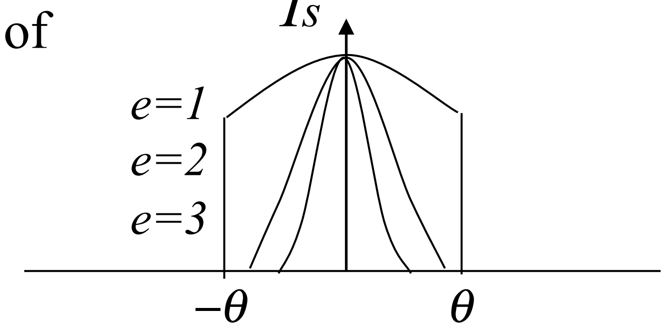
Limit light to a cone whose apex is at p_s and direction l_s and width θ



A more realistic spot light has non-uniform intensity distribution across the cone, this can be modelled as:

$$\boxed{\begin{aligned}&I_S(p,p_S)=(\cos\phi)^e I_p(p,p_0)=(s\bullet l_s)^e I_p(p,p_0)\\&-\theta<\phi<\theta\\&s=\frac{(p-p_S)}{\|p-p_S\|}\end{aligned}}$$

the exponent e determines the rate of drop off



Distant Light Sources

Light rays are parallel

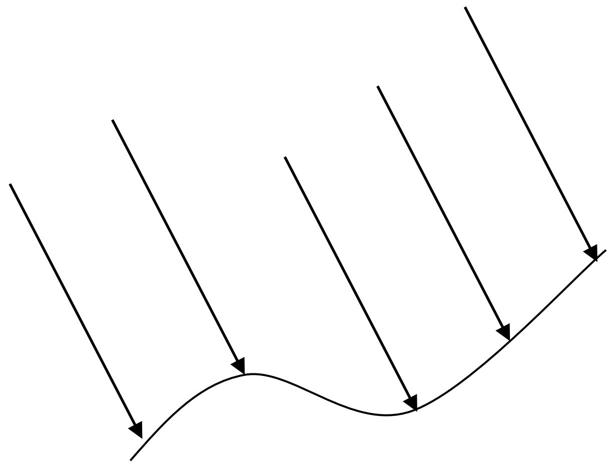
- all rays have the same direction so we don't need to recompute the direction vector (p-p_0) for each surface point
- calculation for parallel light sources is analagous to parallel projection

In homogenous coordinates for parallel projection we can represent a distance light source as a vector:

$$p_0 = [x,y,z,0]$$

rather than a point as used for a point source

Gives more efficient computation in the graphics pipeline



Light-Surface Interaction

When light strikes a surface some is absorbed & some reflected

- light-surface interaction depends on material properties (colour/roughness)
- is a function of wavelength
- shading also depends on the surface orientation relative to light source & viewer

Light-Surface interaction can be classified into three categories:

I) **Specular** - most of the light is reflected along a single axis at the reflection angle ie shiny surface

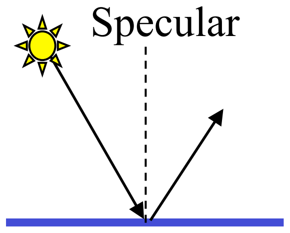
- a mirror is a perfect specular surface

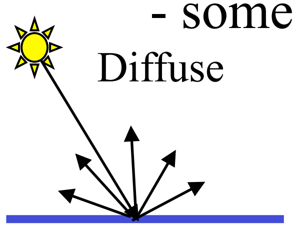
II) **Diffuse** - light is scattered in all directions ie matt surface

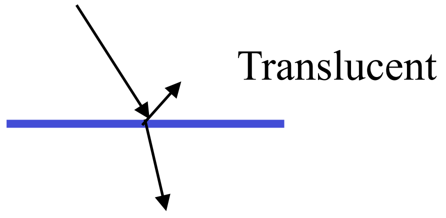
- a perfect diffuse surface scatters light equally in all directions & appears the same from all directions

III) **Translucent** - Some light penetrates the surface to emerge elsewhere through refraction ie glass/water

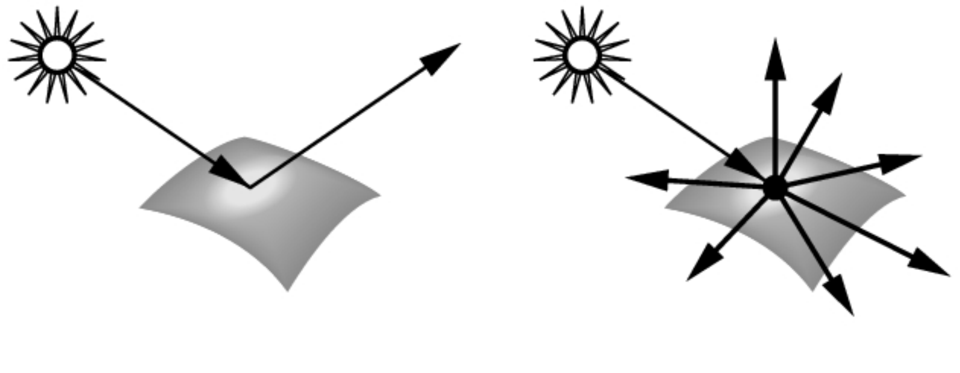
- some incident light may also be reflected







Light-material interaction



3 surface types:

(a) Specular - appear shiny
- light scattered in a narrow range around reflection angle
- mirror is perfectly specular

(b) Diffuse - matte
- light scattered in all directions
- perfect diffuse scatters equally in all directions

(c) Translucent - transparent surfaces (glass,water)
- light penetrates the surfaces is refracted and emerges from from another location + partial reflection at the surface

Phong Reflection Model

Only consider light rays which

(1) Enter camera directly

(2) Are reflected once off an object surface and enter camera

- gives efficient pipeline rendering
- local model - evaluated independently for each surface point
- close enough approximation of physics to give ‘good’ renderings for a variety of lighting conditions and material properties
- Introduced by Phong 1975

The Phong model supports 3 types of light-surface interaction:

(1) Ambient

(2) Diffuse

(3) Specular

For each light source we have corresponding ambient, I_a, diffuse, I_d, and specular, I_s, intensity components

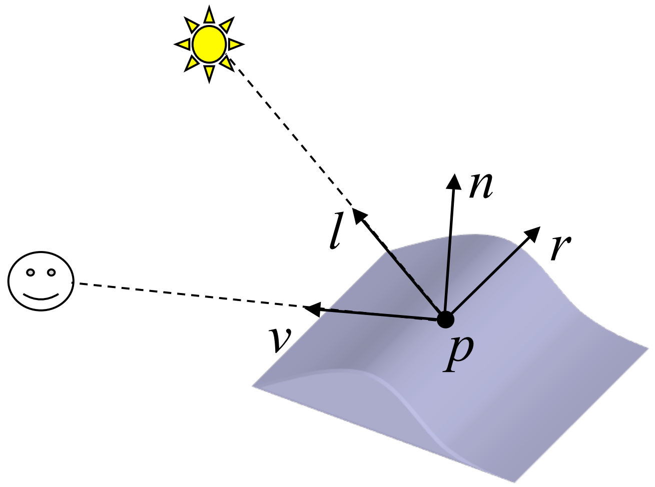
Each intensity component has 3 colour components *(r,g,b)* which are treated independently

Phong Reflection Model II

Model uses 4 unit vectors to calculate the observed colour ‘shading’ at a point on the surface:

(1) *n* - surface normal
(2) *v* - view direction from centre-of-projection
(3) *l* - light direction to a point on the light source (arbitrary point)
(4) *r* - reflection direction of a perfectly reflected ray

Note: r is a function of l and n



Phong III

Relationship between r and n,l

2 constraints on r:

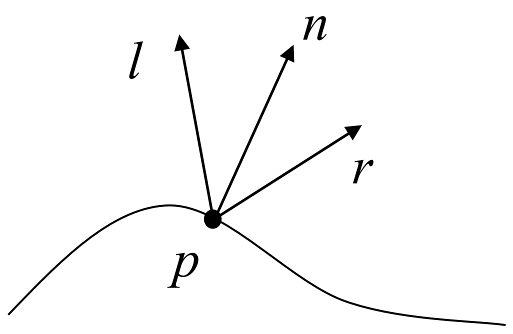
(i) angle of incidence = angle of reflection

$$n.l = n.r$$

(ii) r is in the plane defined by l and n

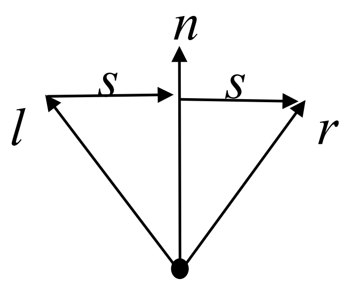
$$r = an + bl$$

a,b are scalars



The vector s from l perpendicular to n is given by:

$$s = n(n.l) - l$$



Then r is given by:

$$r = l+2s$$
$$= 2n(n.l) - l$$

see Angel 6.4.2 for an alternative proof

Phong IV

In the phong model we consider 3 intensity components for each light source

- Ambient I_a, Diffuse, I_d and Specular I_s

For each intensity component we have 3 colours ie $I_a = [I_{ar}, I_{ag}, I_{ab}]$

- each colour/intensity component is treated independently

Phong uses a local model to approximate global illumination effects:

At any point p on the surface we can compute
a 3x3 **illumination matrix** for the i^{th} light source:

$$L_i = \begin{bmatrix} L_{ira} & L_{iga} & L_{iba} \\ L_{ird} & L_{igd} & L_{ibd} \\ L_{irs} & L_{igs} & L_{ibs} \end{bmatrix}$$

where L_i is the matrix of illumination intensities
for each light source & colour at point p
ie L_{ira} is the ambient red intensity component for the i^{th} light source

Note: distance attenuation terms have not yet been included

Phong V

Construct the reflection model by assuming that we can compute **how much of the incident light is reflected** at each point of interest

ie for red diffuse term L_{ird} reflection is R_{ird}

where the contribution to the reflected light intensity is $I_{ird} = R_{ird} L_{ird}$

The reflection term depends on:

(1) material properties

(2) surface orientation

(3) light source direction

(4) distance of light to viewer

For each point we compute a matrix of reflection coefficients: $R_i = \begin{bmatrix} R_{ira} & R_{iga} & R_{iba} \\ R_{ird} & R_{igd} & R_{ibd} \\ R_{irs} & R_{igs} & R_{ibs} \end{bmatrix}$

We compute the contribution to the reflected light intensity for each colour source by summing the components for ambient, diffuse and specular:

$$I_{ir} = R_{ira}L_{ira} + R_{ird}L_{ird} + R_{irs}L_{irs} = I_{ira} + I_{ird} + I_{irs}$$

Phong VI

The **total reflected light intensity** is obtained by summing contributions over all light sources independently for each colour component and adding a **global ambient term**:

$$\boxed{I_r = \sum_i (I_{ira} + I_{ird} + I_{irs}) + I_{ar}}$$

Note: the computation of the total reflected light intensity is the same for each colour component

In vector-matrix form we have:

$$I = \begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix} = \sum_i (I_{ia} + I_{id} + I_{is}) + I_a = L_i^T R_i + I_a$$

where *I* are 3x1 vectors of intensity components and *R,L* are 3x3 matrices of illumination intensity and material reflection coefficients.

Phong reflection model gives total reflected light intensity as a linear sum of reflected component intensities for each intensity & colour component

Phong VII - Ambient Reflection

Intensity of ambient light illumination L_a is the same at all points on the surface
- some of light is absorbed & remaining is reflected
- reflected ambient intensity is the same in all directions as illumination is the same in all directions

Proportion of light reflected is specified by the **'ambient light coefficient'** k_a:

$$R_a = k_a$$
$$0 \le k_a \le 1$$
$$I_a = k_a L_a$$

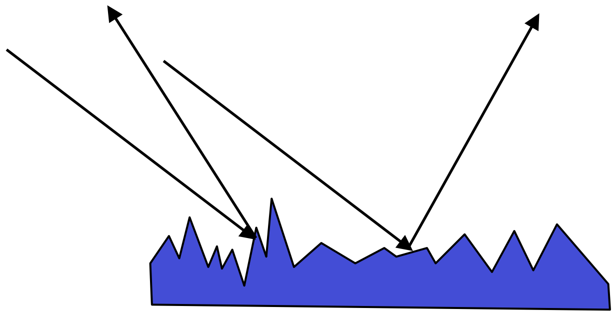
L_a can be the ambient illumination due to either individual sources or the global ambient illumination.

Note: the surface has different ambient coefficient $[k_{ar}\, k_{ag}\, k_{ab}]$
for each colour component

Phong VIII - Diffuse Reflection

Scatters light in all directions

- surface appears the same to viewers from all directions
- amount of light reflected depends on material & position of light relative to surface
- corresponds to a **rough surface**: a small difference in the angle of incidence between rays can result in very different reflection angles



Perfectly diffuse surface **'Lambertian'** surfaces have no preferred direction and are modelled by Lambert's law:

$$\boxed{R_d \propto \cos\theta}$$

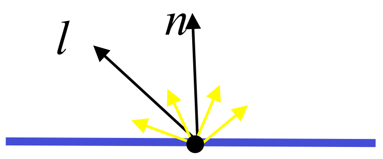
Where θ is the angle of the incident ray to the normal

The proportion of reflected light is given by the **diffuse light coefficient** k_d:

$$\boxed{R_d = k_d \cos\theta = k_d (l \bullet n)}$$

The resulting reflection intensity in all directions given a point source at p_0 is:

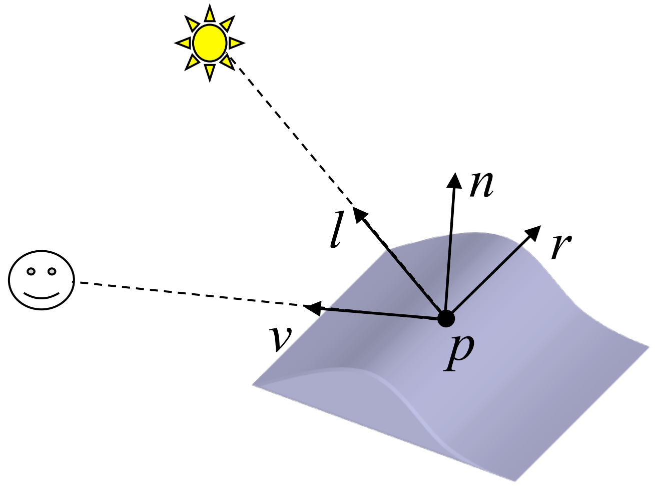
$$\boxed{\begin{aligned} I_d &= \frac{1}{(a + bd + cd^2)} k_d (l \bullet n) L_d \\ d &= \| p - p_0 \| \end{aligned}}$$



Phong IX - Specular Reflection

Reflect incident light along a single axis

- *produce surface highlights*
- *specular surfaces are smooth*
- *perfect specular surface is a mirror*
- *real-specular surfaces are asymetric and reflect light* along a range of angles



Phong proposed an approximate model

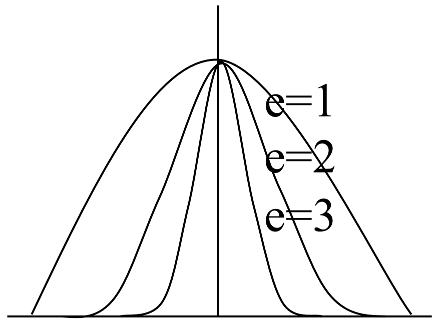
- approximate the specular surface as smooth
- computational cost slightly greater than diffuse

Intensity of light that the viewer sees depends on the angle between the reflection axis *r* and the viewer *v*:

$$I_s = \frac{1}{(a+bd+cd^2)} k_s L_s \cos^e \phi = \frac{1}{(a+bd+cd^2)} k_s L_s (r.v)^e$$

$$d = \| p - p_0 \|$$

$$0 \le k_s \le 1$$



k_s is the **specular light coefficient,** *e* is a "shininess"coefficient

Phong Model

The reflected surface intensity at a point p on the surface due to a point source at p_0 with ambient I_a, diffuse I_d and specular components is computed independently for each colour component as:

$$\boxed{\begin{array}{l} I_i = \dfrac{1}{(a+bd+cd^2)}(k_d L_{id}(l.n) + k_s L_{is}(r.v)^e) + k_{ia} L_a \\ d = \| p - p_0 \| \end{array}}$$

Total reflection intensity is computed for each colour component (r,g,b) by adding the contributions over all light sources

Allowing different specular, diffuse and ambient components for each light source enables realistic lighting with a wide variety of surface effects to be simulated without modeling the full global rendering equations

The Phong model does not model inter-reflections between surfaces

- simulated by coloured ambient lights

Implementation of Phong model

The above analysis considered the Phong model in object space

In practice for a pipeline rendering system the “shading” is done after the object is projected in the image plane space

$$\begin{tikzcd}
\begin{matrix}\text{vertices}\\ \text{normals}\end{matrix} \arrow[r] & \boxed{\text{Model-view}} \arrow[r] & \boxed{\text{Projection}} \arrow[r] & \boxed{\text{“Shading”}} \arrow[r] & \text{image}
\end{tikzcd}$$

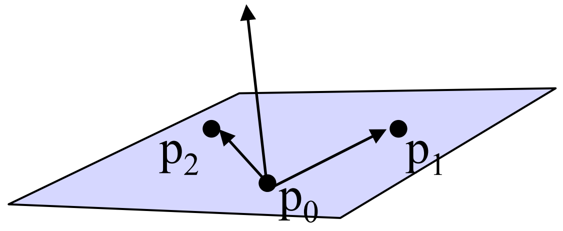
As the projective transformation changes the angles we ensure the correct angle cosines are computed by propagating vectors through the pipeline

Surface Normal Computation

The Phong model provides a general solution for shading computation
- in special cases (flat surface/distant objects) the computation simplified
- in practice we use polygonal faces to represent 3D object surfaces therefore we can simplify computation for polygonal facets

For a planar surface such as a polygon we can compute the surface normal from a 3 non-colinear points:

$$n = (p_2 - p_0) \times (p_1 - p_0)$$

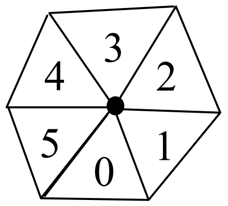


Note: the order is important
convention to define polygon vertices in ccw order

Given a polygonal mesh for normal computation we may require computation of the normals at the vertices rather than for the faces.
A vertex normal can be computed by averaging the face normals around the vertex:

$$n_v = \sum_{i=0}^{N_{face}-1} w_i n_i$$

$$\sum_i w_i = 1$$



where w_i are weights based on the area or angle of each polygon around the vertex

Polygonal Shading

Given: (1) A set of light sources
(2) A viewer (location,direction,projection)
(3) Object material properties
(4) Surface normals

Apply lighting and illumination models for every point on the surface
- in general computation for arbitrary surfaces is expensive due to cost of computing surface normals

Polygonal meshes can approximate arbitrary complex surface with planar faces giving fast normal computation
- for flat polygons we can significantly reduce the shading computation

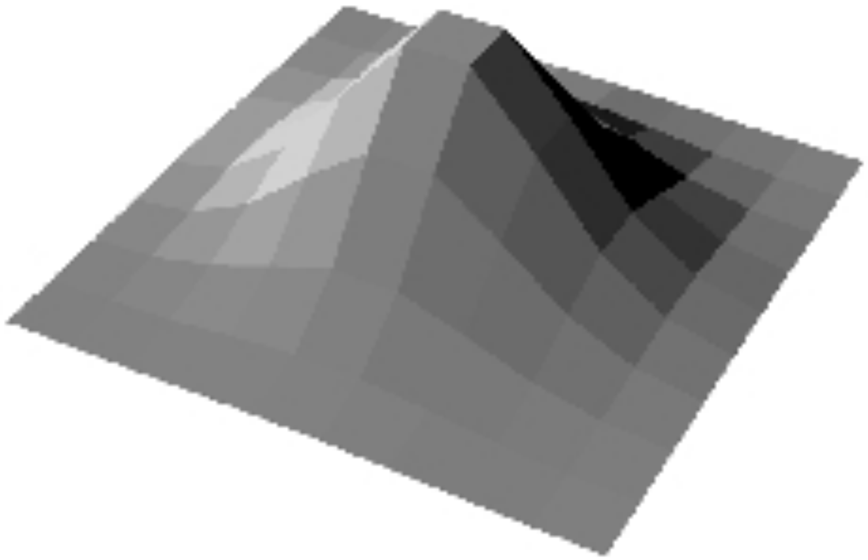
Consider 3 common ways of shading polygons:
(1) Flat
(2) Gouraud (or interpolative)
(3) Phong

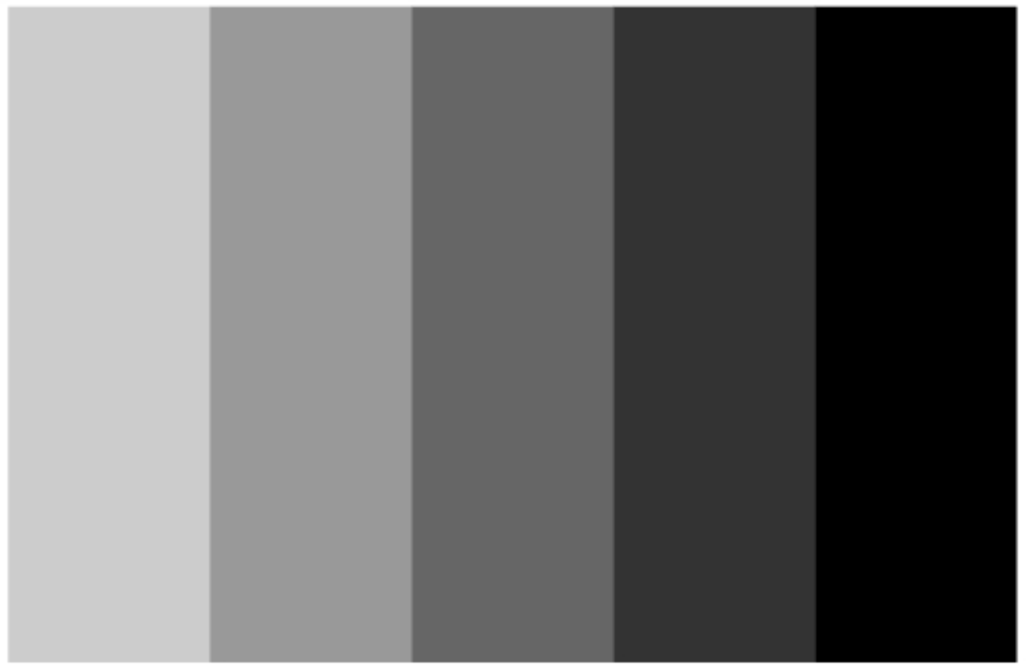
Flat Shading

For flat shading the normal *n* is constant for each polygon face
- assuming the viewer is distant viewpoint vector *v* is also constant
- assuming the light source is distant light vector *l* is also constant

Given these assumption shading computation only needs to be carried out once per polygon
- each point on the polygon has the same colour

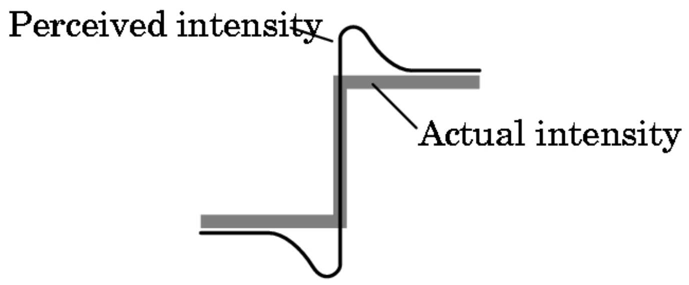
In practice if assumptions are not valid flat shading gives poor results
- polygonal facets clearly visible
- human perception highlights this due to 'lateral inhibition'





Lateral Inhibition

- human visual response to small difference in intensities accentuates the difference
- For sequence of intensities we perceive the brightness to overshoot on one side and undershoot on other
- results in stripes known as 'Mach bands'
- caused by how cones in the eye connect to the optic nerve



Gouraud Shading

Interpolates vertex colour/shading across the polygon face

- introduced by Gouraud 1971 ‘interpolated shading’

For a polygon:

(1) Compute shading colour at each vertex

- lighting calculation from vertex position/normals, material properties, view direction and light direction

(2) Interpolate vertex colour to all points on polygonal face

Vertex normals are defined as the average adjacent face normal (with equal weights)

$$n_v = \frac{1}{N_{face}} \sum_{i=0}^{N_{face}-1} n_i$$

Colour for any point on the triangle is computed via linear interpolation

$$c(\alpha, \beta) = \alpha c_1 + \beta c_2 + (1 - \alpha - \beta) c_3$$

$$\alpha, \beta \in [0,1]$$

Result is a smooth appearance

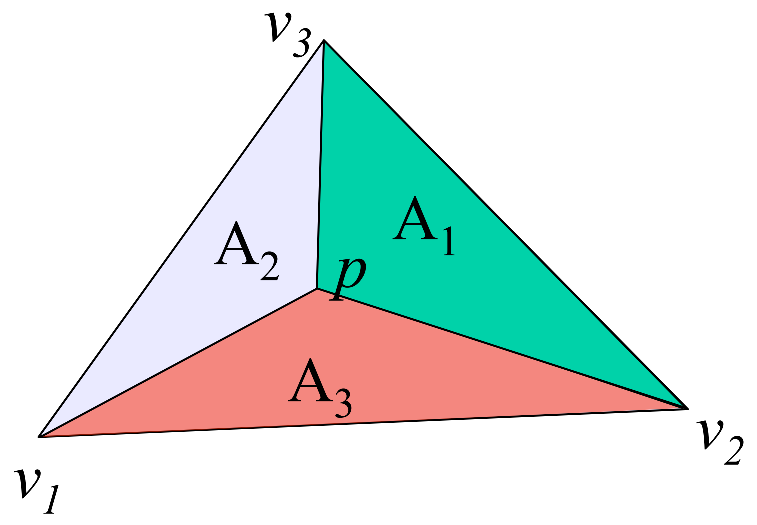
- without any apparent step discontinuities in the shading
- works well provided polygon size is small relative to view distance

Barycentric Coordinates

$$A = \frac{1}{2}\left\|(v_2 - v_1)\times(v_3 - v_1)\right\|$$

$$A_1 = \frac{1}{2}\left\|(v_2 - p)\times(v_3 - p)\right\|$$

$$A_2 = \frac{1}{2}\left\|(v_3 - v_1)\times(p - v_1)\right\|$$



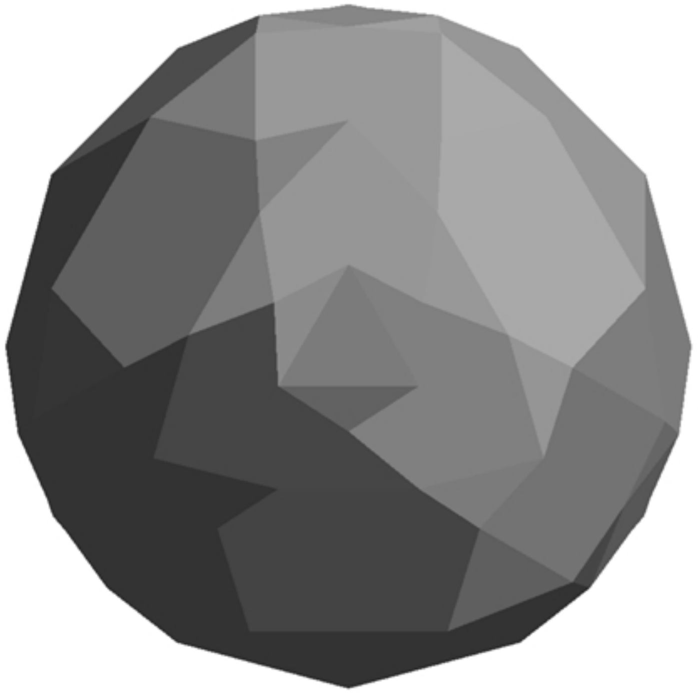
$$\alpha = \frac{A_1}{A}$$

$$\beta = \frac{A_2}{A}$$

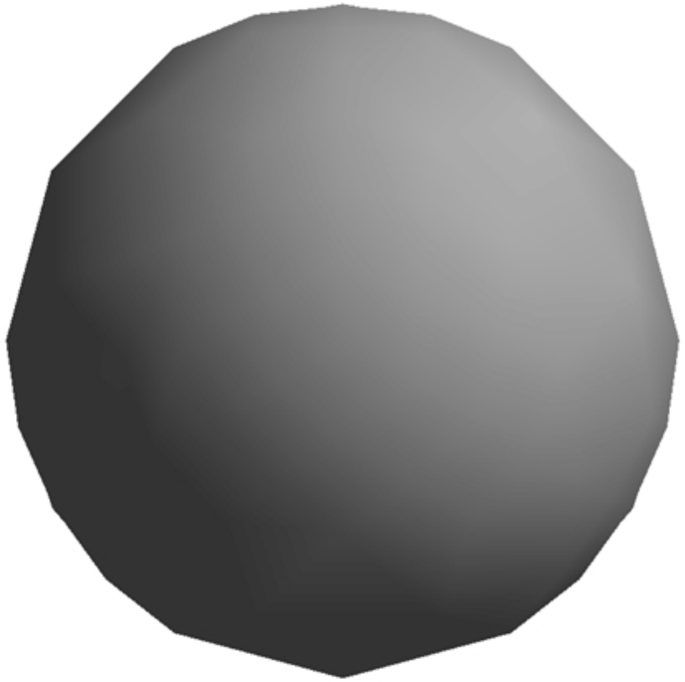
$$p(\alpha, \beta) = \alpha v_1 + \beta v_2 + (1 - \alpha - \beta) v_3$$

$$c(\alpha, \beta) = \alpha c_1 + \beta c_2 + (1 - \alpha - \beta) c_3$$

Flat Shaded



Goraud Shaded

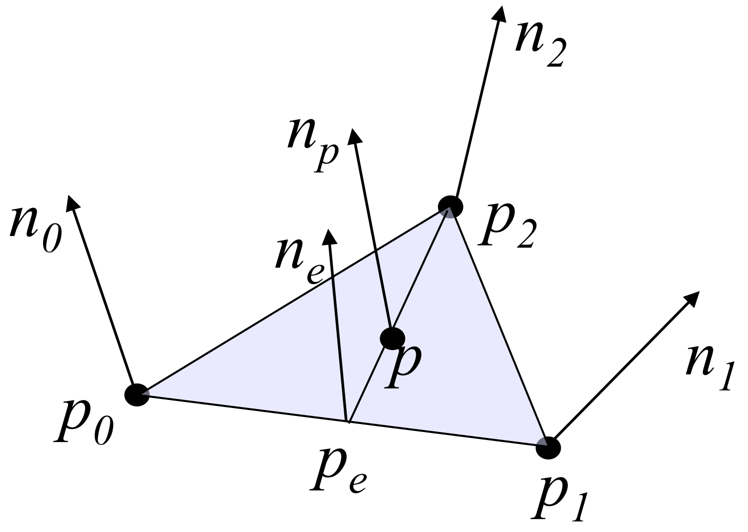


Phong Shading

Interpolate the vertex normals across the polygon

(1) Interpolate along edge(p_0,p_1):

$$p_e(\alpha) = \alpha p_0 + (1-\alpha) p_1$$
$$n_e(\alpha) = \alpha n_0 + (1-\alpha) n_1$$
$$0 \le \alpha \le 1$$



(2) Interpolate along line (p_e,p_2):

$$p(\alpha,\beta) = \beta(\alpha p_0 + (1-\alpha) p_1) + (1-\beta) p_2$$
$$n_p(\alpha,\beta) = \beta(\alpha n_0 + (1-\alpha) n_1) + (1-\beta) n_2$$
$$0 \le \alpha,\beta \le 1$$

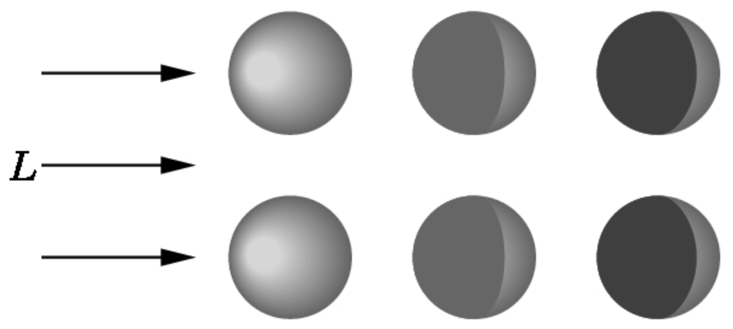
this allows us to compute a continuous **linearly interpolated** normal for any point inside the polygon

- (α,β) can be computed as part of polygon rasterisation ‘scan conversion’
- produces slightly smoother rendering than Gouraud
- much higher computation cost than Gouraud
- Gouraud usually supported in graphics pipeline hardware (real-time) vs. Phong usually implemented in software (off-line)

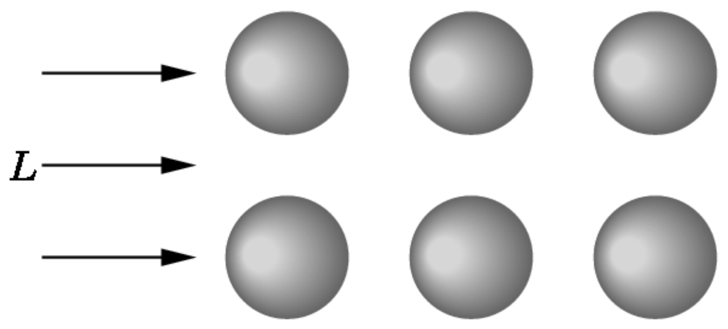
Global Rendering

Phong model uses local lighting computation and no inter-reflection

- this is limited in realism compared to real illumination which is a global process
- no shadows
- no inter-reflection



Global



Local

To achieve highly realistic ‘film quality’ images requires a global approximation
2 solutions proposed:

- Ray Tracing - highly specular surfaces ie reflective/translucent objects
- Radiosity - diffuse illumination ie building interiors

Here we outline briefly these approaches

Ray-Tracing

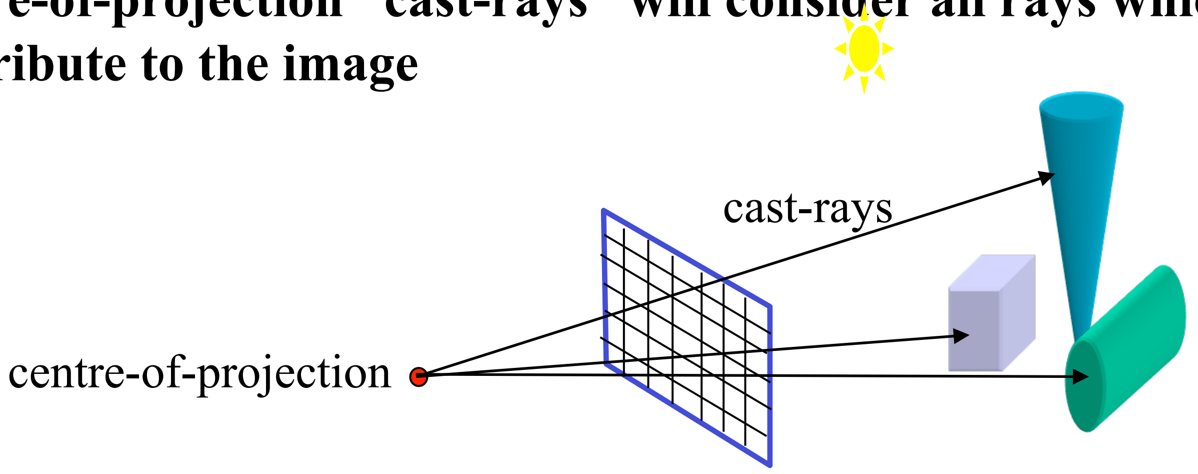
- Appel 1968
- Extension of local light model

Based on the observation that of all the rays of light leaving a source the only ones which contribute to the image are those which enter the camera inside the view frustum and pass through the centre-of-projection

- direct from source
- single surface reflection

+ multiple surface reflections
+ transmission (refraction) through one or more surfaces

Therefore, **reversing the direction of rays and tracing rays from the centre-of-projection ‘cast-rays’ will consider all rays which contribute to the image**



Ray-Tracing II

Cast at least one ray per pixel

(i) intersects a surface or light source

- generate a new '**shadow ray**' from point on surface to each light source
- if shadow ray intersect another surface before light source then point is not lit by source 'shadowed'
- if surfaces are highly specular we can compute new shadow rays as ray bounces from surface-to-surface until it goes off to infinity or hits a light source

(ii) goes off to infinity without hitting anything

- background colour

Generation of shadow rays for specular surface is a **recursive process**

Principal cost of ray-tracing is **hidden-surface visibility** calculation for testing intersection of shadow rays with surfaces

Ray-tracing is very good at handling surfaces which both reflect and transmit

- for a cast ray intersecting a surface we use our reflection model to compute the light reflected (specular & diffuse) and absorbed

Ray-Tracing III

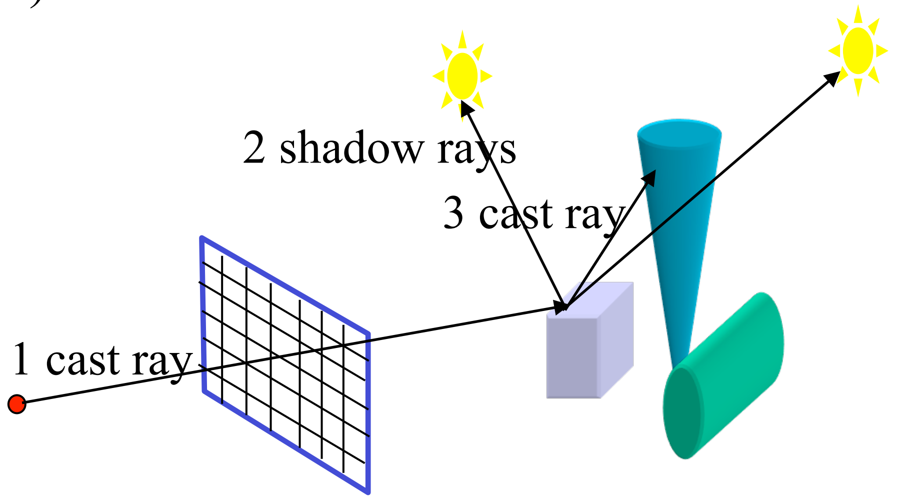
'Cast-rays' hits surface at point *p*:

If a light source is visible (through **'shadow-rays'**)

(i) computer contribution from light source to illumination at *p*

(ii) **cast a new ray** in direction of perfect reflection using reflection model for specular component

(iii) **cast a new ray** in direction of transmitted ray if surface is translucent (glass/water)



Ray-tracing is good for shiny/specular surfaces simulates:

- shadows/inter-reflection/transmission
- poor for diffuse reflection term (rays cast in all directions)

lemog@club-internet.fr
...just A Mouse

Radiosity

Based on numerical methods for solving heat transfer Siegel' 81
Introduced to graphics Goral' 84

Assuming all surfaces are **diffuse** we can obtain a global solution as a global energy balance with diffuse surface-surface interaction.

- Rendering equation for diffuse surface reduces to a simple equation which can be solved numerically

Once we have solved equations we can render scene using any renderer with flat shading from any view

- calculation of form-factors is $O(n^2)$ for n patches
- rendering is as fast as using Phong local lighting model

Radiosity

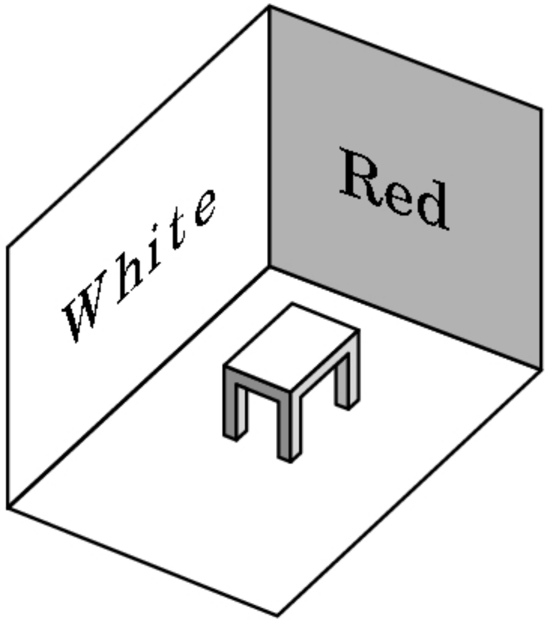
Solution:

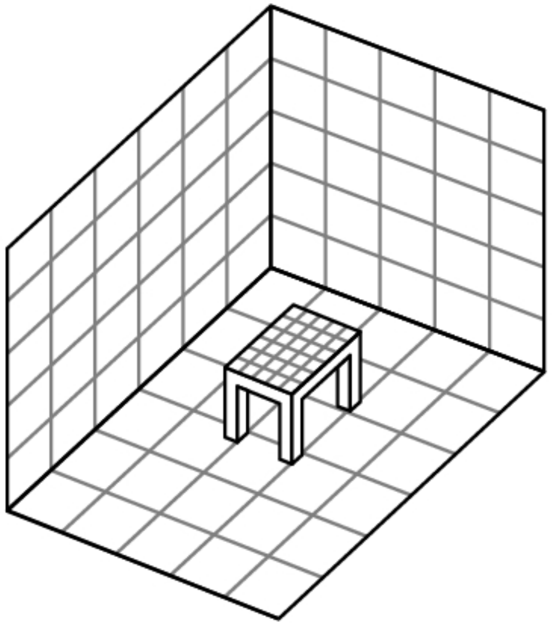
(i) Decompose scene into small flat polygons

- Assume each polygon is perfectly diffuse and renders in a constant shade

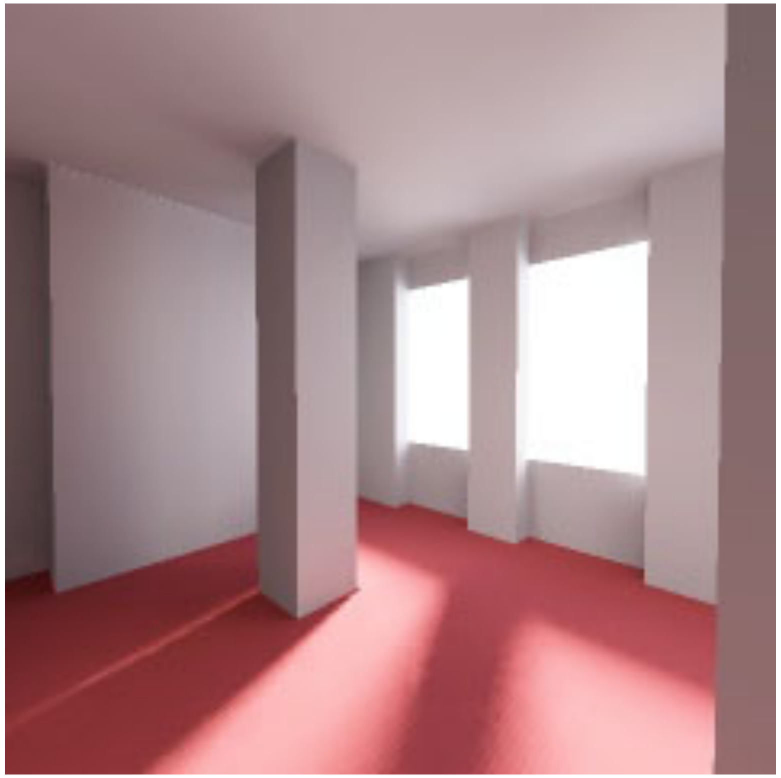
(ii) Consider patches pairwise to compute '**form-factors**' that describe how light leaving one patch can affect the other

(iii) Given the form factors rendering equations reduce to a set of linear equations for **radiosities** ('reflectivity' of the polygons)





Example - Radiosity



Summary

- Reflectance
  - Specular/diffuse/translucent surfaces

- Lighting
  - Ambient/point/distant

- Phong reflection model

- Polygon Shading
  - flat/Goraud/Phong
  - Barycentric coordinates

- Global illumination methods
  - ray tracing
  - radiosity