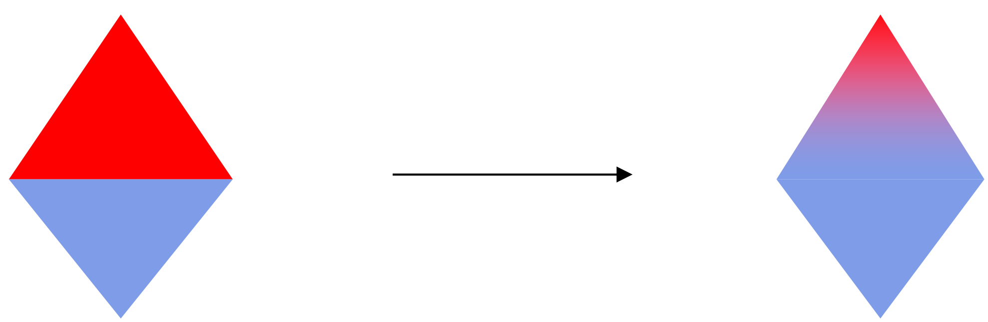
Computer Graphics
CS 543 – Lecture 7 (Part 3)
Lighting, Shading and Materials (Part 3)

Prof Emmanuel Agu

Computer Science Dept.
Worcester Polytechnic Institute (WPI)

Smooth shading

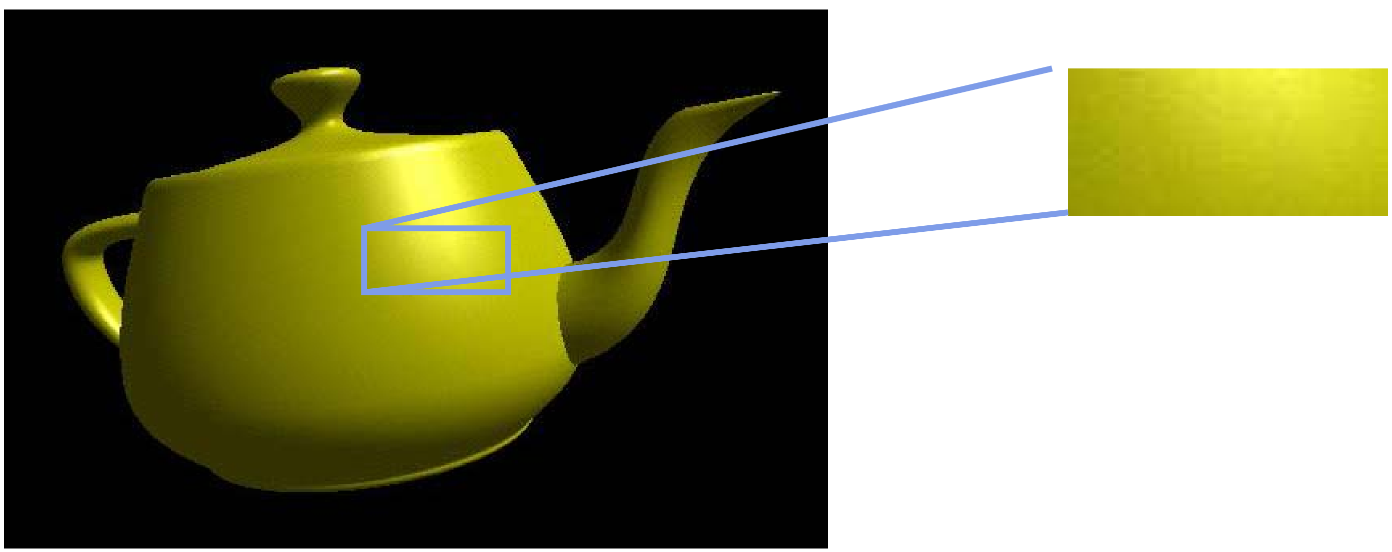
- Fix mach band effect – remove edge discontinuity
- Compute lighting for more points on each face
- 2 popular methods:
  - Gouraud shading
  - Phong shading



Flat shading **Smooth shading**

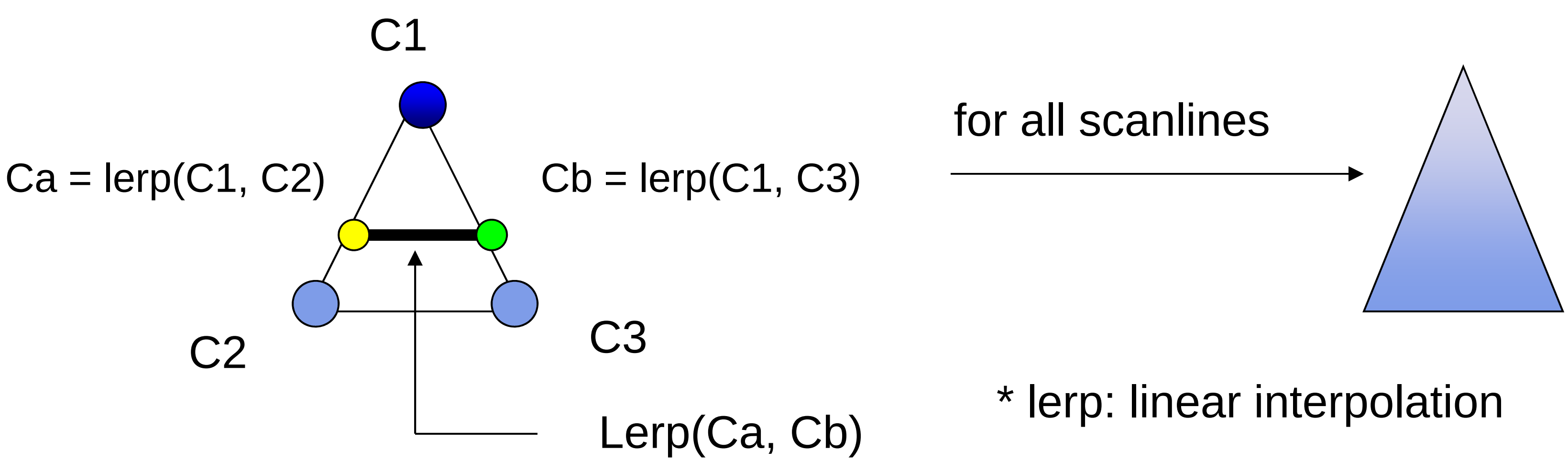
Gouraud Shading

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



Gouraud Shading

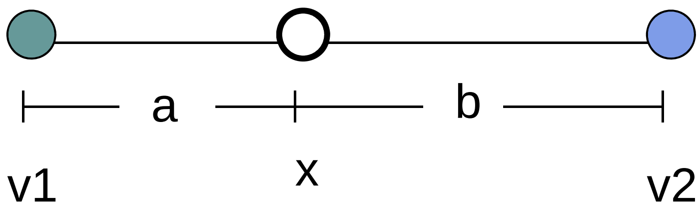
- Compute vertex color in vertex shader
- Shade interior pixels: color **interpolation** (normals are not needed)



* lerp: linear interpolation

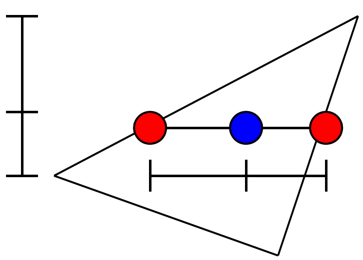
Gouraud Shading

- Linear interpolation



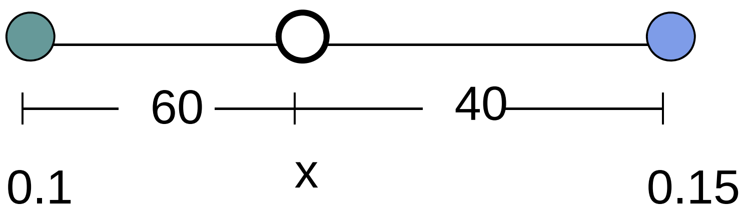
x = b / (a+b) * v1 + a/(a+b) * v2

- Interpolate triangle color
  - use y distance to interpolate two end points in scanline,
  - and use x distance to interpolate interior pixel colors



Linear Interpolation Example

- 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



Red value of x = 40 /100 * 0.1 + 60/100 * 0.15
= 0.04 + 0.09 = 0.13

Similar calculations for Green and Blue values

Gouraud Shading Function (Pg. 433 of Hill)

for(int y = y_{bott}; y < y_{top}; y++) // for each scan line

{

find x_{left} and x_{right}

find $color_{left}$ and $color_{right}$

$color_{inc}$ = ($color_{right}$ – $color_{left}$)/ (x_{right} – x_{left})

for(int x = x_{left}, c = $color_{left}$; x < x_{right}; x++, c+ = $color_{inc}$)

{

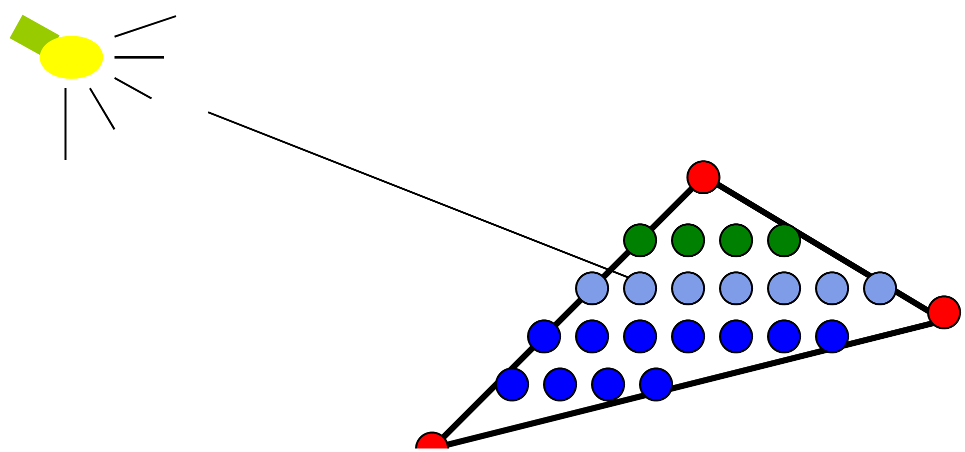
put c into the pixel at (x, y)

}

}

Smooth Shading Implemenation

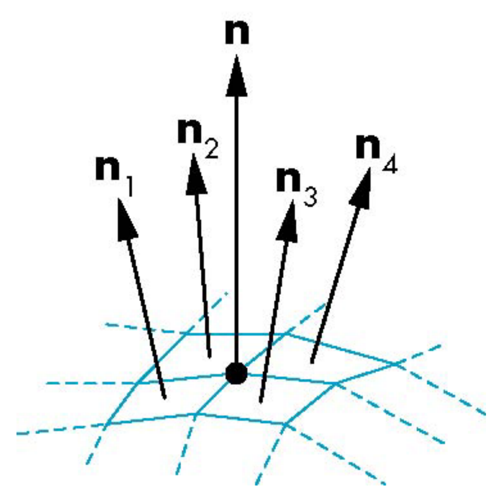
- Use **varying** declaration for interpolation
- Vertex lighting interpolated across entire face pixels if passed to the fragment shader as a varying variable (smooth shading)
  1. **Vertex shader:** Calculate output color in vertex shader, Declare output vertex color as **varying**
  2. **Fragment shader:** Use varying color type, already interpolated!!



Mesh Shading

- 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

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



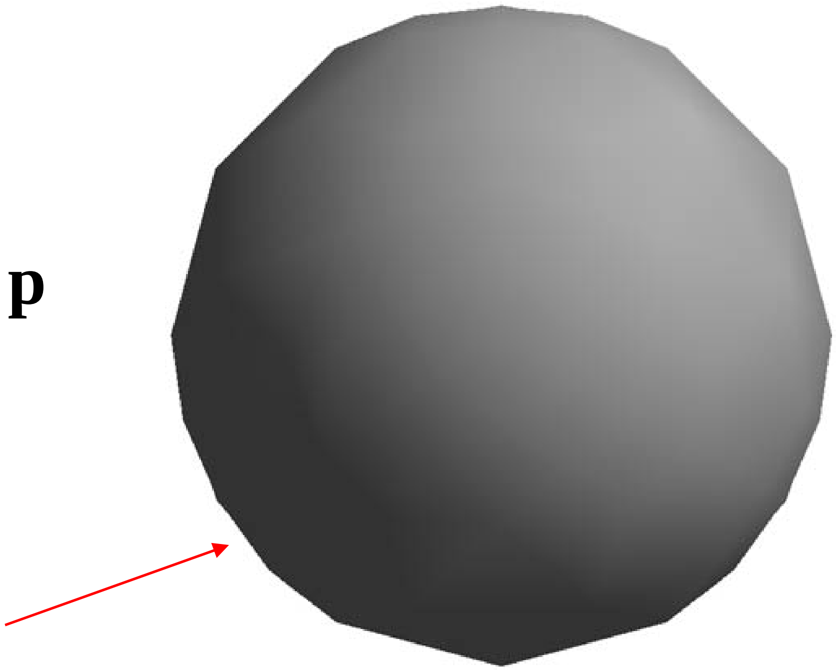
Normals Variability

- Triangles have a single normal
  - Shades at the vertices as computed by the Phong model can be almost same
  - Identical for a distant viewer (default) or if there is no specular component
- Consider a sphere
- Want different normals at

each vertex

Smooth Shading

- We can set a new normal at each vertex
- Easy for sphere model
  - If centered at origin $\mathbf{n} = \mathbf{p}$
- Now smooth shading works
- Note *silhouette edge*

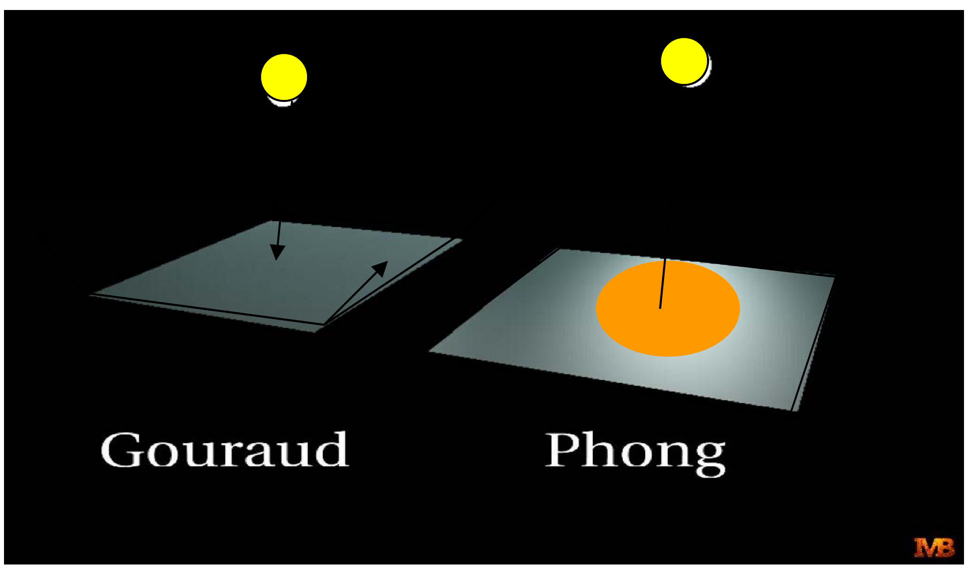


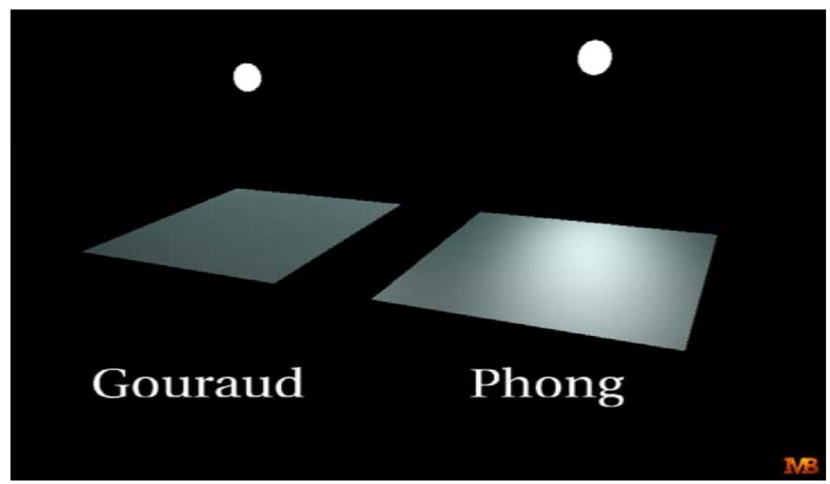
Gouraud Vs Phong Shading

- **Gouraud Shading:** interpolates **vertex colors**
  - Find vertex normals
  - Apply modified Phong model at each vertex
  - Interpolate vertex colors across each polygon
- **Phong shading:** interpolates **vertex normals**
  - Find vertex normals
  - Interpolate vertex normals across edges
  - Interpolate edge normals across polygon
  - Use interpolated normal to apply modified Phong model at each fragment

Gouraud Shading Problem

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



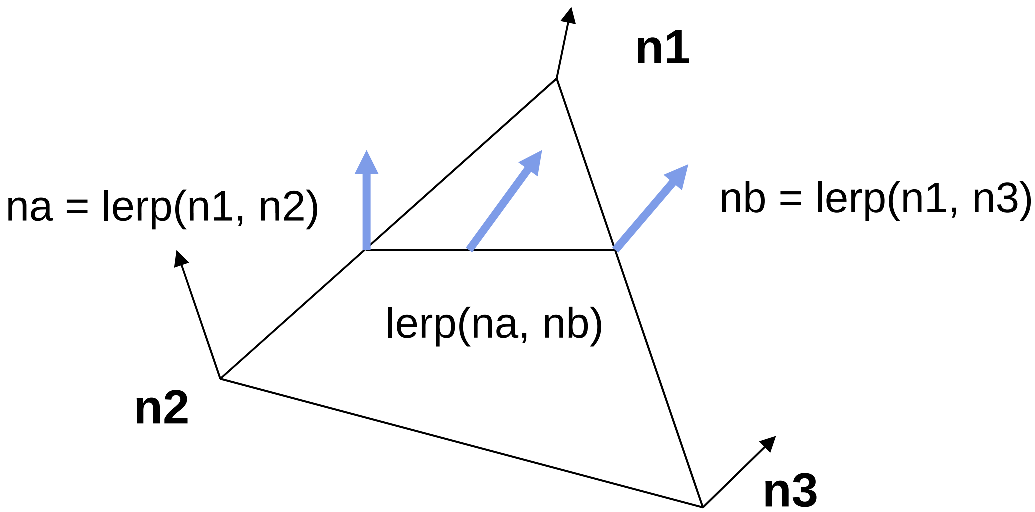


Phong Shading

- Need normals for all pixels – not provided by user
- Instead of interpolating vertex color
  - Interpolate **vertex normal** to calculate normal 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 during rasterization
  - (need to map normal back to world or eye space though)

Phong Shading

- Normal interpolation



Gouraud Vs Phong Shading Comparison

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

Per-Vertex Lighting Shaders I

```
// vertex shader
in vec4 vPosition;
in vec3 vNormal;
out vec4 color;  //vertex shade

// light and material properties
uniform vec4 AmbientProduct, DiffuseProduct, SpecularProduct;
uniform mat4 ModelView;
uniform mat4 Projection;
uniform vec4 LightPosition;
uniform float Shininess;
```

Per-Vertex Lighting Shaders II

```
void main( )
{
    // Transform vertex  position into eye coordinates
    vec3 pos = (ModelView * vPosition).xyz;

    vec3 L = normalize( LightPosition.xyz - pos );
    vec3 E = normalize( -pos );
    vec3 H = normalize( L + E );

    // Transform vertex normal into eye coordinates
    vec3 N = normalize( ModelView*vec4(vNormal, 0.0) ).xyz;
```

Per-Vertex Lighting Shaders III

```
// Compute terms in the illumination equation
    vec4 ambient = AmbientProduct;

    float Kd = max( dot(L, N), 0.0 );
    vec4  diffuse = Kd*DiffuseProduct;
    float Ks = pow( max(dot(N, H), 0.0), Shininess );
    vec4  specular = Ks * SpecularProduct;
    if( dot(L, N) < 0.0 )  specular = vec4(0.0, 0.0, 0.0, 1.0);
    gl_Position = Projection * ModelView * vPosition;

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

Per-Vertex Lighting Shaders IV

```
// fragment shader

in vec4 color;

void main()
{
    gl_FragColor = color;
}
```

Per-Fragment Lighting Shaders I

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

// output values that will be interpolatated per-fragment
out vec3 fN;
out vec3 fE;
out vec3 fL;

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

Per-Fragment Lighting Shaders II

```
void main()
{
   fN = vNormal;
   fE = vPosition.xyz;
   fL = LightPosition.xyz;

   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;

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);
    vec3 L = normalize(fL);

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

Per-Fragment Lighting Shaders V

```
float Kd = max(dot(L, N), 0.0);
    vec4 diffuse = Kd*DiffuseProduct;

    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;
}
```

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

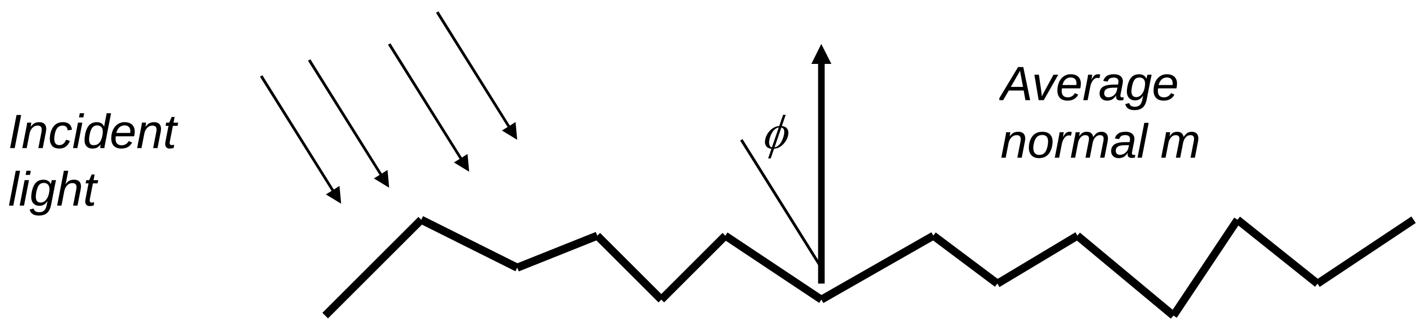
- Similar ambient and diffuse terms to
- More complex specular component than ($cos^{\alpha}\phi$),
- Define new specular term

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

- Where
  - D - Distribution term
  - G – Geometric term
  - F – Fresnel term
- Now, explain each term

Distribution Term, D

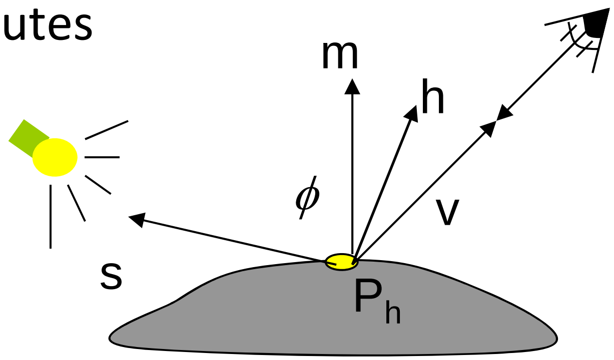
- **Basic idea:** model surfaces as made up of small V-shaped grooves or “microfacets”



- 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 V pointing in direction of halfway vector, **h = s + v**, contributes



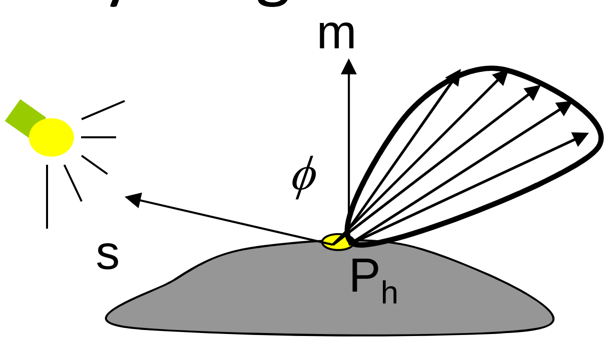
- Define angle δ as deviation of **h** from surface normal
- $D(\delta)$ is fraction of microfacets facing angle δ
- Can actually plug old Phong cosine ($\cos^n \phi$), in as D
- More widely used is Beckmann distribution

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

- Where **m** expresses roughness of surface

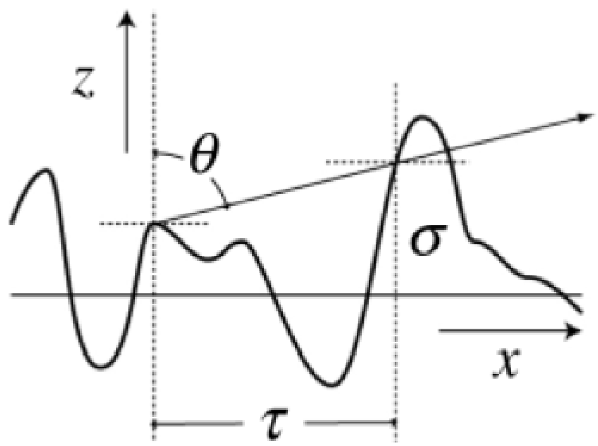
Cook-Torrance Shading Model

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



Microfacet Slope

- Slope



- Beckmann Distribution of Microfacet Slope

$$D(\alpha) = \frac{1}{\sqrt{\pi} m^2 \cos^2 \alpha} e^{-\frac{\tan^2 \alpha}{m^2}} \qquad m = \frac{2\sigma}{\tau}$$

Beckmann

Other Microfacet Distributions

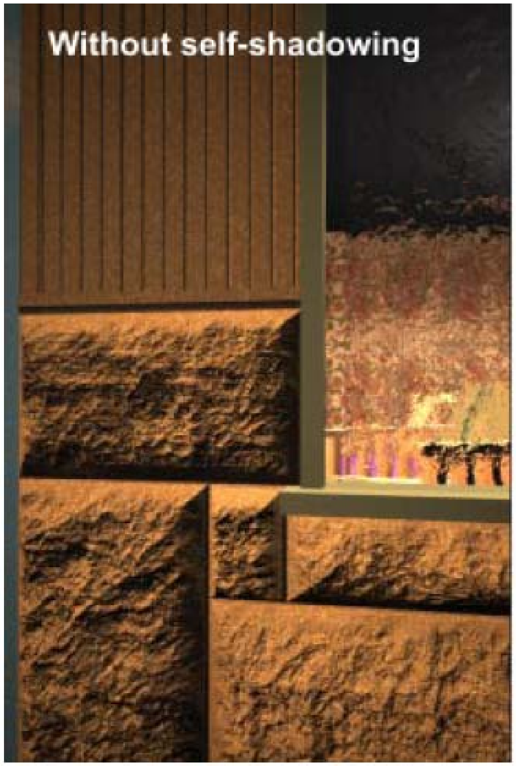
- Some popular distributions

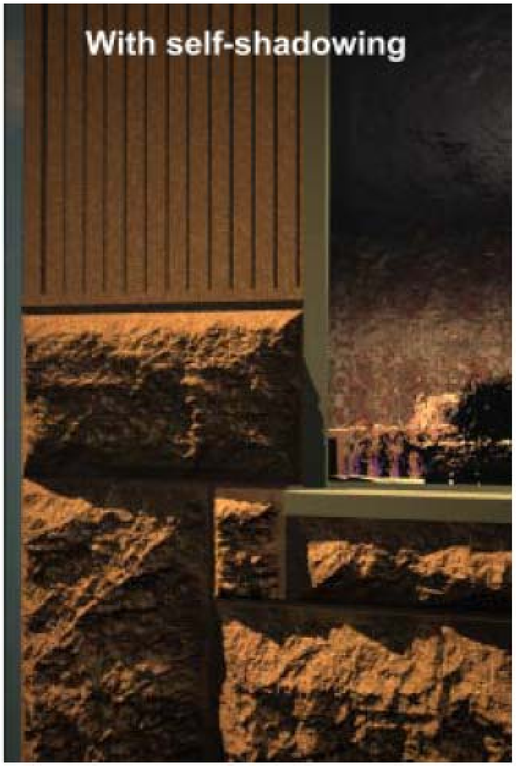
- **Blinn** $D_1(\alpha) = \cos^{c_1} \alpha$ $\quad c_1 = \dfrac{\ln 2}{\ln \cos \beta}$
- **Torrance-Sparrow** $D_2(\alpha) = e^{-(c_2 \alpha)^2}$ $\quad c_2 = \dfrac{\sqrt{2}}{\beta}$
- **Trowbridge-Reitz** $D_3(\alpha) = \dfrac{c_3^2}{(1 - c_3^2)\cos^2 \alpha - 1}$

$$c_3 = \left(\frac{\cos^2 \beta - 1}{\cos^2 \beta - \sqrt{2}} \right)^{1/2}$$

Self-Shadowing

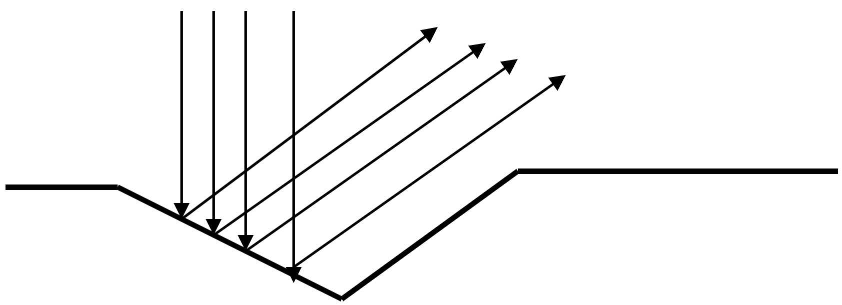
- Geometric Term, G





Geometric Term, G

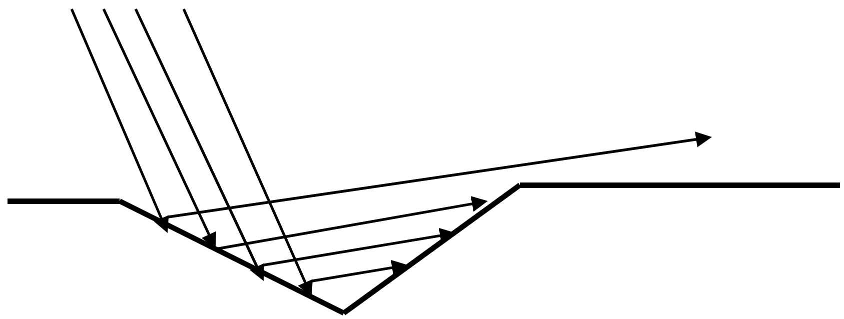
- Surface may be so rough that interior of grooves is blocked from light by edges
- This is known as **shadowing** or **masking**
- Geometric term G accounts for this
- Break G into 3 cases:
- **G, case a:** No self-shadowing



- Mathematically, G = 1

Geometric Term, G

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



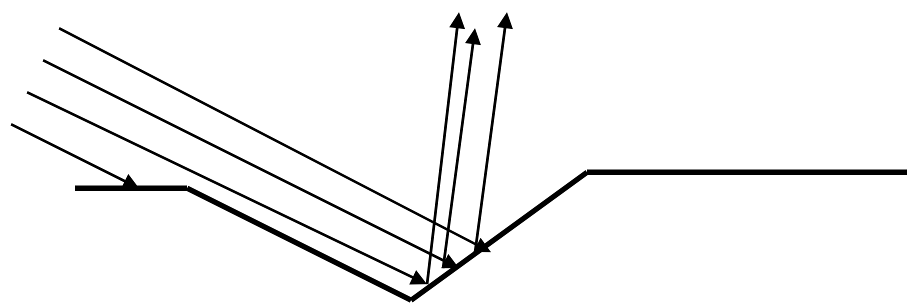
- Mathematically,

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

Geometric Term, G

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

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



- G term is minimum of 3 cases, hence

$$G = \left(1, G_m, G_s\right)$$

Fresnel Term, F

- So, again recall that specular term

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

- 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(ϕ) = **m.s** and $g^2 = \eta^{2+} c^2 + 1$

Fresnel Term, F

- Combining expressions

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

- In above expression for F, could simply use *FDG*
- Why divide by **m.v**?
- Accounts for why when eye is close to surface, more microfacets are seen per solid angle than when eye is close to normal

Fresnel Term, F

- Refractive index, η is actually wavelength dependent which also makes *F* wavelength dependent

$$I_r = I_{ar} k_a F(0,\eta_r) + I_{sr} d\varpi \cdot k_d \times \mathbf{lambert} + I_{sr} k_s d\varpi \frac{F(\phi,\eta_r) DG}{(\mathbf{m}\cdot\mathbf{v})}$$

- Ambient and diffuse terms are based on Fresnel component at normal incidence (recall their values are independent of angle)
- Lambert term is given as before as

$$\mathbf{lambert} = \max\left(0, \frac{\mathbf{s}\cdot\mathbf{m}}{|\mathbf{s}||\mathbf{m}|}\right)$$

- Diffuse term also contains solid angle at hit point, usually set to small value e.g. 0.0001

Fresnel Term, F

- Required that $k_d + k_s = 1$
- For spec, we need $F(\phi, \eta)$
- Usually, $F(0, \eta)$ is available from tables (Terloukian)
- Inserting $\phi = 0$, $c = 1$ in expression for F

$$F = \frac{(\eta - 1)^2}{(\eta + 1)^2}$$

- And

$$\eta = \frac{1 + \sqrt{F_0}}{1 - \sqrt{F_0}}$$

- So, use tabulated $F(0, \eta)$ values to calculate η
- Then use calculated η in original equation for F

Some Fresnel Values, F(0)

- At incident angle 0

Material	Fresnel Value (R,G,B)
Water	0.02, 0.02, 0.02
Plastic	0.05, 0.05, 0.05
Glass	0.08, 0.08, 0.08
Diamond	0.17, 0.17, 0.17
Copper	0.95, 0.64, 0.54
Aluminum	0.91, 0.92, 0.92

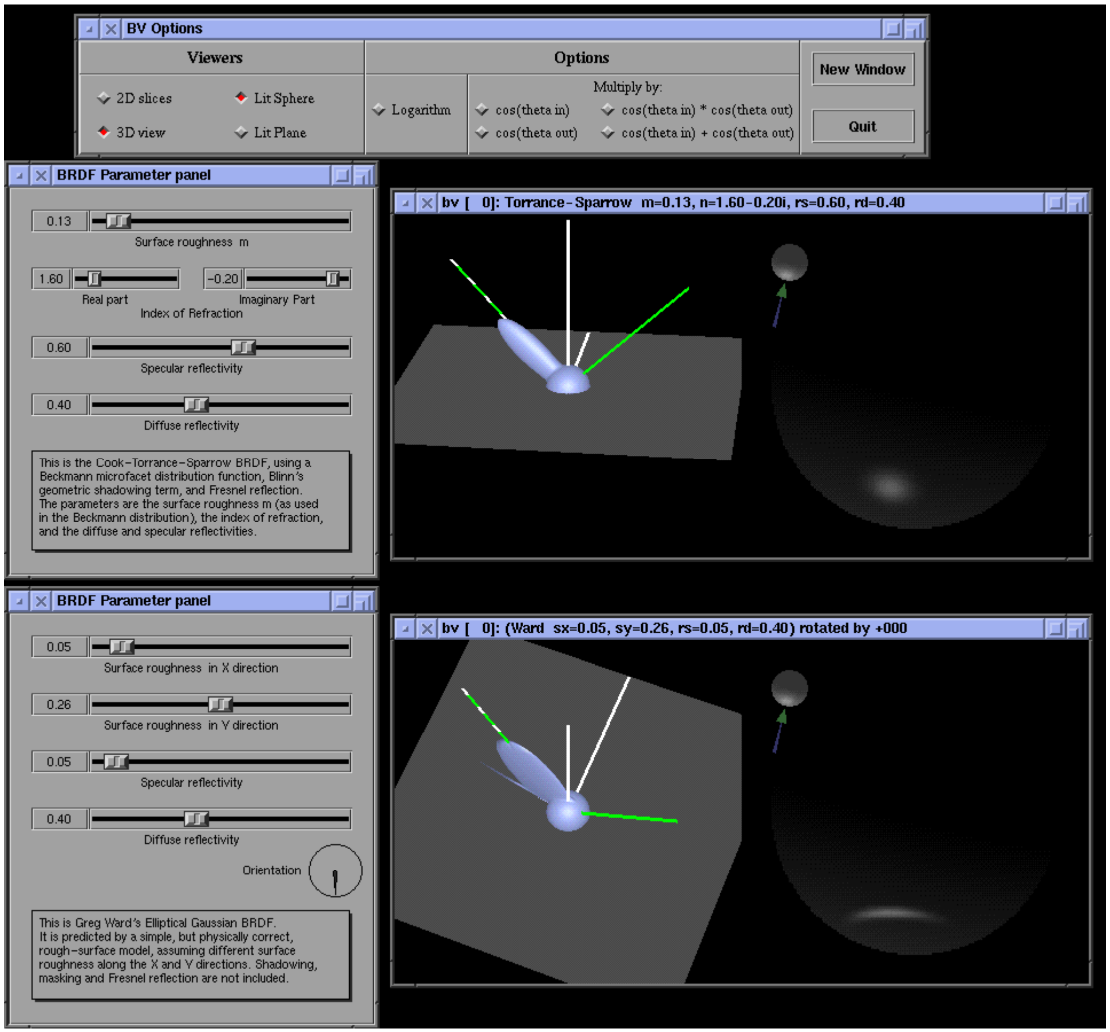
- Schlick approximation to get arbitrary F

$$F(\theta) = F(0) + (1 - F(0))(1 - \cos\theta)^5$$

Final Words

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

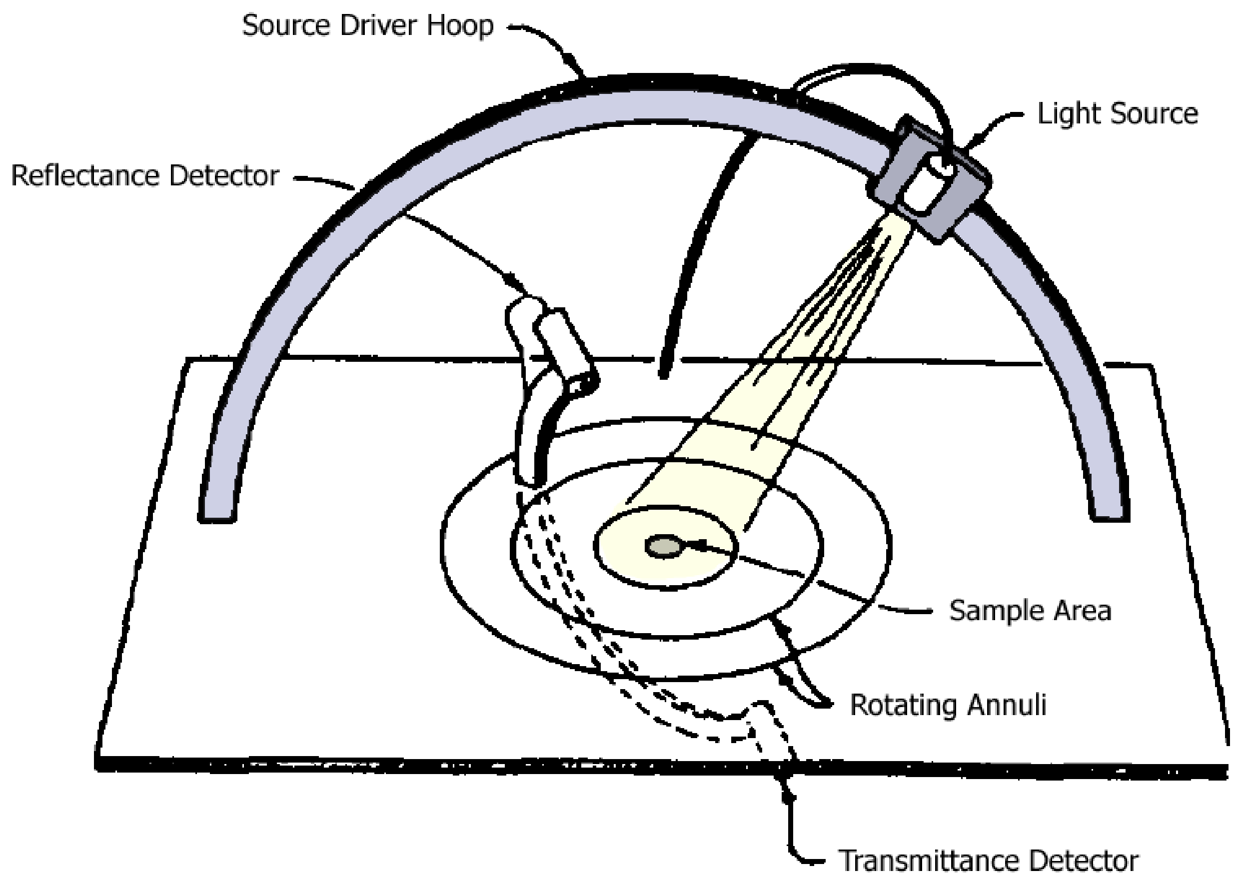
BV BRDF Viewer



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



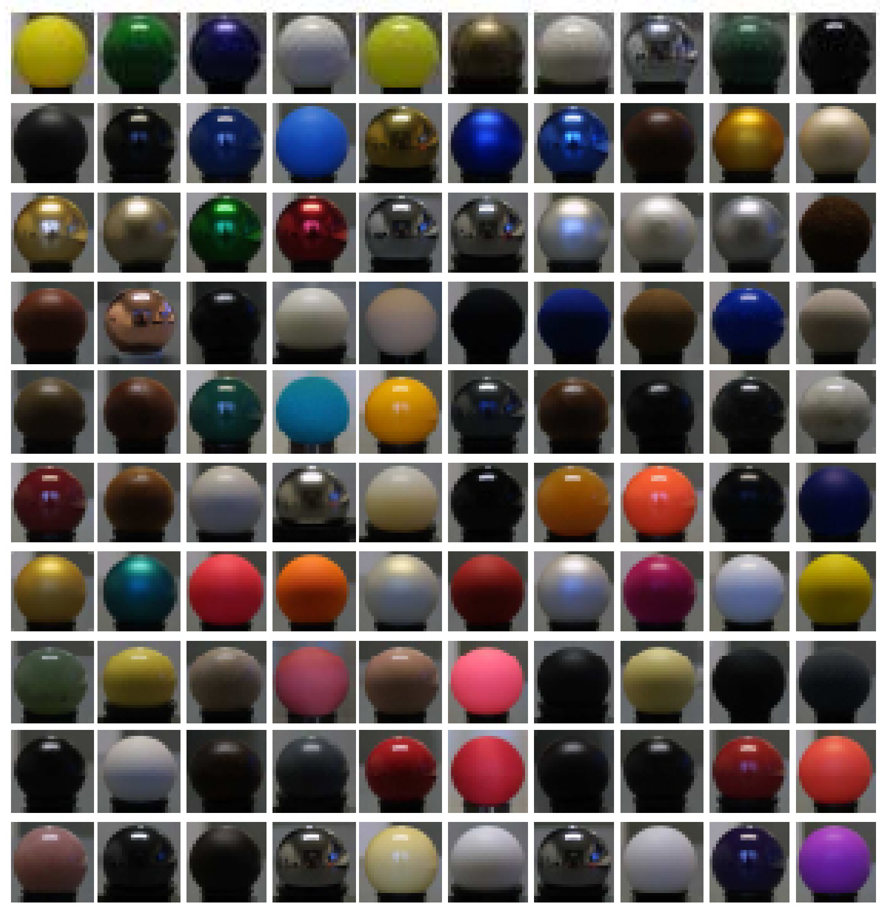
Murray-Coleman and Smith Gonioreflectometer. (Copied and Modified from [Ward92]).

Measured BRDF Samples

- Mitsubishi Electric Research Lab (MERL)

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

- Wojciech Matusik
- MIT PhD Thesis
- 100 Samples



Time-varying BRDF

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

References

- Angel and Shreiner
- Hill and Kelley, chapter 8