UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 4370 Fall 2023

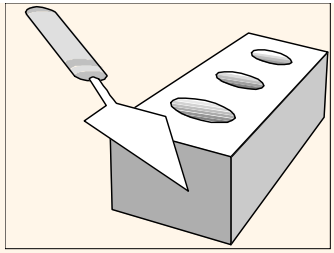
Interactive Computer Graphics

M & W 5:30 to 7:00 PM

Prof. Victoria Hilford

PLEASE TURN your webcam ON

NO CHATTING during LECTURE



COSC 4370

5:30 to 7

PLEASE

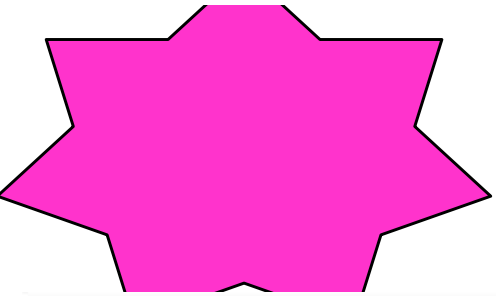
LOG IN

CANVAS

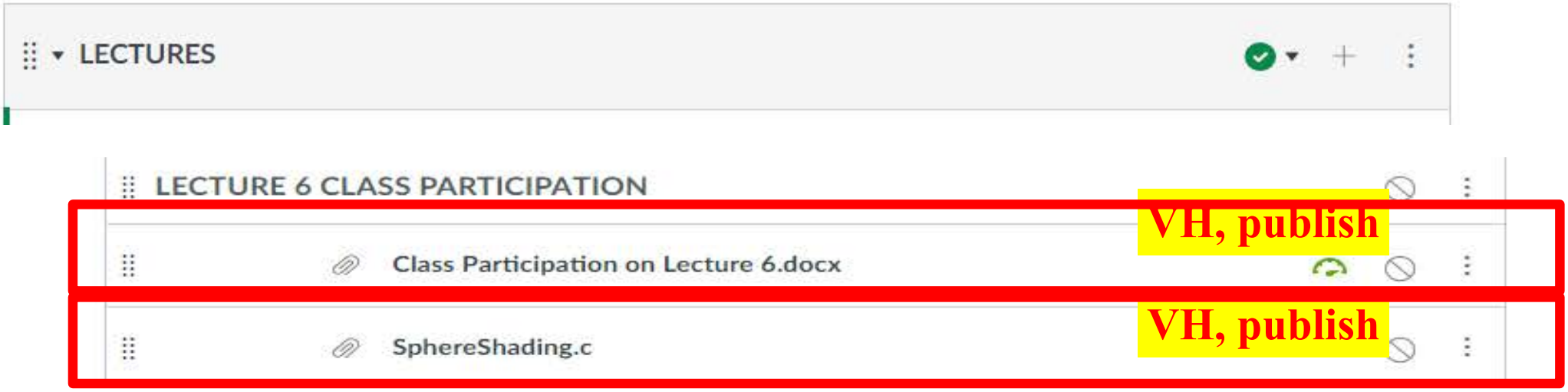
Please close all other windows.

NEXT.

10.02.2023 (M 5:30 to 7) (12)	Homework 5	Lecture 6
10.04.2023 (W 5:30 to 7) (13)	Homework 6	Lecture 7
10.09.2023 (M 5:30 to 7) (14)		PROJECT 2
10.11.2023 (W 5:30 to 7) (15)		EXAM 2 REVIEW
10.16.2023 (M 5:30 to 7) (16)		EXAM 2




CLASS PARTICIPATION - 15%
15% of Total
Class APRTICIPATION on Lecture 6
Not available until Oct 2 at 5:30pm | Due Oct 2 at 7pm | 100 pts
VH, publish


LECTURES
LECTURE 6 CLASS PARTICIPATION
Class Participation on Lecture 6.docx
VH, publish
SphereShading.c
VH, publish

Class APRTICIPATION on Lecture 6

Publish | Edit | ⋮

Download and complete this word document.

Class Participation on Lecture 6.doc

SphereShading.c

You will be prompted when to Upload completed document to CANVAS as **score.doc** (example 100.doc).

Warning:

TA, at random, will inspect the Uploaded document.

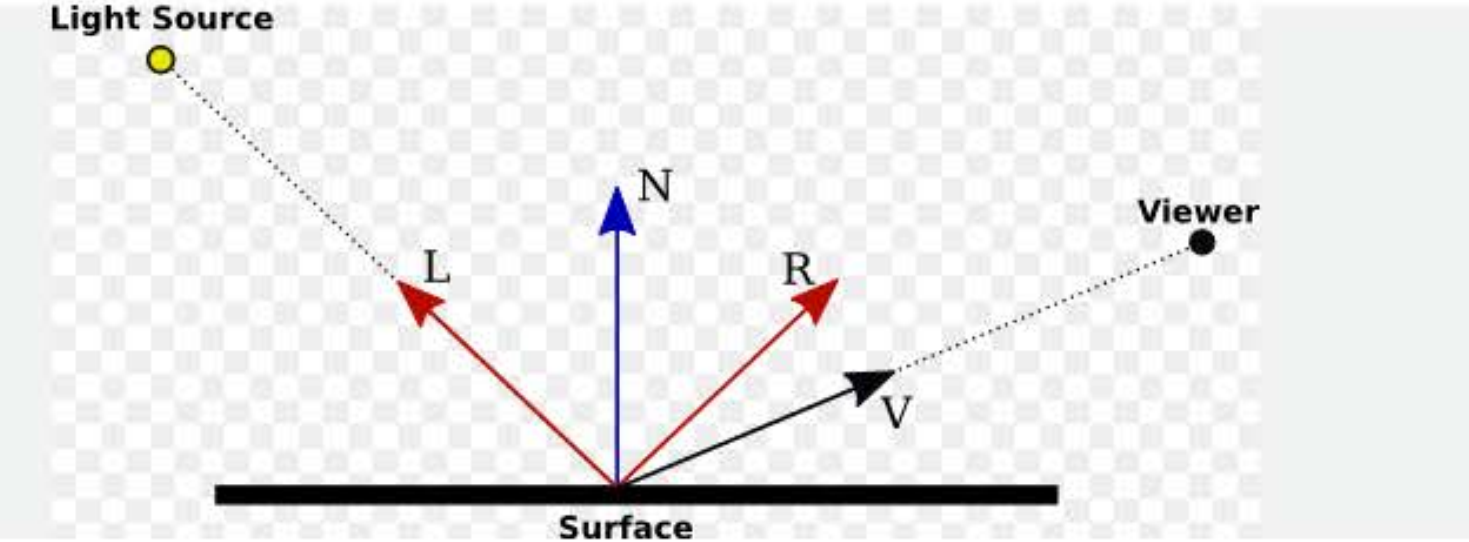
If you score is not honestly entered, you will get a zero.

For the Grader: if the student did not submit, please skip and do not assign a ZERO (MISSING).

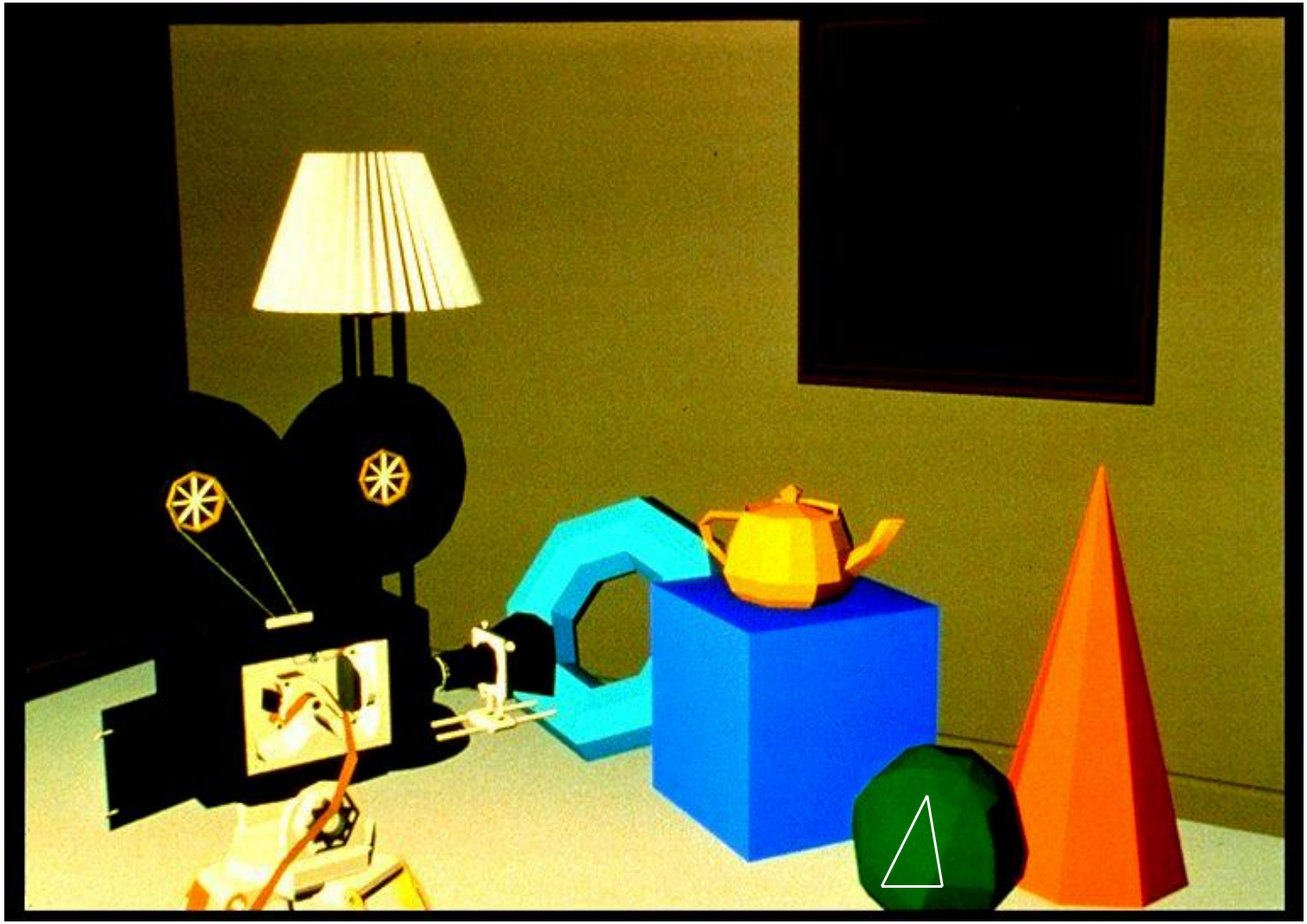
COSC 4370 – Computer Graphics

Lecture 6

Lighting and Shading
Chapter 6

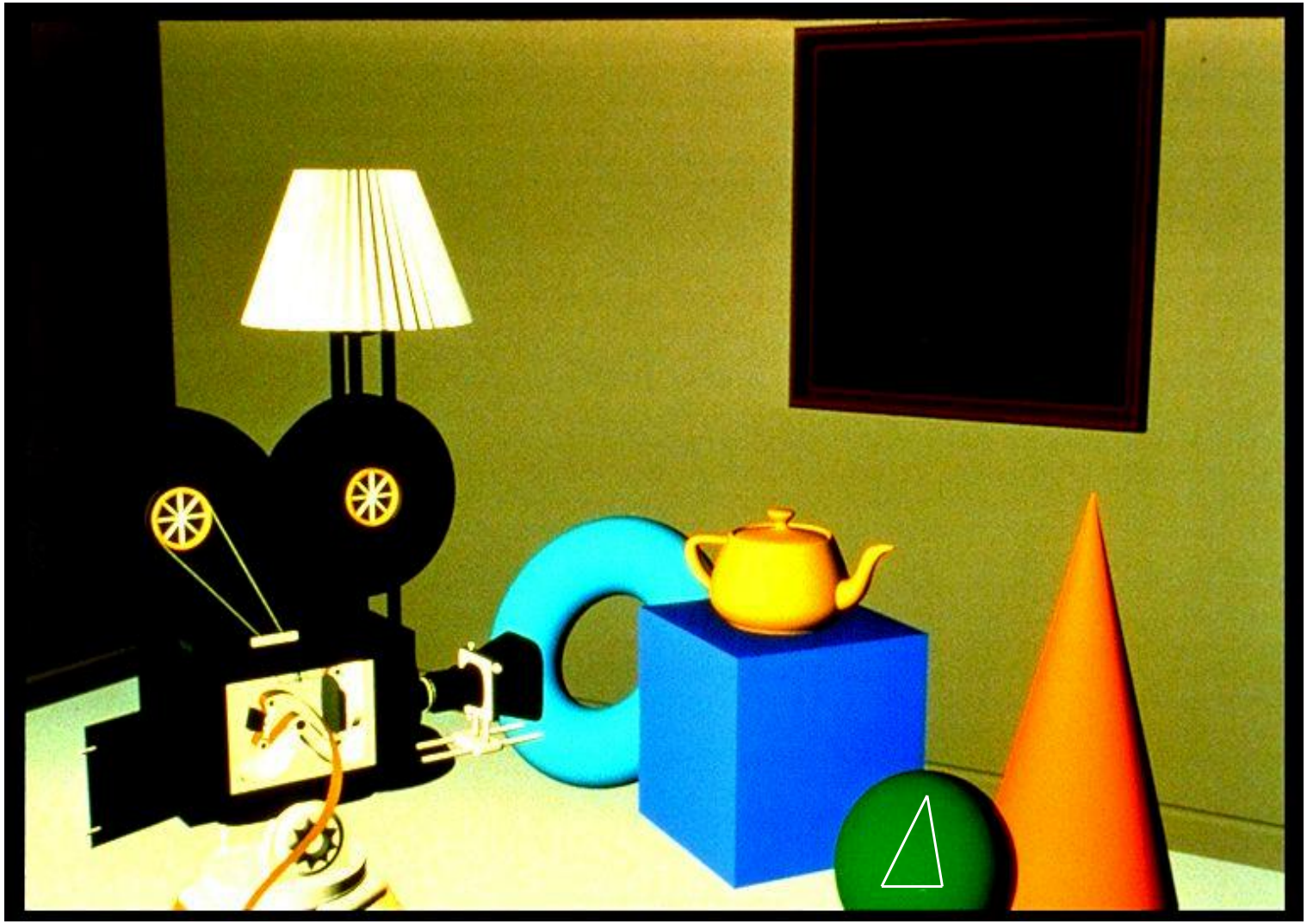


Per-Polygon (Faceted, Flat) Shading



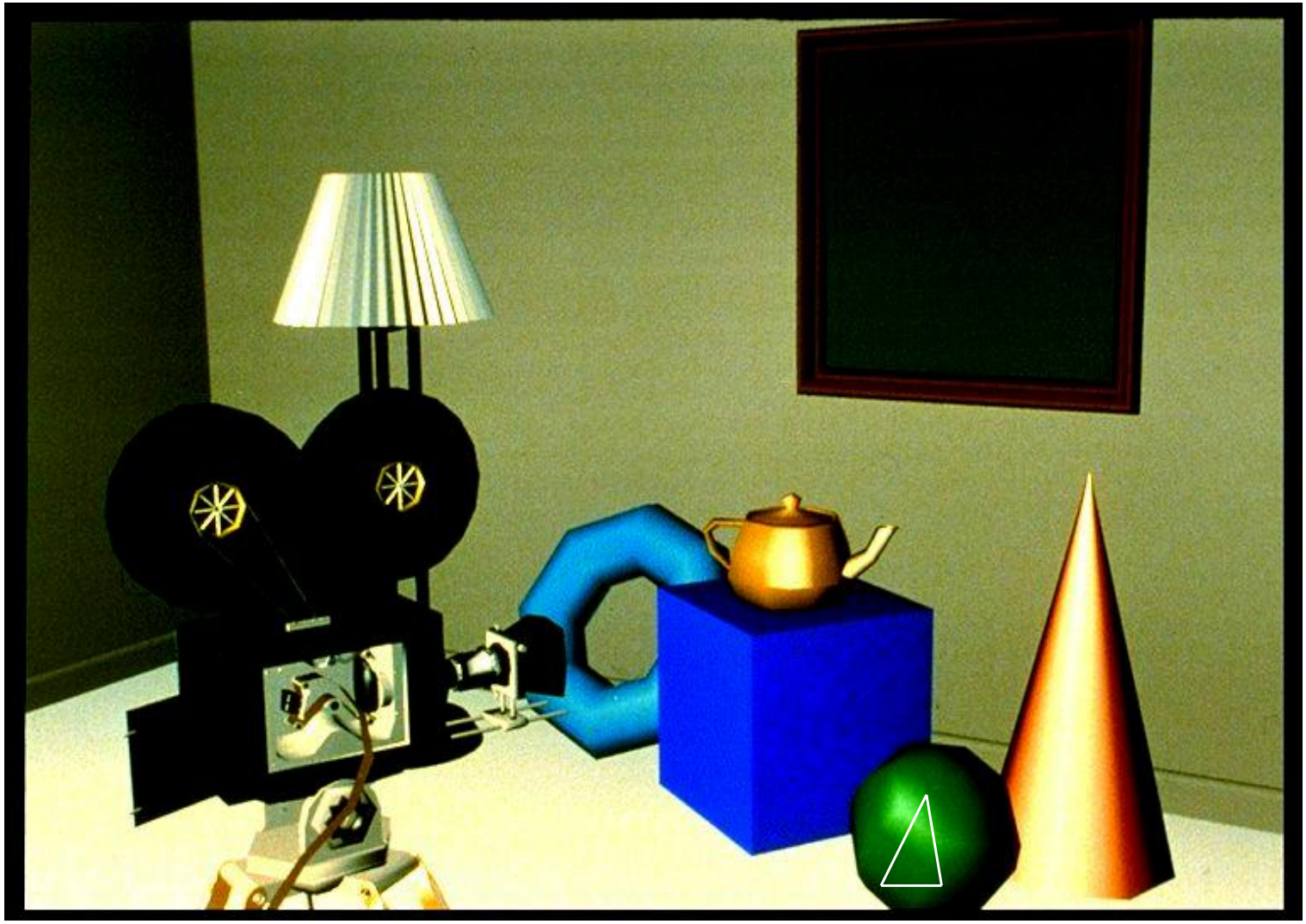
Guess PER POLYGON **?** **FLAT** / SMOOTH **?**

Gouraud Shading



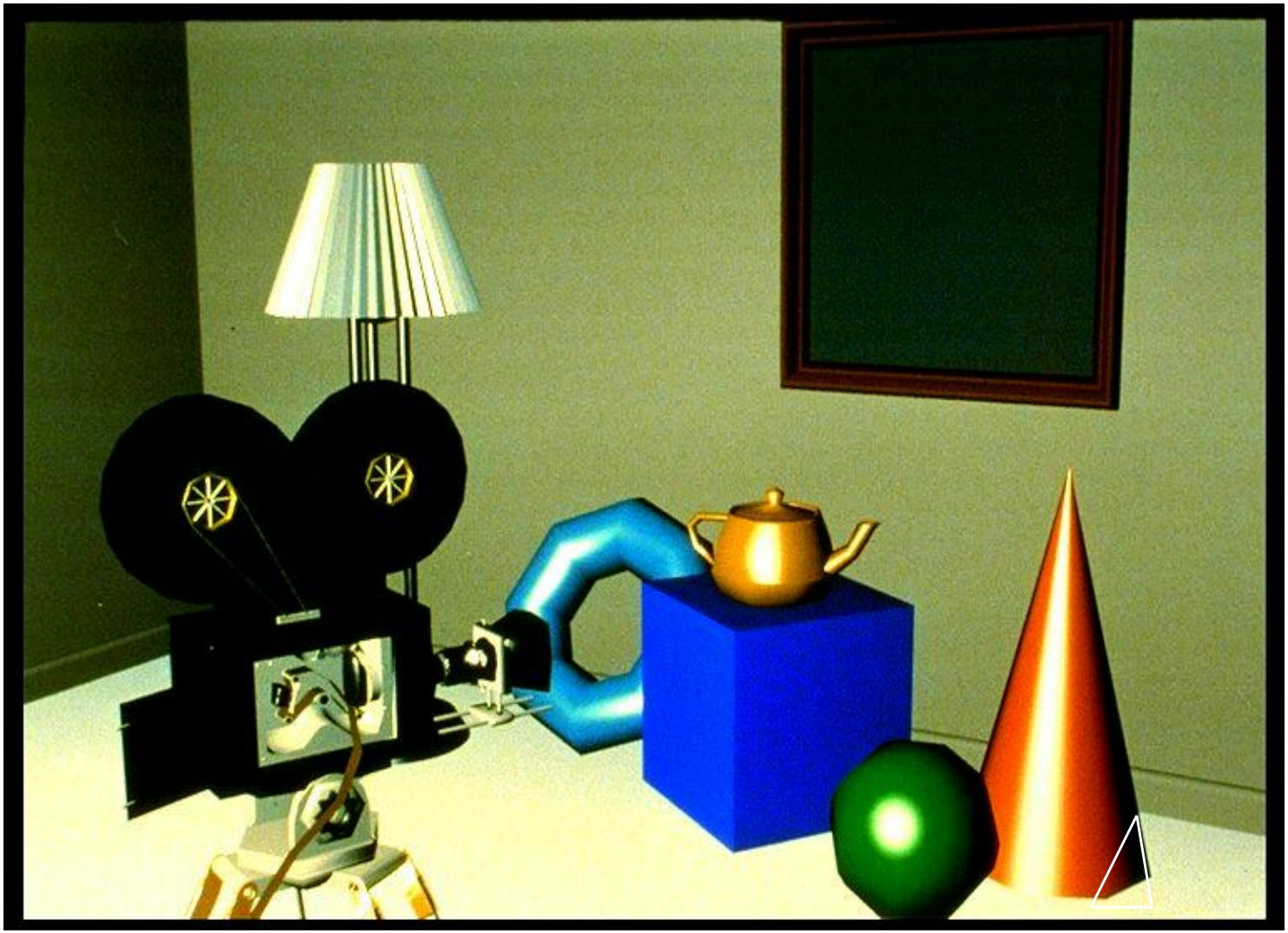
Guess PER POLYGON **?** **FLAT** / SMOOTH **?**

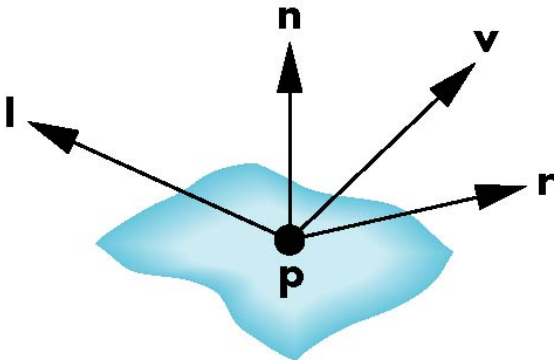
Specular Reflection



Guess specular reflection **?**

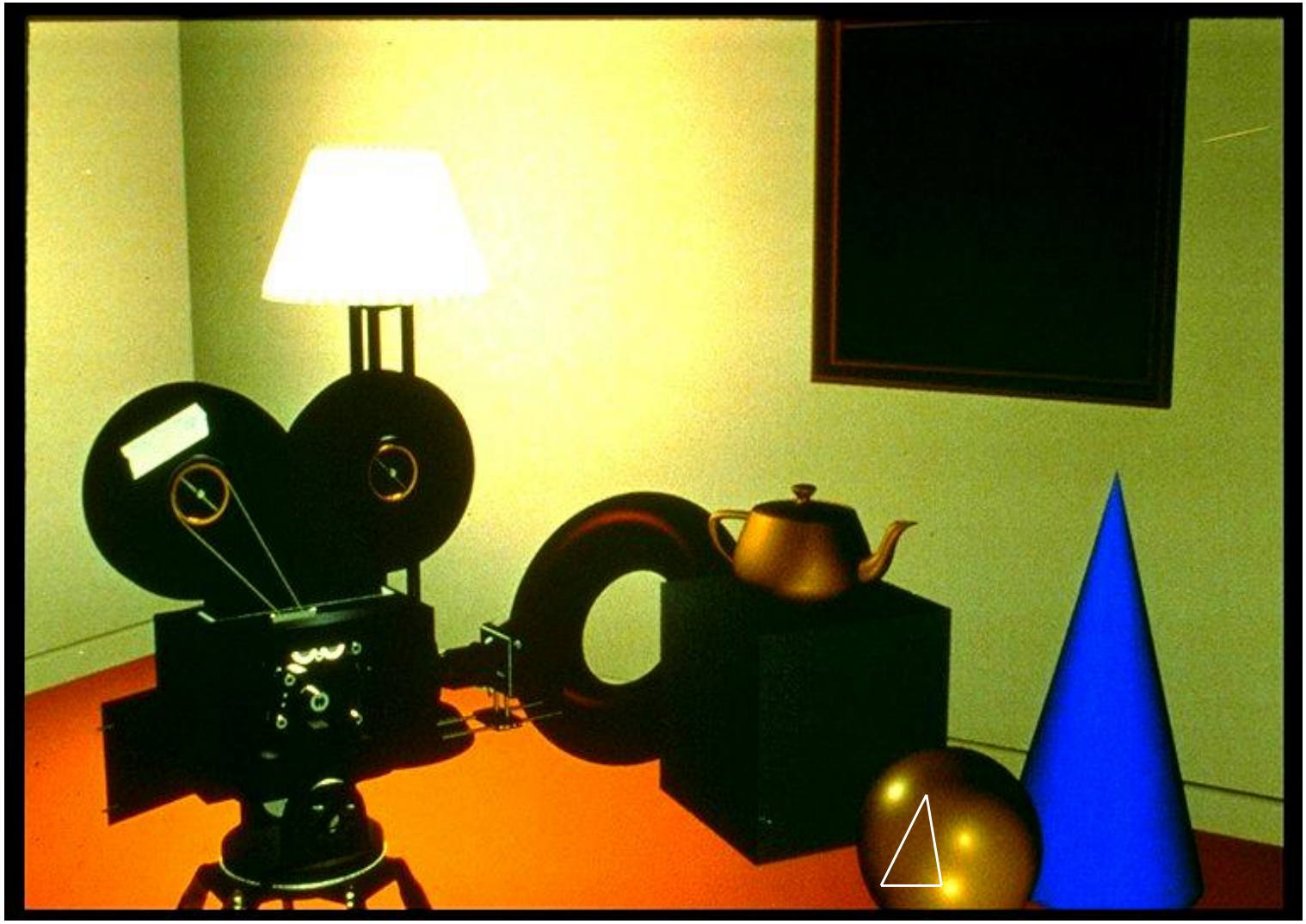
Phong Shading





$$I = k_d I_d \; l \bullet n + k_s I_s (v \bullet r)^{\alpha} + k_a I_a$$

Complex Lighting and Shading



Lighting

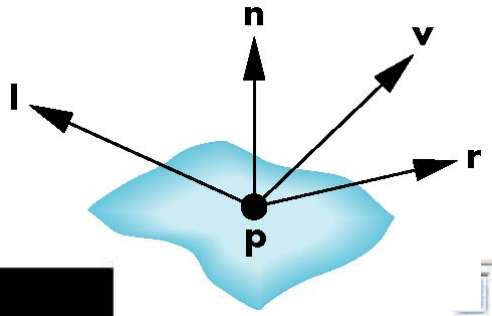
PLATE 17: Light sources in a scene. *(Courtesy of Unigine.)*

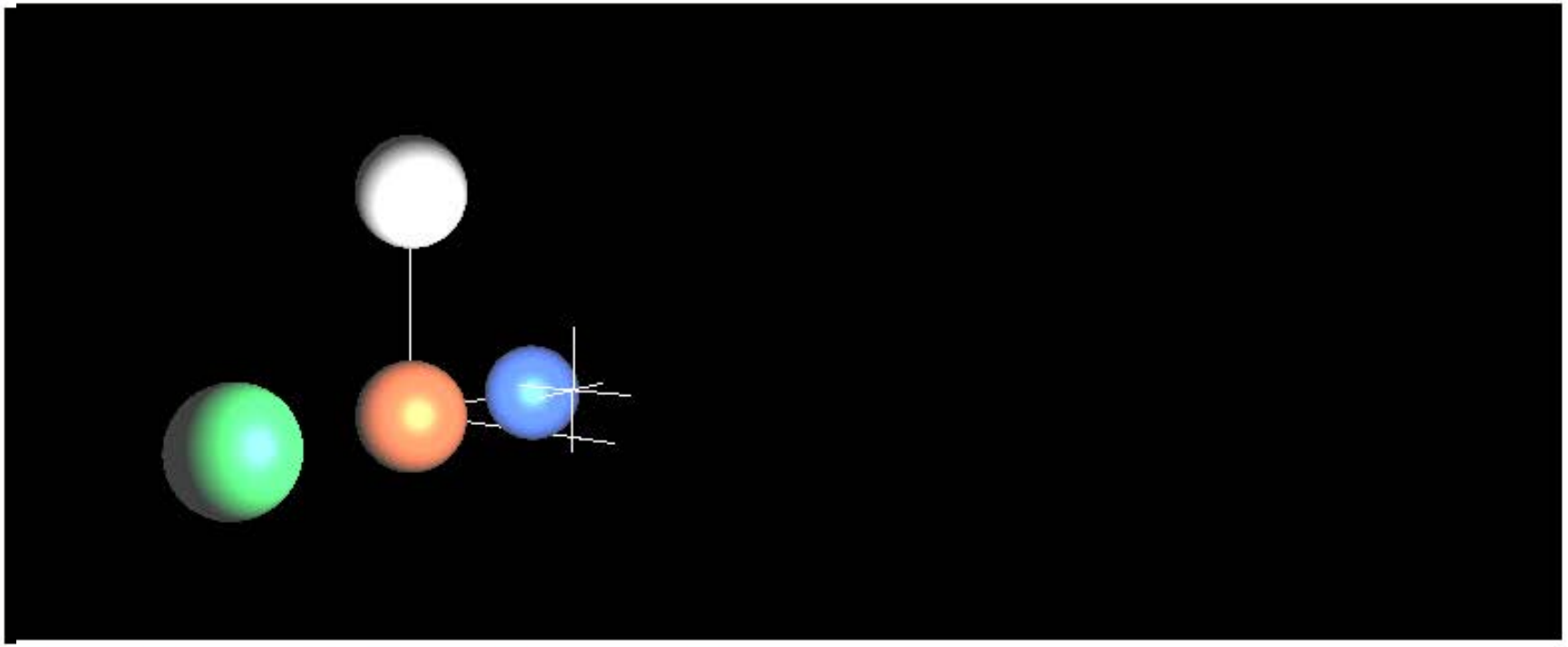
Flat vs Smooth Shading

Guess FLAT / SMOOTH ?

Lighting Applet

$$I = k_d I_d \; l \bullet n + k_s I_s (v \bullet r)^{\alpha} + k_a I_a$$





I = k_a * I_a + k_d * I_l * (N . L) + k_s * I_l * (V . R)^n_s

Enable ambient | Enable diffuse | Enable specular

k_a	0.5	k_d	0.7	k_s	0.7
I_a (R,G & B)	0.5	I_l (R,G & B)	2.0	n_s	5

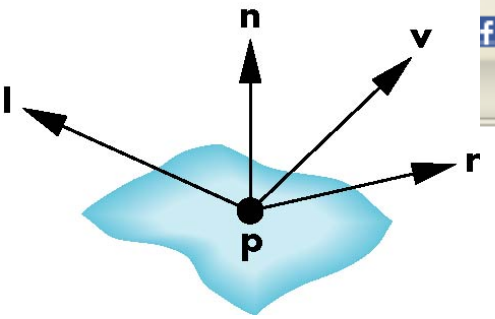
Light Scene | Rotate viewer by 25 degrees

Light attenuation (1/d^2) | Rotate light by 25 degrees

Copy image to window | four_spheres.dat

http://www.cs.princeton.edu/~min/cs426/jar/light.html

$$I = k_d I_d \, l \bullet n + k_s I_s (v \bullet r)^{\alpha} + k_a I_a$$



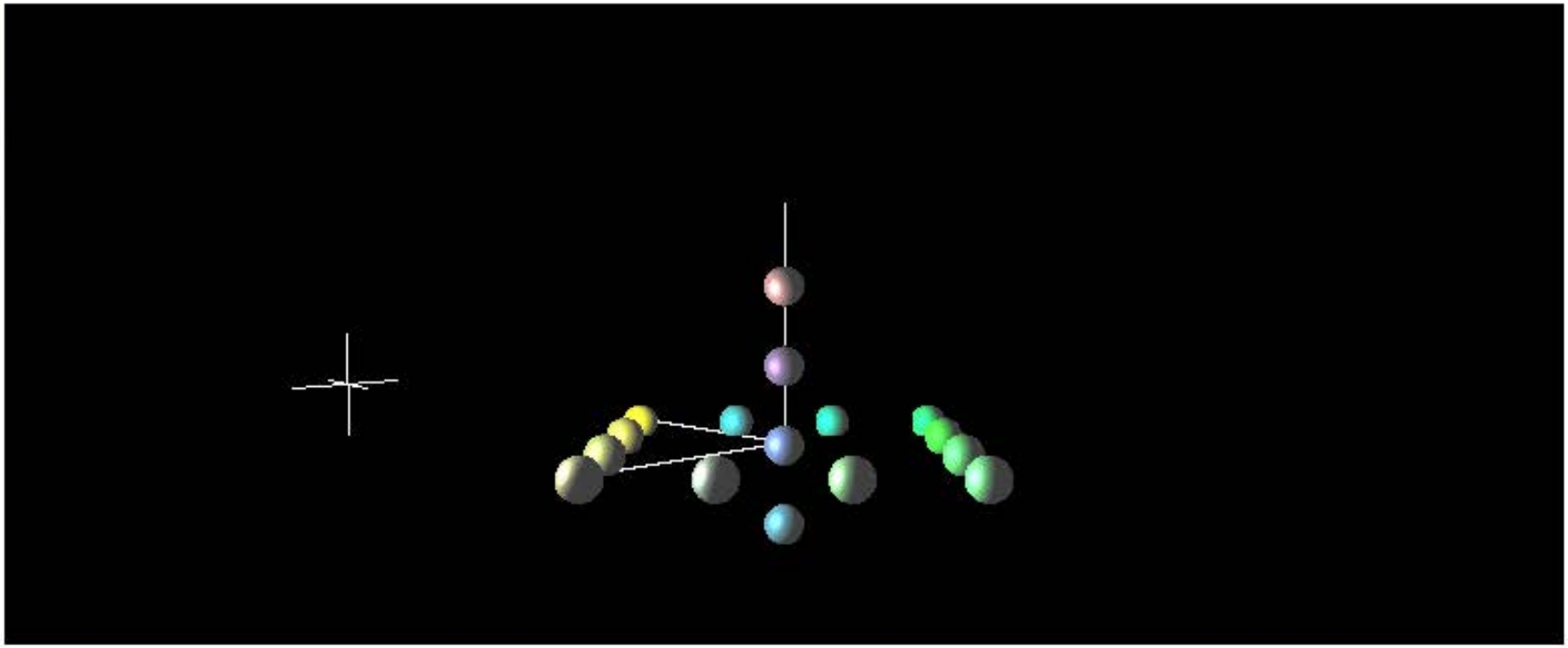
- Specular highlights are done using Phong's shading model.
- The spheres are raytraced individually, going from the furthest to the closest sphere. This means you won't see any shadows.
- If you run this applet on anything less than a 24-bit colour display, the resulting pictures are not going to be that pretty.
- When running this applet under older versions (< 4.x) of Netscape, the function "copy image to window" doesn't work properly: whenever it is redrawn, it shows the current picture. When run using the Appletviewer, it should work fine though.

k here for a list of all applets.

those of you without the book, an explanation of the lighting model parameters:

k_a	material's ambient coefficient
I_a	ambient light (R, G and B)
k_d	material's diffuse coefficient
I_l	light (at crosshair) intensity (R, G and B)
N	surface normal at point of intersection
L	vector to light from point of intersection
k_s	material's specular coefficient
V	vector to viewer from point of intersection
R	vector L reflected at point of intersection
n_s	material's specular reflection parameter

Lighting Applet



I = k_a * I_a + k_d * I_l * (N . L) + k_s * I_l * (V . R)^n_s

☑ Enable ambient ☑ Enable diffuse ☑ Enable specular

k_a	0.5	k_d	0.7	k_s	0.7
I_a (R,G & B)	0.5	I_l (R,G & B)	2.0	n_s	5

Light Scene

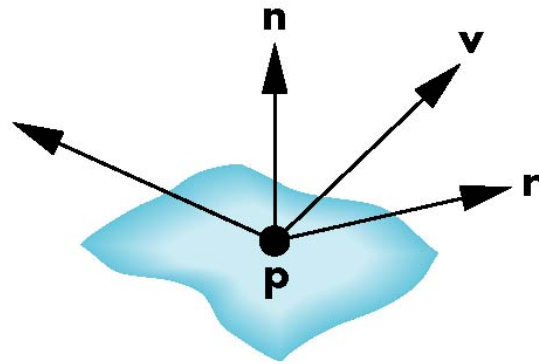
Rotate viewer by 25 degrees

☐ Light attenuation (1/d^2)

Rotate light by 25 degrees

Copy image to window

little_spheres.dat



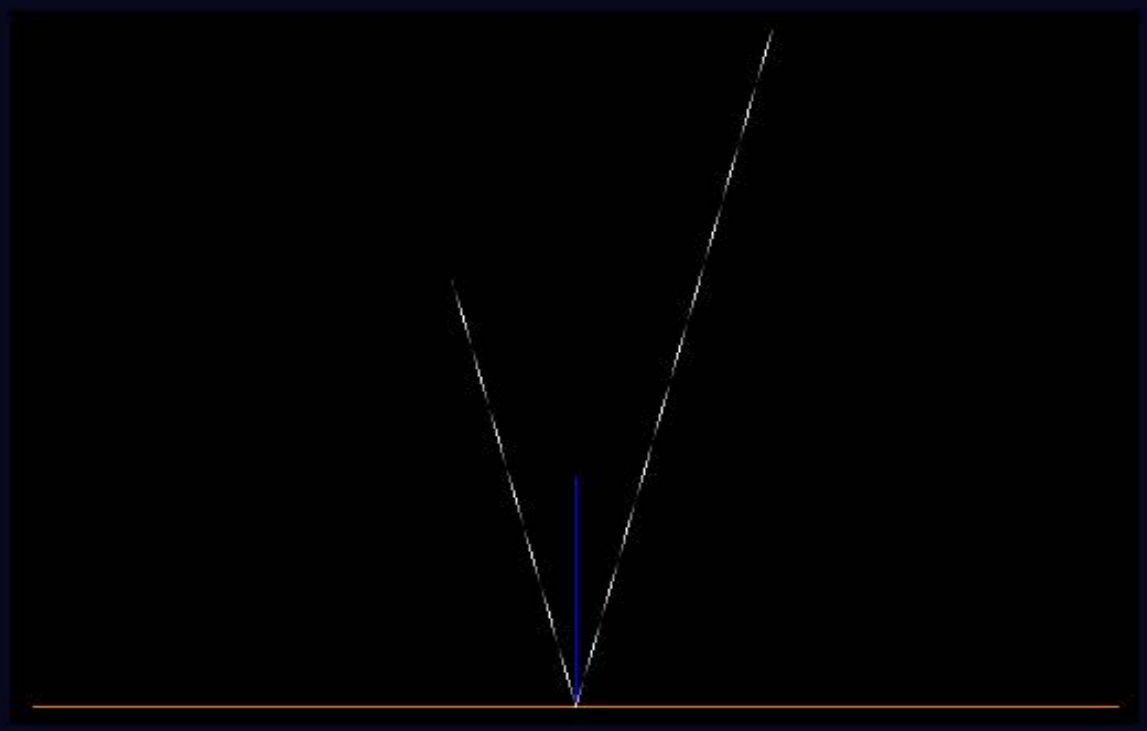
$$I = k_d I_d \, \mathbf{l} \bullet \mathbf{n} + k_s I_s (\mathbf{v} \bullet \mathbf{r})^{\alpha} + k_a I_a$$

VRML – Light Reflection

t Reflection

life.wrl (model/vrml Object)

lick on the picture to change the origin of the light ray. Drag slowly.



n the above diagram the reflecting surface is drawn in red, the normal is drawn in blue.

http://id.mind.net/~zona/mstm/physics/light/rayOptics/reflection/reflection1.html

Shading I

http://www.3dengines.de/feat_phong.html

Objectives

Learn to shade objects so their **images appear three-dimensional**

Introduce the **types of** light-material interactions

Build a simple reflection model---the **Phong** model - **can be used with real time graphics hardware**

Objectives

Learn to shade objects so their images appear three-dimensional

Introduce the **types of** light-material interactions

Build a simple reflection model---the Phong model---that can be used with real time graphics hardware

Why we need shading

- Suppose we build a **model of a sphere** using many polygons and color it with `glColor`. We get something like

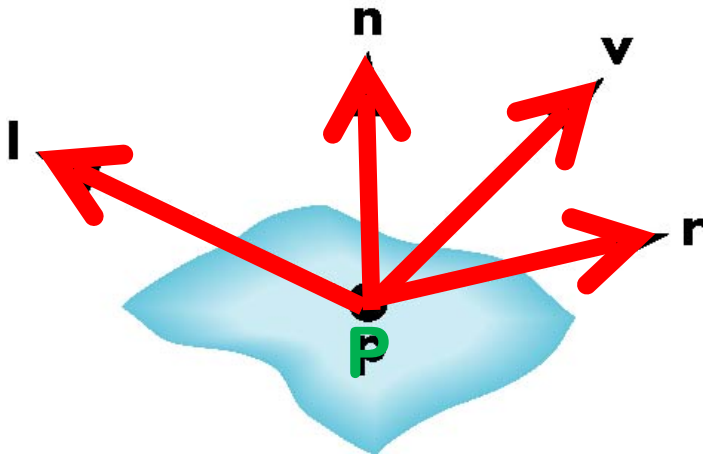
- **But we want**

Shading

- Why does the **image** of a **real sphere look like**

- **Light-material interactions** cause **each point P to have a different color or shade**

- Need to consider **vectors**
  - **Light source(s)** **l**
  - **Material properties** **r**
  - **Location of viewer** **v**
  - **Surface orientation** **P** → **n**



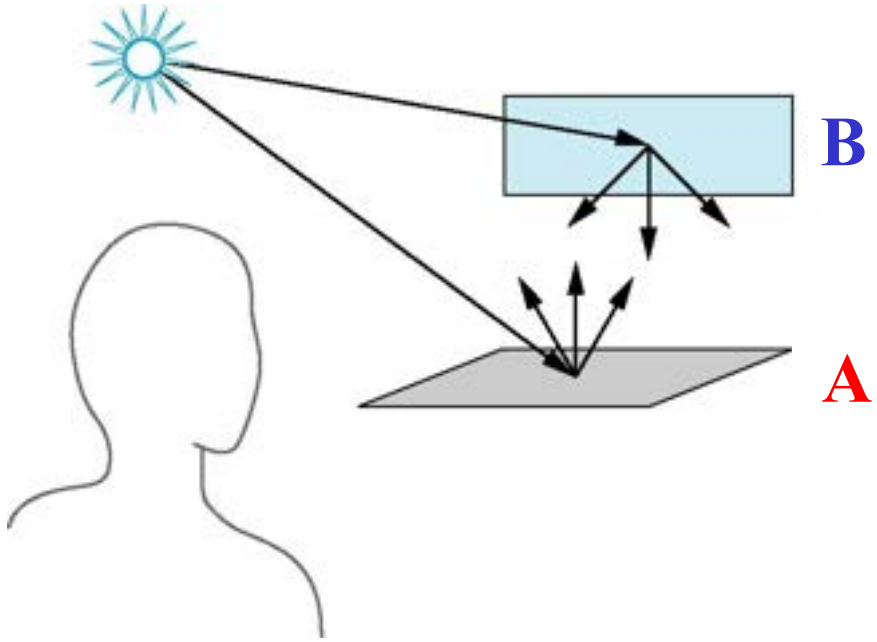
Scattering

- Light strikes **A**
  - Some scattered
  - Some absorbed

- Some of scattered light strikes **B**
  - Some scattered
  - Some absorbed

- Some of this scattered light strikes **A**

and so on



Rendering Equation

The **infinite** scattering and absorption of light can be described by the ***rendering equation***

(**Bidirectional Reflection Distribution Function (BRDF)**)

Cannot be solved in general

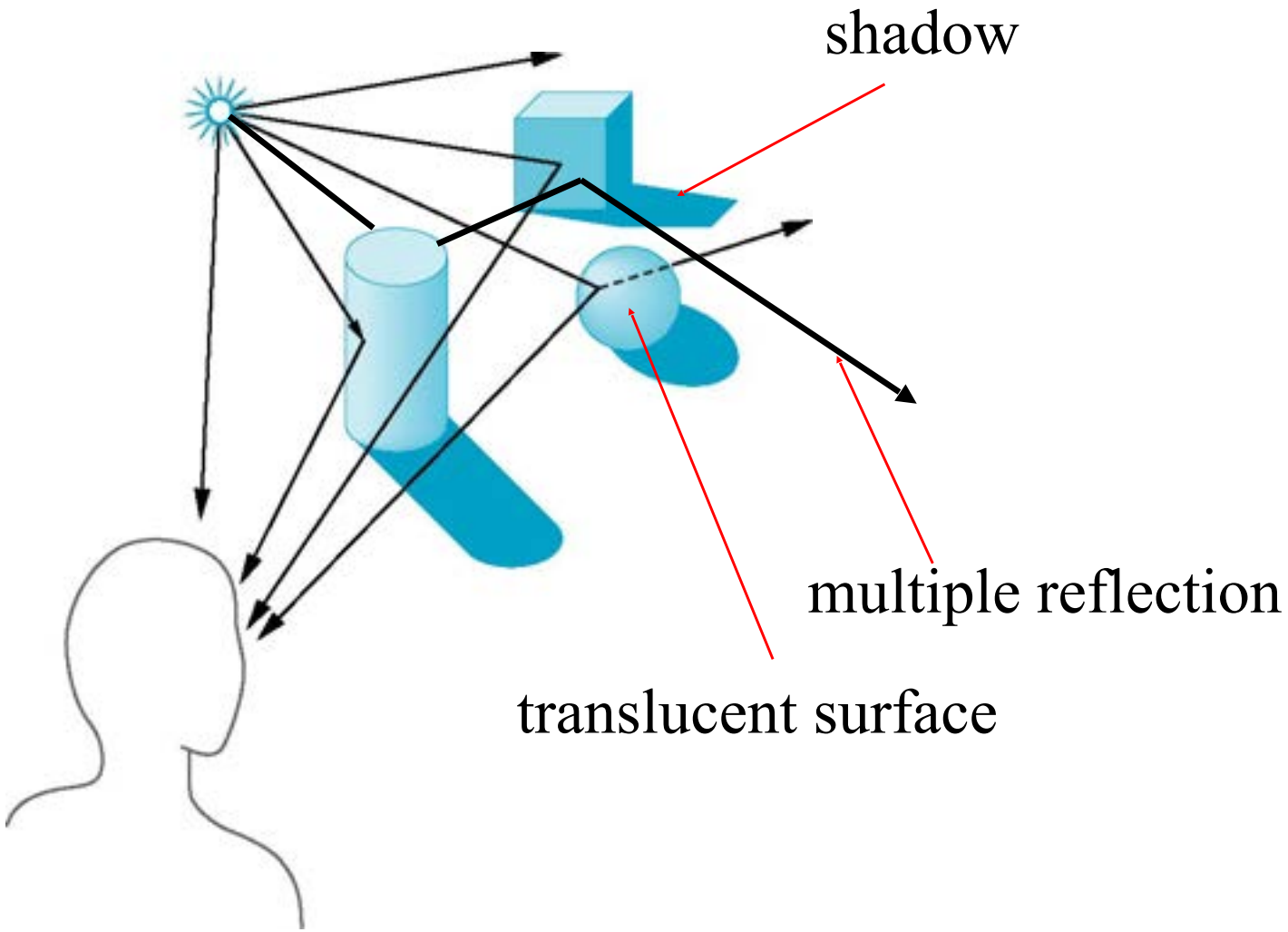
Ray tracing is a special case for perfectly reflecting surfaces

Rendering equation is **global** and includes

Shadows

Multiple scattering from object to object

Global Effects



Local vs Global Rendering

Correct shading requires a global calculation involving **all objects and light sources**

- **Incompatible with pipeline model which shades each polygon independently (local rendering)**

However, in computer graphics, especially real ti
graphics, we are happy if things "look right"

- **Exist many techniques for approximating global effects**

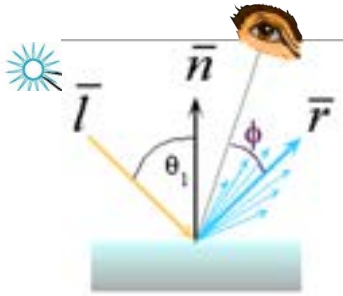
Objectives

Learn to shade objects so their images appear three-dimensional

Introduce the **types of** light-material interactions

Build a simple reflection model---the Phong model---that can be used with real time graphics hardware

Light-Material Interaction



Light that strikes an object is partially absorbed and partially scattered (reflected)

The amount reflected determines the color and brightness of the object

A surface appears red under white light because the red component of the light is reflected and the rest is absorbed

The reflected light is scattered (diffuse) in a manner that depends on the smoothness and orientation of the surface (n)

http://www.walter-fendt.de/ph14e/refraction.htm

Refraction of Light Applet

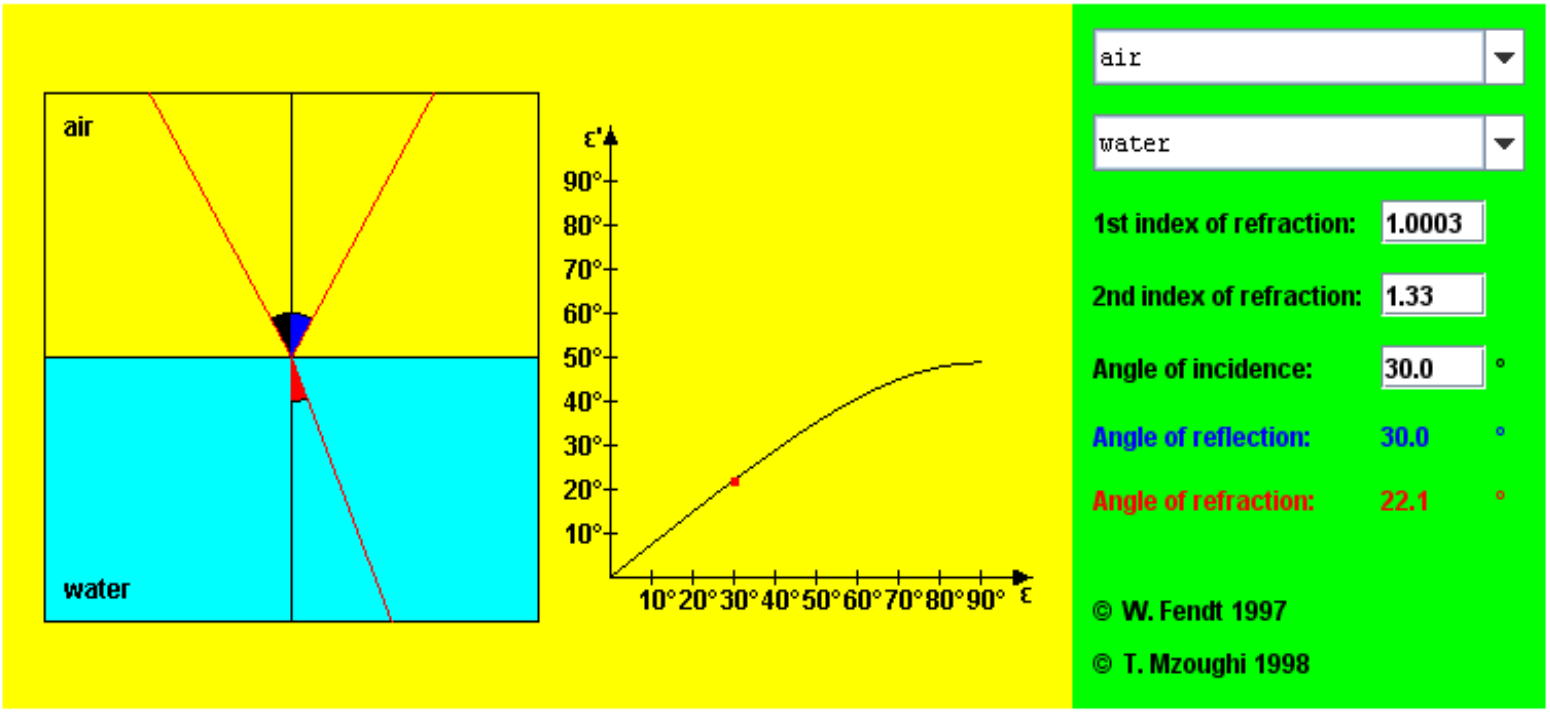
Refraction of Light

A ray of light coming from the top left strikes the boundary surface of two media. (It is possible to choose the substances in both lists.) The medium which has the bigger index of refraction is painted blue, the other yellow. You can vary the incident ray with pressed mouse button. The applet will show the reflected and the refracted ray and calculate the corresponding angles:

Angle of incidence (black)

Angle of reflection (blue)

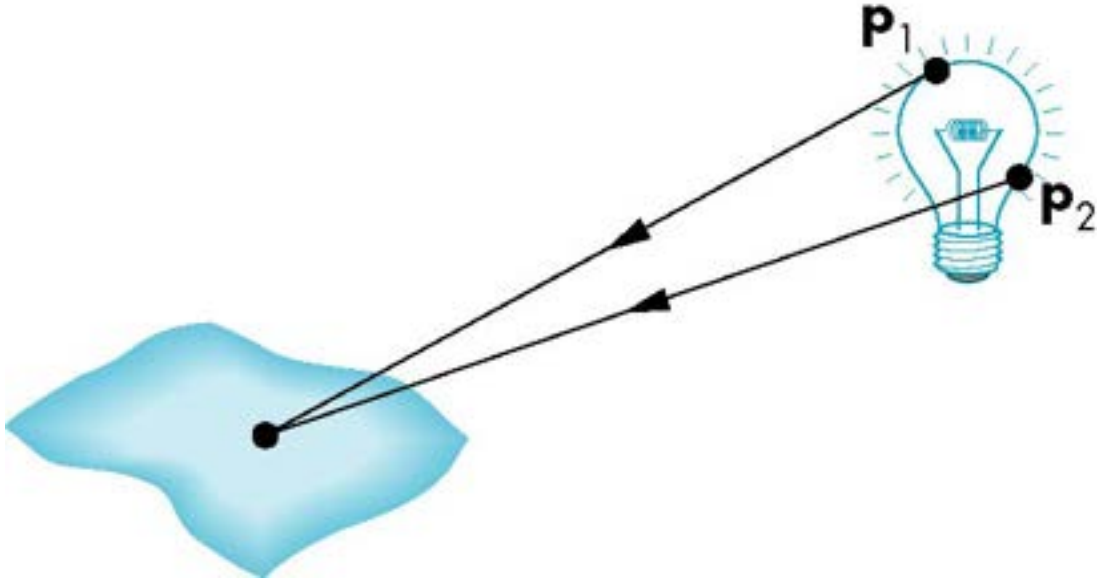
Angle of refraction (red)



http://www.walter-fendt.de/ph14e/refraction.htm

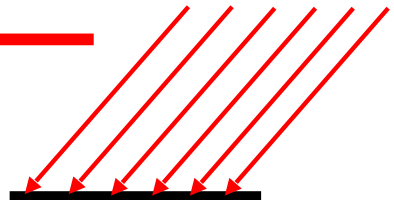
Light Sources

General light sources are difficult to work with because we must **integrate light coming from all points on the source**

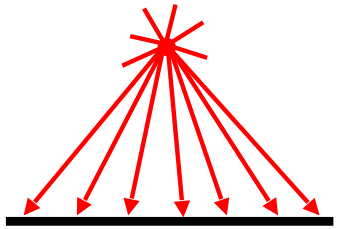


Light Sources

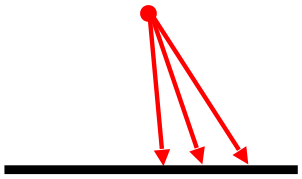
- **directional/parallel** lights
  - **point** at infinity: $(x,y,z,\mathbf{0})^T$



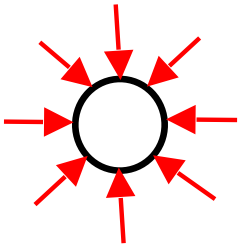
- **point** lights
  - finite **position**: $(x,y,z,\mathbf{1})^T$



- **spot**lights
  - **position**, direction, angle



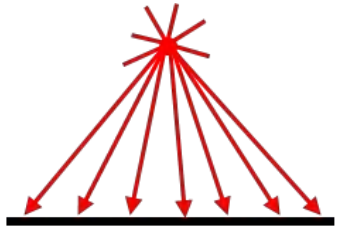
- **ambient** lights



Simple Light Sources

Point source

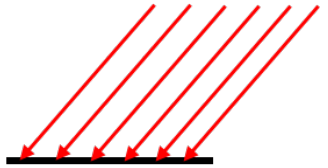
Model with **position** and **color**



point lights

- finite **position**: $(x,y,z,1)^T$

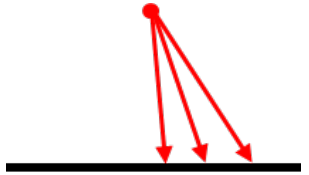
Distant source = **infinite distance away** (**parallel**)



point at infinity: $(x,y,z,0)^T$

Spotlight

Restrict light from **ideal point source**



position, direction, angle

Ambient light

Same amount of light everywhere in scene

Can model contribution of many sources and reflecting surfaces

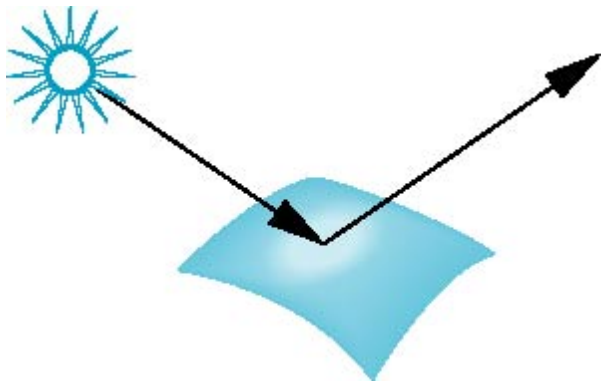
Surface Types

$$I = k_d I_d \; \mathbf{l} \bullet \mathbf{n} + k_s I_s (\mathbf{v} \bullet \mathbf{r})^{\alpha} + k_a I_a$$

The **smoother a surface**, the **more reflected light is concentrated in the direction a perfect mirror would ref the light - specular**

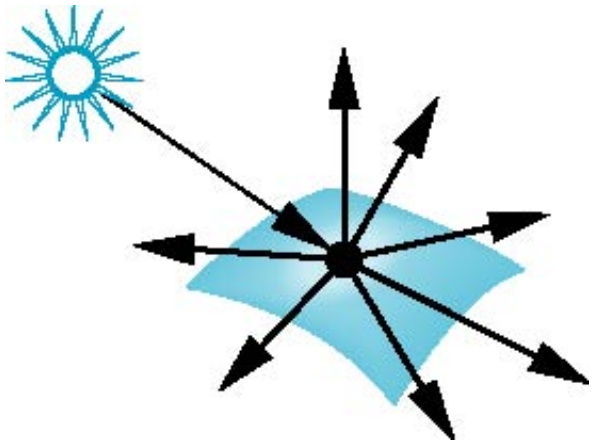
A **very rough surface** scatters light in all directions-**diffuse**

Specular reflections



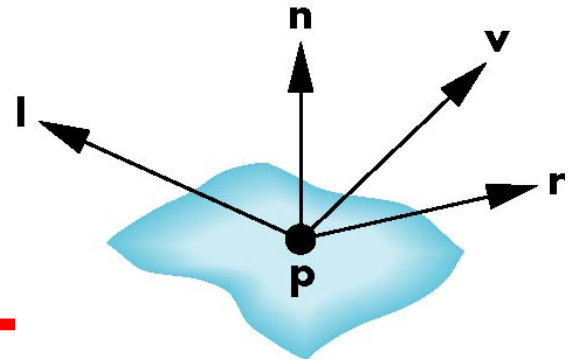
smooth surface

Diffuse reflections



rough surface

Objectives



Learn to shade objects so their images appear three-dimensional

Introduce the **types of** light-material interactions

Build a simple **reflection** model---the **Phong model**-- that can be used with real time graphics hardware

$$I = k_d I_d \; \mathbf{l} \bullet \mathbf{n} + k_s I_s (\mathbf{v} \bullet \mathbf{r})^{\alpha} + k_a I_a$$

Lighting vs. Shading

lighting

simulating the interaction of **light** with **surface**

shading

deciding pixel color

continuum of realism: **when do we do lighting calculation**?

Shading Models

- **flat shading**

  compute **Phong lighting** once for entire polygon

- **Gouraud shading**

  compute **Phong lighting** at the **vertices** and **interpolate** lighting values across polygon

- **Phong shading**

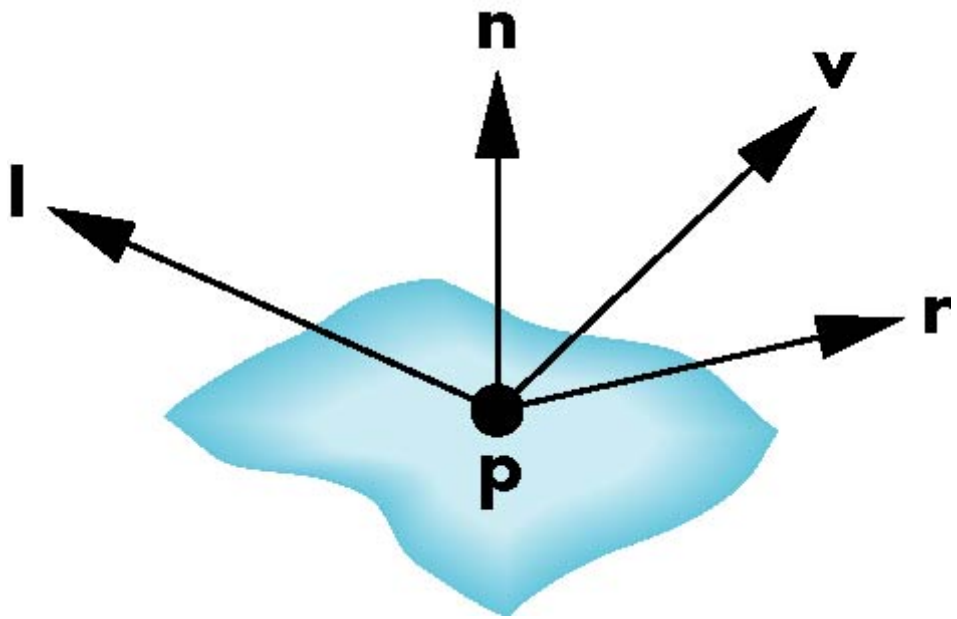
  compute **averaged vertex** normals

  interpolate normals across polygon and perform **Phong lighting** across polygon

Phong Model

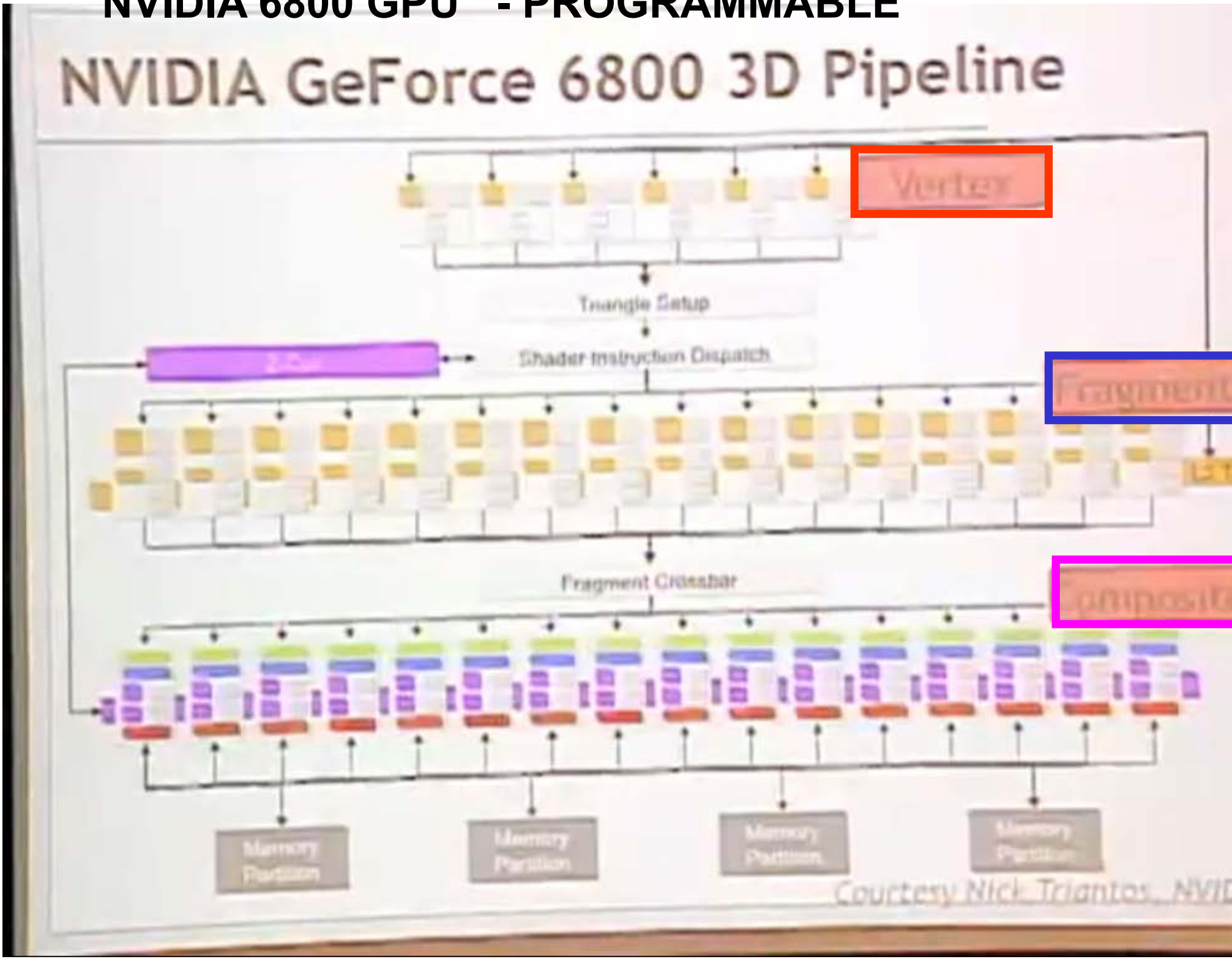
$$I = k_d I_d \; l \bullet n + k_s I_s (v \bullet r)^{\alpha} + k_a I_a$$

- **A simple model that can be computed rapidly**
- Has three components
  - **Ambient**
  - **Diffuse**
  - **Specular**
- Uses four **vectors**
  - **To source** l
  - **To viewer** v
  - **Normal** n
  - **Perfect reflector** r

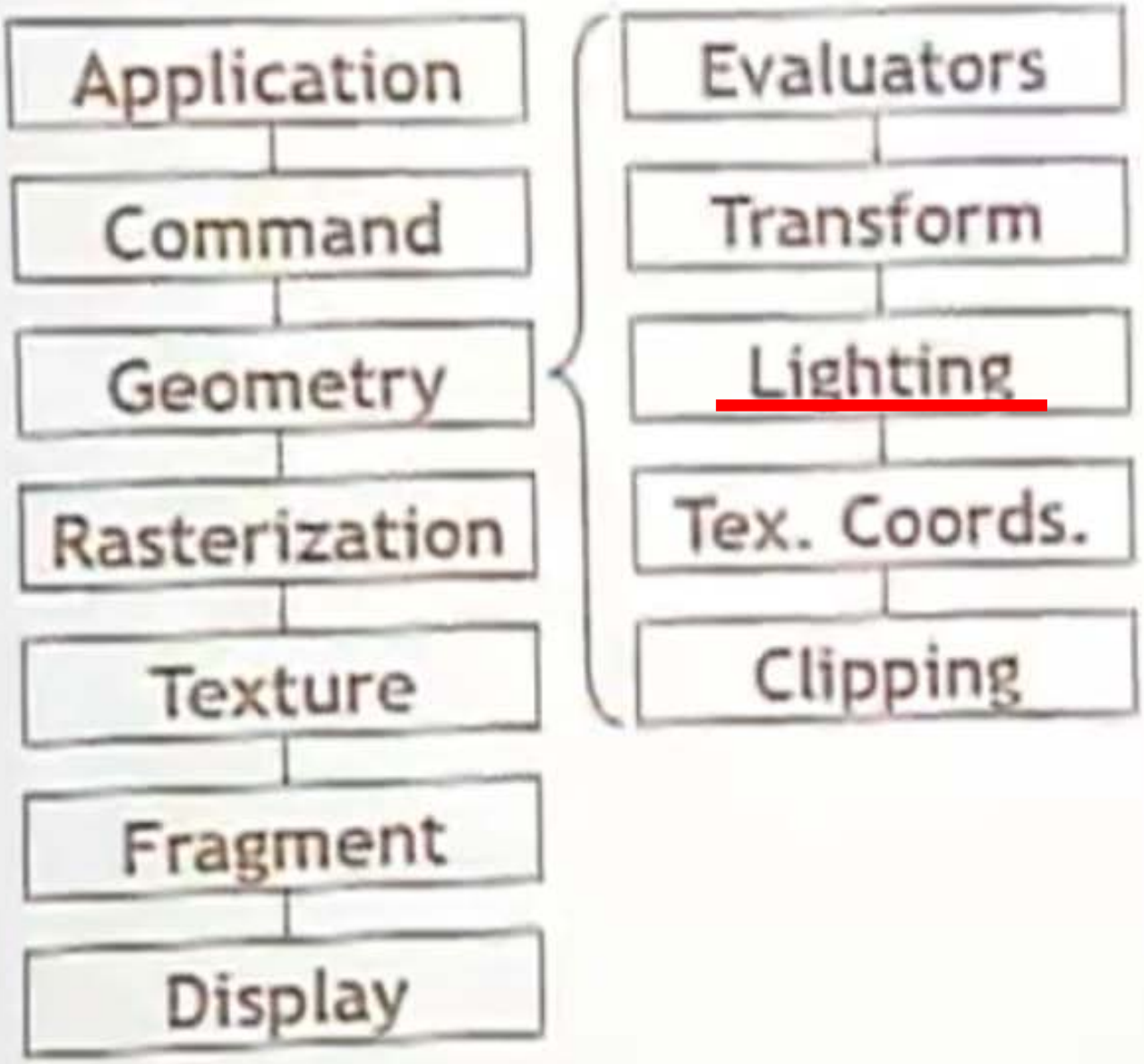


NVIDIA & ATI GPUs
Graphics in REAL-TIME

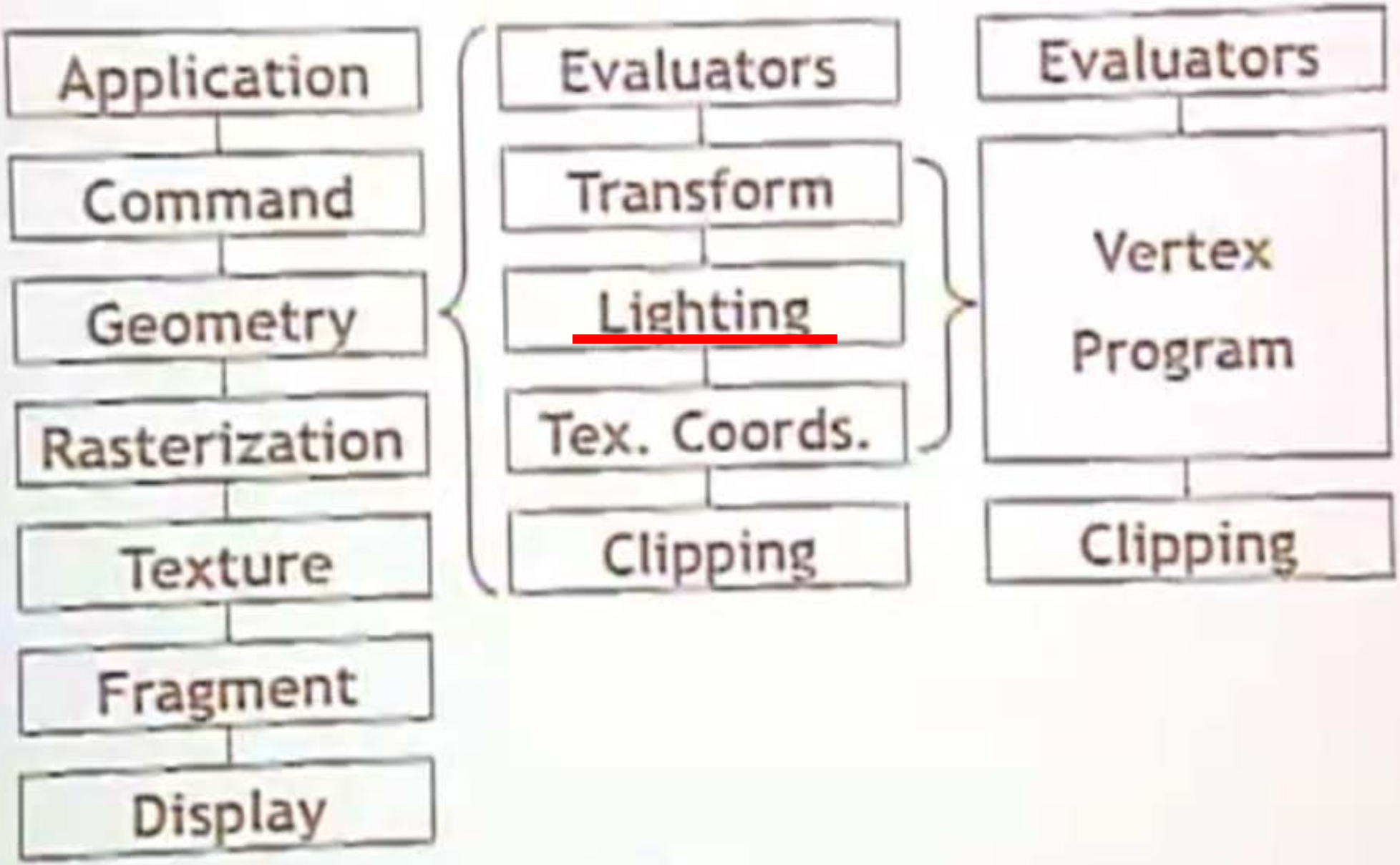
NVIDIA 6800 GPU - PROGRAMMABLE



Vertex Programs in the Graphics Pipeline



Vertex Programs in the Graphics Pipeline



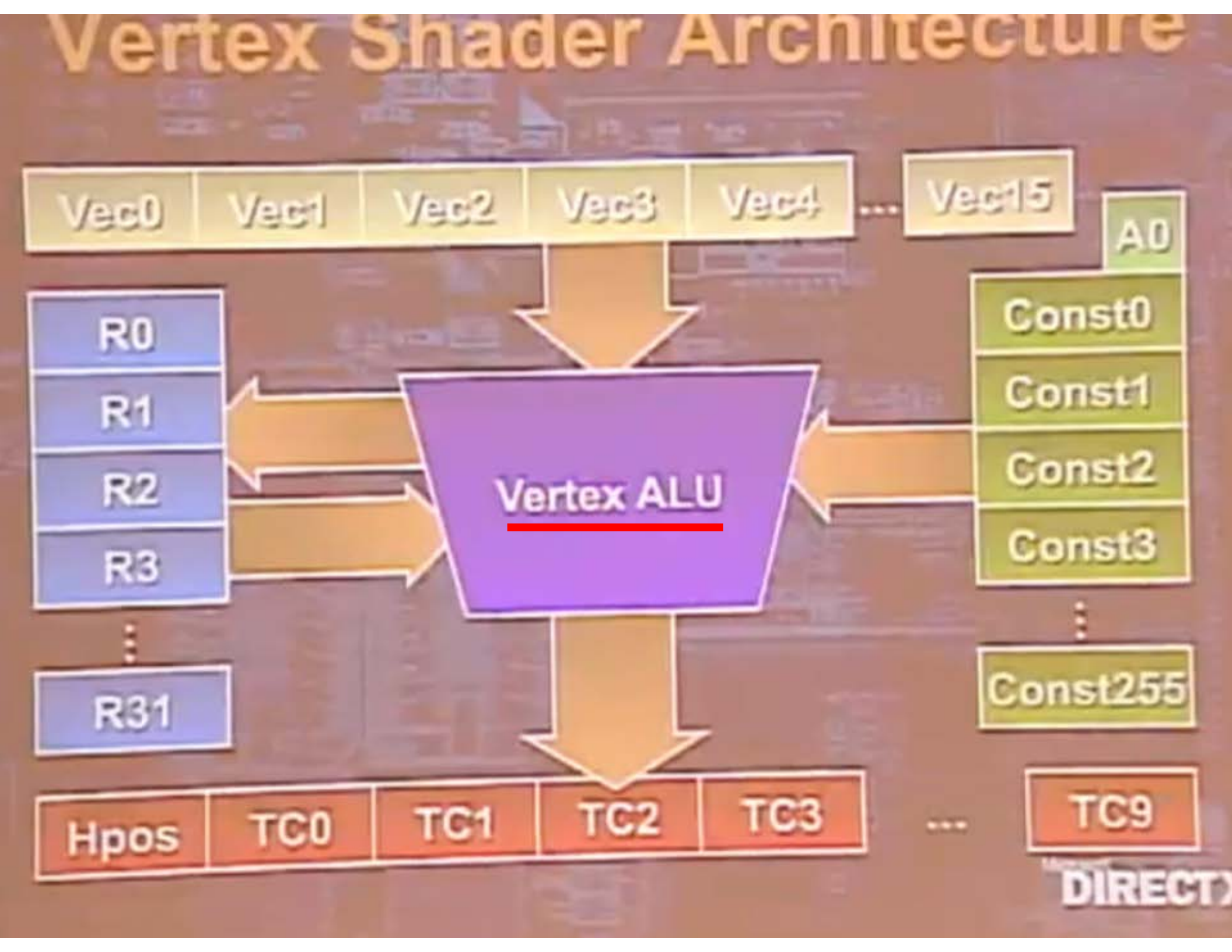
Hardest Thing To Get Used To

When you write a vertex or fragment program,

You Write One Program.

It Runs On Every Vertex/Fragment.

Programming model is "SPMD" (Single Program, Multiple Data)


Vertex Shader Architecture
Vec0
Vec1
Vec2
Vec3
Vec4
Vec15
A0
R0
R1
R2
R3
R31
Vertex ALU
Const0
Const1
Const2
Const3
Const255
Hpos
TC0
TC1
TC2
TC3
TC9
DIRECT

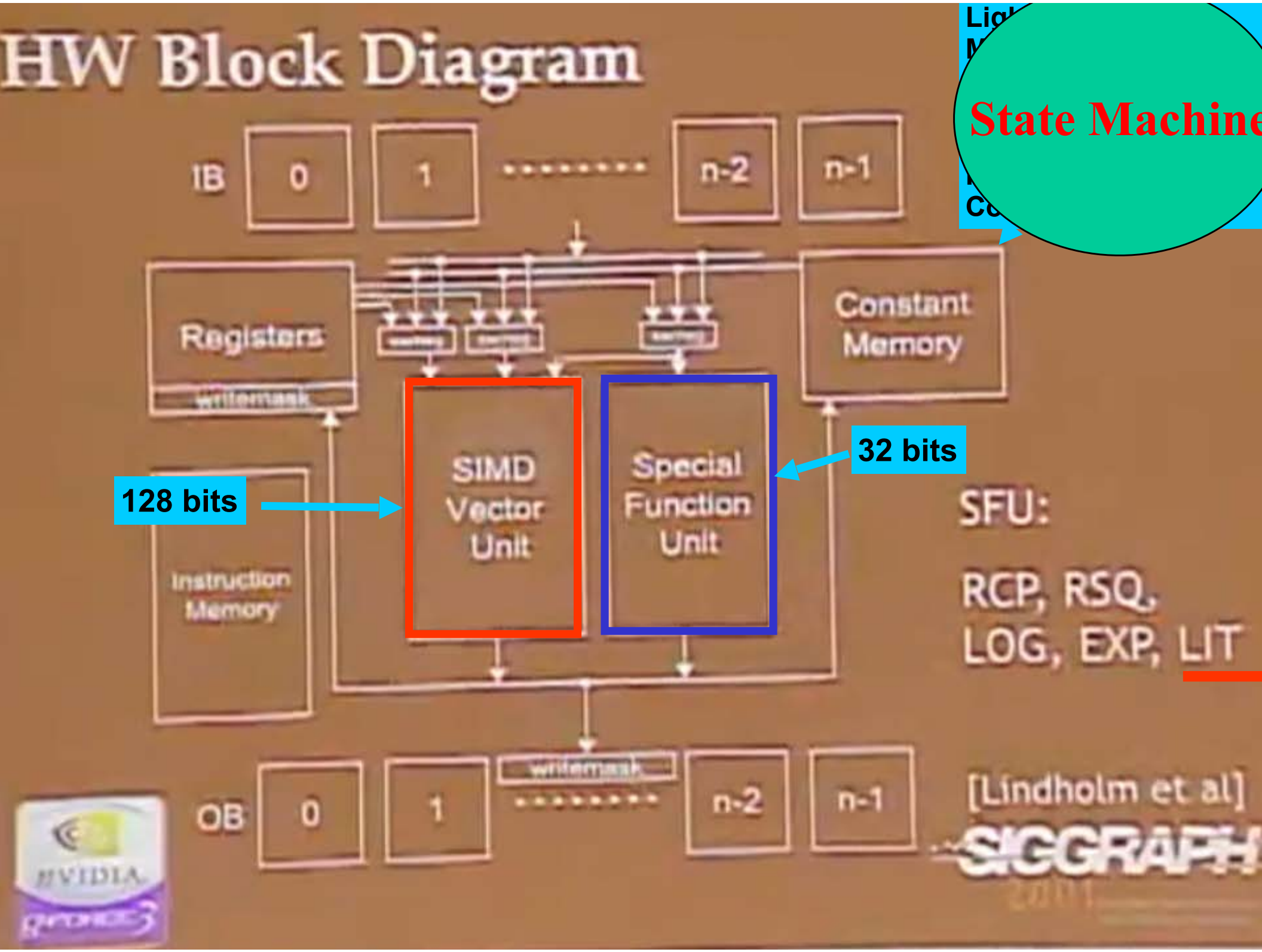
Vertex Input Registers (16)

Attribute Register	Conventional per-vertex Parameter	Conventional Command	Conventional Mapping
0	vertex position	glVertex	x,y,z,w
1	vertex weights	glVertexWeightEXT	w,0,0,1
2	normal	glNormal	x,y,z,1
3	Primary color	glColor	r,g,b,a
4	secondary color	glSecondaryColorEXT	
5	r,g,b,fog coordinate	glFogCoordEXT	fc,0,0,1
6			
7			
8	Texture coord 0	glMultiTexCoord	s,t,r,q
9	Texture coord 1	glMultiTexCoord	s,t,r,q
10	Texture coord 2	glMultiTexCoord	s,t,r,q
11	Texture coord 3	glMultiTexCoord	s,t,r,q
12	Texture coord 4	glMultiTexCoord	s,t,r,q
13	Texture coord 5	glMultiTexCoord	s,t,r,q
14	Texture coord 6	glMultiTexCoord	s,t,r,q
15	Texture coord 7	glMultiTexCoord	s,t,r,q

Vertex Output Registers (16)

Register Name	Description	Component Interpretation
o[HPOS]	Homogeneous clip space position	(x,y,z,w)
o[COL0]	Primary color (front-facing)	(r,g,b,a)
o[COL1]	Secondary color (front-facing)	(r,g,b,a)
o[BFC0]	Back-facing primary color	(r,g,b,a)
o[BFC1]	Back-facing secondary color	(r,g,b,a)
o[FOGC]	Fog coordinate	(f,*,*,*)
o[PSIZ]	Point size	(p,*,*,*)
o[TEX0]	Texture coordinate set 0	(s,t,r,q)
o[TEX1]	Texture coordinate set 1	(s,t,r,q)
o[TEX2]	Texture coordinate set 2	(s,t,r,q)
o[TEX3]	Texture coordinate set 3	(s,t,r,q)
o[TEX4]	Texture coordinate set 4	(s,t,r,q)
o[TEX5]	Texture coordinate set 5	(s,t,r,q)
o[TEX6]	Texture coordinate set 6	(s,t,r,q)
o[TEX7]	Texture coordinate set 7	(s,t,r,q)

HW Block Diagram



Basic Instructions (VS1.0)

17 instructions - all 4x32 FP

MOV	MIN	DP3
MUL	MAX	DP4
ADD	SLT	DST
MAD	SGE	LIT
RCP	ARL	
RSQ		
EXP		
LOG		

Instruction Set

- NO SUB!
- MOV copy from register to register
- **MAD Multiply and Add**
- **Reciprocal, Reciprocal Square Root (Normals)**
- **DP3 Dot Product**
- **DST distance (Pythagorean distance between two vectors)**
- MIN, MAX
- ARL addressing instruction
- **LIT CISC to the n^{th} power; entire blend lighting model that is part of OPENGL. There are 5 components, specular, diffuse, ambient components and a lot of dot products.** The most special purpose instruction that you heard about it.

Program Examples

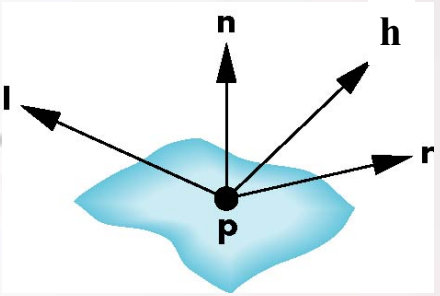
Vector Cross Product

```
# |   i       j       k    | into R2.
# | R0.x    R0.y    R0.z |
# | R1.x    R1.y    R1.z |
MUL R2, R0.zxyw, R1.yzxw;          // swizzle
MAD R2, R0.yzxw, R1.zxyw, -R2; // swizzle, negation
```

Vector Normalize

```
# R1 = (nx,ny,nz)
#
# R0.xyz = normalize(R1)
# R0.w   = 1/sqrt(nx*nx + ny*ny + nz*nz)
DP3 R0.w, R1, R1;
                                   // write-mask
RSQ R0.w, R0.w;
                                   // promotion
MUL R0.xyz, R1, R0.w;
```

Simple Graphics Pipeline




```
# c[0-3]   = Mat;                c[4-7]   = Mat^(-T)
# c[32]    = L;                  c[33]    = H
# c[35].x  = Md * Ld;            c[35].y  = Ma * La
# c[36]    = Ms;                 c[38].x  = s
DP4   o[HPOS].x, c[0], v[OPOS];          # Transform position.
DP4   o[HPOS].y, c[1], v[OPOS];
DP4   o[HPOS].z, c[2], v[OPOS];
DP4   o[HPOS].w, c[3], v[OPOS];
DP3   R0.x, c[4], v[NRML];               # Transform normal.
DP3   R0.y, c[5], v[NRML];
DP3   R0.z, c[6], v[NRML];
DP3   R1.x, c[32], R0;                   # R1.x = L DOT N
DP3   R1.y, c[33], R0;                   # R1.y = H DOT N
MOV   R1.w, c[38].x;                     # R1.w = s
LIT   R2, R1;                            # Compute lighting
MAD   R3, c[35].x, R2.y, c[35].y;        # diffuse + ambient
MAD   o[COL0].xyz, c[36], R2.z, R3;      # + specular
END
```

Vertex IN Vertex OUT

$$I = k_d I_d \; \mathbf{l} \bullet \mathbf{n} + k_s I_s (\mathbf{h} \bullet \mathbf{r})^{\alpha} + k_a I_a$$

Ideal Reflector r

- Normal vector **n** is determined by local orientation
- angle of incidence = angle of reflection
- The three vectors must be coplanar

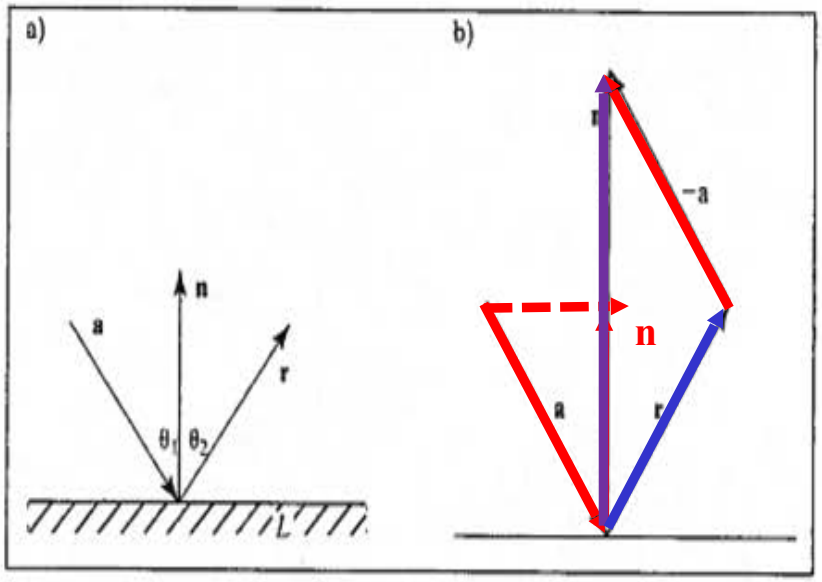


FIGURE 4.13 Reflection of a ray from a surface.

n is normalized; |n| = 1

$(\mathbf{a} \bullet \hat{\mathbf{n}})$ scalar

$(\mathbf{a} \bullet \hat{\mathbf{n}})\,\hat{\mathbf{n}}$ vector

Parallelogram rule

$$\mathbf{r} - \mathbf{a} = 2\,(-\mathbf{a} \bullet \hat{\mathbf{n}})\,\hat{\mathbf{n}}$$

$$\mathbf{r} = \mathbf{a} - 2(\mathbf{a} \bullet \hat{\mathbf{n}})\,\hat{\mathbf{n}}$$

Ideal Reflector r

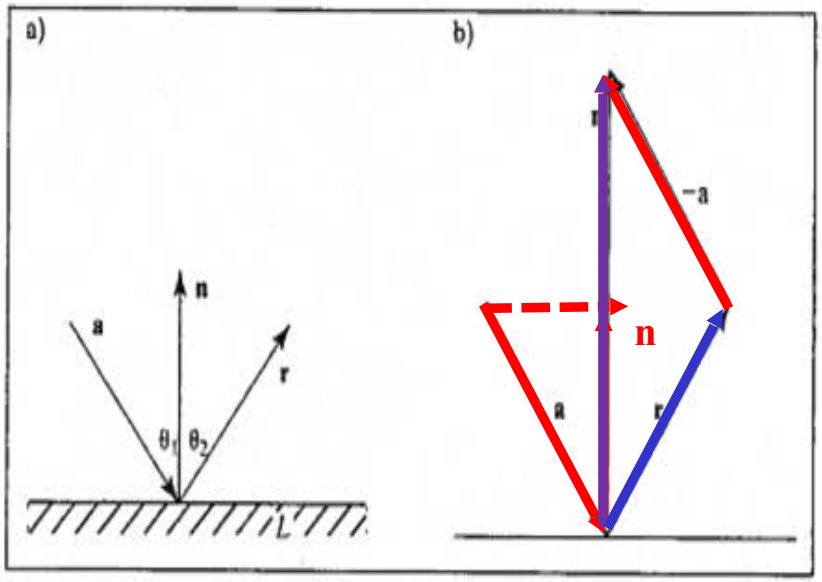


FIGURE 4.13 Reflection of a ray from a surface.

$$\mathbf{r} = \mathbf{a} - 2(\mathbf{a} \bullet \hat{\mathbf{n}})\,\hat{\mathbf{n}}$$

Let a = (4, -2) and **n** = (0, 3). What is **r**?

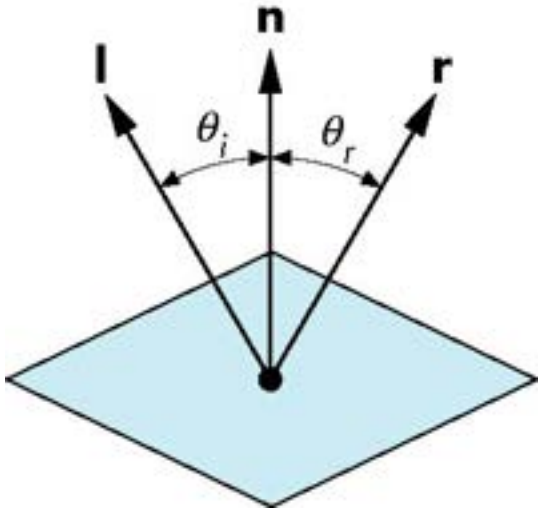
Normalize n $\hat{\mathbf{n}} = (0, 1)$

r = (4, -2) – 2 ((4, -2) •(0, 1)) (0, 1) = (4, -2) -2(-2) (0,1) = (4, -2) – (0, -4) = (4, 2)

Ideal Reflector r

- Normal vector **n** is determined by local orientation
- angle of incidence = angle of reflection
- The three vectors must be coplanar

$$\mathbf{r} = 2\,(\mathbf{l} \bullet \mathbf{n})\,\mathbf{n} - \mathbf{l}$$



CLASS PARTICIPATION 1!
(Next Slide)

Name: _____

Total score: ☐

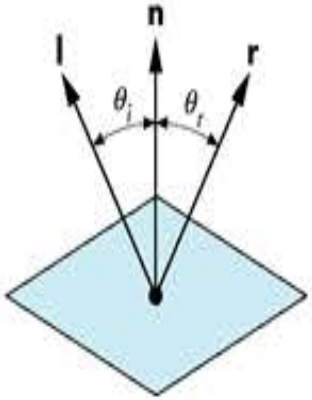
Class PARTICIPATION on Lecture 6.doc **ANSWER SHEET**

(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

1. (20 points) Draw and Calculate the reflection vector r when l = (-1,1) and n = (0,1)

Self Graded - correctly

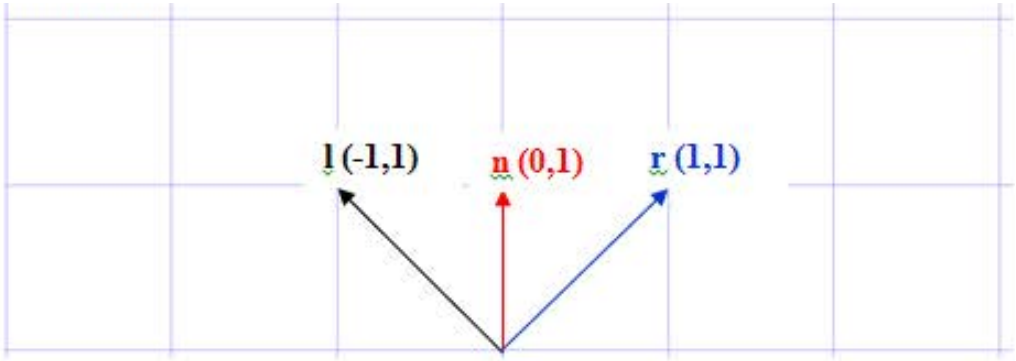


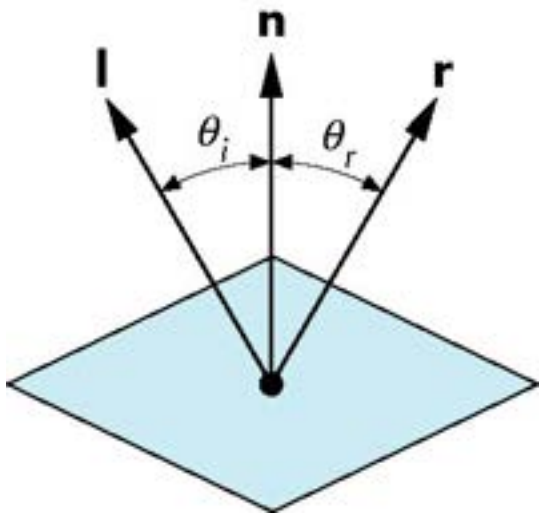
$$\mathbf{r} = 2(\mathbf{l} \bullet \mathbf{n})\,\mathbf{n} - \mathbf{l}$$

Ideal Reflector r

1. **Draw** and **Calculate** the reflection vector **r** when **l** = (-1,1) and **n** = (0,1)

$$\mathbf{r} = 2\,(\mathbf{l} \bullet \mathbf{n})\,\mathbf{n} - \mathbf{l}$$



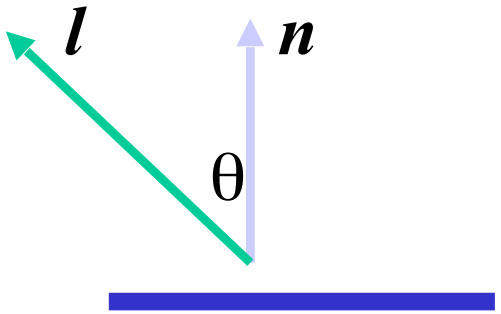


$$\mathbf{r} = 2\,[(-1,1) \bullet (0,1)]\,(0,1) - (-1,1) = 2\,1\,(0,1) - (-1,1) = (0,2) - (-1,1) = (1,1)$$

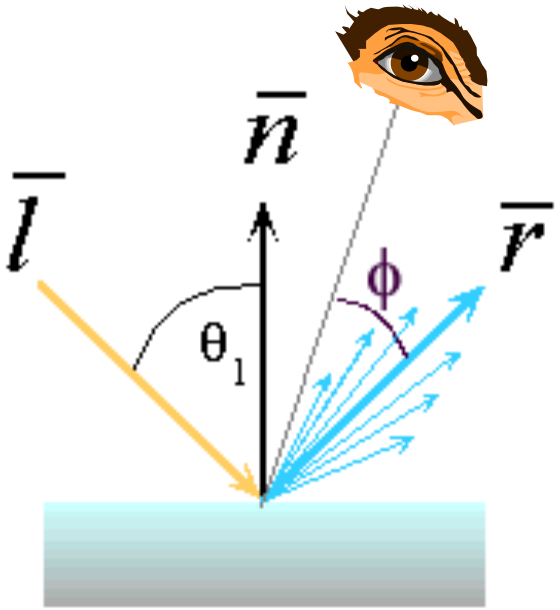
$$\mathbf{r} = (1,1)$$

Reflection Equations

$$I_{diffuse} = k_d I_{light} (\mathbf{n} \bullet \mathbf{l})$$



$$\mathbf{r} = 2 (\mathbf{l} \bullet \mathbf{n}) \mathbf{n} - \mathbf{l}$$



Reflectance Properties

LAMBERTIAN MODEL

$$\phi(N,S,I') = K_d \cos\theta$$

Diffuse

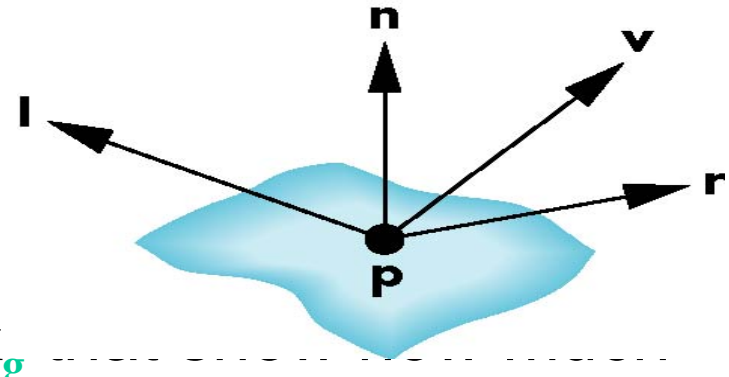
Lambertian Surface ?????

- Perfectly **diffuse** **reflector**
- **Light scattered equally in all directions**

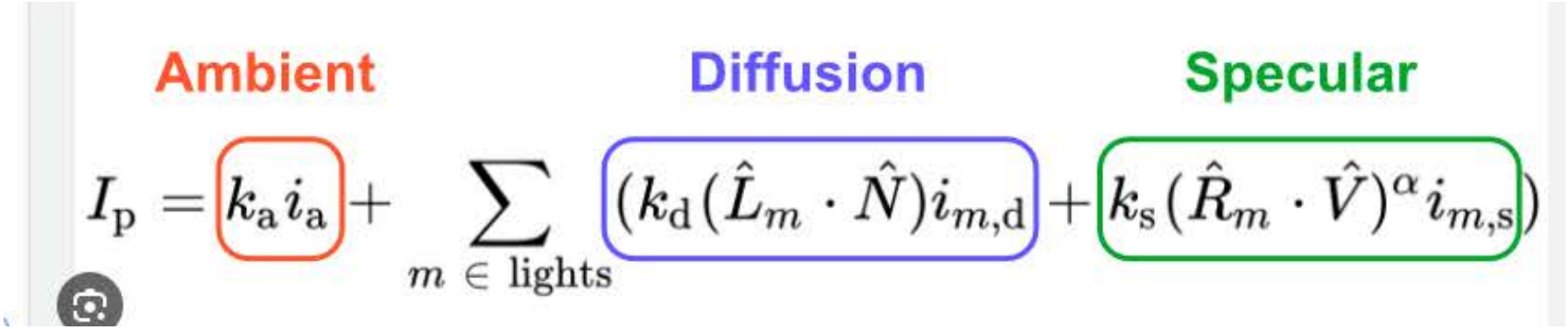
- Amount of light **reflected** is proportional to the vertical component of incoming light **l**

reflected light ~**cos** θ_i

$\cos \theta_i$ = **l** ● **n** if vectors **normalized**



There are also three coefficients, k_r, k_b, k_g of each **color component** is reflected


Ambient
Diffusion
Specular
$I_\mathrm{p} = k_\mathrm{a} i_\mathrm{a} + \sum_{m \in \text{lights}} \left(k_\mathrm{d} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_\mathrm{s} (\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}} \right)$

Reflectance Properties

Diffuse

Diffuse & Specular

LAMBERTIAN MODEL

$$\phi(N,S,V) = K_d \cos\theta$$

PHONG MODEL

$$I_\lambda = \phi(N,S,V) = K_d \cos\theta + K_s \cos^m \alpha$$

K_d=0.3, K_s=0.7, m=2; K_d=0.7, K_s=0.3, m=0.5

Diffuse

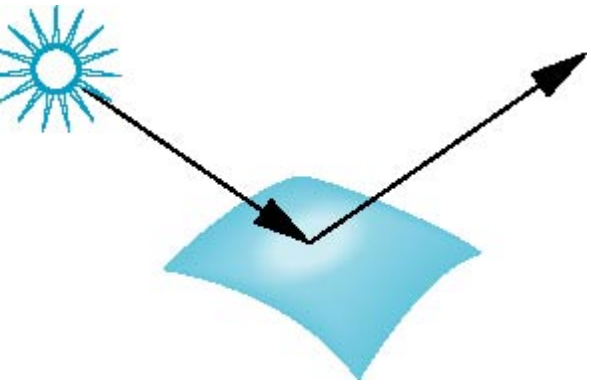
Diffuse 0.3 & Specular 0.7

Diffuse 0.7 & Specular 0.3

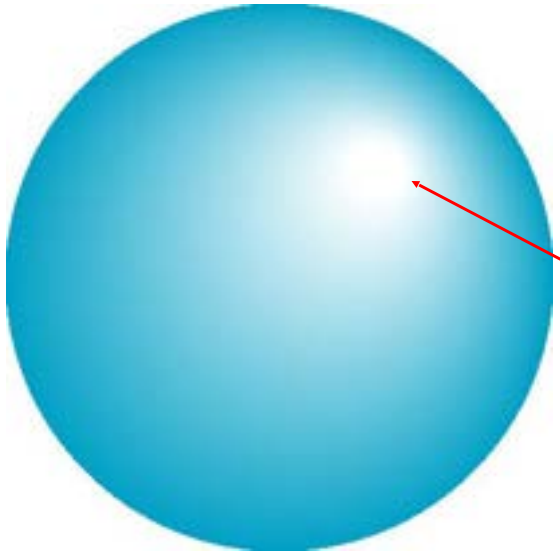
Specular Surfaces

- Most surfaces are neither ideal **Diffuse**rs nor perfectly **Specular** (ideal reflectors)
- **Smooth surfaces** show specular highlights due to incoming light being **reflected** in directions concentrated close to the direction of a **perfect reflection**

Specular reflections

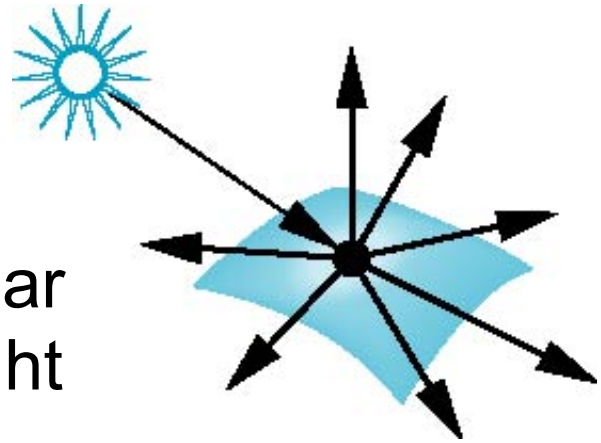


smooth surface



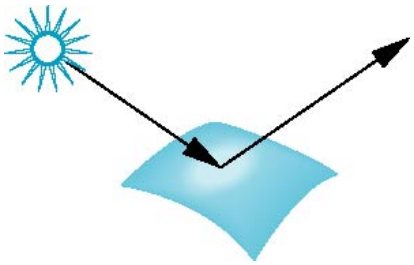
specular highlight

Diffuse reflections



rough surface

Modeling Specular Reflections



- **Phong** proposed using a **term that dropped off** as the **angle between** the **viewer v** and the ideal reflection **r** increased

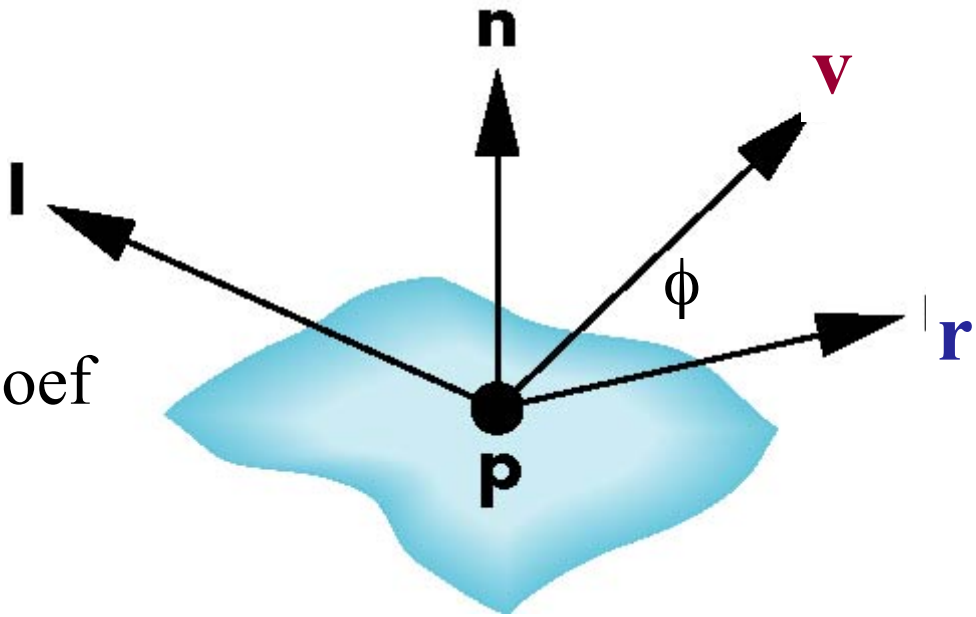
$$\mathbf{I}_r \sim k_s \mathbf{I} \cos^{\alpha}\phi$$

reflected intensity

absorption coef

incoming intensity

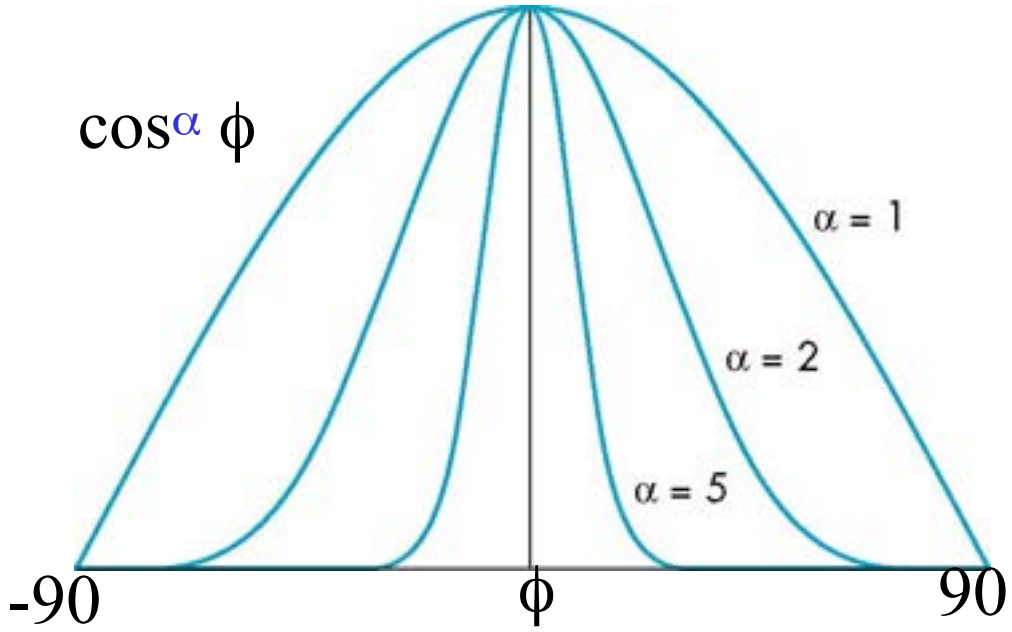
α shininess coef



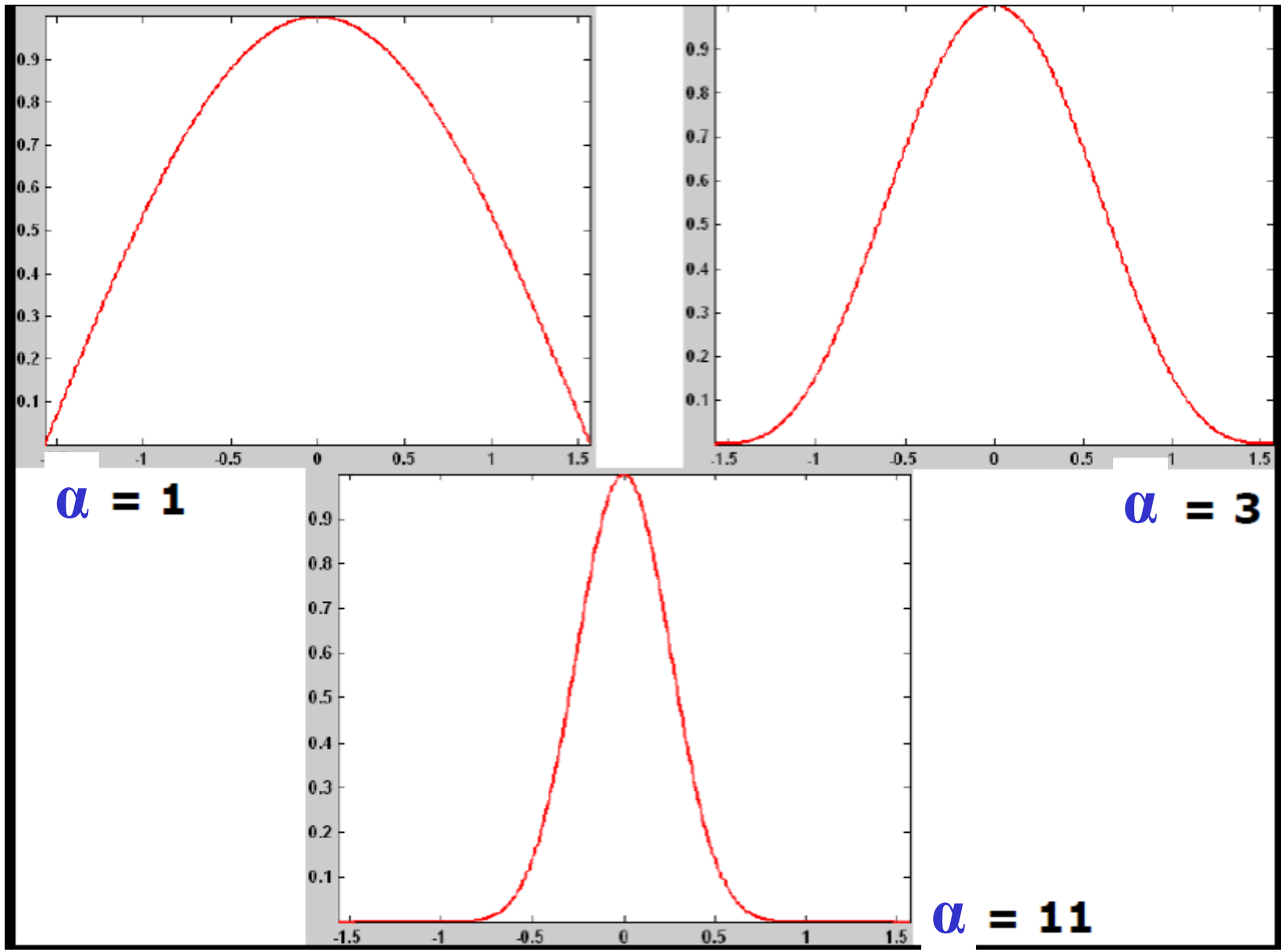
The Shininess Coefficient α

Values of α between 100 and 200 correspond to **metals**

Values between 5 and 10 give surface that look like **plastic**



The Shininess Coefficient α



Shading II

shading
deciding **pixel color**
continuum of realism: **when do we do lighting calculation**?

Objectives

- Continue discussion of **shading**
- Introduce **modified Phong model**
- Consider computation of required vectors

Ambient Light

- Ambient light is the result of multiple interactions between (large) light sources and the **objects** in the environment
- Amount and color depend on both the color of the light(s) and the material properties of the **object**

- Add $k_a I_a$ to **diffuse** and **specular** terms

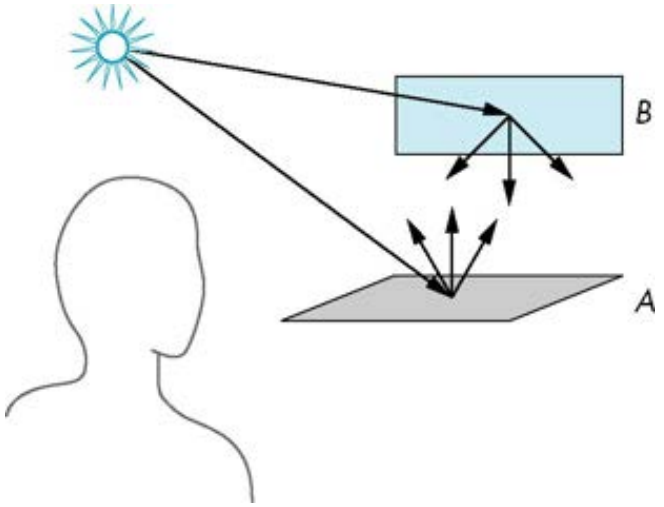
reflection coef

intensity of **ambient** light

Distance Terms

- The light from a **point source** that reaches a **surface** is inversely proportional to **the square of the** distance d^2 between them

- We can add a factor of the form $1/(ad + bd + cd^2)$ to the **diffuse** and **specular** terms



- The constants a, b, c and linear terms ad and bd soften the effect of the **point source**

Light Sources

- In the **Phong Model**, we add the results from each **light source**
- **Each light source has separate diffuse, specular, and ambient terms** to allow for maximum flexibility even though this form does not have a physical justification
- Separate red, green and blue components
- Hence, 9 coefficients for **each point source**

$$I_{dr}, I_{dg}, I_{db}, I_{sr}, I_{sg}, I_{sb}, I_{ar}, I_{ag}, I_{ab}$$

Material Properties

Material properties match light source properties

Nine **absorption coefficients**

- k_{dr}, k_{dg}, k_{db}, k_{sr}, k_{sg}, k_{sb}, k_{ar}, k_{ag}, k_{ab}

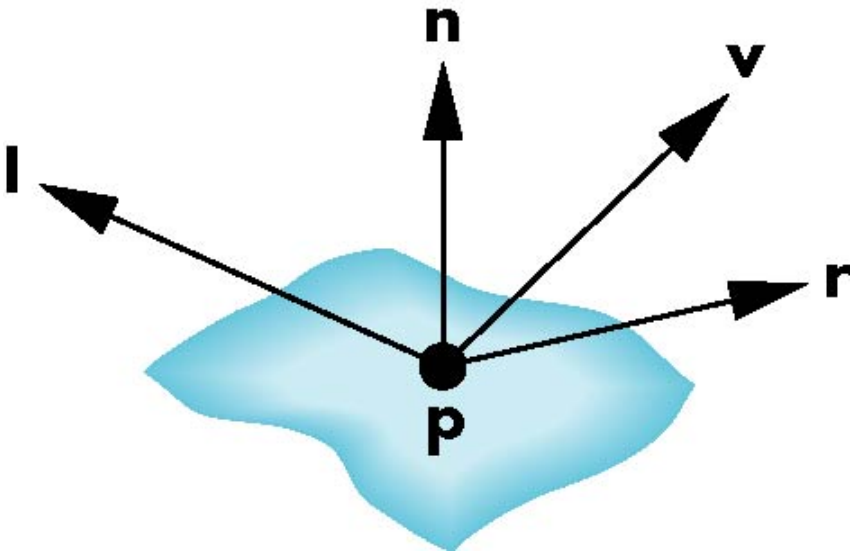
Shininess coefficient α

Adding up the Components

For **each light source** and **each color component**, the **Phong model** can be written (without the distance terms) as

$$I = k_d I_d \; \mathbf{l} \bullet \mathbf{n} + k_s I_s (\mathbf{v} \bullet \mathbf{r})^{\alpha} + k_a I_a$$

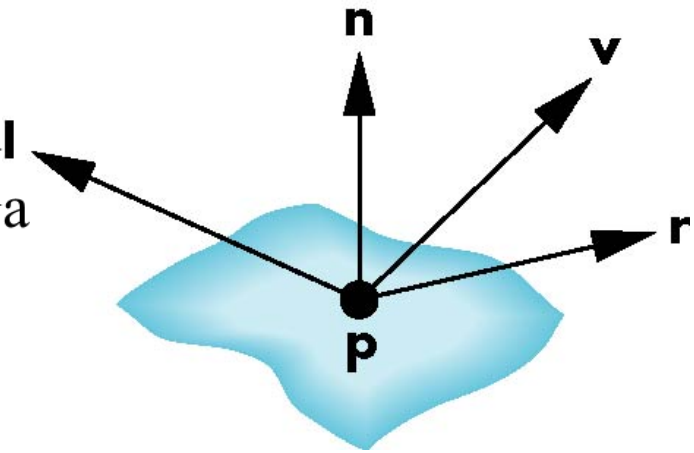
For each color component we add contributions from all sources



Modified Phong Model

The specular term in the Phong model is problematic because it requires the calculation of a new reflection vector r and view vector v for each vertex

$$I = k_d I_d \, l \bullet n + k_s I_s (v \bullet r)^{\alpha} + k_a I_a$$

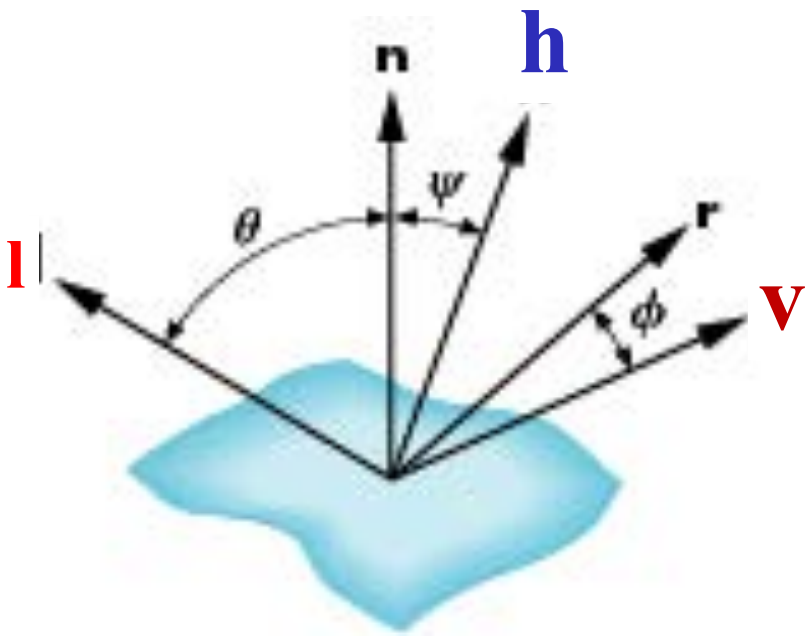


Blinn suggested an approximation using the halfway vector that is more efficient

The Halfway Vector h

h is **normalized vector** halfway between **l** and **v**

$$\mathbf{h} = (\mathbf{l} + \mathbf{v}) / |\mathbf{l} + \mathbf{v}|$$



Using the halfway vector h

- Replace $(\mathbf{v} \cdot \mathbf{r})^{\alpha}$ by $(\mathbf{n} \cdot \mathbf{h})^{\beta}$
- **β is chosen to match shininess**

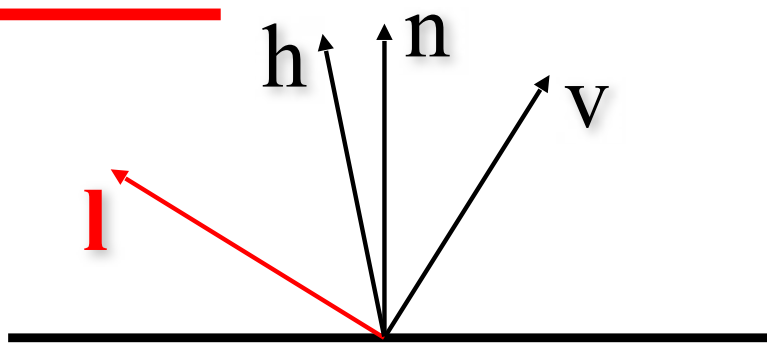
- Note that halfway angle is half of angle between **v** and **r** if vectors are **coplanar**

- Resulting model is known as the **modified Phong or Blinn lighting model**

  Specified in **OpenGL** standard

Reflection Equations

Blinn improvement

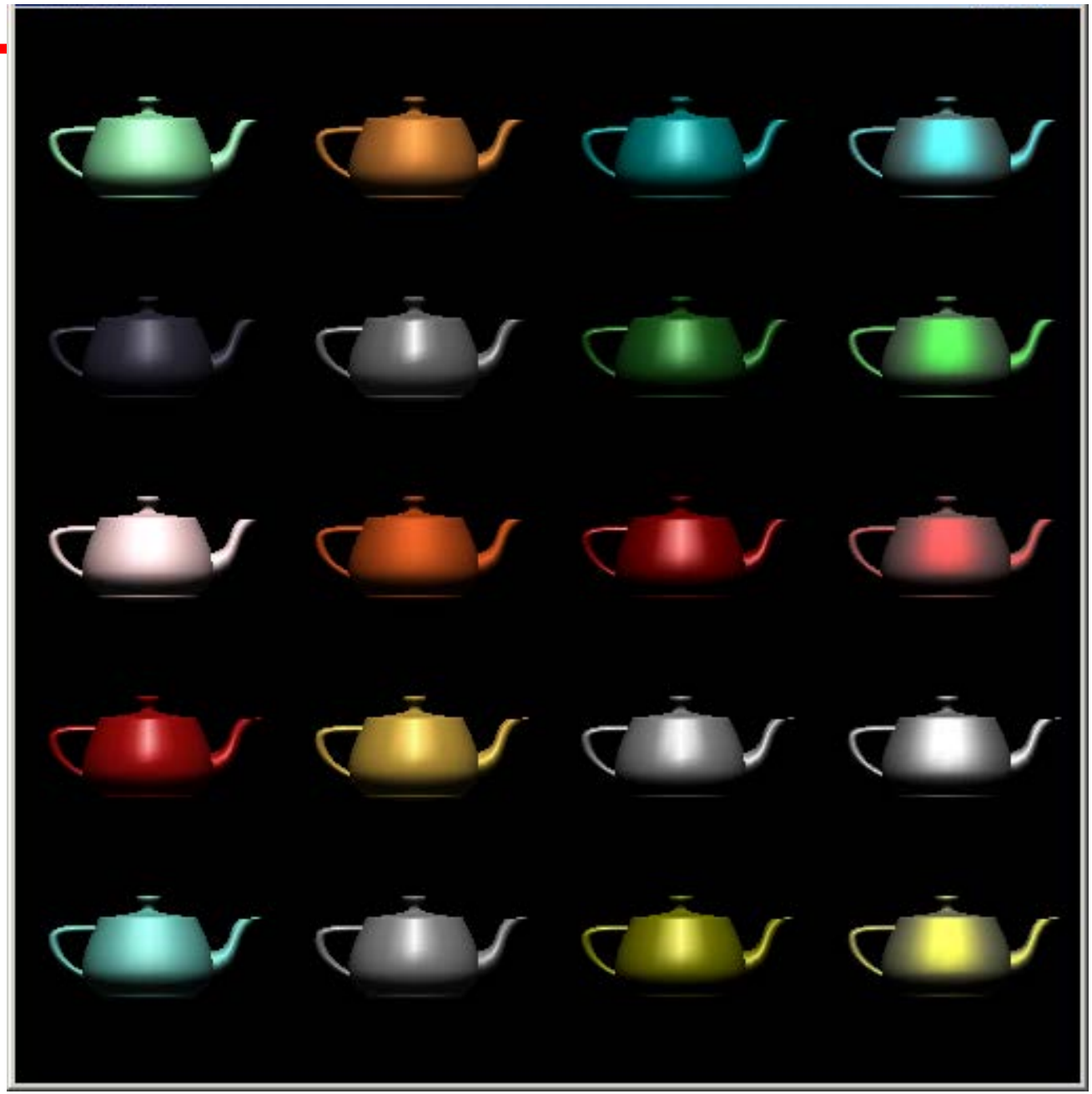


full Phong lighting model

combine ambient, diffuse, specular components

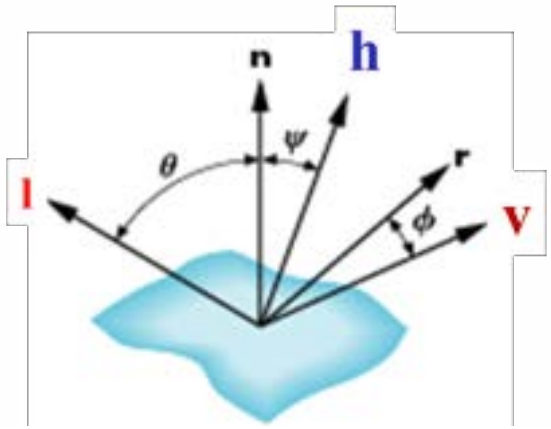
Example

Only differences in these teapots are the **parameters** in the **modified Phong model**



Computation of Vectors

l and **v** are specified by the **application**



Can computer **r** from **l** and **n**

Problem is determining n

For simple surfaces **n** can be determined but how we determine **n** differs depending on **underlying representation of surface**

OpenGL leaves determination of normal to application

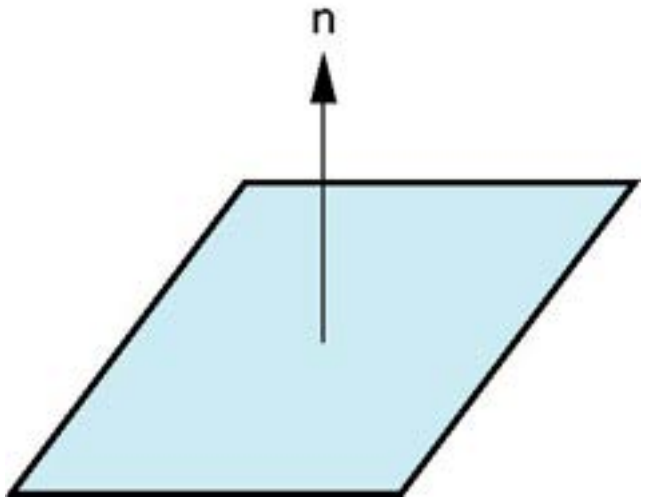
Exception for **GLU quadrics** and **Bezier surfaces** (Chapter 11)

Plane Normals

- Equation of plane: $ax + by + cz + d = 0$

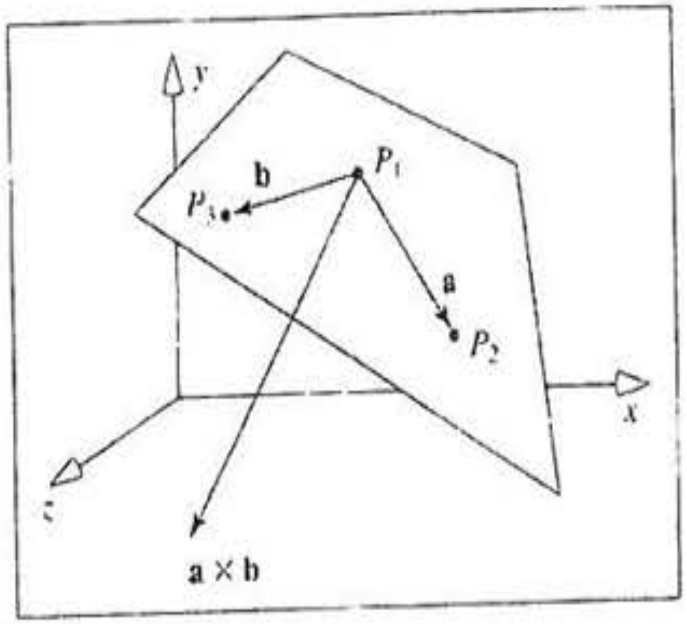
- We know that plane is determined by three points p_1, p_2, p_3

- Normal **n** can be obtained by ….



CLASS PARTICIPATION 2!
(Next Slide)

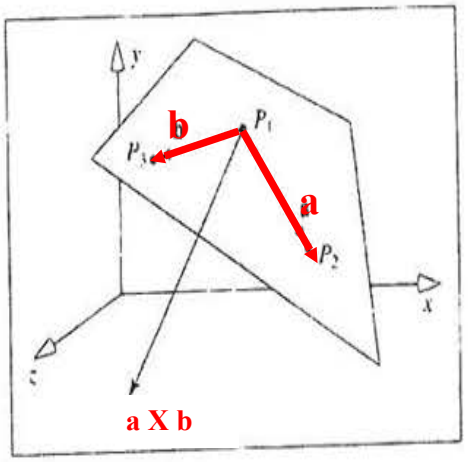
2. (20 points) Write the normal **n** to a plane defined by 3 points **P1**, **P2**, **P3** (same as normal **n** to a plane defined by two vectors **a** and **b**)



ANSWER:

Self Graded - correctly

Plane Normals

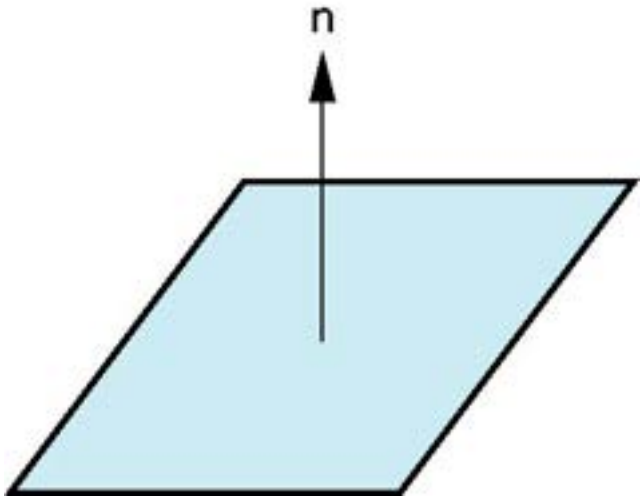


2. Write the normal **n** to a plane defined by 3 points P1, **P2, P3** (same as normal **n** to a plane defined by two vectors **a** and **b**)

- We know that plane is determined by three points p_1, p_2, p_3 or normal **n** and p_1

- Normal **n** can be obtained by

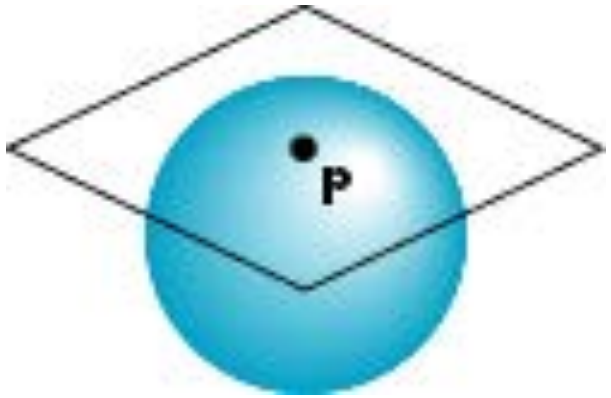
$$\mathbf{n} = (p_2 - p_1) \times (p_3 - p_1)$$



Normal to Sphere

- Implicit function **f(x,y.z)=0**
- Normal given by **gradient**
- Sphere **f(p)=p·p-1**

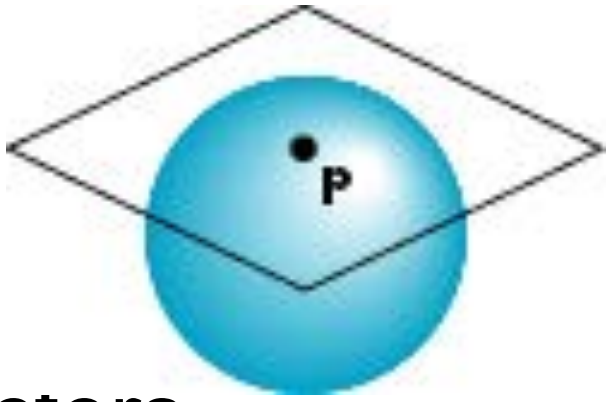
$$\mathbf{n} = [\partial f/\partial x, \partial f/\partial y, \partial f/\partial z]^T = \mathbf{p}$$



Parametric Form

- For sphere

$$\mathbf{x}=x(u,v)=\cos u \sin v$$
$$\mathbf{y}=y(u,v)=\cos u \cos v$$
$$\mathbf{z}=z(u,v)=\sin u$$



- Tangent plane determined by vectors

$$\partial \mathbf{p}/\partial \mathbf{u} = [\partial x/\partial u, \partial y/\partial u, \partial z/\partial u]T$$
$$\partial \mathbf{p}/\partial \mathbf{v} = [\partial x/\partial v, \partial y/\partial v, \partial z/\partial v]T$$

- **Normal n** given by **cross product**

$$\mathbf{n} = \partial \mathbf{p}/\partial u \times \partial \mathbf{p}/\partial v$$

General Case

We can compute **parametric normals** for other simple cases

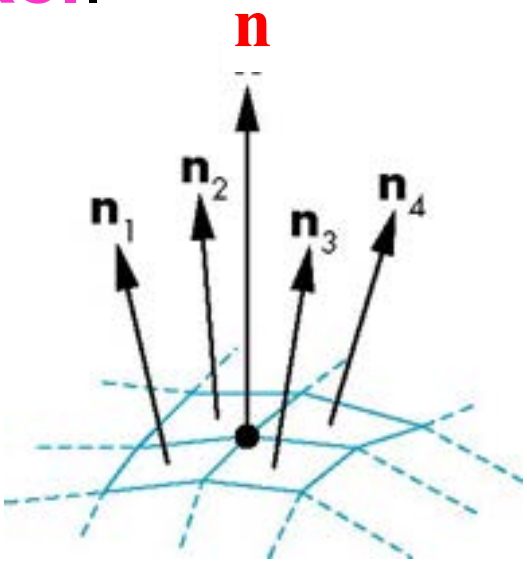
- Quadrics
- Parametric polynomial surfaces
  - Bezier surface patches (Chapter 11)

Gourad Shading

For polygonal models, Gouraud proposed we use the **average of the normals of the polygons that share the vertex**

The advantage is that **it is computationally the less expensive**, only requiring the evaluation of the intensity equation at the polygon **vertices**, and then **bilinear interpolation of these values for each pixel**.

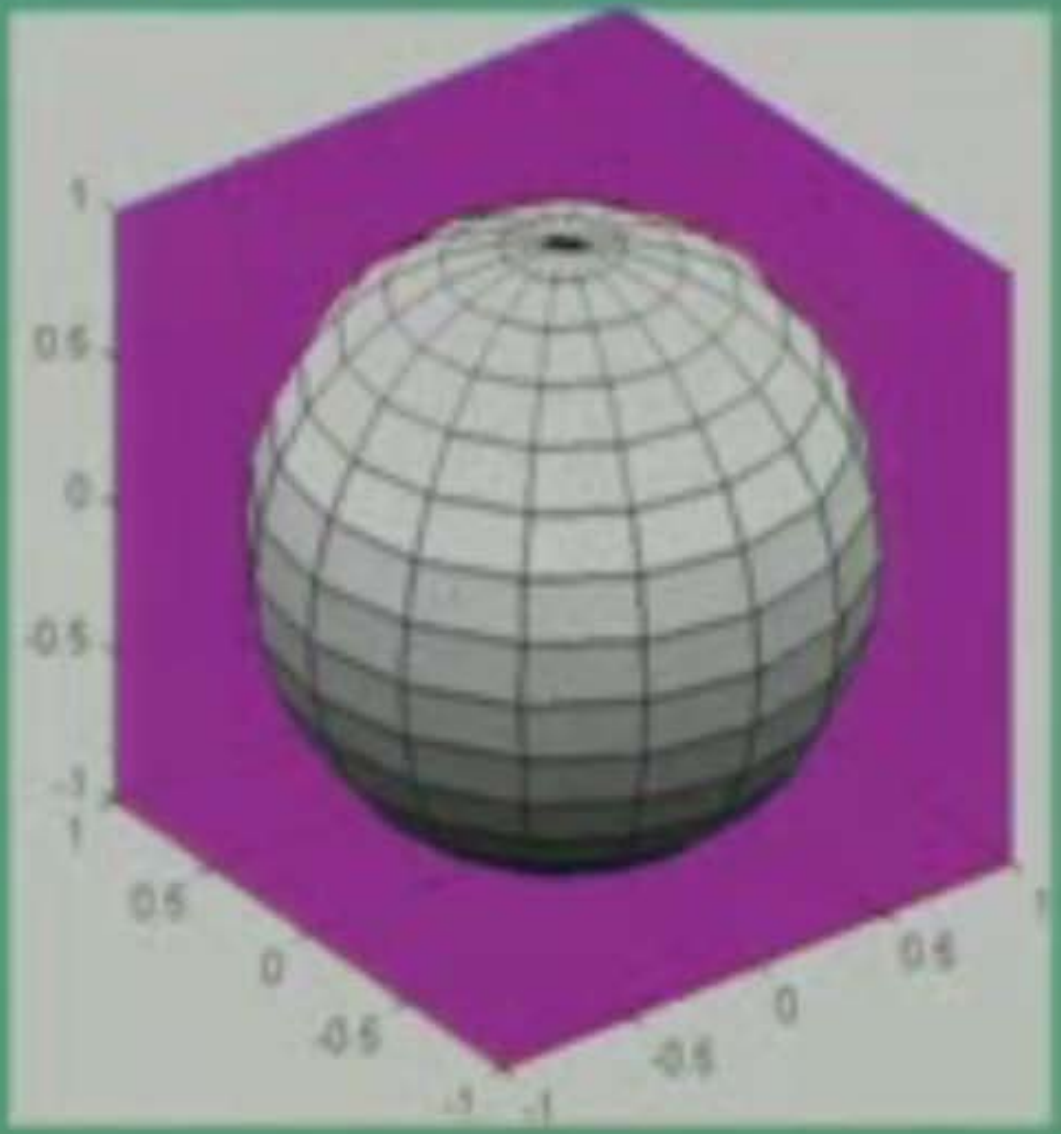
$$\mathbf{n} = (\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4)/\ |\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4|$$



Shading Model for Polygon

Constant Shading

- Also called *faceted shading, or flat shading.*



Illustrating the different shadings of a sphere.

Faceted

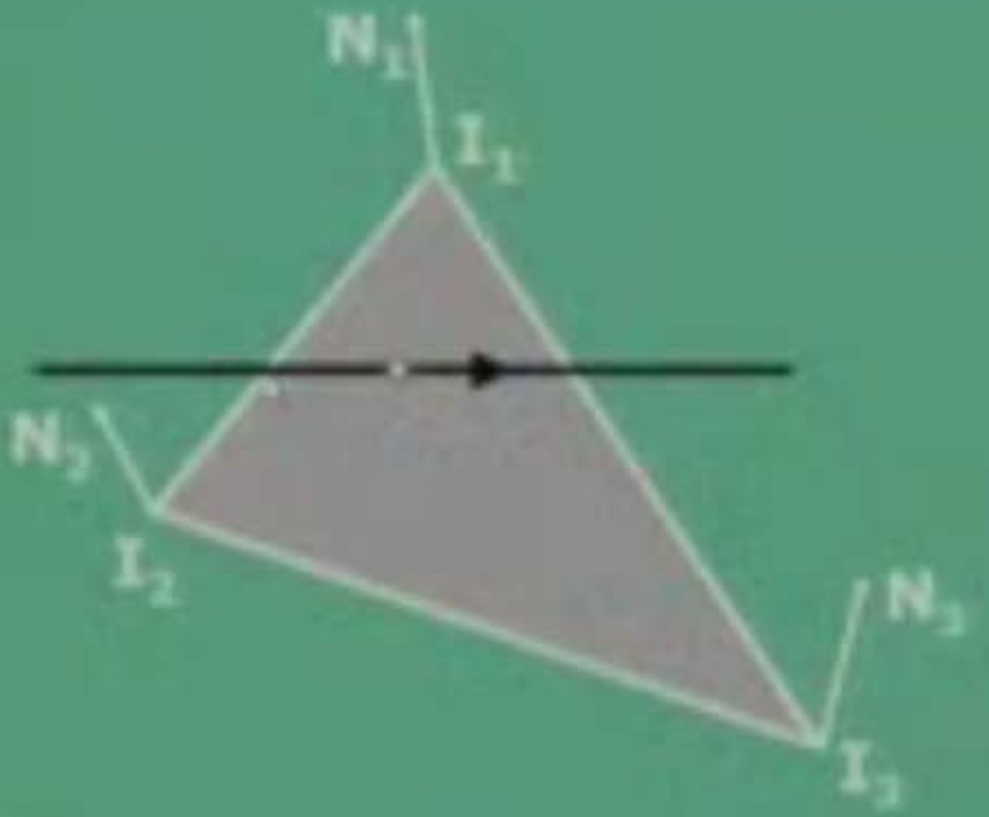
Gouraud/
Smooth/
Interpolated

Illustrating the different shadings of a sphere.

Faceted

Gouraud/
Smooth/
Interpolated

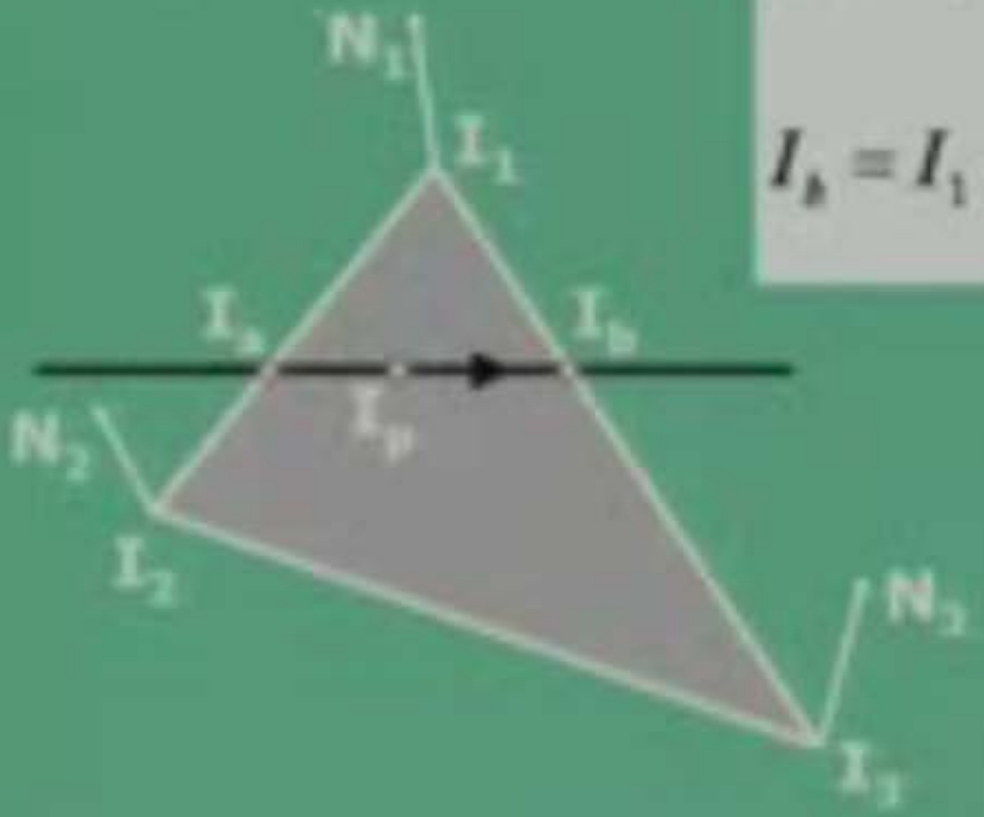
Illustrating the principle of Gouraud Shading or Color/Intensity Interpolation shading



Illustrating the principle of Gouraud Shading or Color/Intensity Interpolation shading

$$I_a = I_1 - (I_1 - I_2)\left(\frac{y_1 - y_a}{y_1 - y_2}\right);$$

$$I_b = I_1 - (I_1 - I_3)\left(\frac{y_1 - y_a}{y_1 - y_3}\right);$$

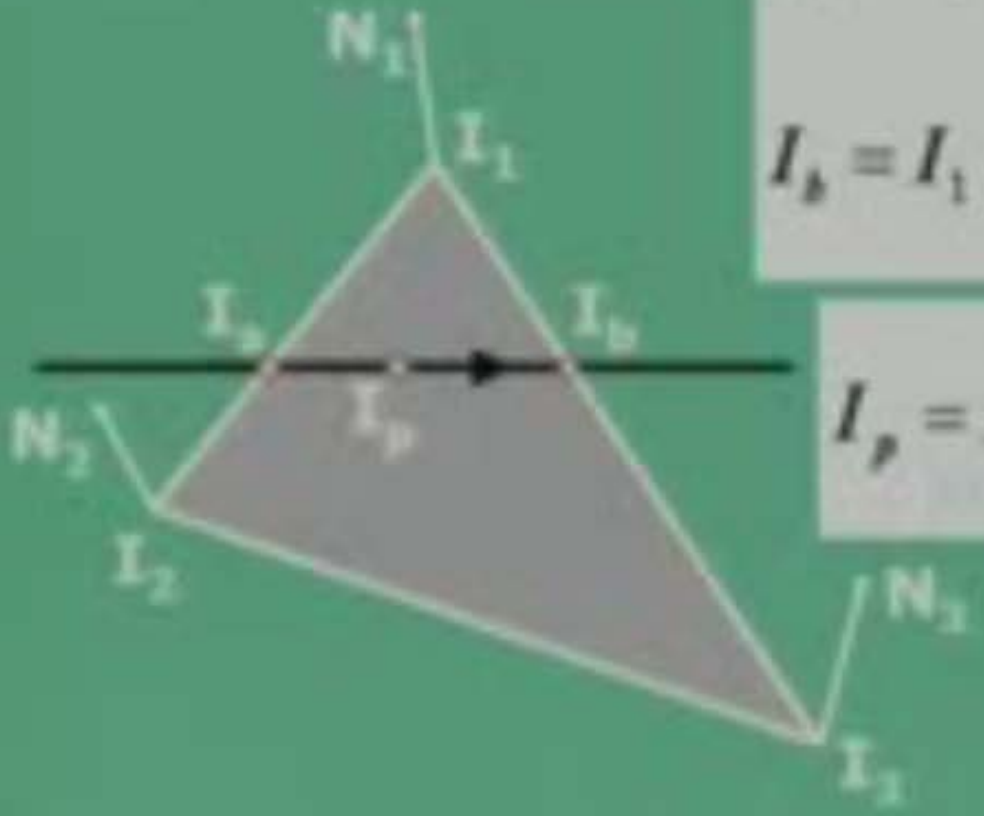


Illustrating the principle of Gouraud Shading or Color / Intensity Interpolation shading

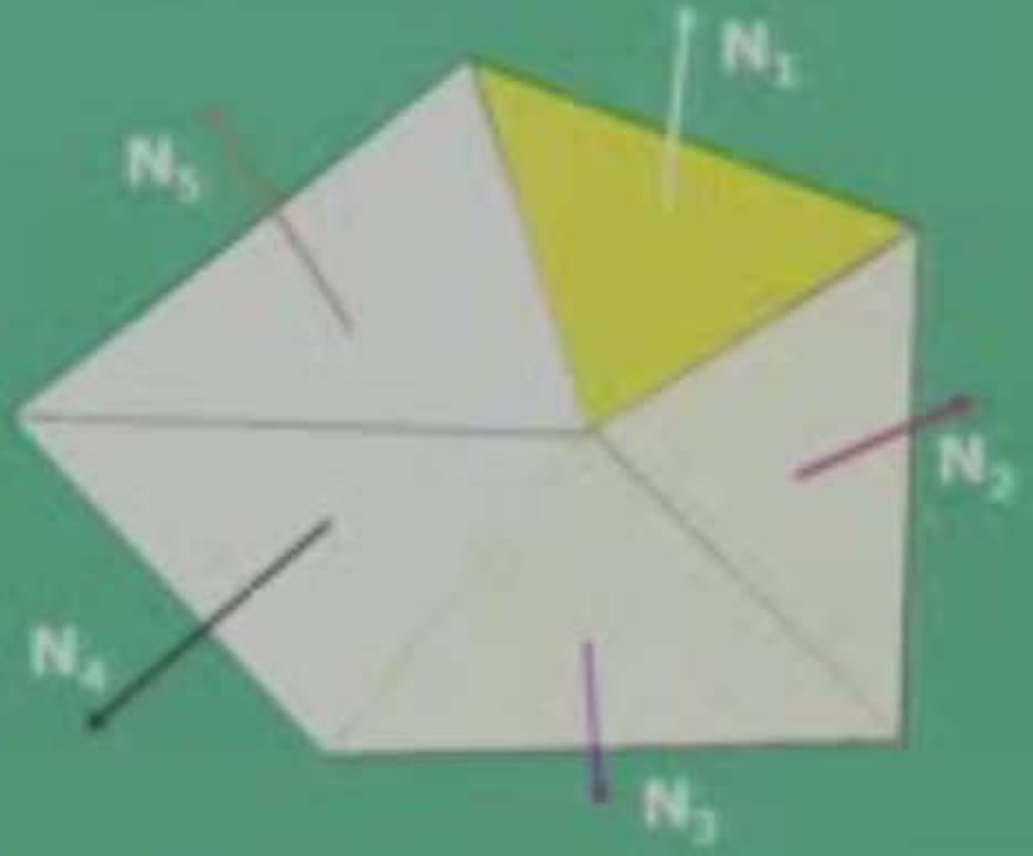
$$I_a = I_1 - (I_1 - I_2)\left(\frac{y_1 - y_a}{y_1 - y_2}\right);$$

$$I_b = I_1 - (I_1 - I_3)\left(\frac{y_1 - y_a}{y_1 - y_3}\right);$$

$$I_p = I_b - (I_b - I_a)\left(\frac{x_b - x_p}{x_b - x_a}\right)$$



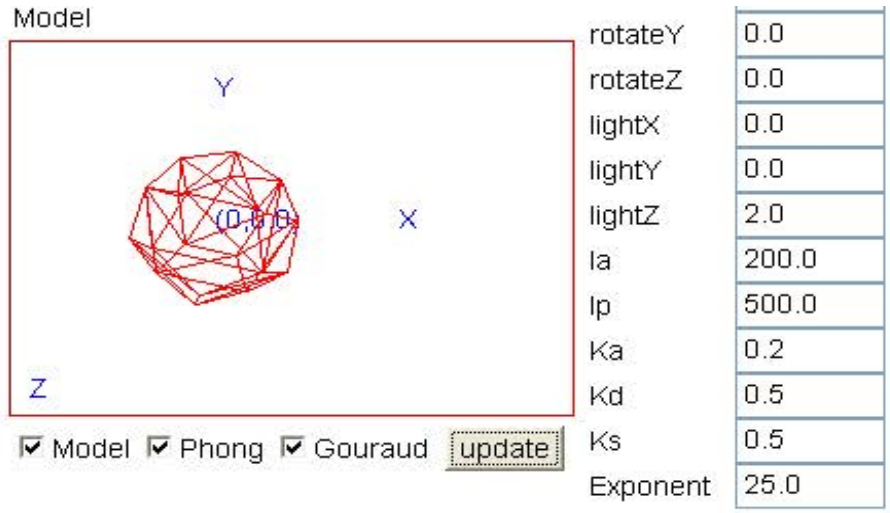
Illustrating the principle of Gouraud Shading or Color/Intensity Interpolation shading

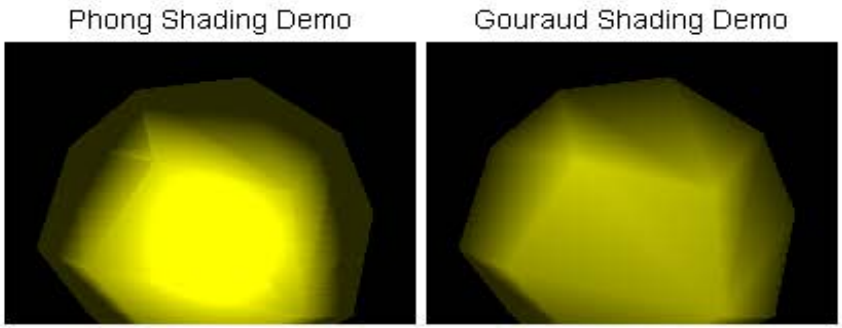


Illustrating the difference in shadings in case of a tea-pot.

Shading - Windows Internet Explorer

Phong Shading vs Gouraud Shading





http://www.nbb.cornell.edu/neurobio/land/OldStudentProjects/cs490-95to96/guo/report.html

Shading in OpenGL
Chapter 6 .7

Objectives

- Introduce the **OpenGL** shading functions

- Discuss **polygonal shading**
  - Flat
  - Smooth
  - Gouraud

Steps in OpenGL shading

1. Enable **shading** and select **model**
2. Specify **normals**
3. Specify **material properties**
4. Specify **lights**

Normals

- In **OpenGL** the **normal vector** is **part of the state**

- Set by **glNormal*()**

```
glNormal3f(x, y, z);
glNormal3fv(p);
```

- Usually we want to **set the normal** to have **unit length** so cosine calculations are correct

  Length can be affected by transformations

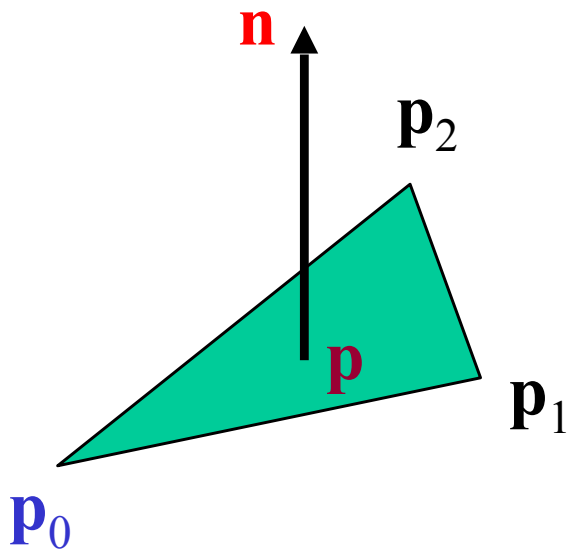
  Note that scaling does not preserved **length**

glEnable(GL_NORMALIZE) allows for **auto normalization** at a performance penalty

Normal for Triangle

```
void normal(point p)
{
   /* normalize a vector */
    double sqrt();
    float d =0.0;
    int i;
    for(i=0; i<3; i++) d+=p[i]*p[i];
    d=sqrt(d);
    if(d>0.0) for(i=0; i<3; i++) p[i]/=d;
}
```

Express **n** as a dot product ● and cross product ×

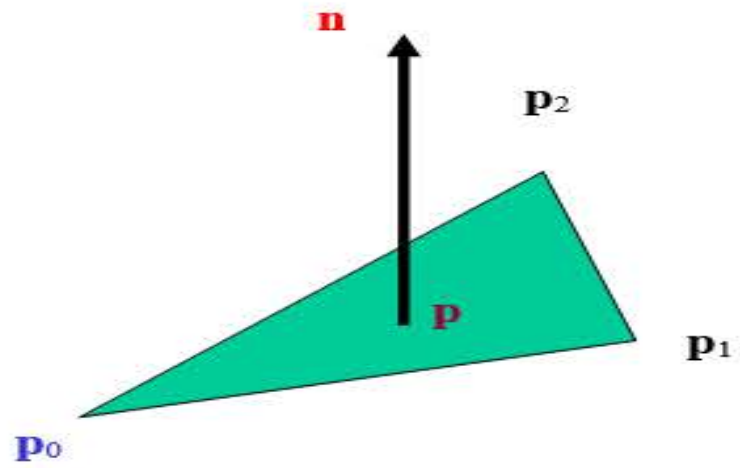


Normalize **n**

Note that right-hand rule determines outward face

CLASS PARTICIPATION 3, 4!
(Next Slide)

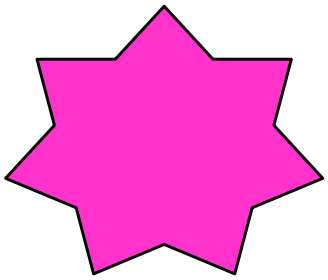
3. (20 points) Express the normal **n** to a triangle both as a dot product • and as cross product X.



ANSWER:

Self Graded - correctly

4. (20 points) Normalize **n** to $\hat{\mathbf{n}}$



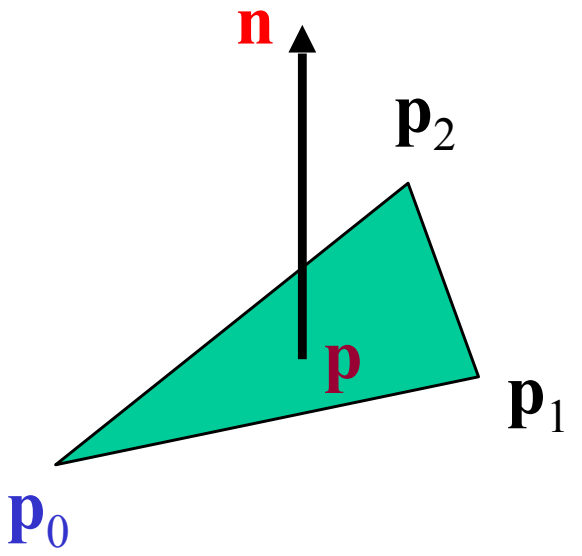
Normal for Triangle

```
void normal(point p)
{
   /* normalize a vector */
    double sqrt();
    float d =0.0;
    int i;
    for(i=0; i<3; i++) d+=p[i]*p[i];
    d=sqrt(d);
    if(d>0.0) for(i=0; i<3; i++) p[i]/=d;
}
```

Express $\mathbf{n}$ as a dot product $\bullet$ and cross product $\times$

plane $\quad \mathbf{n} \bullet (\mathbf{p} - \mathbf{p}_0) = 0$

$\mathbf{n} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0)$



Normalize $\mathbf{n}$

normalize $\hat{\mathbf{n}} \leftarrow \mathbf{n} / |\mathbf{n}|$

Note that right-hand rule determines outward face

5. (20 points) Create **Lecture6** Empty Project:

Download SphereShading.c from CANVAS.

Class APRTICIPATION on Lecture 6

Publish | Edit

Download and complete this word document.

Class Participation on Lecture 6.doc

SphereShading.c

You will be prompted when to Upload completed document to CANVAS as score.doc (example 100.doc).

Warning:

TA, at random, will inspect the Uploaded document.

If you score is not honestly entered, you will get a zero.

For the Grader: if the student did not submit, please skip and do not assign a ZERO (MISSING).

Self Graded - correctly

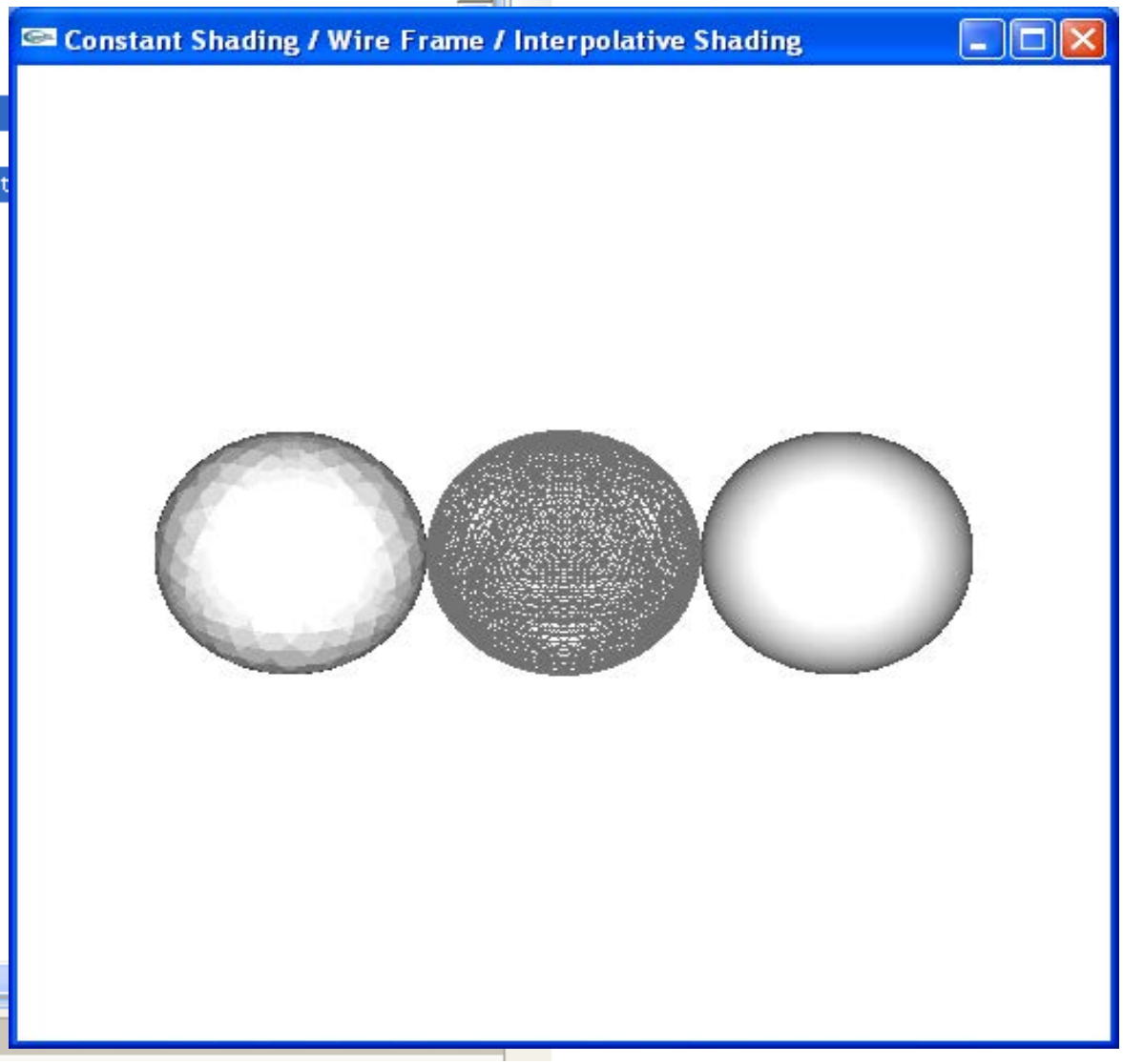
Comparison

Constant Shading/ Wire Frame/ Interpolative Shading

Debug Win32

ing.c*

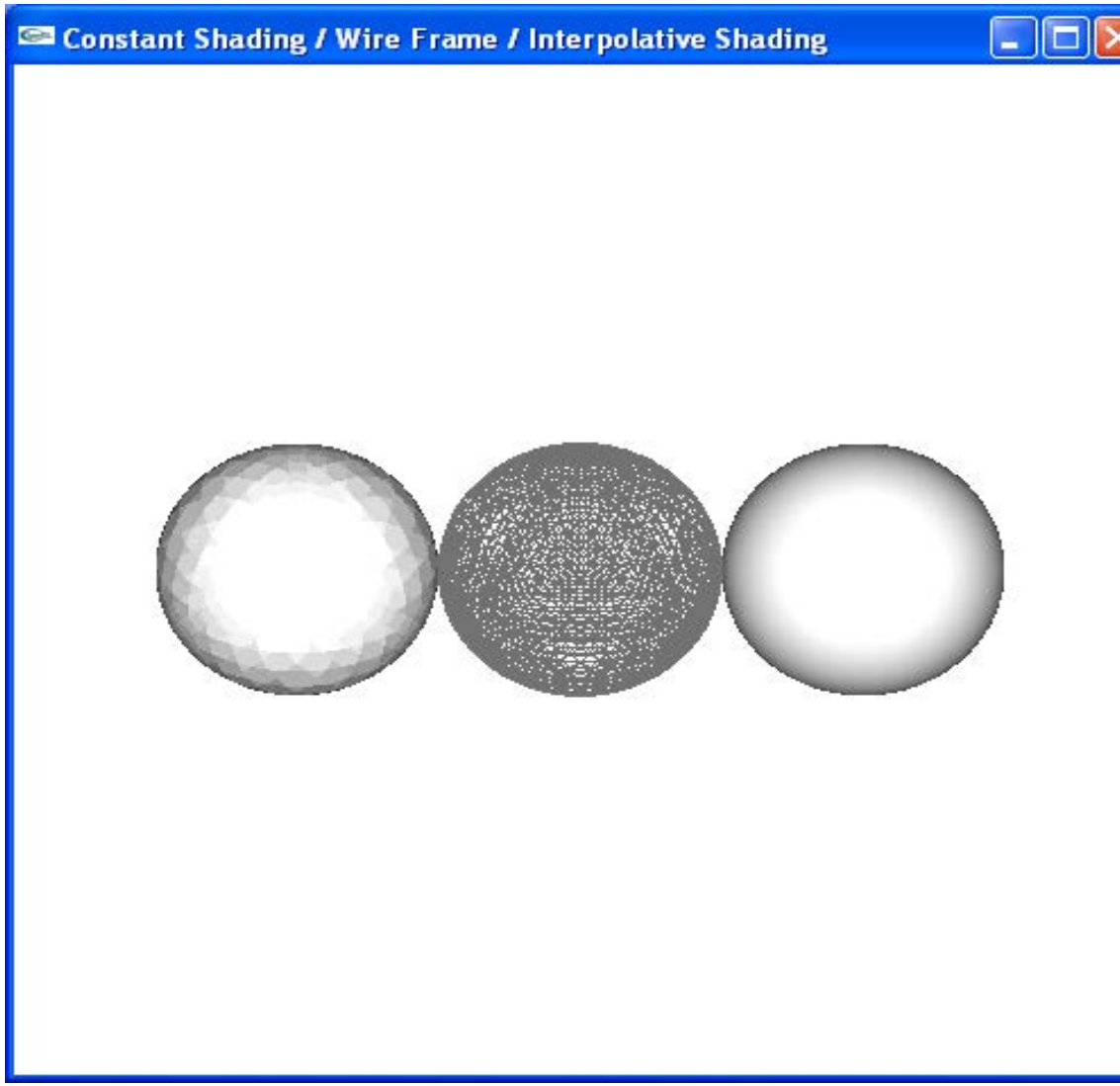
```
* SphereShading.c */
splay(void)

displays all three modes, side by side */
lear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
oadIdentity();

node 0 = wire frame*/
e=0;
rahedron(n);

mode 1 = constant shading */
e=1;
ranslatef(-2.0, 0.0,0.0);
rahedron(n);

node 2 = interpolative shading */
e=2;
ranslatef( 4.0, 0.0,0.0);
rahedron(n);

lush();
```



Comparison

Constant Shading/ Wire Frame/ Interpolative Shading

```
void triangle( point a, point b, point c)

/* display one triangle using a line loop for wire frame, a single
normal for constant shading, or three normals for interpolative shading */
{
     if (mode==0)
         glBegin(GL_LINE_LOOP);
     else
        glBegin(GL_POLYGON);
             if(mode==1) glNormal3fv(a);
             if(mode==2) glNormal3fv(a);
             glVertex3fv(a);
             if(mode==2) glNormal3fv(b);
             glVertex3fv(b);
             if(mode==2) glNormal3fv(c);
             glVertex3fv(c);
        glEnd();
}
```



Enabling Shading

Shading calculations are enabled by

`glEnable(GL_LIGHTING)`

Once lighting is enabled, **glColor**() ignored

Must enable each light source individually

`glEnable(GL_LIGHTi)` i=0,1…..

Can choose light model parameters

`glLightModeli(parameter, GL_TRUE)`

- `GL_LIGHT_MODEL_LOCAL_VIEWER` do not use simplifying distant viewer assumption in calculation

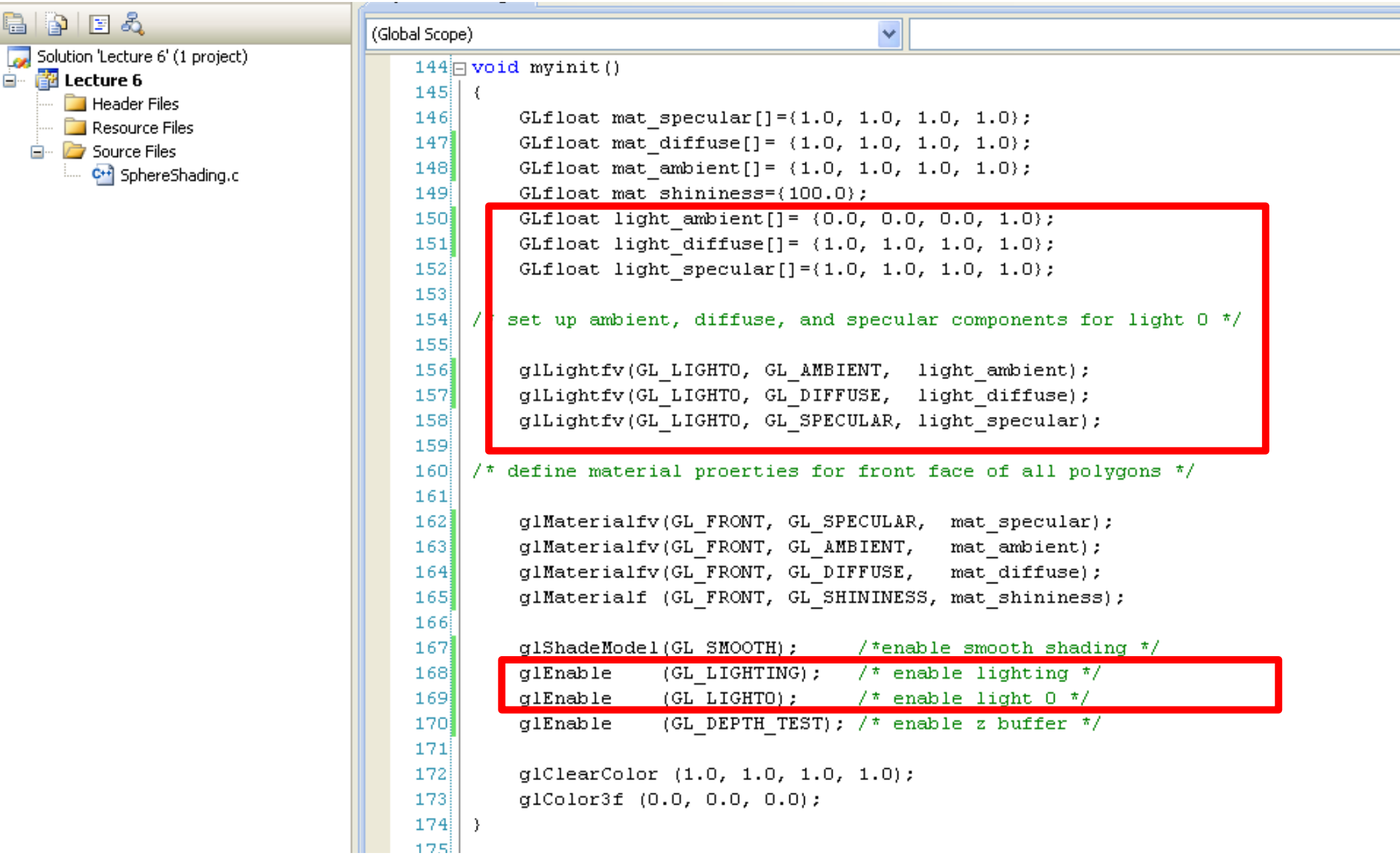
- `GL_LIGHT_MODEL_TWO_SIDED` shades both sides of polygons independently

Defining a Point Light Source

For **each light source**, we can set an RGB**A** for the diffuse, specular, and ambient components, and for the position

```
GL float diffuse0[]   ={1.0, 0.0, 0.0, 1.0};
GL float specular0[]  ={1.0, 0.0, 0.0, 1.0};
GL float ambient0[]   ={1.0, 0.0, 0.0, 1.0};
Glfloat  light0_pos[] ={1.0, 2.0, 3,0, 1.0};

            glEnable(GL_LIGHTING);
             glEnable(GL_LIGHT0);
glLightv(GL_LIGHT0, GL_POSITION, light0_pos);
 glLightv(GL_LIGHT0, GL_AMBIENT,  ambient0);
 glLightv(GL_LIGHT0, GL_DIFFUSE,  diffuse0);
glLightv(GL_LIGHT0, GL_SPECULAR, specular0);
```

Defining a Point Light Source




```
144 void myinit()
145 {
146     GLfloat mat_specular[]={1.0, 1.0, 1.0, 1.0};
147     GLfloat mat_diffuse[]= {1.0, 1.0, 1.0, 1.0};
148     GLfloat mat_ambient[]= {1.0, 1.0, 1.0, 1.0};
149     GLfloat mat_shininess={100.0};
150     GLfloat light_ambient[]= {0.0, 0.0, 0.0, 1.0};
151     GLfloat light_diffuse[]= {1.0, 1.0, 1.0, 1.0};
152     GLfloat light_specular[]={1.0, 1.0, 1.0, 1.0};
153
154 /* set up ambient, diffuse, and specular components for light 0 */
155
156     glLightfv(GL_LIGHT0, GL_AMBIENT,  light_ambient);
157     glLightfv(GL_LIGHT0, GL_DIFFUSE,  light_diffuse);
158     glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
159
160 /* define material proerties for front face of all polygons */
161
162     glMaterialfv(GL_FRONT, GL_SPECULAR,  mat_specular);
163     glMaterialfv(GL_FRONT, GL_AMBIENT,   mat_ambient);
164     glMaterialfv(GL_FRONT, GL_DIFFUSE,   mat_diffuse);
165     glMaterialf (GL_FRONT, GL_SHININESS, mat_shininess);
166
167     glShadeModel(GL_SMOOTH);      /*enable smooth shading */
168     glEnable     (GL_LIGHTING);   /* enable lighting */
169     glEnable     (GL_LIGHT0);     /* enable light 0 */
170     glEnable     (GL_DEPTH_TEST); /* enable z buffer */
171
172     glClearColor (1.0, 1.0, 1.0, 1.0);
173     glColor3f (0.0, 0.0, 0.0);
174 }
175
```

Distance and Direction

• We can add a factor of the form $1/(ad + bd + cd^2)$ to the diffuse and specular terms

The source colors are specified in RGB**A**

The position is given in homogeneous coordinates

If w =1.0, we are specifying a finite location

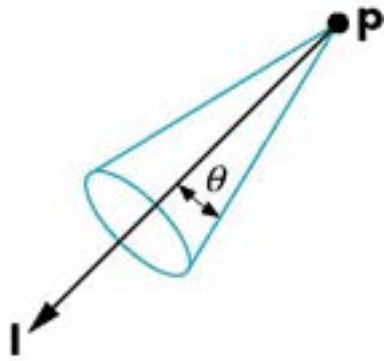
If **w =0.0**, we are specifying a parallel source with the given **direction vector**

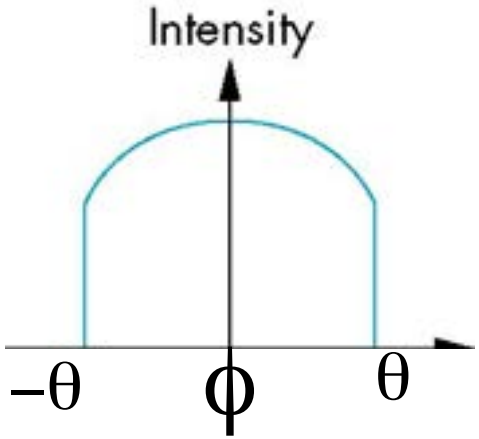
The coefficients in the **distance terms** are by default **a=1.0 (constant terms), b=c=0.0 (linear and quadratic terms).**

```
a= 0.80;
glLightf(GL_LIGHT0, GLCONSTANT_ATTENUATION, a);
```

Spotlights

- Use `glLightv` to set
  - Direction `GL_SPOT_DIRECTION`
  - Cutoff `GL_SPOT_CUTOFF`
  - Attenuation `GL_SPOT_EXPONENT`
    - Proportional to $\cos^{\alpha}\phi$





Global Ambient Light

- **Ambient light** depends on color of light sources

  A **red light** in a white room will cause a red ambient term that disappears when the light is turned off

- **OpenGL** also allows a global ambient term that is often helpful for testing

```
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient)
```

Moving Light Sources

- **Light sources** are geometric objects whose positions or directions are affected by the **model-view** matrix
- Depending on where we place the position (direction) setting function, we can
  - Move the **light source(s)** with the **object**(s)
  - Fix the **object**(s) and move the **light source(s)**
  - Fix the **light source(s)** and move the **object**(s)
  - Move the **light source(s)** and **object**(s) independently

Material Properties

- Material properties are also part of the **OpenGL** state and match the terms in the **modified Phong model**
- Set by `glMaterialv()`

```
GLfloat ambient[]  = {0.2, 0.2, 0.2, 1.0};
GLfloat diffuse[]  = {1.0, 0.8, 0.0, 1.0};
GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};
          GLfloat shine = 100.0
glMaterialf(GL_FRONT, GL_AMBIENT,   ambient);
glMaterialf(GL_FRONT, GL_DIFFUSE,   diffuse);
glMaterialf(GL_FRONT, GL_SPECULAR,  specular);
 glMaterialf(GL_FRONT, GL_SHININESS, shine);
```

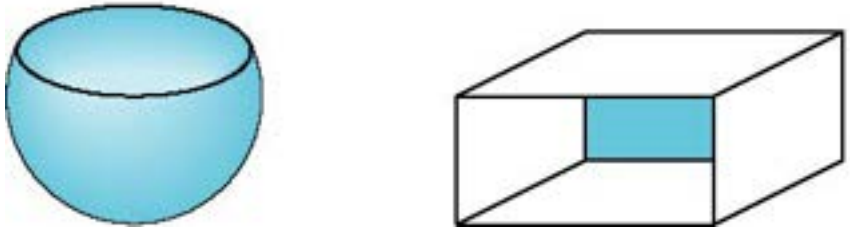
Front and Back Faces

The default is shade **only front faces** which works correctly for convex objects

If we set two sided lighting, **OpenGL** will shade both sides of a surface

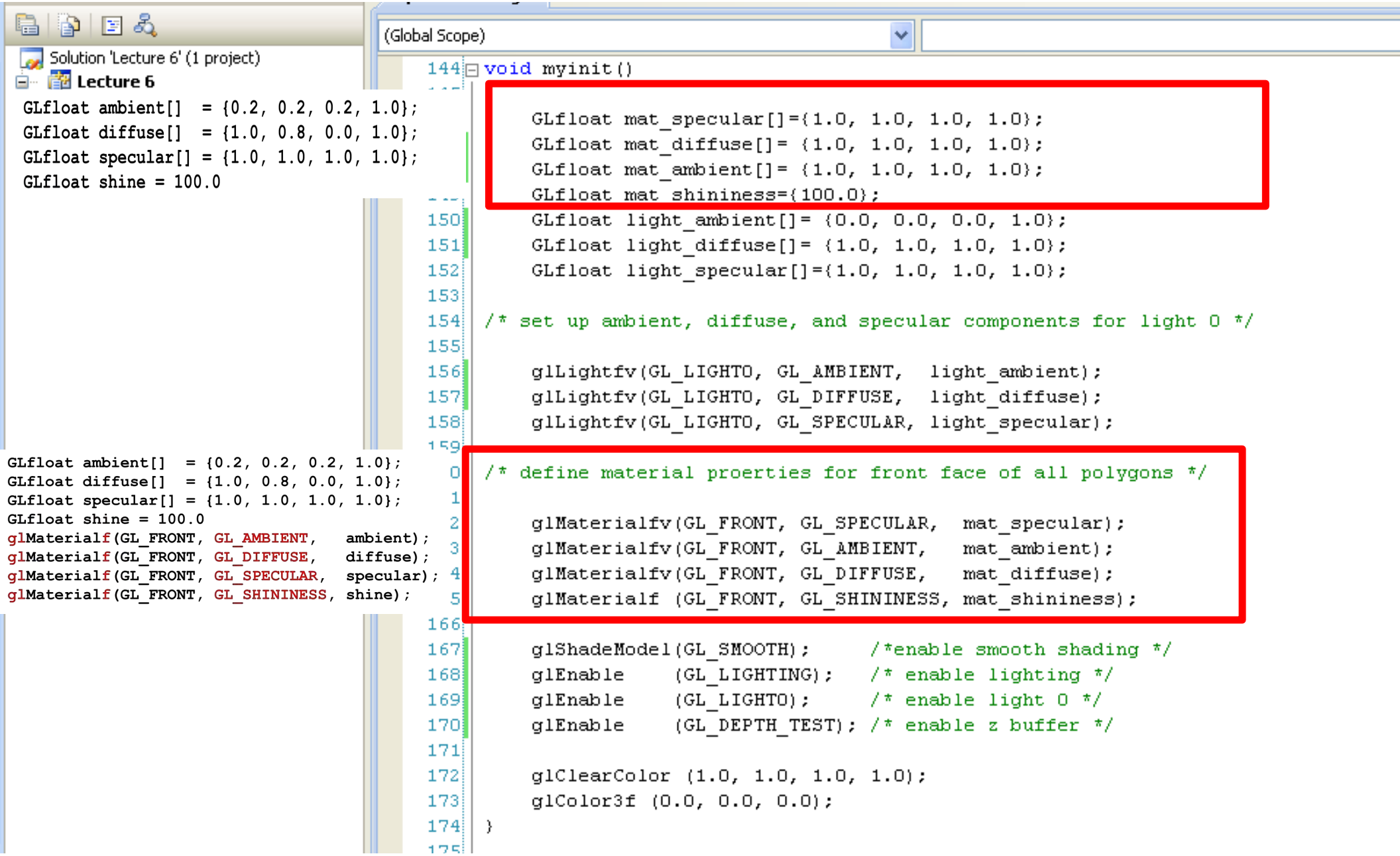
Each side can have its own properties which are set by using `GL_FRONT`, `GL_BACK`, or `GL_FRONT_AND_BACK` in `glMaterialf`

back faces not visible



back faces visible

Defining Material Properties



Emissive Term

- We can simulate a **light source** in **OpenGL** by giving a material an emissive component

- **This component is unaffected by any sources or transformations**

```
  GLfloat emission[] = (0.0, 0.3, 0.3, 1.0);
glMaterialf(GL_FRONT, GL_EMISSION, emission);
```

Transparency

- Material properties are specified as RGB**A** values
- The **A value** can be used to make the surface translucent
- **The default is that all surfaces are opaque** regardless of **A**
- Later we will **enable blending** and use this feature

Efficiency

- Because material properties are part of the state, if we **change materials for many surfaces**, we can affect performance

- We can make the code cleaner by defining a material structure and setting all materials during initialization

```
typedef struct materialStruct {
                GLfloat ambient[4];
                GLfloat diffuse[4];
               GLfloat specular[4];
                GLfloat shineness;
          } MaterialStruct;
```

- We can then select a **material by a pointer**

Polygonal Shading

- **Shading calculations** are done for **each vertex**
  Vertex colors become **vertex shades**

- By default, **vertex** shades are **interpolated across the polygon**

```
glShadeModel(GL_SMOOTH);
```

- If we use `glShadeModel(GL_FLAT);` the **color at the first vertex will determine the shade of the whole polygon**

Polygon Normals

- Polygons have a **single normal**

  Shades at the vertices as computed by the Phong model can be almost same

  Identical for a distant viewer (default) or if there is no **specular** component

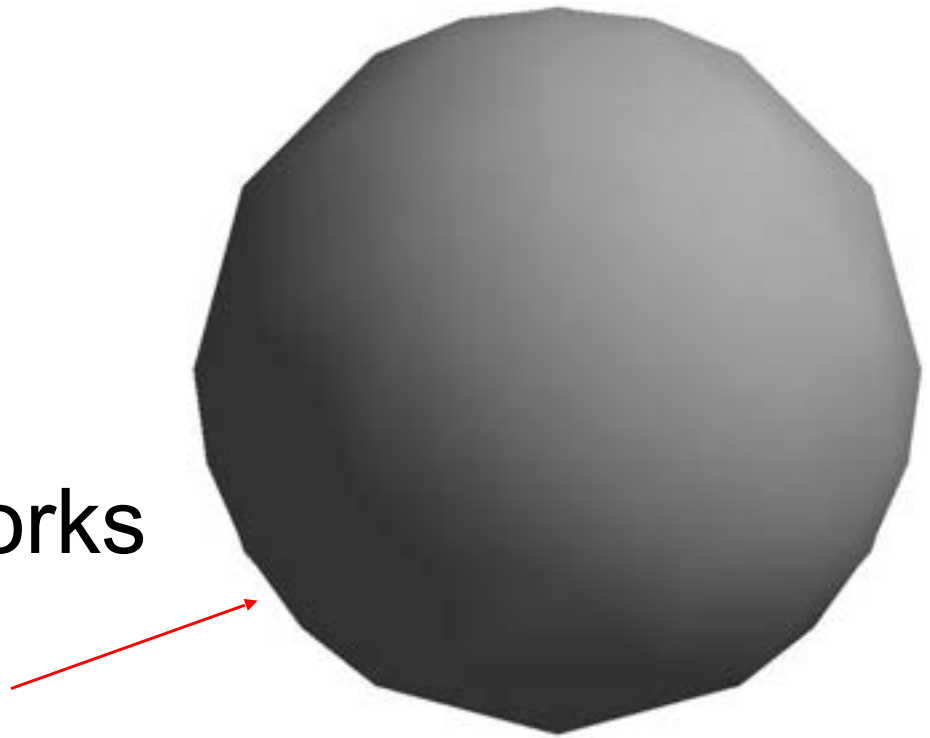
- Consider model of sphere
- Want **different normals at each vertex** even though this concept is not quite correct mathematically

Smooth Shading

- We can set a new normal at each vertex
- Easy for sphere model

  If centered at origin $\mathbf{n} = \mathbf{p}$

- Now **smooth shading** works
- Note *silhouette edge*

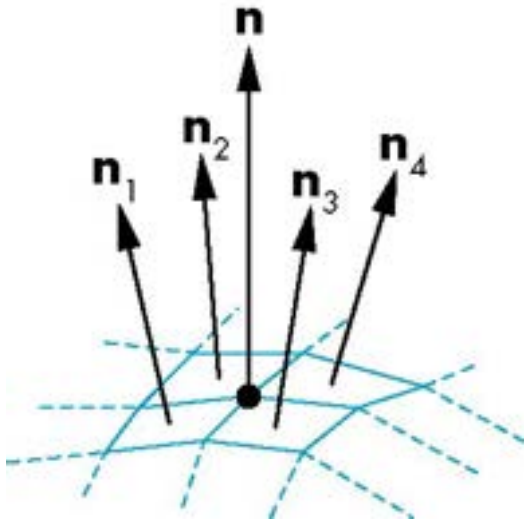


Mesh Shading

The previous example is not general because we **knew the normal at each vertex analytically**

For polygonal models, **Gouraud proposed** we use the **average of the normals around a mesh vertex**

$$\mathbf{n} = (\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4)/\ |\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4|$$



Gouraud and Phong Shading

- **Gouraud Shading**
  - **Find average normal at each vertex (vertex normals)**
  - Apply **modified Phong model** at **each vertex**
  - **Interpolate vertex shades across each polygon**

- **Phong shading**
  - **Find vertex normals**
  - **Interpolate vertex** normals across edges
  - Interpolate edge normals across polygon
  - Apply **modified Phong model** at **each fragment**

Comparison

If the polygon mesh approximates surfaces with a high curvatures, **Phong shading may look smooth** while **Gouraud shading may show edges**

Phong shading requires much more work than Gouraud shading

- Until recently not available in real time systems
- Now can be done using **fragment shaders** (see Chapter 9)

Both need data structures to represent meshes so we can obtain vertex normals

You will be prompted when to Upload completed document to CANVAS as score.doc (example 100.doc).

Warning:

TA, at random, will inspect the Uploaded document.

If you score is not honestly entered you will get a zero.

Please rename document to score.doc (example 100.doc)
Warning: if you score is not honestly honest you will get a zero.

Total score: ☐

Class PARTICIPATION on Lecture 6.doc ANSWER SHEET

(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

VH, it closes at 7:00 PM

NEXT.

10.04.2023 (W 5:30 to 7) (13)	Homework 6	Lecture 7
10.09.2023 (M 5:30 to 7) (14)		PROJECT 2
10.11.2023 (W 5:30 to 7) (15)		EXAM 2 REVIEW
10.16.2023 (M 5:30 to 7) (16)		EXAM 2

0.04.2023 (W 5:30 to 7) (13)	Homework 6	Lecture 7

VH, publish

At 6:45 PM.

End Class 12

VH, Download Attendance Report
Rename it:
10.2.2023 Attendance Report FINAL

VH, upload Lecture 6 to CANVAS.