Announcements

- Project 1 due today at midnight.
- Homework 1 will be posted later today.

Basics of Textures

Basics of texture mapping in OpenGL

Texture Mapping

- A way of adding surface details

- Two ways can achieve the goal:
  - Model the surface with more polygons
    - » Slows down rendering speed
    - » Hard to model fine features

Texture Mapping

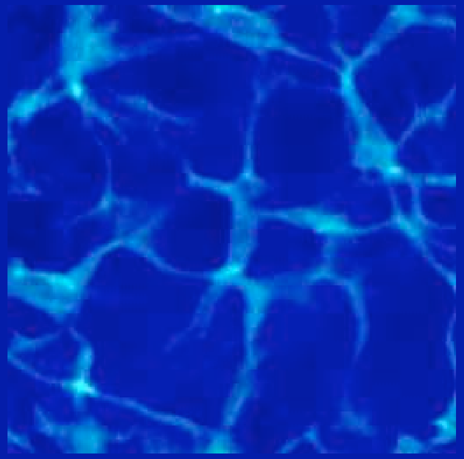
- A way of adding surface details

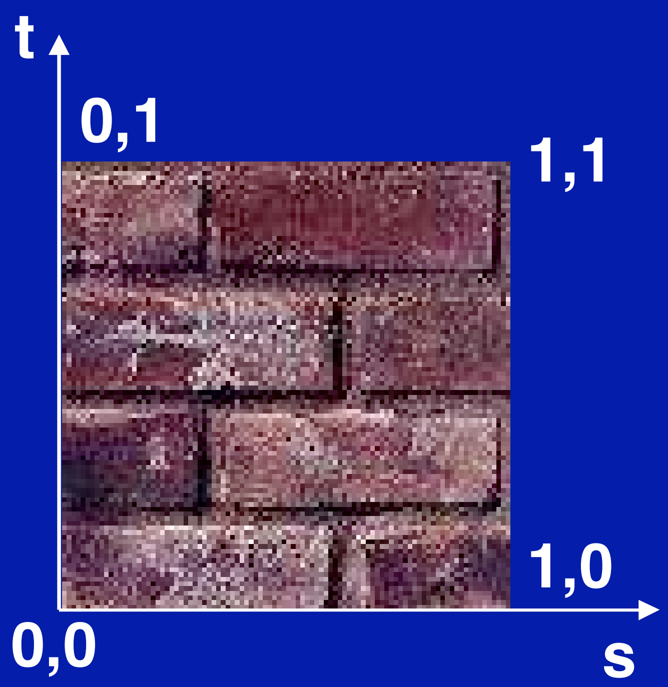
- Two ways can achieve the goal:
  - Model the surface with more polygons
    - » Slows down rendering speed
    - » Hard to model fine features
  - Map a texture to the surface
    - » This lecture
    - » Image complexity does not affect complexity of processing

I live in my own little world but that's all right, they know me there.

The texture

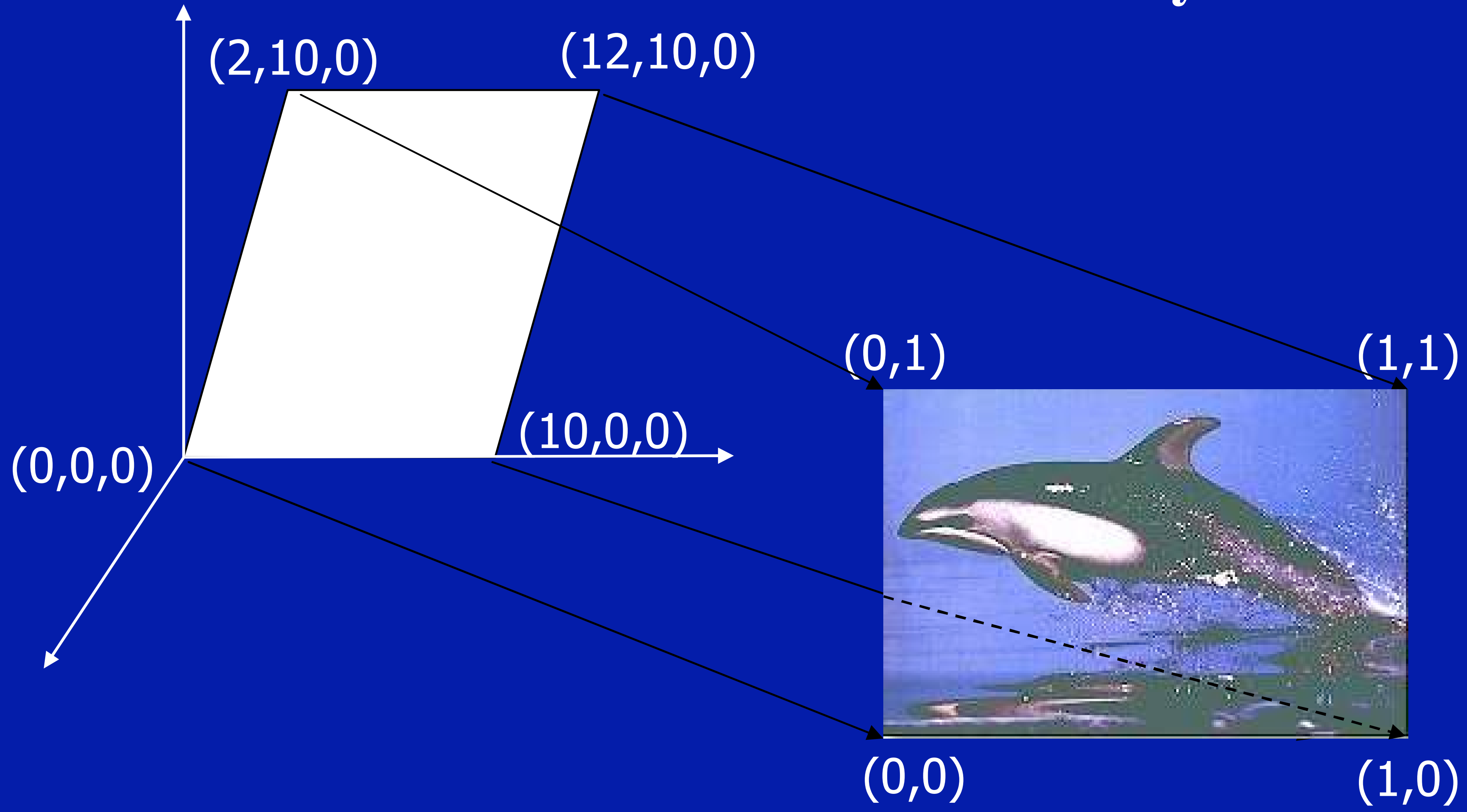
- Texture is a bitmap image
- 2D array: texture[height][width][4]
- Pixels of the texture called *texels*
- Texel coordinates (s,t) scaled to [0,1] range





Map textures to surfaces

The polygon can have arbitrary size and shape



The drawing itself

- Use GLTexCoord2f(s,t) to specify texture coordinates
- Example:

```
glEnable(GL_TEXTURE_2D)
glBegin(GL_QUADS);
glTexCoord2f(0.0,0.0);  glVertex3f(0.0,0.0,0.0);
glTexCoord2f(0.0,1.0);  glVertex3f(2.0,10.0,0.0);
glTexCoord2f(1.0,0.0);  glVertex3f(10.0,0.0,0.0);
glTexCoord2f(1.0,1.0);  glVertex3f(12.0,10.0,0.0);
glEnd();
glDisable(GL_TEXTURE_2D)
```

- State machine: Texture coordinates remain valid until you change them or exit texture mode via

  glDisable (GL_TEXTURE_2D)

Color blending

- Final pixel color = f (texture color, object color)

- How to determine the color of the final pixel?
  - GL_REPLACE – use texture color to replace object color
  - GL_BLEND – linear combination of texture and object color
  - GL_MODULATE – multiply texture and object color

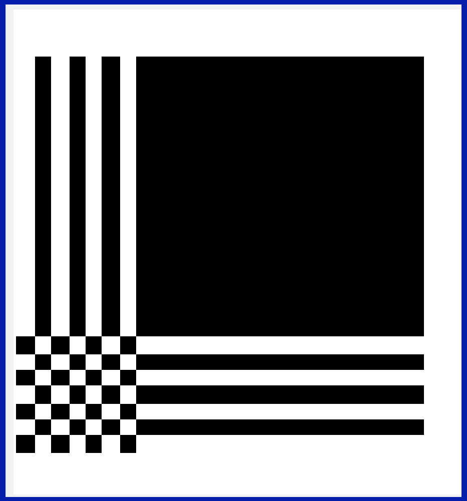
- Example:
  - glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

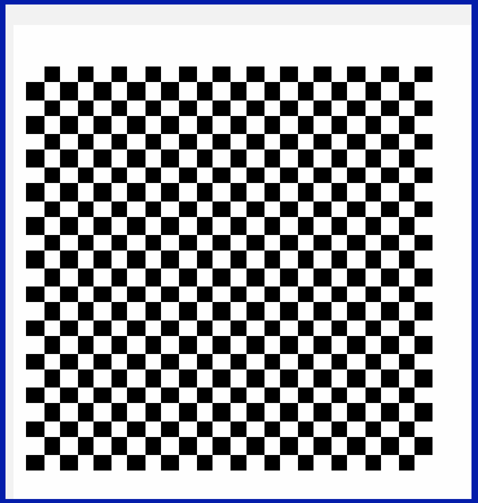
What happens if texture coordinates outside [0,1] ?

- Two choices:
  - Repeat pattern (GL_REPEAT)
  - Clamp to maximum/minimum value (GL_CLAMP)

- Example:
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP)

What happens if texture coordinates outside [0,1] ?

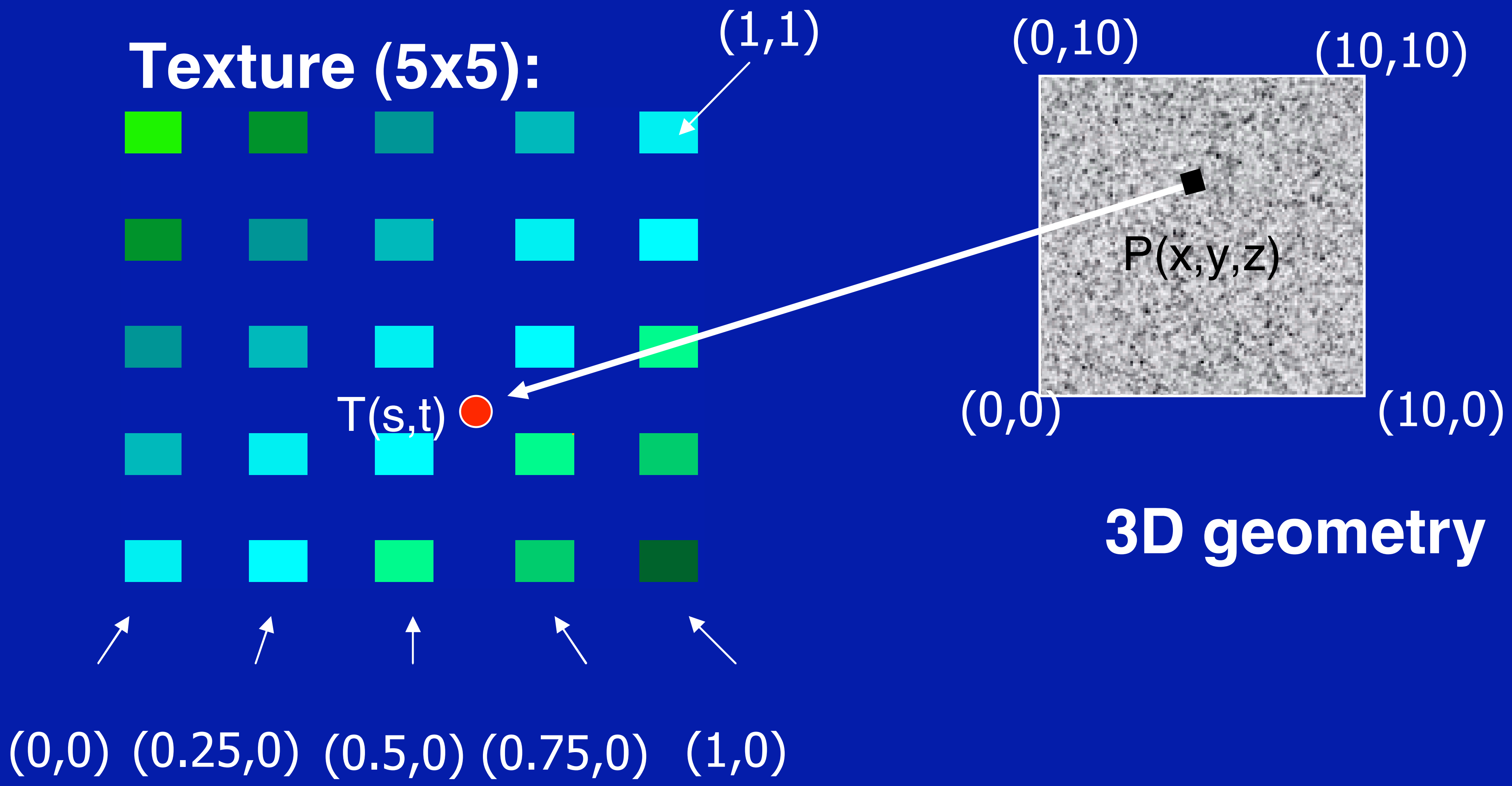

clamp


repeat

```
glTexCoord2f(0.0, 0.0);  glVertex3f(0.0,  0.0,  0.0);
glTexCoord2f(0.0, 3.0);  glVertex3f(0.0, 10.0, 0.0);
glTexCoord2f(3.0, 0.0);  glVertex3f(10.0, 0.0, 0.0);
glTexCoord2f(3.0, 3.0);  glVertex3f(10.0, 10.0, 0.0);
```

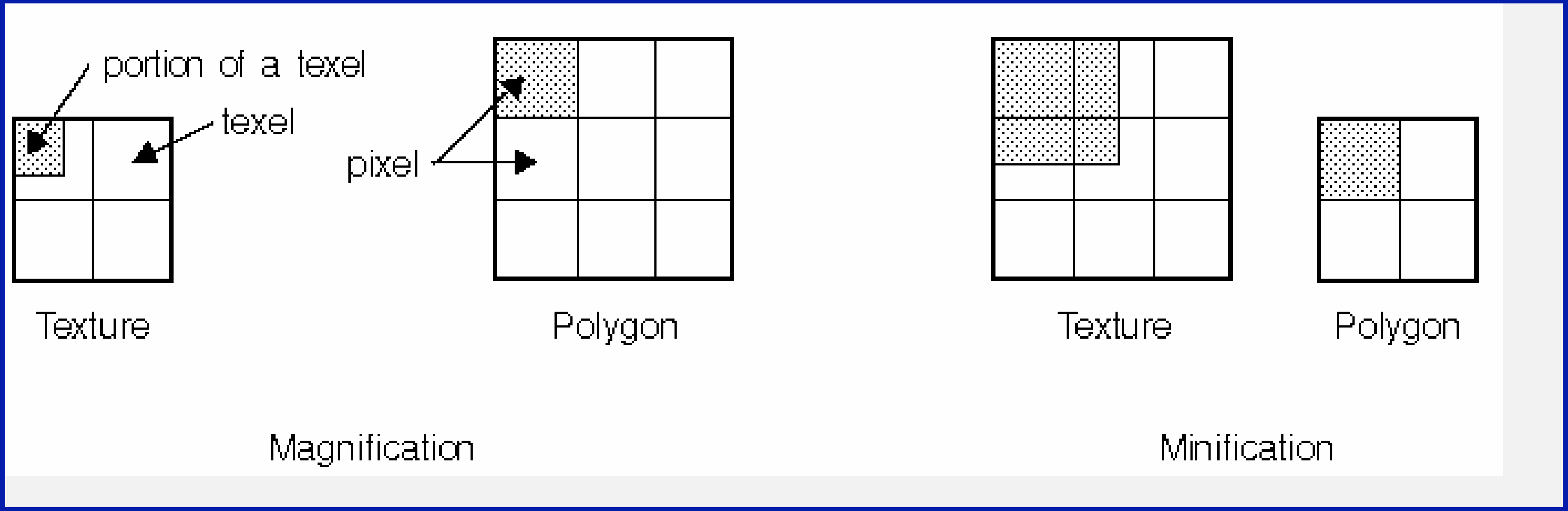
Texture Value Lookup

- For given texture coordinates (s,t), we can find a unique image value, corresponding to the texture image at that location



Interpolating colors

- Some (s,t) coordinates not directly at pixel in the texture, but in between

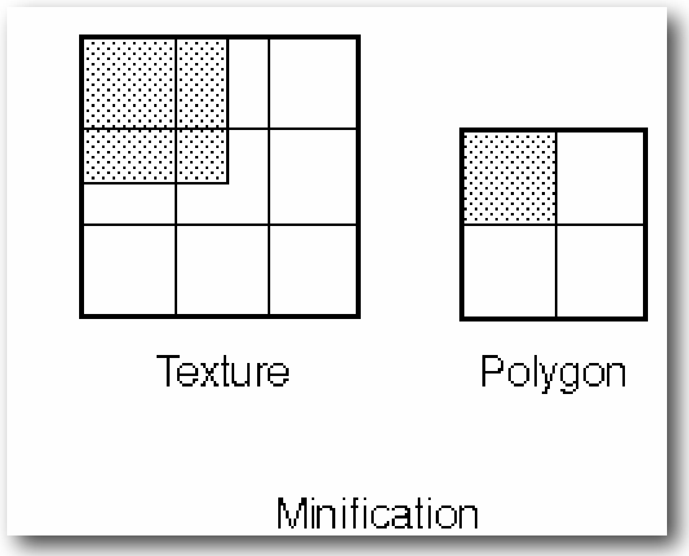


Interpolating colors

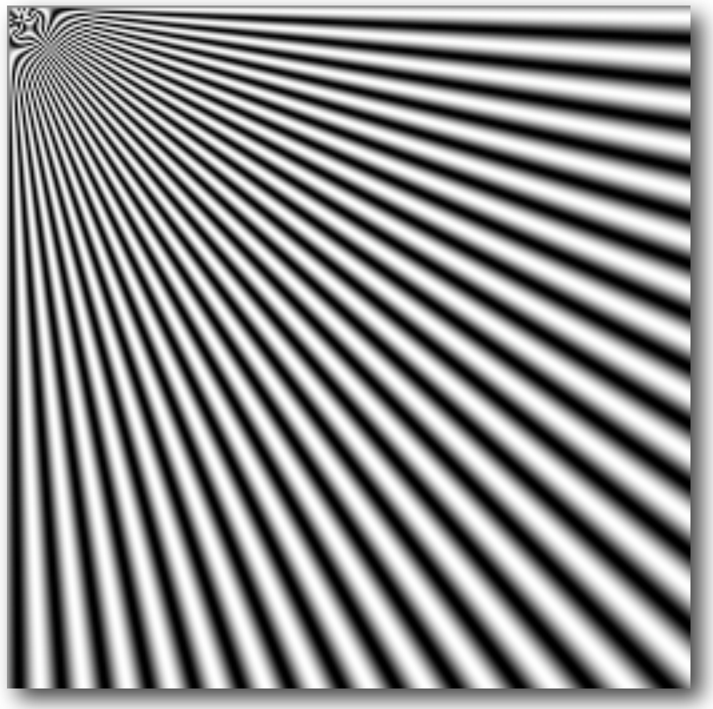
- Solutions:
  - Nearest neighbor
    - Use the nearest neighbor to determine color
    - Faster, but worse quality
    - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  - Linear interpolation
    - Incorporate colors of several neighbors to determine color
    - Slower, better quality
    - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)

Other solutions

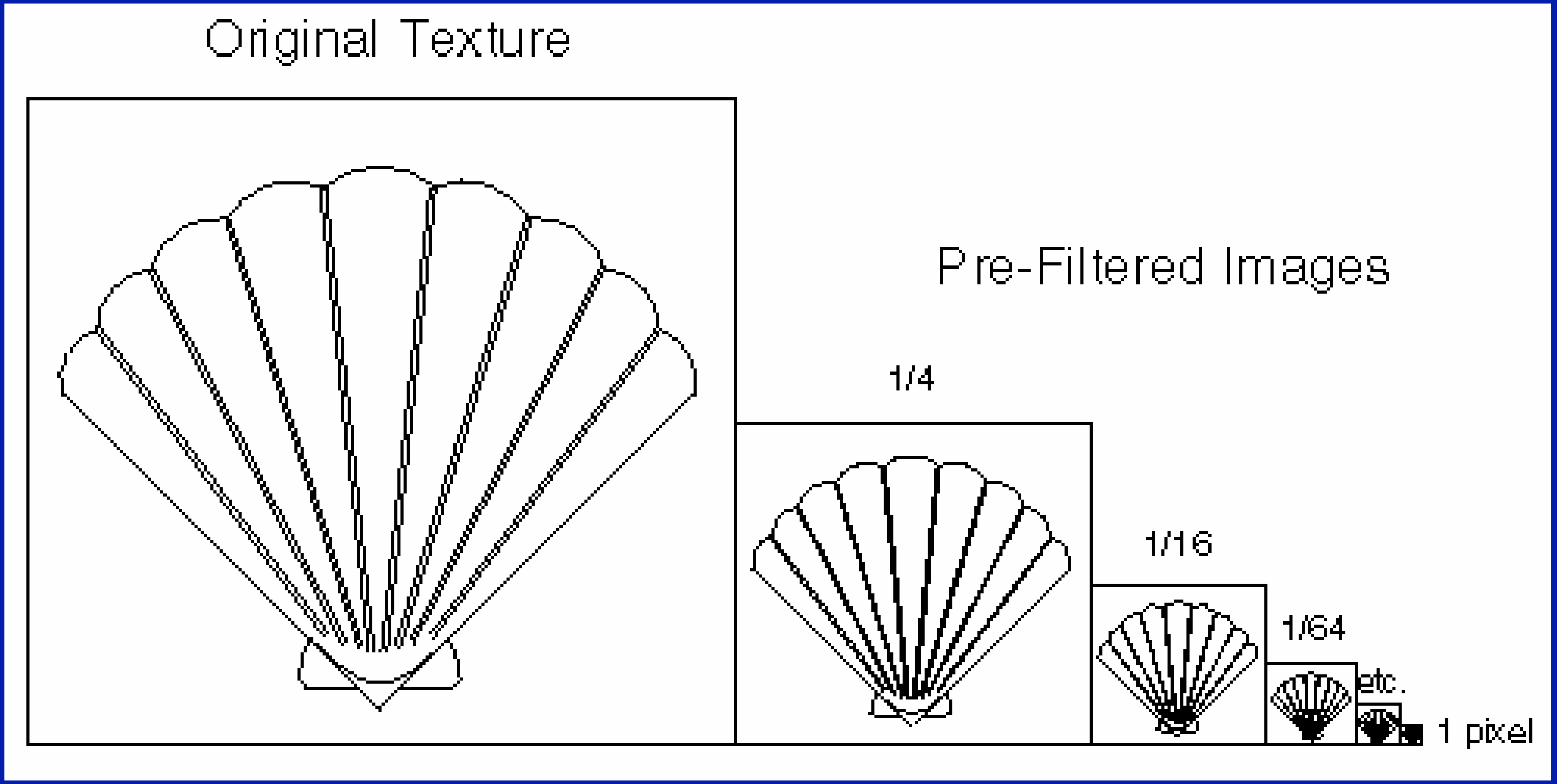
- Signal processing.
- What is wrong with linear interpolation...

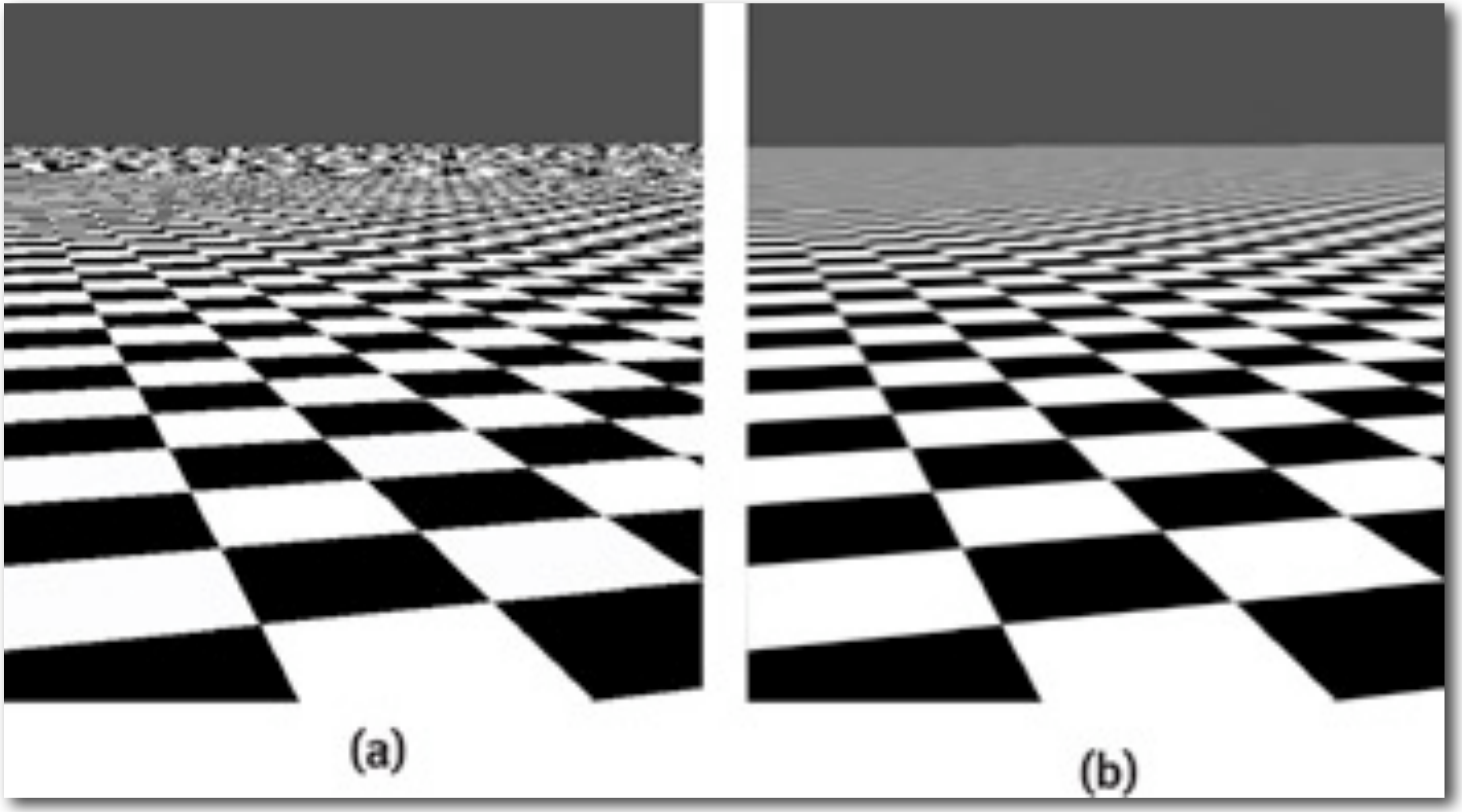


Antialiasing...



Texture Levels




(a)
(b)

Texture mapping in OpenGL

- In init():
  - Specify texture
    - Read image from file into an array in memory or generate the image using the program
  - Specify texture mapping parameters
    - Wrapping, filtering, etc.
  - Define (activate) the texture
- In display():
  - Enable GL texture mapping
  - Draw objects: Assign texture coordinates to vertices
  - Disable GL texture mapping

Specifying texture mapping parameters

- Use glTexParameteri
- Example:

```
// texture wrapping on
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_REPEAT); // repeat pattern in s texture coordinate
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_REPEAT); // repeat pattern in t texture coordinate

// use nearest neighbor for both minification and magnification
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
```

Defining (activating) texture

- Do once in init() to set up initial pattern
- To use another texture, make further calls in display() to glTexImage2D, specifying another image
  - But this is slow: use Texture Objects itself
- The dimensions of texture images **must be powers of 2**
  - if not, rescale image or pad with zeros
- glTexImage2D(Glenum target, Glint level,
  Glint internalFormat, int width, int height, Glint border,
  Glenum format, Glenum type, Glvoid* img)
- Example:
  - glTexImage2D(GL_TEXTURE_2D, 0,
    GL_RGBA, 256, 256, 0,
    GL_RGBA, GL_UNSIGNED_BYTE, pointerToImage)

Enable/disable texture mode

- Can do in init() or successively in display()
- glEnable(GL_TEXTURE_2D)
- glDisable(GL_TEXTURE_2D)

- Successively enable/disable texture mode to switch between drawing textured/non-textured polygons
- Changing textures:
  - Only one texture active at any given time
  - make another call to glTexImage2D to make another pattern active

The drawing itself

- Use GLTexCoord2f(s,t) to specify texture coordinates
- State machine: Texture coordinates remain valid until you change them or exit texture mode via glDisable (GL_TEXTURE_2D)
- Example:

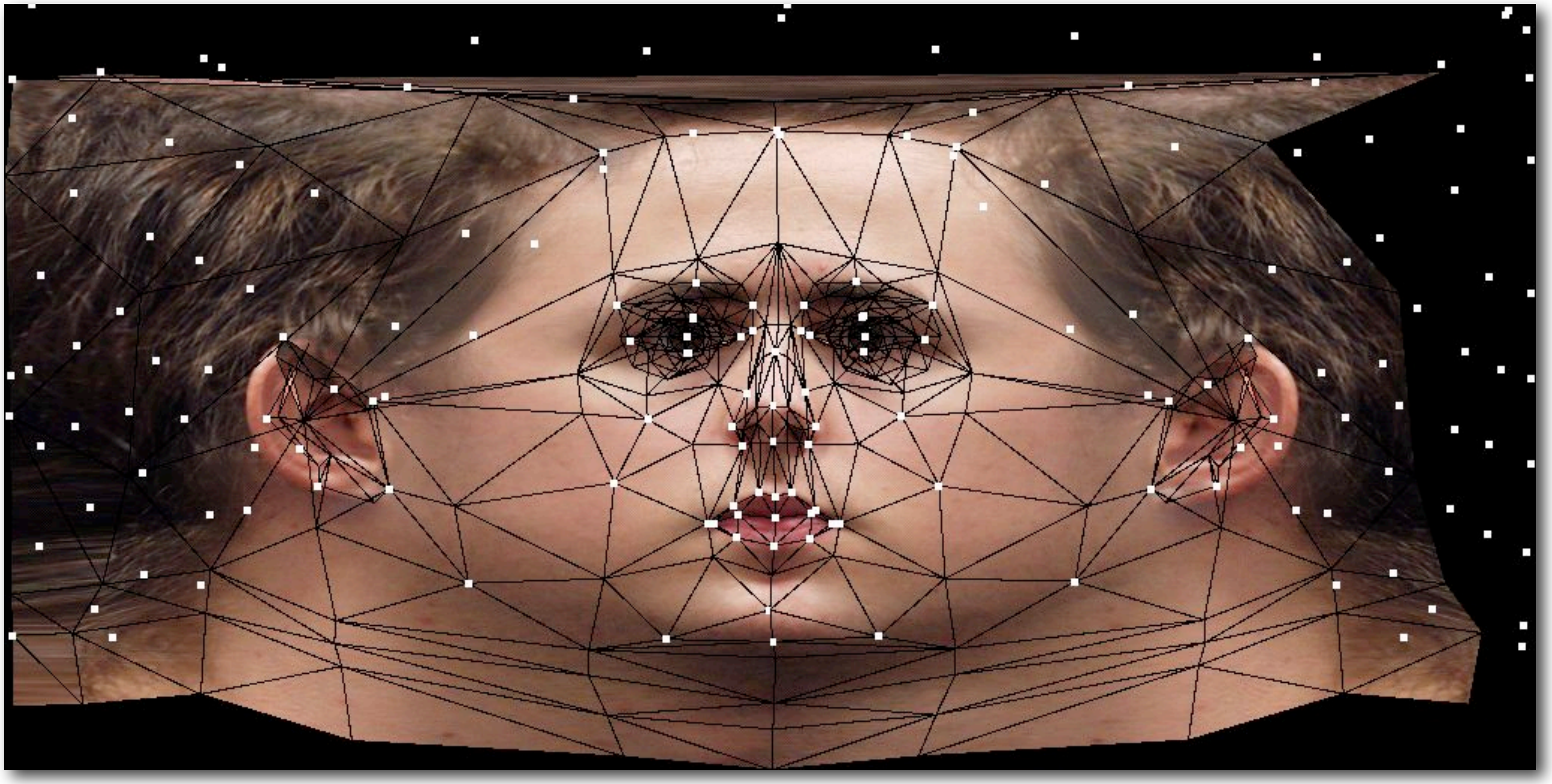
```
glEnable(GL_TEXTURE_2D)
glBegin(GL_QUADS);
glTexCoord2f(0.0,0.0); glVertex3f(-2.0,-1.0,0.0);
glTexCoord2f(0.0,1.0); glVertex3f(-2.0,1.0,0.0);
glTexCoord2f(1.0,0.0); glVertex3f(0.0,1.0,0.0);
glTexCoord2f(1.0,1.0); glVertex3f(0.0,-1.0,0.0);
…
glEnd();
glDisable(GL_TEXTURE_2D)
```

Everything together

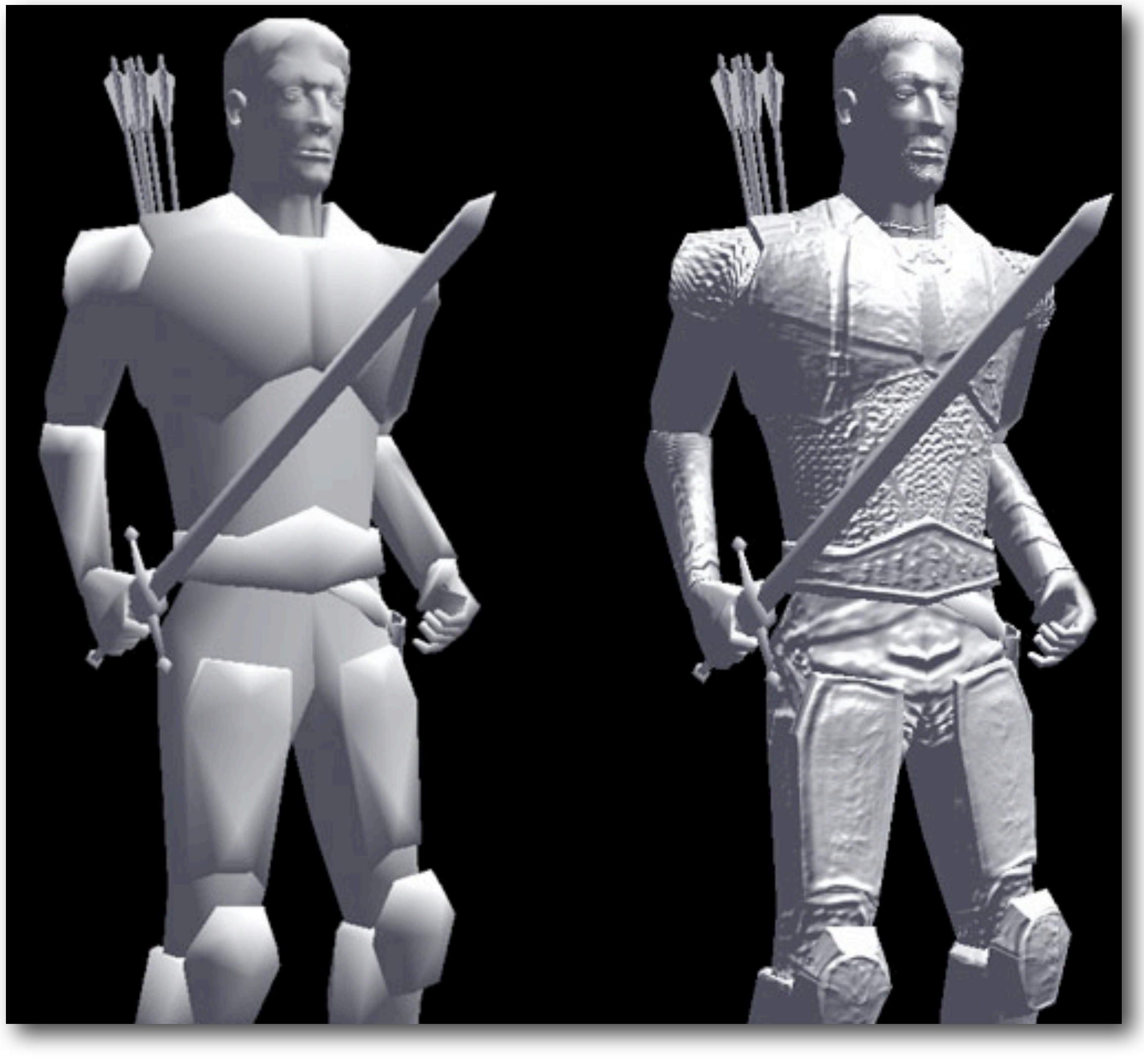
```
void init(void):
{
…
put image into 2D memory array; // can use libpicio library

// specify texture parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT); // repeat pattern in s
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT); // repeat pattern in t

// use nearest neighbor for both minification and magnification
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

// make the pattern at location pointerToImage the active pattern
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 256, 256, 0,
              GL_RGBA, GL_UNSIGNED_BYTE, pointerToImage)


…
}
```

Everything together (contd.)

```
void display(void):
{
…
// no blending, use texture color directly
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE, GL_REPLACE);
// turn on texture mode
glEnable(GL_TEXTURE_2D);

glBegin(GL_QUADS); // draw a quad
glTexCoord2f(0.0,0.0); glVertex3f(-2.0,-1.0,0.0);
glTexCoord2f(0.0,1.0); glVertex3f(-2.0,1.0,0.0);
glTexCoord2f(1.0,0.0); glVertex3f(0.0,1.0,0.0);
glTexCoord2f(1.0,1.0); glVertex3f(0.0,-1.0,0.0);
…
glEnd();

// turn off texture mode
glDisable(GL_TEXTURE_2D);

// draw some non-texture mapped objects
…
// switch back to texture mode, etc.
…
}
```



Generalizations of Texture Mapping?



Shading

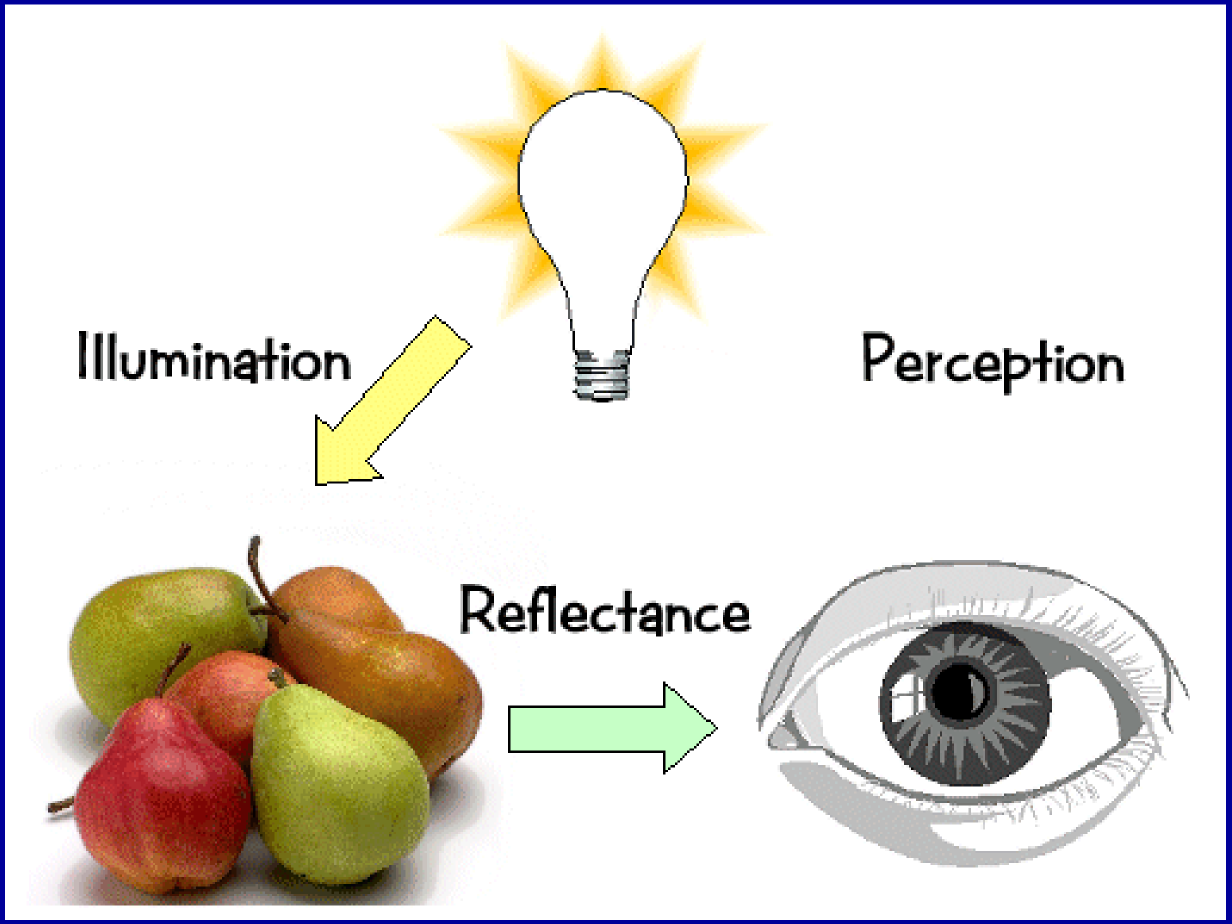
Light
Lambertian Shading
Phong Shading

Shirley Chapter 9

COMPUTER GRAPHICS

15-462

Light Transport



Shading (today)
Ray Tracing
Radiosity
Texture Mapping
Reflection Models
Scan Conversion

Later:

Non-photorealistic rendering
Image-based Rendering

What are the patterns of light in this room?

Projector as light source

Light transmitted through windows

Blue light reflecting from screen

Blackboard is matte surface

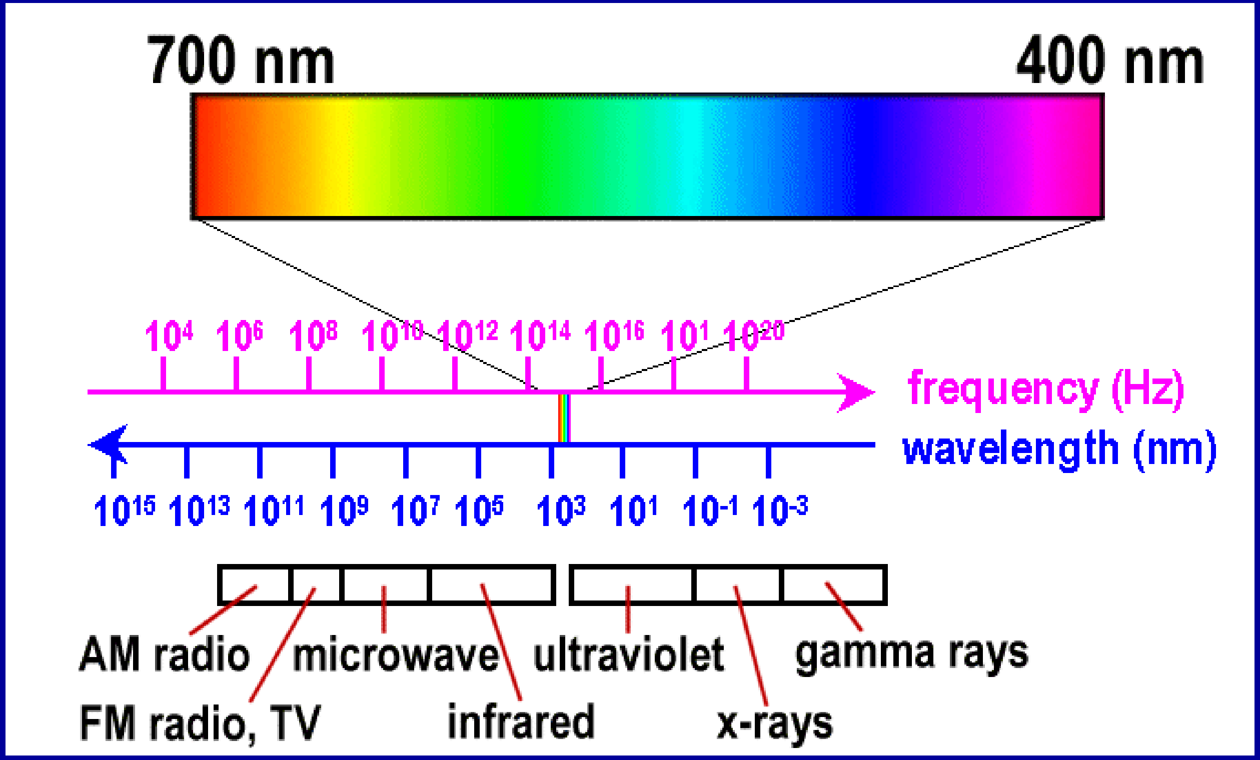
Edge of screen is shiny surface

Shadows underneath the desks

Physics of Light and Color

Electromagnetic (EM) radiation

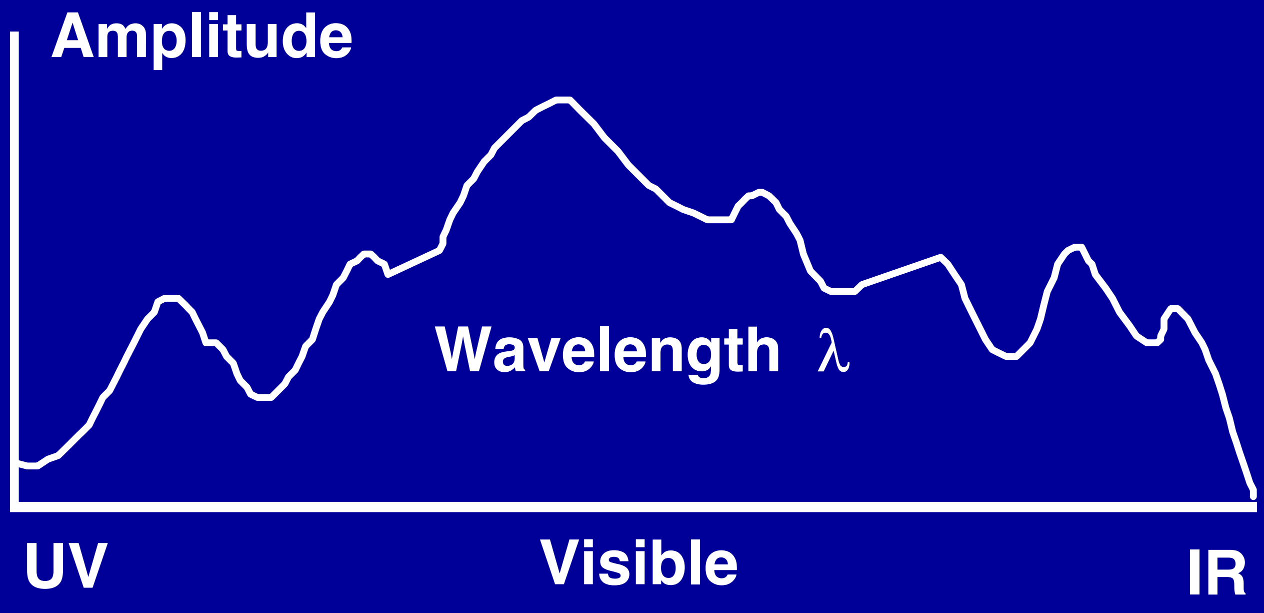
- Different colors correspond to radiation of different wavelengths λ
- Intensity of each wavelength specified by amplitude
- Frequency $\nu = 2\pi / \lambda$
  - long wavelength is low frequency
  - short wavelength is high frequency



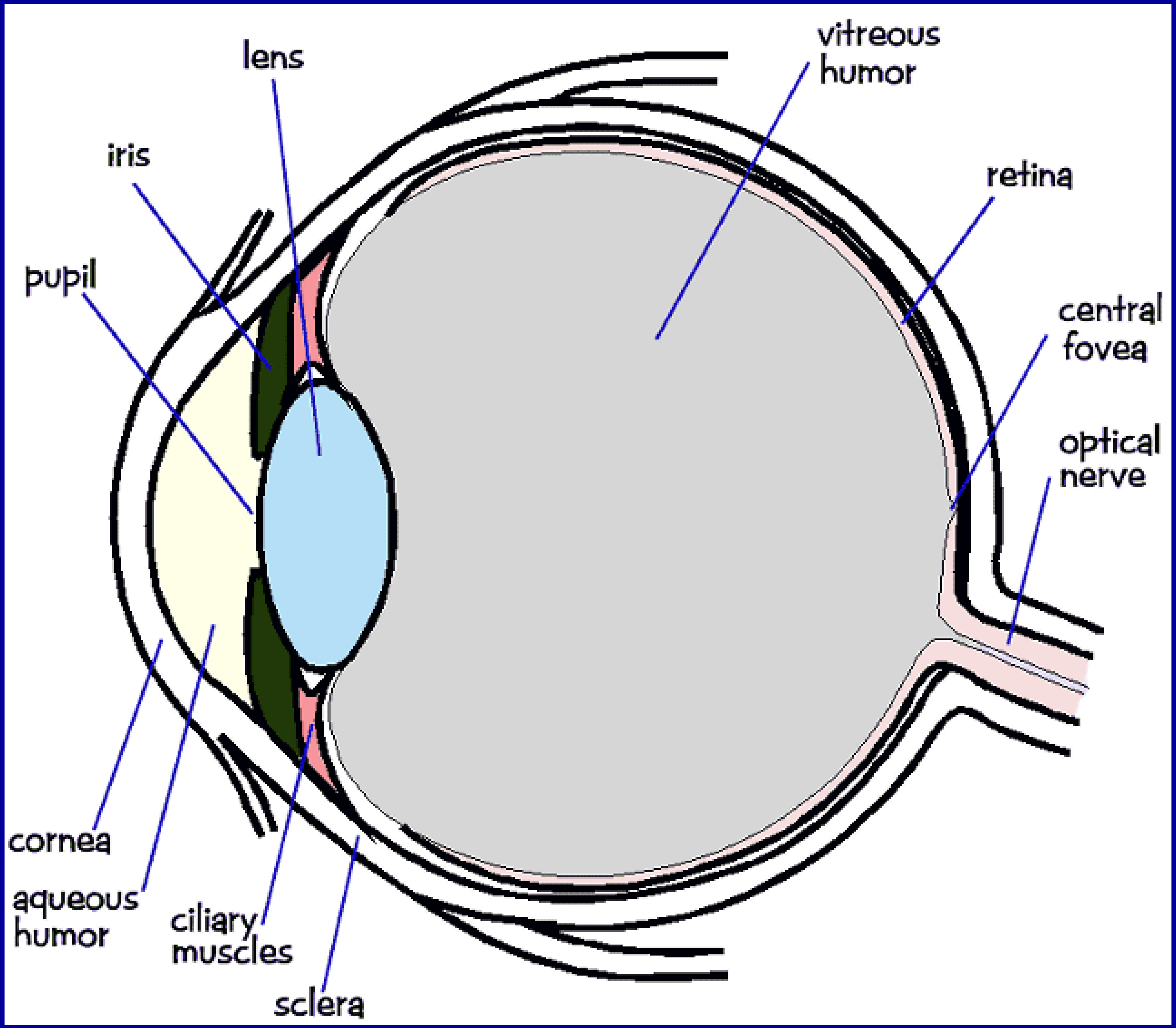
We perceive EM radiation with λ in the 400-700 nm range

Color: What's There vs. What We See

- Human eyes respond to “visible light”
  - tiny piece of spectrum between infra-red and ultraviolet
- Color defined by the emission spectrum of the light source
  - amplitude vs wavelength (or frequency) plot

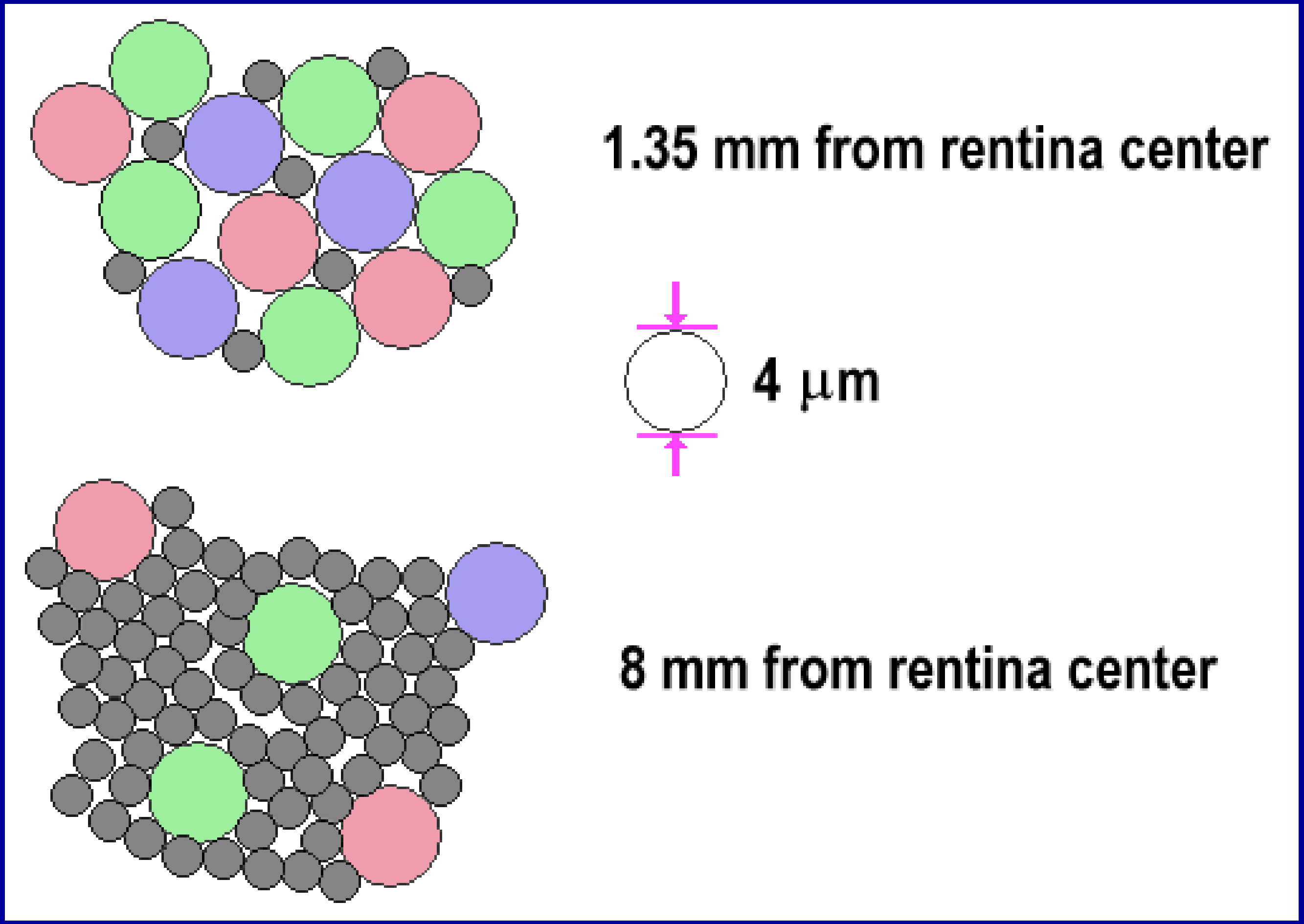


The Eye

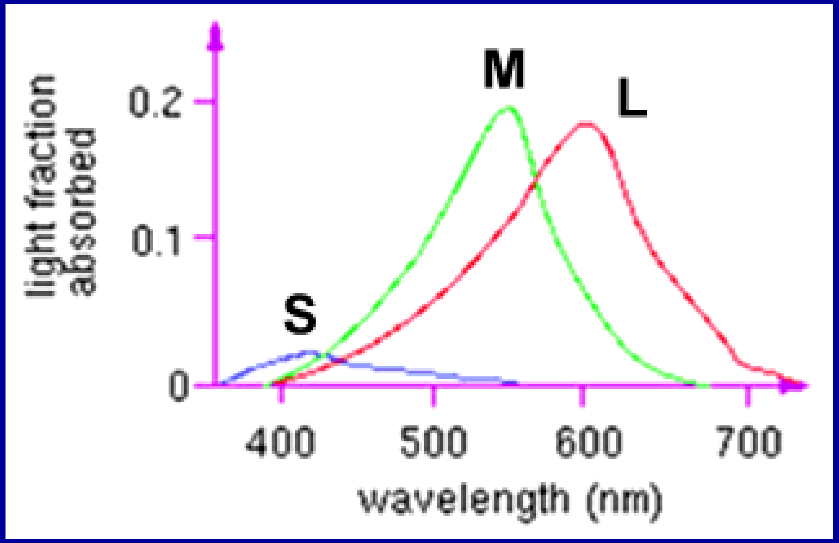


- The image is formed on the *retina*
- Retina contains two types of cells: *rods* and *cones*
- Cones measure color (red, green, blue)
- Rods responsible for monochrome night-vision

The Fovea



Cones are most densely packed within a region of the retina called the *fovea*



Three types of cones: S,M,L

Corresponds to 3 visual pigments

Roughly speaking:

S responds to blue

M responds to green

L responds to red

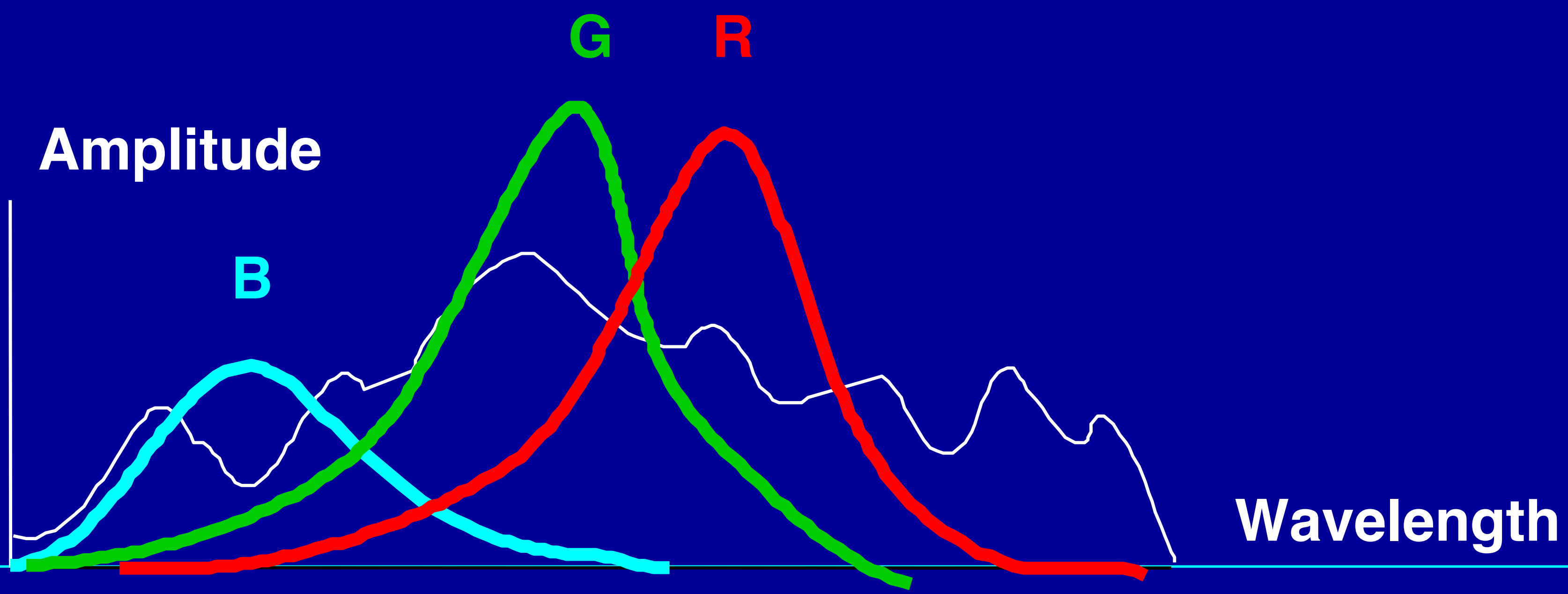
Not uniform sensitivity

Colorblindness

deficiency of one cone/pigment type

Color Filters

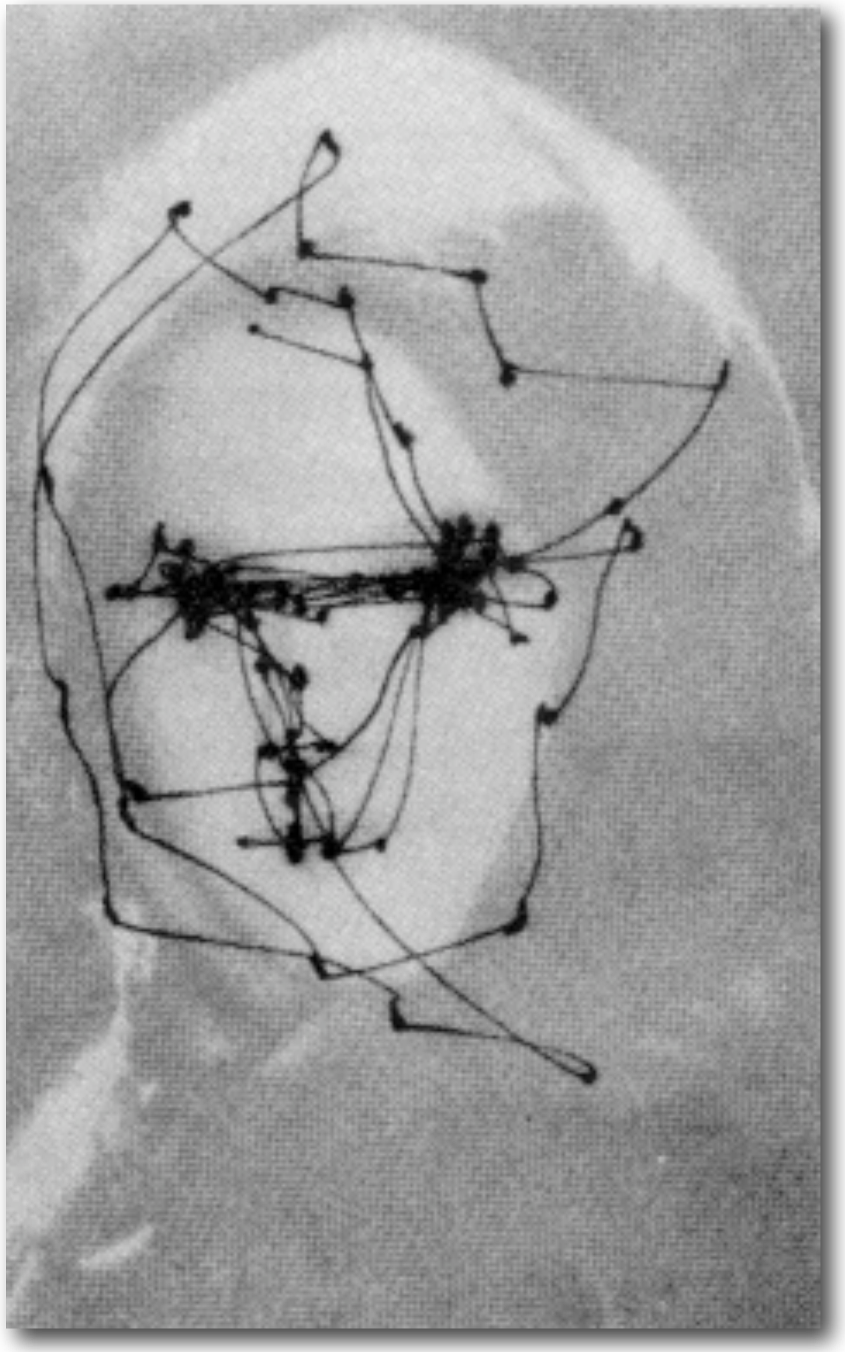
- Rods and cones can be thought of as filters
  - Cones detect red, green or blue parts of spectrum
  - Rods detect average intensity across spectrum
- A physical spectrum is a complex function of wavelength
  - But what we see can be described by just three numbers—the color filter outputs
  - How can we encode a whole function with just three numbers?
  - We can't—we can't distinguish certain colors--*metamers*



Vision and the brain

- The retina is part of the central nervous system
- 2 million fibers from retina to lateral geniculate nucleus (*LGN*), 10 million from there to brain.
- Primary connection is *Primary Visual Cortex* or *V1*
  - 2 cm^2 on back of brain
  - Hypothesis: V1 gets used as a sort of image buffer for higher processing in the rest of the brain
- Steps:
  - Saccade ends
  - Retina accumulates image
  - LGN opens connections, image gets written to V1
  - Rest of brain accesses that info
  - Meanwhile, a point of interest is being generated for next saccade
  - Next saccade happens perhaps 250ms later; go back to step 1
- All automatic; eye tracking systems can discern attention but pointing with eyes doesn’t work very well for user interfaces.

Saccades



Color Models

- Okay, so our visual system is quite limited, but maybe this is good news. . .
- We can avoid computing and reproducing the full color spectrum since people only have three color channels
  - everything would be much more complex if we perceived the full spectrum
    - transmission would require much higher bandwidths
    - display would require much more complex methods
  - real-time color 3D graphics is feasible
  - any scheme for describing color requires only three values
  - lots of different color spaces--related by matrix transformations

Color Spaces

Spectrum

- allows any radiation (visible or invisible) to be described
- usually unnecessary and impractical

RGB

- convenient for display (CRT uses red, green, and blue phosphors)
- not very intuitive

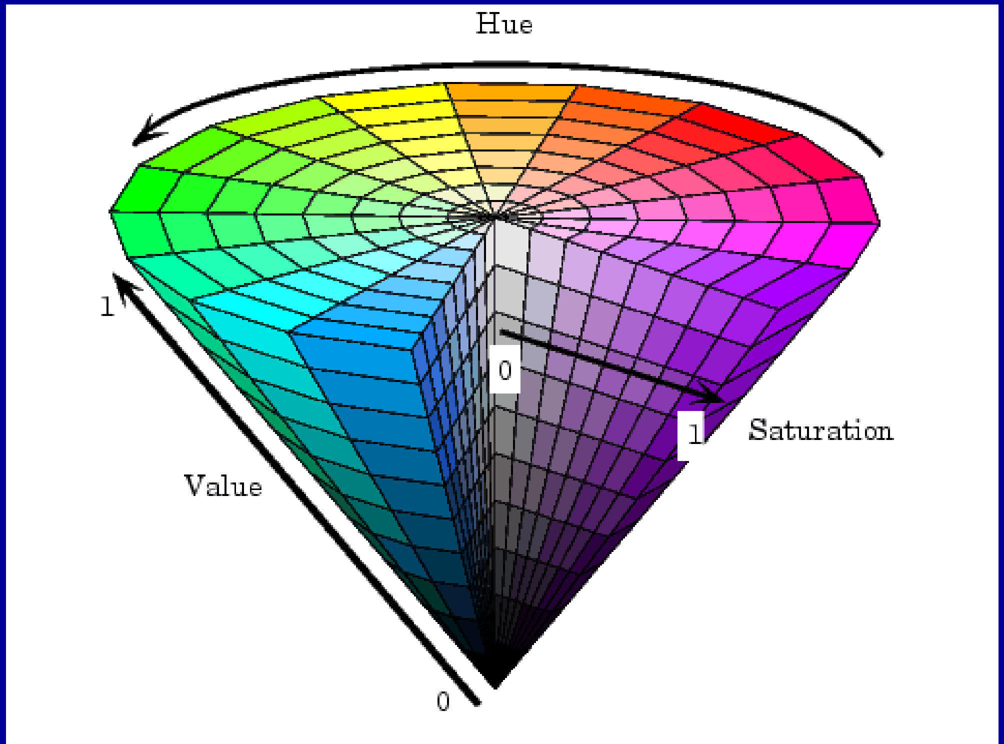
HSV

- an intuitive color space
- H is hue - what color is it? S is saturation or purity - how non-gray is it? V is value - how bright is it?
- H is cyclic therefore it is a non-linear transformation of RGB

CIE XYZ

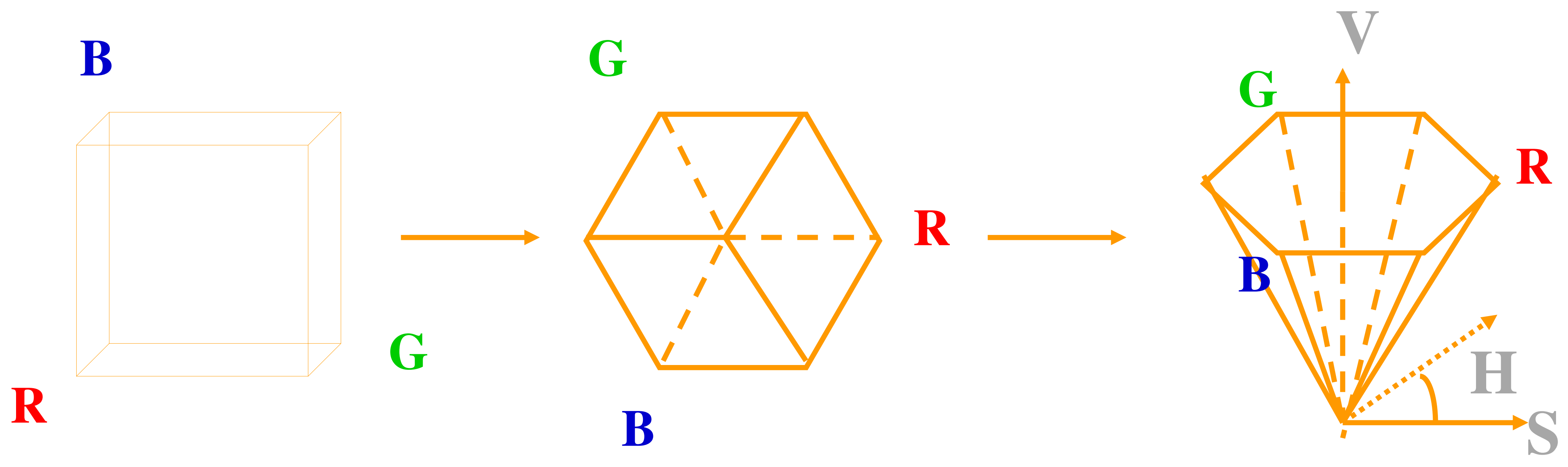
- a linear transform of RGB used by color scientists

HSV



Hue: color
Saturation: how non-grey
Value: brightness

From mathworks



Better Color Models?

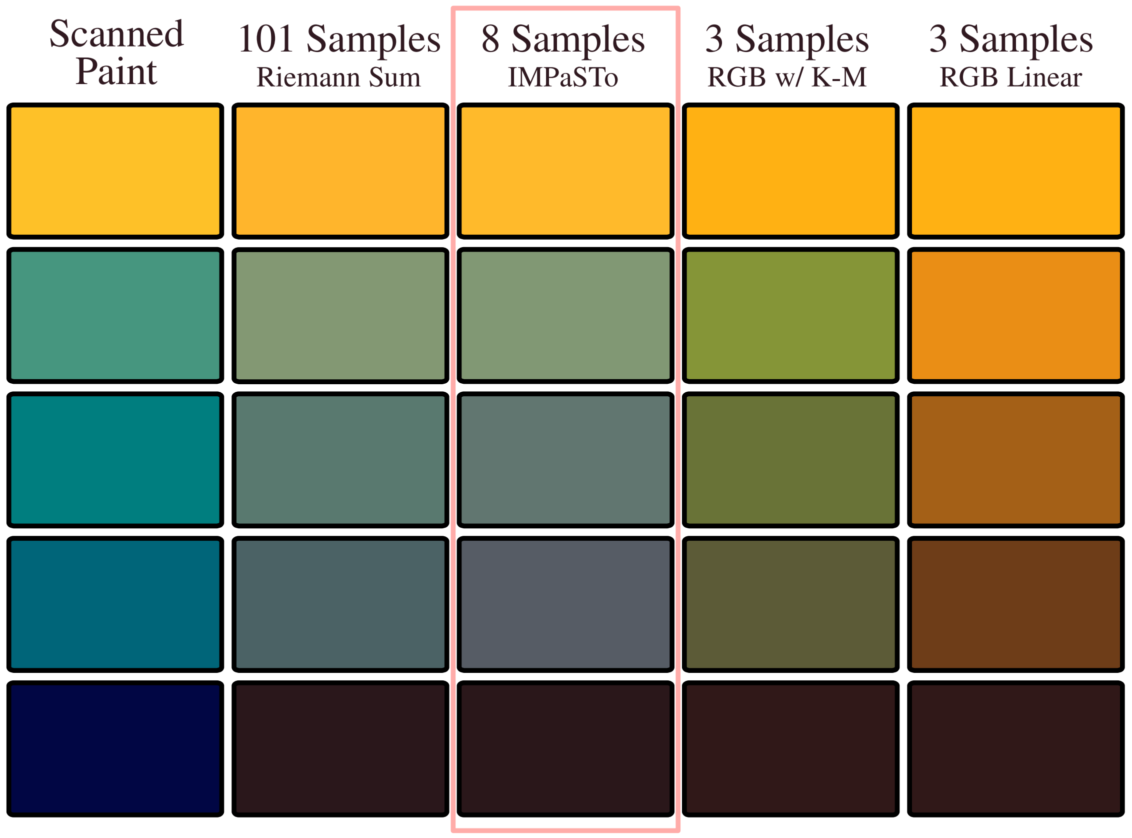
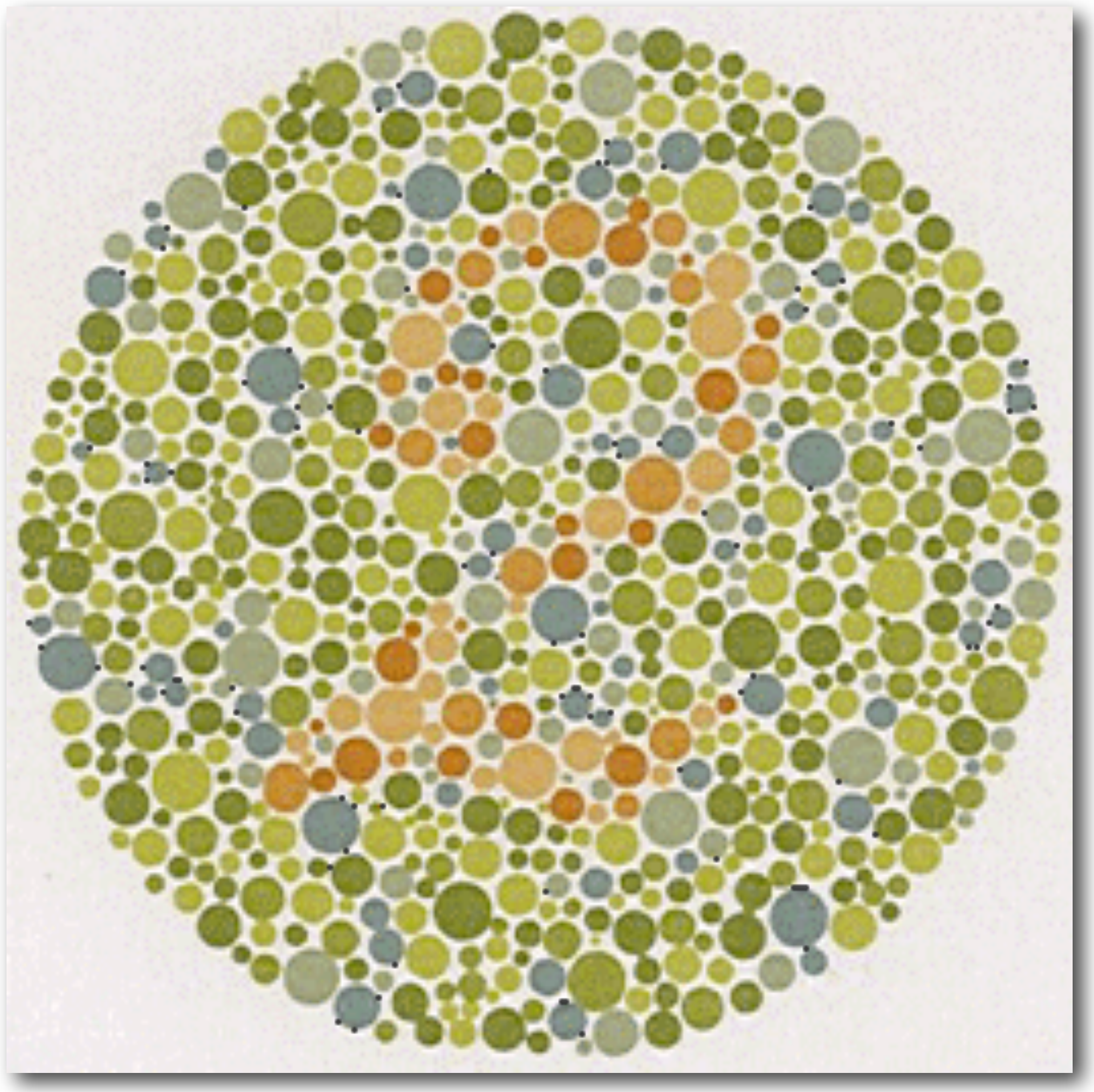


Figure 11: *A painting created with IMPaSTo, after a painting by*

source:
IMPaSTo: a realistic, interactive model for paint
William Baxter, Jeremy Wendt, Ming C. Lin
NPAR 2004, June 2004, pp. 45-56.

Tetrachromacy



Additive vs. Subtractive Color

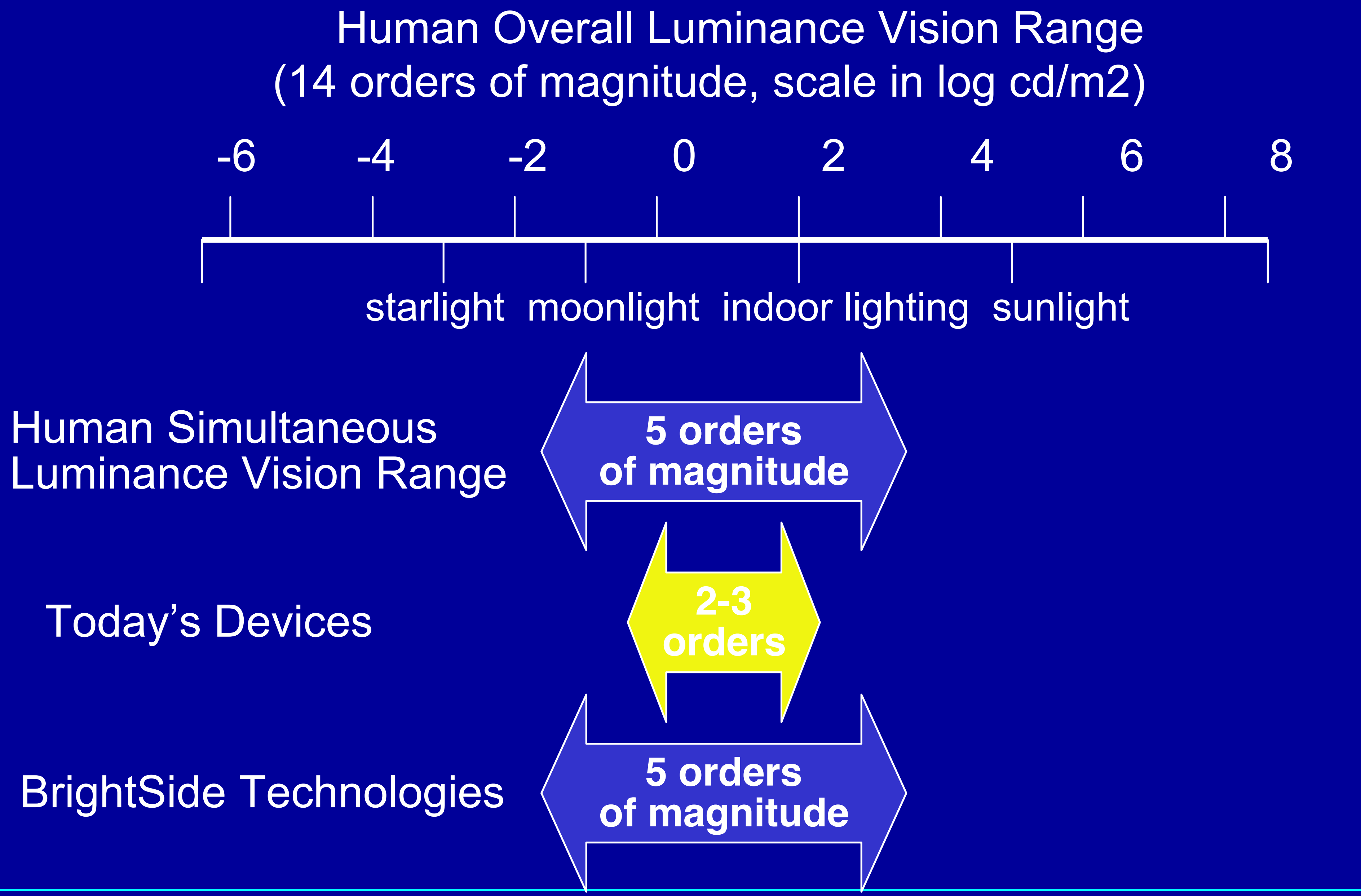
- Working with light: additive primaries
  - Red, green and blue components are added by the superposition property of electromagnetism
  - Conceptually: start with black, primaries add light
- Working with pigments: subtractive primaries
  - Typical inks (CMYK): cyan, magenta, yellow, black
  - Conceptually: start with white, pigments filter out light
  - The pigments remove parts of the spectrum

dye color	**absorbs**	**reflects**
cyan	red	blue and green
magenta	green	blue and red
yellow	blue	red and green
black	all	none

  - Inks interact in nonlinear ways--makes converting from monitor color to printer color a challenging problem
  - Black ink (K) used to ensure a high quality black can be printed

What about displays?

Humans can't see most of the spectrum but displays can't display most of what we can see



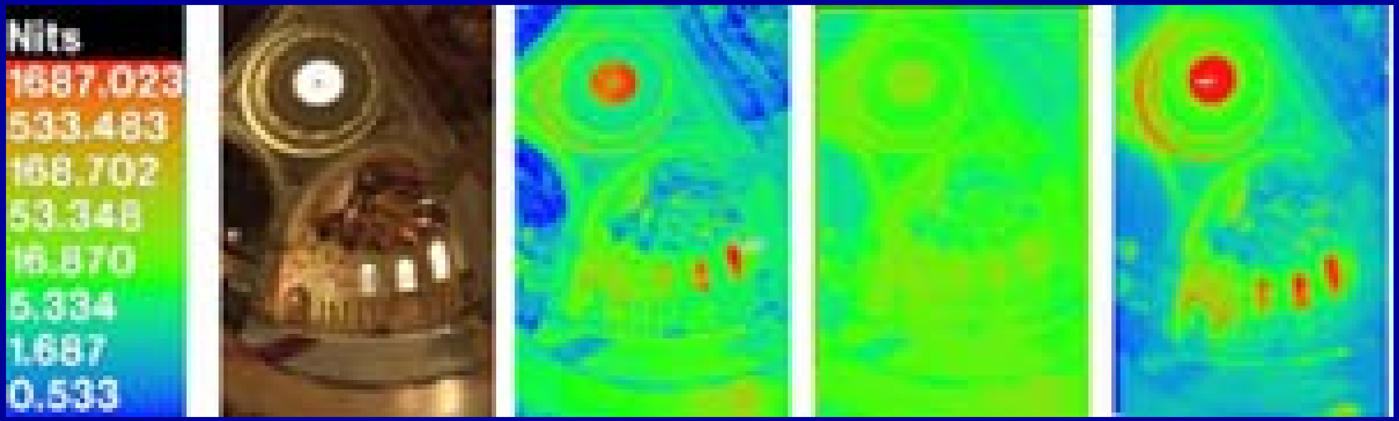
What about displays?

Conventional CRTs have 600:1 dynamic range

Flat-panel LCDs are 500:1.

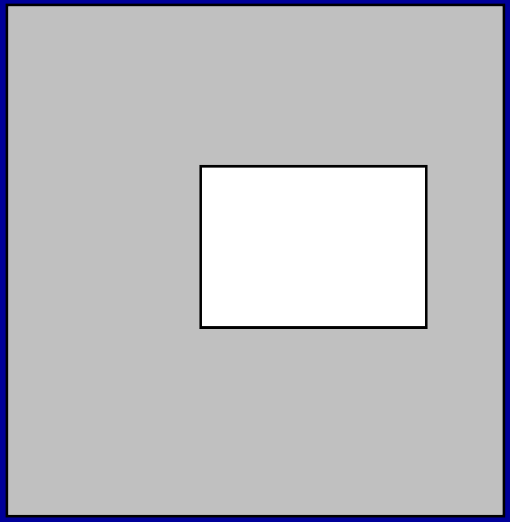
BrightSide's HDR displays achieve 200,000:1

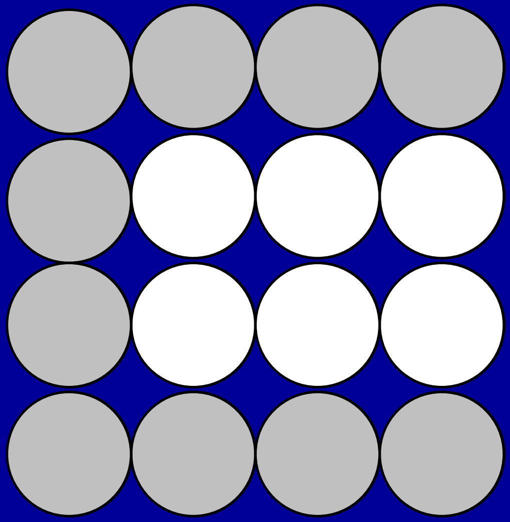
10 times higher brightness than any commercially available display while at the same time delivering a black that is over 10 times darker than that of conventional displays. http://www.brightsidetech.com

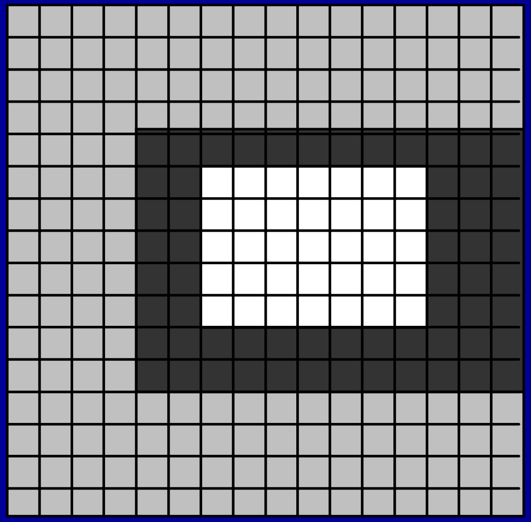


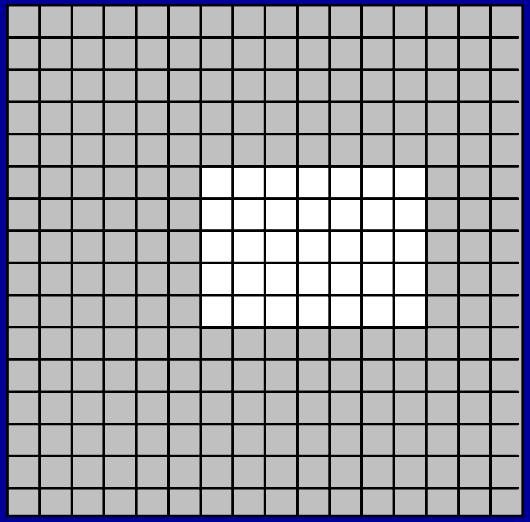
HDR image, range, conventional display, HDR display

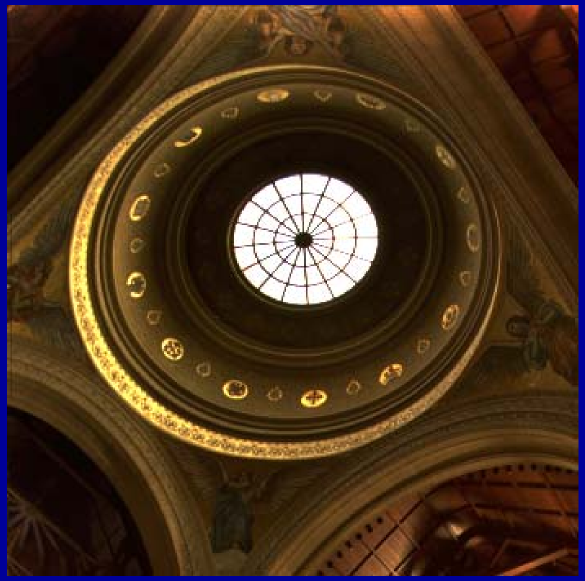
How does it work?


HDR Image


LED array


LCD with correction


Output image

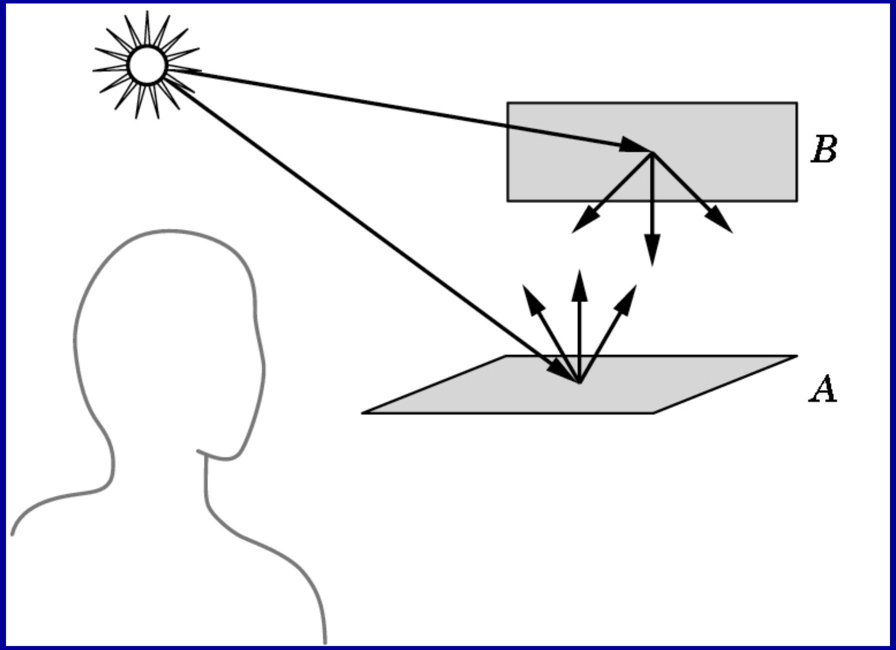


Do we need this?

My advisor told me that we didn’t need color displays for normal computing…

Could have too much dynamic range however (eyes have to adjust)

Shading: Illumination

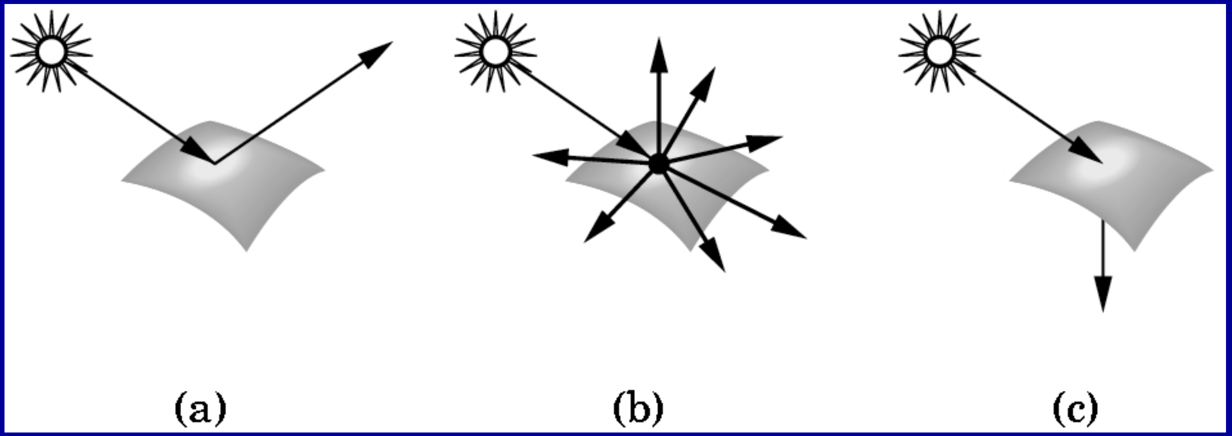


Light Sources emit light

- EM spectrum
- Position and direction

Surfaces reflect light

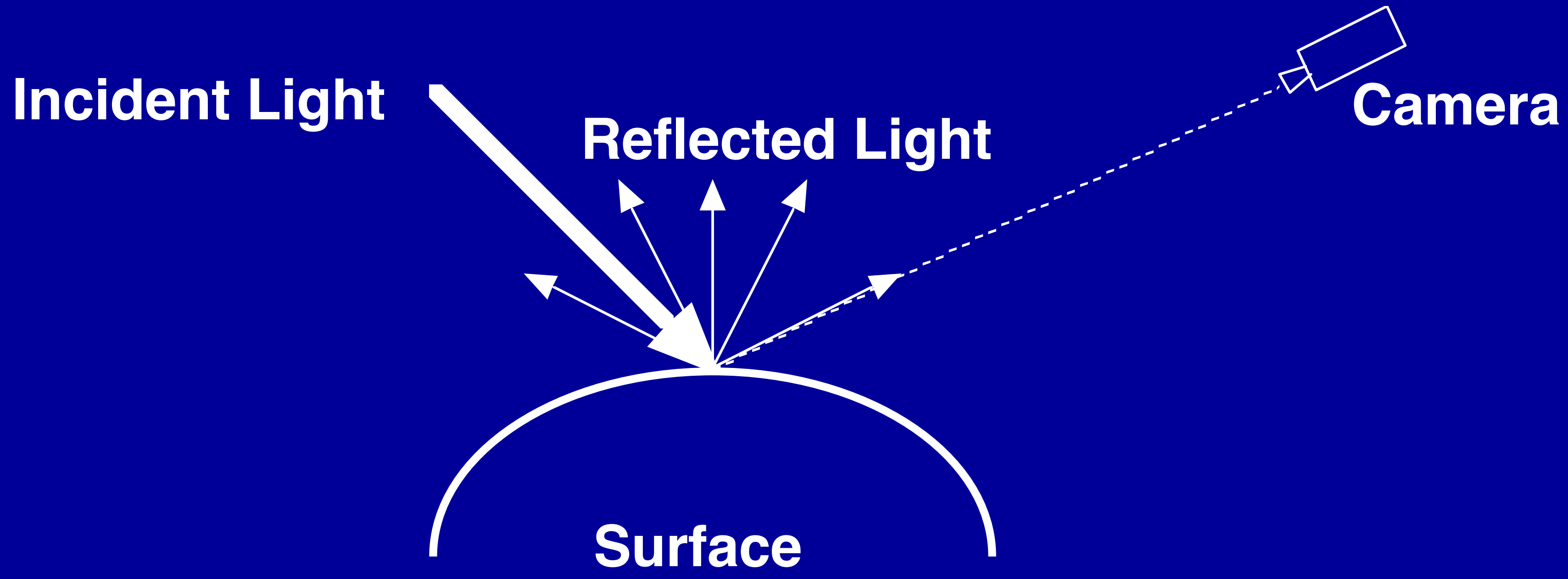
- Reflectance
- Geometry (position, orientation, micro-structure)
- Absorption
- Transmission



Illumination determined by the interactions between light sources and surfaces

Surface Reflection

- When light hits an opaque surface some is absorbed, the rest is reflected (some can be transmitted too--but ignore that for now)
- The reflected light is what we see
- Reflection is not simple and varies with material
  - the surface's micro structure define the details of reflection
  - variations produce anything from bright specular reflection (mirrors) to dull matte finish (chalk)



What we will learn about today

Light (with color)

Specular highlights

Shadows

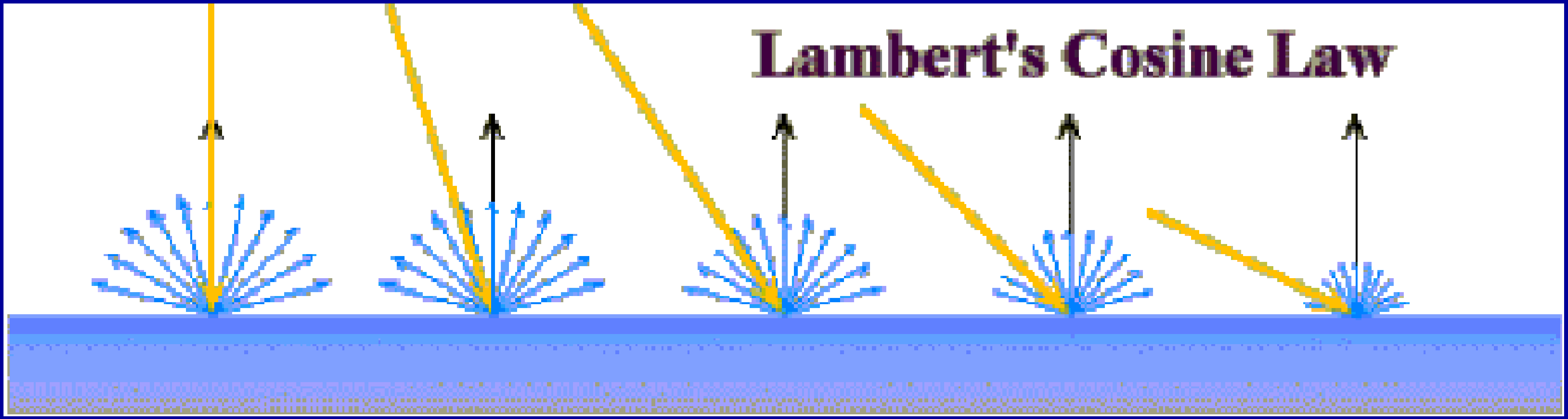
No transmission of light through surfaces

No reflections from other surfaces

Diffuse Reflection

- Simplest kind of reflector (also known as *Lambertian Reflection*)
- Models a matte surface -- rough at the microscopic level
- Ideal diffuse reflector
  - incoming light is scattered equally in all directions
  - viewed brightness does not depend on viewing direction
  - brightness *does* depend on direction of illumination

Illumination direction



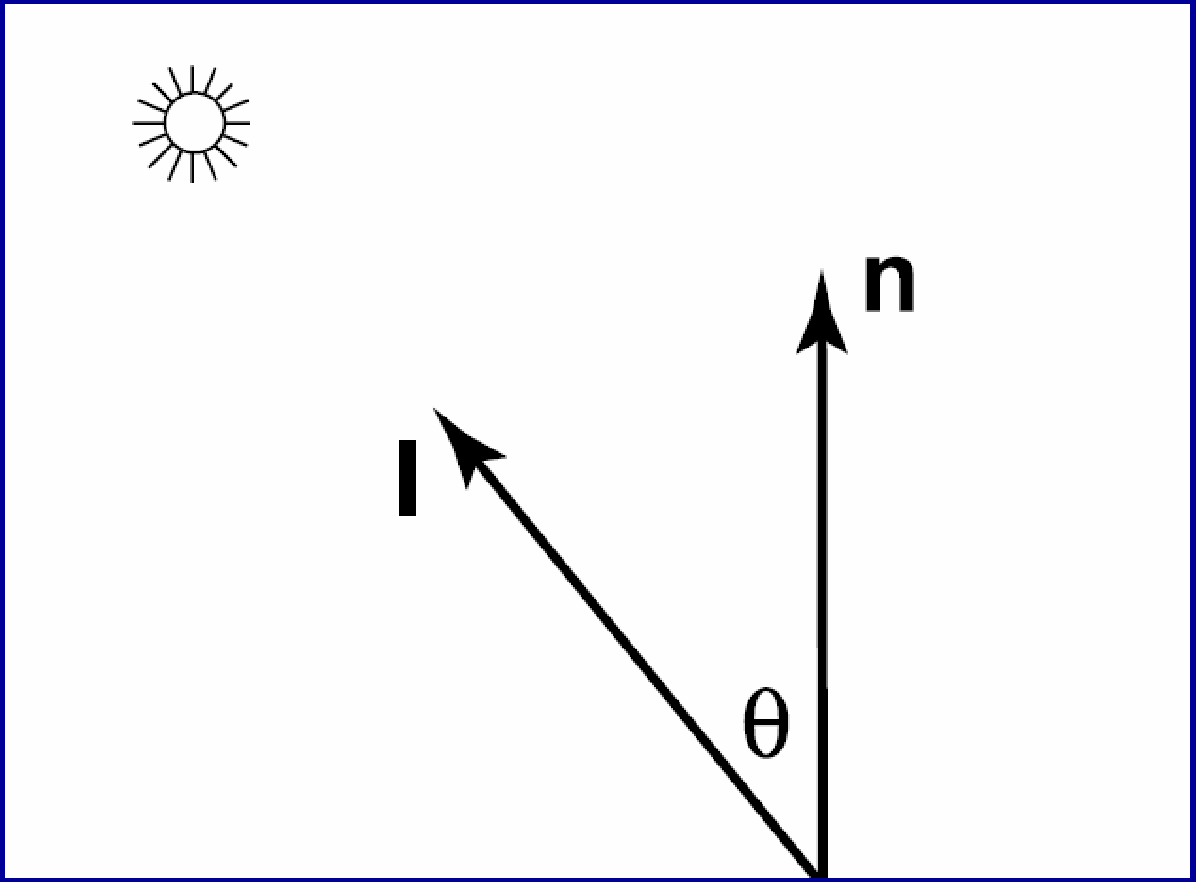
Lambertian Shading Model

$$c \propto \cos\theta$$

$$c \propto n \bullet l$$

n	: surface normal
l	: direction to light
θ	: Light/Normal angle

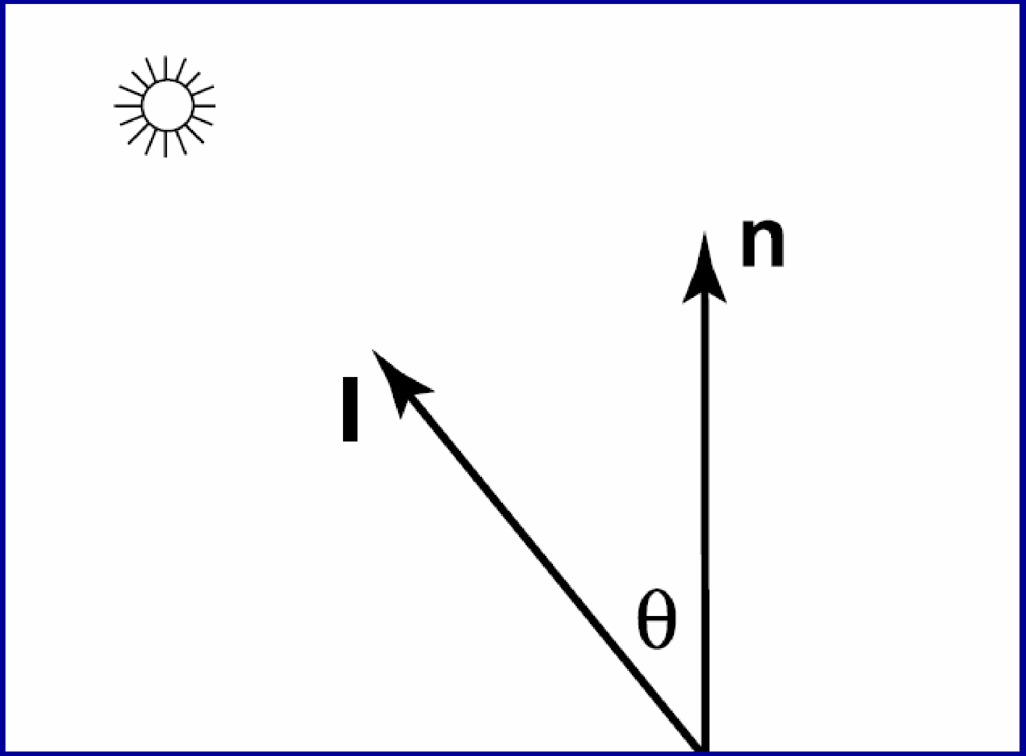
$$\cos\theta = \frac{n \bullet l}{|n||l|}$$



Lambert’s Law

$$
\begin{aligned}
I_{diffuse} &= k_d I_{light} \cos\theta \\
&= k_d I_{light} (n \bullet l)
\end{aligned}
$$

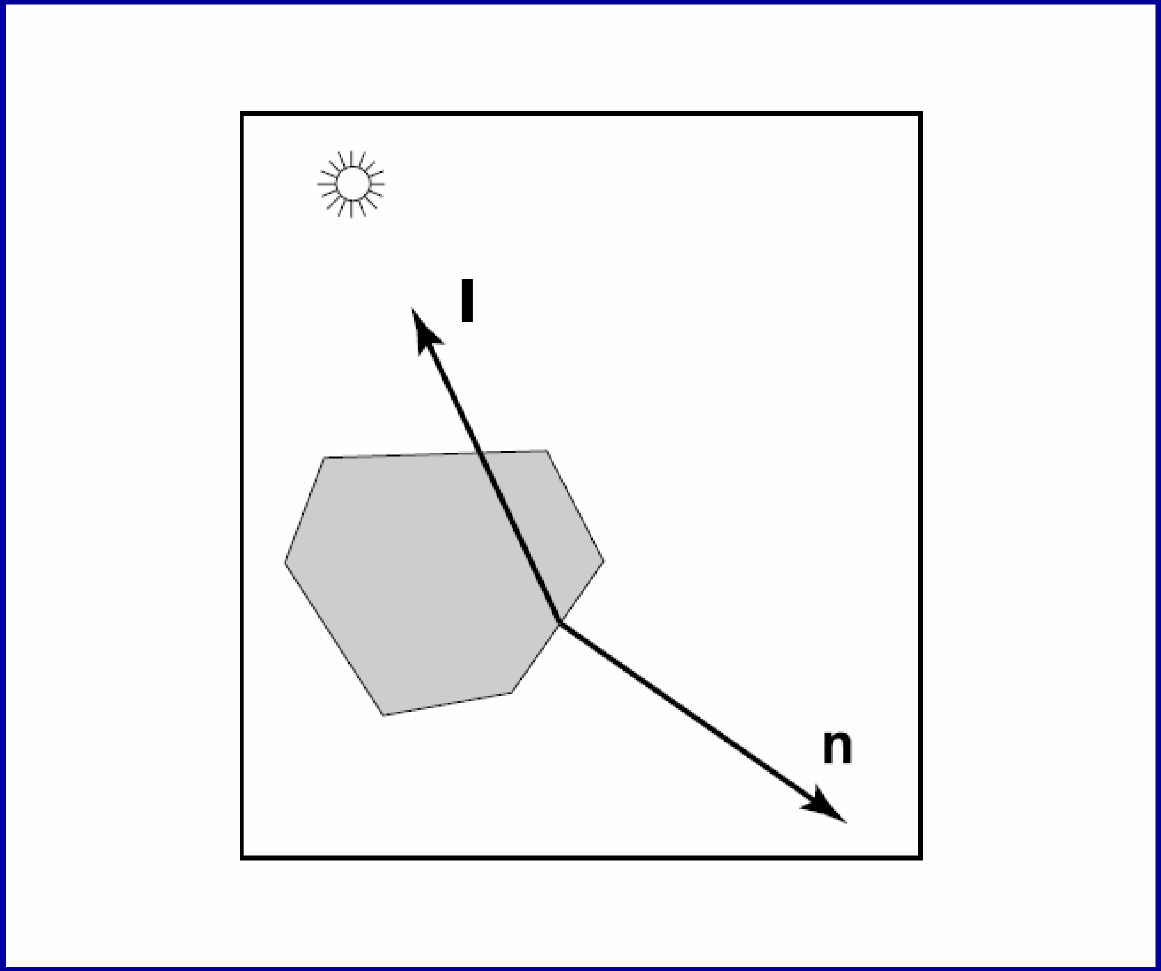
I_{light}	:	Light Source Intensity
k_d	:	Surface reflectance coefficient in [0,1]
θ	:	Light/Normal angle

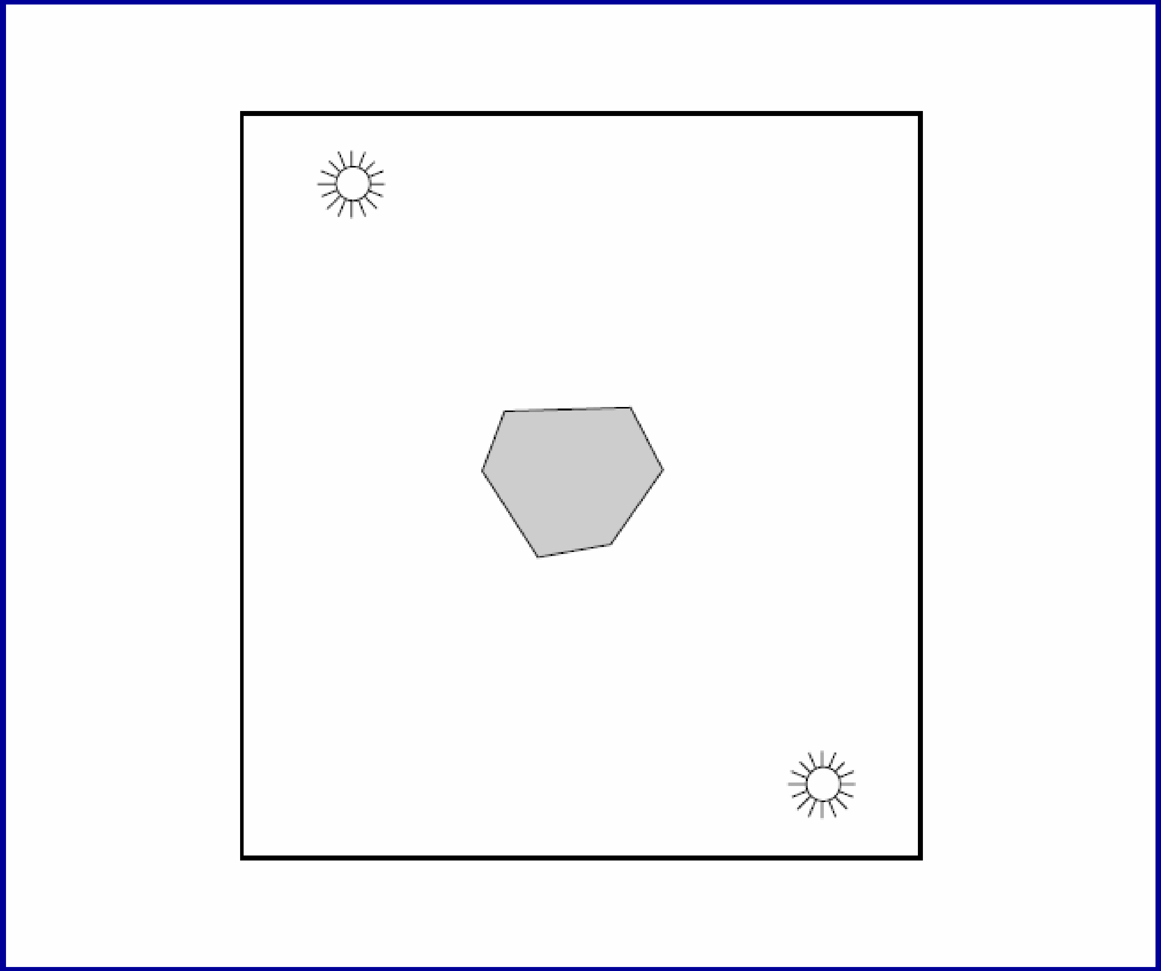


What happens for surfaces facing away from the light?

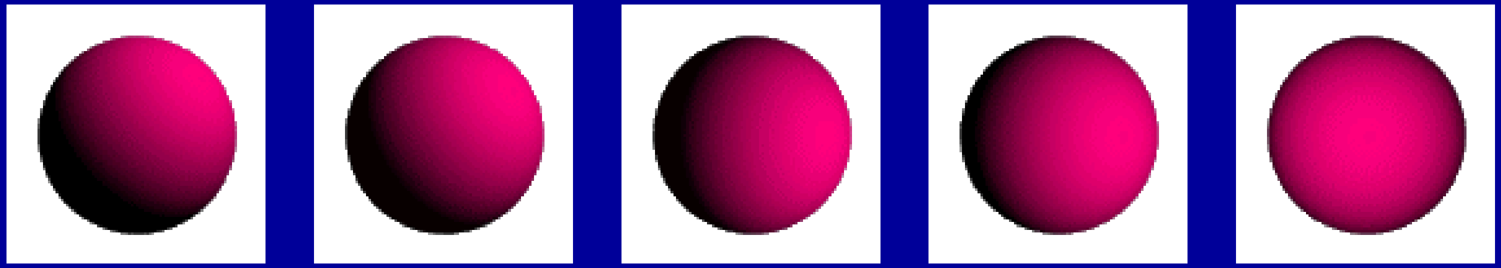
$$c = c_r c_l \max(0, n \bullet l)$$

$$c = c_r c_l |n \bullet l|$$ Two-sided light





Examples of Diffuse Illumination



Same sphere lit diffusely from different lighting angles

What happens with surfaces facing away from the light?

- Pitch black—not exactly realistic

How to solve?

- Several light sources—dim light source at eye, for example
- Ambient light

Ambient + Diffuse Reflection

$$I_{d+a} = k_a I_a + k_d I_{light} (n \bullet l)$$

$$c = c_r (c_a + c_l \max(0, n \bullet l))$$

I_a : Ambient light intensity (global)

k_a : Ambient reflectance (local)

Diffuse illumination plus a simple ambient light term

- a TRICK to account for a background light level caused by multiple reflections from all objects in the scene (less harsh appearance)

Further Simple Illumination Effects

- Light attenuation:
  - light intensity falls off with the square of the distance from the source - so we add an extra term for this

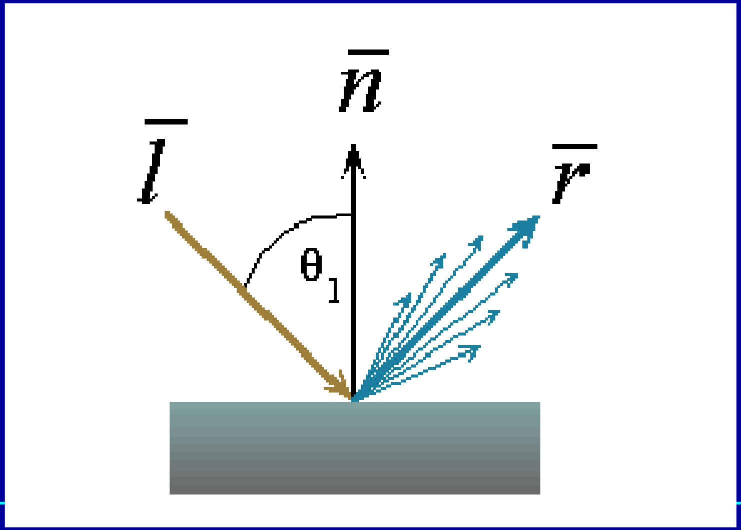
$$I_{d+a} = k_a I_a + f_{att} k_d I_{light} (n \bullet l) \qquad \text{where} \quad f_{att} = \frac{1}{d^2}$$

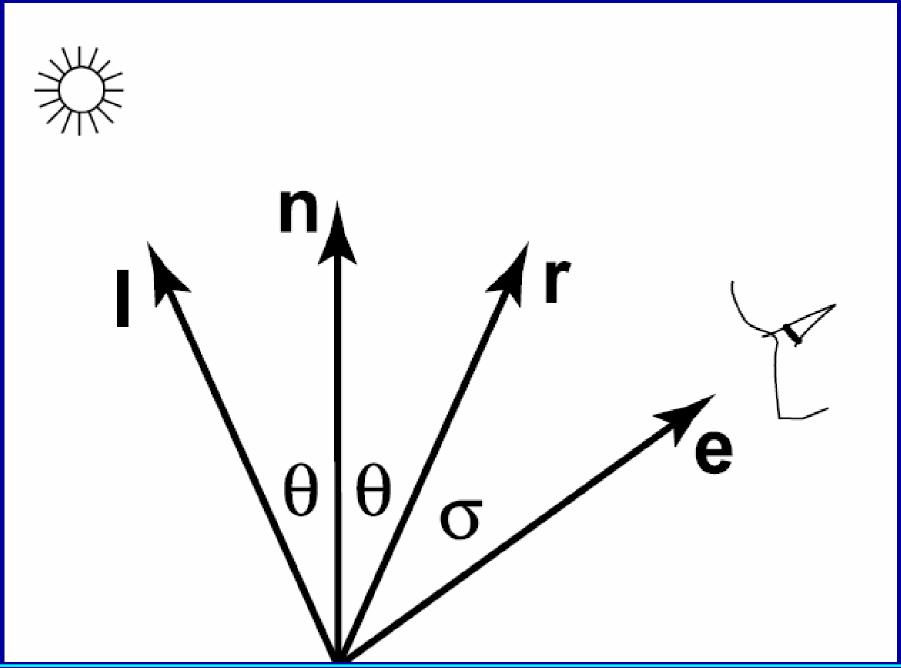
with d the light source to surface distance—more complicated formulae are possible

- Colored lights and surfaces:
  - just have three separate equations for RGB
- Atmospheric attenuation:
  - use viewer-to-surface distance to give extra effects
  - the distance is used to blend the object’s radiant color with a “far” color (e.g., a nice hazy gray)

Specular Reflection (Phong Shading)

- Shiny surfaces change appearance when viewpoint is varied
  - specularities (highlights) are view-dependent
  - caused by surfaces that are microscopically smooth (tile floors, gloss paint, whiteboards)
- For shiny surfaces part of the incident light reflects coherently
  - an incoming ray is reflected in a single direction (or narrow beam)
  - direction is defined by the incoming direction and the surface normal
  - when σ is near zero, viewer sees reflection





Phong Illumination

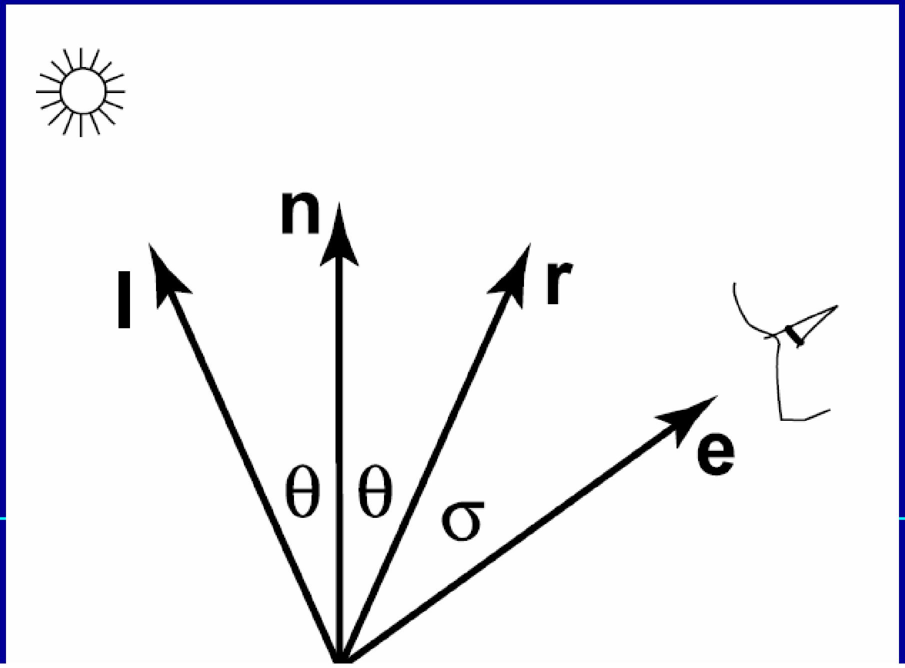
- One function that approximates specular falloff is called the *Phong Illumination* model

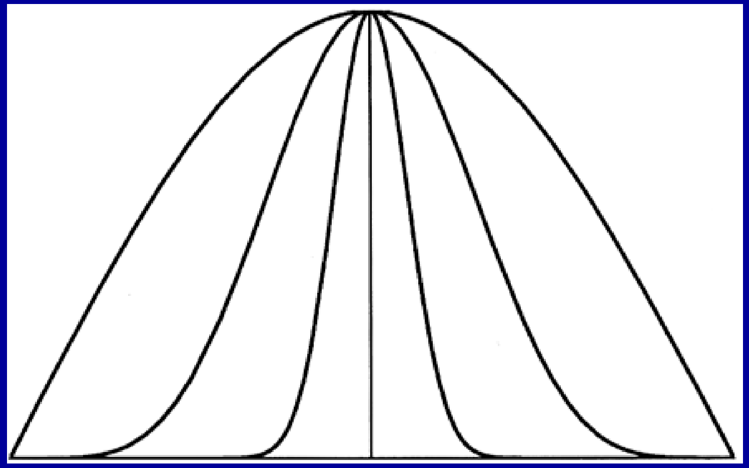
$$c = c_l (e \bullet r)$$

$$c = c_l \max(0, e \bullet r)^p$$

$$I_{specular} = k_s I_{light} (e \bullet r)^p$$

k_s	: Specular reflectance
p	: Rate of specular falloff (phong exponent)

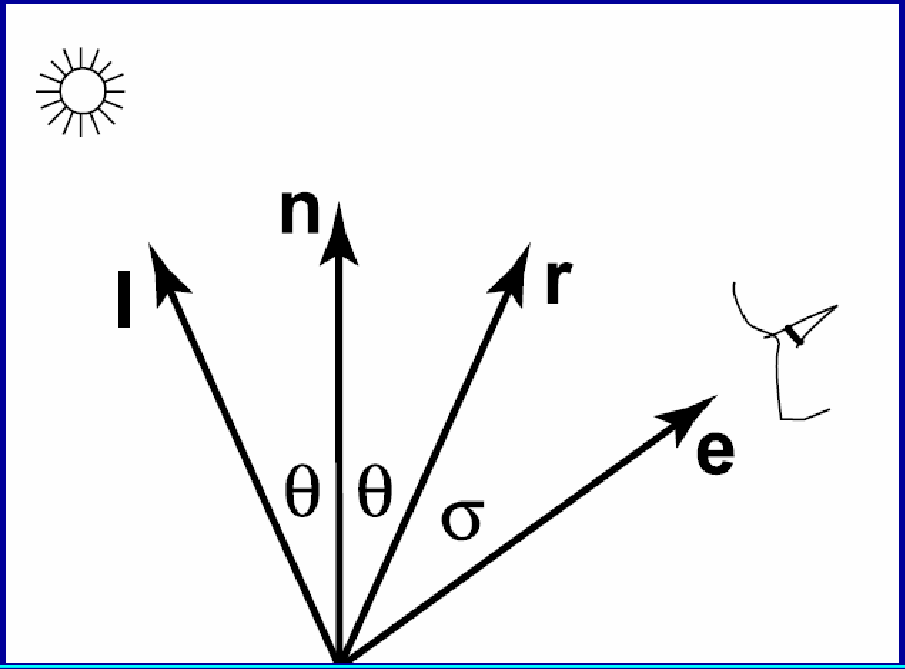




Greater p , more focused beam

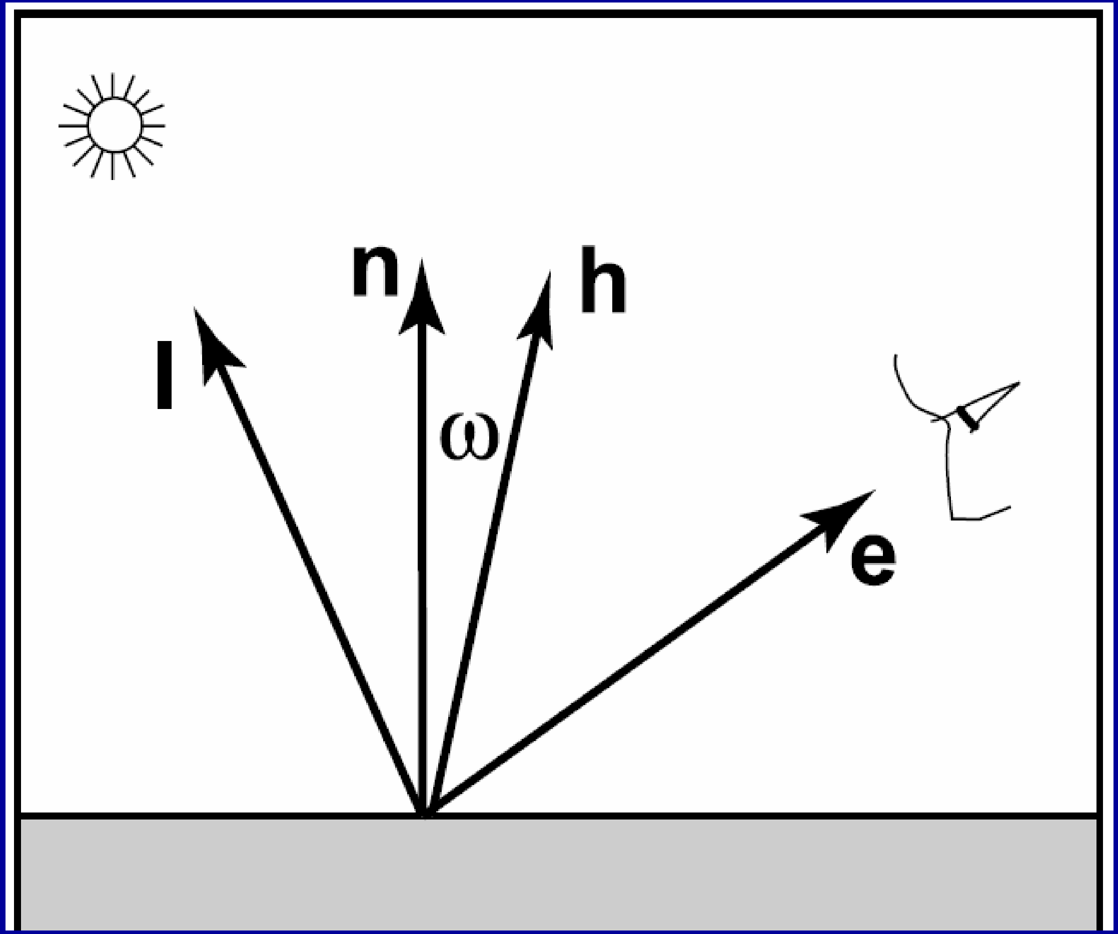
Computing the Reflected Ray

blackboard

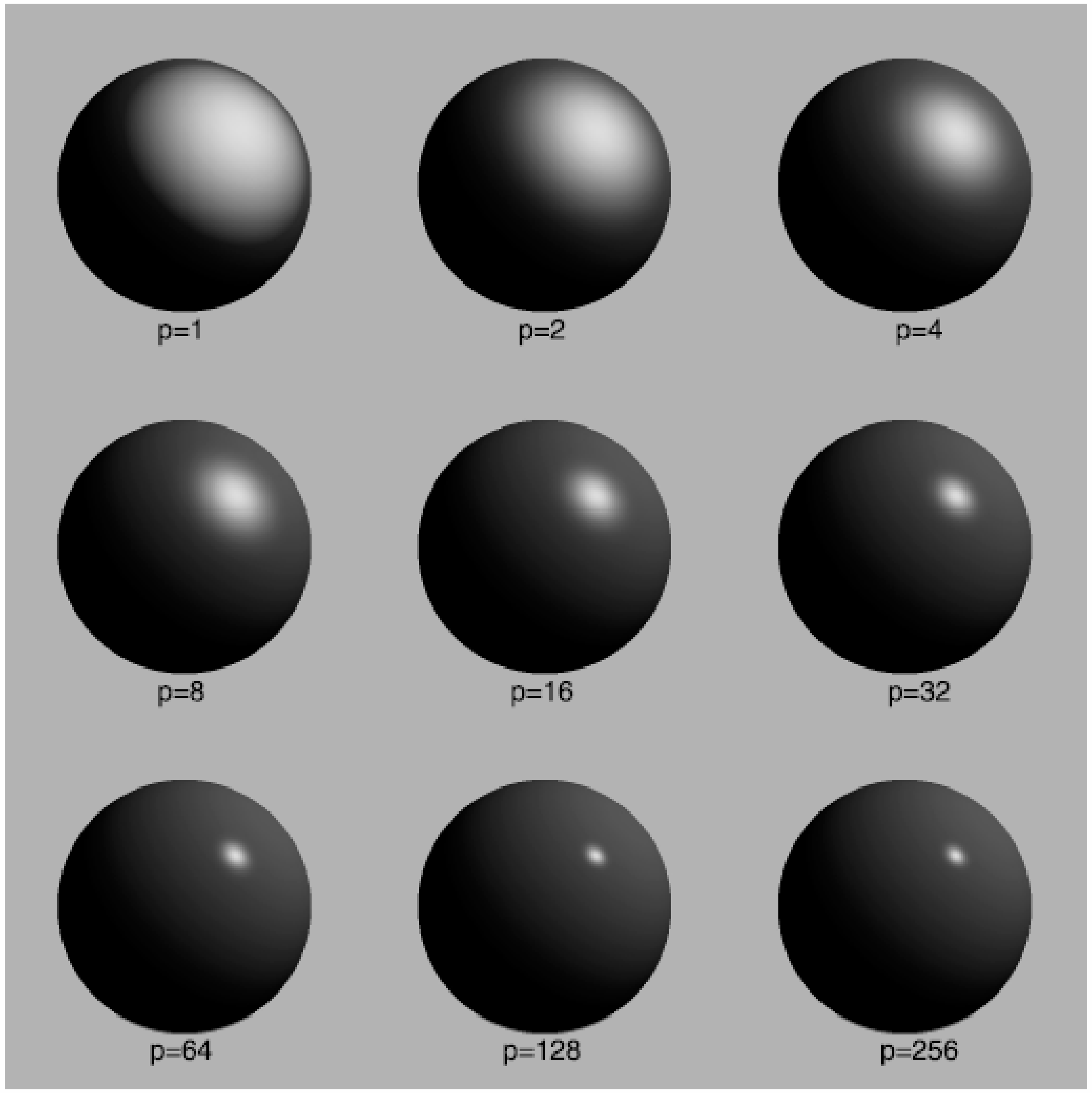


Or an approximation to the Reflected Ray

blackboard

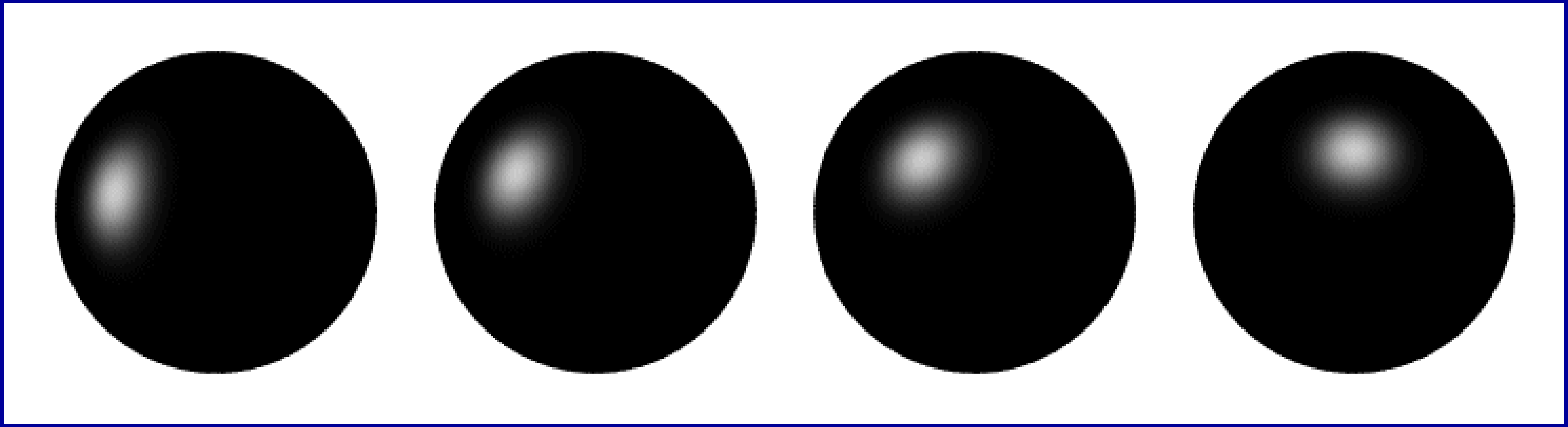


Phong Illumination

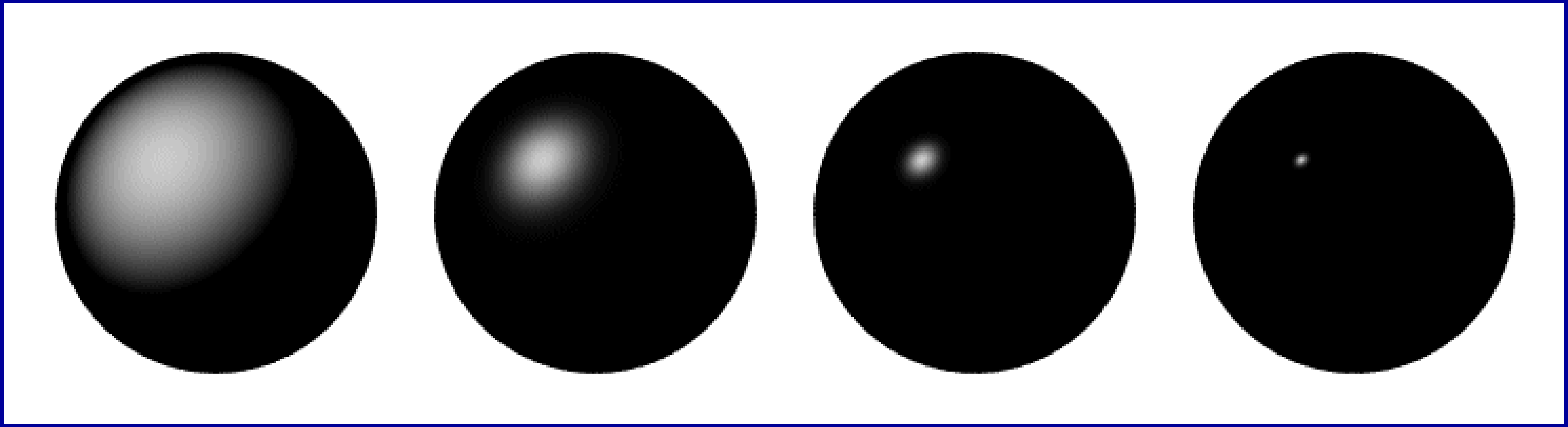


No real physical basis but provides approximately the right answer

Phong Illumination

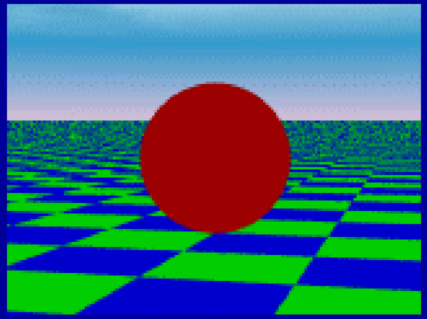


Moving the light source

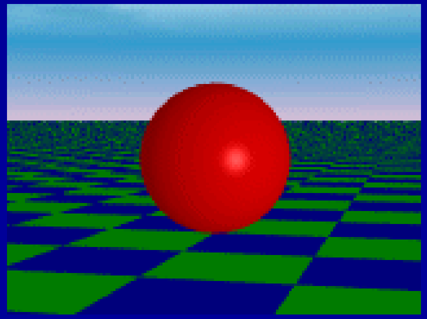


Changing p

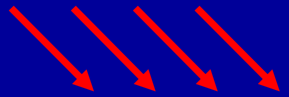
Types of Light Sources

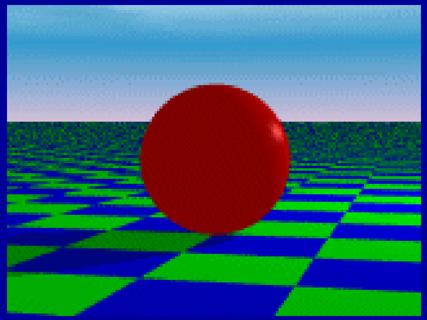


- Ambient: equal light in all directions
  - a hack to model inter-reflections

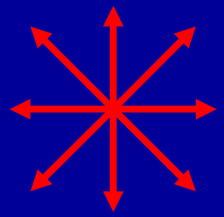


- Directional: light rays oriented in same direction
  - good for distance light sources (sunlight)

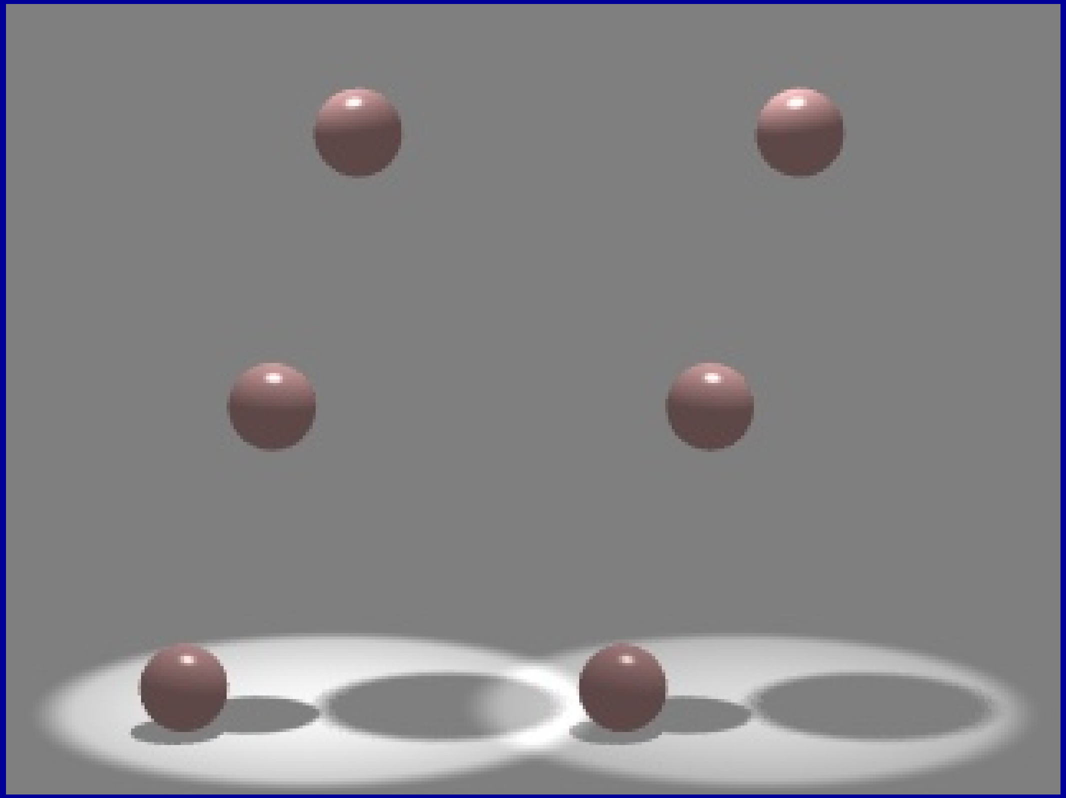


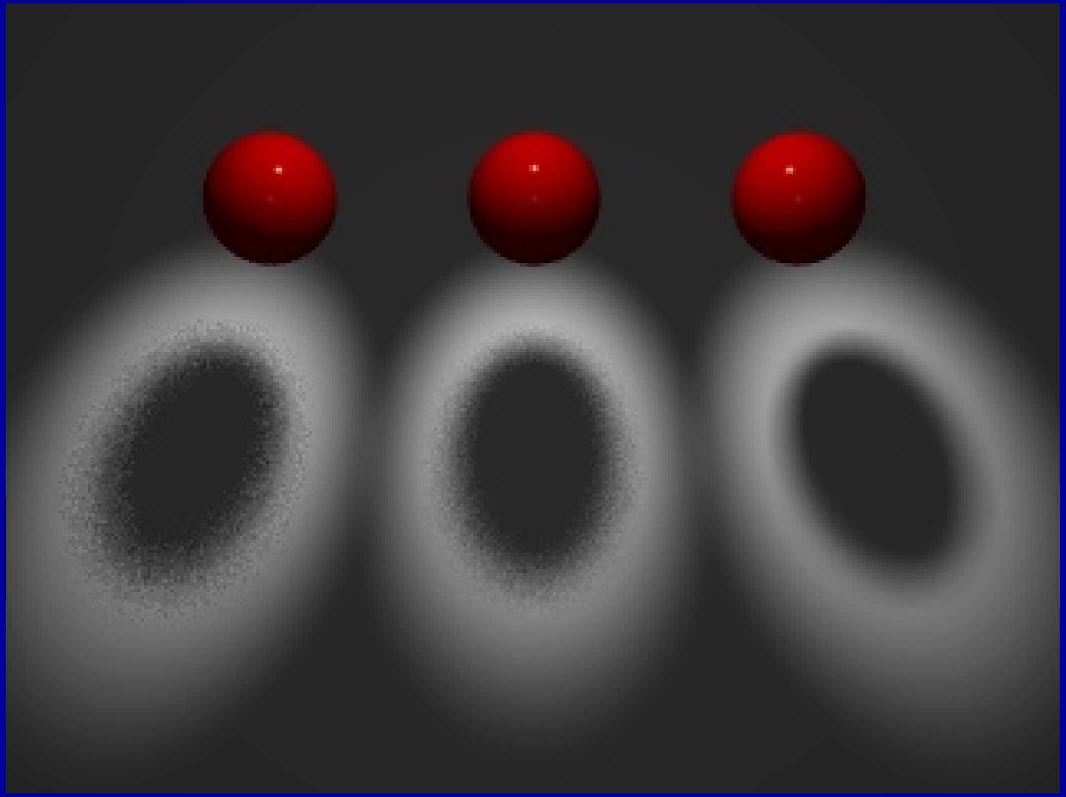


- Point: light rays diverge from a single point
  - approximation to a light bulb (but harsher)



More Light Sources





- Spotlight: point source with directional fall-off
  - intensity is maximal along some direction D, falls off away from D
  - specified by color, point, direction, fall-off parameters
- Area Source: Luminous 2D surface
  - radiates light from all points on its surface
  - generates soft shadows

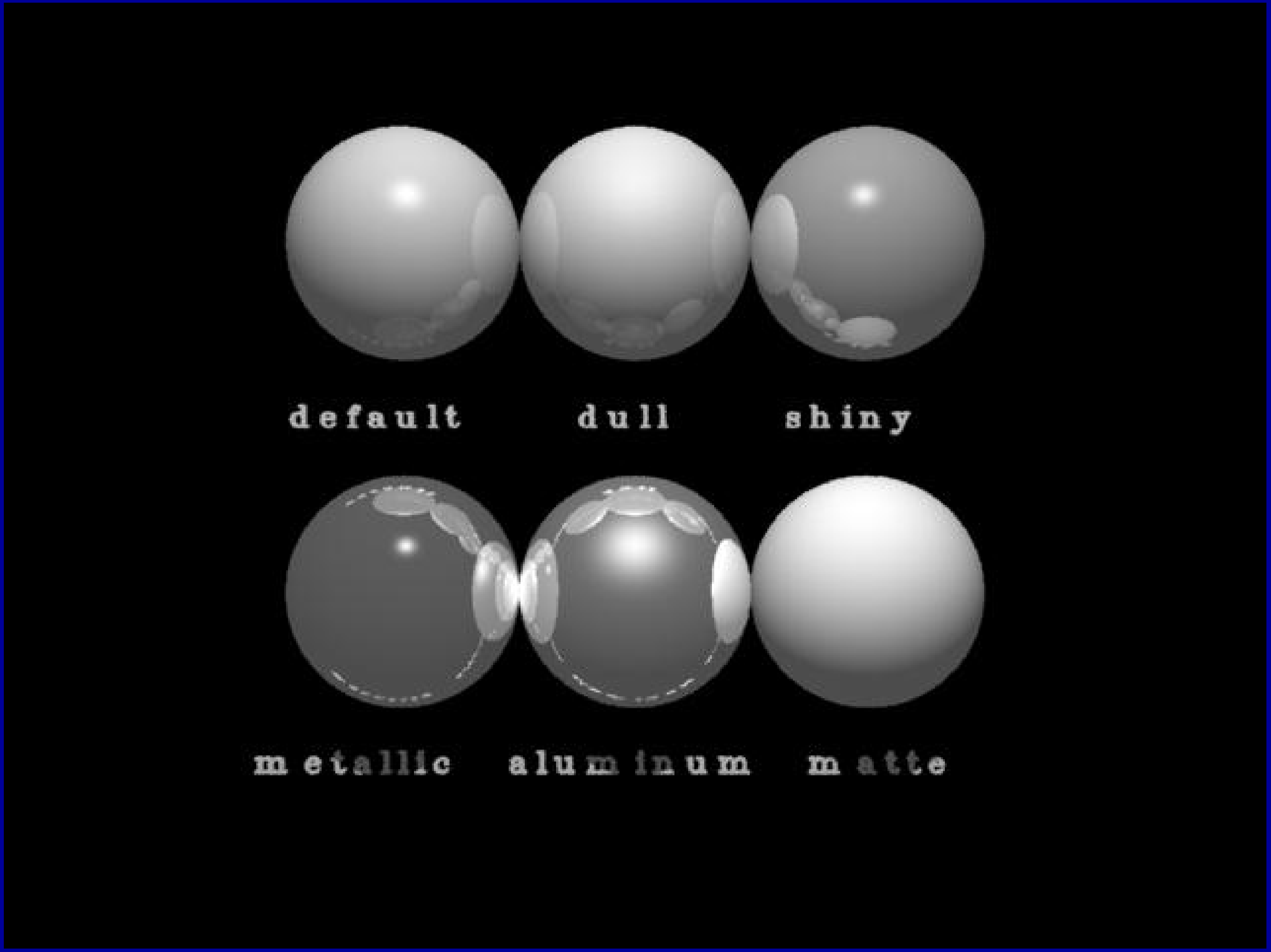
Putting It All Together

- Combining ambient, diffuse, and specular illumination

$$I = k_a I_a + f_{att} I_{light} \left[k_d \cos\theta + k_s (\cos\phi)^p \right]$$

- For multiple light sources
  - Repeat the diffuse and specular calculations for each light source
  - Add the components from all light sources
  - The ambient term contributes only once
- The different reflectance coefficients can differ.
  - Simple “metal”: k_a and k_d share material color, k_s is white
  - Simple plastic: k_s also includes material color
  - More on these kinds of parameters when we talk about reflectance models in a few weeks

Some Examples



OpenGL Materials

```
GLfloat white8[] = {.8, .8, .8, 1.}, white2 = {.2,.2,.2,1.},black={0.,0.,0.};
GLfloat mat_shininess[] = {50.};          /* Phong exponent */

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, black);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, white8);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, white2);
glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);
```

OpenGL Lighting

```
GLfloat white[] = {1., 1., 1., 1.};
GLfloat light0_position[] = {1., 1., 5., 0.}; /* directional light (w=0) */

glLightfv(GL_LIGHT0, GL_POSITION, light0_position);
glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
glLightfv(GL_LIGHT0, GL_SPECULAR, white);
glEnable(GL_LIGHT0);

glEnable(GL_NORMALIZE);     /* normalize normal vectors */
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);    /* two-sided lighting*/

glEnable(GL_LIGHTING);
```

Transmission with Refraction

- Refraction:
  - the bending of light due to its different velocities through different materials
- Refractive index:
  - light travels at speed *c/n* in a material of refractive index *n*
  - *c* is the speed of light in a vacuum
  - varies with wavelength hence rainbows and prisms

MATERIAL	INDEX OF REFRACTION
Air/Vacuum	1
Water	1.33
Glass	about 1.5
Diamond	2.4

Snell's Law

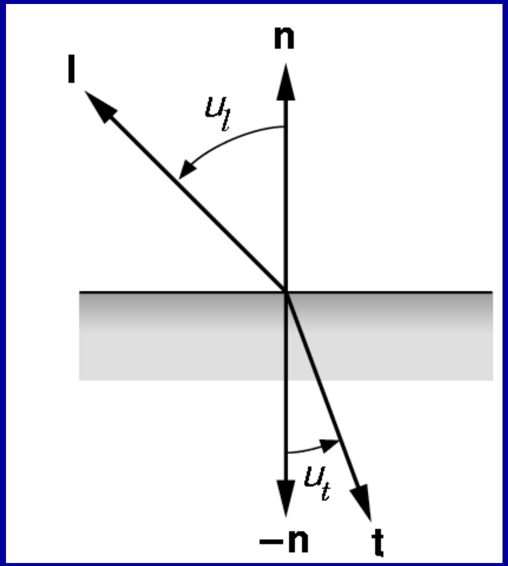
Light bends by the physics *principle of least time*

- light travels from point A to point B by the fastest path
- when passing from a material of index n_1 to one of index n_2 *Snell's law* gives the angle of refraction:
  $n_1 \sin \theta_1 = n_2 \sin \theta_2$
  where θ_1 and θ_2 are the angles from perpendicular

When traveling into a denser material (larger *n*), light bends to be more perpendicular (eg air to water) and vice versa

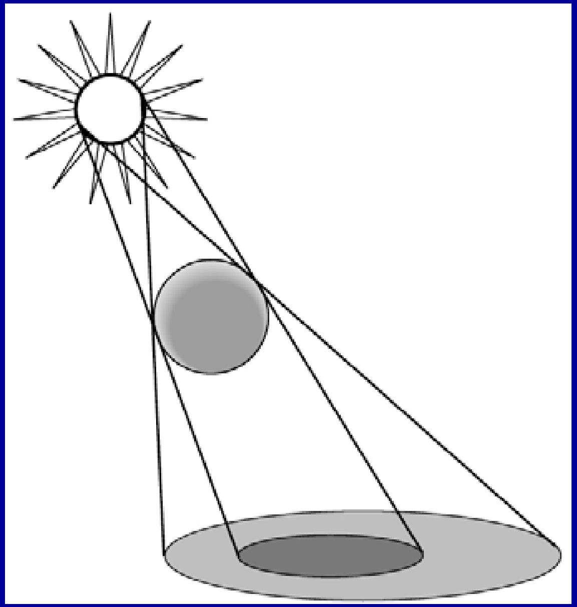
- light travels further in the faster material
- if the indices are the same the light doesn't bend

When traveling into a less dense material total internal reflection occurs if $\theta_1 > \sin^{-1}(n_2/n_1)$



Shadows

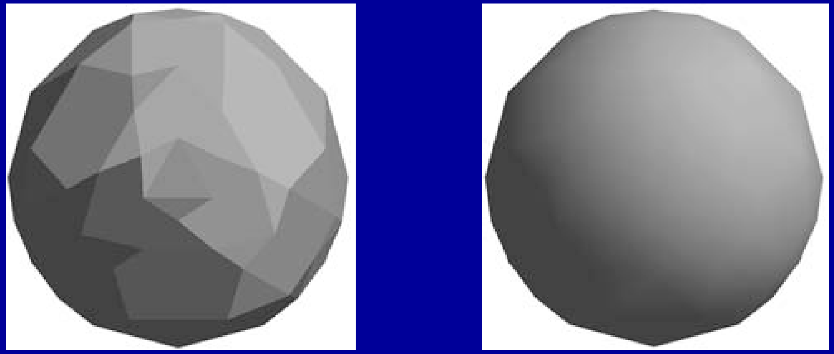
- Shadows occur where objects are hidden from a light source
  - omit any intensity contribution from hidden light sources
- Working out what it hidden is simply a visibility problem
  - can the light source see the object?
  - use the z-buffer shadow algorithm:
    - run the algorithm from the light source’s viewpoint
    - save the z-buffer as the shadow buffer
    - run the real z-buffer algorithm, transforming each point into the light source’s coordinates and comparing the z value against the shadow buffer



Shading

Given an equation to calculate surface radiance, we still must apply it to the real model

- Usually performed during scan conversion
- There are efficient methods for doing this quickly (which we will discuss in more detail later in the semester



blackboard

Flat shaded

Gouraud: Normal at vertex is average of normals for adjacent faces

Phong: interpolate normals instead of intensities

Uniformly shaded surfaces are still unrealistic

Real objects have surface features, or texture

One option: use a huge number of polygons with appropriate surface coloring and reflectance characteristics

Texture mapping gets you more detail at less cost

- Assign radiance based on an image

Or use *Procedural shaders* to specify any function you want to define radiance

- Generate radiance on the fly, during shading