Lecture 12

Today

- Texturing
  - UV space
  - UV coordinates in shader
- Scanline Algorithm
- Phong Shading
  - Flat/Bump

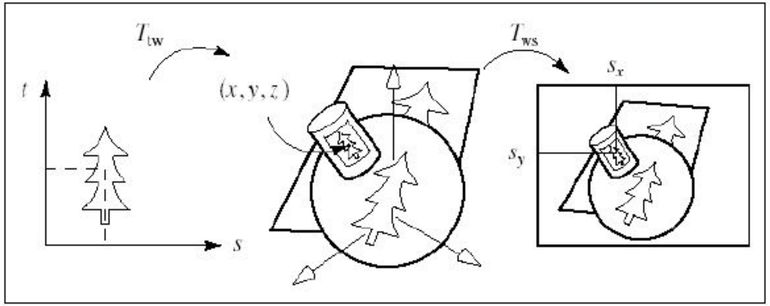
Texturing

How do texture coordinates get used?

Coordinate Systems Involved

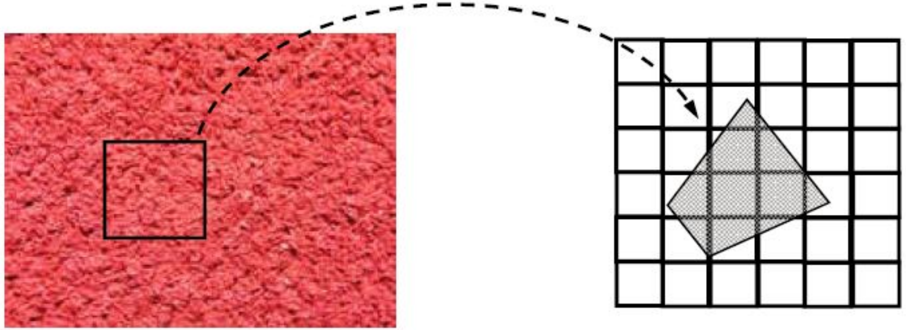
User Defined　　Viewing+Projection

FIGURE 8.35 Drawing texture on several objects of different shape.



$(s_x, s_y) = T_{ws}(T_{tw}(s,t))$

Texture to Screen

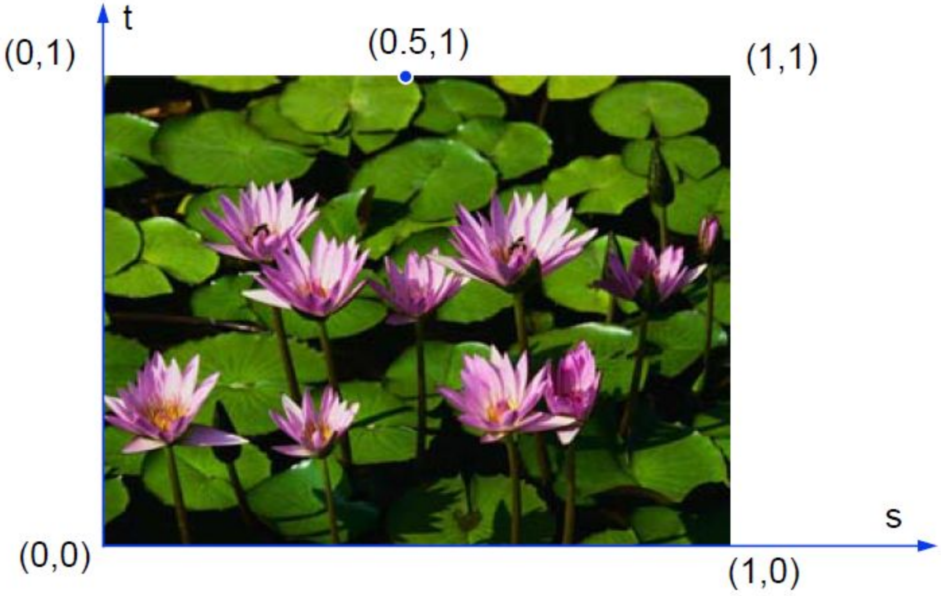


$(s_x, s_y) = T_{ws}(T_{tw}(s,t))$

We would have to calculate pixel coverages

Textures are Images

They are always assigned the shown parametric coordinates (s,t)

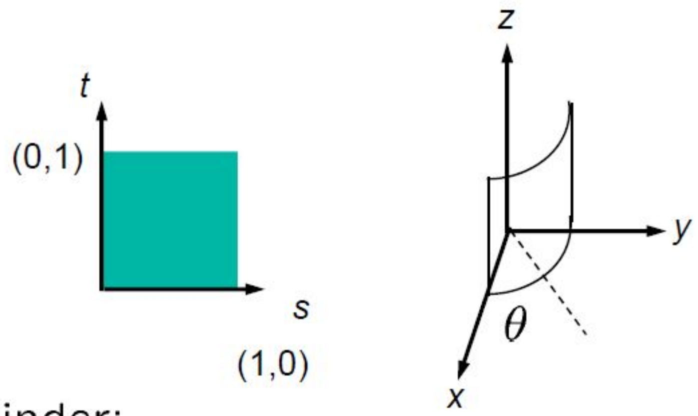


From Texture to World (Object)

To apply a texture to an object we have to find a correspondance between (s,t) and and some object coordinate system

- Mapping via a parametric representation of the object space
- Manually

Example 3: Square Texture to Cylinder



Parametric form of cylinder:

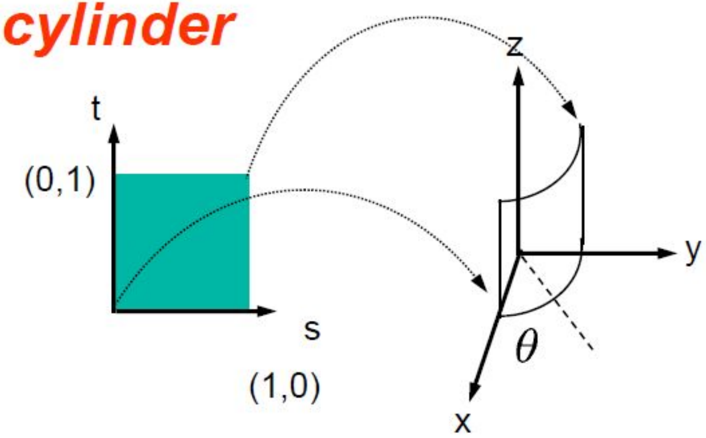
$$x = r\cos\theta,\;\; y = r\sin\theta,\;\; z$$

Surface parameters: $u = \theta, \quad v = z$

with $0 \leq u \leq \pi/2$, and $0 \leq v \leq 1$

Example 3: Square Texture to Cylinder

Square texture to cylinder



We pick the following linear transformation that maps $(s,t) = (0,0)$ to $(u,v) = (0,0)$ and $(s,t) = (1,1)$ to $(u,v) = (\frac{\pi}{2}, 1)$:

$$u = s\frac{\pi}{2}, \quad v = t$$

From Texture to World (Object)

To apply a texture to an object we have to find a correspondance between (s,t) and and some object coordinate system

- Mapping via a parametric representation of the object space
- Manually

Mapping the Texture to an Object Parametric Representation

Linear transformation

From texture space (s,t) to object space (u,v)

$u = u(s,t) = a_u\, s + b_u\, t + c_u$

$v = v(s,t) = a_v\, s + b_v\, t + c_v$

s in [0,1]

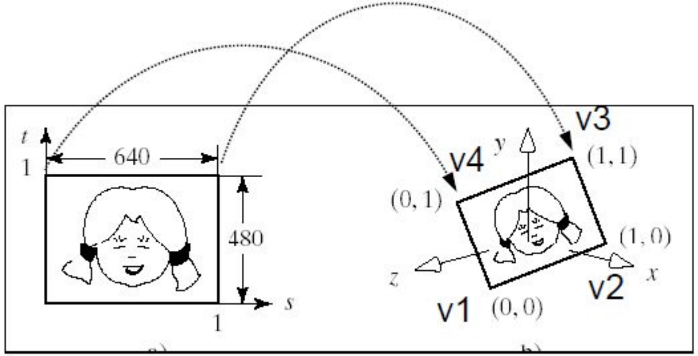
t in [0,1]

Example: Image to a Quadrilateral

Simply

$u = u(s,t) = s$

$v = v(s,t) = t$



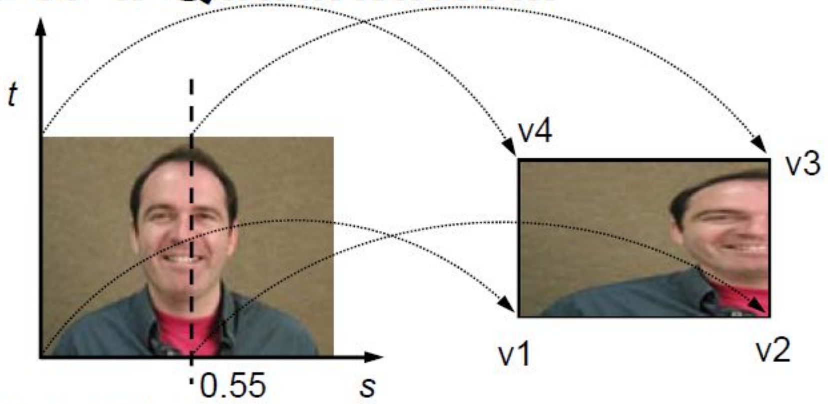

```
glTexCoord2f(0,0) ;  glVertex3dv(v1) ;
glTexCoord2f(1,0) ;  glVertex3dv(v2) ;
glTexCoord2f(1,1) ;  glVertex3dv(v3) ;
glTexCoord2f(0,1) ;  glVertex3dv(v4) ;
```

Example 2: Piece of Image to a Quadrilateral

Use only left part

$u = u(s,t) = 0.55s$

$v = v(s,t) = t$




```
glTexCoord2f(0,0) ;  glVertex3dv(v1) ;
glTexCoord2f(0.55,0) ;  glVertex3dv(v2) ;
glTexCoord2f(0.55,1) ;  glVertex3dv(v3) ;
glTexCoord2f(0,1) ;  glVertex3dv(v4) ;
```

Packing textures for efficiency

Scanline Algorithm

Generating interpolated values per pixel

How Does this Work With the Graphics Pipeline?

Only polygons

Only vertices go down the graphics pipeline

Interior points?

- *Calculate texture coordinates by interpolation along scanlines*

Rendering the Texture

Scanline in screen space

- Generating *s*,*t* coordinates for each pixel

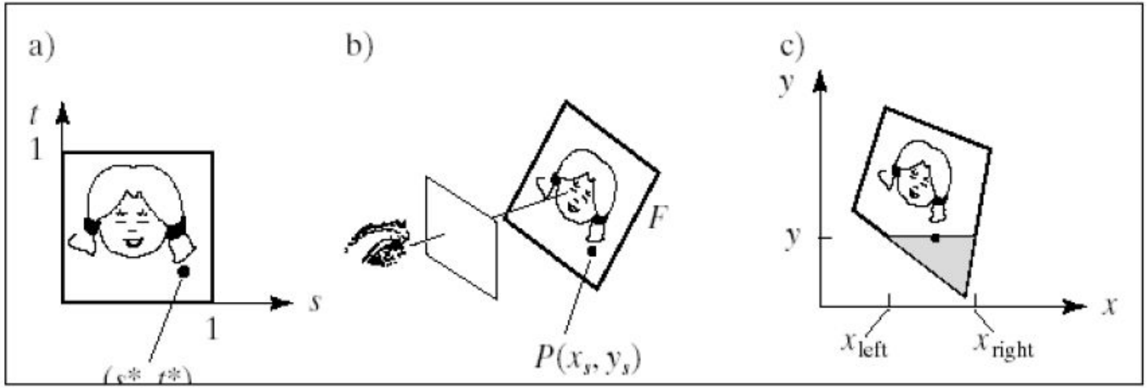


FIGURE 8.39 Rendering a face in a camera snapshot.

Step One: Rasterization

- The GPU must generate an outline of your shape in pixels.
- Loop through each edge in your shape
- Run an algorithm that returns a pixel set that overlaps a line segment.
  - DDA algorithm, Bresenham's algorithm

Step Two: Scanlines

- For each triangle,
  - For all rows,
    - Each side of the triangle overlapped with a first column and a last column.
    - (These came from the outline we calculated already from our line drawing algorithms)
    - Loop horizontally from the start pixel to end pixel.
      - This is one “Scanline”
- This visits all pixels inside each triangle.
  - Color each one by sampling data using barycentric coords.

Interpolation of Texture Coordinates

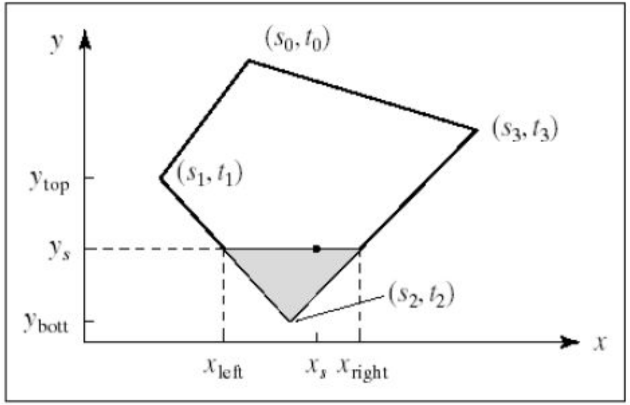
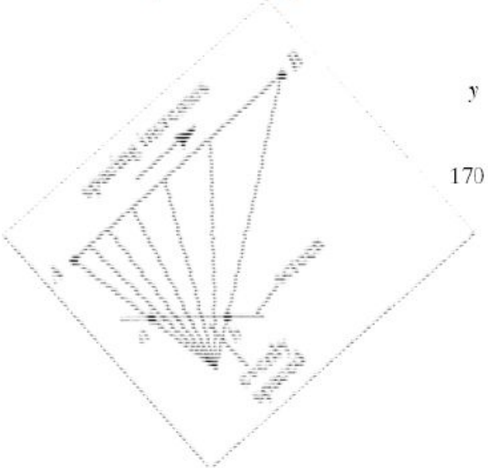


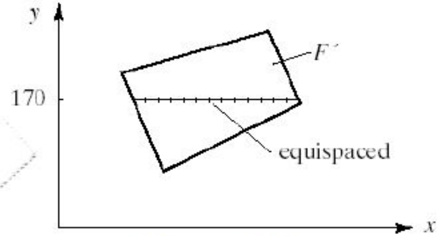
FIGURE 8.40 Incremental calculation of texture coordinates.

Problem

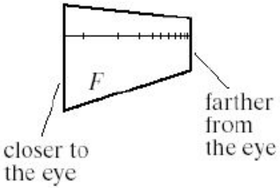
Perspective foreshortening

- Scan conversion takes equal steps along scanline in screen space
- Equal steps in screen space are **not** equal steps in world space



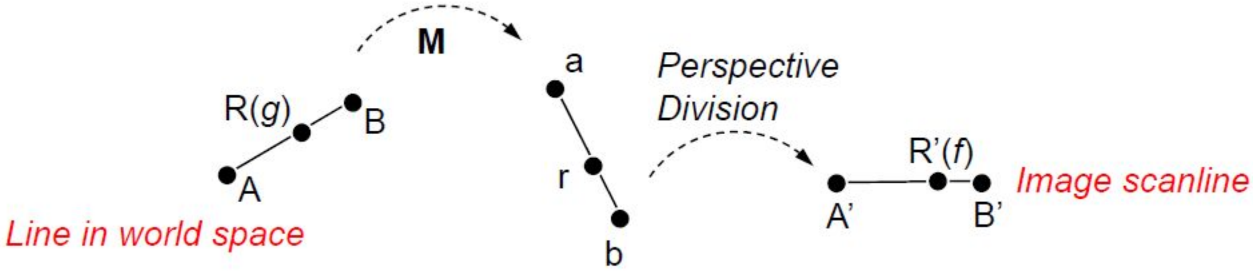


Screen Space



3D World Space

Reminder: In-Between Points



$$
\left.\begin{aligned}
R'_1(f) &= \frac{lerp(a_1, b_1, g)}{lerp(a_4, b_4, g)} \\
R'_1(f) &= lerp\left(\frac{a_1}{a_4}, \frac{b_1}{b_4}, f\right)
\end{aligned}\right\} \Rightarrow g = \frac{f}{lerp(\frac{b_4}{a_4}, 1, f)}
$$

substituting this in $R(g) = (1-g)A + gB$ yields

$$
R_1 = \frac{lerp(\frac{A_1}{a_4}, \frac{B_1}{b_4}, f)}{lerp(\frac{1}{a_4}, \frac{1}{b_4}, f)}
$$

and similarly for R_2 and R_3

Interpolating Information (Incrementally)

Color, Normal, Texture coordinates

Right edge (1,2):

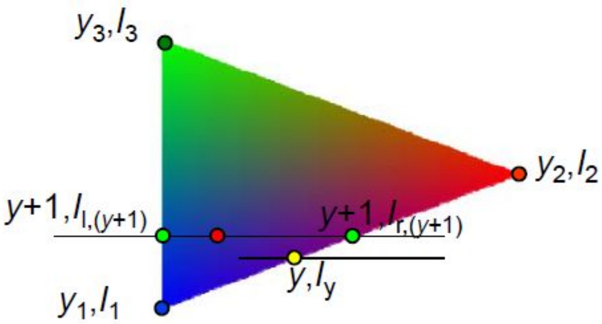
$$\frac{I_{r,(y+1)} - I_{r,y}}{(y+1) - y} = \frac{I_1 - I_2}{y_1 - y_2} \Rightarrow I_{r,(y+1)} = I_{r,y} + \frac{I_1 - I_2}{y_1 - y_2}$$

Left Edge (1,3):

$$\frac{I_{l,(y+1)} - I_{l,y}}{(y+1) - y} = \frac{I_1 - I_3}{y_1 - y_3} \Rightarrow I_{l,(y+1)} = I_{l,y} + \frac{I_1 - I_3}{y_1 - y_3}$$

Along scanline:

$$\frac{I_{(x+1)} - I_x}{(x+1) - x} = \frac{I_r - I_l}{x_r - x_l} \Rightarrow I_{r,(y+1)} = I_{r,y} + \frac{I_r - I_l}{x_r - x_l}$$



Interpolating Information (Incrementally)

Color, Normal, Texture coordinates

Right edge (1,2):

$$\frac{I_{r,(y+1)} - I_{r,y}}{(y+1) - y} = \frac{I_1 - I_2}{y_1 - y_2} \Rightarrow I_{r,(y+1)} = I_{r,y} + \frac{I_1 - I_2}{y_1 - y_2}$$

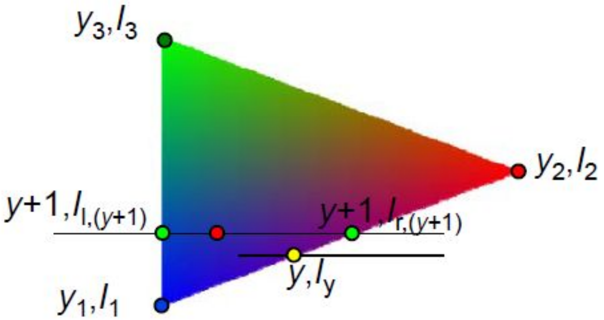
Left Edge (1,3):

$$\frac{I_{l,(y+1)} - I_{l,y}}{(y+1) - y} = \frac{I_1 - I_3}{y_1 - y_3} \Rightarrow I_{l,(y+1)} = I_{l,y} + \frac{I_1 - I_3}{y_1 - y_3}$$

Along scanline:

$$\frac{I_{(x+1)} - I_x}{(x+1) - x} = \frac{I_r - I_l}{x_r - x_l} \Rightarrow I_{r,(y+1)} = I_{r,y} + \frac{I_r - I_l}{x_r - x_l}$$

Constant along the line



Phong Shading and Lighting

Lighting formulas

Light equation

- The two shaders must put together some equation to come up with pixel brightness
- We can find out what sort of equation we can make by looking at what the inputs of the shader are - what information is available?

Info that a light equation could use.
What contributes most to final brightness?

- Per element (triangle):
  - Per vertex in each element:
    - Positions
    - Normals (perpendicular vectors for all points)
    - Colors if you want some per vertex
    - Coords in texture space
- Per whole draw call:
  - Matrices
  - Texture image to lookup, if any
  - Flags - color this shape solid? etc.
  - Light position (and specify if point or vector)
  - Material properties of shape - how chalky/shiny its interaction with light source is

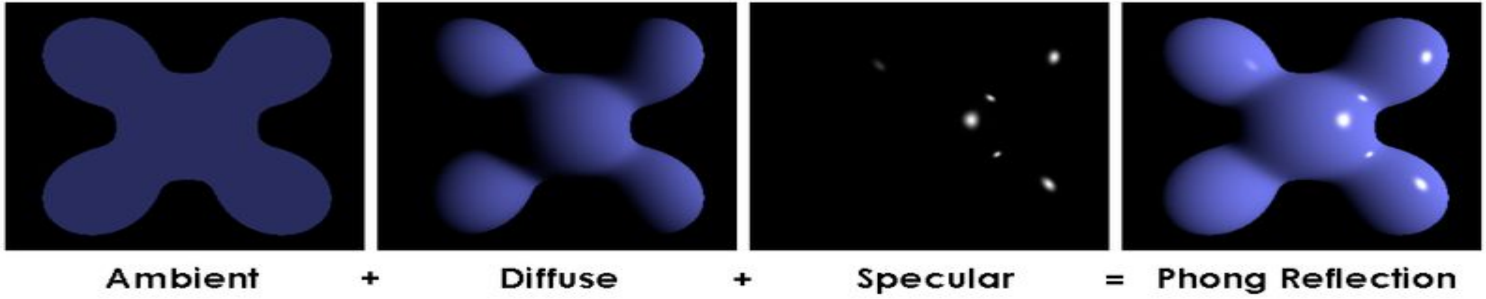
Info that a light equation could use.
What contributes most to final brightness?

- Normals (perpendicular vectors for all points)

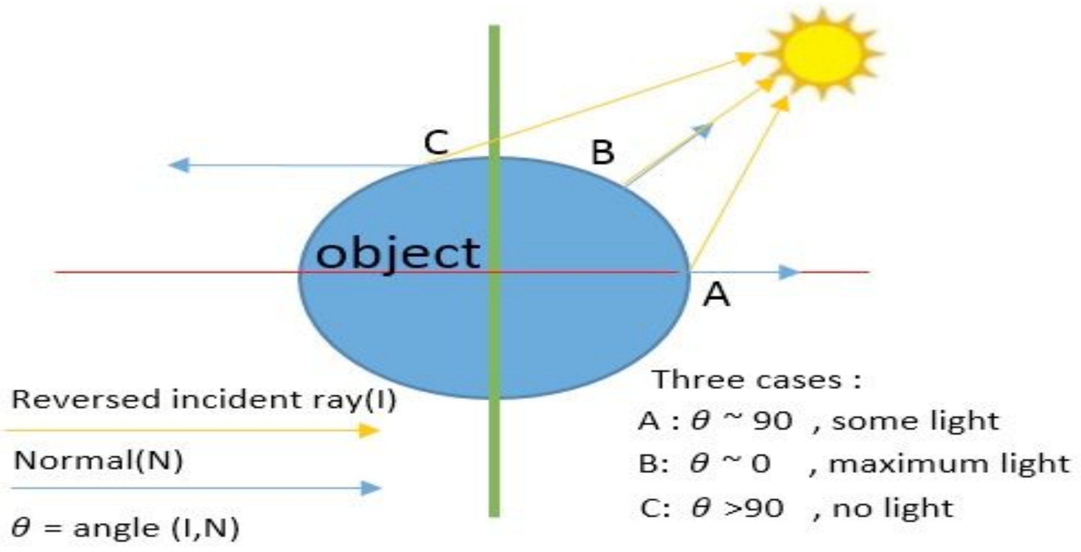
Light equation starts with just all this info

- Normal is the most important input to light equation
- Intermediate calculations: Compute eye, L
- Our equation choice: Phong model


Components of light

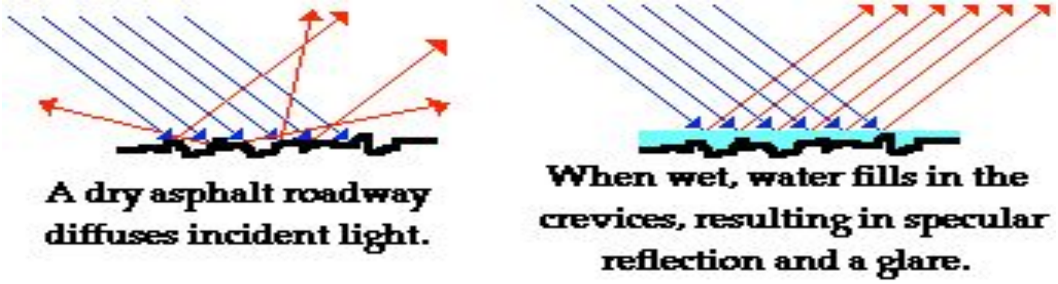


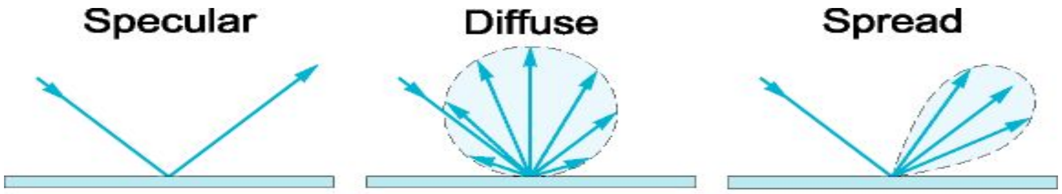
Lambert’s law



“The amount of reflected light is proportional with the cosine (dot product) of the angle between the normal and incident vector”

Combining components of light

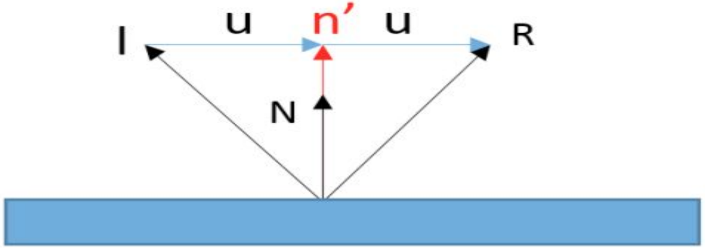




Specular, diffuse, and spread reflection from a surface.

Light equation

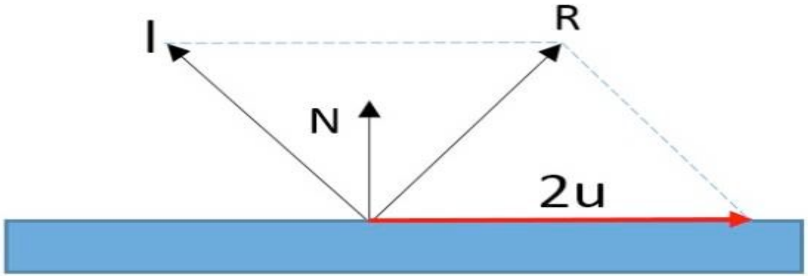
- Calculating R, the (non-physical, made-up) reflection of the point light source



The $\vec{n'}$ is the projection of $\vec{I}$ on $\vec{N}$

$$\vec{n'} = (\vec{N} \cdot \vec{I})\, \vec{N}, \text{ with } \|\vec{N}\|^2 = 1$$

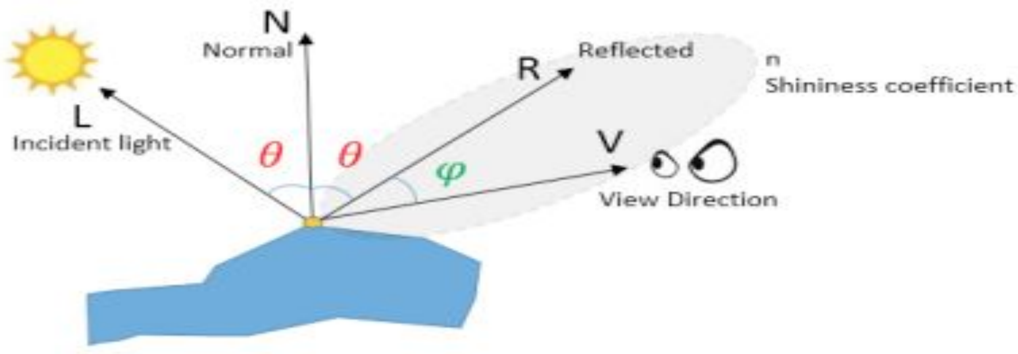
$$\vec{u} = \vec{n'} - \vec{I}$$



$$\vec{R} = \vec{I} + 2\vec{u} = \vec{I} + 2(\vec{n'} - \vec{I})$$

$$\vec{R} = 2(\vec{N} \cdot \vec{I})\, \vec{N} - \vec{I}$$

Light equation

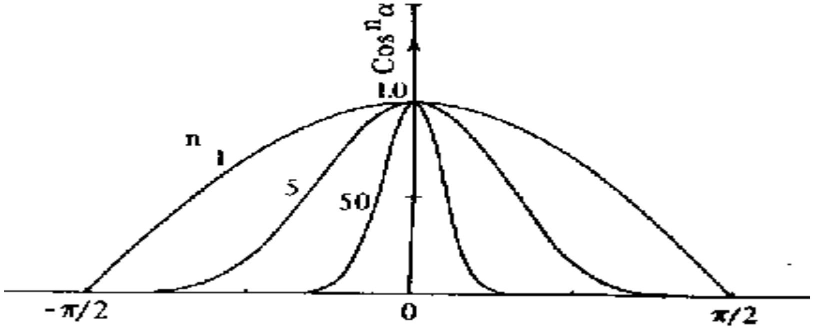


I = emissive + ambient + diffuse + specular

$$
\begin{aligned}
\text{emissive} &= k_e \\
\text{ambient} &= k_a * ambientColor \\
\text{diffuse} &= k_d * lightColor * \cos(\theta) \\
&= k_d * lightColor * \max(0, N \cdot L) \\
\text{specular} &= k_s * lightColor * \cos(\varphi)^n \\
&= k_s * lightColor * \max(0, R \cdot V)^{\,n}
\end{aligned}
$$

Specular term - Smoothness exponent effect

- Exponentiating a function that has values < 1 draws those values closer to zero
- Higher exponent = smaller region where point light's reflection is considered "aligned" with the viewer.
- Smaller shiny spot



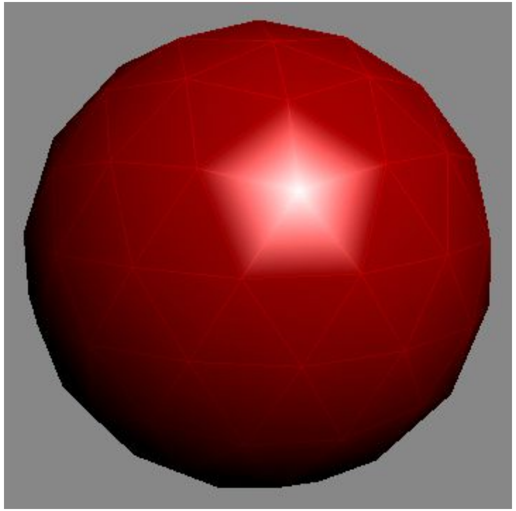
Material properties - coefficients

Name	Ambient			Diffuse			Specular			Shininess
emerald	0.0215	0.1745	0.0215	0.07568	0.61424	0.07568	0.633	0.727811	0.633	0.6
jade	0.135	0.2225	0.1575	0.54	0.89	0.63	0.316228	0.316228	0.316228	0.1
obsidian	0.05375	0.05	0.06625	0.18275	0.17	0.22525	0.332741	0.328634	0.346435	0.3
pearl	0.25	0.20725	0.20725	1	0.829	0.829	0.296648	0.296648	0.296648	0.088
ruby	0.1745	0.01175	0.01175	0.61424	0.04136	0.04136	0.727811	0.626959	0.626959	0.6
turquoise	0.1	0.18725	0.1745	0.396	0.74151	0.69102	0.297254	0.30829	0.306678	0.1
brass	0.329412	0.223529	0.027451	0.780392	0.568627	0.113725	0.992157	0.941176	0.807843	0.21794872
bronze	0.2125	0.1275	0.054	0.714	0.4284	0.18144	0.393548	0.271906	0.166721	0.2
chrome	0.25	0.25	0.25	0.4	0.4	0.4	0.774597	0.774597	0.774597	0.6
copper	0.19125	0.0735	0.0225	0.7038	0.27048	0.0828	0.256777	0.137622	0.086014	0.1
gold	0.24725	0.1995	0.0745	0.75164	0.60648	0.22648	0.628281	0.555802	0.366065	0.4
silver	0.19225	0.19225	0.19225	0.50754	0.50754	0.50754	0.508273	0.508273	0.508273	0.4
black plastic	0.0	0.0	0.0	0.01	0.01	0.01	0.50	0.50	0.50	.25
cyan plastic	0.0	0.1	0.06	0.0	0.50980392	0.50980392	0.50196078	0.50196078	0.50196078	.25
green plastic	0.0	0.0	0.0	0.1	0.35	0.1	0.45	0.55	0.45	.25
red plastic	0.0	0.0	0.0	0.5	0.0	0.0	0.7	0.6	0.6	.25
white plastic	0.0	0.0	0.0	0.55	0.55	0.55	0.70	0.70	0.70	.25
yellow plastic	0.0	0.0	0.0	0.5	0.5	0.0	0.60	0.60	0.50	.25
black rubber	0.02	0.02	0.02	0.01	0.01	0.01	0.4	0.4	0.4	.078125
cyan rubber	0.0	0.05	0.05	0.4	0.5	0.5	0.04	0.7	0.7	.078125
green rubber	0.0	0.05	0.0	0.4	0.5	0.4	0.04	0.7	0.04	.078125
red rubber	0.05	0.0	0.0	0.5	0.4	0.4	0.7	0.04	0.04	.078125
white rubber	0.05	0.05	0.05	0.5	0.5	0.5	0.7	0.7	0.7	.078125
yellow rubber	0.05	0.05	0.0	0.5	0.5	0.4	0.7	0.7	0.04	.078125

Multiply the shininess by 128!

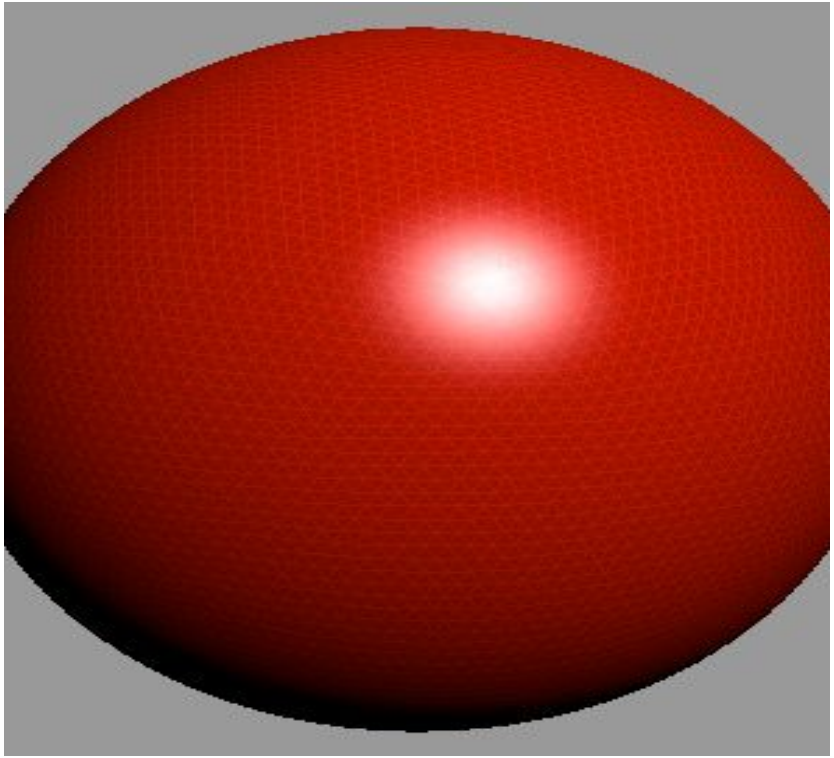
Gouraud shading

- It's the simplest method. Just assign brightnesses per vertex and interpolate.
- The specular highlight performs poorly at edges.



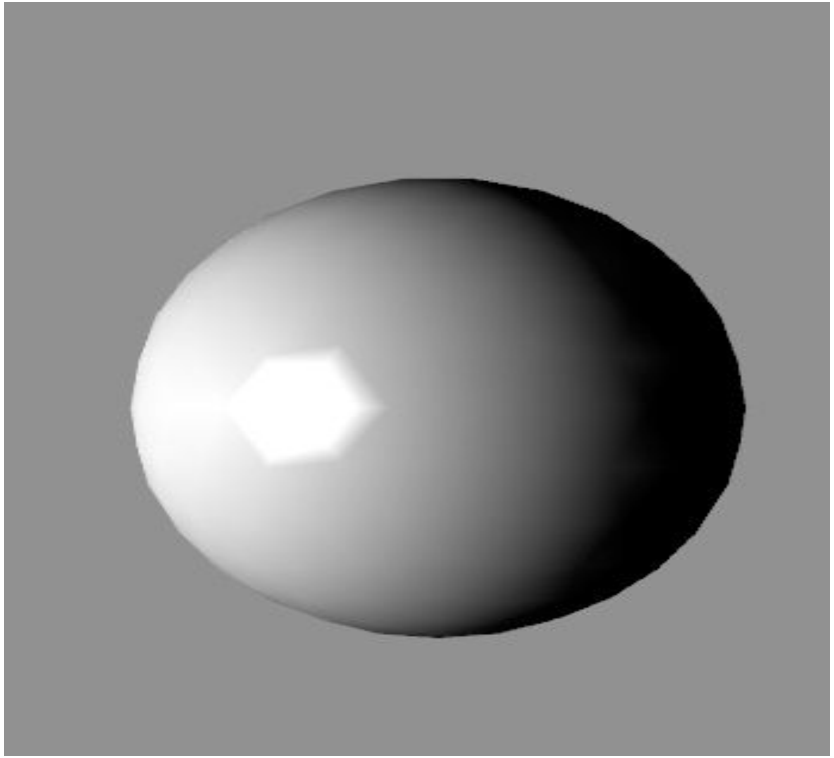
Gouraud shading

- Only solution - make edges matter less by increasing polygon count.



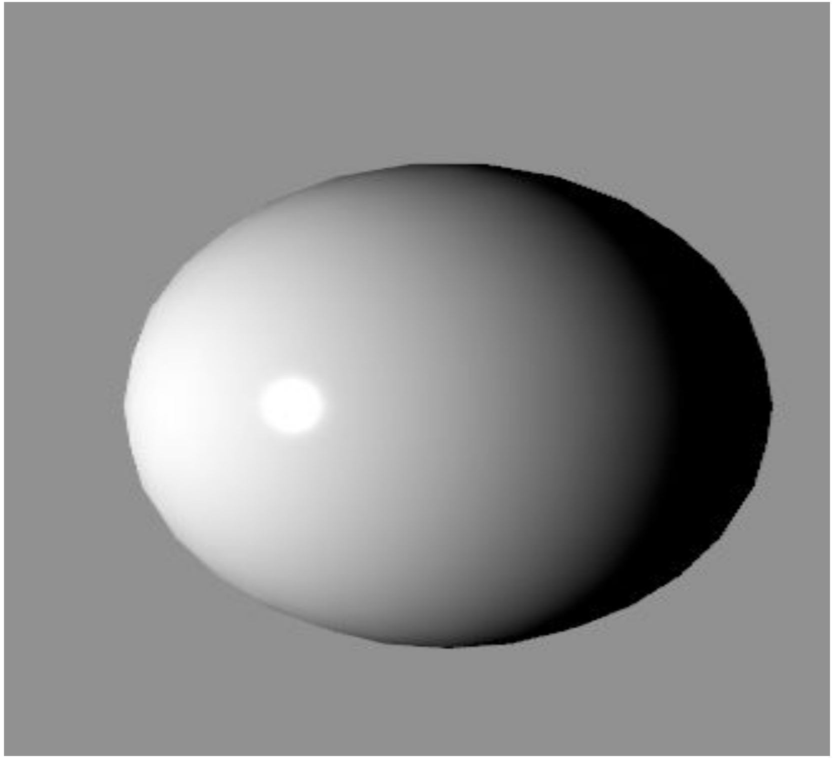
Light equation

- Gouraud shading
  - Diffuse and specular components calculated at every point and added together
  - Linearly interpolate the brightnesses



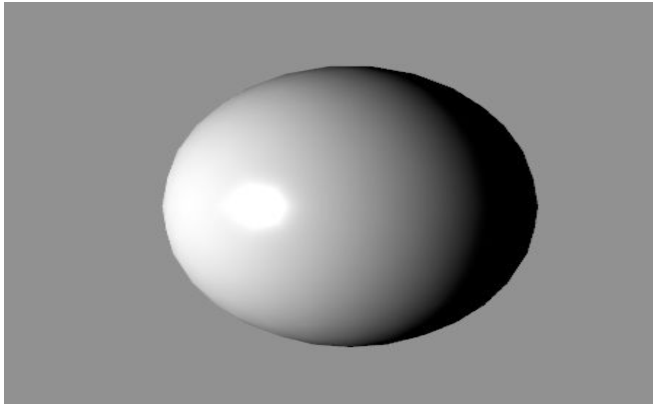
Light equation

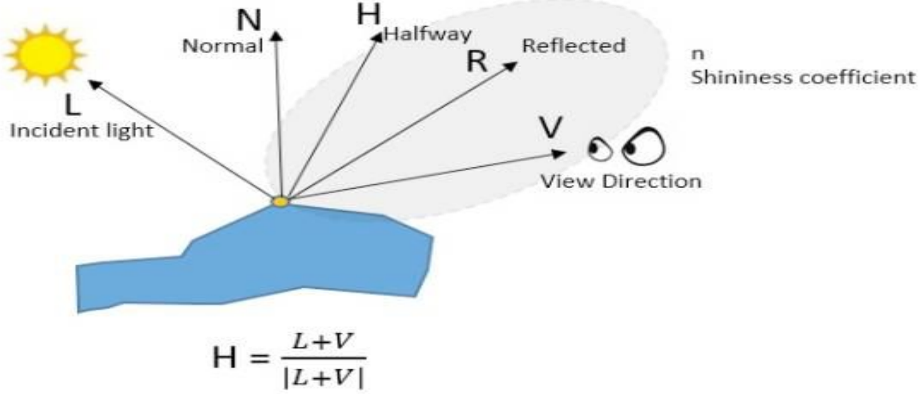
- Phong shading
  - Linearly interpolate the <u>normals</u> across each triangle
  - Only when you get to an actual pixel do you calculate the specular and diffuse brightnesses
  - At <u>every pixel</u>, a much finer scale than Gouraud



Phong-Blinn

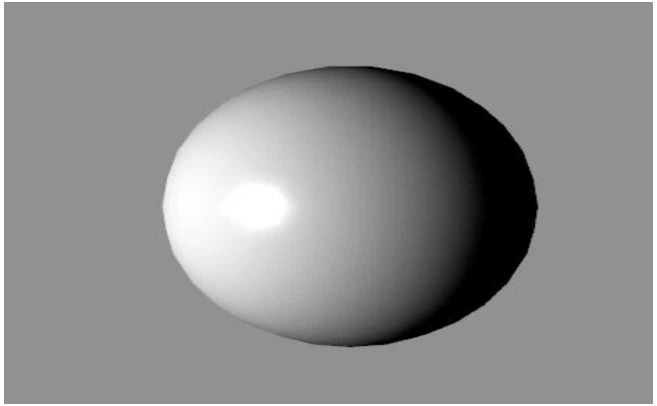
- Combine V and L, the two constants in the scene, into one vector
- Given H = halfway between V and L, Use (H dot N) instead of (R dot V)

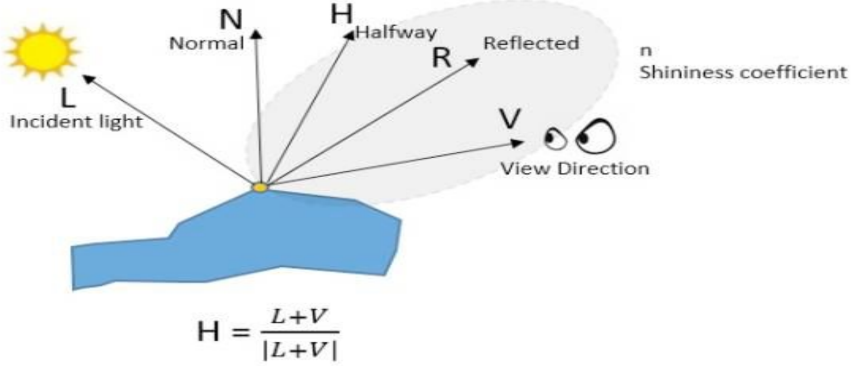




Phong-Blinn

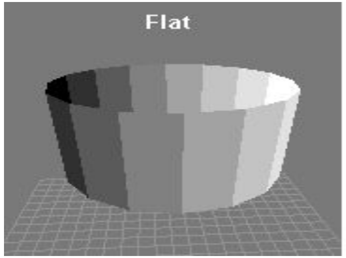
- If directional light, you can compute H once per frame per light source and it's the same everywhere in the scene - no dependence on normal, just viewer and light
- Re-use it instead of re-calcuating in shader - shader only has to dot H with each N - Cheaper.
- Also behaves better at glancing angles

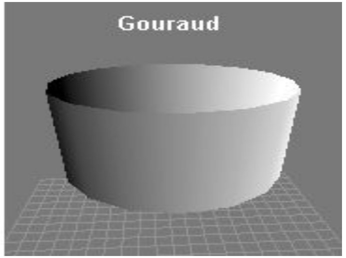




Light equation

- How many times to do the lighting equation?
  - Once per triangle - flat
  - Once per point - gouraud
  - Once per pixel - smooth / "phong shading"





Flat vs Smooth Shading

Flat vs Smooth Shading

- Smooth Shading:
  - All three normals are not the same.
    - Allows interpolation of infinity different normals as you move from one extreme point of the triangle to the other
    - This creates a different lighting formula estimate and different color at each pixel
    - Interpolation done in fragment shader (per pixel in triangle), and fed into the lighting formula

Flat vs Smooth Shading

- Flat shading
  - Nothing to interpolate. All three extreme points hold the same value. That value gets spread across whole triangle, giving the same color result everywhere.
  - The GPU still most likely does all the same interpolation math
    - (Flat shading doesn’t actually speed it up, but just produces repetitive color answers across the triangle).

Bump Mapping

Bump Mapping

Bump Mapping

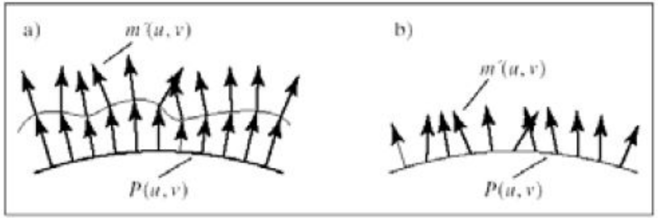


FIGURE 8.50 On the nature of bump mapping.

$P'(u,v) = P(u,v) + \text{texture}(u,v)\, m(u,v)$

Approximation by Blinn:

$m'(u,v) = m(u,v) + [(m\ P_v)\ \text{texture}_u - (m\ P_u)\ \text{texture}_v]$

All functions evaluated at (u,v)