Lecture 05

Rendering on Game Engine

Lighting, Materials and Shaders

WANG XI

GAMES 104

2022

Participants of Rendering Computation

- **Lighting**
  - Photon emit, bounce, absorb and perception is the origin of everything in rendering
- **Material**
  - How matter react to photon
- **Shader**
  - How to train and organize those micro-slaves to finish such a vast and dirty computation job between photon and materials

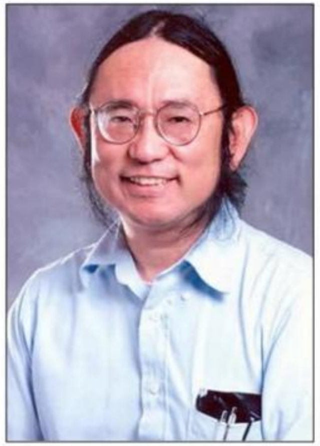
An interesting adventure story joined by smart graphics scientists and engineers based on evolution of hardware

The Rendering Equation

James Kajiya,"The Rendering Equation."

SIGGRAPH 1986.

Energy equilibrium:

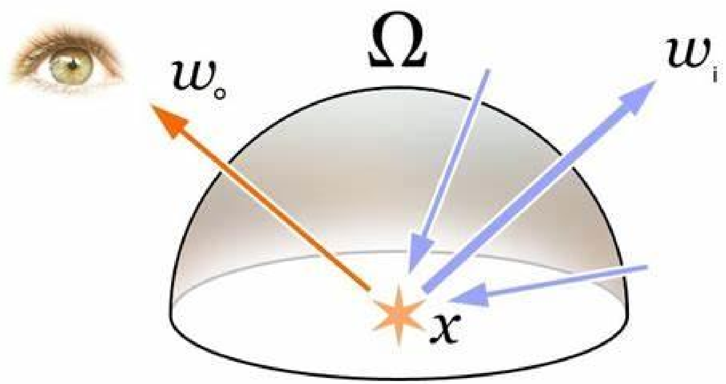


$$L_o(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_{H^2} f_r(\mathbf{x},\omega_o,\omega_i) L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i$$

outgoing emitted reflected

Radiance and Irradiance

The Rendering Equation



outgoing/observed radiance

emitted radiance (e.g., light source)

incoming radiance

angle between incoming direction and normal

$$L_o(\mathbf{x}.\omega_o) = L_e(\mathbf{x},\omega_o) + \int_{H^2} f_r(\mathbf{x},\omega_o,\omega_i) L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i$$

point of interest

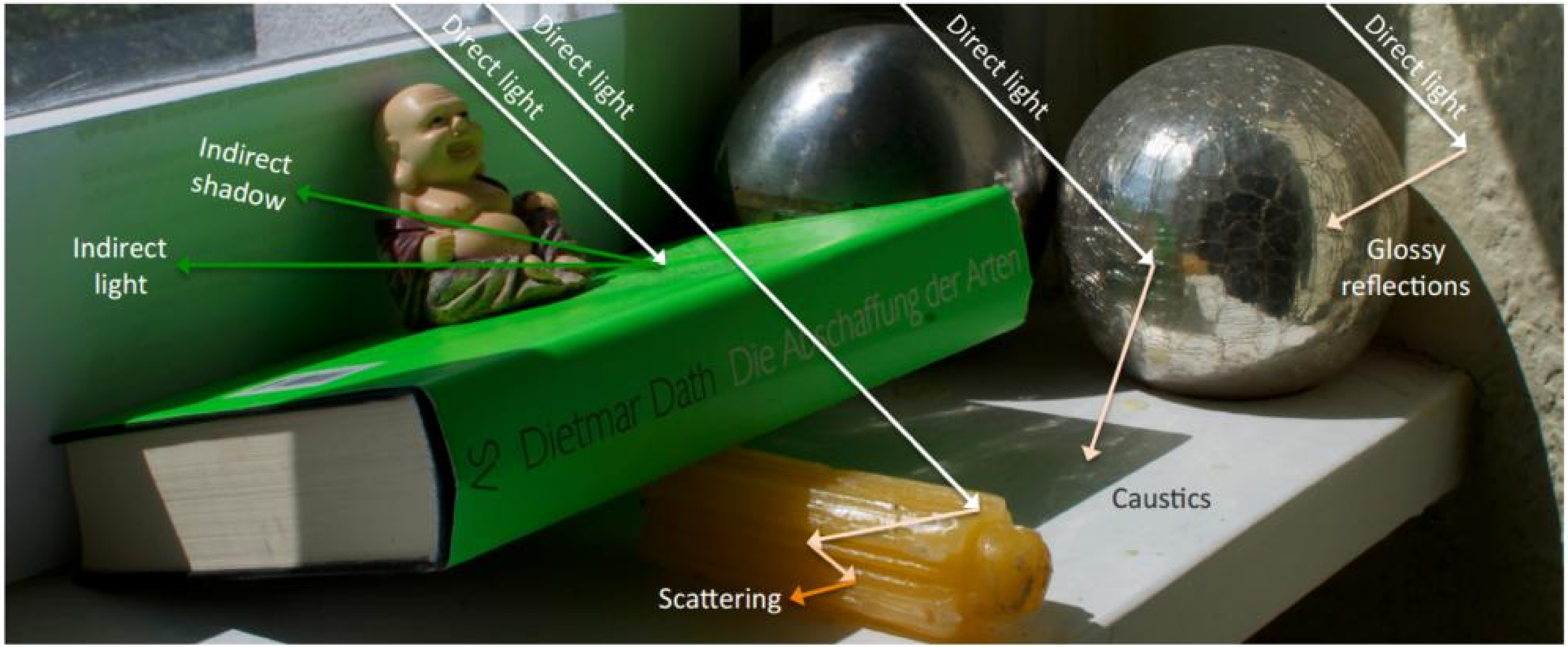
direction of interest

all directions in hemisphere

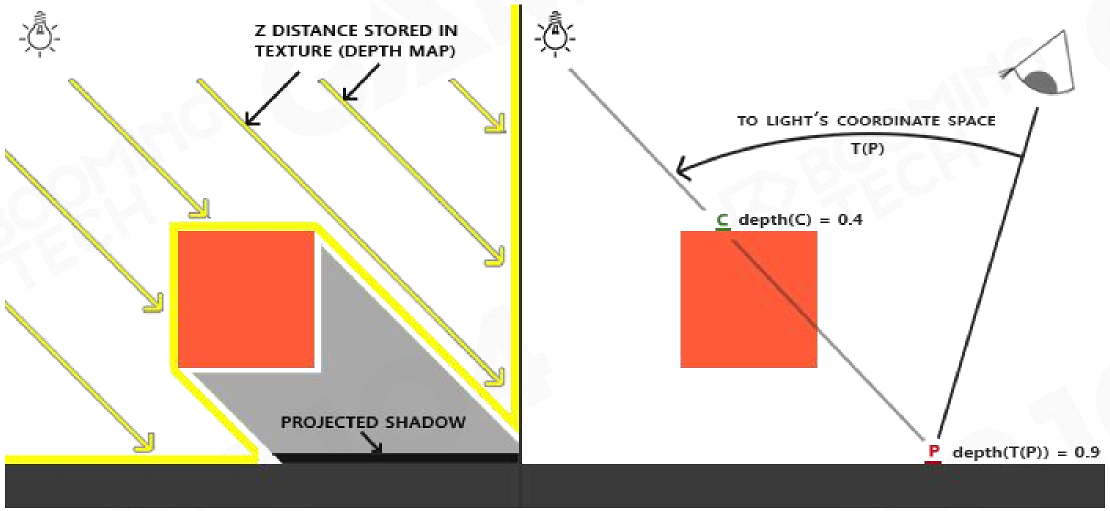
scattering function

incoming direction

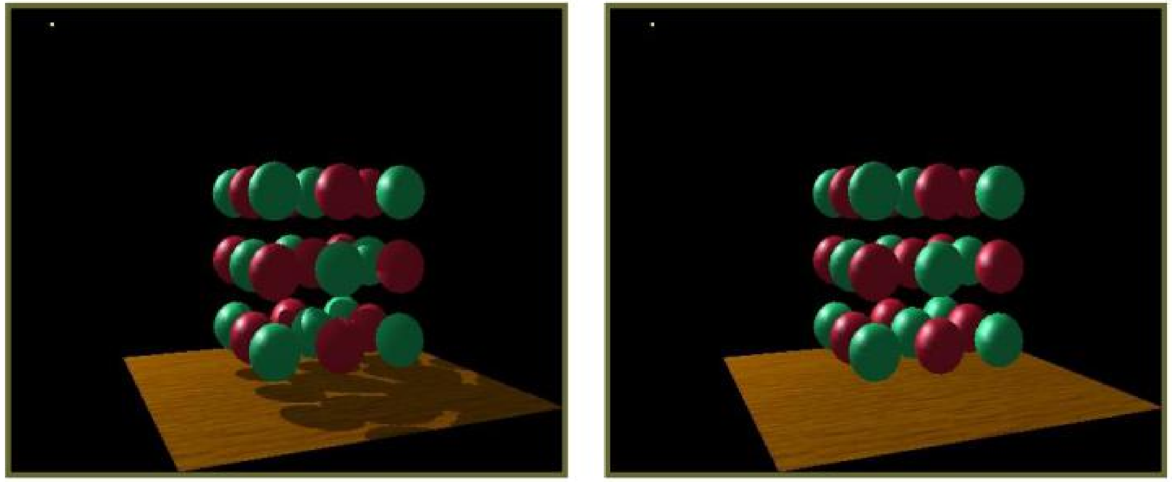
Complexity of Real Rendering



The 1st Challenge: 1a Visibility to Lights

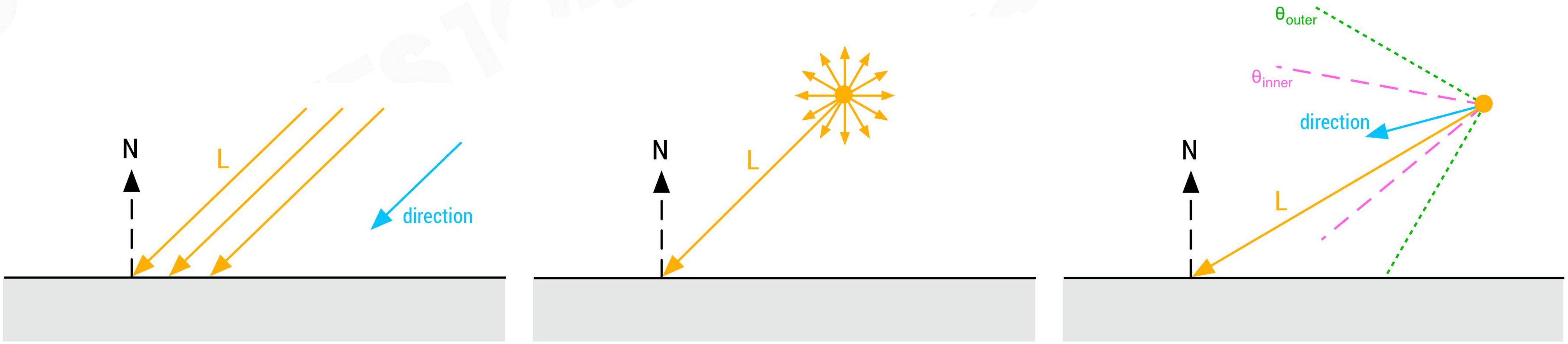


Ray Casting Toward Light Source



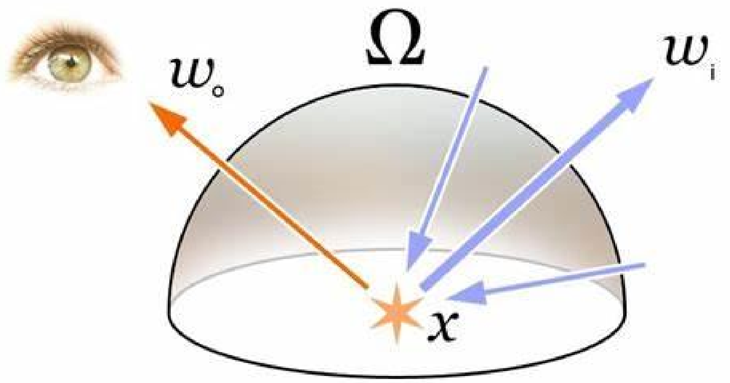
Shadow on and off

The 1st Challenge: 1b Light Source Complexity



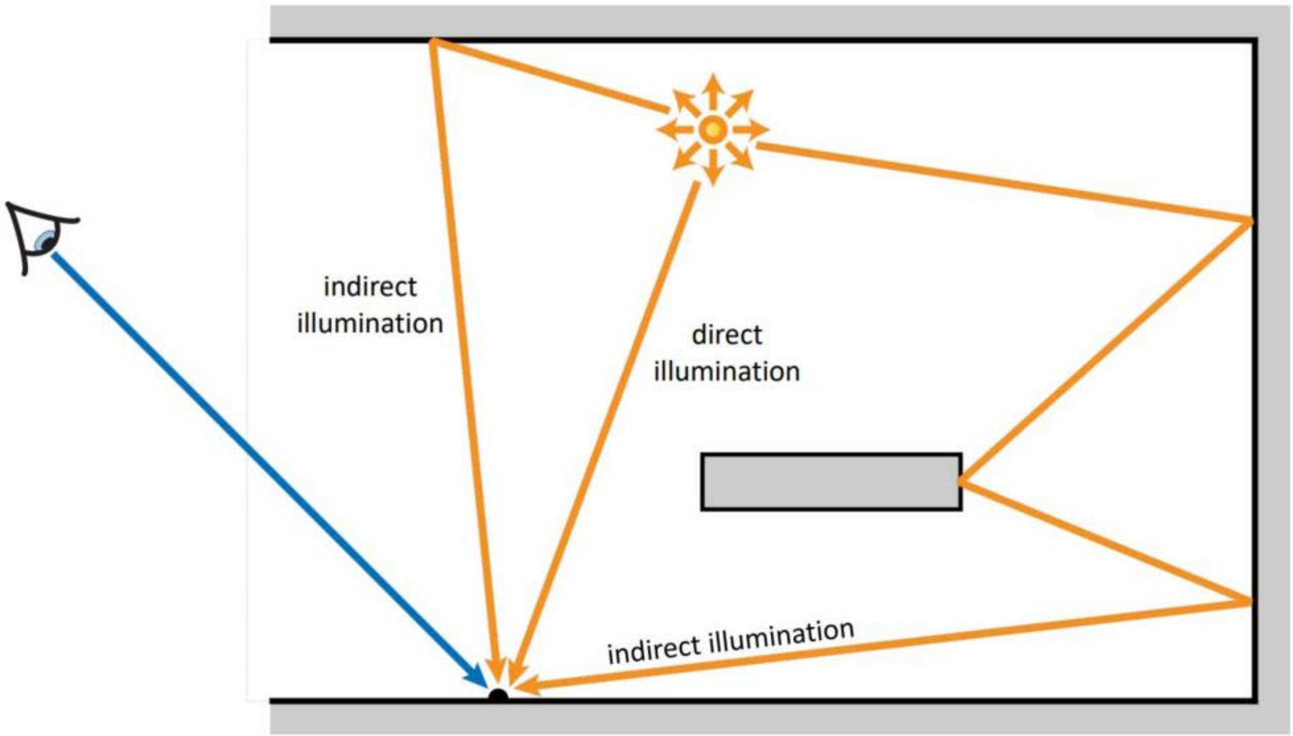
The 2nd Challenge: How to do Integral Efficiently on Hardware

- Brute-force way sampling
- Smarter sampling, i.e., Monta Carlo
- Derive fast analytical solutions
  - Simplify the f_r :
    - Assumptions the optical properties of materials
    - Mathematical representation of materials
  - Simplify the L_i :
    - Deal with directional light, point light and spot light only
    - A mathematical representation of incident light sampling on a hemisphere，for ex：IBL and SH

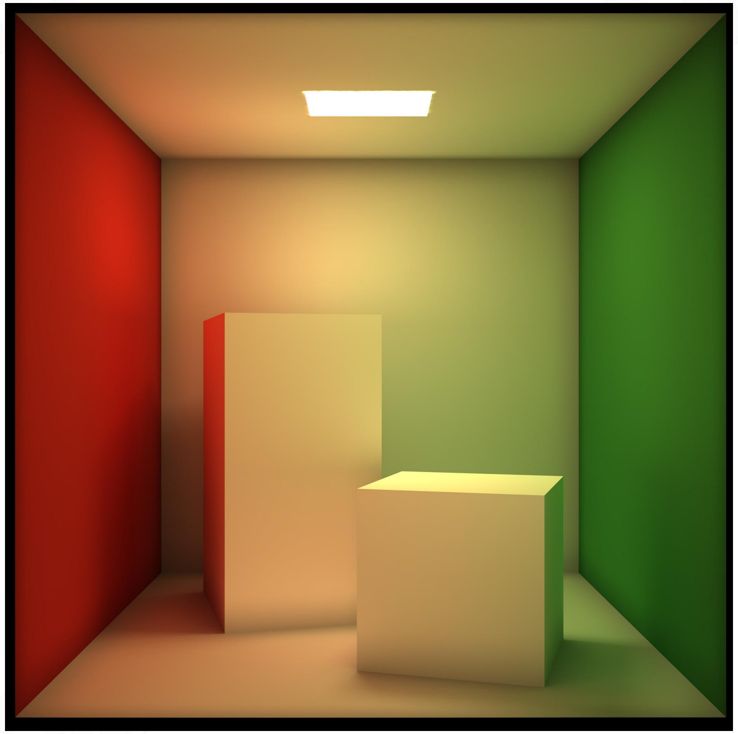


$$L_o(\mathbf{x},\omega_o) = \int_{H^2} \boxed{f_r(\mathbf{x},\omega_o,\omega_i)}\boxed{L_i(\mathbf{x},\omega_i)}\cos\theta_i d\omega_i$$

The 3rd Challenge: Any matter will be light source

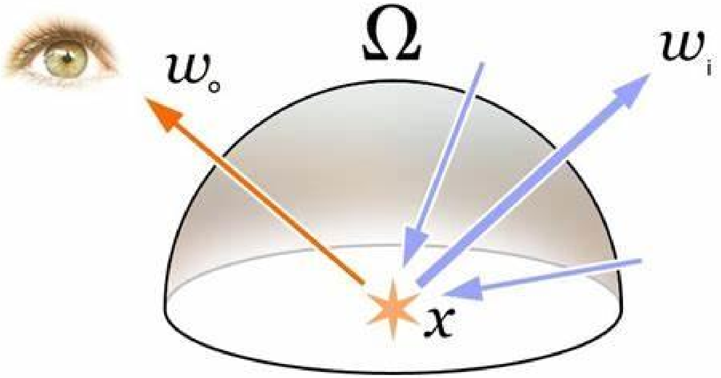


Direct vs. Indirect Illumination



Global Illumination (GI)

Three Main Challenges



$$L_o(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_{H^2} f_r(\mathbf{x},\omega_o,\omega_i) L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i$$

- The 1st Challenge: How to get incoming radiance for any given incoming direction
- The 2nd Challenge: Integral of lighting and scatting function on hemisphere is expensive
- The 3rd Challenge: To evaluate incoming radiance, we have to compute yet another integral, i.e. rendering equation is recursive

Starting from Simple

Forget some abstract concepts for a while, i.e. radiosity, microfacet and BRDF etc

Simple Light Solution

- Using simple light source as main light
    - Directional light in most cases
    - Point and spot light in special case
- Using ambient light to hack others
    - A constant to represent mean of complex hemisphere irradiance
- Supported in graphics API


```
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Environment Map Reflection

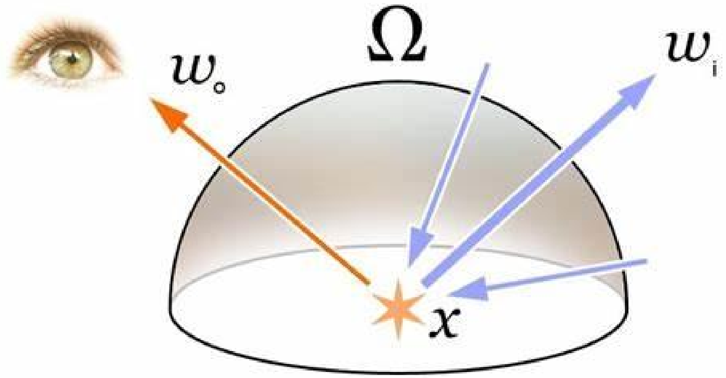
- Using environment map to enhance glossary surface reflection
- Using environment mipmap to represent roughness of surface

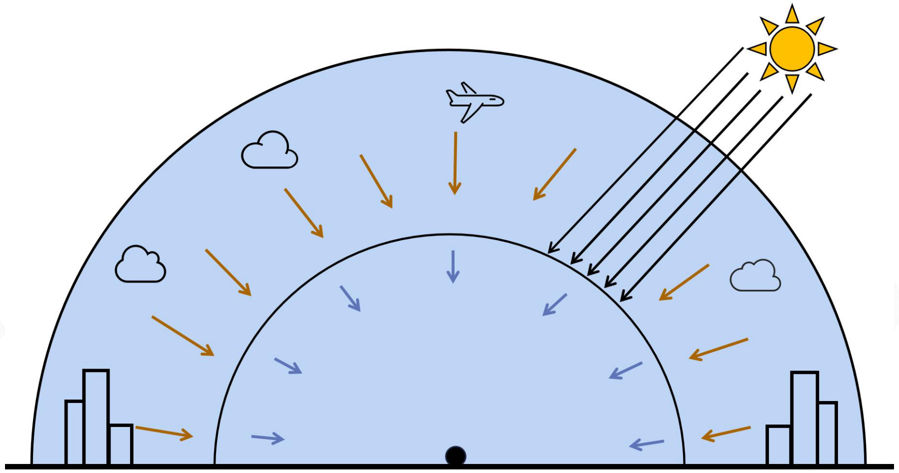
```
void main()
{
    vec3 N = normalize(normal);
    vec3 V = normalize(camera_position - world_position);

    vec3 R = reflect(V, N);

    FragColor = texture(cube_texture, R);
}
```

Early stage exploration of image-based lighting

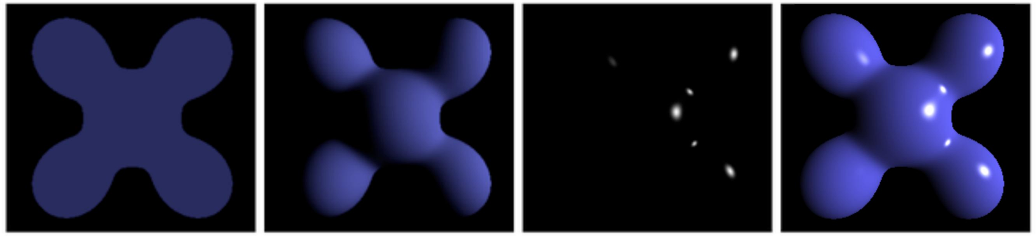
Math Behind Light Combo

- Main Light
  - Dominant Light
- Ambient Light
  - Low-frequency of irradiance sphere distribution
- Environment Map
  - High-frequency of irradiance sphere distribution



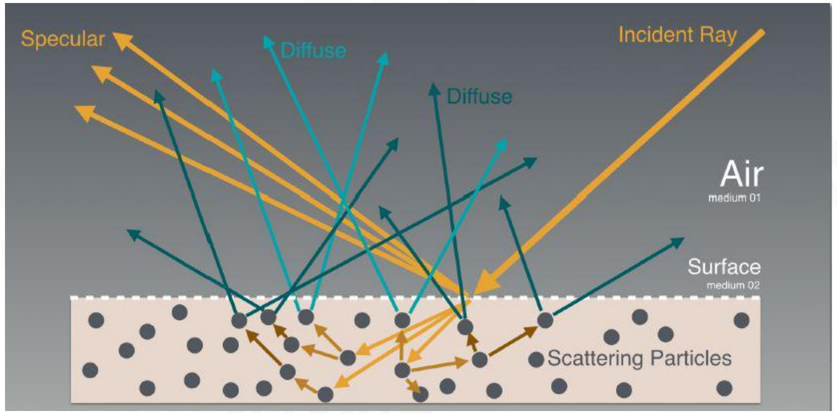


Blinn-Phong Materials



Ambient + Diffuse + Specular = Blinn-Phong Reflection

$$
\begin{aligned}
L &= L_a + L_d + L_s \\
&= k_a\, I_a + k_d\,(I/r^2)\max(0, \mathbf{n}\cdot\mathbf{l}) + k_s\,(I/r^2)\max(0, \mathbf{n}\cdot\mathbf{h})^p
\end{aligned}
$$

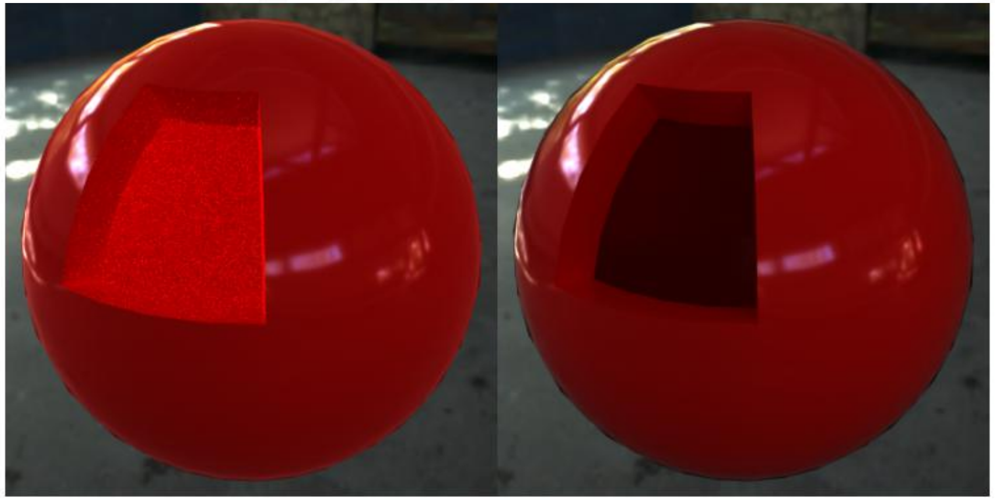



```
// set material ambient
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, Ka);
// set material diffuse
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, Kd);
// set material specular
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, Ks);
```

$$
L_o(\boldsymbol{x}, \omega_o) = \int_{H^2} \boxed{f_r(\boldsymbol{x}, \omega_o, \omega_i)} L_i(\boldsymbol{x}, \omega_i) \cos\theta_i \, d\omega_i
$$

Problem of Blinn-Phong

- Not energy conservative
  - Unstable in ray-tracing



Left non-energy conserving model lead a lot of noise compare Right energy conserving model

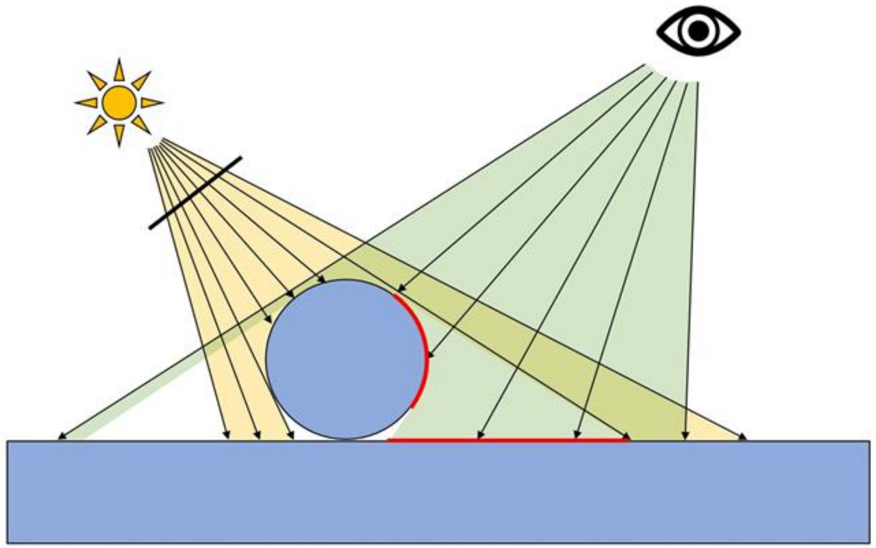
- Hard to model complex realistic material

Traditional shading

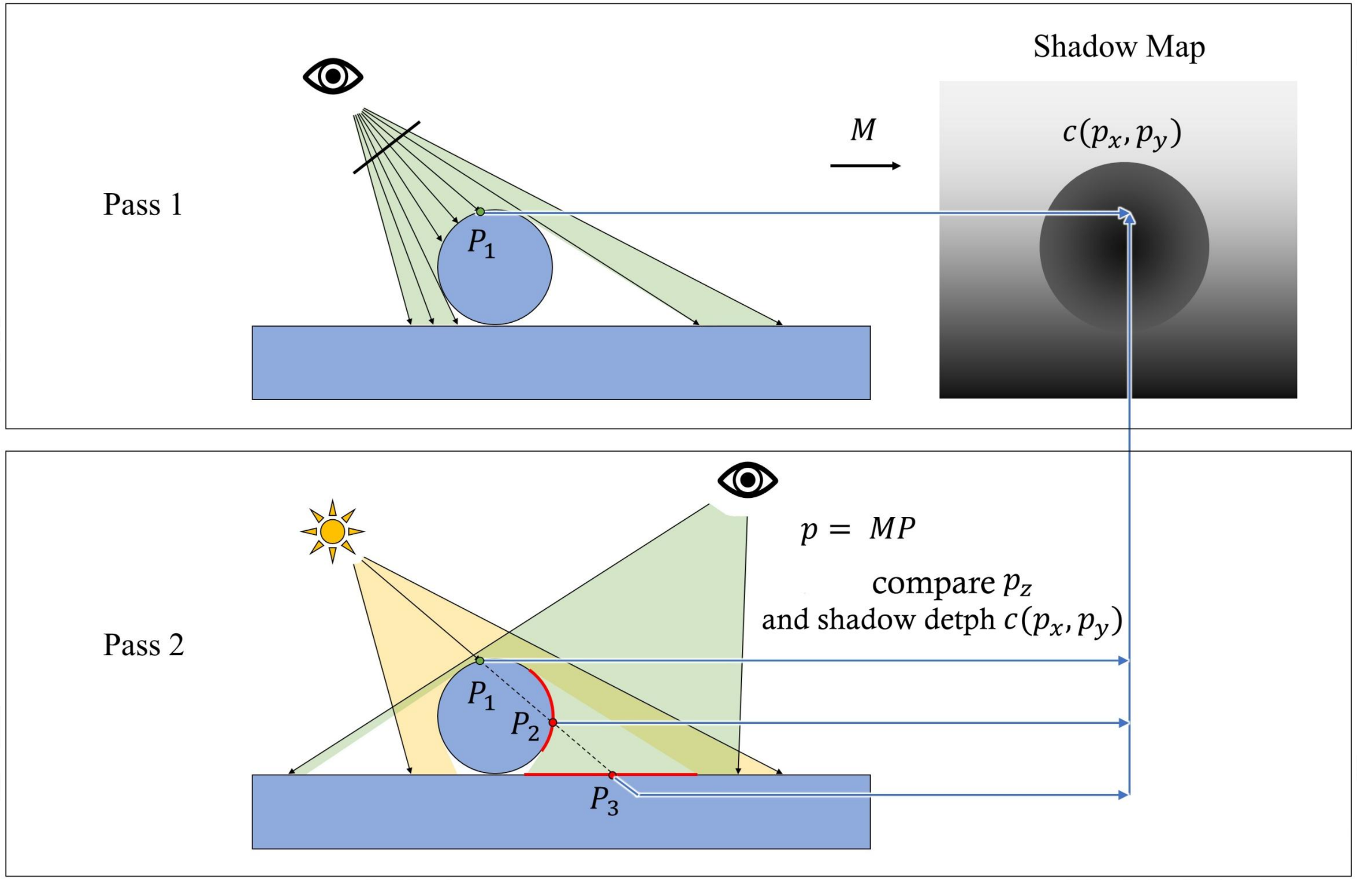
PBR shading

Shadow

- Shadow is nothing but space when the light is blocked by an opaque object
- Already obsolete method
  - planar shadow
  - shadow volume
  - projective texture



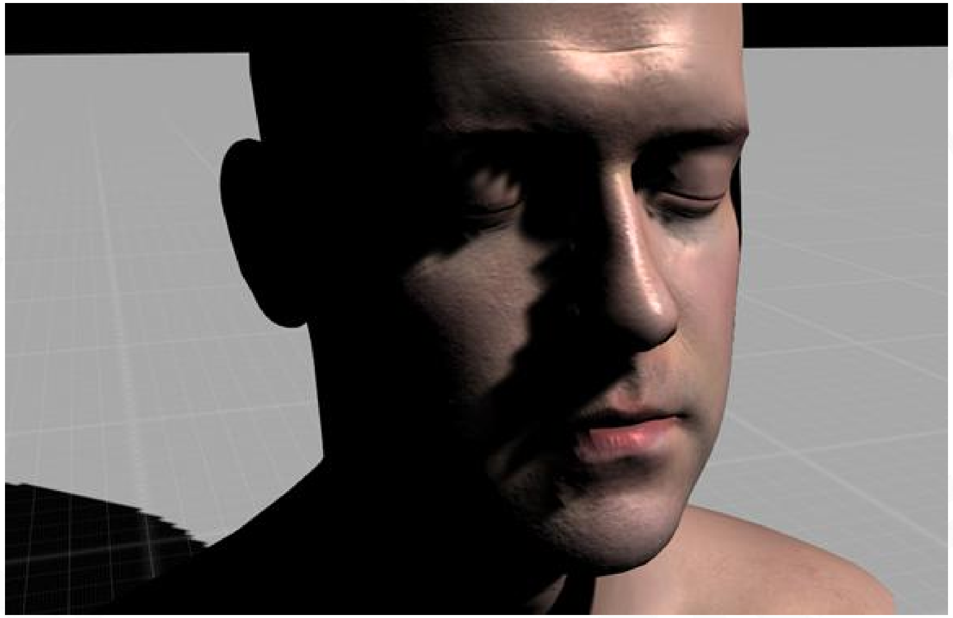
Shadow Map



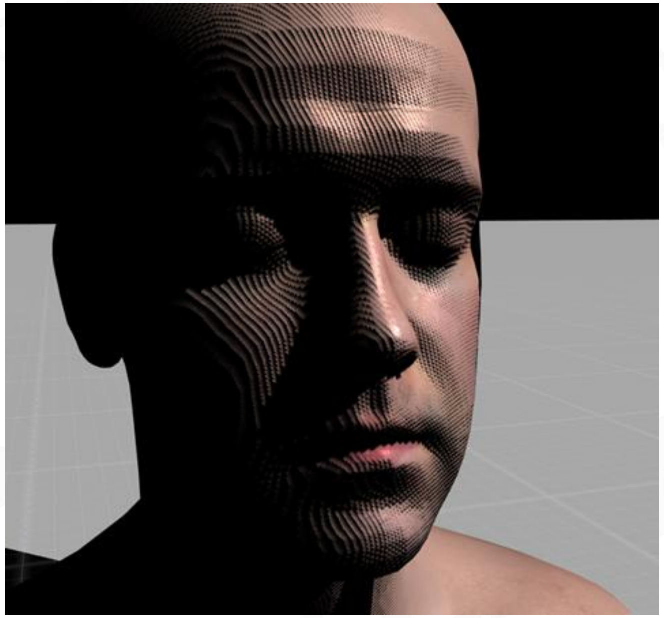

```
//project our 3D position to the shadow map
vec4 proj_pos = shadow_viewproj * pos;

//from homogeneous space to clip space
vec2 shadow_uv = proj_pos.xy / proj_pos.w;

//from clip space to uv space
shadow_uv = shadow_uv * 0.5 + vec2(0.5);

//get point depth (from -1 to 1)
float real_depth = proj_pos.z / proj_pos.w;

//normalize from [-1..+1] to [0..+1]
real_depth = real_depth * 0.5 + 0.5;

//read depth from depth buffer in [0..+1]
float shadow_depth = texture(shadowmap, shadow_uv).x;

//compute final shadow factor by comparing
float shadow_factor = 1.0;
if (shadow_depth < real_depth)
    shadow_factor = 0.0;
```

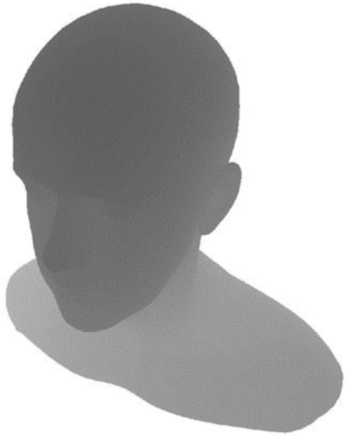
Problem of Shadow Map

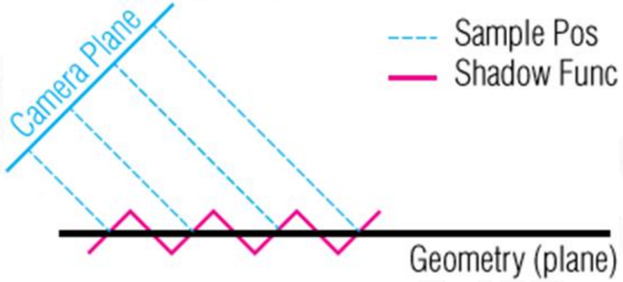
Resolution is limited on texture



Depth precision is limited in texture







Basic Shading Solution

- Simple light + Ambient
  - dominent light solves No. 1b
  - ambient and EnvMap solves No. 3 challanges
- Blinn-Phong material
  - solve No. 2 challange
- Shadow map
  - solve No.1a challange

Doom3

$$L_r(\mathbf{x},\vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x},\vec{\omega}_i,\vec{\omega}_r) L_i(\mathbf{x},\vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$

Cheap, Robust and Easy Modification

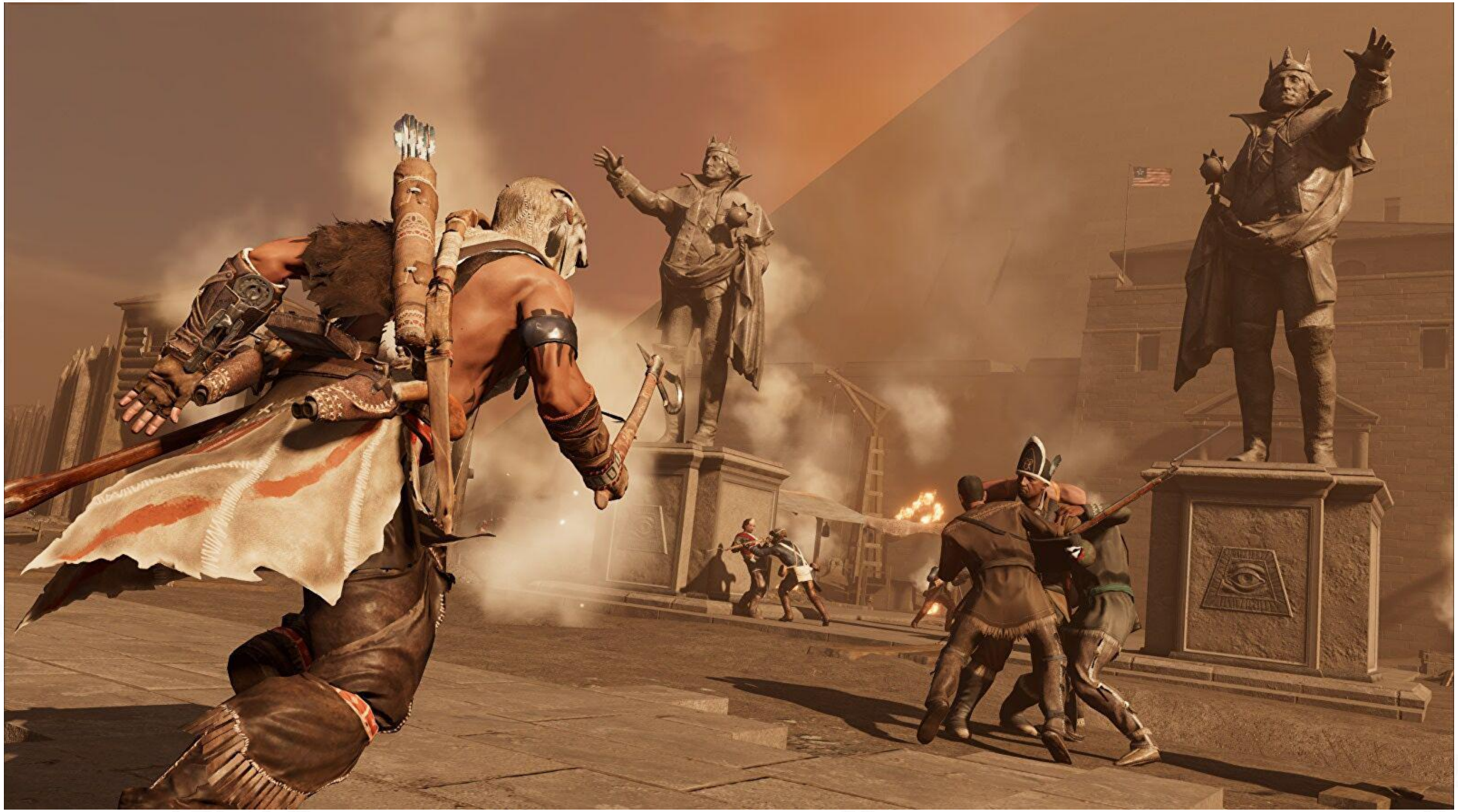
First Wave of AAA Quality

AAA Quality of 15 Years Ago

Assassin's Creed

AAA Quality of 10 Years Ago

Assassin's Creed III



AAA Quality of 5 Years Ago

Assassin's Creed: Origins

Pre-computed Global Illumination

Why Global Illumination is Important

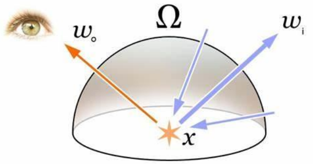
Direct illumination

Direct + indirect illumination

How to Represent Indirect Light

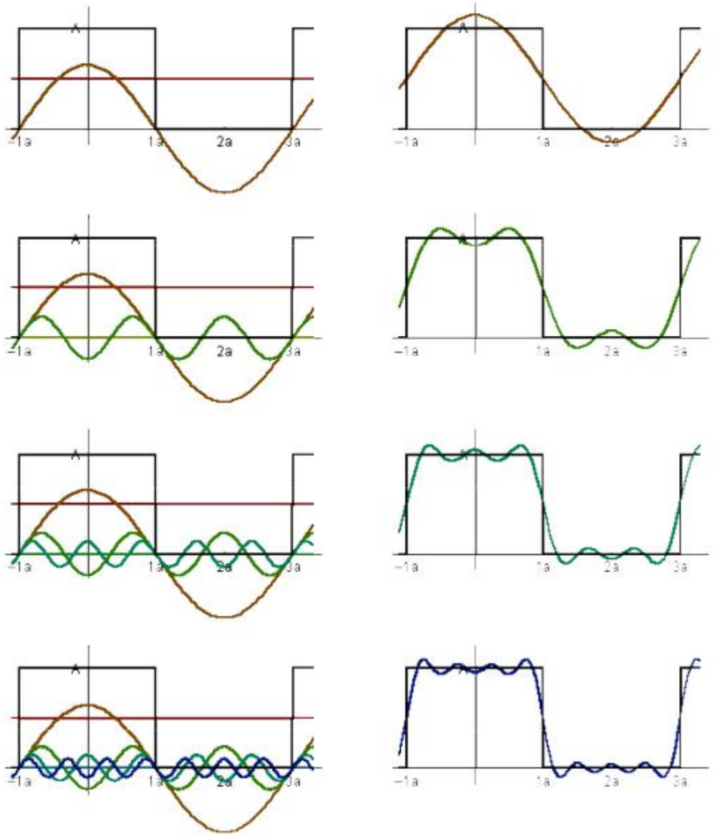
- Good compression rate
  - We need to store millions of radiance probes in a level
- Easy to do integration with material function
  - Use polynomial calculation to convolute with material BRDF

$$L_r(\mathbf{x},\vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x},\vec{\omega}_i,\vec{\omega}_r)\boxed{L_i(\mathbf{x},\vec{\omega}_i)}\cos\theta_i \,\mathrm{d}\vec{\omega}_i$$



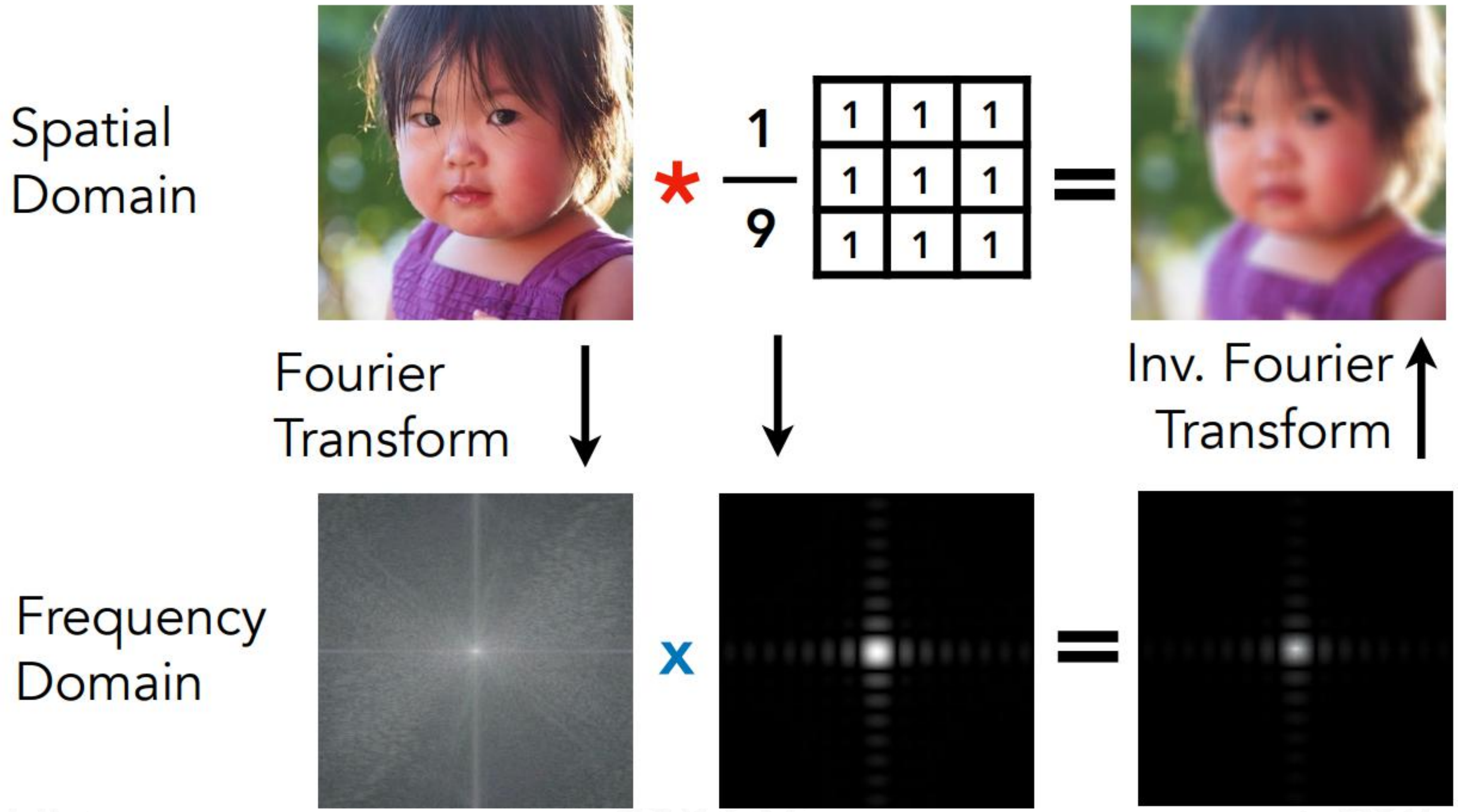
Fourier Transform

Joseph Fourier 1768 - 1830



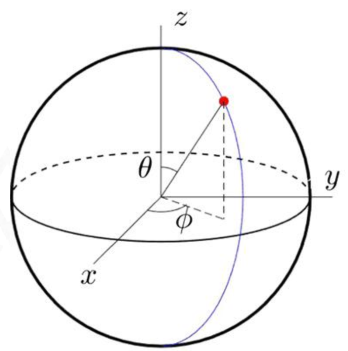
$$f(x) = \frac{A}{2} + \frac{2A\cos(t\omega)}{\pi} - \frac{2A\cos(3t\omega)}{3\pi} + \frac{2A\cos(5t\omega)}{5\pi} - \frac{2A\cos(7t\omega)}{7\pi} + \cdots$$

Convolution Theorem

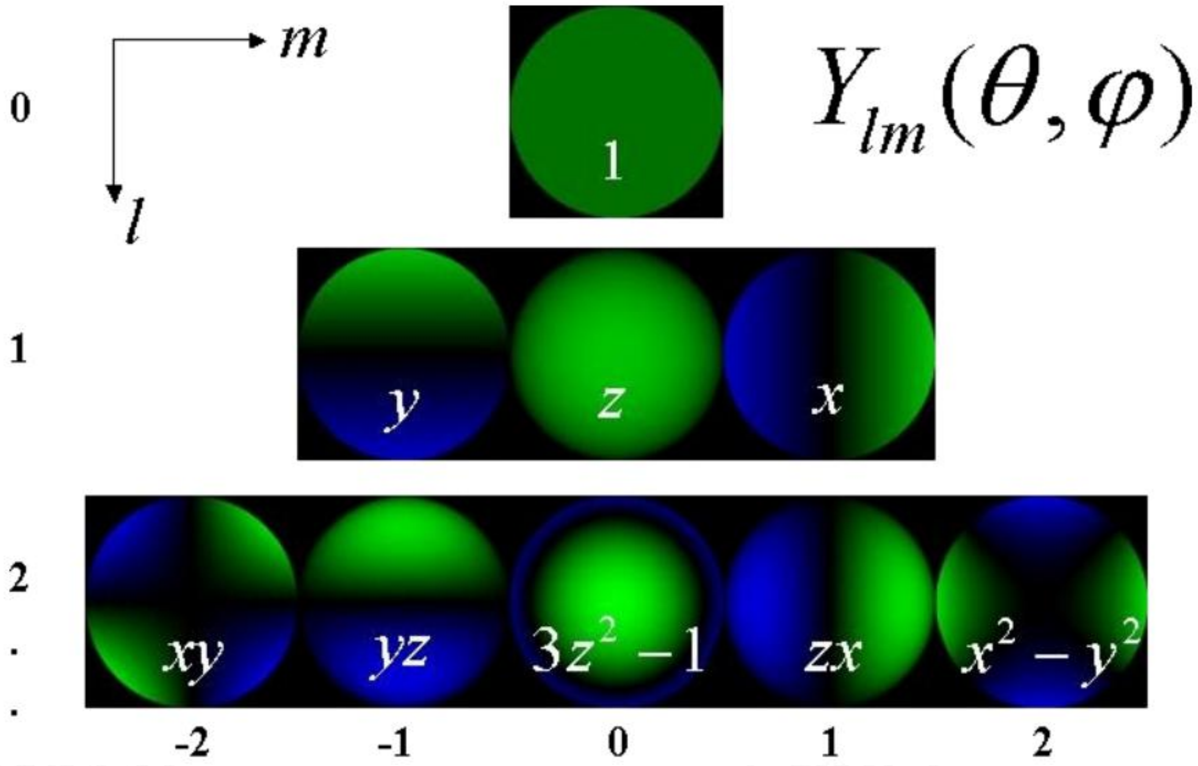


Spherical Harmonics

$$Y_{lm}(\theta,\phi) \quad = \quad N_{lm}P_{lm}(\cos\theta)e^{Im\phi}$$



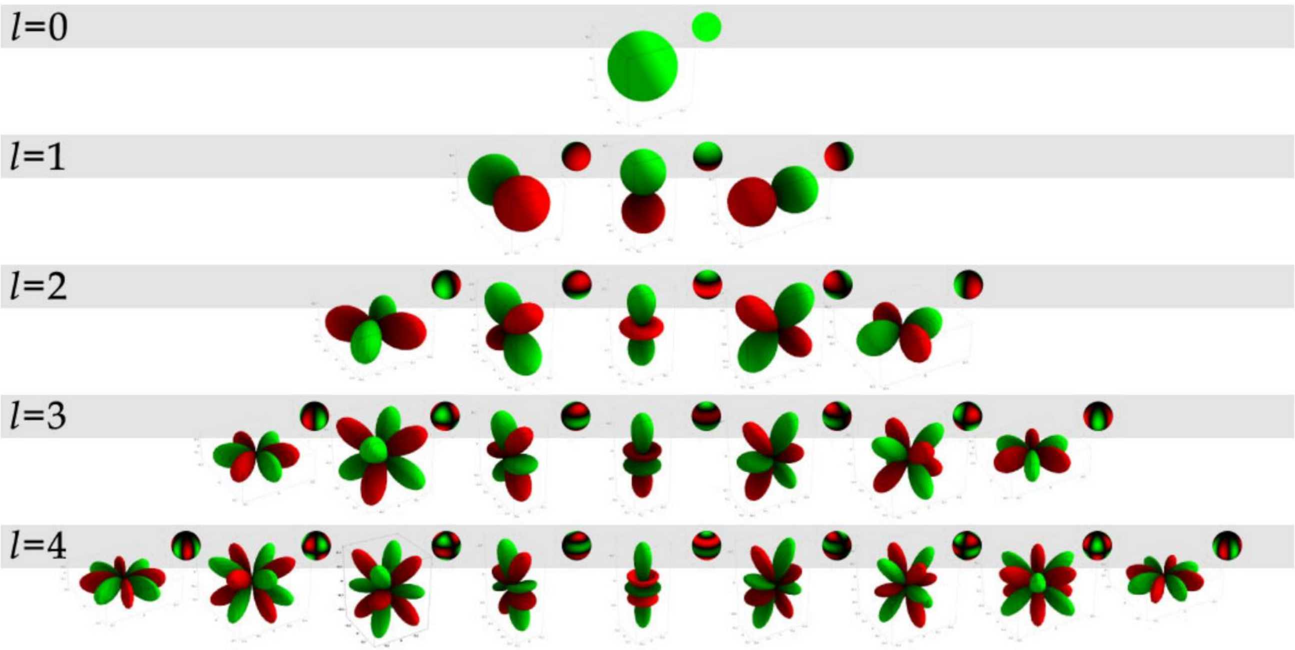
$$x = \sin\theta\cos\phi$$
$$y = \sin\theta\sin\phi$$
$$z = \cos\theta$$

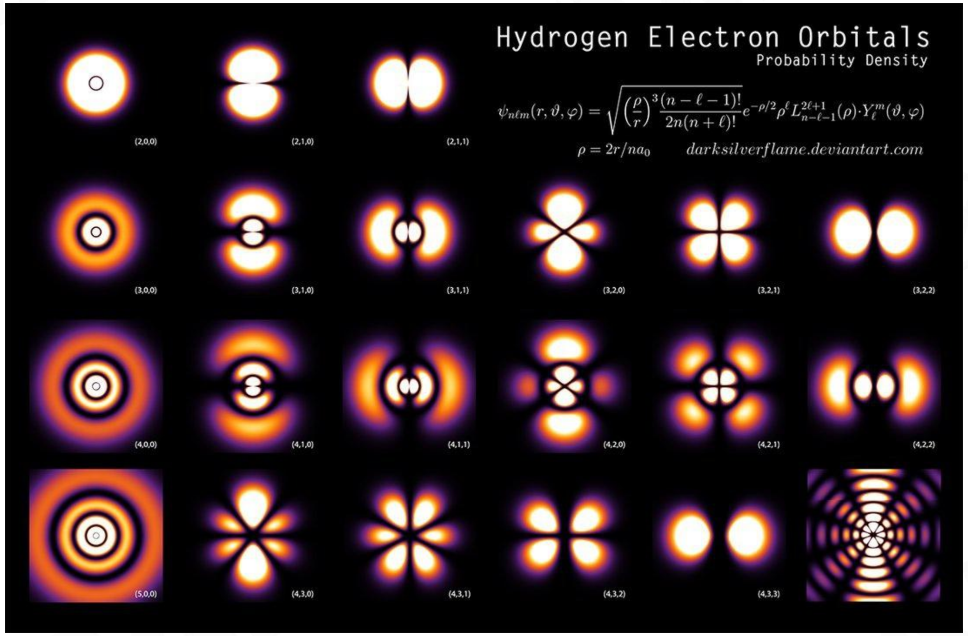


Complex sphere integration can be approximated by quadratic polynomial：

$$\int_{\theta=0}^{\pi}\int_{\phi=0}^{2\pi} L(\theta,\phi)Y_{lm}(\theta,\phi)\sin\theta d\theta d\phi \approx \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^T M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

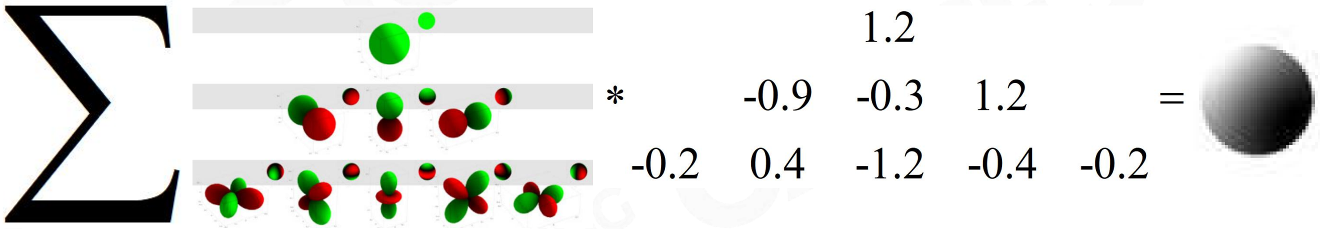
Spherical Harmonics

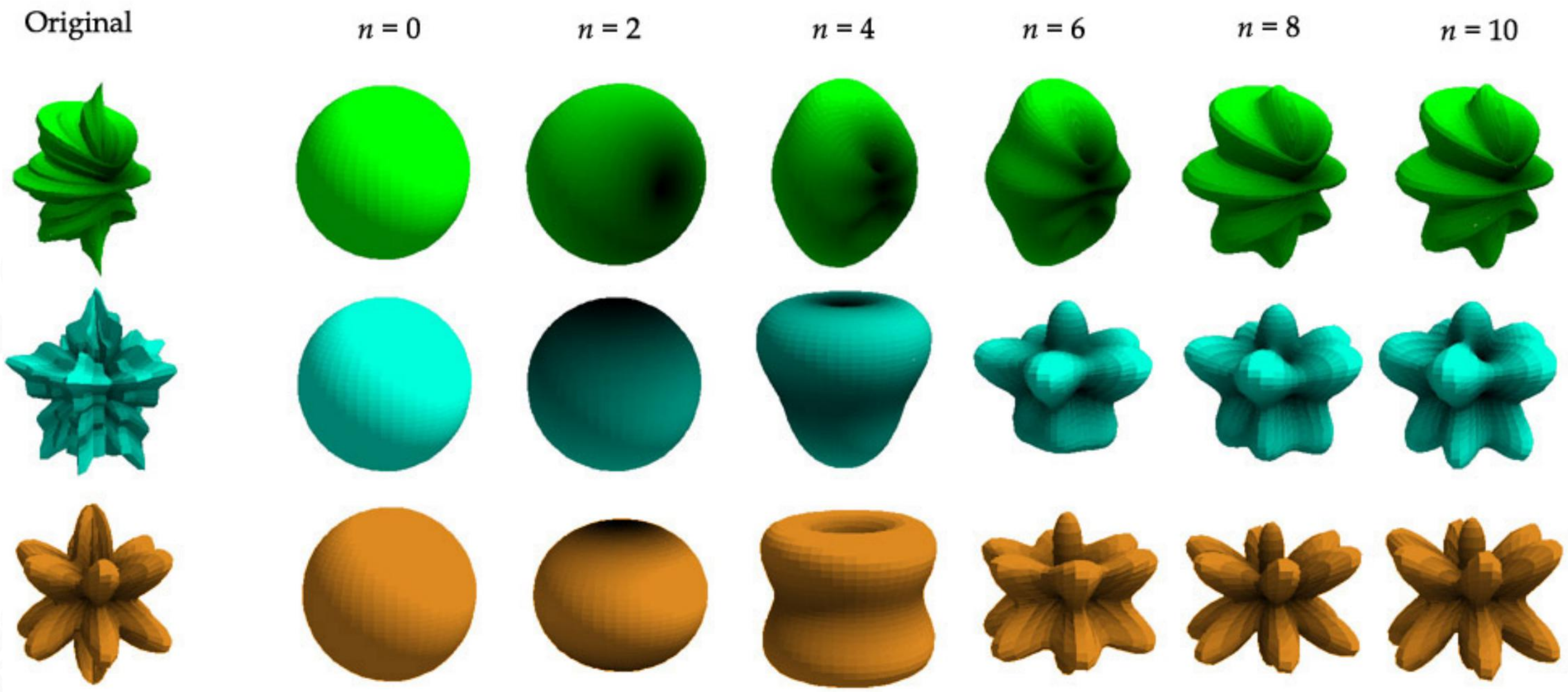




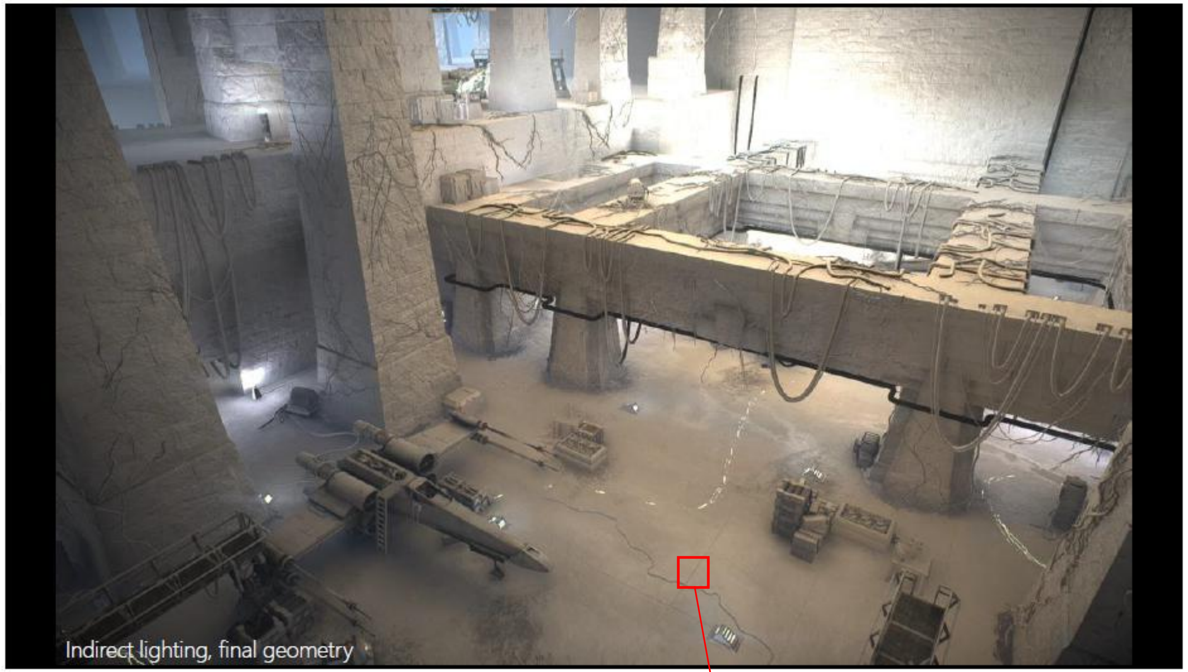
Spherical Harmonics, a mathematical system analogous to the Fourier transform but defined across the surface of a sphere. The SH functions in general are defined on imaginary numbers

Spherical Harmonics Encoding

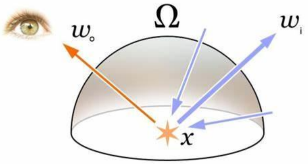




Sampling Irradiance Probe Anywhere

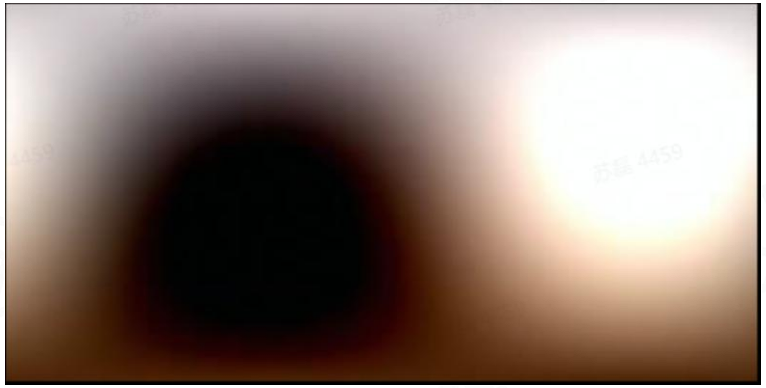


$$L_r(\mathbf{x},\vec{\omega}_r) = \int_{H^2} f_r(\mathbf{x},\vec{\omega}_i,\vec{\omega}_r) \boxed{L_i(\mathbf{x},\vec{\omega}_i)} \cos\theta_i \, \mathrm{d}\vec{\omega}_i$$



Compress Irradiance Probe to SH1

Source Irradiance Probe



Compressed Irradiance Probe By SH1

```
SHL1 shEvaluateL1(vec3 p)
{
  float Y0 = 0.282095f; // sqrt(1/fourPi)
  float Y1 = 0.488603f; // sqrt(3/fourPi)
  SHL1 sh;
  sh[0] = Y0;
  sh[1] = Y1 * p.y;
  sh[2] = Y1 * p.z;
  sh[3] = Y1 * p.x;
  return sh;
}
```

Reconstruct Irradiance In Shader

EA. (2018). Precomputed Global Illumination in Frostbite. Retrieved from https://www.ea.com/frostbite/news/precomputed-global-illumination-in-frostbite

Store and Shading with SH

- Use 4 RGB textures to store 12 SH coefficients
  - $L0$ coefficients in HDR (BC6H texture)
  - $L1$ coefficients in LDR (3x BC7 or BC1 textures)
- Total footprint for RGB SH lightmaps:
  - **32 bits (4 bytes) / texel** for BC6+BC7, high quality mode
  - **20 bits (2.5 bytes) / texel** for BC6+BC1, low quality mode

Just RGBA8 color

- shEvaluateL1 can be merged with shApplyDiffuseConvolutionL1

```
SHL1 shEvaluateL1(vec3 p)
{
  float Y0 = 0.282095f; // sqrt(1/fourPi)
  float Y1 = 0.488603f; // sqrt(3/fourPi)
  SHL1 sh;
  sh[0] = Y0;
  sh[1] = Y1 * p.y;
  sh[2] = Y1 * p.z;
  sh[3] = Y1 * p.x;
  return sh;
}
```

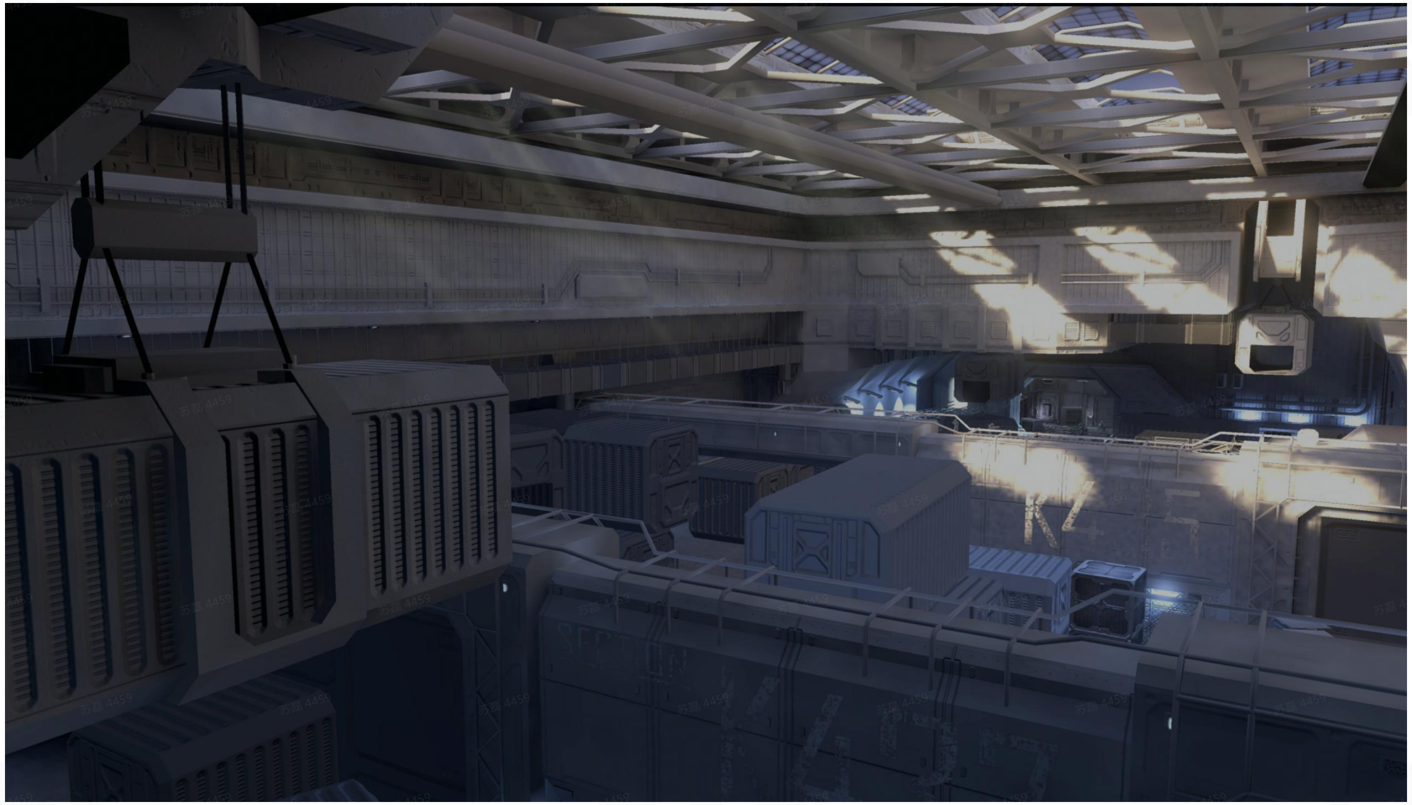
```
void shApplyDiffuseConvolutionL1(SHL1& sh)
{
  float A0 = 0.886227f; // pi/sqrt(fourPi)
  float A1 = 1.023326f; // sqrt(pi/3)
  sh[0] *= A0;
  sh[1] *= A1;
  sh[2] *= A1;
  sh[3] *= A1;
}
```

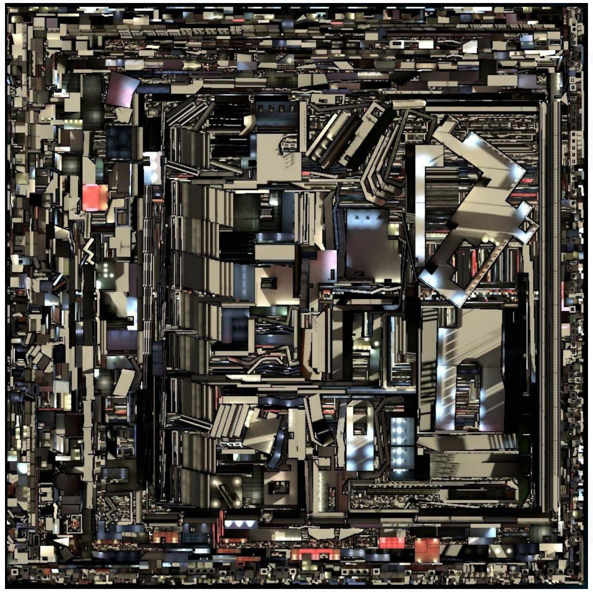
```
SHL1 shEvaluateDiffuseL1(vec3 p)
{
    float AY0 = 0.25f;
    float AY1 = 0.50f;
    SHL1 sh;
    sh[0] = AY0;
    sh[1] = AY1 * p.y;
    sh[2] = AY1 * p.z;
    sh[3] = AY1 * p.x;
    return sh;
}
```

Simple diffuse shading

EA. (2018). Precomputed Global Illumination in Frostbite. Retrieved from https://www.ea.com/frostbite/news/precomputed-global-illumination-in-frostbite

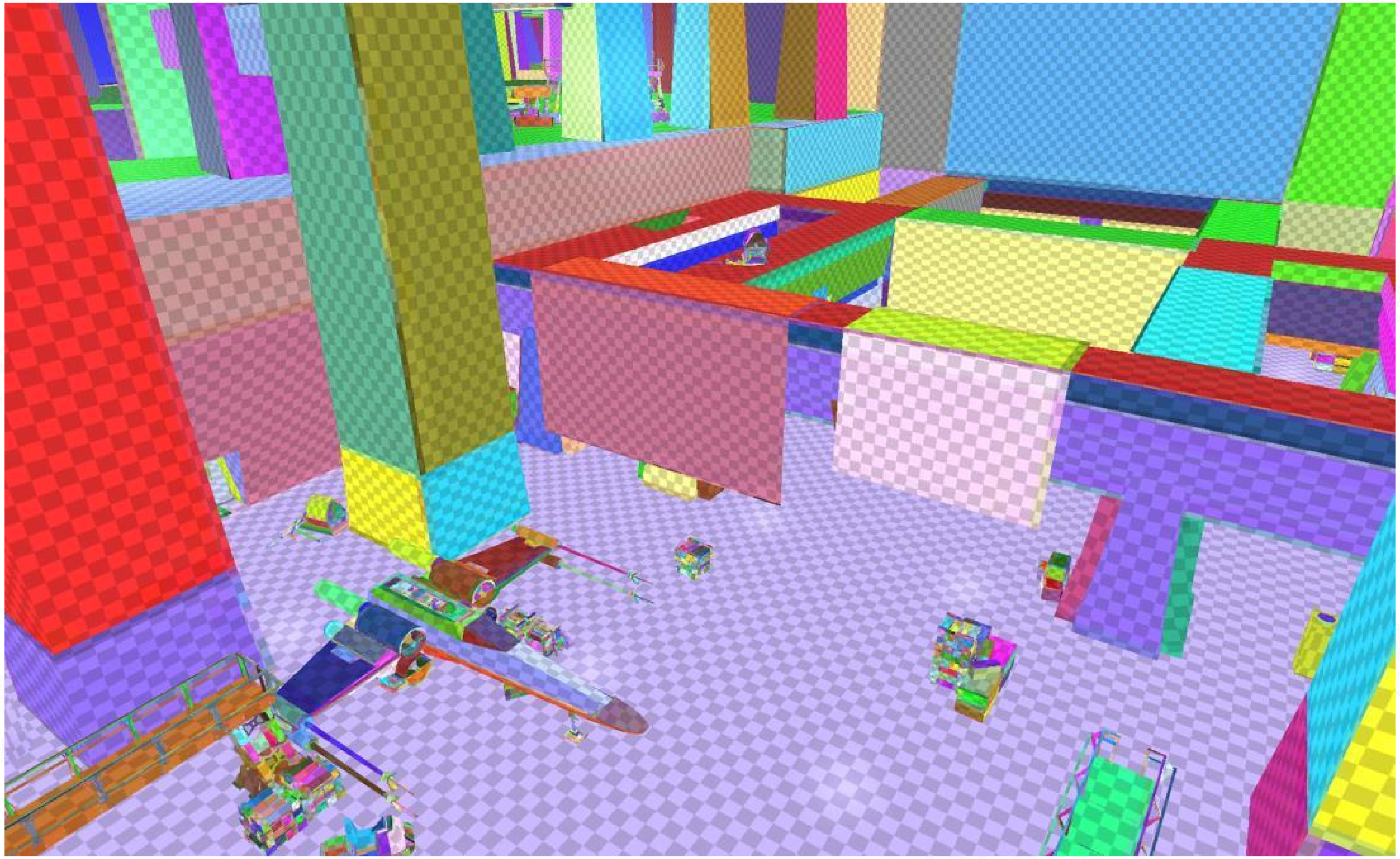
SH Lightmap: Precomputed GI





- Parameterized all scene into huge 2D lightmap atlas
- Using offline lighting farm to calculate irradiance probes for all surface points
- Compress those irradiance probes into SH coefficients
- Store SH coefficients into 2D atlas lightmap textures

Lightmap: UV Atlas



Lightmap density

- Low-poly proxy geometry
- Fewer UV charts/islands
- Fewer lightmap texels are wasted

Lightmap: Lighting

Indirect lighting, final geometry

- Project lightmap from proxies to all LODs
- Apply mesh details
- Add short-range, high-frequency lighting detail by HBAO

Lightmap: Lighting + Direct Lighting

Direct + indirect lighting, final geometry

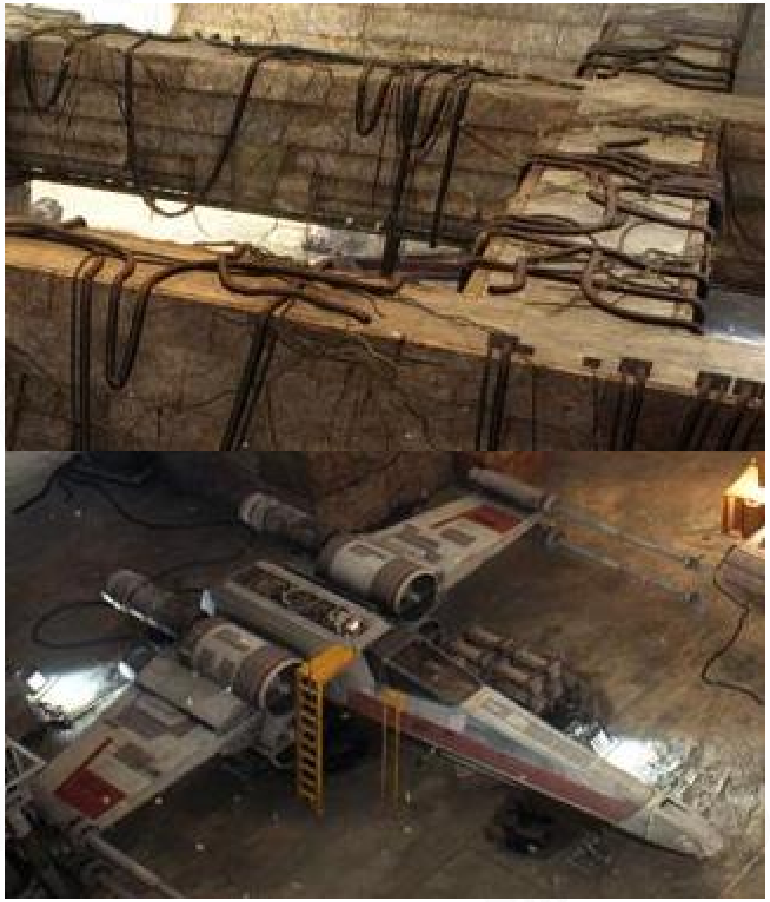
- Compute direct lighting dynamically

Final Shading with Materials

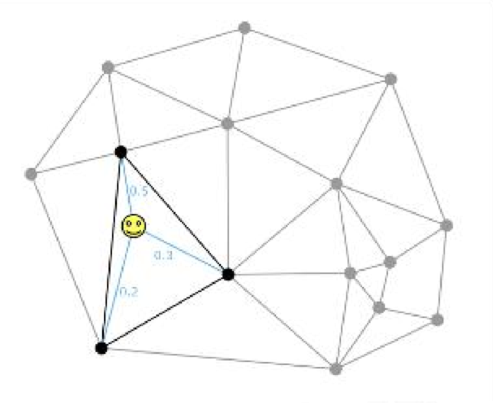
Final frame
- Combined with materials

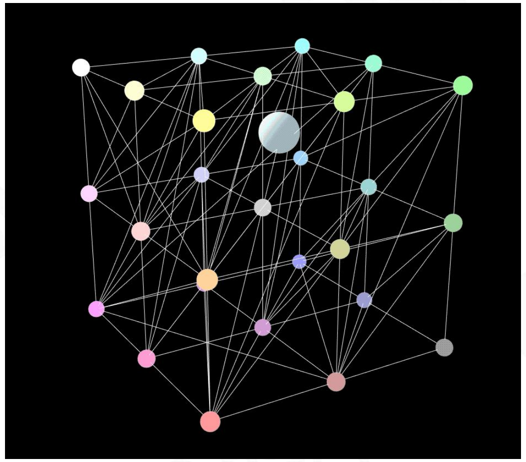
Lightmap

- **Pros**
  - Very efficient on runtime
  - Bake a lot of fine details of GI on environment
- **Cons**
  - Long and expensive precomputation (lightmap farm)
  - Only can handle static scene and static light
  - Storage cost on package and GPU

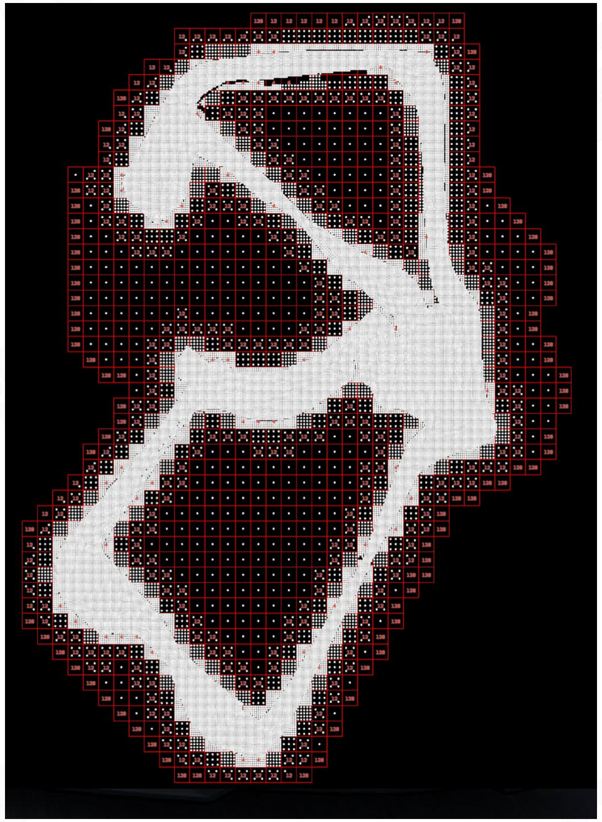


Light Probe: Probes in Game Space

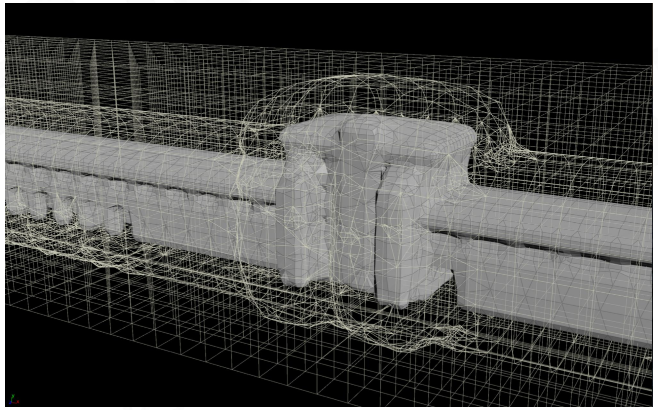


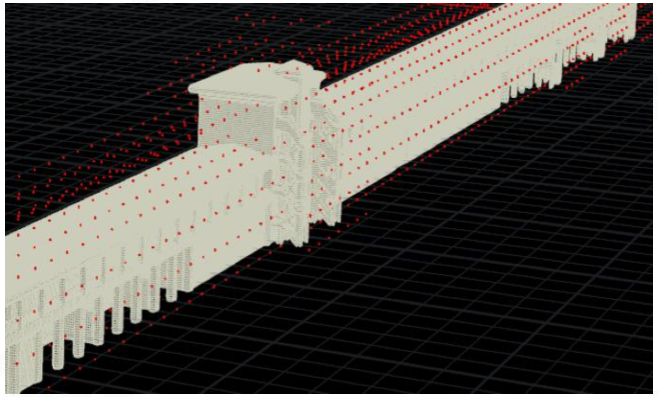


Light Probe Point Generation

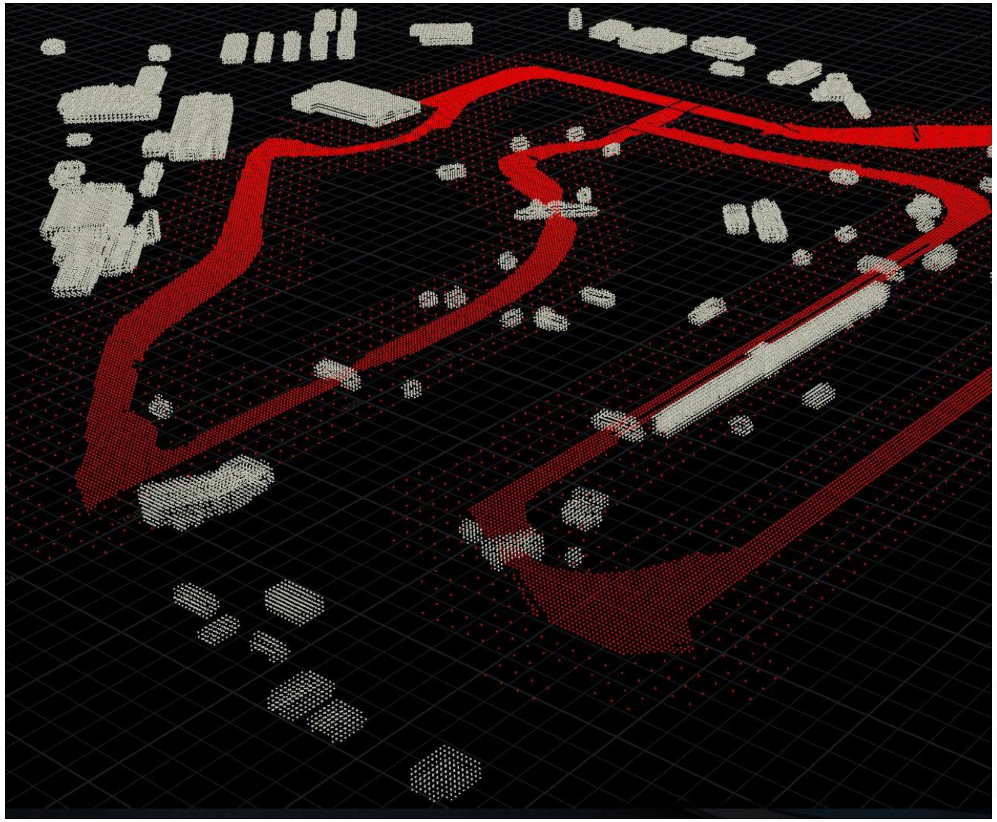


Generate by terrain and road





Generate by voxel



Final Light Probe Cloud

Reflection Probe

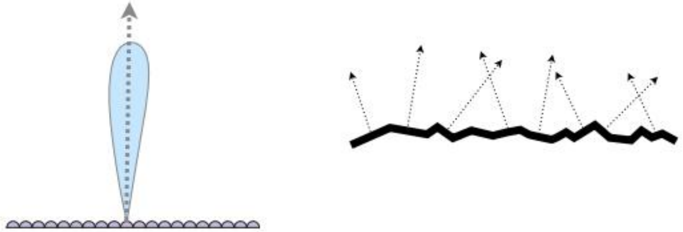
Light Probes + Reflection Probes

- **Pros**
  - Very efficient on runtime
  - Can be applied to both static and dynamic objects
  - Handle both diffuse and specular shading
- **Cons**
  - A bunch of SH light probes need some precomputation
  - Can not handle fine detail of GI. I.e, soft shadow on overlapped structures

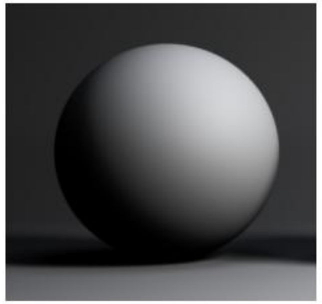
Physical-Based Material

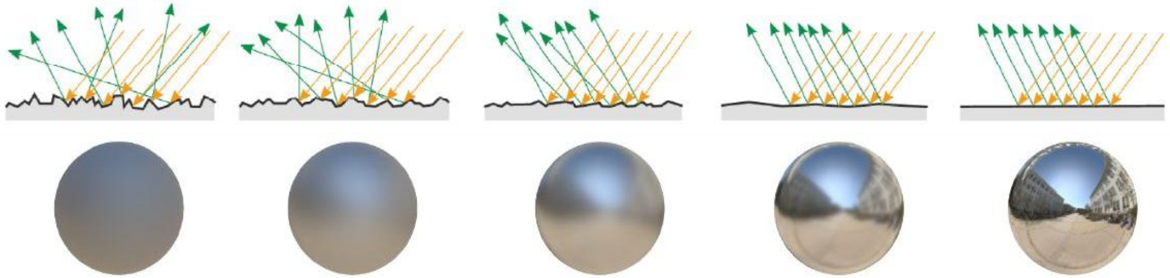
Microfacet Theory

- Key: the distribution of microfacets' normals
  - Concentrated <==> glossy



  - Spread <==> diffuse

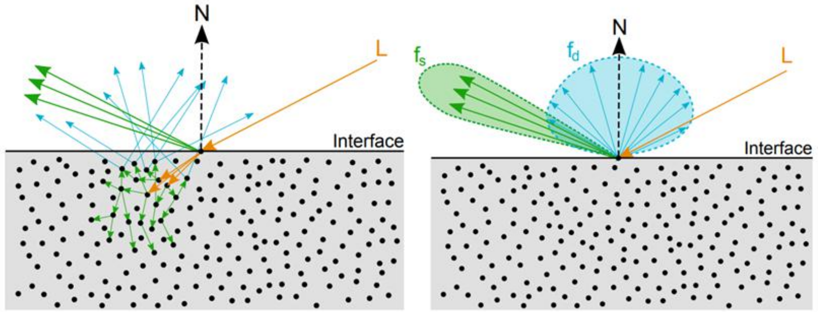




BRDF Model Based on Microfacet

$$L_o(\boldsymbol{x},\omega_o) = \int_{H^2} \boxed{f_r(\boldsymbol{x},\omega_o,\omega_i)} L_i(\boldsymbol{x},\omega_i)\cos\theta_i d\omega_i$$

$$f_r = k_d f_{Lambert} + f_{CookTorrance}$$



$$f_{Lambert} = \frac{c}{\pi}$$

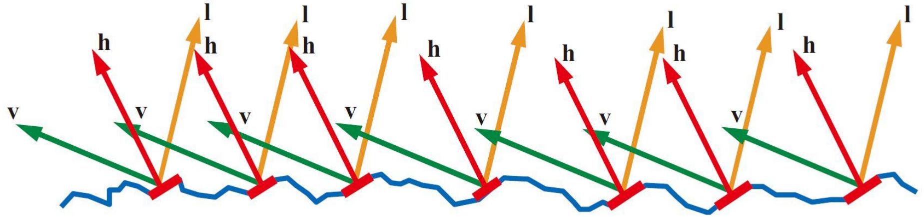
diffuse

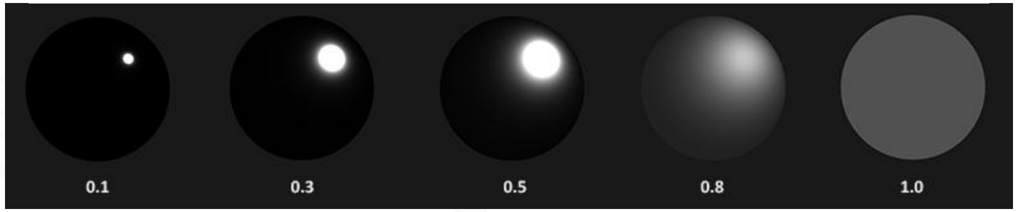
$$f_{CookTorrance} = \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}$$

spectual

$$L_o(\boldsymbol{x},\omega_o) = \int_{H^2} \left(k_d \frac{c}{\pi} + \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} \right) L_i(\boldsymbol{x},\omega_i)(\omega_i \cdot n) d\omega_i$$

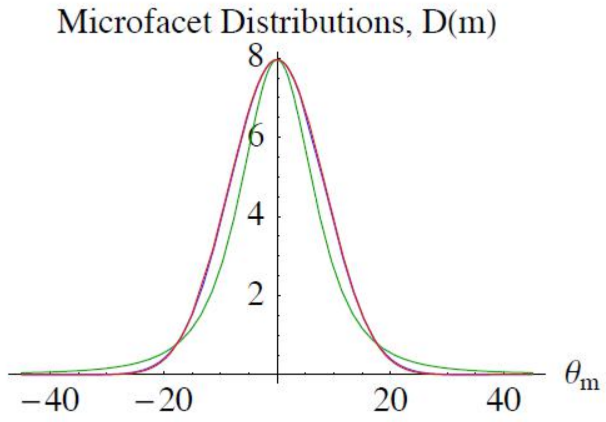
Normal Distribution Function





$$f_{CookTorrance} = \frac{\boxed{D}FG}{4(\omega_o \cdot n)(\omega_i \cdot n)}$$

$$NDF_{GGX}(n,h,\alpha) = \frac{\alpha^2}{\pi\left((n\cdot h)^2(\alpha^2-1)+1\right)^2}$$



Beckmann (red)

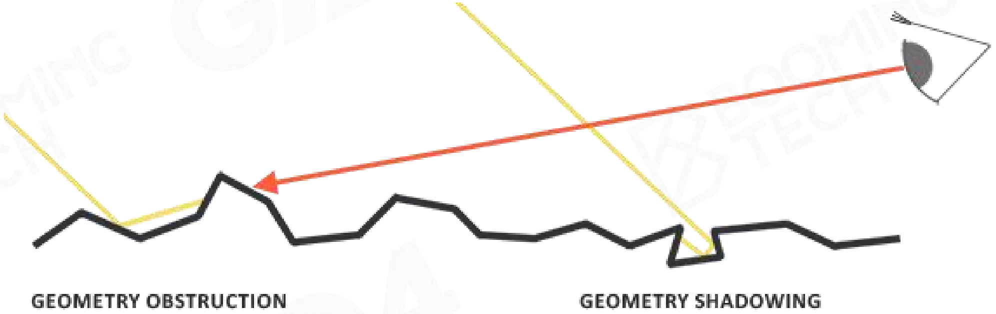
Phong (blue)

GGX (green)

```
// Normal Distribution Function using GGX Distribution
float D_GGX(float NoH, float roughness)
{
    float a2 = roughness * roughness;
    float f = (NoH * NoH) * (a2 - 1.0) + 1.0;
    return a2 / (PI * f * f);
}
```

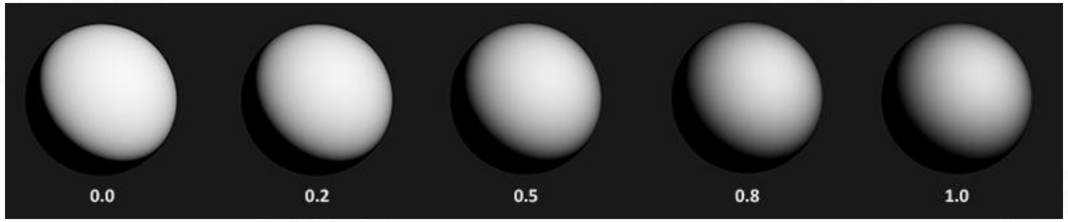
Geometric Attenuation Term (self-shadowing)

$$f_{CookTorrance} = \frac{DF\boxed{G}}{4(\omega_o \cdot n)(\omega_i \cdot n)}$$



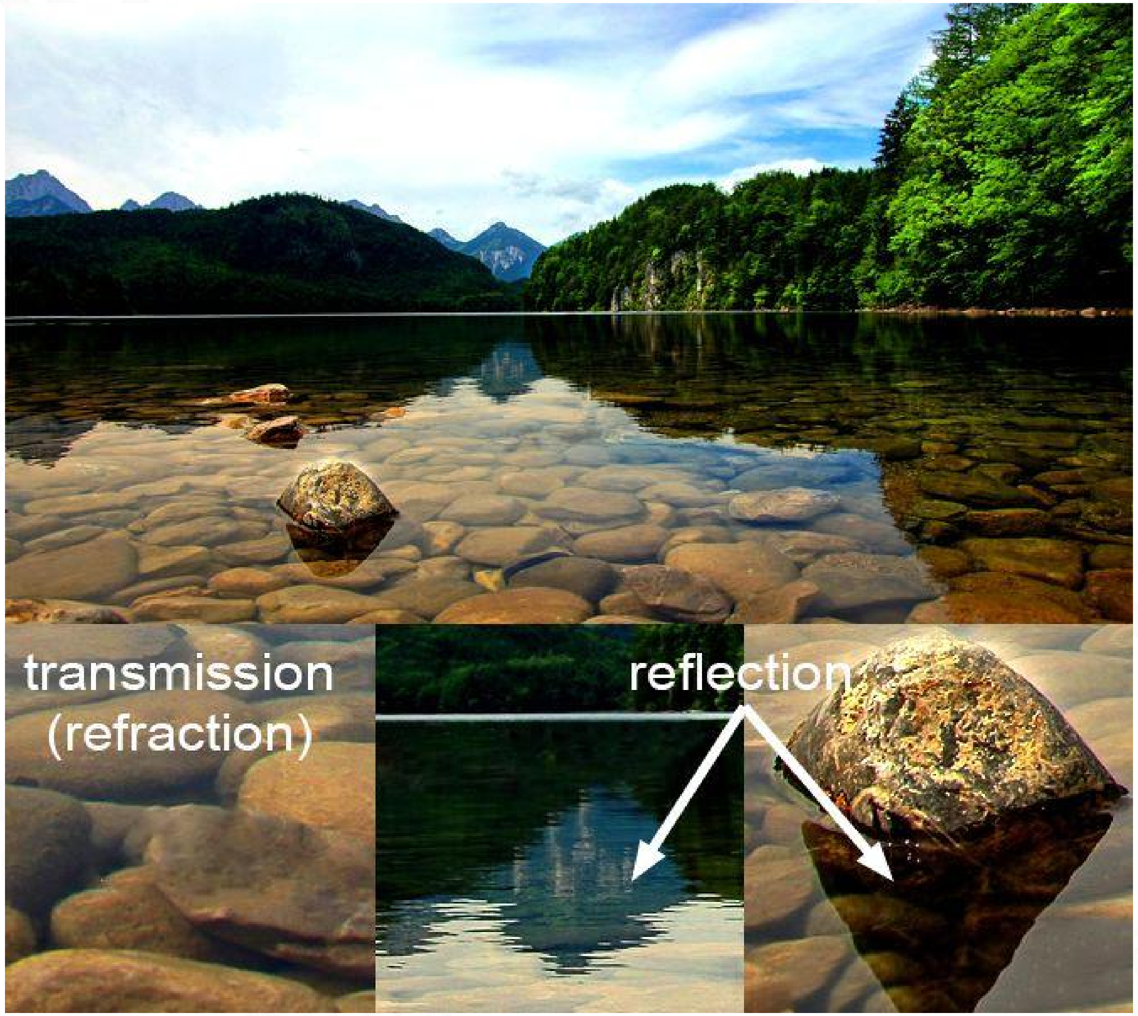
$$G_{Smith}(l,v) = G_{GGX}(l) \cdot G_{GGX}(v)$$

$$G_{GGX}(v) = \frac{n \cdot v}{(n \cdot v)(1-k)+k} \qquad k = \frac{(\alpha+1)^2}{8}$$




```
// Geometry Term: Geometry masking/shadowing due to microfacets
float GGX(float NdotV, float k) {
    return NdotV / (NdotV * (1.0 - k) + k);
}

float G_Smith(float NdotV, float NdotL, float roughness)
{
    float k = pow(roughness + 1.0, 2.0) / 8.0;
    return GGX(NdotL, k) * GGX(NdotV, k);
}
```

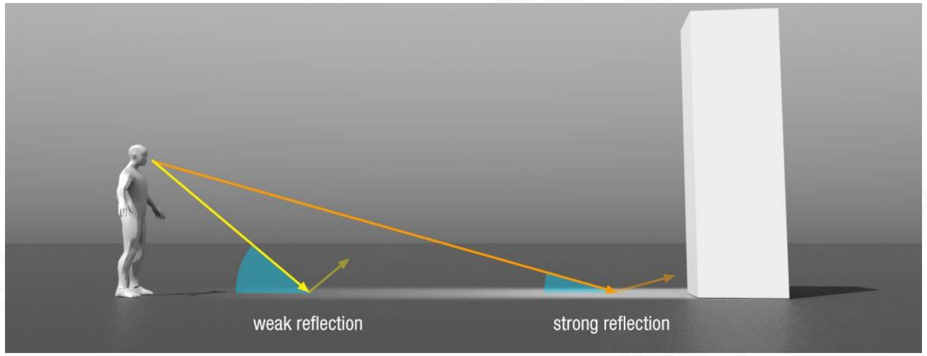
Fresnel Equation



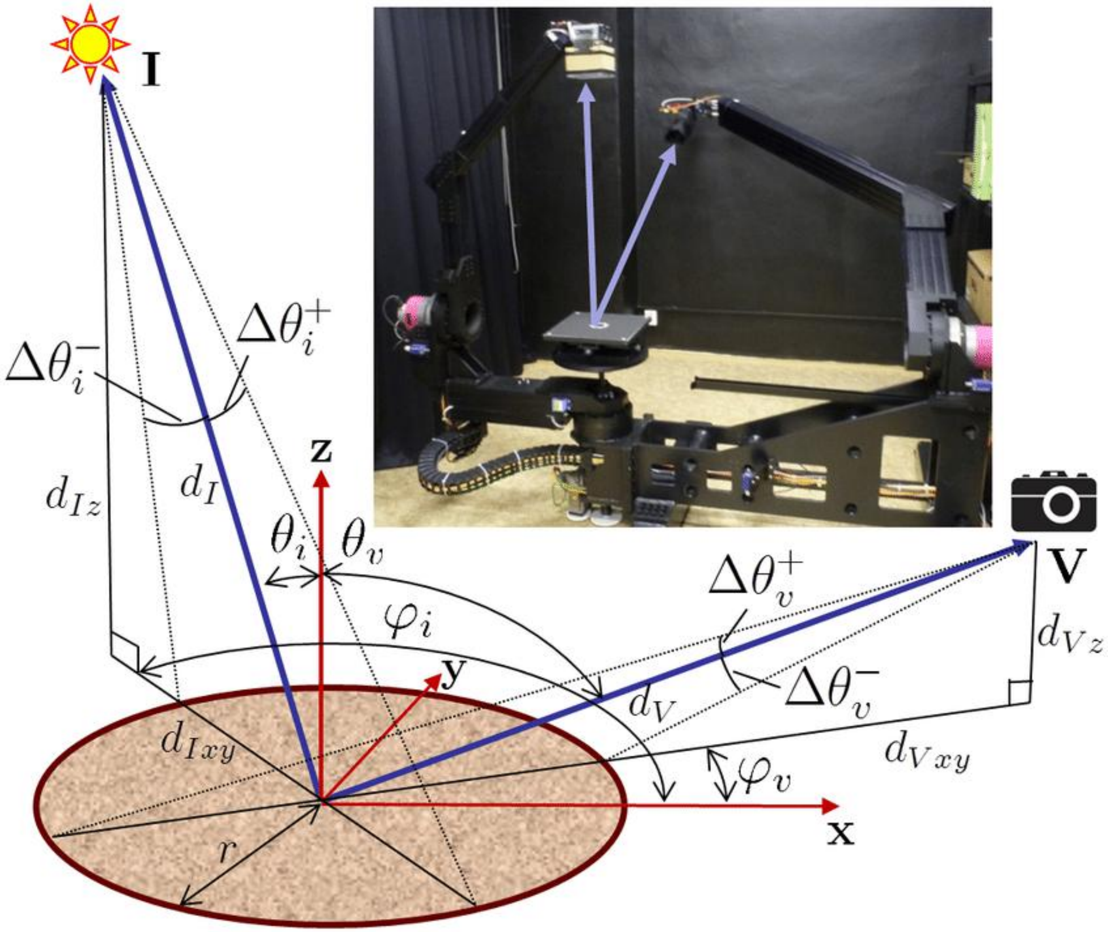
$$f_{CookTorrance} = \frac{D\boxed{F}G}{4(\omega_o \cdot n)(\omega_i \cdot n)}$$

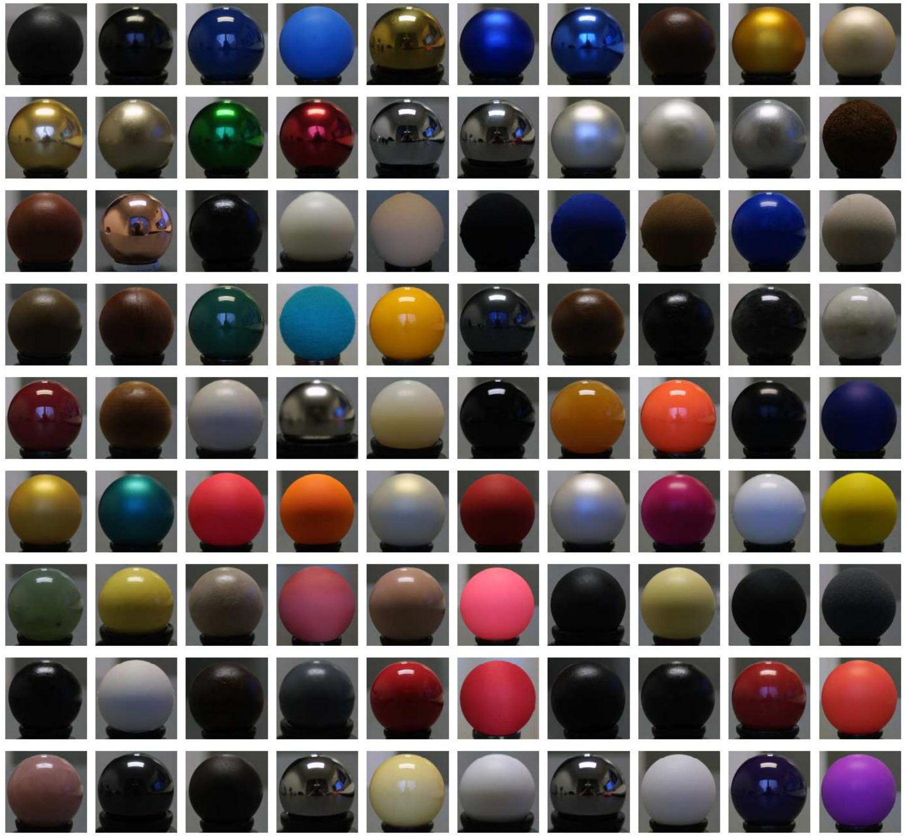
$$F_{Schlick}(h, v, F_0) = F_0 + (1 - F_0)\left(1 - (v \cdot h)\right)^5$$

```
// Fresnel term with scalar optimization
float F_Schlick(float VoH, float f0)
{
    float f = pow(1.0 - VoH, 5.0);
    return f0 + (1.0 - f0) * f;
}
```



Physical Measured Material





MERL BRDF Database of measured materials

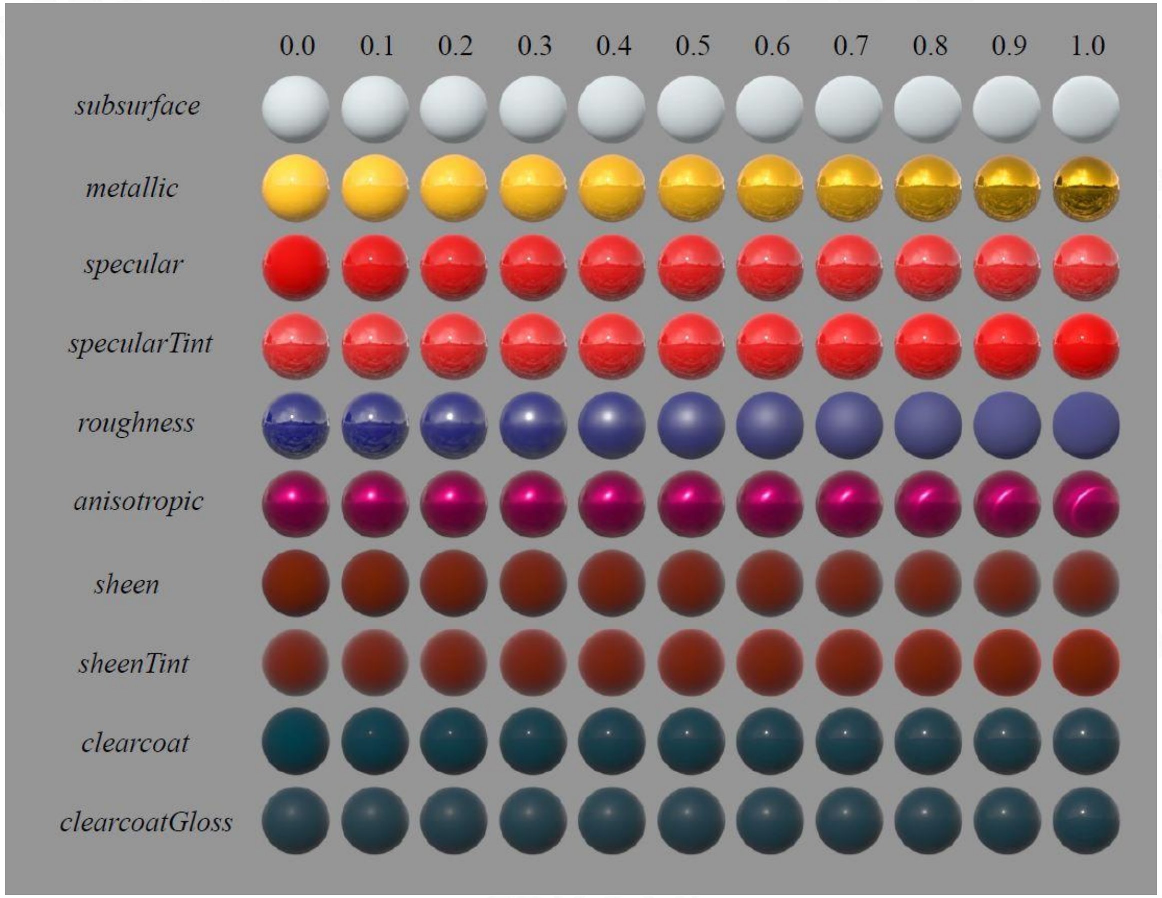
Disney Principled BRDF

Principles to follow when implementing model:

- Intuitive rather than physical parameters should be used
- There should be as few parameters as possible
- Parameters should be zero to one over their plausible range
- Parameters should be allowed to be pushed beyond their plausible range where it makes sense
- All combinations of parameters should be as robust and plausible as possible

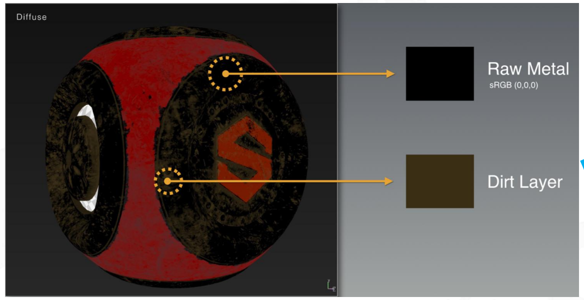
Brent Burley
Principal Software Engineer at Walt Disney Animation Studios

Disney Principle Material Parameters

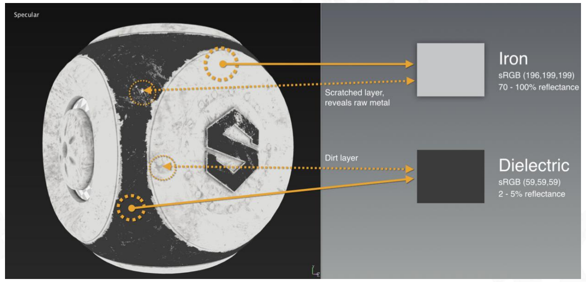


PBR Specular Glossiness

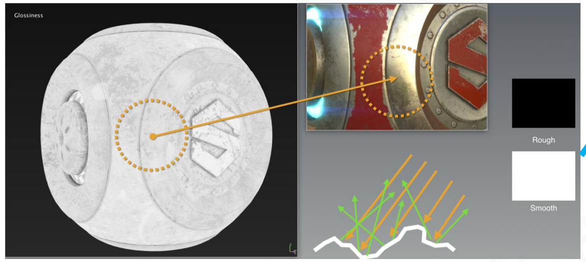
Material - SG



Diffuse - RGB - sRGB



Specular - RGB - sRGB

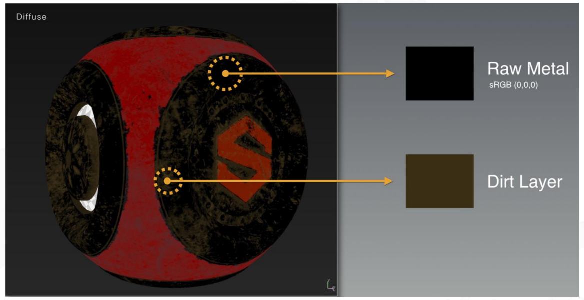


Glossiness - Grayscale - Linear

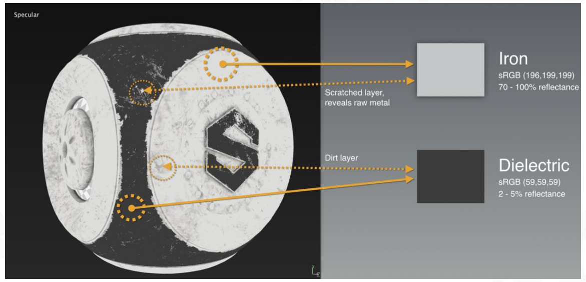
```
struct SpecularGlossiness
{
    float3 specular;
    float3 diffuse;
    float3 normal;
    float glossiness;
};

SpecularGlossiness getPBRParameterSG()
{
    SpecularGlossiness specular_glossiness;
    specular_glossiness.diffuse = sampleTexture(diffuse_texture, uv).rgb;
    specular_glossiness.specular = sampleTexture(specular_texture, uv).rgb;
    specular_glossiness.normal = sampleTexture(normal_texture, uv).rgb;
    specular_glossiness.glossiness = sampleTexture(gloss_texture, uv).r;
    return specular_glossiness;
}
```

PBR Specular Glossiness

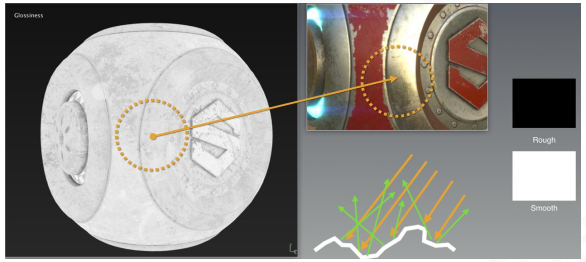
Material - SG



Diffuse - RGB - sRGB



Specular - RGB - sRGB



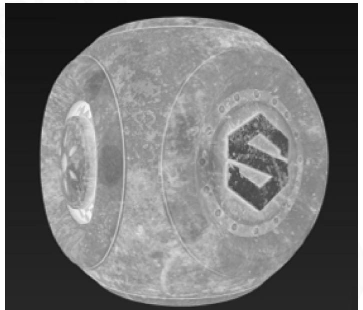
Glossiness - Grayscale - Linear

```
float3 calculateBRDF(SpecularGlossiness specular_glossiness)
{
    float3 half_vector = normalize(view_direction + light_direction);
    float N_dot_L = saturate(dot(specular_glossiness.normal, light_direction));
    float N_dot_V = abs(dot(specular_glossiness.normal, view_direction));
    float3 N_dot_H = saturate(dot(specular_glossiness.normal, half_vector));
    float3 V_dot_H = saturate(dot(view_direction, half_vector));

    // diffuse
    float3 diffuse = k_d * specular_glossiness.diffuse / PI;

    // specular
    float roughness = 1.0 - specular_glossiness.glossiness;
    float3 F0 = specular_glossiness.specular;

    float D = D_GGX(N_dot_H, roughness);
    float3 F = F_Schlick(V_dot_H, F0);
    float G = G_Smith(N_dot_V, N_dot_L, roughness);
    float denominator = 4.0 *  N_dot_V * N_dot_L + 0.001

    float3 specular = (D * F * G) / denominator;

    // brdf
    return diffuse + specular;
}

void PixelShaderSG()
{
    SpecularGlossiness specular_glossiness = getPBRParameterSG();
    float3 brdf_reflection = calculateBRDF(specular_glossiness);
    return brdf_reflection * light_intensity * cos(light_incident_angle)
}
```

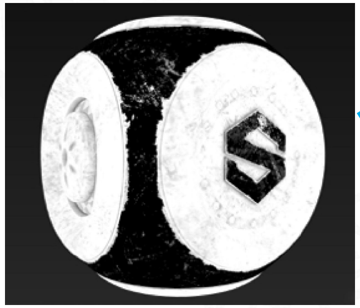
PBR Metallic Roughness

Material - MR

Base Color - RGB - sRGB



Roughness - Grayscale - Linear



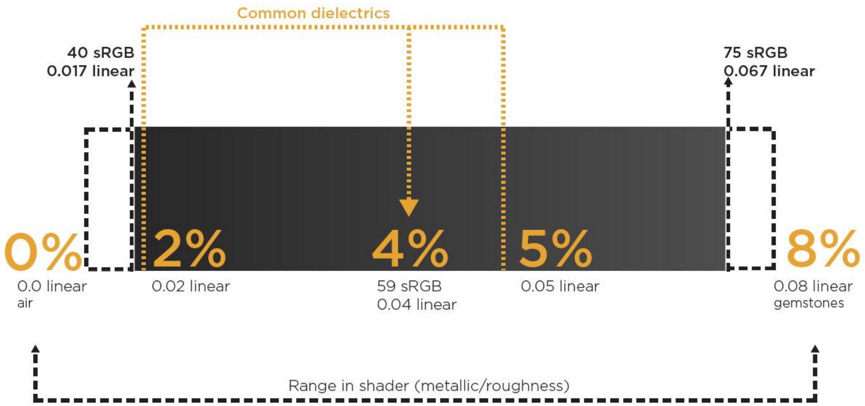
Metallic - Grayscale - Linear

```
struct MetallicRoughness
{
    float3 base_color;
    float3 normal;
    float roughness;
    float metallic;
};
```

Convert MR to SG

```
SpecularGlossiness ConvertMetallicRoughnessToSpecularGlossiness(MetallicRoughness metallic_roughness)
{
    float3 base_color = metallic_roughness.base_color;
    float roughness = metallic_roughness.roughness;
    float metallic = metallic_roughness.metallic;

    float3 dielectricSpecularColor = float3(0.08f * dielectricSpecular);
    float3 specular = lerp(dielectricSpecularColor, base_color, metallic);
    float3 diffuse = base_color - base_color * metallic;

    SpecularGlossiness specular_glossiness;
    specular_glossiness.specular = specular;
    specular_glossiness.diffuse = diffuse;
    specular_glossiness.glossiness = 1.0f - roughness;

    return result;
}
```

Dielectric	F0 (Linear)	F0 (sRGB)	Color
Water	0.02	39	
Living tissue	0.02–0.04	39–56	
Skin	0.028	47	
Eyes	0.025	44	
Hair	0.046	61	
Teeth	0.058	68	
Fabric	0.04–0.056	56–67	
Stone	0.035–0.056	53–67	
Plastics, glass	0.04–0.05	56–63	
Crystal glass	0.05–0.07	63–75	
Gems	0.05–0.08	63–80	
Diamond-like	0.13–0.2	101–124	



PBR Pipeline MR vs SG

MR

Pros

- Can be easier to author and less prone to errors caused by supplying incorrect dielectric F0 data
- Uses less texture memory, as metallic and roughness are both grayscale maps

Cons

- No control over F0 for dielectrics in map creation. However, most implementations have a specular control to override the base 4% value
- Edge artifacts are more noticeable, especially at lower resolutions

SG

Pros

- Edge artifacts are less apparent
- Control over dielectric F0 in the specular map

Cons

- Because the specular map provides control over dielectric F0, it is more susceptible to use of incorrect values. It is possible to break the law of conservation if handled incorrectly in the shader
- Uses more texture memory with an additional RGB map

Image-Based Lighting (IBL)

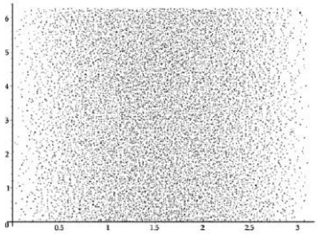
Basic Idea of IBL

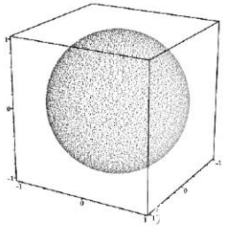
- An image representing distant lighting from all directions.

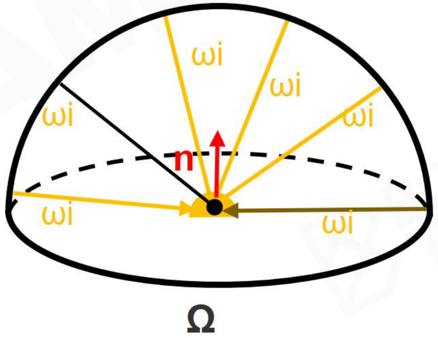
- How to shade a point under the lighting?
  Solving the rendering equation:

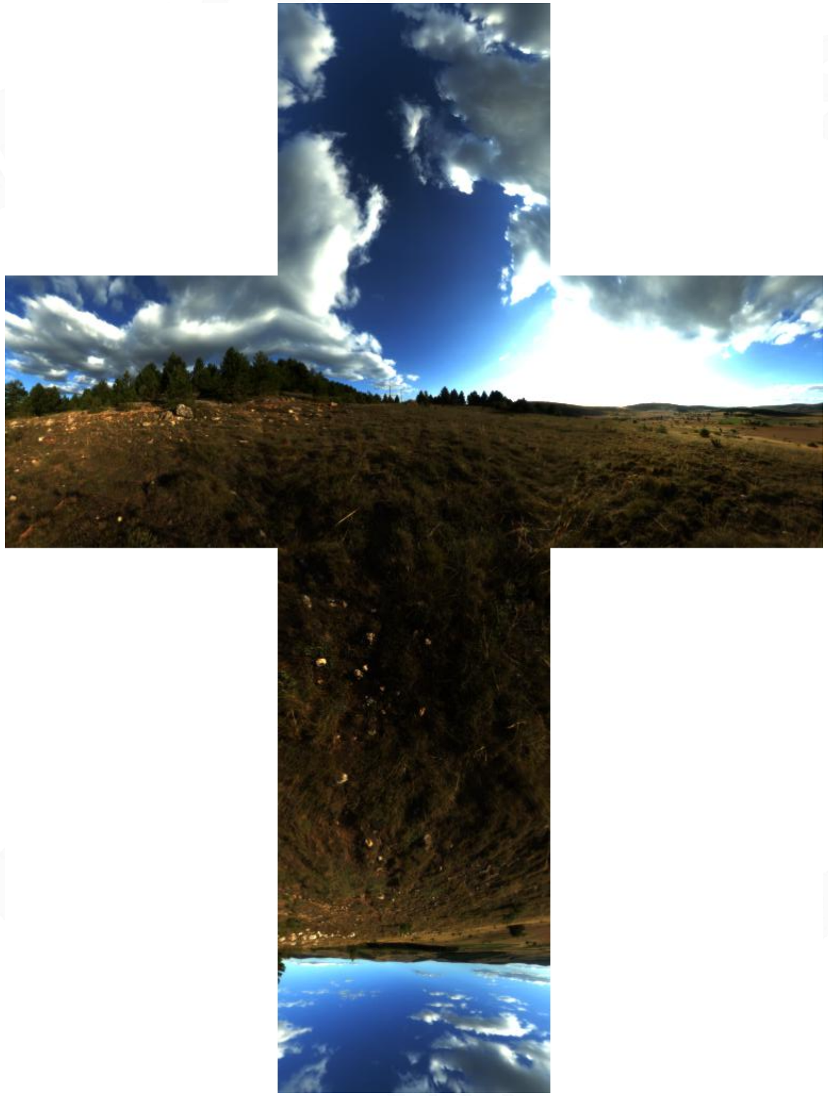
$$L_o(\boldsymbol{x},\omega_o) = \int_{H^2} f_r(\boldsymbol{x},\omega_o,\omega_i) L_i(\boldsymbol{x},\omega_i)\cos\theta_i d\omega_i$$

- Using Monte Carlo integration
  Large amount of sampling - Slow!









Recall BRDF Function

$$L_o(\mathbf{x},\omega_o) = \int_{H^2} \boxed{f_r(\mathbf{x},\omega_o,\omega_i)} L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i$$

$$f_r = k_d \boxed{f_{Lambert}} + \boxed{f_{CookTorrance}}$$

diffuse specular

$$\begin{aligned} L_o(\mathbf{x},\omega_o) &= \int_{H^2} (k_d f_{Lambert} + f_{CookTorrance}) L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i \\ &= \int_{H^2} k_d f_{Lambert} L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i + \int_{H^2} f_{CookTorrances} L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i \\ &= \boxed{L_d(\mathbf{x},\omega_o)} + \boxed{L_s(\mathbf{x},\omega_o)} \end{aligned}$$

Diffuse Irradiance Map

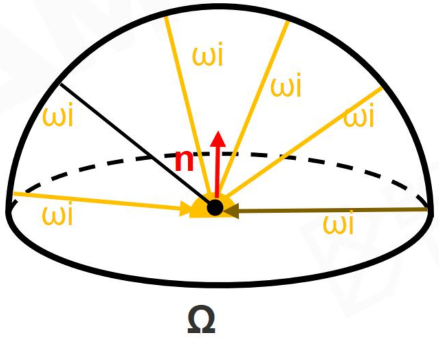
- Irradiance Map

$$f_{Lambert} = \frac{c}{\pi}$$

diffuse

$$\boxed{L_d(\boldsymbol{x},\omega_o)} = \int_{H^2} k_d f_{Lambert} L_i(\boldsymbol{x},\omega_i)\cos\theta_i d\omega_i$$

$$\approx k_d^* c \boxed{\frac{1}{\pi}\int_{H^2} L_i(\boldsymbol{x},\omega_i)\cos\theta_i d\omega_i}$$



Diffuse Irradiance Map

Specular Approximation

specular

$$L_s(\boldsymbol{x},\omega_o)=\int_{H^2}\boxed{f_{CookTorrances}}L_i(\boldsymbol{x},\omega_i)\cos\theta_i d\omega_i$$

$$f_{CookTorrance}=\frac{DFG}{4(\omega_o\cdot n)(\omega_i\cdot n)} \Rightarrow (\alpha, F_0, \theta, \ldots)$$

Let's approximation it by Split Sum

$$L_s(\boldsymbol{x},\omega_o)=\boxed{\frac{\int_{H^2}f_{CookTorrances}L_i(\boldsymbol{x},\omega_i)\cos\theta_i d\omega_i}{\int_{H^2}f_{CookTorrances}\cos\theta_i d\omega_i}}\cdot\boxed{\int_{H^2}f_{CookTorrances}\cos\theta_i d\omega_i}$$

Lighting term

BRDF term

Approximation: part (1/2)

The Lighting Term :

$$L_s(\mathbf{x},\omega_o)=\boxed{\frac{\int_{H^2} f_{CookTorrances} L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i}{\int_{H^2} f_{CookTorrances} \cos\theta_i d\omega_i}}\cdot\int_{H^2} f_{CookTorrances} \cos\theta_i d\omega_i$$

$$\approx\frac{\sum_k^N L(\omega_i^k)\boxed{G(\omega_i^k)}}{\sum_k^N G(\omega_i^k)} \rightarrow \alpha$$

Pre-filtered environment map

$\alpha(roughness)$

Approximation: part (2/2)

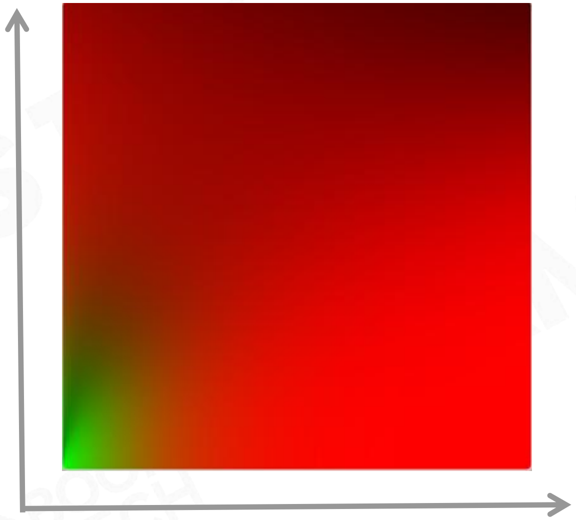
The BRDF Term:

$(\alpha, \cos\theta_i)$

$$L_s(\mathbf{x},\omega_o) = \frac{\int_{H^2} f_{CookTorrances} L_i(\mathbf{x},\omega_i)\cos\theta_i d\omega_i}{\int_{H^2} f_{CookTorrances}\cos\theta_i d\omega_i} \cdot \boxed{\int_{H^2} f_{CookTorrances}\cos\theta_i d\omega_i}$$

$$\approx F_0 \int_{H^2} \boxed{\frac{f_{CookTorrances}}{F}\left(1-(1-\cos\theta_i)^5\right)\cos\theta_i d\omega_i} + \int_{H^2}\frac{f_{CookTorrances}}{F}(1-\cos\theta_i)^5\cos\theta_i d\omega_i$$

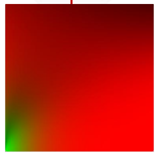
LUT



$$\approx F_0 \cdot A + B \approx F_0 * LUT.r + LUT.g$$

Quick Shading with Precomputation

$$L(\omega_o) = \boxed{k_d^* c \cdot IrranianceMap(n)} + \boxed{PreFiltered(R,\alpha) \cdot (F_0 * LUT.r \; + \; LUT.g)}$$



Shading PBR with IBL

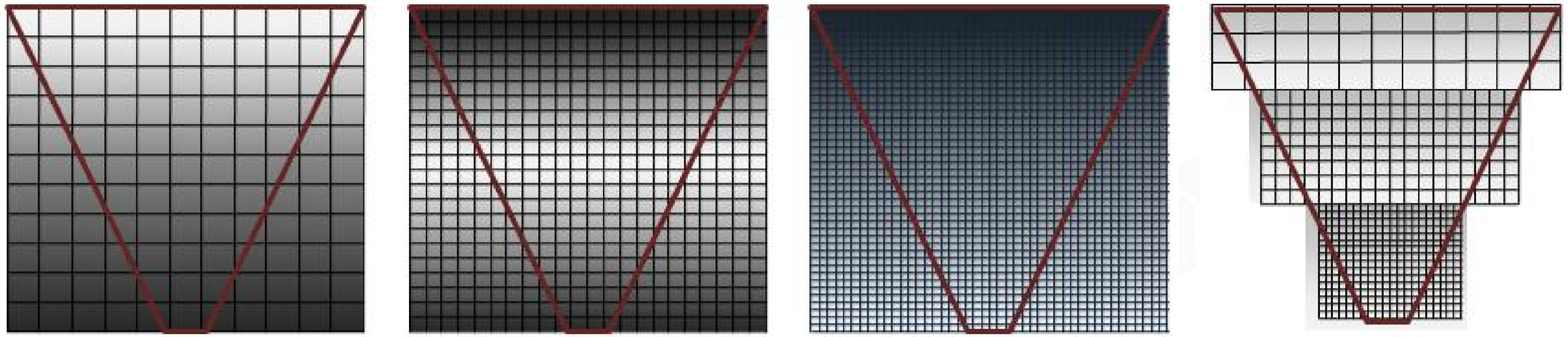
IBL OFF

IBL ON

Classic Shadow Solution

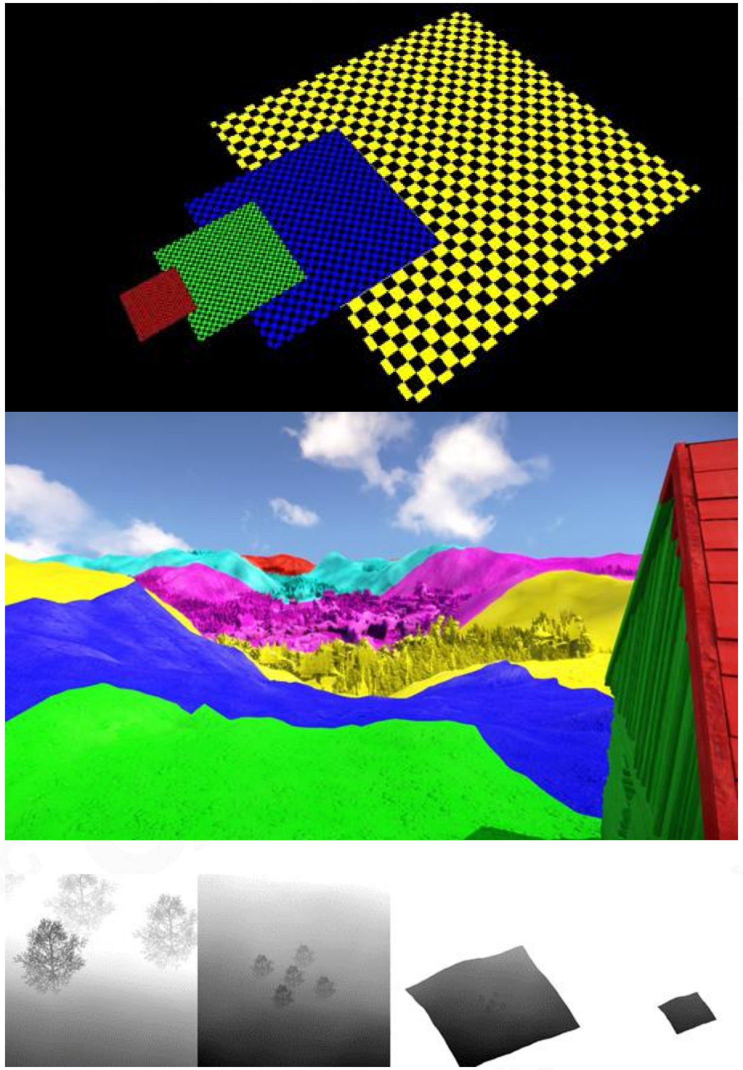
Big World and Cascade Shadow

- Partition the frustum into multiple frustums
- A shadow map is rendered for each sub frustum
- The pixel shader then samples from the map that most closely matches the required resolution



Steps of Cascade Shadow

```
splitFrustumToSubfrusta();
calculateOrthoProjectionsForEachSubfrustum();
renderShadowMapForEachSubfrustum();
renderScene();

vs_main()
{
    calculateWorldPosition()
    ...
}

ps_main()
{
    transformWorldPositionsForEachProjections()
        sampleAllShadowMaps()
        compareDepthAndLightingPixel()
        ....
}
```

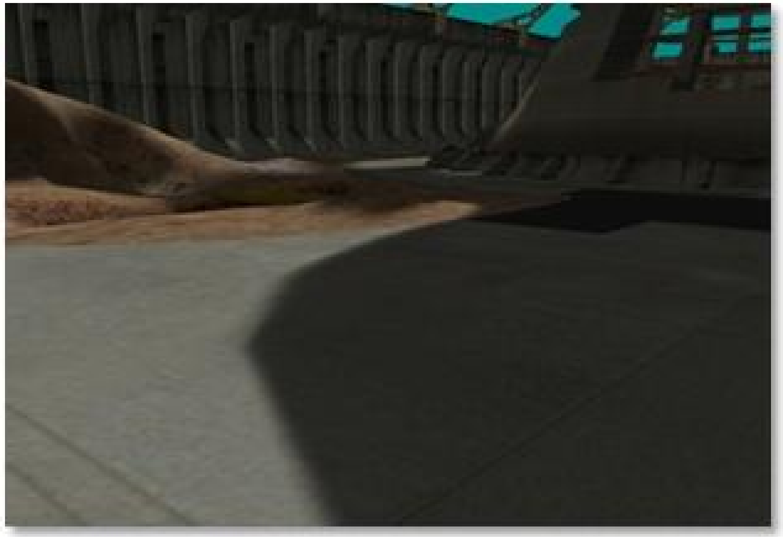


Blend between Cascade Layers

1. A visible seam can be seen where cascades overlap

2. between cascade layers because the resolution does not match

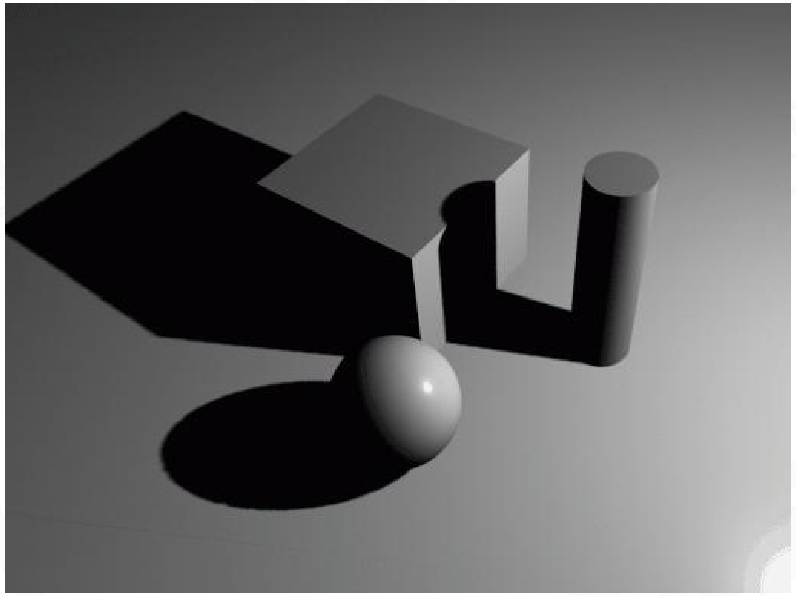
3. The shader then linearly interpolates between the two values based on the pixel's location in the blend band



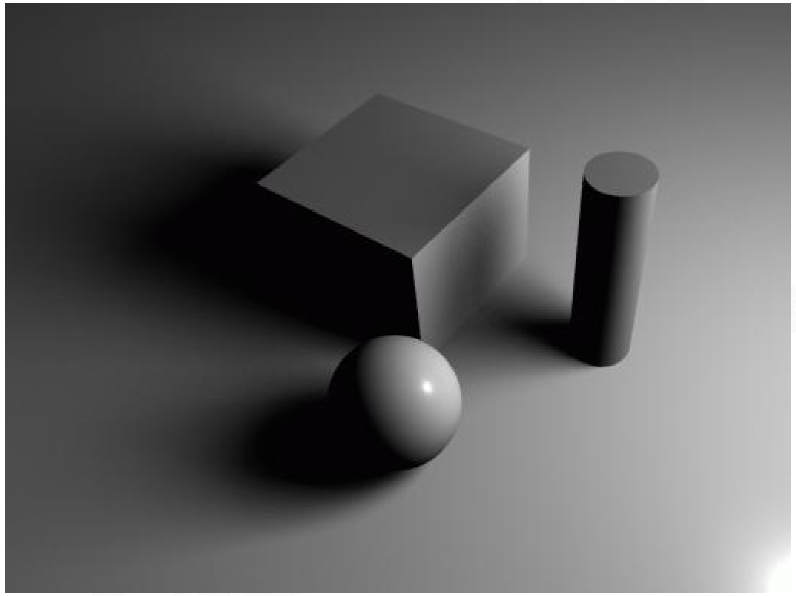
Pros and Cons of Cascade Shadow

- **Pros**
  - best way to prevalent errors with shadowing: perspective aliasing
  - fast to generate depth map, 3x up when depth writing only
  - provide fairly good results
- **Cons**
  - Nearly impossible to generate high quality area shadows
  - No colored shadows. Translucent surfaces cast opaque shadows

Hard Shadow vs Realistic Shadow



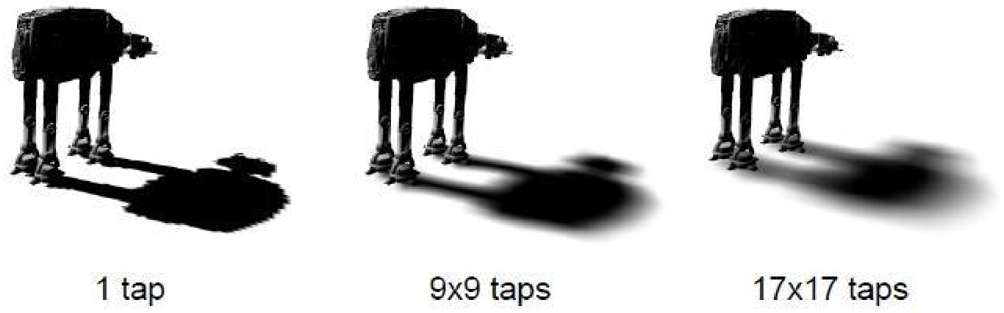
Hard Shadow



Realistic Shadow

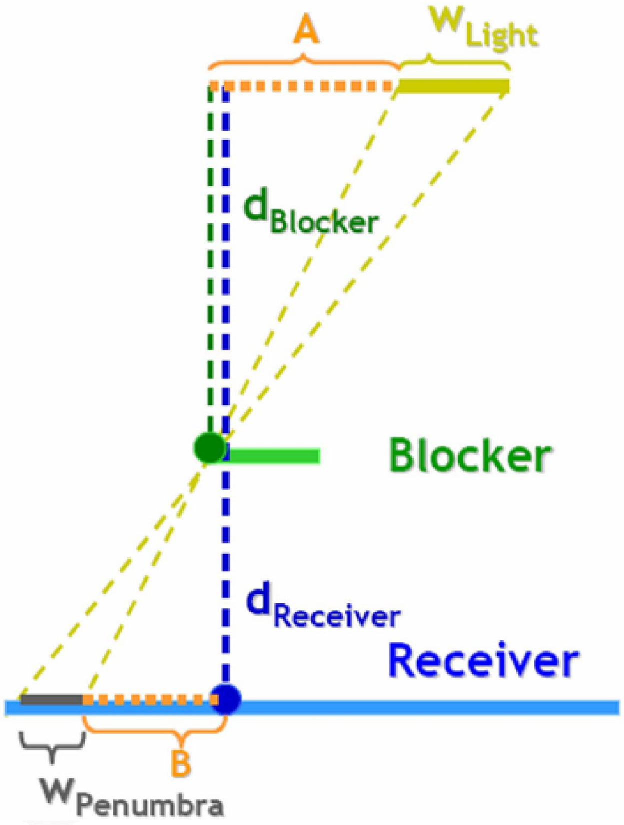
PCF - Percentage Closer Filter

- **Target problem**
  - The shadows that result from shadow mapping aliasing is serious
- **Basic idea**
  - Sample from the shadow map around the current pixel and compare its depth to all the samples
  - By averaging out the results we get a smoother line between light and shadow



PCSS - Percentage Closer Soft Shadow

- Target problem
  - Suffers from aliasing and under sampling artifacts
- Basic idea
  - Search the shadow map and average the depths that are closer to the light source
  - Using a parallel planes approximation



$$w_{Penumbra} = \frac{(d_{Receiver} - d_{Blocker}) \cdot w_{Light}}{d_{Blocker}}$$

Variance Soft Shadow Map

- **Target problem**
  - Rendering plausible soft shadow in real-time
- **Basic idea**
  - Based on Chebyshev's inequality, using the average and variance of depth, we can approximate the percentage of depth distribution directly instead of comparing a single depth to a particular region(PCSS)

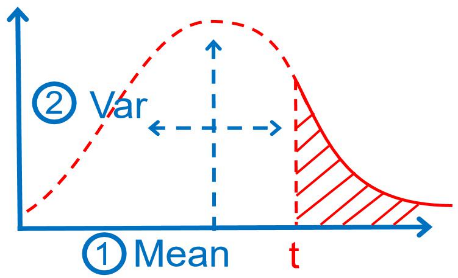
$$\mu = E(x)$$
$$\sigma^2 = E(x^2) - E(x)^2$$

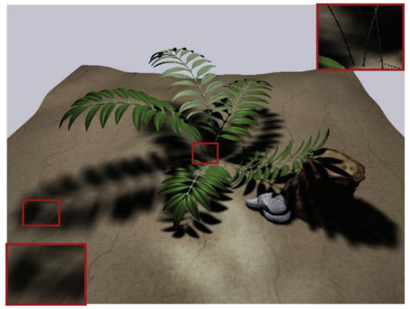
$$P(x > t) \leq \frac{\sigma^2}{\sigma^2 + (t-\mu)^2}$$

μ : mean

σ^2 : variance

Doesn't even assume Gaussian distribution!





VSSM (148 fps)

PCSS (10 fps)

Summary of Popular AAA Rendering

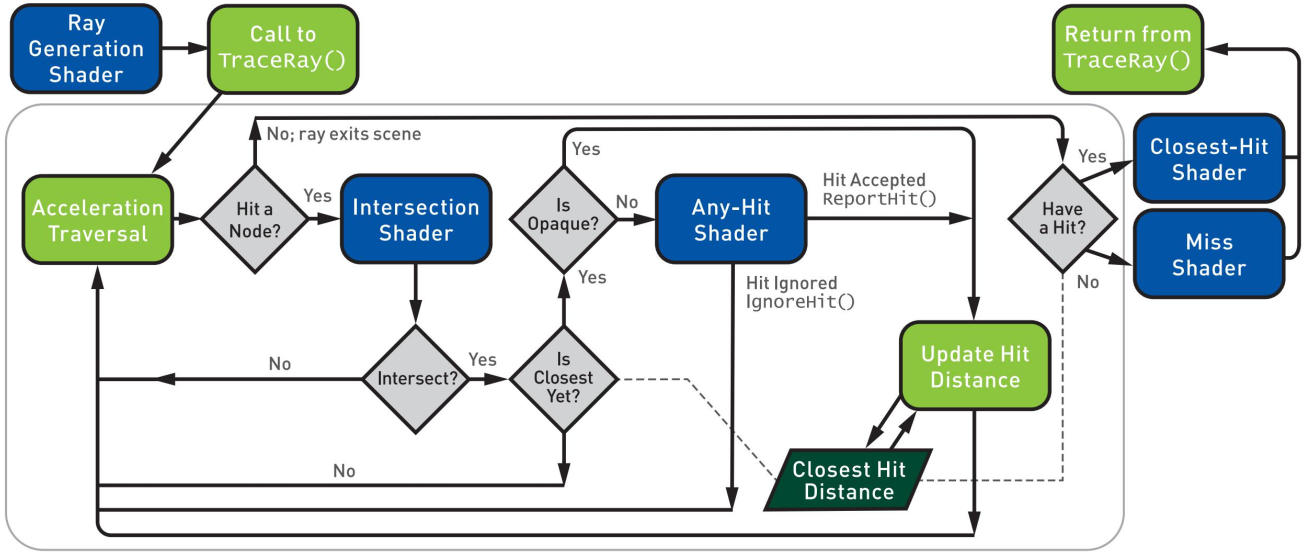
- Lightmap + Light probe
- PBR + IBL
- Cascade shadow + VSSM

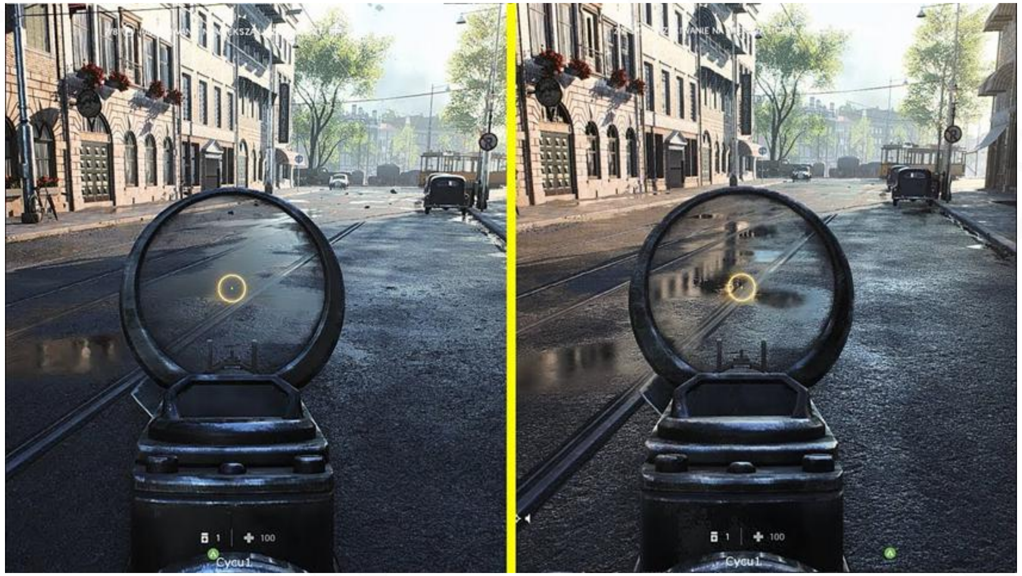
Moving Wave of High Quality

Quick Evolving of GPU

- More flexible new shader model
  - Compute shader
  - Mesh shader
  - Ray-tracing shader
- High performance parallel architecture
  - Warp or wave architecture
- Fully opened graphics API
  - DirectX 12 and Vulkan

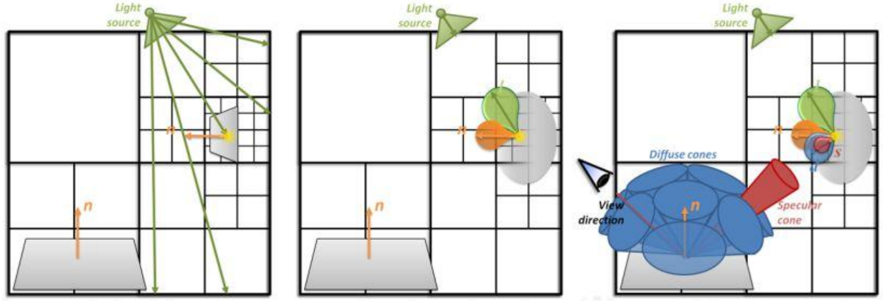
Real-Time Ray-Tracing on GPU





Real-Time Global Illumination

Screen-space GI
SDF Based GI
Voxel-Based GI （SVOGI/VXGI)
RSM / RTX GI



GI Off

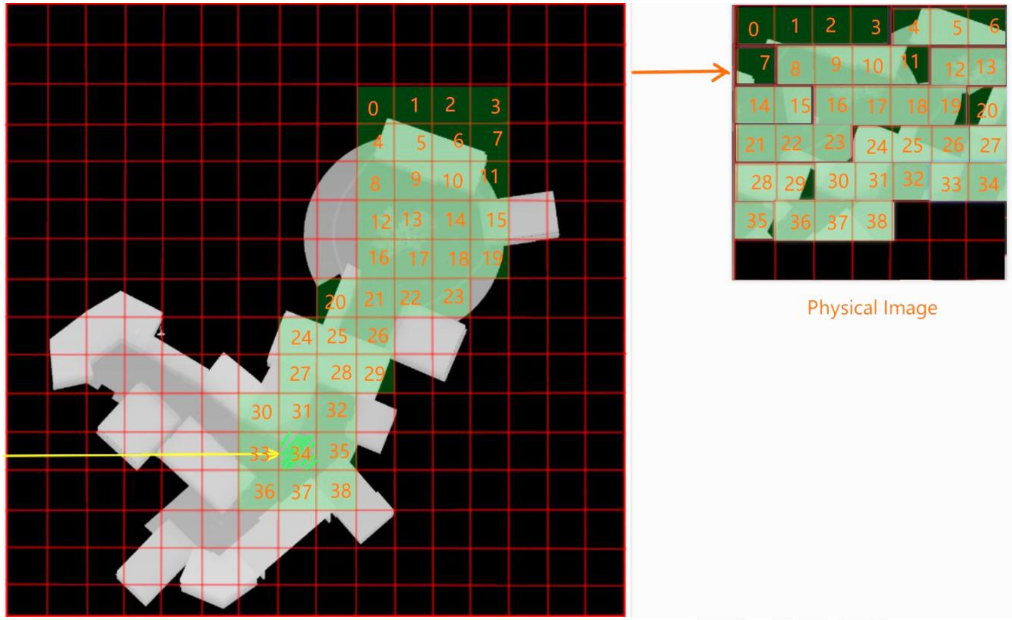
GI On

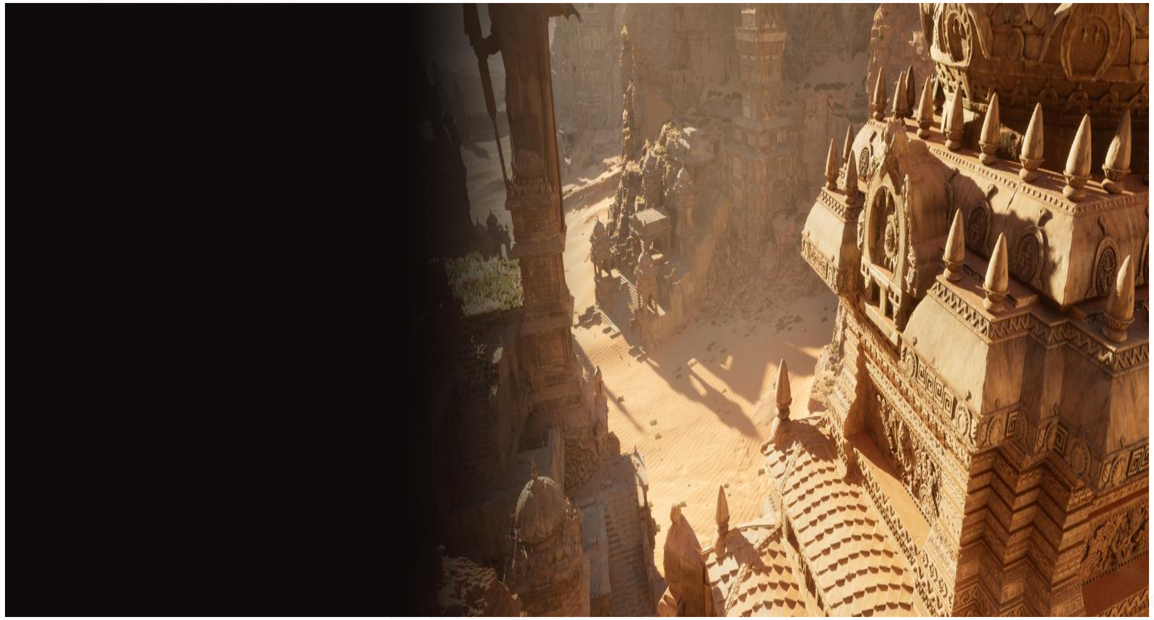
More Complex Material Model

BSDF (Strand-based hair)

BSSRDF

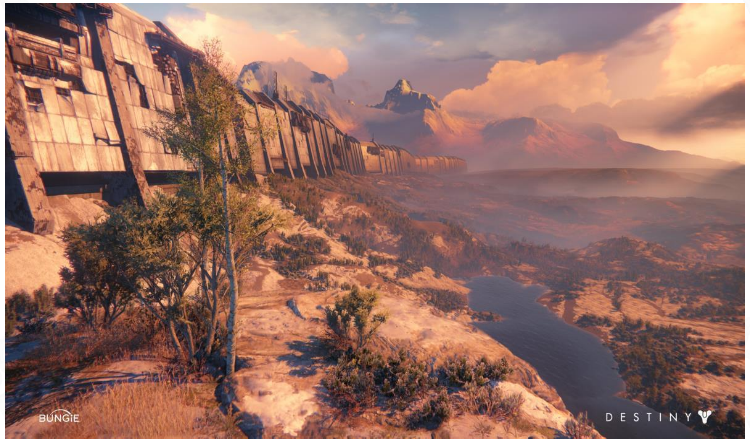
Virtual Shadow Maps

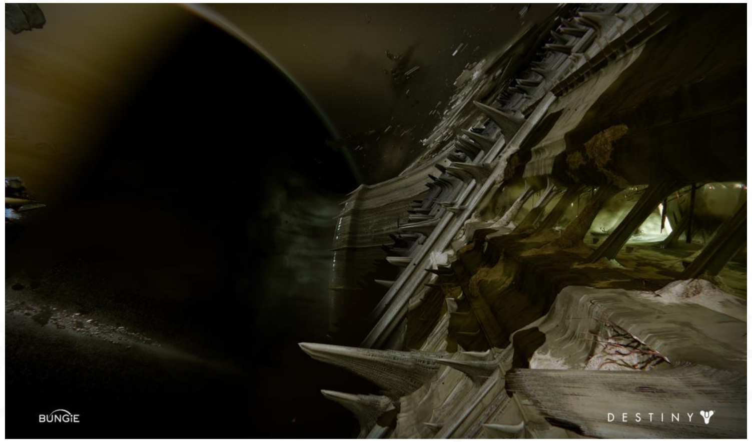




Shader Management

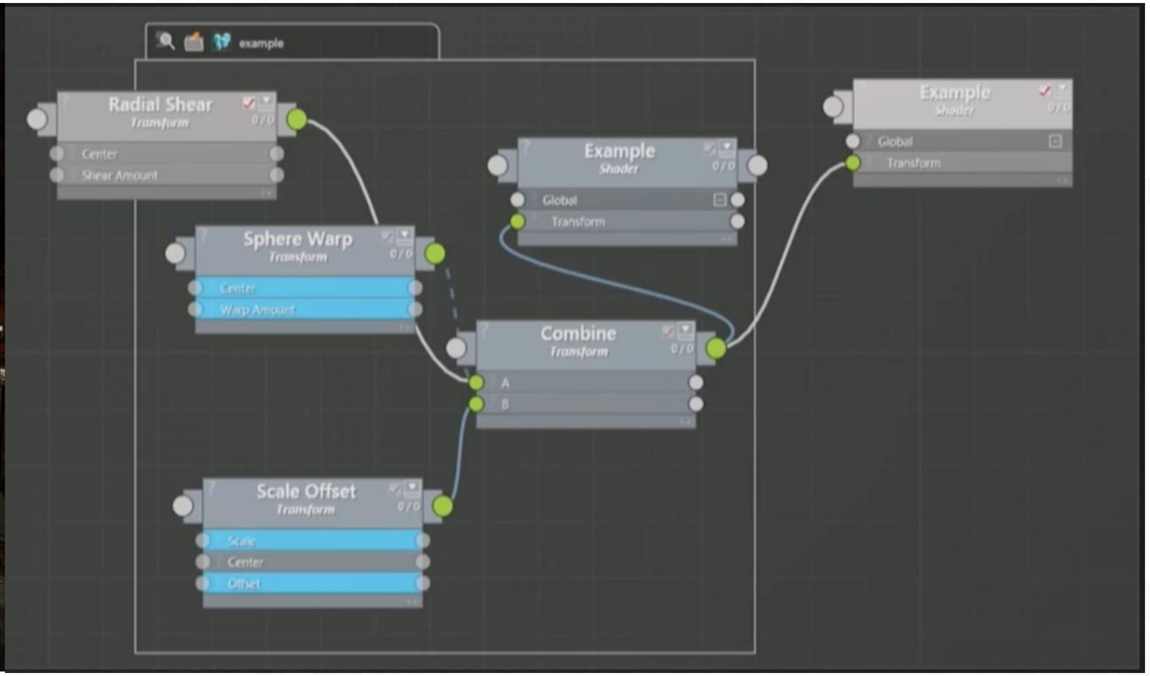
Ocean of Shaders





Blow of Shaders

Artist Create Infinite More Shaders



Uber Shader and Variants

A combination of shader for all possible light types, render passes and material types

- Shared many state and codes
- Compile to many variant short shaders by pre-defined macro

```
// sky light
#if ENABLE_SKY_LIGHT
    #if  MATERIAL_TWOSIDED && LQ_TEXTURE_LIGHTMAP
    if (NoL == 0)
    {
    #endif

    #if MATERIAL_SHADINGMODEL_SINGLELAYERWATER
        ShadingModelContext.WaterDiffuseIndirectLuminance += SkyDiffuseLighting;
    #endif

        Color += SkyDiffuseLighting * half3(ResolvedView.SkyLightColor.rgb) *
            ShadingModelContext.DiffuseColor * MaterialAO;

    #if MATERIAL_TWOSIDED && LQ_TEXTURE_LIGHTMAP
    }
    #endif
#endif
```

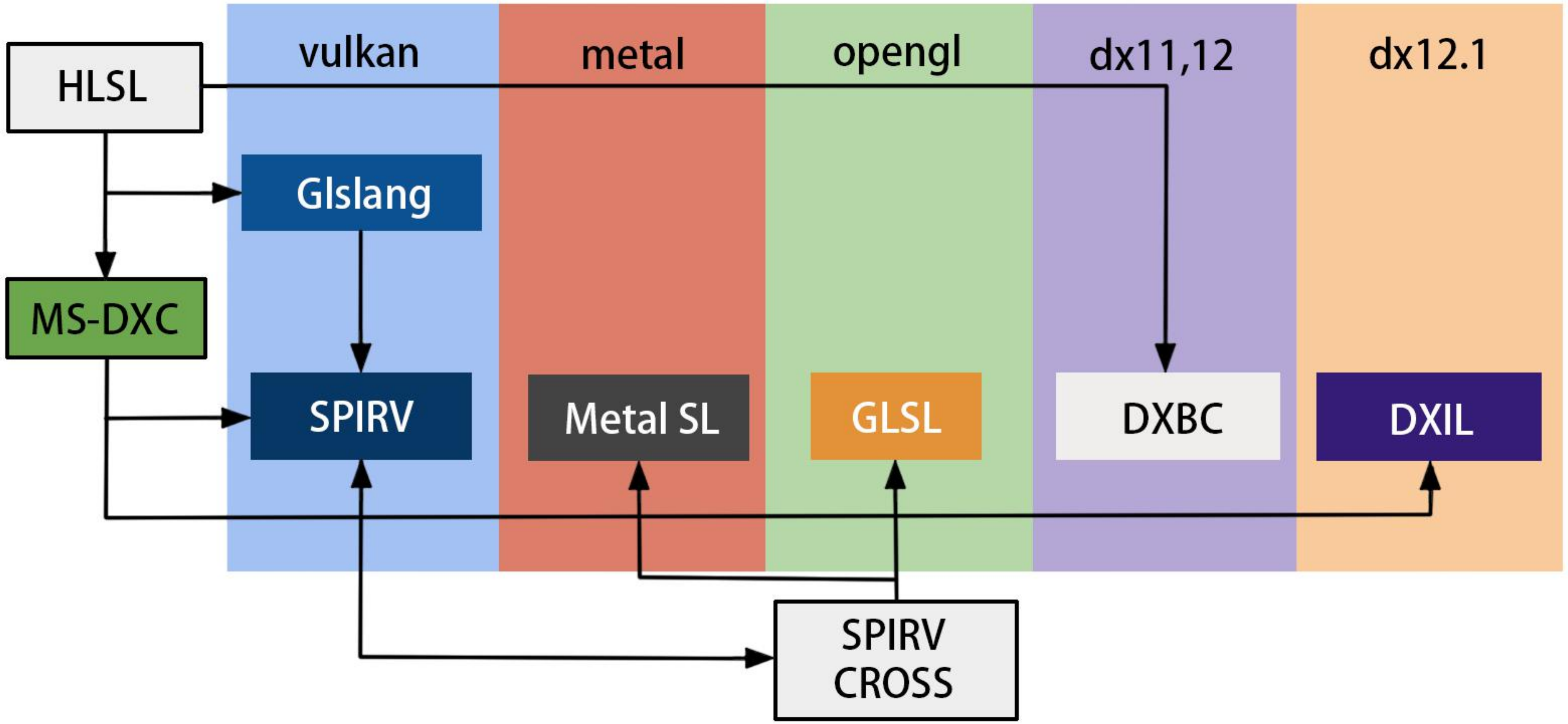
Shader Variants Example In Real Game

opacue_distortion_deferred_d3d11_ps_0.shader.ast	2022/4/11 10:52	AST 文件	15 KB
opacue_distortion_deferred_d3d11_ps_0_deferred.shader.ast	2022/4/11 10:52	AST 文件	15 KB
opacue_distortion_deferred_d3d11_ps_1.shader.ast	2022/4/11 10:52	AST 文件	16 KB
opacue_distortion_deferred_d3d11_ps_1_deferred.shader.ast	2022/4/11 10:52	AST 文件	16 KB
opacue_distortion_deferred_instance_d3d11_ps_0.shader.ast	2022/4/11 10:52	AST 文件	15 KB
opacue_distortion_deferred_instance_d3d11_ps_0_deferred.shader.ast	2022/4/11 10:52	AST 文件	15 KB
opacue_distortion_deferred_instance_d3d11_ps_1.shader.ast	2022/4/11 10:52	AST 文件	16 KB
opacue_distortion_deferred_instance_d3d11_ps_1_deferred.shader.ast	2022/4/11 10:52	AST 文件	16 KB
opacue_distortion_depth_only_d3d11_ps_0.shader.ast	2022/4/11 10:52	AST 文件	2 KB
opacue_distortion_depth_only_d3d11_ps_0_deferred.shader.ast	2022/4/11 10:52	AST 文件	2 KB
opacue_distortion_depth_only_d3d11_ps_1.shader.ast	2022/4/11 10:52	AST 文件	3 KB
opacue_distortion_depth_only_d3d11_ps_1_deferred.shader.ast	2022/4/11 10:52	AST 文件	4 KB
opacue_distortion_forward_simple_d3d11_ps_0.shader.ast	2022/4/11 10:52	AST 文件	9 KB
opacue_distortion_forward_simple_d3d11_ps_0_deferred.shader.ast	2022/4/11 10:52	AST 文件	9 KB
opacue_distortion_forward_simple_d3d11_ps_1.shader.ast	2022/4/11 10:52	AST 文件	9 KB
opacue_distortion_forward_simple_d3d11_ps_1_deferred.shader.ast	2022/4/11 10:52	AST 文件	9 KB
opacue_deferred_beyond_distance_d3d11_ps_1117011_deferred.shader.ast	2022/4/11 10:39	AST 文件	13 KB
opacue_deferred_beyond_distance_d3d11_ps_1117100.shader.ast	2022/4/11 10:39	AST 文件	11 KB
opacue_deferred_beyond_distance_d3d11_ps_1117101.shader.ast	2022/4/11 10:39	AST 文件	13 KB
opacue_deferred_beyond_distance_d3d11_ps_1117101_deferred.shader.ast	2022/4/11 10:39	AST 文件	13 KB
opacue_deferred_beyond_distance_d3d11_ps_1117110.shader.ast	2022/4/11 10:39	AST 文件	13 KB
opacue_deferred_beyond_distance_d3d11_ps_1117110_deferred.shader.ast	2022/4/11 10:39	AST 文件	13 KB
opacue_deferred_beyond_distance_d3d11_ps_1117111.shader.ast	2022/4/11 10:39	AST 文件	14 KB
opacue_deferred_beyond_distance_d3d11_ps_1117111_deferred.shader.ast	2022/4/11 10:39	AST 文件	14 KB
opacue_deferred_d3d11_ps_1117101.shader.ast	2022/4/11 10:39	AST 文件	11 KB
opacue_deferred_d3d11_ps_1117101_deferred.shader.ast	2022/4/11 10:39	AST 文件	11 KB
opacue_deferred_d3d11_ps_1117110.shader.ast	2022/4/11 10:39	AST 文件	11 KB
opacue_deferred_d3d11_ps_1117110_deferred.shader.ast	2022/4/11 10:39	AST 文件	11 KB
opacue_deferred_d3d11_ps_1117111.shader.ast	2022/4/11 10:39	AST 文件	13 KB
opacue_deferred_d3d11_ps_1117111_deferred.shader.ast	2022/4/11 10:39	AST 文件	13 KB
opacue_deferred_instance_d3d11_ps_1117011.shader.ast	2022/4/11 10:39	AST 文件	12 KB
opacue_deferred_instance_d3d11_ps_1117100_deferred.shader.ast	2022/4/11 10:39	AST 文件	10 KB
opacue_deferred_instance_d3d11_ps_1117101.shader.ast	2022/4/11 10:39	AST 文件	11 KB
opacue_deferred_instance_d3d11_ps_1117101_deferred.shader.ast	2022/4/11 10:39	AST 文件	11 KB
opacue_deferred_instance_d3d11_ps_1117110.shader.ast	2022/4/11 10:39	AST 文件	11 KB
opacue_deferred_instance_d3d11_ps_1117110_deferred.shader.ast	2022/4/11 10:39	AST 文件	11 KB
opacue_deferred_instance_d3d11_ps_1117111.shader.ast	2022/4/11 10:39	AST 文件	13 KB
opacue_deferred_instance_d3d11_ps_1117111_deferred.shader.ast	2022/4/11 10:39	AST 文件	13 KB
opacue_depth_only_d3d11_ps_1117100.shader.ast	2022/4/11 10:39	AST 文件	4 KB
opacue_depth_only_d3d11_ps_1117101_deferred.shader.ast	2022/4/11 10:39	AST 文件	4 KB
opacue_depth_only_d3d11_ps_1117110.shader.ast	2022/4/11 10:39	AST 文件	4 KB
opacue_depth_only_d3d11_ps_1117110_deferred.shader.ast	2022/4/11 10:39	AST 文件	4 KB
opacue_depth_only_d3d11_ps_1117111.shader.ast	2022/4/11 10:39	AST 文件	4 KB
opacue_depth_only_d3d11_ps_1117111_deferred.shader.ast	2022/4/11 10:39	AST 文件	4 KB
opacue_forward_scene_d3d11_ps_1117010_deferred.shader.ast	2022/4/11 10:39	AST 文件	19 KB
opacue_forward_scene_d3d11_ps_1117011.shader.ast	2022/4/11 10:39	AST 文件	19 KB

particle_complex_particle_d3d11_vs_1101313111.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1101314000.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1101314001.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1101314010.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1101314011.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1101314100.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1101314101.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1101314110.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1101314111.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110000000.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110000001.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110000010.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110000011.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110000100.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110000101.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110000110.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110000111.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110001000.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110001001.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110001010.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110001011.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110001100.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110001101.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110001110.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110001111.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110002000.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110002001.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110002010.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110002011.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110002100.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110002101.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110002110.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110002111.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110003000.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110003001.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110003010.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110003011.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110003100.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110003101.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110003110.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110003111.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110004000.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110004001.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110004010.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110004011.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110004100.shader.ast	2022/4/11 10:58	AST 文件	7 KB
particle_complex_particle_d3d11_vs_1110004101.shader.ast	2022/4/11 10:58	AST 文件	7 KB

165 uber shader generated 75,536 shaders for runtime

Cross Platform Shader Compile

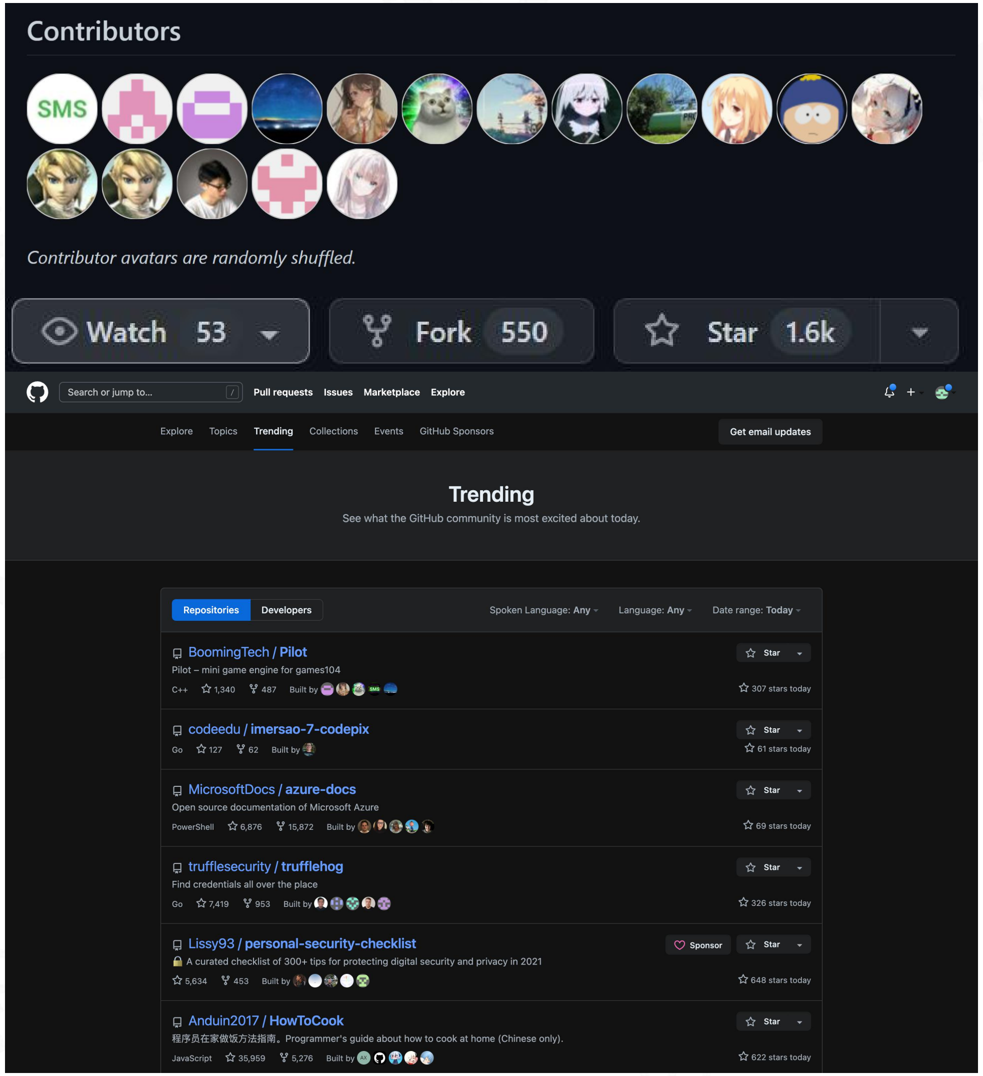


Shader Cross-Compiler Flow

Pilot Engine

- Thanks to the contributors
- The latest version of pilot already supports M1

PILOT
Game engine



Labor Day Holiday Arrangement

- Lecture 8 on May 2 will be postponed to May 9
- All subsequent classes will be postponed

Lecture 05 Contributor

- 一将
- 光哥
- 炯哥
- 玉林
- 小老弟
- 建辉
- 爵爷
- Jason
- 砚书
- BOOK
- MANDY
- 俗哥
- 金大壮
- Leon
- 梨叔
- Shine
- 邓导
- Judy
- QIUU
- C佬
- 阿乐
- 阿熊
- CC
- 大喷

Q&A

Enjoy ;)
Coding

Course Wechat

Follow us for further information