

Computer Graphics (CS 543)

Lecture 9 (Part 2): Lighting, Shading and Materials (Part 3)

Prof Emmanuel Agu

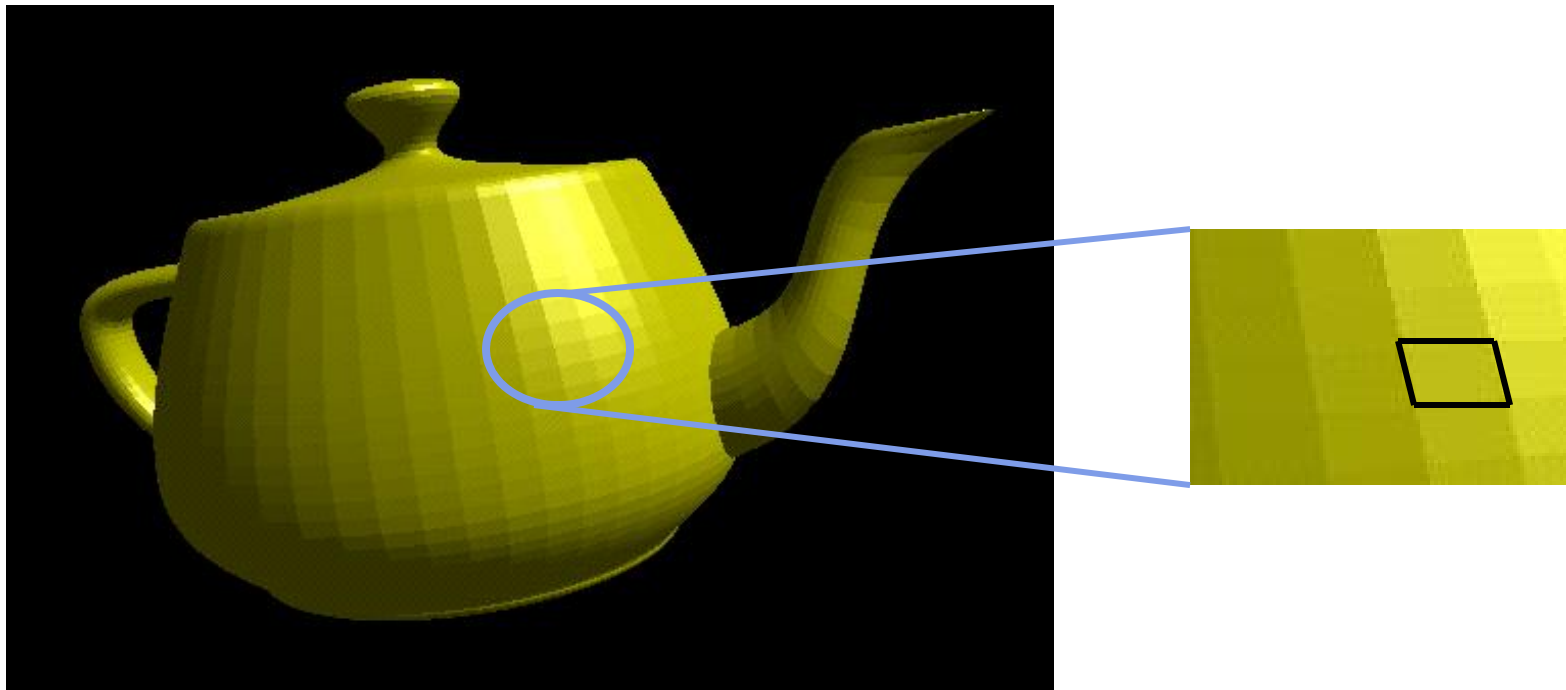
*Computer Science Dept.
Worcester Polytechnic Institute (WPI)*





Recall: Flat Shading

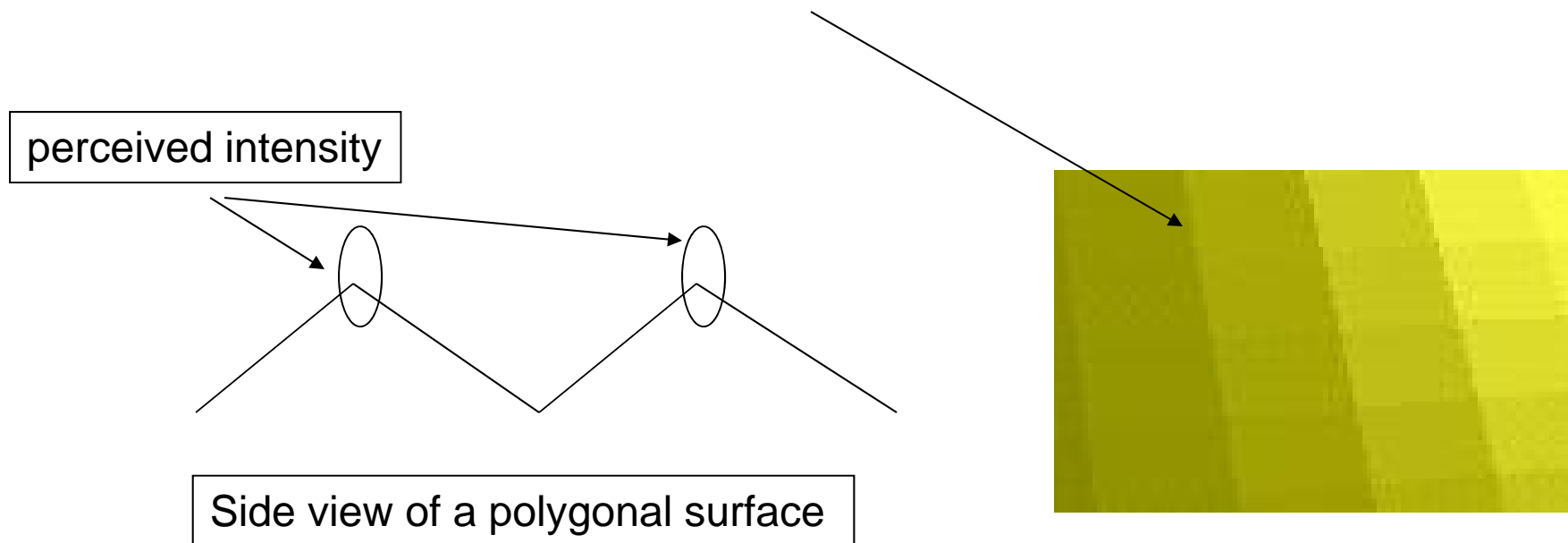
- compute lighting once for each face, assign color to whole face





Recall: Mach Band Effect

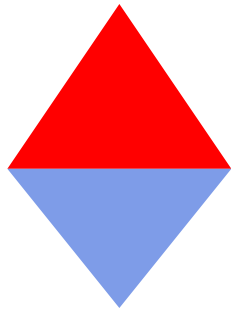
- Flat shading suffers from “mach band effect”
- Mach band effect – human eyes accentuate the discontinuity at the boundary



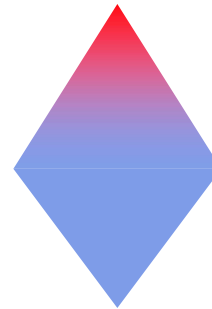
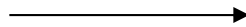


Recall: Smooth shading

- Fix mach band effect – remove edge discontinuity
- Compute lighting for more points on each face
- 2 popular methods:
 - Gouraud shading
 - Phong shading



Flat shading

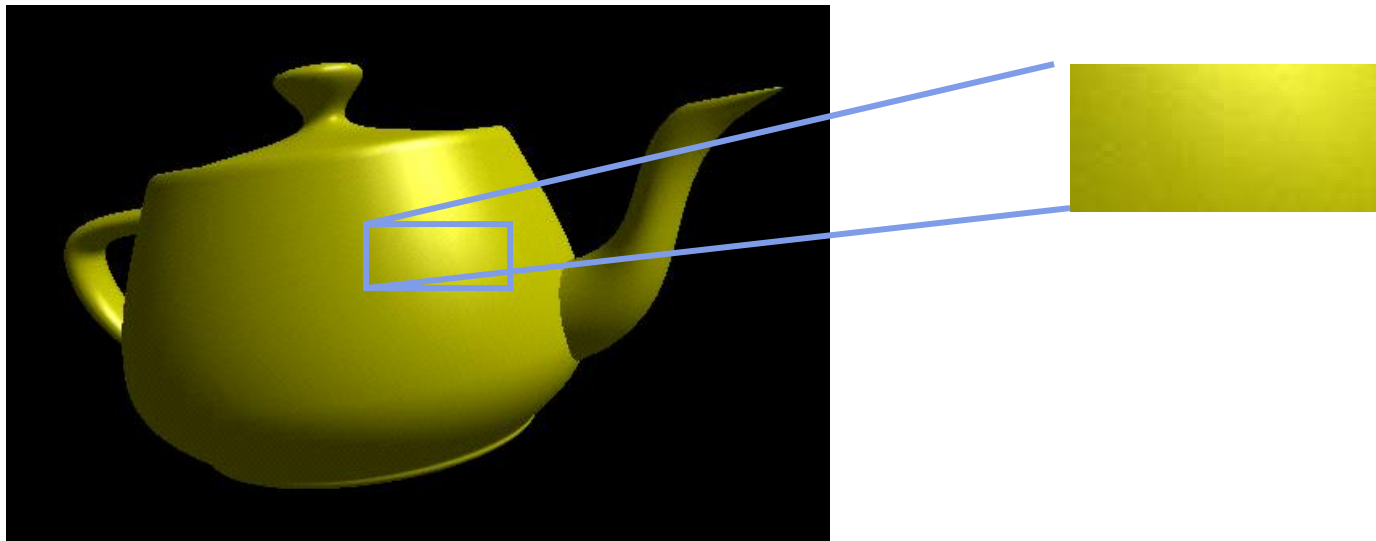


Smooth shading



Recall: Gouraud Shading

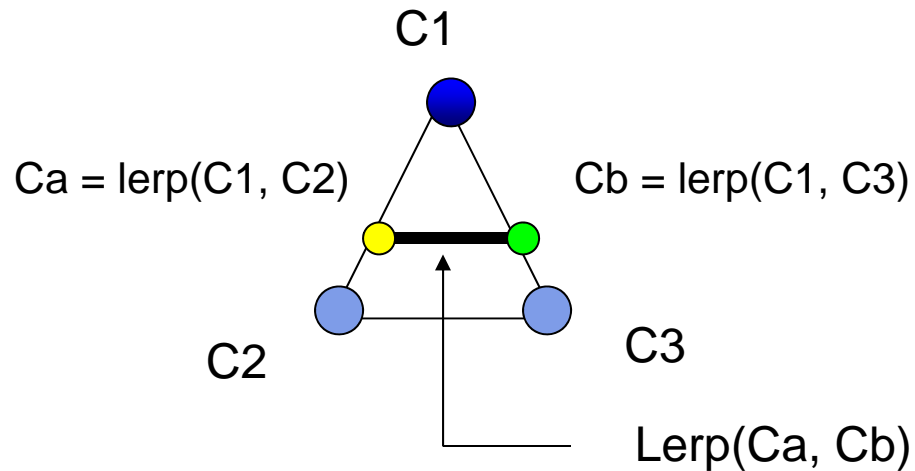
- Lighting calculated for each polygon vertex
- Colors are interpolated for interior pixels
- Interpolation? Assume linear change from one vertex color to another



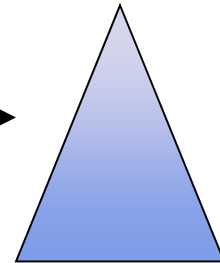
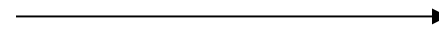


Gouraud Shading

- Compute vertex color in vertex shader
- Shade interior pixels: vertex color **interpolation**



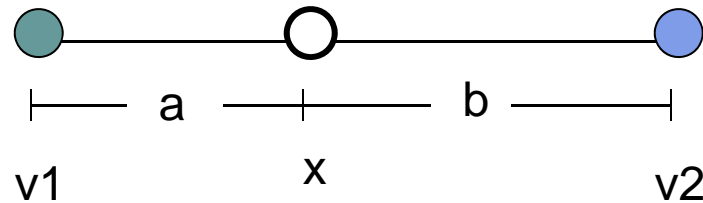
for all scanlines



* lerp: linear interpolation

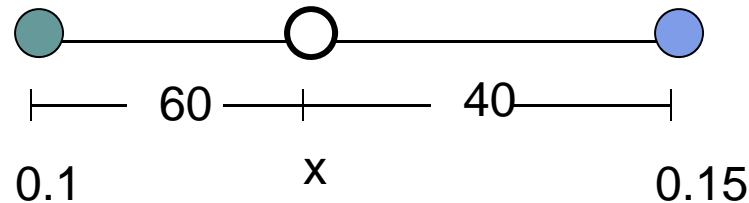


Linear interpolation Example



$$x = \frac{b}{(a+b)} * v1 + \frac{a}{(a+b)} * v2$$

- If $a = 60$, $b = 40$
- RGB color at $v1 = (0.1, 0.4, 0.2)$
- RGB color at $v2 = (0.15, 0.3, 0.5)$
- Red value of $v1 = 0.1$, red value of $v2 = 0.15$



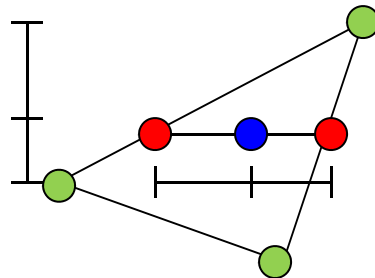
$$\begin{aligned} \text{Red value of } x &= 40/100 * 0.1 + 60/100 * 0.15 \\ &= 0.04 + 0.09 = 0.13 \end{aligned}$$

Similar calculations for Green and Blue values

Gouraud Shading



- Interpolate triangle color
 1. Interpolate y distance of end points (**green dots**) to get color of two end points in scanline (**red dots**)
 2. Interpolate x distance of two end points in scanline (**red dots**) to get color of pixel (**blue dot**)



Gouraud Shading Function (Pg. 433 of Hill)



```
for(int y = Ybott; y < Ytop; y++) // for each scan line
{
    find xleft    and xright
    find colorleft and colorright
    colorinc = (colorright - colorleft) / (xright - xleft)
    for(int x = xleft, c = colorleft; x < xright; x++, c+ = colorinc)
    {
        put c into the pixel at (x, y)
    }
}
```



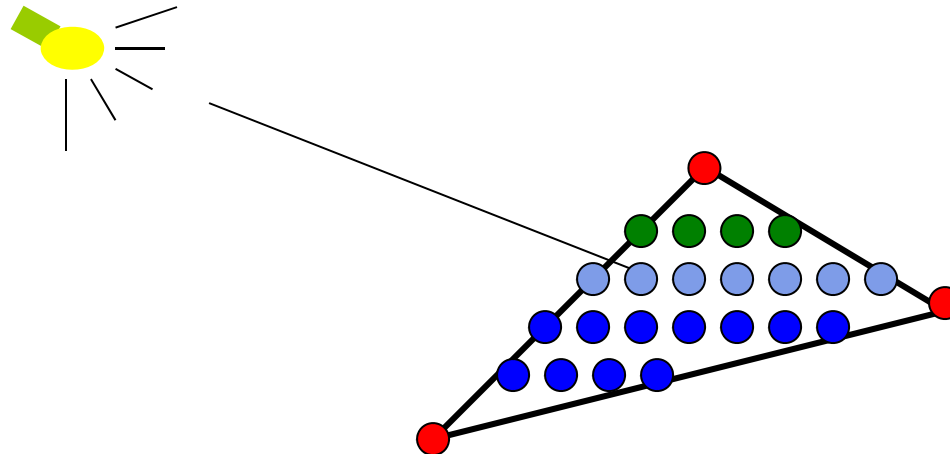
Gouraud Shading Implementation

- Vertex lighting interpolated across entire face pixels if passed to fragment shader in following way

1. **Vertex shader:** Calculate output color in vertex shader, Declare output vertex color as **out**

$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^\beta + k_a I_a$$

2. **Fragment shader:** Declare color as **in**, use it, already interpolated!!

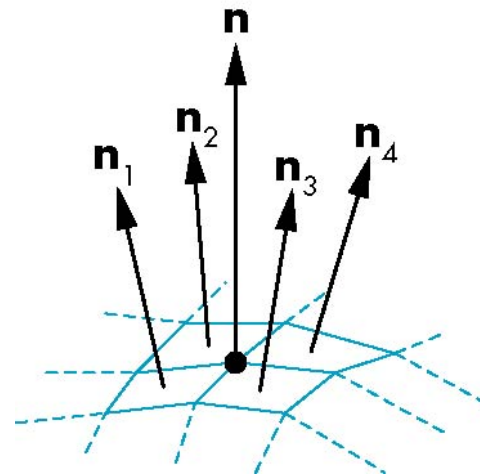




Calculating Normals for Meshes

- For meshes, already know how to calculate face normals (e.g. Using Newell method)
- For polygonal models, Gouraud proposed using average of normals around a mesh vertex

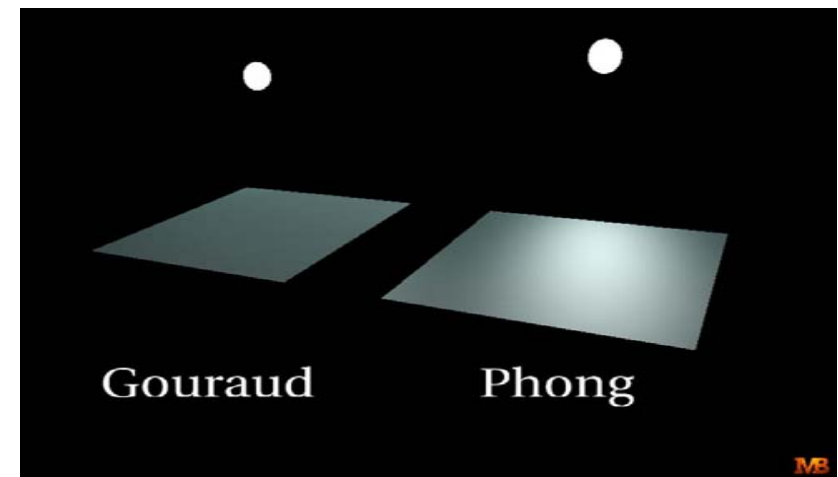
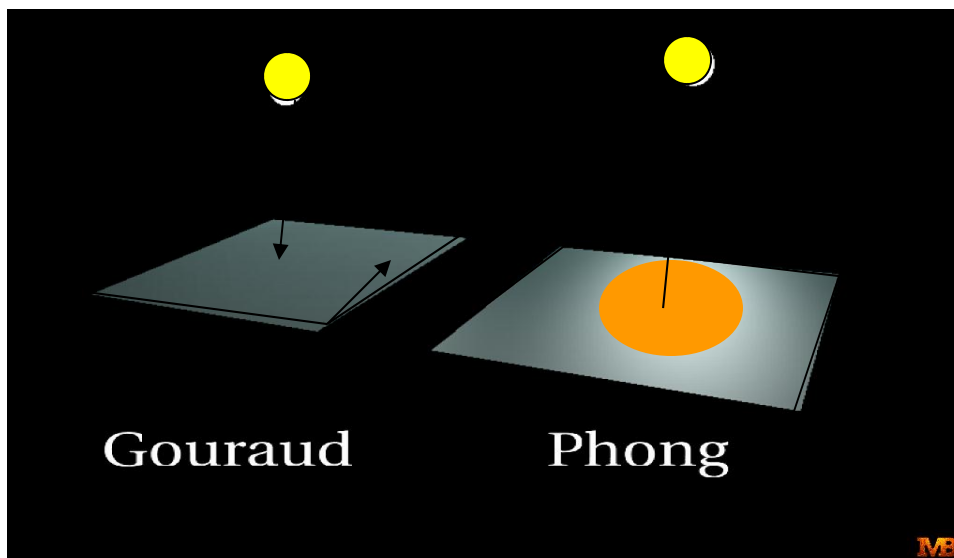
$$\mathbf{n} = (\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4) / |\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4|$$





Gouraud Shading Problem

- If polygon mesh surfaces have high curvatures, Gouraud shading may show edges
- Lighting in the polygon interior can be inaccurate
- Phong shading may look smooth





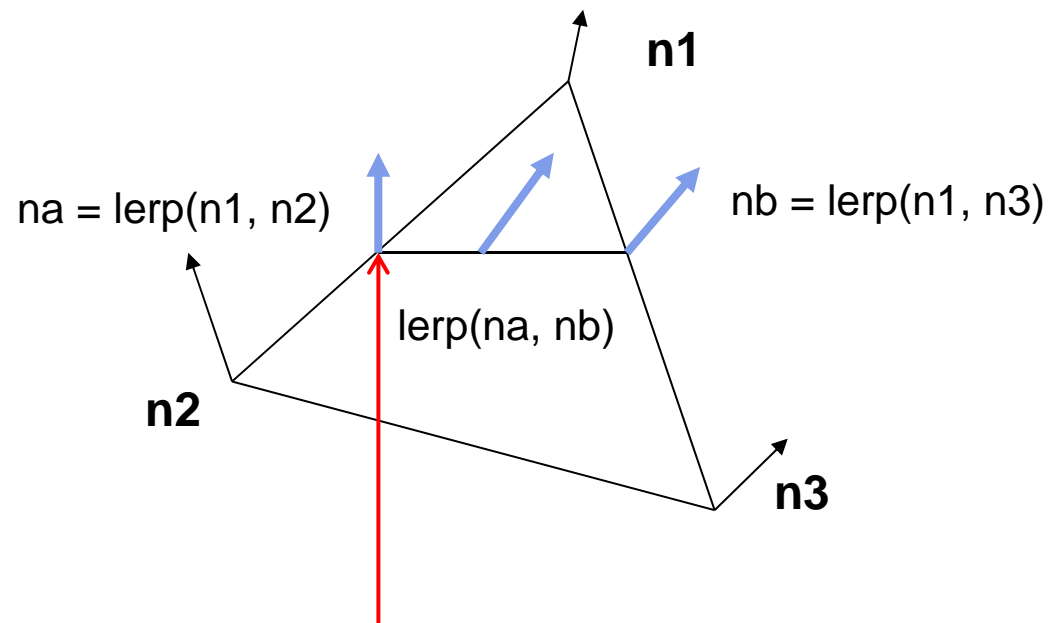
Phong Shading

- Need normals for all pixels – not provided by user
- Instead of interpolating vertex color
 - Interpolate **vertex normal and vectors** to calculate normal (and vectors) at each *each pixel* inside polygon
 - Use pixel normal to calculate Phong at pixel (**per pixel lighting**)
- Phong shading algorithm interpolates normals and compute lighting in fragment shader



Phong Shading (Per Fragment)

- Normal interpolation



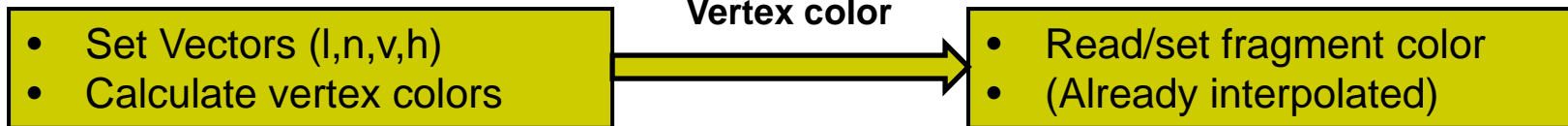
At each pixel, need to interpolate
Normals (n) and vectors v and l



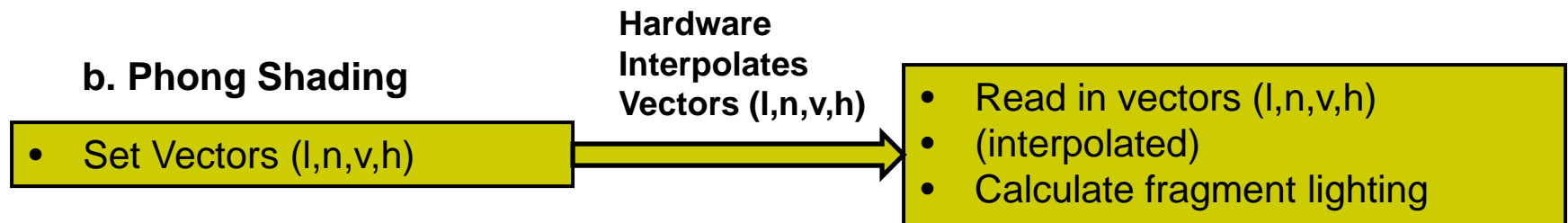
Gouraud Vs Phong Shading Comparison

- Phong shading more work than Gouraud shading
 - Until recently not available in real time systems
 - Now can be done using fragment shaders

a. Gouraud Shading



b. Phong Shading



Per-Fragment Lighting Shaders I



```
// vertex shader
in vec4 vPosition;
in vec3 vNormal;
```

```
// output values that will be interpolated per-fragment
```

```
out vec3 fN;
out vec3 fE;
out vec3 fL;
```

← Declare variables **n**, **v**, **l** as **out** in vertex shader

```
uniform mat4 ModelView;
uniform vec4 LightPosition;
uniform mat4 Projection;
```


Per-Fragment Lighting Shaders II



```
void main()
```

```
{
```

```
    fN = vNormal;  
    fE = vPosition.xyz;  
    fL = LightPosition.xyz;
```

← Set variables **n**, **v**, **l** in vertex shader

```
    if( LightPosition.w != 0.0 ) {  
        fL = LightPosition.xyz - vPosition.xyz;  
    }
```

```
    gl_Position = Projection*ModelView*vPosition;  
}
```



Per-Fragment Lighting Shaders III

```
// fragment shader
```

```
// per-fragment interpolated values from the vertex shader
```

```
in vec3 fN;
```

```
in vec3 fL;
```

```
in vec3 fE;
```

← Declare vectors n, v, l as **in** in fragment shader
(Hardware interpolates these vectors)

```
uniform vec4 AmbientProduct, DiffuseProduct, SpecularProduct;
```

```
uniform mat4 ModelView;
```

```
uniform vec4 LightPosition;
```

```
uniform float Shininess;
```

Per-Fragment Lighting Shaders IV



```
void main()
{
    // Normalize the input lighting vectors

    vec3 N = normalize(fN);
    vec3 E = normalize(fE); ← Use interpolated variables n, v, l
    vec3 L = normalize(fL);   in fragment shader

    vec3 H = normalize( L + E ); ←
    vec4 ambient = AmbientProduct;
```

$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^\beta + k_a I_a$$



Per-Fragment Lighting Shaders V

```
float Kd = max(dot(L, N), 0.0); ← Use interpolated variables n, v, l
    vec4 diffuse = Kd*DiffuseProduct; in fragment shader

float Ks = pow(max(dot(N, H), 0.0), Shininess);
    vec4 specular = Ks*SpecularProduct;

// discard the specular highlight if the light's behind the vertex
if( dot(L, N) < 0.0 )
    specular = vec4(0.0, 0.0, 0.0, 1.0);

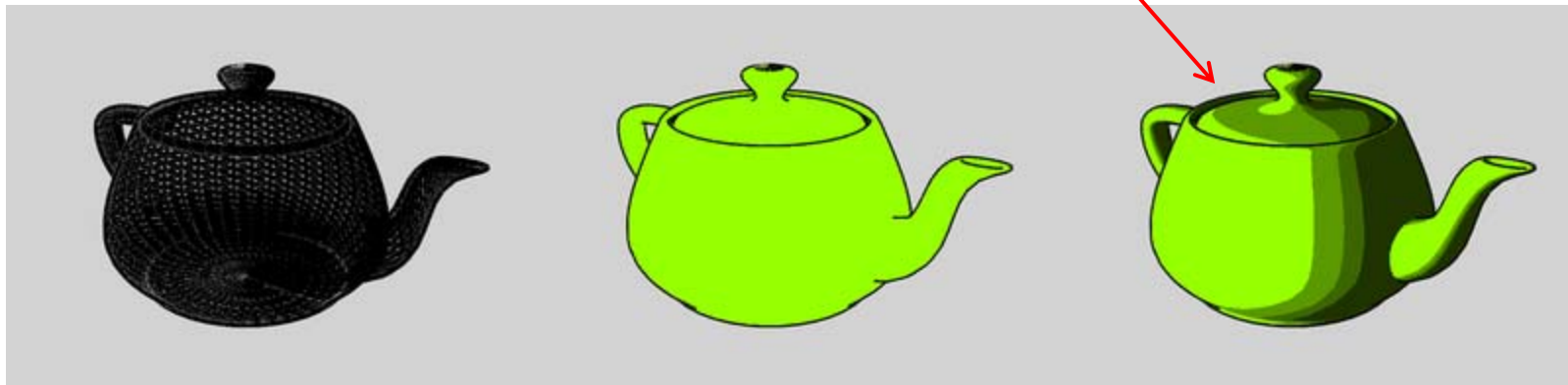
gl_FragColor = ambient + diffuse + specular;
gl_FragColor.a = 1.0;
}
```

$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^\beta + k_a I_a$$



Toon (or Cel) Shading

- Non-Photorealistic (NPR) effect
- Shading in bands of color





Toon (or Cel) Shading

- How?
- Consider $(\mathbf{l} \cdot \mathbf{n})$ diffuse term (or $\cos \theta$) term

$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^\beta + k_a I_a$$

- Clamp values to get toon shading effect

$\mathbf{l} \cdot \mathbf{n}$	Value used
Between 0.75 and 1	0.75
Between 0.5 and 0.75	0.5
Between 0.25 and 0.5	0.25
Between 0.0 and 0.25	0.0



BRDF Evolution

- BRDFs have evolved historically
- 1970's: Empirical models
 - Phong's illumination model
- 1980s:
 - Physically based models
 - Microfacet models (e.g. Cook Torrance model)
- 1990's
 - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000's
 - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000's
 - Measurement & acquisition of time-varying BRDFs (ripening, etc)

Physically-Based Shading Models



- Phong model produces pretty pictures
- Cons: empirical (fudged?) ($\cos^\alpha \phi$), plastic look
- Shaders can implement more lighting/shading models
- Big trend towards Physically-based models
- Physically-based?
 - Based on physics of how light interacts with actual surface
 - Dig into Optics/Physics literature and adapt results
- Classic: Cook-Torrance shading model (TOGS 1982)



Cook-Torrance Shading Model

- Similar ambient and diffuse terms to
- More complex specular component than $(\cos^\alpha \phi)$,
- Define new specular term

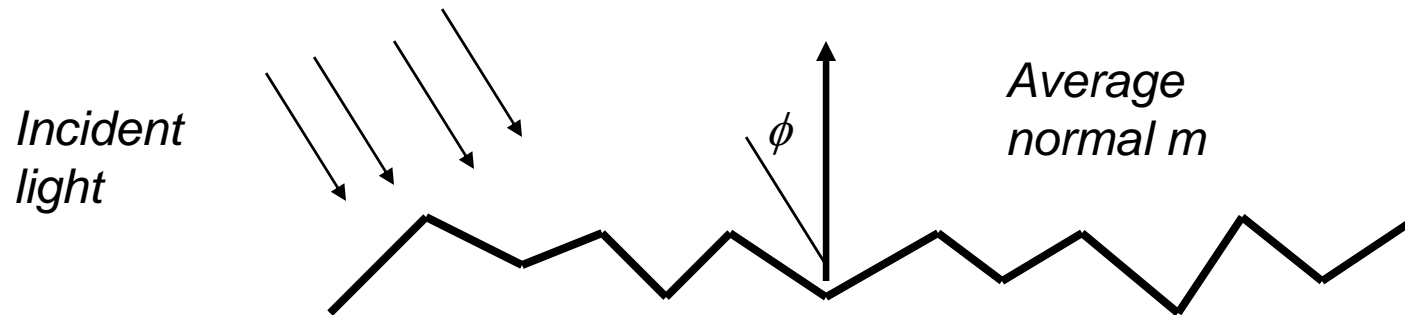
$$\cos^\alpha \phi \rightarrow \frac{F(\phi, \eta)DG}{(\mathbf{n} \cdot \mathbf{v})}$$

- Where
 - D - Distribution term
 - G – Geometric term
 - F – Fresnel term



Distribution Term, D

- **Basic idea:** model surfaces as made up of small V-shaped grooves or “microfacets”

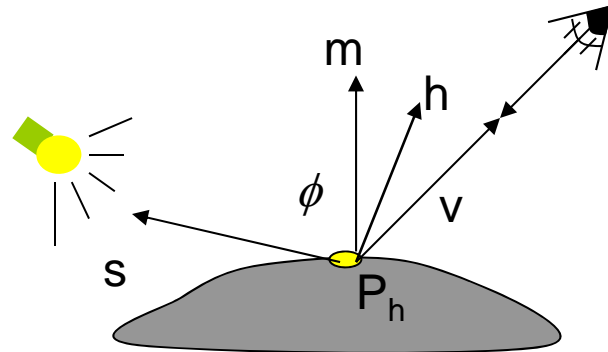


- Many grooves occur at each surface point
- Only perfectly facing grooves contribute
- D term expresses groove directions
- D expresses direction of aggregates (distribution)
- E.g. half of grooves at hit point face 30 degrees, etc



Cook-Torrance Shading Model

- Only microfacets with normal of \mathbf{V} pointing in direction of halfway vector, $\mathbf{h} = \mathbf{s} + \mathbf{v}$, contributes



- Define angle δ as deviation of \mathbf{h} from surface normal
- $D(\delta)$ is fraction of microfacets facing angle δ
- Can actually plug old Phong cosine ($\cos^n \phi$), in as D
- More widely used is Beckmann distribution

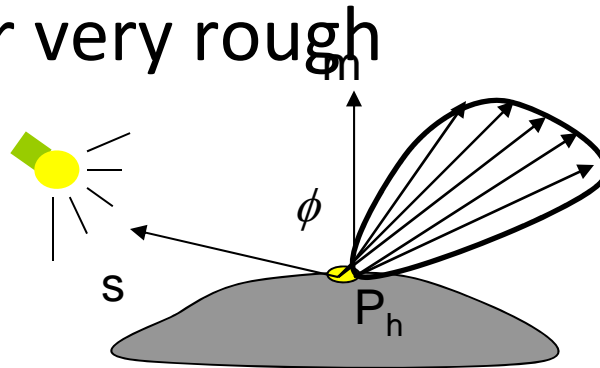
$$D(\delta) = \frac{1}{4\mathbf{m}^2 \cos^4(\delta)} e^{-\left(\frac{\tan(\delta)}{\mathbf{m}}\right)^2}$$

- Where \mathbf{m} expresses roughness of surface



Cook-Torrance Shading Model

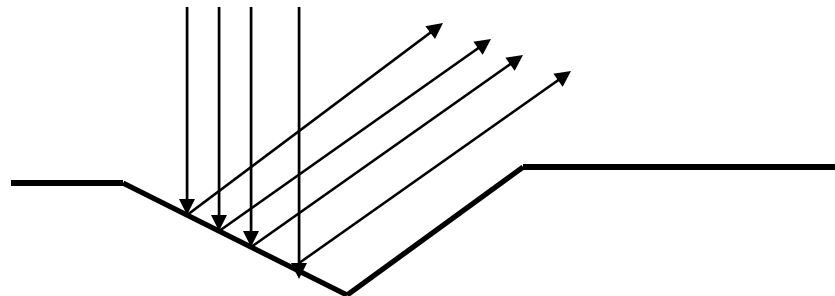
- m is actually Root-mean-square (RMS) value of slope of V-groove
- Basically, m expresses slope of V-groove
- $m = 0.2$ for nearly smooth
- $m = 0.6$ for very rough





Geometric Term, G

- Surface may be so rough that interior of grooves is blocked from light by edges
- This is known as **shadowing** or **masking**
- Geometric term G accounts for this
- Break G into 3 cases:
- G, case a: No self-shadowing

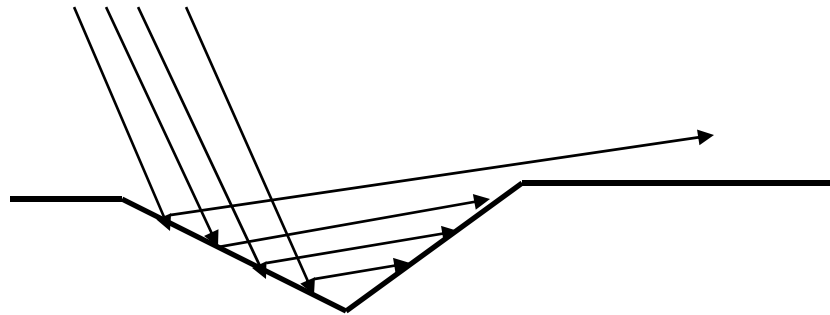


- Mathematically, $G = 1$



Geometric Term, G

- G, case b: No blocking of incident light, partial blocking of exiting light (**masking**)



- Mathematically,

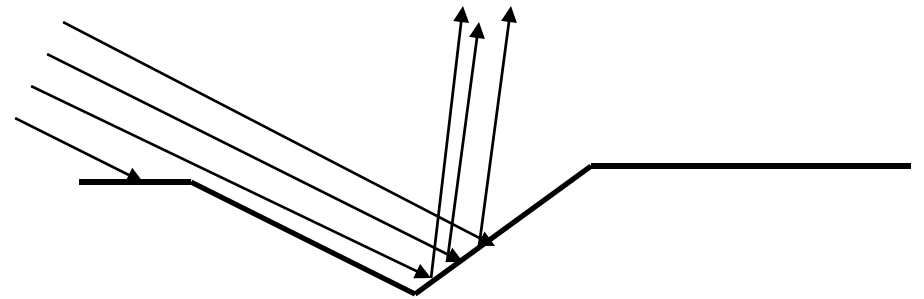
$$G_m = \frac{2(\mathbf{m} \cdot \mathbf{h})(\mathbf{m} \cdot \mathbf{h})}{\mathbf{h} \cdot \mathbf{s}}$$



Geometric Term, G

- G , case c: Partial blocking of incident light, no blocking of exiting light (**shadowing**)
- Mathematically,

$$G_m = \frac{2(\mathbf{m} \cdot \mathbf{h})(\mathbf{m} \cdot \mathbf{s})}{\mathbf{h} \cdot \mathbf{s}}$$



$$G = (1, G_m, G_s)$$

- G term is minimum of 3 cases, hence



Fresnel Term, F

- So, again recall that specular term

$$spec = \frac{F(\phi, \eta) DG}{(\mathbf{n} \cdot \mathbf{v})}$$

- Microfacets are not perfect mirrors
- F term, $F(\phi, \eta)$ gives fraction of incident light reflected
- ϕ is incident angle, η is refractive index of material

$$F = \frac{1}{2} \frac{(g - c)^2}{(g + c)^2} \left\{ 1 + \left(\frac{c(g + c) - 1}{c(g - c) - 1} \right)^2 \right\}$$

- where $c = \cos(\phi) = \mathbf{n} \cdot \mathbf{s}$ and $g^2 = \eta^2 + c^2 + 1$

Other Physically-Based BRDF Models



- Oren-Nayar – Lambertian not specular
- Aishikhminn-Shirley – Grooves not v-shaped. Other Shapes
- Microfacet generator
- BRDF viewer

BV BRDF Viewer



The image displays the BV BRDF Viewer interface, which is divided into several panels:

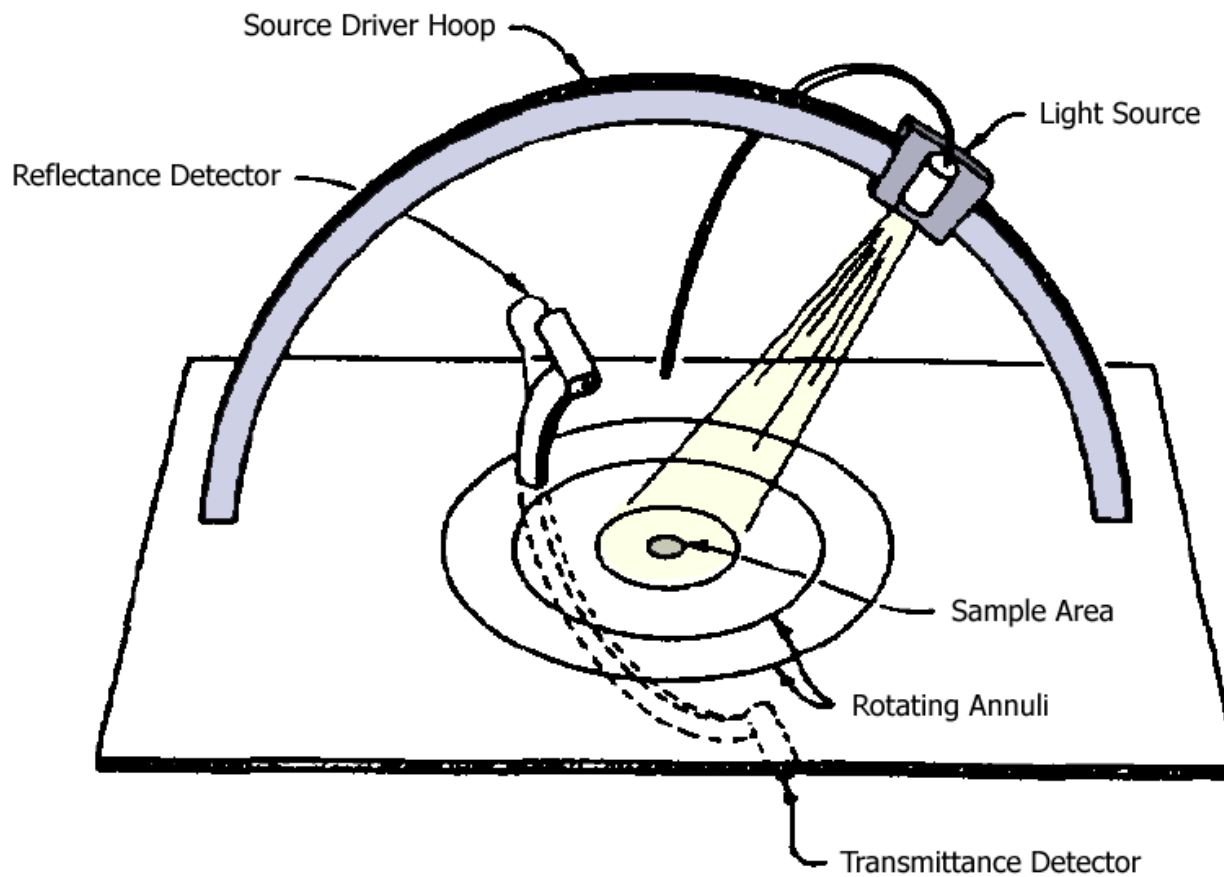
- BV Options:** Contains 'Viewers' (2D slices, Lit Sphere, 3D view, Lit Plane) and 'Options' (Logarithm, Multiply by: $\cos(\theta_{in})$, $\cos(\theta_{in}) * \cos(\theta_{out})$, $\cos(\theta_{out})$, $\cos(\theta_{in}) + \cos(\theta_{out})$). Buttons for 'New Window' and 'Quit' are also present.
- BRDF Parameter panel (Top):** Shows parameters for the Cook-Torrance-Sparrow BRDF:
 - Surface roughness m : 0.13
 - Index of Refraction: Real part 1.60, Imaginary Part -0.20
 - Specular reflectivity: 0.60
 - Diffuse reflectivity: 0.40A text box explains: "This is the Cook-Torrance-Sparrow BRDF, using a Beckmann microfacet distribution function, Blinn's geometric shadowing term, and Fresnel reflection. The parameters are the surface roughness m (as used in the Beckmann distribution), the index of refraction, and the diffuse and specular reflectivities."
- BRDF Parameter panel (Bottom):** Shows parameters for Greg Ward's Elliptical Gaussian BRDF:
 - Surface roughness in X direction: 0.05
 - Surface roughness in Y direction: 0.26
 - Specular reflectivity: 0.05
 - Diffuse reflectivity: 0.40
 - Orientation: A circular dial icon.A text box explains: "This is Greg Ward's Elliptical Gaussian BRDF. It is predicted by a simple, but physically correct, rough-surface model, assuming different surface roughness along the X and Y directions. Shadowing, masking and Fresnel reflection are not included."
- 3D Viewport (Top):** Titled "bv [0]: Torrance-Sparrow m=0.13, n=1.60-0.20i, rs=0.60, rd=0.40". It shows a blue sphere on a grey plane with incident and reflected rays.
- 3D Viewport (Bottom):** Titled "bv [0]: (Ward sx=0.05, sy=0.26, rs=0.05, rd=0.40) rotated by +000". It shows a blue sphere on a grey plane, rotated, with incident and reflected rays.



BRDF Evolution

- BRDFs have evolved historically
- 1970's: Empirical models
 - Phong's illumination model
- 1980s:
 - Physically based models
 - Microfacet models (e.g. Cook Torrance model)
- 1990's
 - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000's
 - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000's
 - Measurement & acquisition of time-varying BRDFs (ripening, etc)

Measuring BRDFs



Murray-Coleman and Smith Gonioreflectometer. (Copied and Modified from [Ward92]).

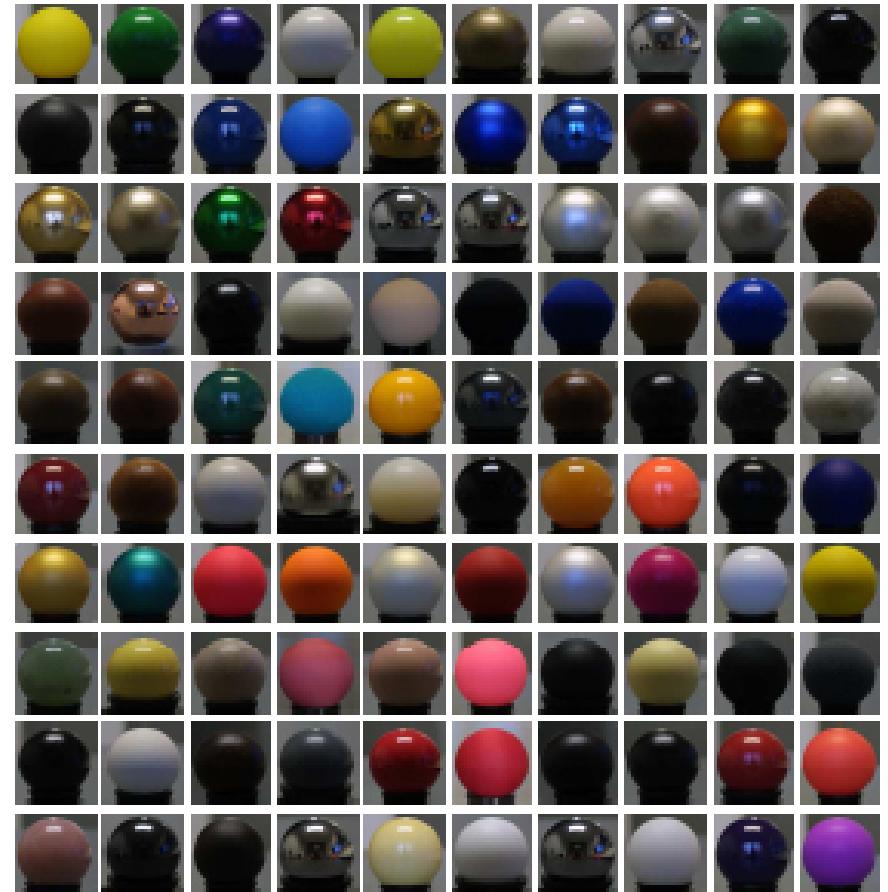


Measured BRDF Samples

- Mitsubishi Electric Research Lab (MERL)

<http://www.merl.com/brdf/>

- Wojciech Matusik
- MIT PhD Thesis
- 100 Samples





BRDF Evolution

- BRDFs have evolved historically
- 1970's: Empirical models
 - Phong's illumination model
- 1980s:
 - Physically based models
 - Microfacet models (e.g. Cook Torrance model)
- 1990's
 - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000's
 - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- **Late 2000's**
 - **Measurement & acquisition of time-varying BRDFs (ripening, etc)**



Time-varying BRDF

- BRDF: How different materials reflect light
- Time varying?: how reflectance changes over time





References

- Interactive Computer Graphics (6th edition), Angel and Shreiner
- Computer Graphics using OpenGL (3rd edition), Hill and Kelley