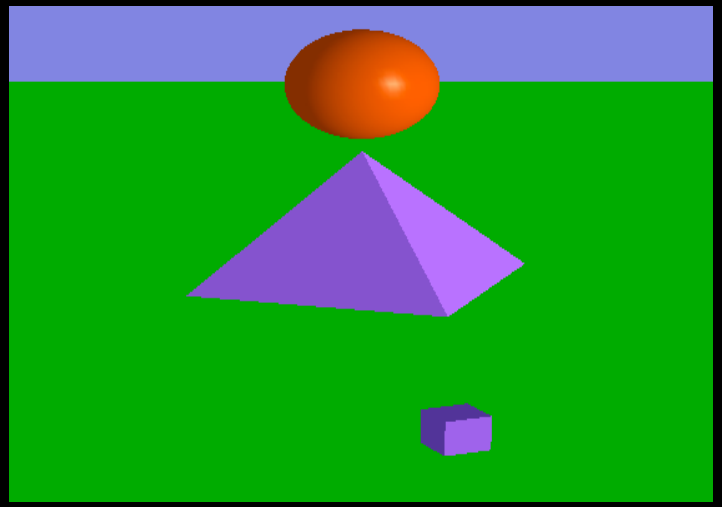
15-462, Fall 2004

Nancy Pollard
Mark Tomczak

Shading



Shading Concepts
Shading Equations
Lambertian, Gouraud shading
Phong Illumination Model
Non-photorealistic rendering

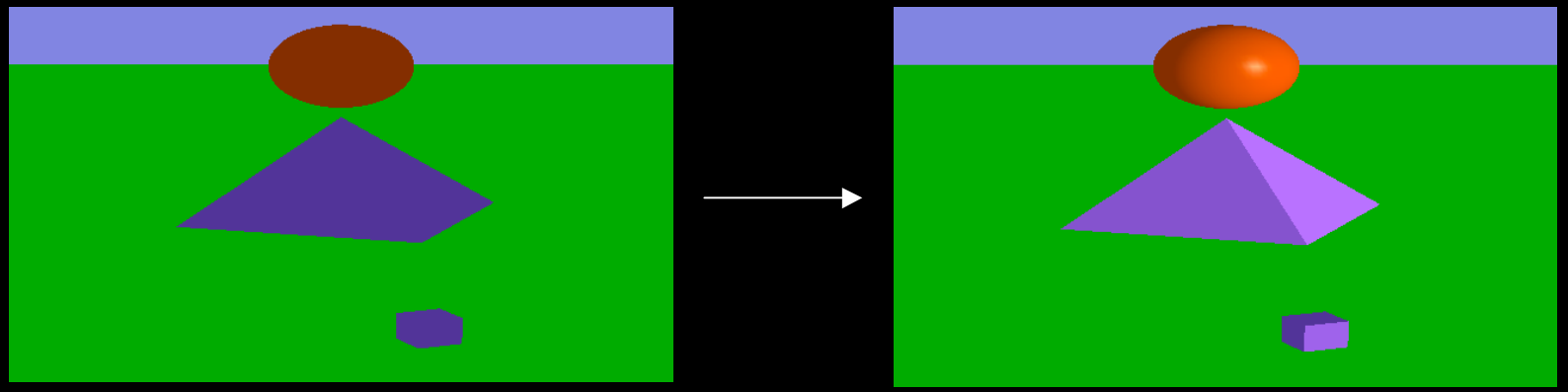
[Shirly, Ch. 8]

Announcements

- Written assignment #2 due Tuesday
  - Handin at beginning of class
- Programming assignment #2 out Tuesday

Why Shade?

- Human vision uses shading as a cue to form, position, and depth
- Total handling of light is very expensive
- Shading models can give us a good approximation of what would “really” happen, much less expensively
- Average and approximate



Outline

- Lighting models (OpenGL oriented)
  - Light styles
  - Lambertian shading
  - Gouraud shading
- Reflection models (Phong shading)
- Non-Photorealistic rendering

Common Types of Light Sources

- Ambient light: no identifiable source or direction
- Point source: given only by point
- Distant light: given only by direction
- Spotlight: from source in direction
  - Cut-off angle defines a cone of light
  - Attenuation function (brighter in center)
- Light source described by a luminance
  - Each color is described separately
  - $I = [I_r \;\; I_g \;\; I_b]^T$ (I for intensity)
  - Sometimes calculate generically (applies to r, g, b)

Ambient Light

- Intensity is the same at all points
- This light does not have a direction (or .. it is the same in all directions)

$$\mathbf{I}_a = \begin{bmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{bmatrix}$$

Point Source

- Given by a point p_0
- Light emitted from that point equally in all directions

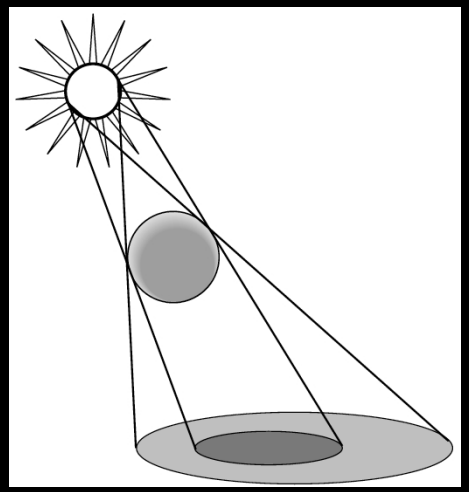
$$\mathbf{I}(\mathbf{p}_0) = \begin{bmatrix} I_r(\mathbf{p}_0) \\ I_g(\mathbf{p}_0) \\ I_b(\mathbf{p}_0) \end{bmatrix}$$

- Intensity decreases with square of distance

$$\mathbf{I}(\mathbf{p}, \mathbf{p}_0) = \frac{1}{|\mathbf{p} - \mathbf{p}_0|^2} \mathbf{I}(\mathbf{p}_0)$$

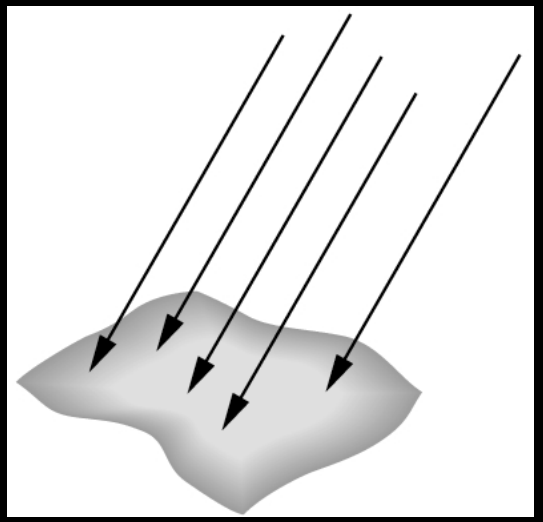
One Limitation of Point Sources

- Shading and shadows inaccurate
- Example: penumbra (partial “soft” shadow)



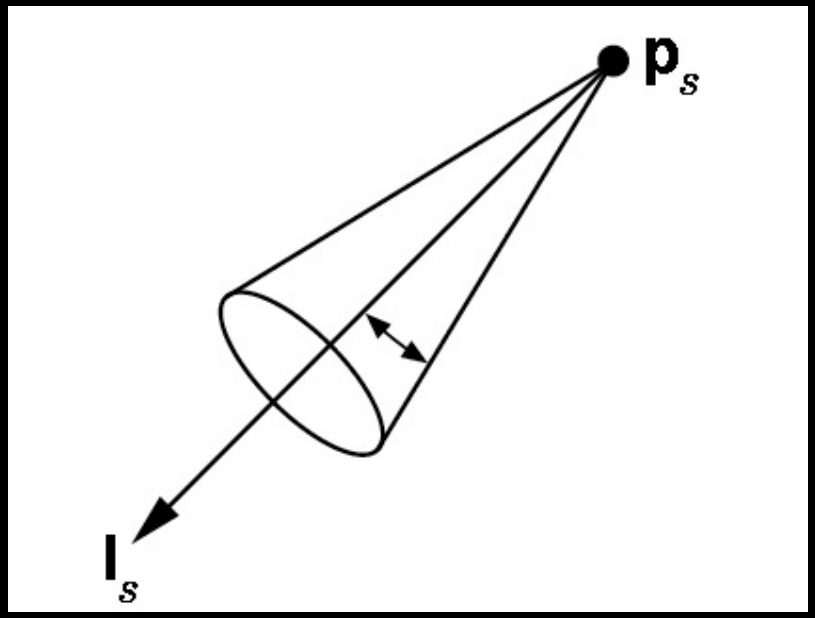
Distant Light Source

- Given by a vector v
- Intensity does not vary with distance (all distances are the same .. infinite!)



Spotlight

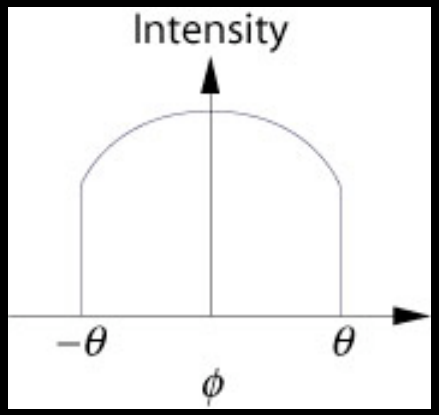
- Most complex light source in OpenGL
- Light still emanates from point
- Cut-off by cone determined by angle q



Spotlight Attenuation

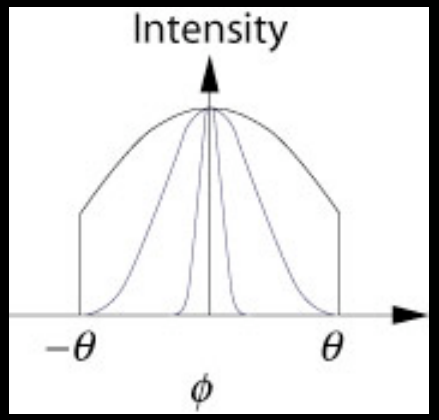
- Spotlight is brightest along l_s
- Vector v with angle f from p to point on surface
- Intensity determined by cos f
- Corresponds to projection of v onto l_s
- Spotlight exponent e determines rate of dropoff

$$I = \cos^e(\phi) = (\mathbf{v} \cdot \mathbf{l}_s)^e$$



for e = 1

for e > 1
curve narrows

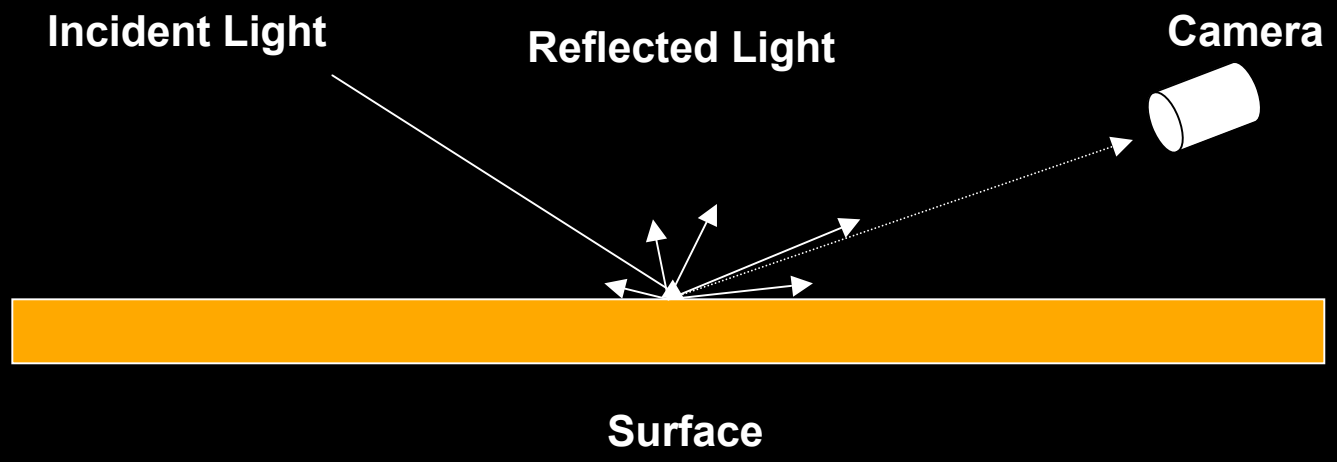


The Life of a Photon

What can happen to a photon that interacts with an object?

Surface Reflection

- When light hits an opaque surface some is absorbed, the rest is reflected (some can be transmitted too--but never mind 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)

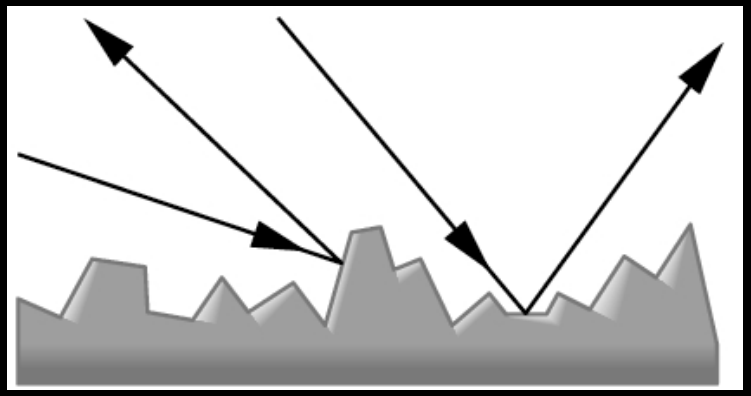


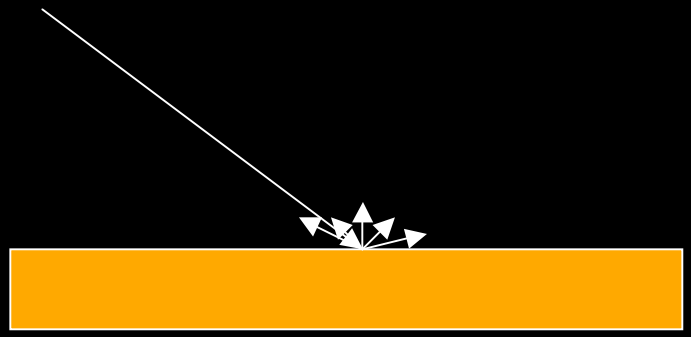
Basic Calculation

- Calculate each primary color separately
- Start with global ambient light
- Add reflections from each light source
- Clamp to [0, 1]
- Reflection decomposed into
  - Ambient reflection
  - Diffuse reflection
  - Specular reflection
- Based on ambient, diffuse, and specular lighting and material properties

Lambertian (Diffuse) Reflection

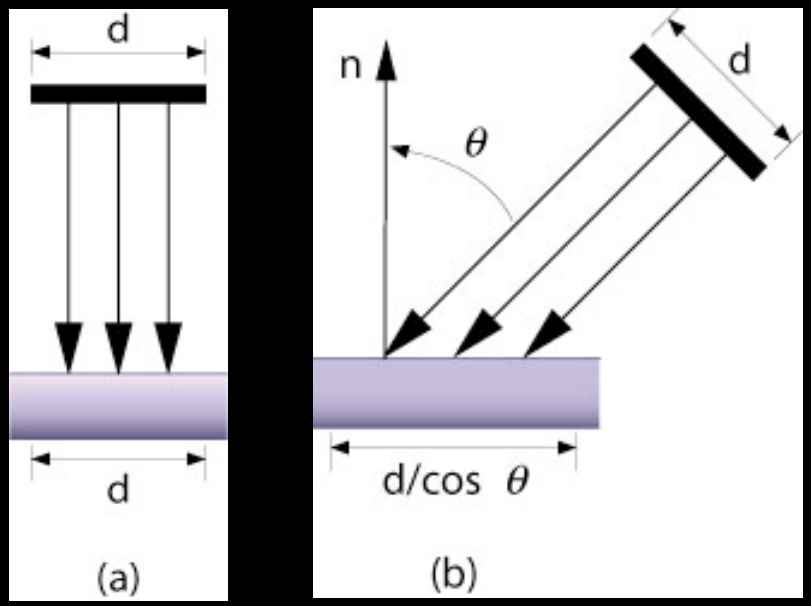
- Diffuse reflector scatters light
- Assume equally all direction
- Called Lambertian surface
- Diffuse reflection coefficient k_d, $0 \cdot k_d \cdot 1$
- Angle of incoming light still critical





Lambert's Law

- Intensity depends on angle of incoming light
- Recall
  - l = unit vector to light
  - n = unit surface normal
  - q = angle to normal
- cos q = l * n
- $I_d = k_n (l * n) L_d$
- With attenuation:

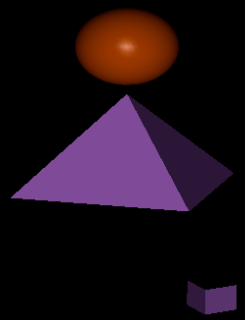


$$I_d = \frac{k_d}{a + bq + cq^2}(\mathbf{l} \cdot \mathbf{n}) L_d$$

q = distance to light source,
L_d = diffuse component of light

Small problem…

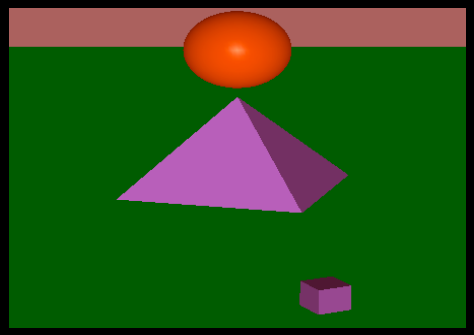
- Too dark!
- Everything is very starkly lit
- “Spooky”
- Why?



Ambient Light

- Reflected light (even diffuse reflection) reflects off of other surfaces
- Light is scattered by the air; does not always travel a straight path
- Modeling all that reflection and distortion would be very complicated
- *Simplify, Simplify*

  --Henry David Thoreau

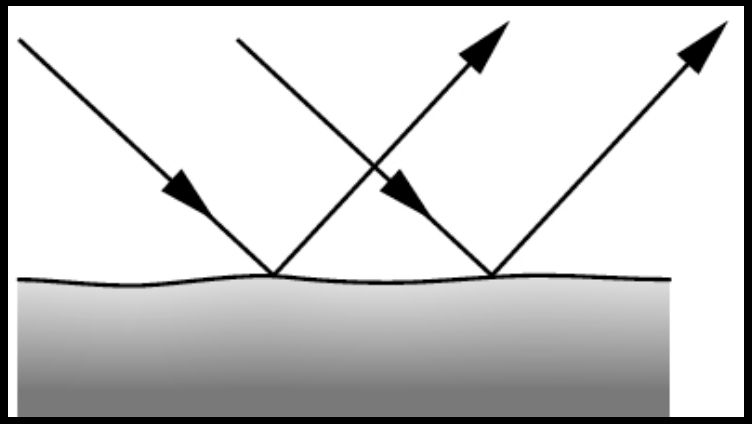


Ambient Reflection

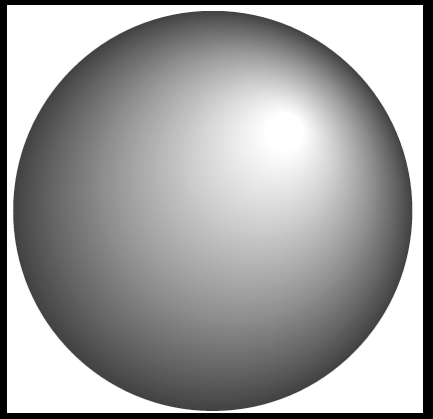
- Pretend some minimum light energy incident on every point in space from every direction
- Intensity of ambient light uniform at every point
- Ambient reflection coefficient k_a, $0 <= k_a <= 1$
- May be different for every surface and r,g,b
- Determines reflected fraction of ambient light
- L_a = ambient component of light source
- Ambient intensity $I_a = k_a L_a$
- Note: L_a is not a physically meaningful quantity

Specular Reflection

- Specular reflection coefficient k_s, $0 \cdot k_s \cdot 1$
- Shiny surfaces have high specular coefficient
- Used to model specular highlights
- Do not get mirror effect (need other techniques)



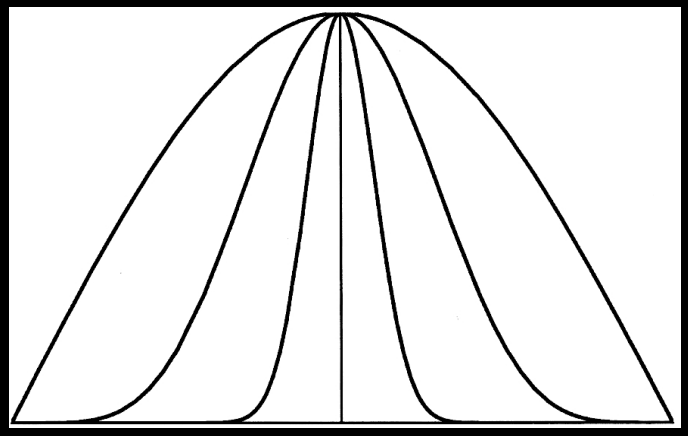
specular reflection



specular highlights

Shininess Coefficient

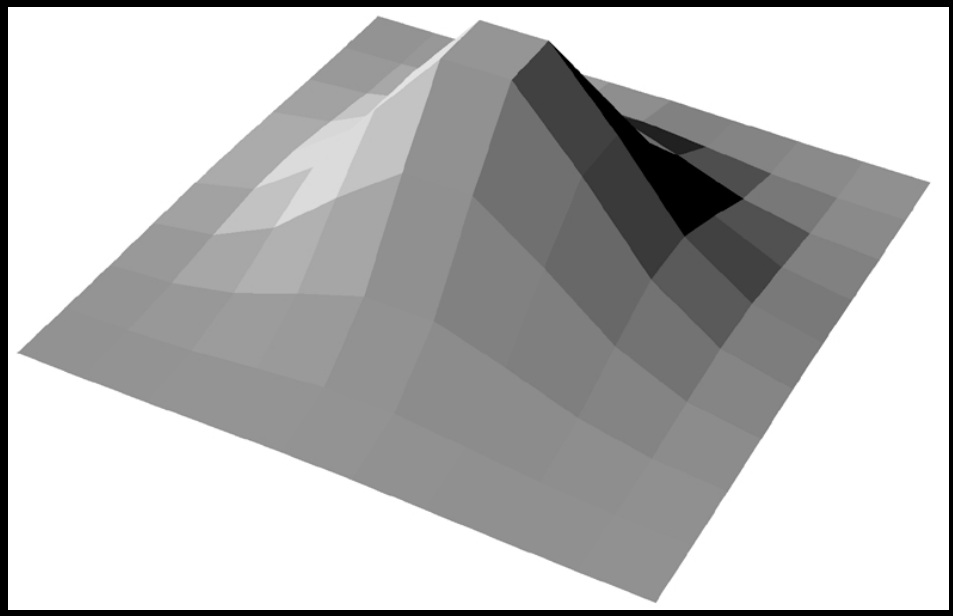
- L_s is specular component of light
- r is vector of perfect reflection of l about n
- v is vector to viewer
- f is angle between v and r
- $I_s = k_s L_s \cos^a f$
- a is shininess coefficient
- Compute cos f = r * v
- Requires |r| = |v| = 1
- Multiply distance term
- *Equation look familiar?*



Higher a is narrower

Flat Shading Assessment

- Inexpensive to compute
- Appropriate for objects with flat faces
- Less pleasant for smooth surfaces



Flat Shading and Perception

- *Lateral inhibition*: exaggerates perceived intensity
- *Mach bands*: perceived "stripes" along edges

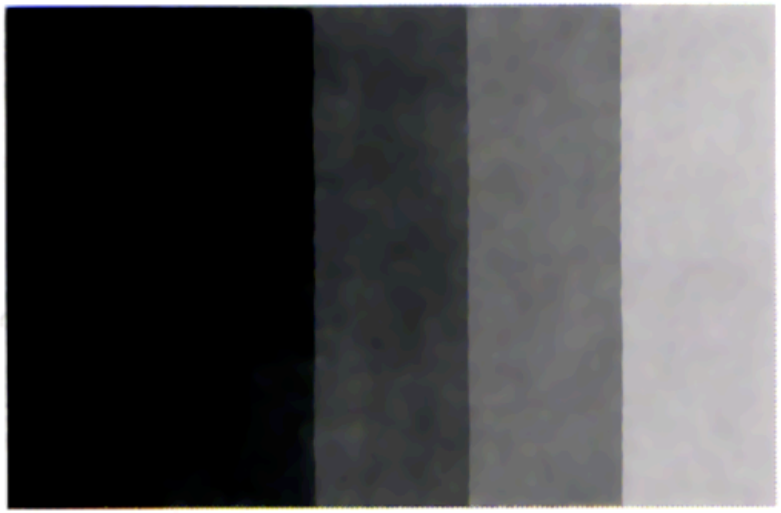


Figure 6.28 Step chart.

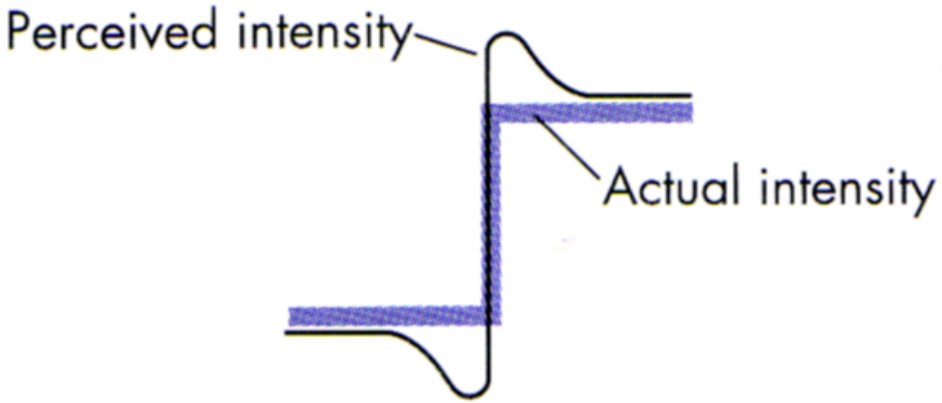
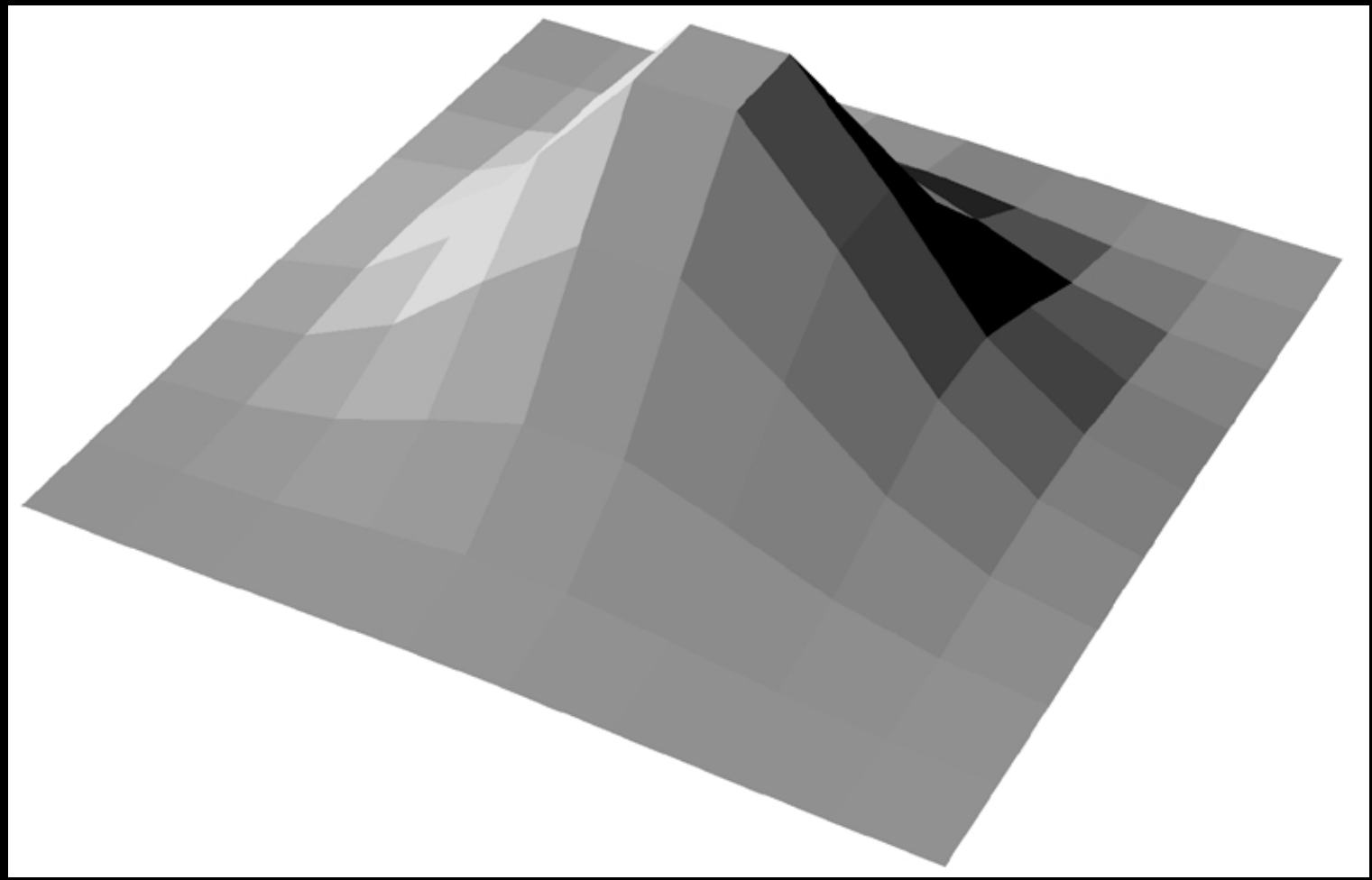


Figure 6.29 Perceived and actual intensities at an edge.



Interpolative Shading

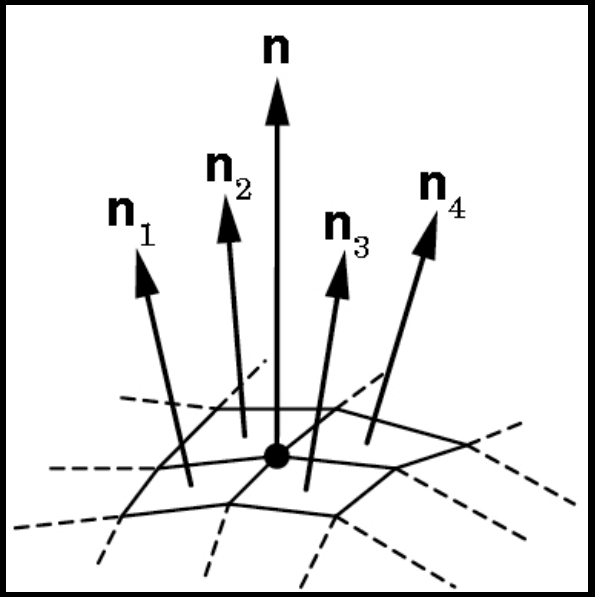
- Enable with **glShadeModel(GL_SMOOTH);**
- Calculate color at each vertex
- Interpolate color in interior
- Compute during scan conversion (rasterization)
- Much better image (see Assignment 1)
- More expensive to calculate
- Consider two types: Gouraud and Phong

Gouraud Shading

- Special case of interpolative shading
- How do we calculate vertex normals?
- Gouraud: average all adjacent face normals

$$\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{|\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4|}$$

- Requires knowledge about which faces share a vertex—adjacency info

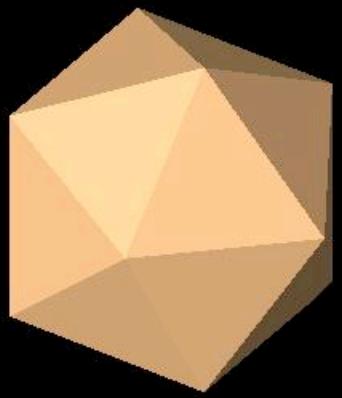


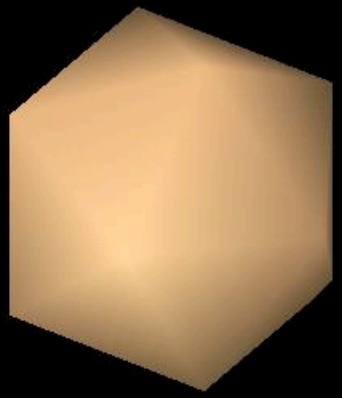
Data Structures for Gouraud Shading

- Sometimes vertex normals can be computed directly (e.g. height field with uniform mesh)
- More generally, need data structure for mesh
- Key: which polygons meet at each vertex

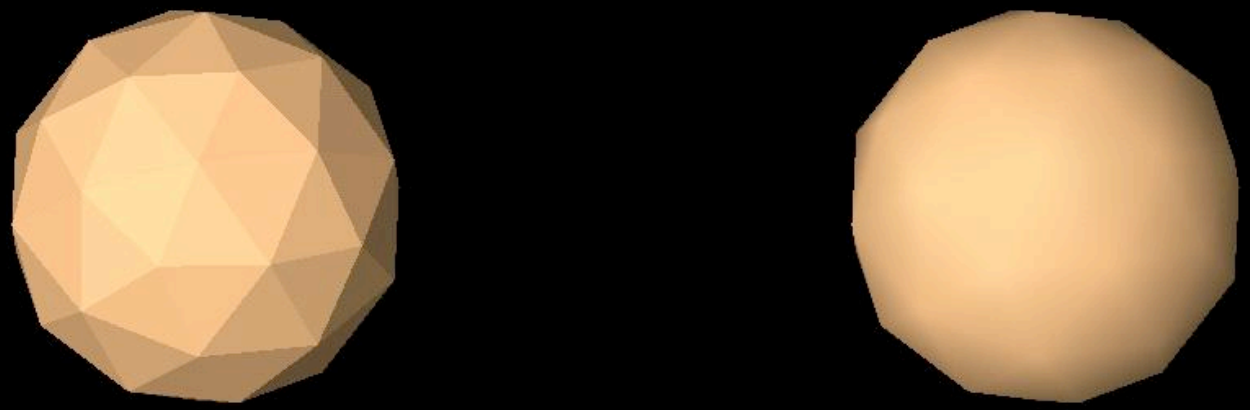
Icosahedron with Sphere Normals

- Interpolation vs flat shading effect



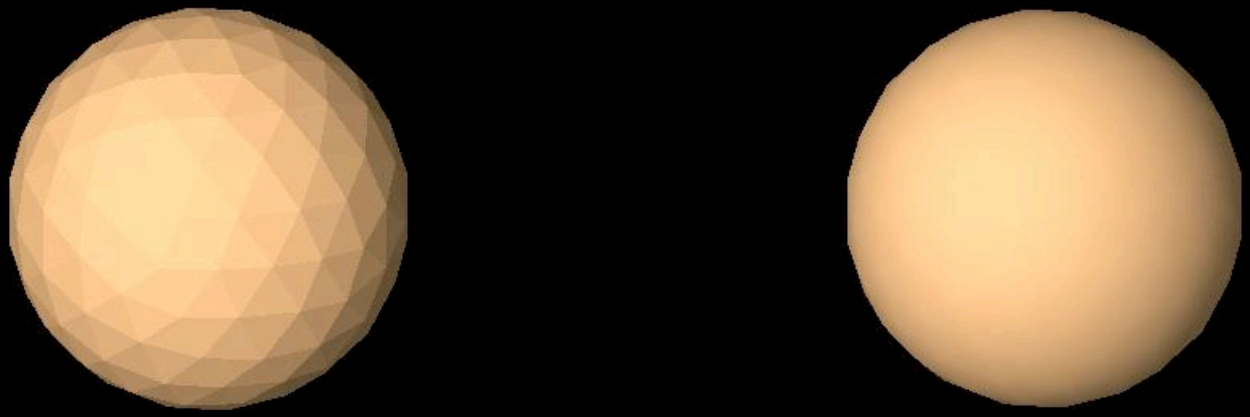


One Subdivision



Two Subdivisions

- Each time, multiply number of faces by 4



Three Subdivisions

- Reasonable approximation to sphere

Lighting in OpenGL

- Very similar to color
  - …But different

Enabling Lighting and Lights

- Lighting in general must be enabled

```
glEnable(GL_LIGHTING);
```

- Each individual light must be enabled

```
glEnable(GL_LIGHT0);
```

- OpenGL supports at least 8 light sources
  - More depending on graphics card
  - What if you need more than the card supports?

Global Ambient Light

- Set ambient intensity for entire scene

```
GLfloat al[] = {0.2, 0.2, 0.2, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, al);
```

  - The above is default

- Also: properly light backs of polygons

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDED, GL_TRUE)
```

Defining a Light Source

- Use vectors {r, g, b, a} for light properties
- Beware: light source will be transformed!

```
GLfloat light_ambient[]  = {0.2, 0.2, 0.2, 1.0};
GLfloat light_diffuse[]  = {1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_AMBIENT,  light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE,  light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Point Source vs Directional Source

- Directional light given by “position” vector

```
GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

- Point source given by “position” point

```
GLfloat light_position[] = {-1.0, 1.0, -1.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Spotlights

- Create point source as before
- Specify additional properties to create spotlight

```
GLfloat sd[] = {-1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, sd);
glLightf (GL_LIGHT0, GL_SPOT_CUTOFF,   45.0);
glLightf (GL_LIGHT0, GL_SPOT_EXPONENT,  2.0);
```

Defining Material Properties

- Material properties stay in effect (like color)
- Set both specular coefficients and shininess

```
GLfloat mat_a[]  = {0.1, 0.5, 0.8, 1.0};
GLfloat mat_d[]  = {0.1, 0.5, 0.8, 1.0};
GLfloat mat_s[]  = {1.0, 1.0, 1.0, 1.0};
GLfloat low_sh[] = {5.0};
glMaterialfv(GL_FRONT, GL_AMBIENT,   mat_a);
glMaterialfv(GL_FRONT, GL_DIFFUSE,   mat_d);
glMaterialfv(GL_FRONT, GL_SPECULAR,  mat_s);
glMaterialfv(GL_FRONT, GL_SHININESS, low_sh);
```

- Diffuse component is analogous

Defining and Maintaining Normals

- Define unit normal before each vertex

```
glNormal3f(nx, ny, nz);
glVertex3f(x, y, z);
```

- Length changes under some transformations
- Ask OpenGL to re-normalize (always works)

```
glEnable(GL_NORMALIZE);
```

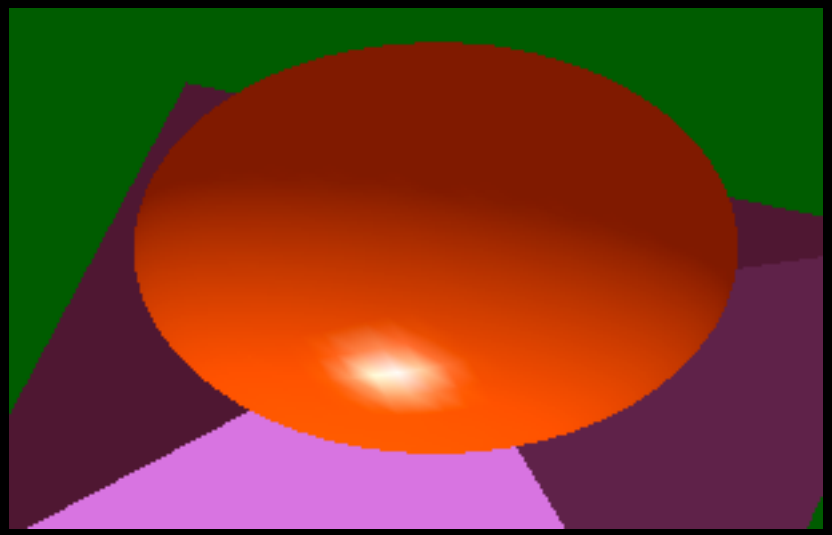
- Ask OpenGL to re-scale normal (works for **uniform** scaling, rotate, translate)

```
glEnable(GL_RESCALE_NORMAL);
```

A Demonstration

So what *doesn't* it do?

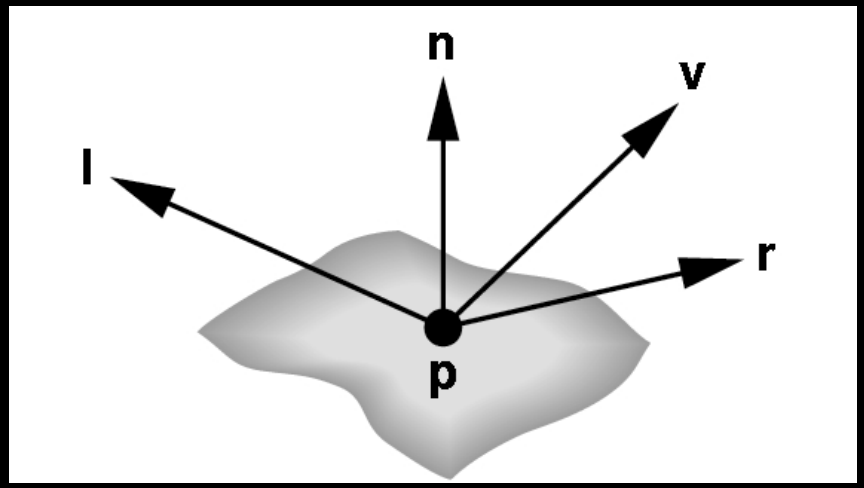
- Sphere can look a bit "off" close up
- Specular reflection not quite right
- Why? We interpolate colors linearly, but specular result is non-linear



Phong Illumination Model

- Interpolate normals instead of colors (barycentric coordinates)
- Calculate color for arbitrary point on surface
- Basic inputs are material properties and l, n, v:

l = vector to light source
n = surface normal
v = vector to viewer
r = reflection of l at p
(determined by l and n)



Summary of Phong Model

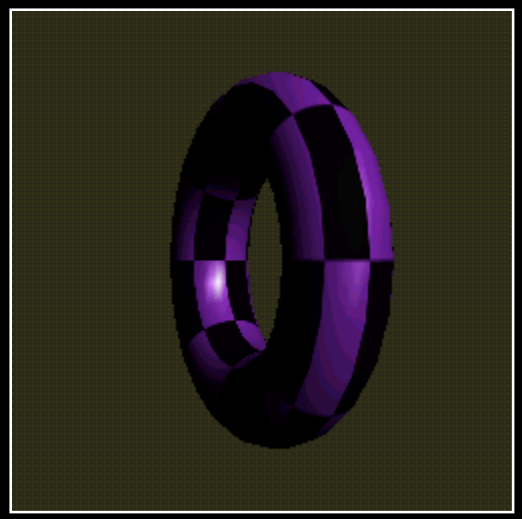
- Light components for each color:
  - Ambient (L_a), diffuse (L_d), specular (L_s)
- Material coefficients for each color:
  - Ambient (k_a), diffuse (k_d), specular (k_s)
- Distance q for surface point from light source

$$I = \frac{1}{a + bq + cq^2}(k_d L_d(\mathbf{l}\cdot\mathbf{n}) + k_s L_s(\mathbf{r}\cdot\mathbf{v})^{\alpha}) + k_a L_a$$

l = vector from light
n = surface normal
r = l reflected about n
v = vector to viewer

Phong Shading Results

Michael Gold, Nvidia



Phong Lighting
Gouraud Shading

Why not always use Phong?

Phong Lighting,
Phong Shading

Raytracing Example

Martin Moeck,
Siemens Lighting

Radiosity Example

Restaurant Interior. Guillermo Leal, Evolucion Visual

Non-photorealistic rendering

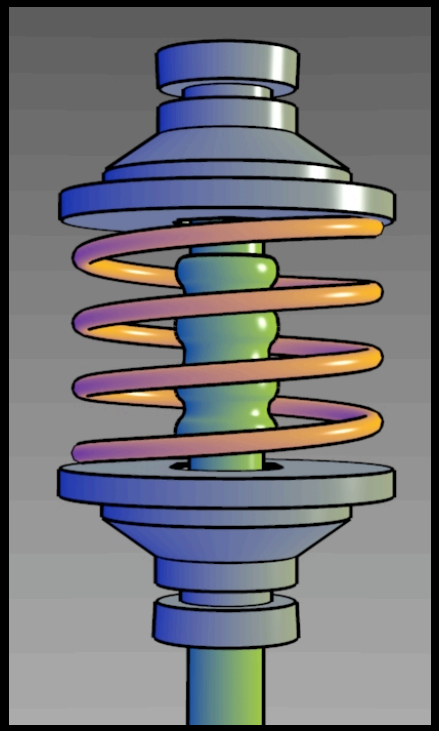
- Human brain is an amazing pattern recognition system
- Throws out most detail
- Basic idea: Simplify a model to convey specific information

NPR techniques

- Sihlouette generation
- Crease rendering
- Cool-to-warm shading

Silhouettes

- Generate an outline of the object
- Where is “edge” of object relative to viewer?

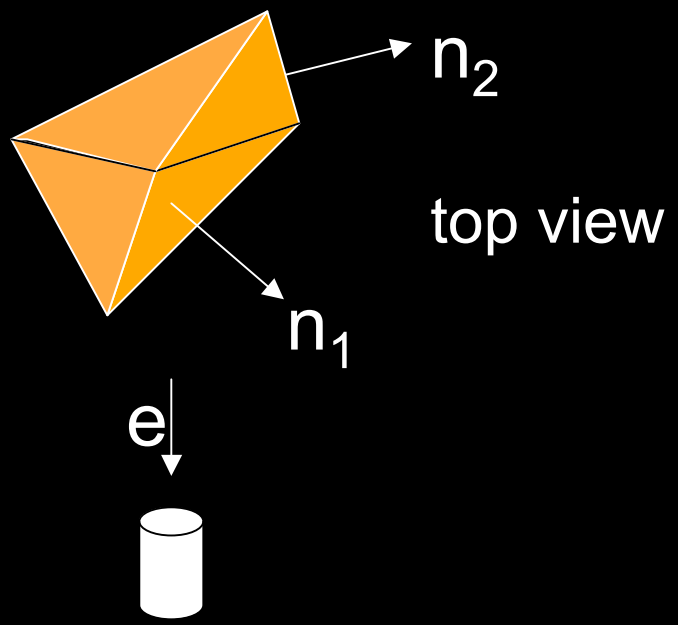


Amy Gooch - Bruce Gooch - Peter Shirley - Elaine Cohen

SIGGRAPH ‘98

Silhouettes

- Consider adjacent polygons, p_1 and p_2, with normals n_1 and n_2
- Compute $(e^* n_1)$ $(e^* n_2)$
- If <= 0, one poly is toward viewer, the other is away
  - So, draw silhouette



Corners and creases

- Sharp changes in shape should be highlighted
  - But, we can't just highlight every shared edge (wireframe mode)
- Compare normals of adjacent edges
- If $n_1 * n_2 <$ threshold, draw edge

Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, Anthony Santella,

ACM Transactions on Graphics, July 2003

Cool-to-warm shading

- Simple way to highlight surface curvature
- Rather than using shadow and non-shadow, shade between two contrasting colors (red-blue)
- $k_w = (1+n*l) / 2$
- $C=k_w\ c_w + (1-k_w)c_c$

Related technique: Cel shading

- Similar to basic non-photorealistic technique
- Only allow shading colors to be drawn from a small palette
- “Cartoonish” models greatly help

(tmZs)

Questions?