Rendering
Basic Shading

CS 415: Game Development

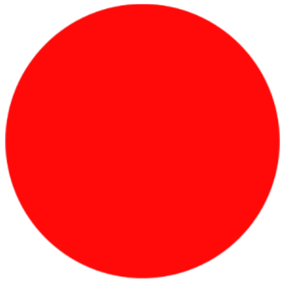
Professor Eric Shaffer

ILLINOIS

Shading

Shading refers to the process of determining the color for a pixel

Shading is one of the key elements of 3D photorealistic rendering…or any kind rendering



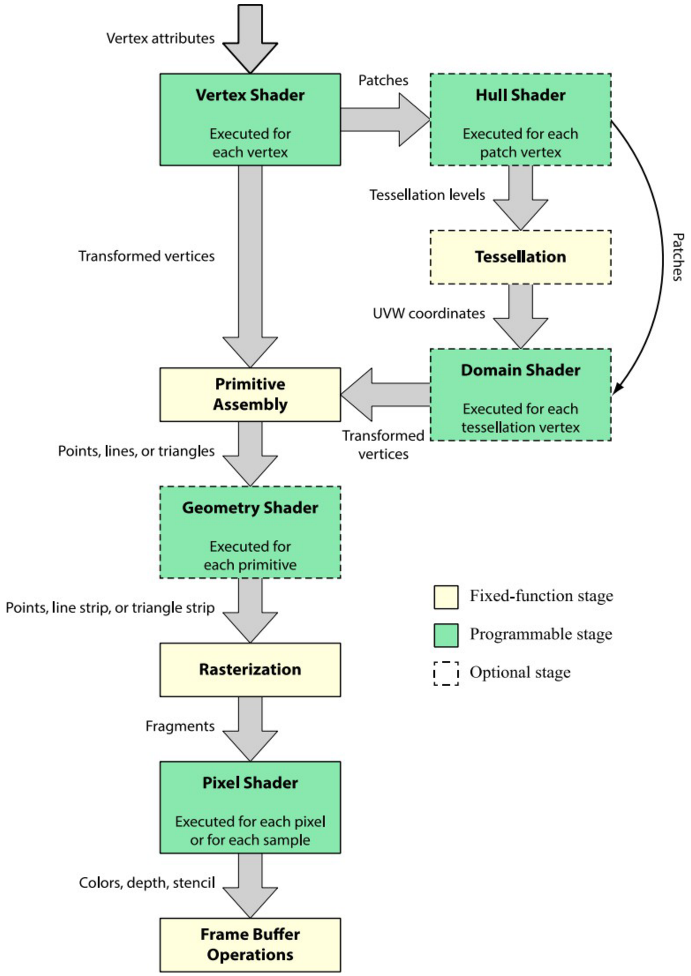
Flat shading

- Same color over entire surface of a sphere
- Looks 2D

Shading with Phong reflection model

- Varies color with geometry and light position
- Looks 3D

What Are Programmable Shaders?



Modern GPUs introduced the idea of shader programs

The word *shader* has come to mean any of a number of programmable GPU Units

Architecture on left is just one example

Not all engines make use of all these shader types Not all

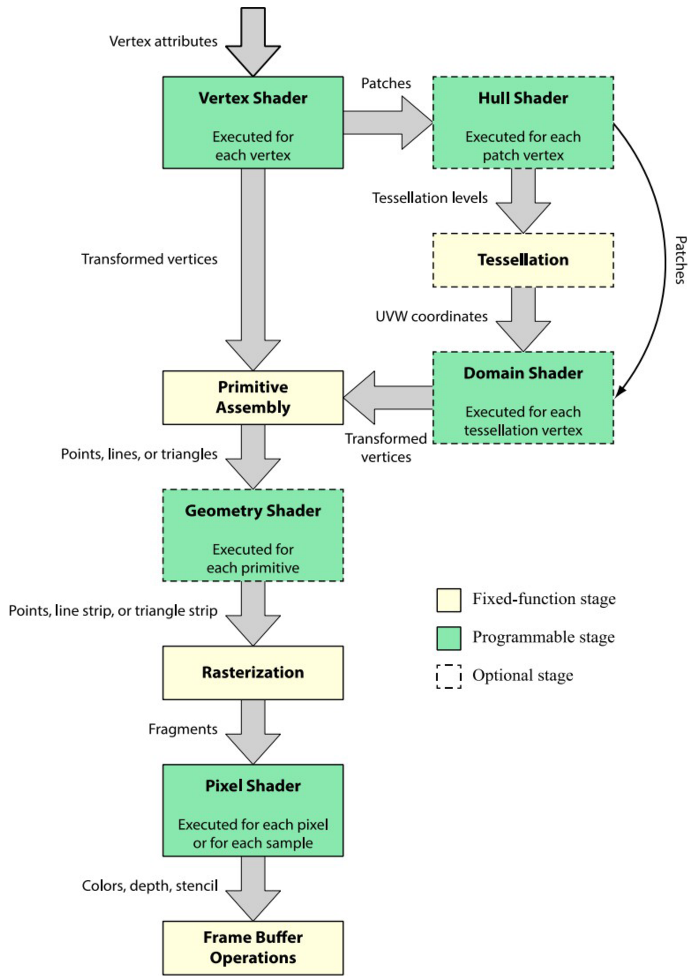
rendering APIs support all these shaders

There are other shader types as well

- Tessellation Shader
- Mesh Shaders
- Ray Tracing Shaders
- Compute Shaders

Since shaders are programmable, used for purposes beyond their initial design

Two Most Important Programmable Shaders



Vertex Shader

- Each vertex in the scene geometry gets processed by a vertex shader program
- Usually just geometric transformations related to animation and projection, etc.
- Program is identical running across many cores of GPU vertices stream through

Pixel Shader (or Fragment Shader)

- Runs for each pixel produced by rasterization
- Usually just generates a color for that pixel ← ***this is actual shading***
- Program is identical running across many cores of GPU vertices stream through

The word **fragment** is used to describe a pixel that is produced during rasterization but isn't necessarily the final pixel value you will see on screen (maybe hidden by a surface in front of it, etc.).

Modeling Light Reflection: The Simplest Shader

Most of the time shading means modeling the reflection of light off a surface

What the simplest mathematical model?

Represent the amount of incoming light and the amount of that light reflected by the surface

- (r_i, g_i, b_i) is the color of the incoming light with each channel in the range [0,1]
- (r_k, g_k, b_k) is the reflectance of the material with each channel in the range [0,1]

1 indicates total absorption
2 indicates total reflectance

The shader would then compute a component-wise product:

$$(r_i, g_i, b_i) \times (r_k, g_k, b_k) = (r_i r_k, g_i g_k, b_i b_k)$$

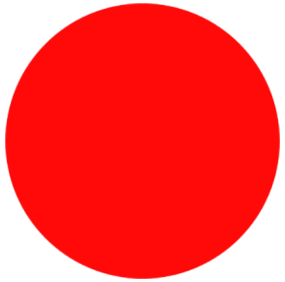
Modeling Light Reflection: The Simplest Shader

Represent the amount of incoming light and the amount of that light reflected by the surface

- (r_i, g_i, b_i) is the color of the incoming light with each channel in the range [0,1]
- (r_k, g_k, b_k) is the reflectance of the material with each channel in the range [0,1]

The shader would then compute a component-wise product:

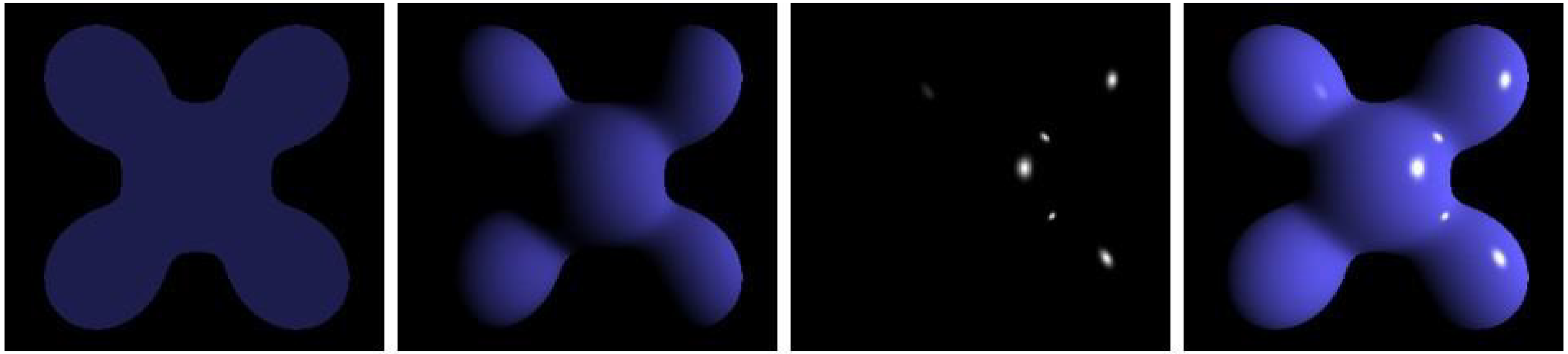
$$(r_i, g_i, b_i) \times (r_k, g_k, b_k) = (r_i r_k, g_i g_k, b_i b_k)$$



Flat shading

- Same color over entire surface of a sphere
- Looks 2D

The Second Simplest Shader: Phong Reflection



Ambient + Diffuse + Specular = Phong Reflection

$$I_\text{p} = k_\text{a} i_\text{a} + \sum_{m \in \text{lights}} \left(k_\text{d} (\hat{L}_m \cdot \hat{N}) i_{m,\text{d}} + k_\text{s} (\hat{R}_m \cdot \hat{V})^\alpha i_{m,\text{s}} \right)$$

Well…there's a lot of symbols but the ideas and math are simple

Bui Tuong Phong

- December 14, 1942 – July 1975
- Born in Hanoi
- Earned his PhD in 2 years at the University of Utah (1973)
  - Worked with Professor Ivan Sutherland
  - Dissertation work was the Phong reflectance model
  - Also produced model and realistic image of a VW bug

Graphics and Image Processing — W. Newman, Editor

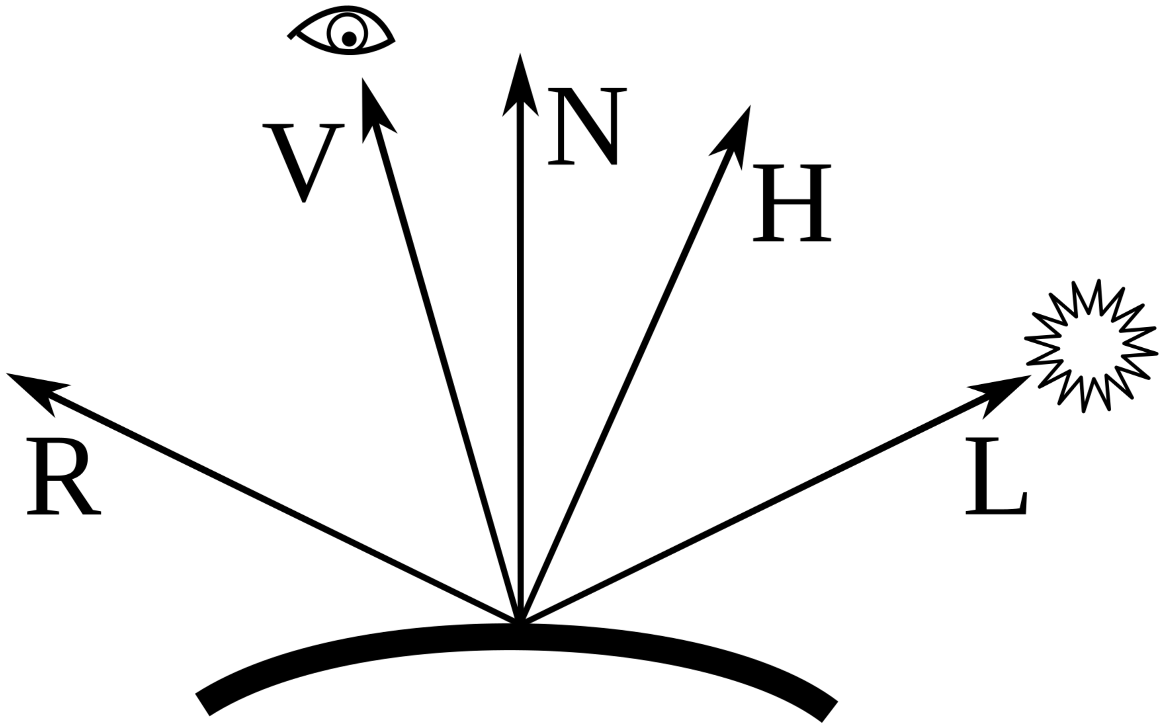
Illumination for Computer Generated Pictures

Bui Tuong Phong
University of Utah

Fig. 9. Improved shading, applied to the example of Figure 2.

ILLINOIS

The Second Simplest Shader: Phong Reflection

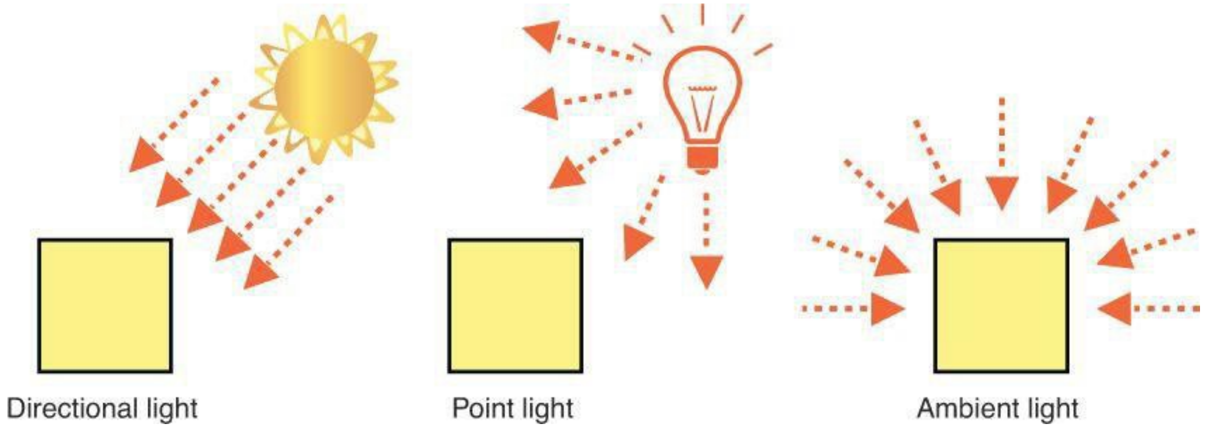


$$I_{\mathrm{p}} = k_{\mathrm{a}} i_{\mathrm{a}} + \sum_{m \in \mathrm{lights}} \left(k_{\mathrm{d}} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_{\mathrm{s}} (\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,\mathrm{s}} \right)$$

ILLINOIS

Simple Light Source Models

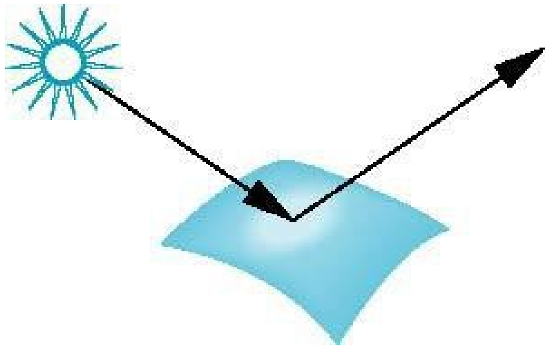
- Point source
  - Model with position and color
- Directional source
  - Distant source = infinite distance away (parallel)
- Ambient light
  - Same amount of light everywhere in scene
  - Models indirect light from reflecting surfaces



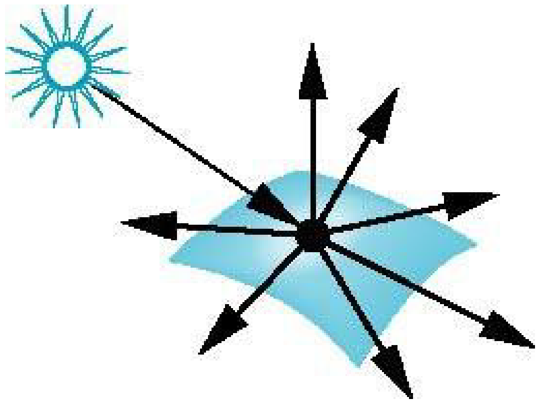
ILLINOIS

Surface Types

- Consider light traveling along a specific ray
- The smoother a surface, the more reflected light is concentrated in a single direction
  - Perfect mirror reflects perfectly in a single direction
- A very rough surface scatters light in all directions

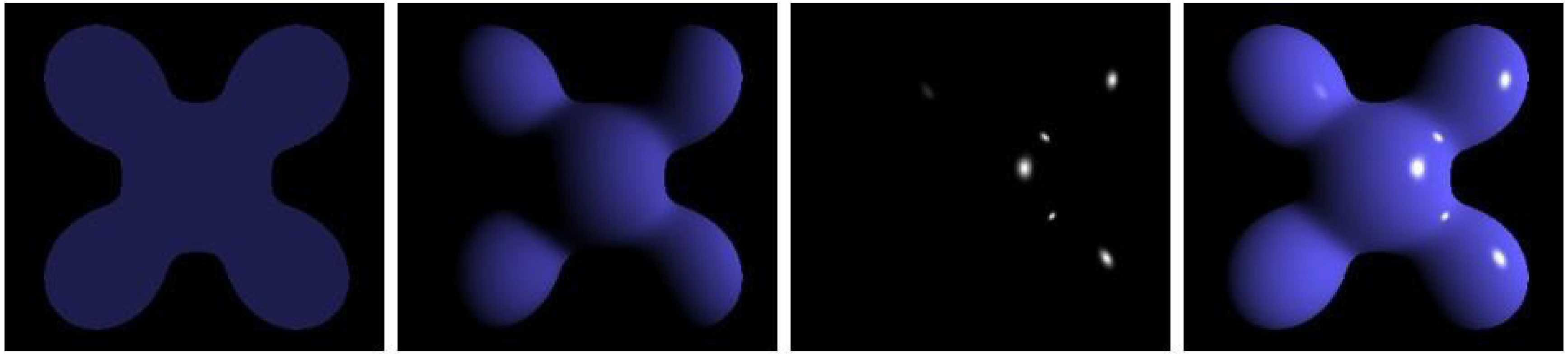


smooth surface = specular



rough surface = diffuse

The Phong Reflection Model: Term by Term



Ambient + Diffuse + Specular = Phong Reflection

$$I_{\mathrm{p}} = k_{\mathrm{a}} i_{\mathrm{a}} + \sum_{m \in \text{lights}} \left(k_{\mathrm{d}} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_{\mathrm{s}} (\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,\mathrm{s}} \right)$$

Ambient Light

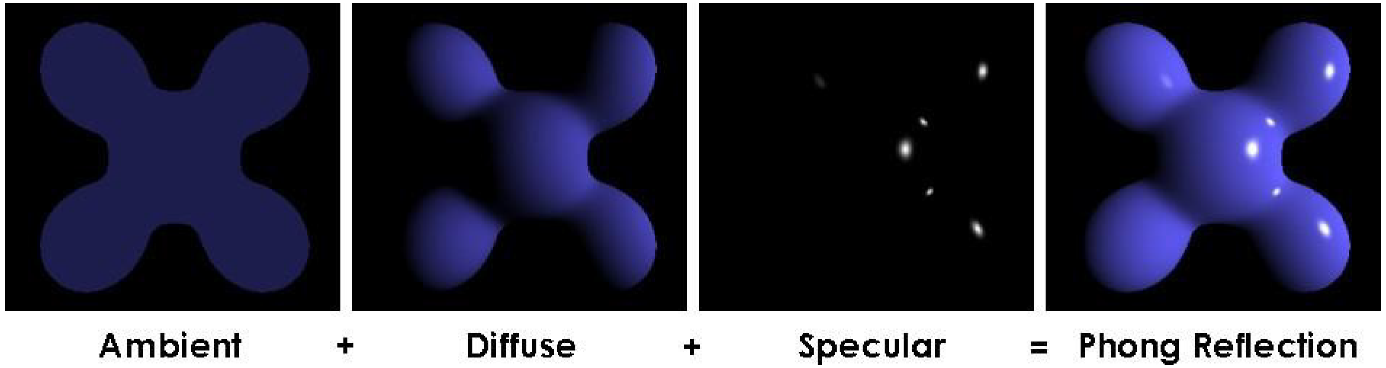
Result of multiple interactions between light sources and surfaces

- Kind of a hack meant to account for indirect light

Add k_a I_a to diffuse and specular terms

reflection intensity of ambient light

Remember that ki multiplications are component-wise multiplications of rgb values
$(k_r, k_g, k_b)(i_r, i_g, i_b) = (k_r i_r, k_g i_g, k_b i_b)$

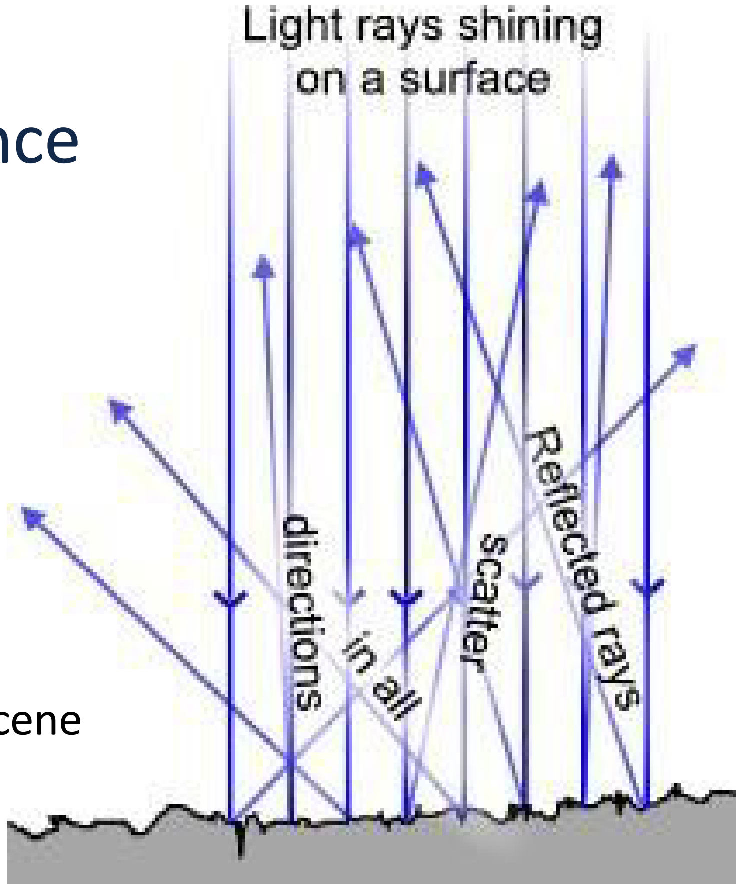


ILLINOIS

Modeling a Lambertian Surface – Diffuse Reflection

- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is affected by the angle of incidence
  - reflected light proportional to ***cosine of angle between L and N***
  - if vectors normalized

$$\cos(\theta) = \mathrm{L} \cdot \mathrm{N}$$



$$k_{\mathrm{d}}(\hat{L}_m \cdot \hat{N})i_{m,\mathrm{d}}$$

The diffuse term in Phong reflection

k_d is the diffuse reflectance of the surface
$i_{m,d}$ is the diffuse color of m of n lights in the scene

ILLINOIS

Diffuse Reflection Example

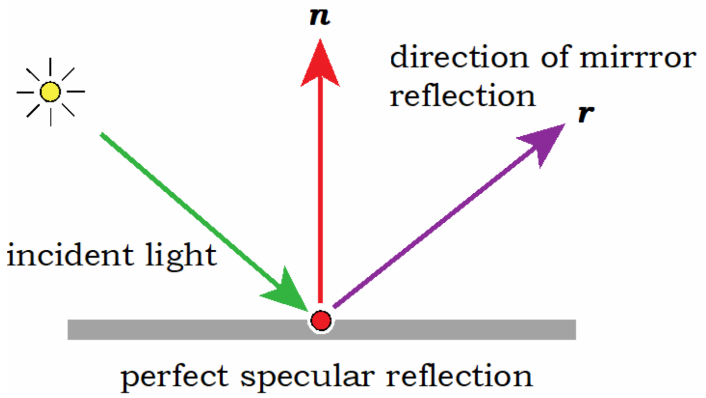
On the left, a barrel is illuminated by ambient light only. On the right, the barrel is illuminated by ambient light and one direct light source, the direction to which increases by 30 degrees to the right in each image. Lambert’s cosine law causes the shading to darken as the angle between the surface normal and the direction to the light increases.

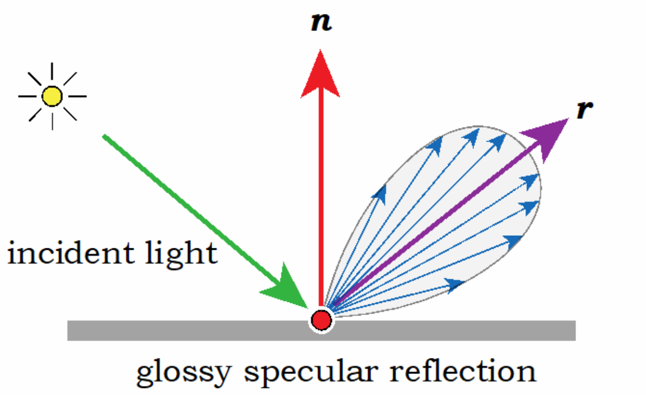
Lengyel, Eric. Foundations of Game Engine Development, Volume 2: Rendering (p. 115).

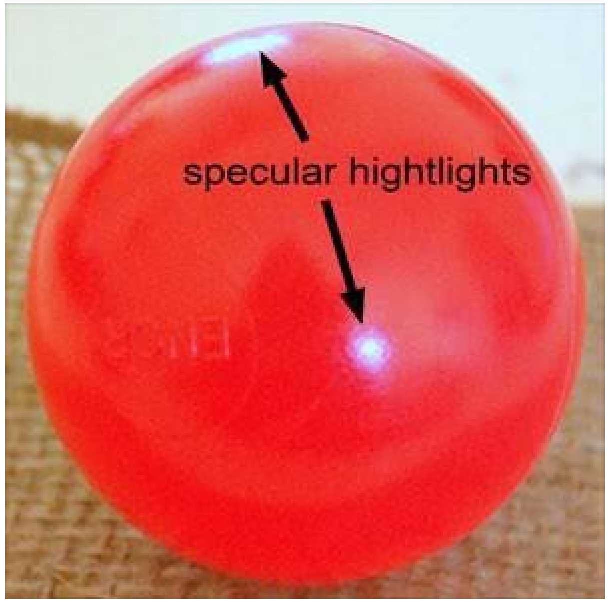
ILLINOIS

Specular Reflection

- Perfect specular reflection
  - Light is reflected in the single direction r
  - …the mirror reflection direction
- Glossy specular reflection
  - Scattering clustered around mirror reflection direction







ILLINOIS

Specular Reflection

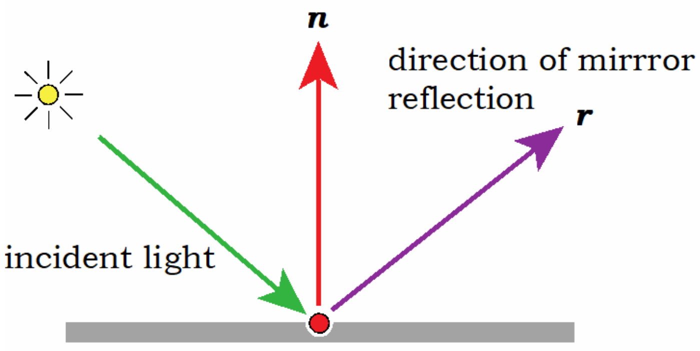
$$k_s(\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,s}$$

The specular term in Phong reflection

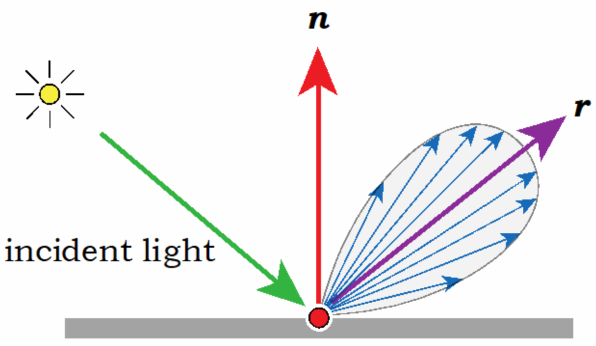
k_s is the specular reflectance of the surface
$i_{m,s}$ is the specular color of light m
R_m is the reflection vector for light m
V is the view vector from pixel to camera
α is the shininess coefficient

Specular reflection will be most powerful in the model when the view and reflection directions are closely aligned

$$(R_m \cdot V) = \cos(\phi)$$



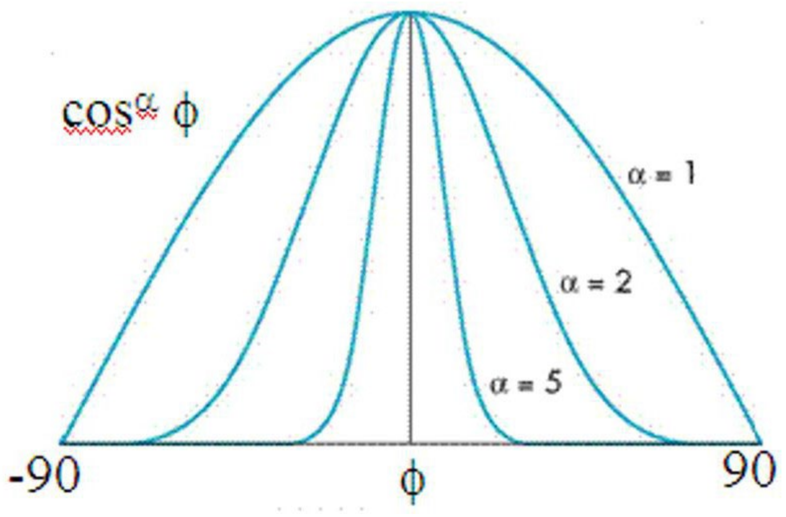
perfect specular reflection



glossy specular reflection

Specular Reflection

$$k_s(\hat{R}_m \cdot \hat{V})^\alpha i_{m,s}$$



Reflectance determined by

- Alignment of view vector with mirror reflection vector
- Shininess coefficient

High coefficient means smoother look

- Maybe 100 for metal
- Maybe 10 for plastic

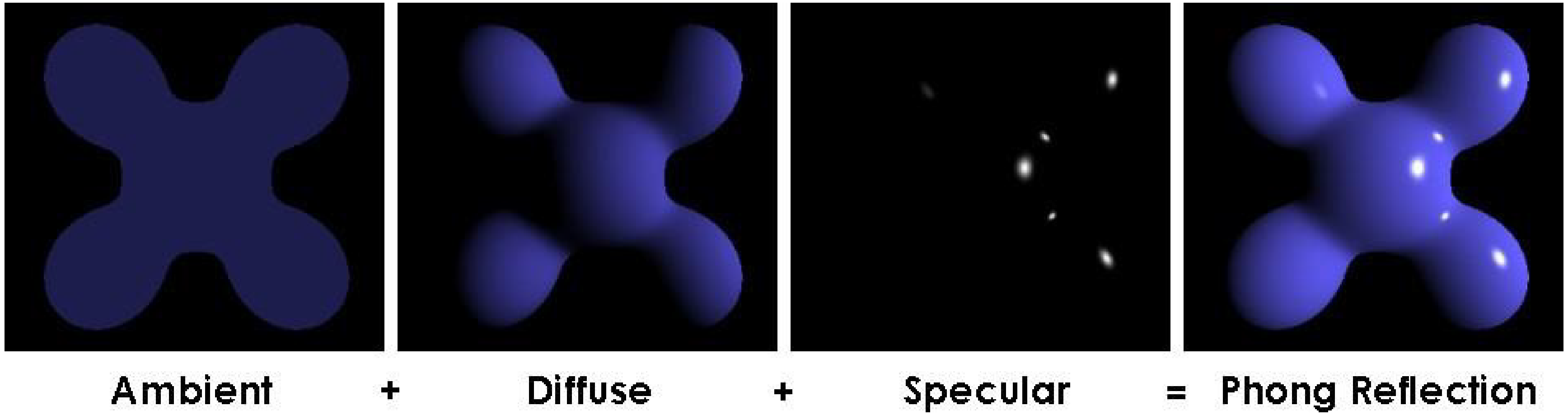
Specular Reflection Example

On the left, a sci-fi weapon is rendered with diffuse shading only. On the right, specular shading is added with specular powers of 10, 50, and 200 from left to right.

Lengyel, Eric. Foundations of Game Engine Development, Volume 2: Rendering (p. 117).

ILLINOIS

Phong Reflection Model: Computed on Each Pixel



$$I_p = k_a i_a + \sum_{m \in \text{lights}} (k_d (\hat{L}_m \cdot \hat{N}) i_{m,d} + k_s (\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,s})$$

What data do we need at each pixel and how does it get there?

What Is Shader Code?

There are a lot of shader languages

- OpenGL GLSL and Vulkan GLSL
- HLSL
- NVIDIA Slang (uses AI!)

For Unreal Shader Programs:

1. Start out as HLSL

2. …go through a bunch of translation…

3. …some as a pre-process…some at run-time…

4. Eventually gets loaded on to the GPU

D: > test > test.slang > test

```
struct ShadingPoint : IDifferentiable
{
    float3 L, V, N;
}

[Differentiable]
float3 evalBRDF(int materialID, ShadingPoint sp)
{
    /*...*/
}

void test()
{
    bwd_diff(evalBRDF)(0, )
}
```

```
func bwd_diff(evalBRDF)(materialID: int, sp: InOut<DifferentialPair<ShadingPoint>>, resultGradient: float3) -> void
```

Defined in d:\test\test.slang(7)

ILLINOIS

GLSL Example

```
#version 330

// data from vertex shader
in vec3 o_normal;
in vec3 o_toLight;
in vec3 o_toCamera;
in vec2 o_texcoords;

// color for framebuffer
out vec4 resultingColor;

// parameters of the light and possible values
uniform vec3 u_lightAmbientIntensitys;  // = vec3(0.6, 0.3, 0);
uniform vec3 u_lightDiffuseIntensitys;  // = vec3(1, 0.5, 0);
uniform vec3 u_lightSpecularIntensitys;  // = vec3(0, 1, 0);

// parameters of the material and possible values
uniform vec3 u_matAmbientReflectances;  // = vec3(1, 1, 1);
uniform vec3 u_matDiffuseReflectances;  // = vec3(1, 1, 1);
uniform vec3 u_matSpecularReflectances;  // = vec3(1, 1, 1);
uniform float u_matShininess;  // = 64;
```

GLSL Example

```
// returns intensity of reflected ambient lighting
vec3 ambientLighting()
{
   return u_matAmbientReflectance * u_lightAmbientIntensity;
}

// returns intensity of diffuse reflection
vec3 diffuseLighting(in vec3 N, in vec3 L)
{
   // calculation as for Lambertian reflection
   float diffuseTerm = clamp(dot(N, L), 0, 1) ;
   return u_matDiffuseReflectance * u_lightDiffuseIntensity *diffuseTerm;
}
```

GLSL Example

```
// returns intensity of specular reflection
vec3 specularLighting(in vec3 N, in vec3 L, in vec3 V)
{
   float specularTerm = 0;

   // calculate specular reflection only if
  // the surface is oriented to the light source
   if(dot(N, L) > 0)
   {
      // half vector
      vec3 H = normalize(L + V);
      specularTerm = pow(dot(N, H), u_matShininess);
   }
   return u_matSpecularReflectance * u_lightSpecularIntensity
* specularTerm;
}
```

This is actually the Blinn-Phong model which uses a half-vector approximation to the reflection vector…you can't trust that code you randomly grab from the internet will do what it's label says…we've been lied to

GLSL Example

```
void main(void)
{
    // normalize vectors after interpolation
    vec3 L = normalize(o_toLight);
    vec3 V = normalize(o_toCamera);
    vec3 N = normalize(o_normal);

    // get Phong reflectance components
    float Iamb = ambientLighting();
    float Idif = diffuseLighting(N, L);
    float Ispe = specularLighting(N, L, V);

    // diffuse color of the object from texture
    vec3 diffuseColor = texture(u_diffuseTexture,o_texcoords).rgb;

    // combination of all components and diffuse color of the object
    resultingColor.xyz = diffuseColor * (Iamb + Idif + Ispe);
    resultingColor.a = 1;
}
```

Pixel Shaders Require Input Data

Parameters need to get passed from the Vertex Shader to Pixel Shader

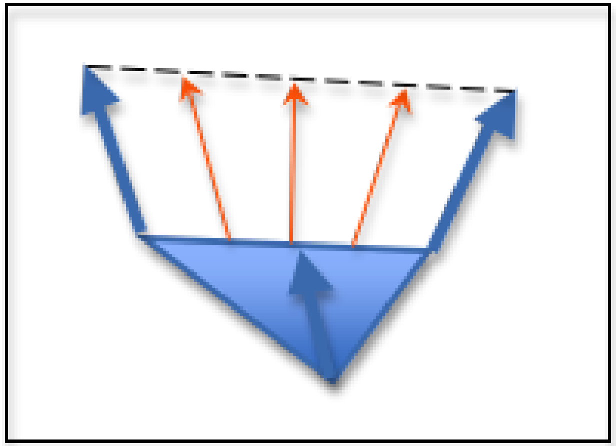
Each triangle has 3 vertices…but is covered by many pixels

How do we map vertex data to pixels?

One approach: pixel data is linearly interpolated from vertex data

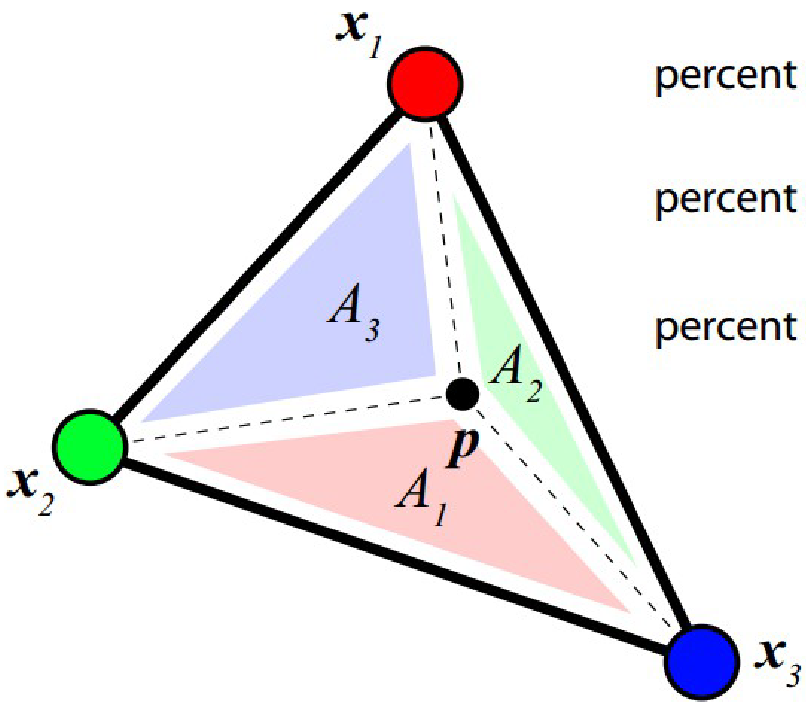
Each vertex has some attributes:

- Normal vector
- Position
- Texture coordinates



Compute a weighted average of each vertex attribute at each pixel

Barycentric Interpolation



percent red $= \frac{A_1}{A} = \lambda_1$

percent green $= \frac{A_2}{A} = \lambda_2$

percent blue $= \frac{A_3}{A} = \lambda_3$

"barycentric coordinates"

$\Sigma_i \lambda_i = 1$

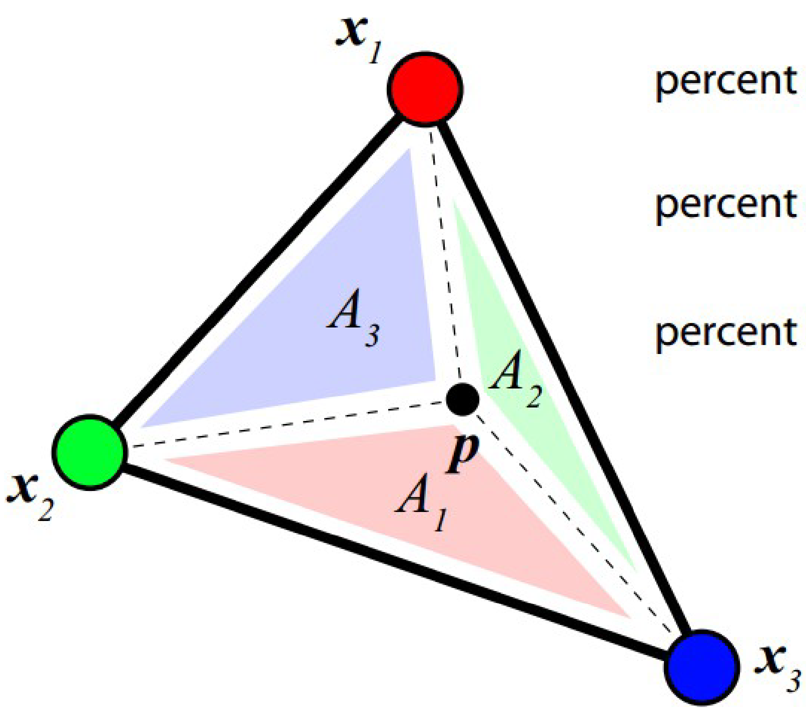
Value at $\boldsymbol{p}$:

$$(A_1\mathbf{x}_1 + A_2\mathbf{x}_2 + A_3\mathbf{x}_3)/A$$

Interpolated normal at point p would be $N_p = \lambda_1 N_1 + \lambda_2 N_2 + \lambda_3 N_3$

ILLINOIS

Barycentric Interpolation



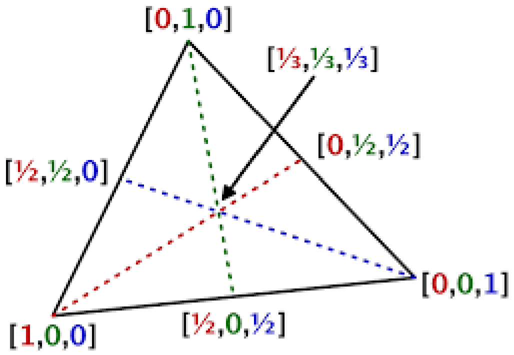
$$\text{percent red} = \frac{A_1}{A} = \lambda_1$$
$$\text{percent green} = \frac{A_2}{A} = \lambda_2$$
$$\text{percent blue} = \frac{A_3}{A} = \lambda_3$$

"barycentric coordinates"

$$\Sigma_i \lambda_i = 1$$

Value at $\boldsymbol{p}$:

$$(A_1\boldsymbol{x}_1 + A_2\boldsymbol{x}_2 + A_3\boldsymbol{x}_3)/A$$



Interpolated normal at point p would be $N_p = \lambda_1 N_1 + \lambda_2 N_2 + \lambda_3 N_3$

Shaders: Some Things to Know

Games use lots of shader programs (vertex, pixel, tessellation ray tracing, etc.)

Lots of surfaces can be modeled by a single reflectance model
- Can be implemented as single shader program.
- Different parameters generate different appearances

Some surfaces like cloth require lots of specialized shader programs

Unreal Engine in particular generates lots of shaders

- A UE5 *material* is a model of surface reflectance…

- You should think of materials as an input to a shader program
  But, when you create a Material with the UE5 Material Editor,
  it will usually create multiple HLSL Shader Programs

- No…I don’t know why

ILLINOIS

Shaders and Pre-Processing

Initial shader compilation during development can require a lot of time

Some things that can help shorten the dev time:

1. Always use the same computer
2. Set Shader Compiler Priority to Normal / Higher

   https://dev.epicgames.com/community/learning/tutorials/7B09/unreal-engine-speed-up-compiling-shaders

3. Maybe change the materials a bit to generate fewer shaders

   https://techarthub.com/seven-tricks-to-speed-up-shader-compilation-in-unreal-engine/

Shader Stutter…What Is It?

In many cases, shaders in PC games are compiled (final phase) at run time and loaded during the game

- Once the shader is compiled it will be retained (at least on SSD if not the GPU)
- Stuttering is a delay between frames. This results in laggy and annoying gameplay.
- There are many issues that can lead to stuttering
- Shader compilation is probably the most common in modern titles.
- If shaders are not pre-compiled, each time the player encounters a new object or effect, the game may stutter while the shaders are loading for that scene.
- Replaying the same level after all the shaders are compiled will usually remove the stutter.

Shader Stutter Example (Callisto Protocol)

You supposedly can see frame rate drops in Elden Ring too…so it's not just some game you never heard of…

ILLINOIS

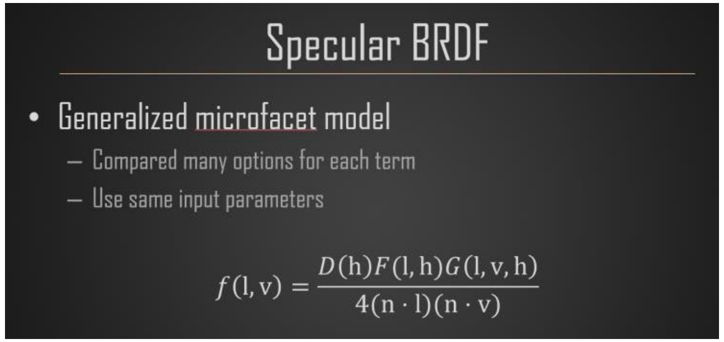
Physically Based Shading

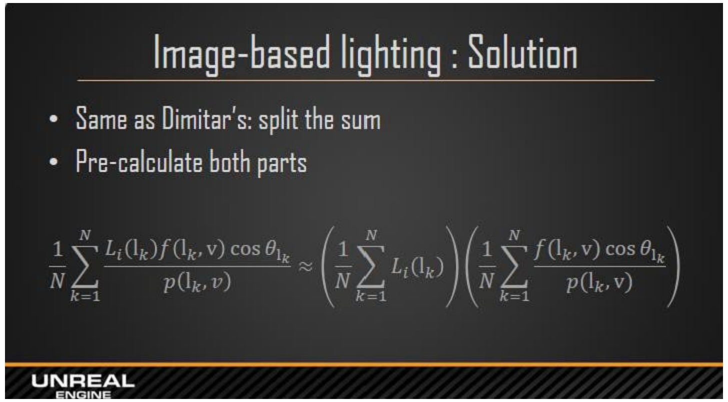
The Phong reflection model is non-physical…just sort of looks mostly right

- Does not accurately model physics of light reflection

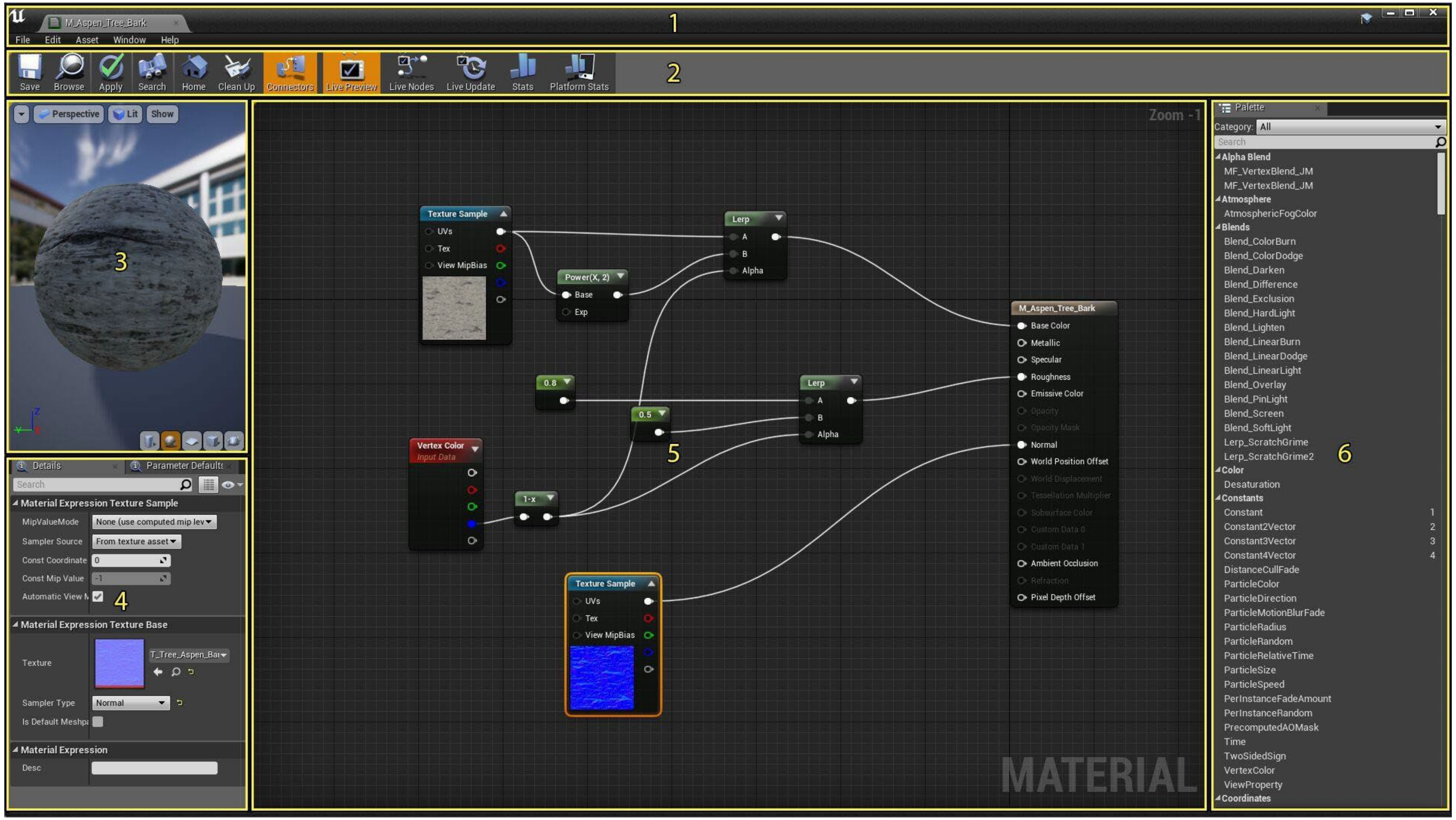
Modern shading has moved on to more physically based models

Much closer to photorealism





Unreal Engine Material Editor



Modern material models are more complex than the Phong model

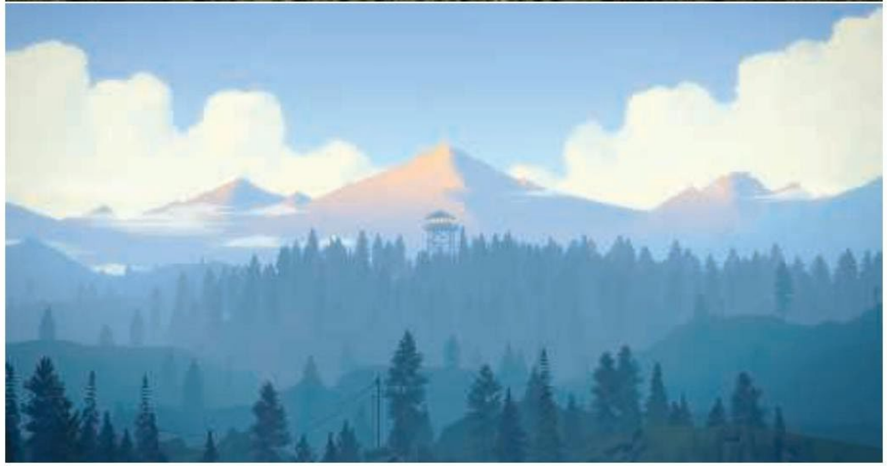
But…it's still just some not-too-complicated math to generate a color….

ILLINOIS

Power of Programmable Shaders

Can render scenes in a procedurally defined art style for a given game

Unreal Engine 5 rendering



Scene from *Firewatch*