Physically-based Rendering

A look at complex lighting

Programming – Computer Graphics

Contents

- Lighting Approximation
- Bidirectional Reflectance Distribution Functions
- Physically-based Lighting
- Oren-Nayar Diffuse Reflectance
- Cook-Torrance Specular Reflectance
  - Beckmann Distribution
  - Fresnel

Lighting Approximation

- We've discussed Phong Lighting
  - Combines Ambient, Diffuse and Specular terms
- Phong approximates lighting using Lambertian Terms (dot products basically) to calculate the diffuse and specular
  - Looks decent enough for most video games
- Phong is a form of Bidirectional Reflectance Distribution Function

Bidirectional Reflectance Distribution Functions

- BRDFs are functions that describe how light is reflected off of surfaces
  - There are many types of BRDFs, all of which take the form:

$$I_{brdf} = f(l, v)$$

Where I_{brdf} is colour of the reflected light (irradiance),
l is the vector to the light, v is a vector to the viewer,
and $f(l, v)$ is the function that calculates the irradiance

- Some BRDFs are called Physically-based as they also simulate light energy interaction with a surface

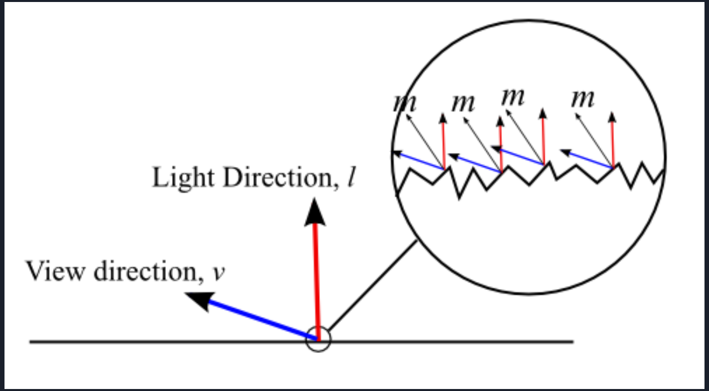
Physically-based Lighting

- Physically-based BRDFs try to mimic the physics of reflection

Both of these are computer generated!

Physically-based Lighting

- They make use of microfacets
  - Small holes and texture on surfaces that effect the light's reflecting vector
  - Plaster and wood have high microfacet counts
  - Mirrors have near zero microfacets
  - Can be thought of as surface "roughness"



Physically-based Lighting

- There are a few different models that use microfacets, but the two common ones are:
  - Oren-Nayar diffuse reflectance
  - Cook-Torrance specular reflectance
- There are many benefits of these lighting models
  - They better simulate light interacting with surfaces
  - One material / shader type can be used to simulate many different material surfaces
  - Glass, Wood, Plaster, Plastic, Carpet, Skin, Metal, Dirt…

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Oren-Nayar Diffuse Reflectance

- Developed by Michael Oren and Shree K. Nayar
- More accurately simulates the diffuse reflectance on a wide range of natural surface types compared to the Lambertian Model
- Uses a roughness value to simulate microfacets
  - 0.0 to 1.0, with 0.0 being a completely smooth surface

Oren-Nayar Diffuse Reflectance

- The mathematical model is a lot more complex than the Phong model

$$L_r = \frac{\rho}{\pi} \cdot \cos\theta_i \cdot (A + (B \cdot \max[0, \cos(\emptyset_i - \emptyset_r)] \cdot \sin\alpha \cdot \tan\beta)) \cdot L_i$$

  – Where…

$$\sigma = surface\ roughness$$

$$A = 1 - 0.5\frac{\sigma^2}{\sigma^2 + 0.33} \qquad B = 0.45\frac{\sigma^2}{\sigma^2 + 0.09}$$

$$\alpha = \max(\theta_i, \theta_r) \qquad \beta = \min(\theta_i, \theta_r)$$

- Yikes! Luckily it can be broken down a bit and simplified

Oren-Nayar Diffuse Reflectance

- The Oren-Nayar portion of the equation is just

$$L_r = \underbrace{\frac{\rho}{\pi} \cdot \cos\theta_i}_{\text{A Lambert Term}} \cdot \overbrace{(A + (B \cdot \max[0, \cos(\emptyset_i - \emptyset_r)] \cdot \sin\alpha \cdot \tan\beta))} \cdot \underbrace{L_i}_{\text{Light and Material Colour}}$$

- The Lambert Term is simply a dot product between the surface normal and the light vector

Oren-Nayar Diffuse Reflectance

- A and B are easy to equate as they just use a roughness value that is a number between 0.0 and 1.0, represented as σ

$$L_r = \frac{\rho}{\pi} \cdot \cos\theta_i \cdot \underbrace{(A + (B}_{} \cdot \max[0, \cos(\emptyset_i - \emptyset_r)] \cdot \sin\alpha \cdot \tan\beta)) \cdot L_i$$

$$\sigma = surface\ roughness$$

$$A = 1 - 0.5\frac{\sigma^2}{\sigma^2 + 0.33}$$

$$B = 0.45\frac{\sigma^2}{\sigma^2 + 0.09}$$

```
float R2 = roughness * roughness;

float A = 1.0f - 0.5f * R2 / (R2 + 0.33f);
float B = 0.45f * R2 / (R2 + 0.09f);
```

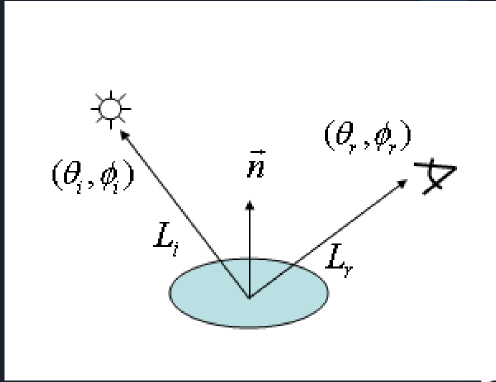
Shader Form

Oren-Nayar Diffuse Reflectance

- The next bit is a little trickier

$$L_r = \frac{\rho}{\pi} \cdot \cos\theta_i \cdot (A + (B \cdot \overbrace{\max[0, \cos(\emptyset_i - \emptyset_r)]} \cdot \sin\alpha \cdot \tan\beta)) \cdot L_i$$

  - $\emptyset_i$ represents the light direction angle
  - $\emptyset_r$ represents the view vector angle
  - Luckily it can be reduced to a vector math form!



Oren-Nayar Diffuse Reflectance

$$\max[0, \gamma\,] = \max[0, \cos(\emptyset_i - \emptyset_r)]$$

$$\gamma = ||E - N * (N \cdot E)|| \cdot ||L - N * (N \cdot L)||$$

– ***E*** is a vector from the surface to the viewer

– ***N*** is the surface normal

– ***L*** is a vector the light is coming from (surface to light)

```
float NdL = max( 0.0f, dot( N, L ) );
float NdE = max( 0.0f, dot( N, E ) );

// CX = max(0, cos(r,i))
vec3 lightProjected = normalize( L - N * NdL );
vec3 viewProjected = normalize( E - N * NdE);
float CX = max( 0.0f, dot( lightProjected, viewProjected ) );
```

Oren-Nayar Diffuse Reflectance

- And the next portion is a little tricky…

$$L_r = \frac{\rho}{\pi} \cdot \cos\theta_i \cdot (A + (B \cdot \max[0, \cos(\emptyset_i - \emptyset_r)] \cdot \overbrace{\sin\alpha \cdot \tan\beta})) \cdot L_i$$

- Where…

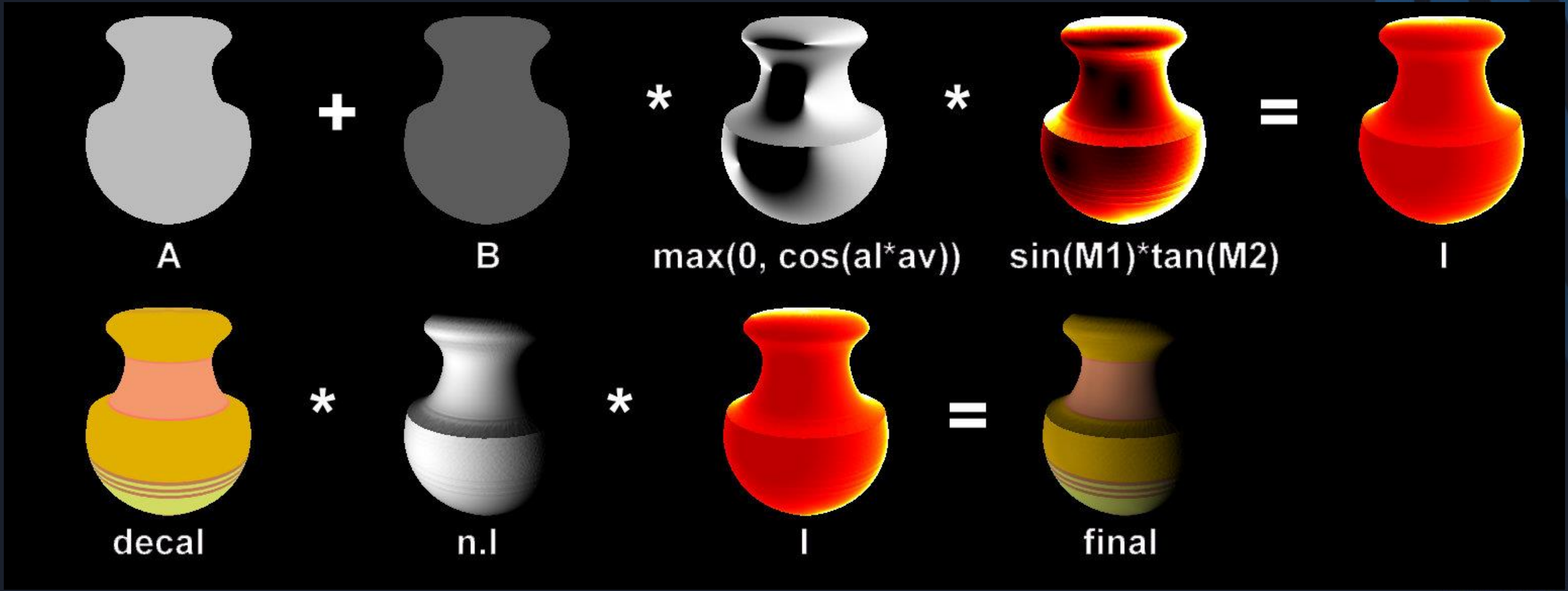
$$\alpha = \max(\theta_i, \theta_r) = \max(\mathrm{acos}(N \cdot E), \mathrm{acos}(N \cdot L))$$

$$\beta = \min(\theta_i, \theta_r) = \min(\mathrm{acos}(N \cdot E), \mathrm{acos}(N \cdot L))$$

```
// DX = sin(alpha) * tan(beta)
float alpha = sin( max( acos( NdE ), acos( NdL ) ) );
float beta = tan( min( acos( NdE ), acos( NdL ) ) );
float DX = alpha * beta;
```

Oren-Nayar Diffuse Reflectance

- A rundown on the results of the model

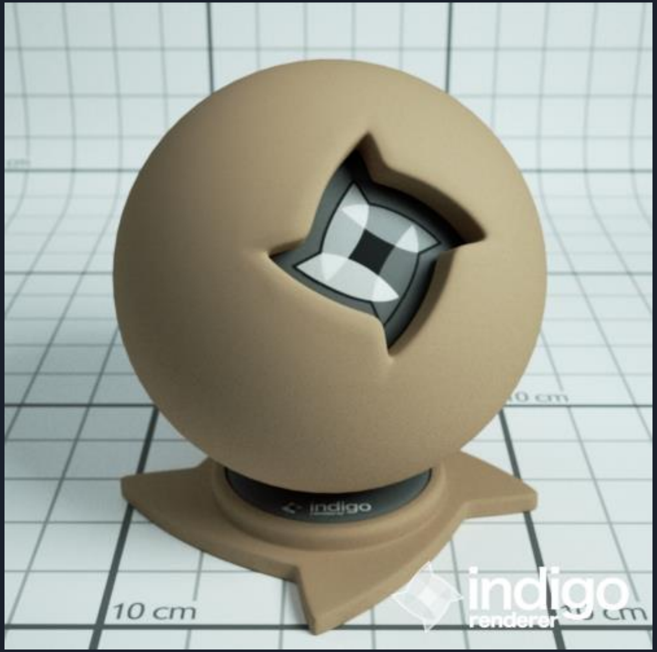


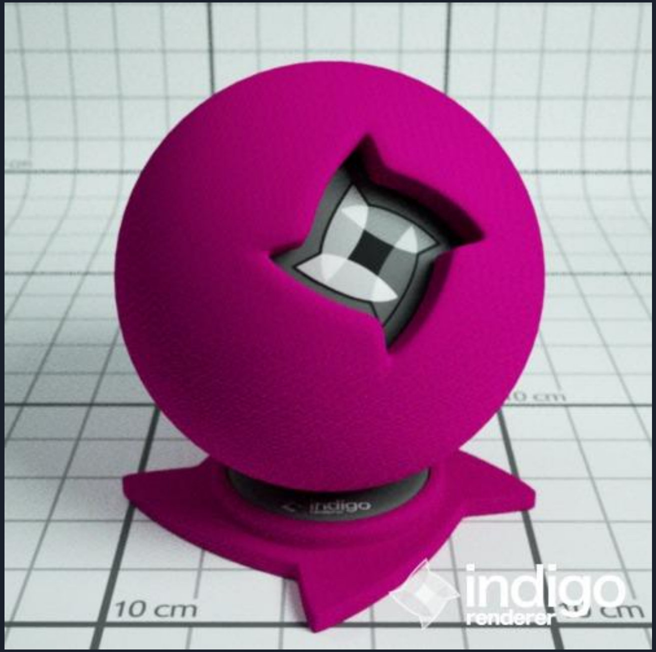
Oren-Nayar Diffuse Reflectance

```
float NdL = max( 0.0f, dot( N, L ) );
float NdE = max( 0.0f, dot( N, E ) );

float R2 = roughness * roughness;

// Oren-Nayar Diffuse Term
float A = 1.0f - 0.5f * R2 / (R2 + 0.33f);
float B = 0.45f * R2 / (R2 + 0.09f);

// CX = Max(0, cos(l,e))
vec3 lightProjected = normalize( L - N * NdL );
vec3 viewProjected = normalize( E - N * NdE);
float CX = max( 0.0f, dot( lightProjected, viewProjected ) );

// DX = sin(alpha) * tan(beta)
float alpha = sin( max( acos( NdE ), acos( NdL ) ) );
float beta = tan( min( acos( NdE ), acos( NdL ) ) );
float DX = alpha * beta;

// Calculate Oren-Nayar, replaces the Phong Lambertian Term
float OrenNayar = NdL * (A + B * CX * DX);
```

Oren-Nayar Diffuse Reflectance

- It might seem like a lot of work for very little gain, but when applied to materials with different roughness values it can be a drastic change





Cook-Torrance Specular Reflectance

- A good partner to Oren-Nayar Diffuse
- Published by Robert Cook and Kenneth Torrance
- Like Oren-Nayar, uses a roughness value to simulate microfacets
  – Calculates specular reflection rather than diffuse
- The model accounts for light wavelengths at varying angles and thus handles true colour shifts in specular highlights

Cook-Torrance Specular Reflectance

- The Cook-Torrance mathematical model is…

$$S_{cook-torrance} = \frac{DFG}{\pi(E \cdot N)(N \cdot L)}$$

- Where…

$$D = \frac{1}{m^2(N \cdot H)^4} e^{-\left(\frac{1-(N \cdot H)^2}{m^2(N \cdot H)^2}\right)}$$

$$F = R_0 + (1 - R_0)(1 - N \cdot E)^5$$

$$G = \min\left(1, \frac{2(H \cdot N)(E \cdot N)}{E \cdot H}, \frac{2(H \cdot N)(L \cdot N)}{E \cdot H}\right)$$

SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Cook-Torrance Specular Reflectance

- First, let's replace a few of those letters…
  - ***D*** is Beckmann's Distribution
  - ***F*** is a Fresnel Term
  - ***G*** is a Geometric Attenuation Factor
  - ***N*** is the surface normal
  - ***L*** is the light vector
  - ***E*** is a vector from the surface to the viewer
  - ***H*** is the vector average of the light vector L and view vector E

$$S_{cook-torrance} = \frac{DFG}{\pi(E \cdot N)(N \cdot L)}$$

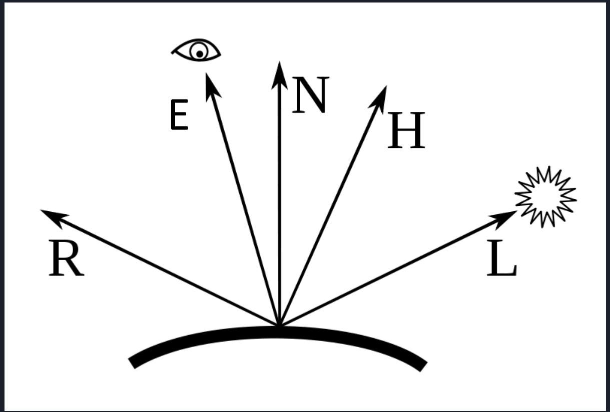
- With the bottom part we can usually ignore the ***N*** dot ***L*** as it is controlled by the diffuse portion of a full light equation
  - So the bottom part of Cook-Torrance is fairly simple, so lets focus on the top part…

Beckmann Distribution

- The D represents a distribution function
  - In this example, Beckmann Distribution
- Represents a function of surface roughness
  - Can use the same roughness value as Oren-Nayar!
  - ***m*** in the equation represents roughness
- Also in the equation is a H
  - This represents a half vector
  - It is equal to the average of the light vector and the view vector
  - Easily calculated by summing the two vectors then normalising

$$S_{cook-torrance} = \frac{\boxed{D}FG}{\pi(E \cdot N)(N \cdot L)}$$

$$D_{Beckmann} = \frac{1}{m^2(N \cdot H)^4} e^{-\left(\frac{1-(N \cdot H)^2}{m^2(N \cdot H)^2}\right)}$$



Beckmann Distribution – Shader Form

- Beckmann Distribution converted into shader form is much easier to follow…
  - Where m is the surface roughness

$$D = \frac{e^{\alpha}}{m^2 \times (N \cdot H)^2 \times (N \cdot H)^2}$$

$$\alpha = \frac{-(1 - (N \cdot H)^2)}{(N \cdot H)^2 \times m^2}$$

```
float R2 = roughness * roughness;
vec3 H = normalize( L + E ); // light and view half vector
float NdH = max( dot( N, H ), 0.0f );
float NdH2 = NdH * NdH;

float exponent = -(1 - NdH2) / (NdH2 * R2);
float D = pow( e, exponent ) / (R2 * NdH2 * NdH2);
```

Fresnel Term

- The F in the equation is a Fresnel Term
  - A method to describe the behaviour of light at certain angles with materials of different refractive properties

$$S_{cook-torrance} = \frac{D\boxed{F}G}{\pi(E \cdot N)(N \cdot L)}$$

- For performance we can use what's called Schlick's Approximation to approximate a Fresnel Term in our shader

$$F = R_0 + (1 - R_0)(1 - N \cdot E)^5$$

  - Schlick uses the term R_0 to represent the reflection coefficient that is typically calculated using the Reflection and Refraction properties of the material being lit
  - As we are typically lighting opaque materials we can reduce it to a scalar of our choosing
  - The following would be the shader-form of the equation, with reflectionCoefficient being a shader uniform using a float passed in by the user, or as a material property

```
float F = reflectionCoefficient + (1 - reflectionCoefficient) * pow( 1 - NdE, 5 );
```

Geometric Attenuation Factor

- The G in the equation represents a Geometric Attenuation Factor (GAF) that simulates shadowing caused by the microfacets in the surface

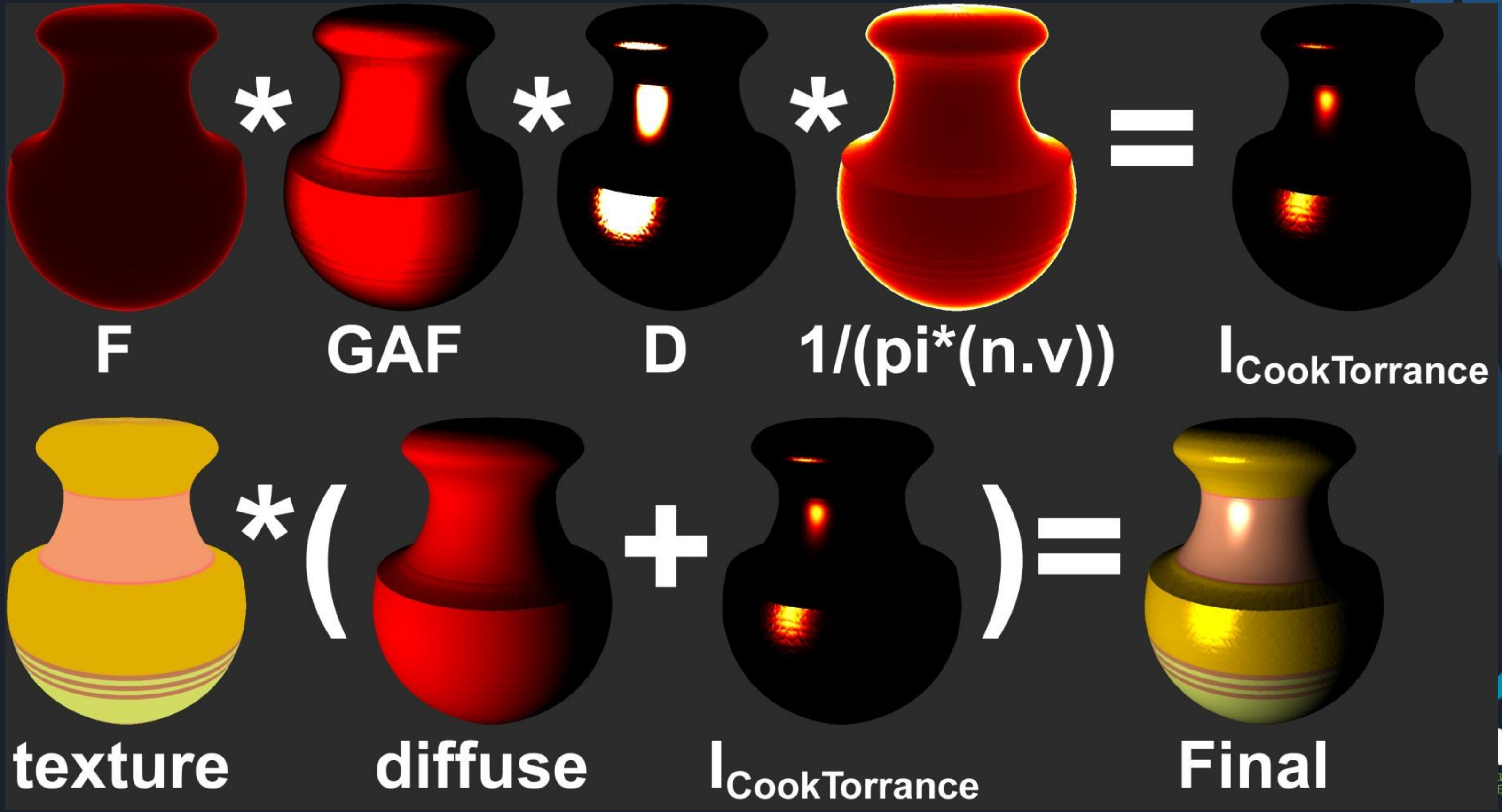
$$S_{cook-torrance} = \frac{DF\boxed{G}}{\pi(E \cdot N)(N \cdot L)}$$

- Its calculation is long but fairly easy to implement

$$G = \min(1, \frac{2(H \cdot N)(E \cdot N)}{E \cdot H}, \frac{2(H \cdot N)(L \cdot N)}{E \cdot H})$$

```
float X = 2.0f * NdH / dot( E, H );
float G = min(1, min(X * NdE, X * NdL));
```

Cook-Torrance Specular Reflectance



Cook-Torrance Specular Reflectance

```
float NdH = max( 0.0f, dot( N, H ) );
float NdH2 = NdH * NdH;
float e = 2.71828182845904523536028747135f;
float pi = 3.1415926535897932384626433832f;

// Beckman's Distribution Function D
float exponent = -(1 - NdH2) / (NdH2 * R2);
float D = pow( e, exponent ) / (R2 * NdH2 * NdH2);

// Fresnel Term F
float F = reflectionCoefficient + (1 – reflectionCoefficient) * pow( 1 - NdE, 5 );

// Geometric Attenuation Factor G
float X = 2.0f * NdH / dot( E, H );
float G = min(1, min(X * NdL, X * NdE));

// Calculate Cook-Torrance
float CookTorrance = max( (D*G*F) / (NdE * pi), 0.0f );
```

Oren-Nayar + Cook-Torrance

- Combining both models for Physically-Based Diffuse and Specular provides great results

All That For What?

- So after all that math, N's, L's and dot products, what are the main benefits?

- More realistic visuals!

- A single shader to represent all surfaces
  - One of the bottlenecks in graphics programming is changing the state of the GPU
  - Swapping shaders, changing textures, etc

- Having a single shader that can accurately represent metal, plastic, skin etc can increase render workload by decreasing the different shaders needed!

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Summary

- Bidirectional Reflectance Distribution Functions are functions that attempt to calculate light reflecting off surfaces
- Physically-Based Lighting better simulates real-life lighting
- Modern GPU hardware can handle the complex equations needed
- Some current-gen games have already implemented Physically-Based Lighting models

Further Reading

- Allegorithmic, *The Comprehensive PBR Guide*, www.allegorithmic.com
  - https://www.allegorithmic.com/pbr-guide

- Akenine-Möller, T, Haines, E, 2008, *Real-Time Rendering*, 3rd Edition, A.K. Peters

- Humphreys, G, Pharr, M, 2010, *Physically Based Rendering*, 2nd Edition, Morgan Kaufman