Computer Graphics:

Introduction to Blender Materials

By: Nick | ACM at UCSD

Before we begin...

Download Blender:

https://www.blender.org/download/

Rendering in Movies

How do animation studios render cinematic-quality images like these?

Coco | Pixar 2017

Raytracing Crash Course

The answer is: raytracing

We covered this a little bit at the end of the last workshop, but we will be going much more in-depth in this workshop

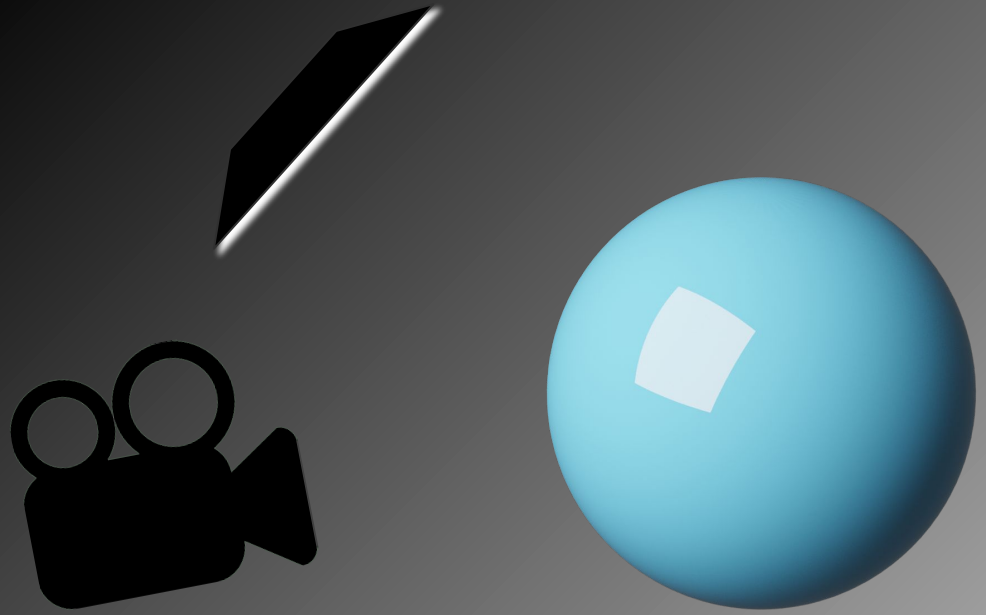
Specifically, we will be covering how to describe a material, or how light interacts with the surface of an object.

- ➔ Is it shiny, diffuse, skin-like, etc?
- ➔ Also sorry to those in 167/168, these next few slides will be review

Light Transport (in real life)

Take a simple scene with a light, a sphere, and a camera:

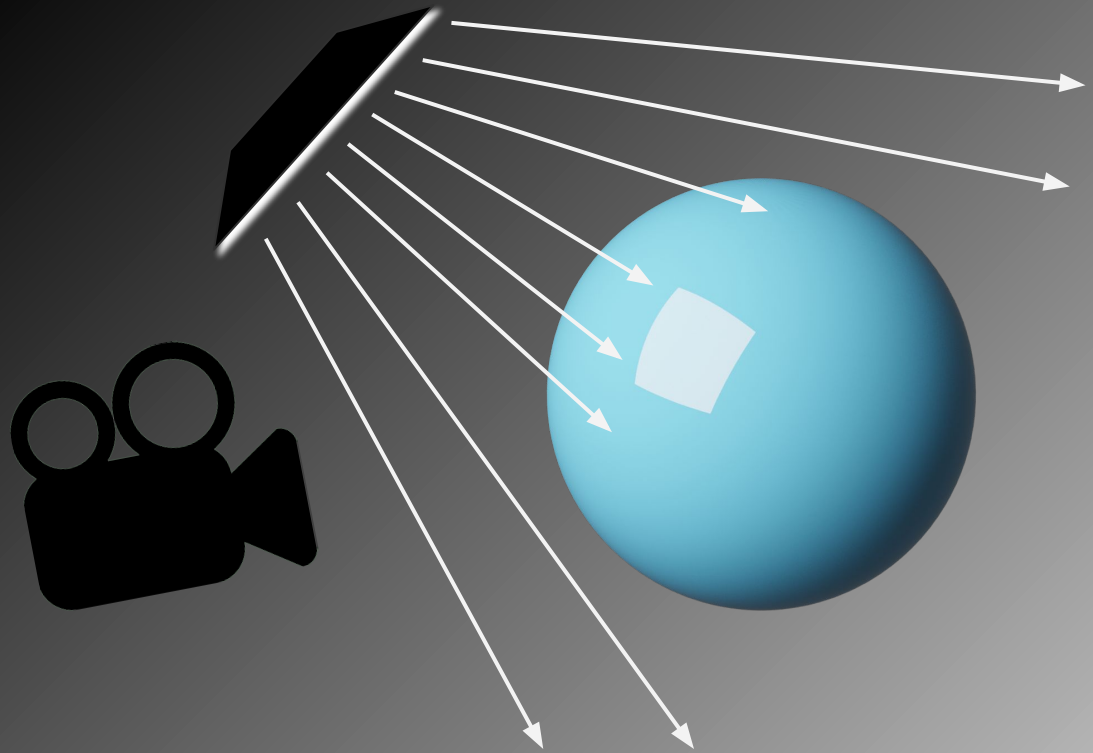
How does the camera capture this image in the real world?



Light Transport (in real life)

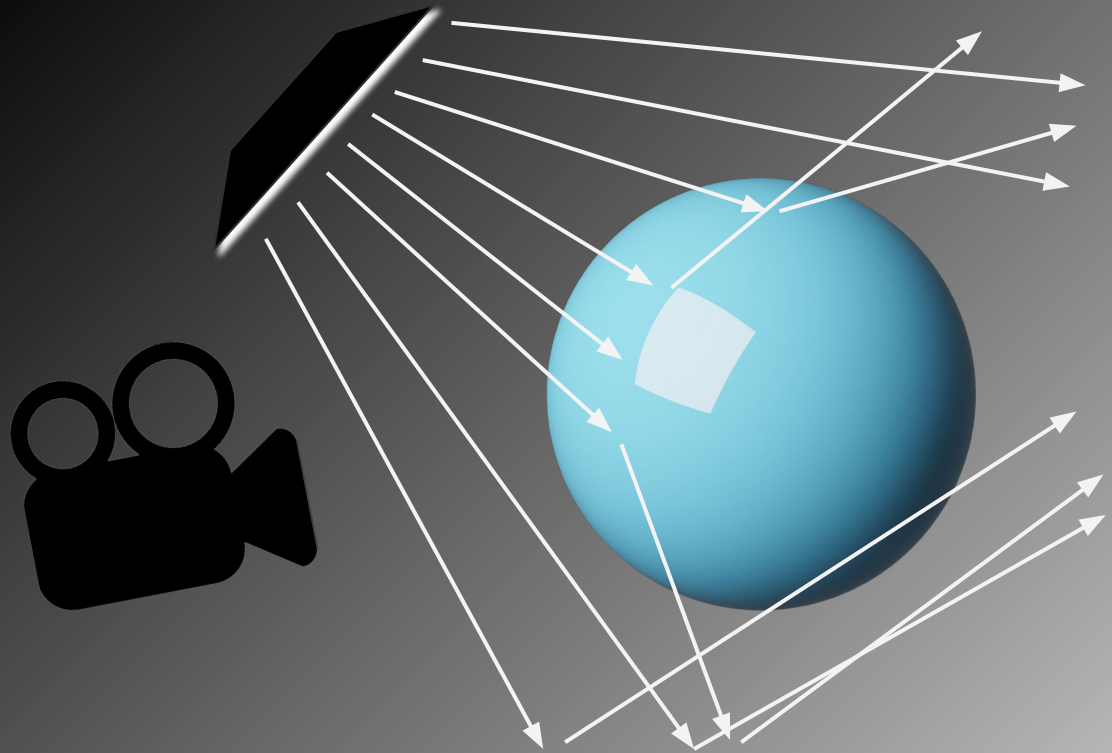
The emissive surface illuminates the scene:

Some photons hit a surface; some do not and don’t contribute much to anything



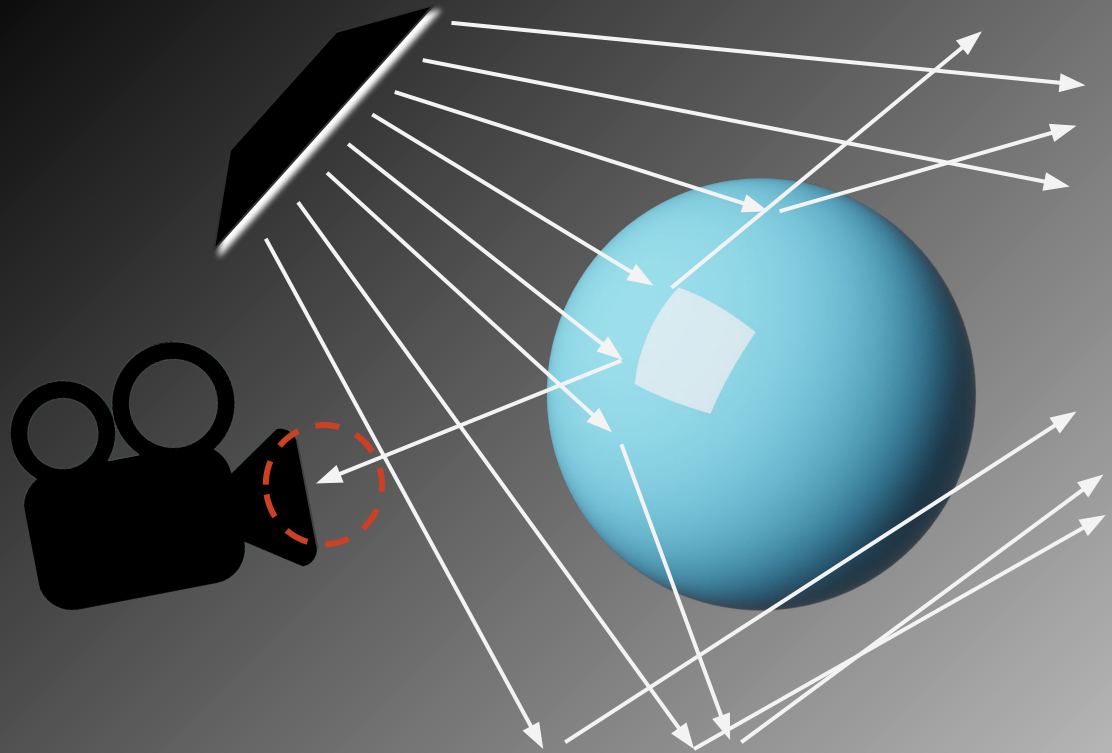
Light Transport (in real life)

These photons then get reflected, refracted, or absorbed:



Light Transport (in real life)

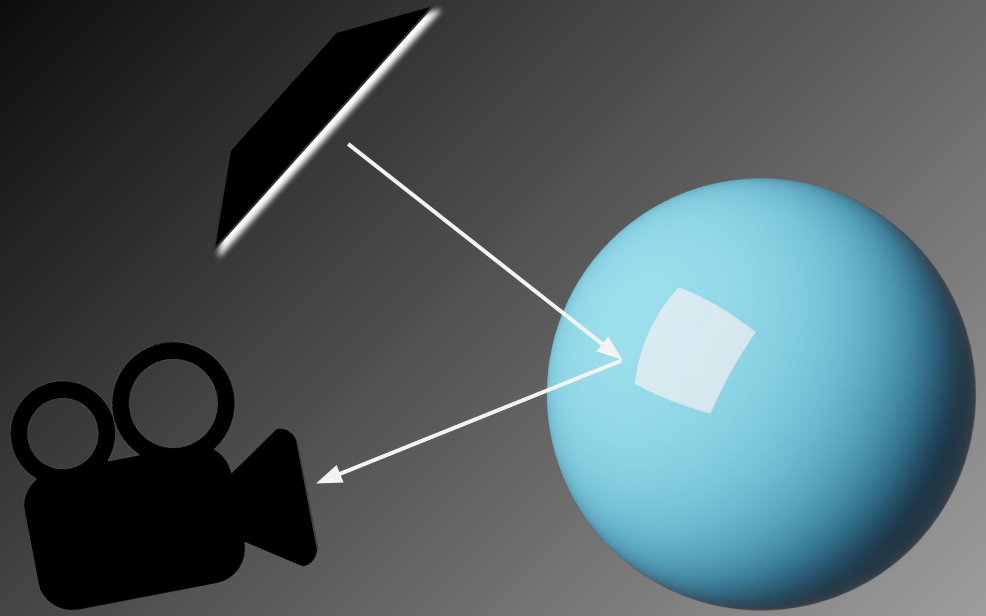
Until some eventually hit
the camera film or sensor:



Light Transport (in real life)

Obviously, this would take a while to compute

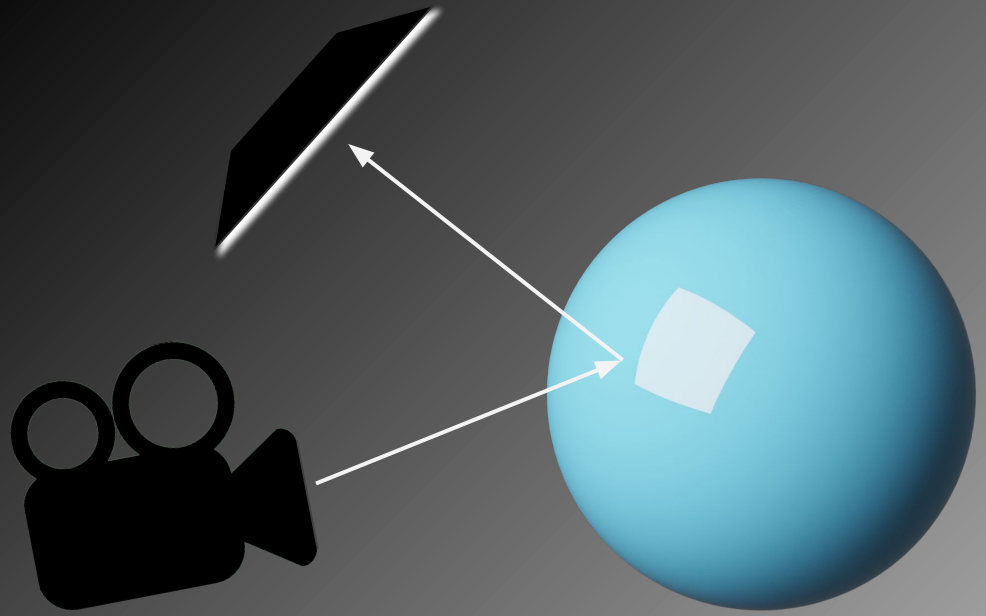
You would likely have to compute billions of rays to have a decent image



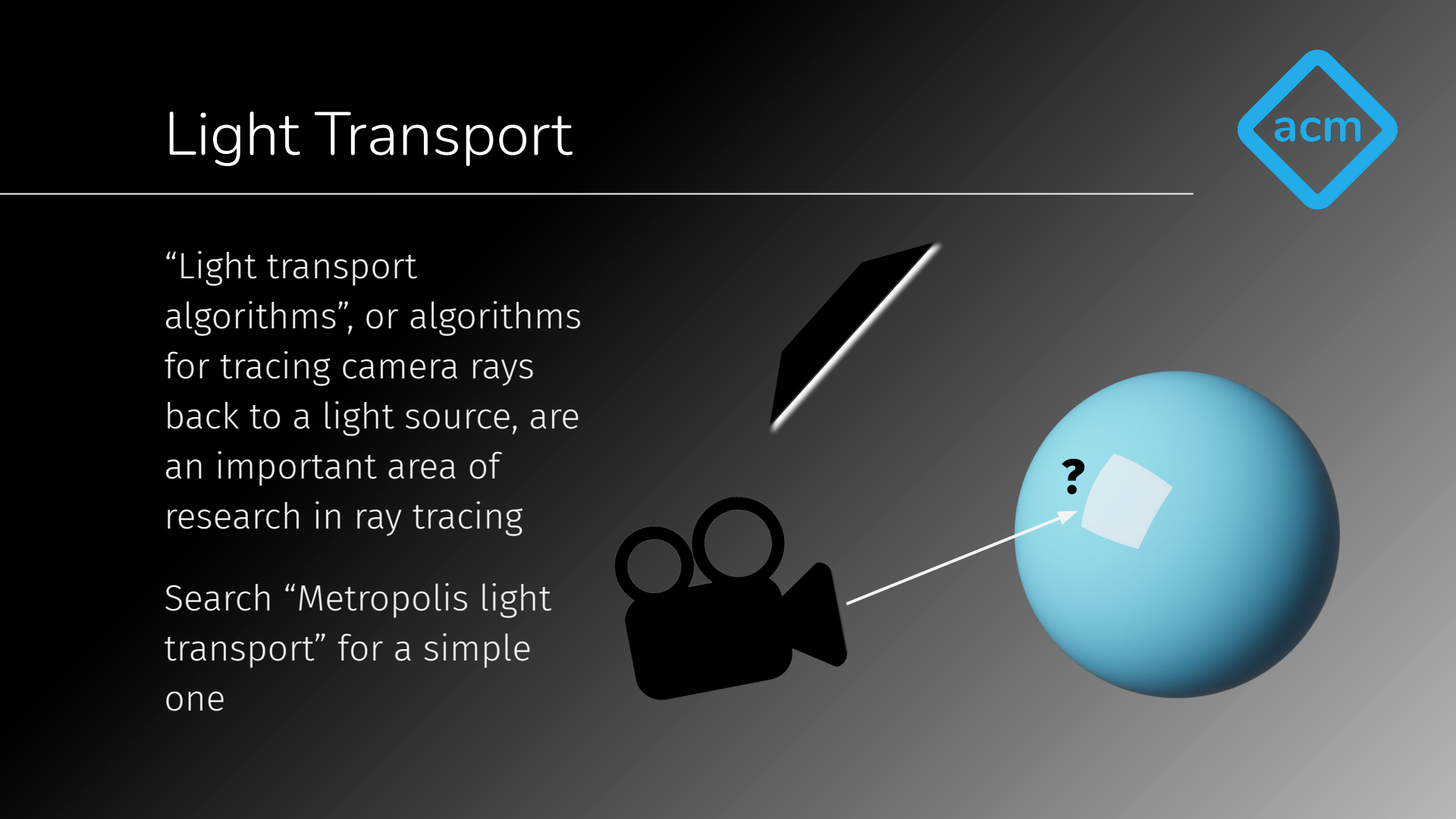
Light Transport

Instead, we try this in reverse:

This way, we only need to compute the color for surfaces visible to the camera



Light Transport



"Light transport algorithms", or algorithms for tracing camera rays back to a light source, are an important area of research in ray tracing

Search "Metropolis light transport" for a simple one

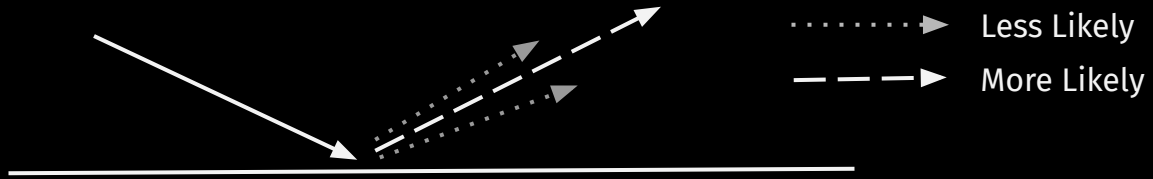
Defining Materials

Now that we know how ray tracing works, we need to figure out a way to define how light rays should interact with an object

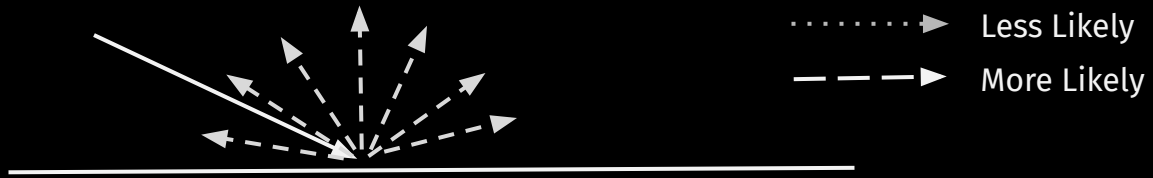
Mathematically, we can define a distribution of possible ways the ray can be scattered given a few inputs like incident light angle, surface normal (tangent direction), etc

Defining Materials

For example, a shiny surface might have a distribution like this:

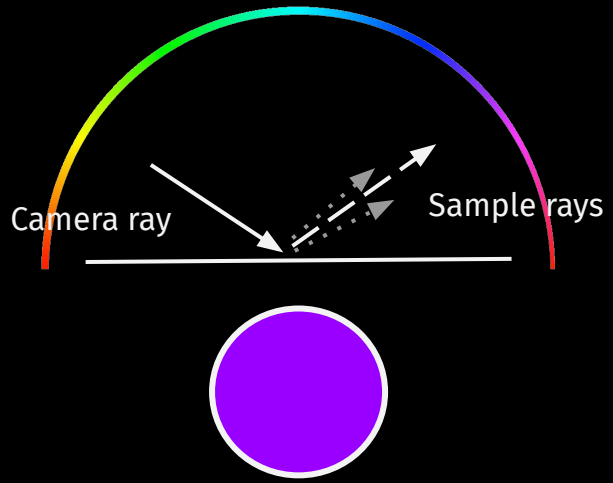


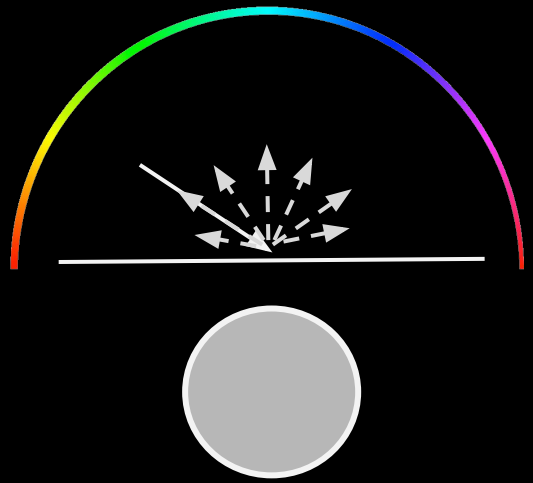
While a non-shiny (or diffuse) surface might have a more even distribution:



Defining Materials

When reversing the rays to come from the camera, we can multiply this distribution with the incoming radiance, or light, at a point to compute the final color of the surface:





Defining Materials

The distribution also encodes the color of an object, so for example a blue diffuse object would simply be biased to take in more blue colors, even if it isn’t particularly reflective in the direction the blue is in

The Rendering Equation

What I just explained can be described using the infamous "rendering equation":

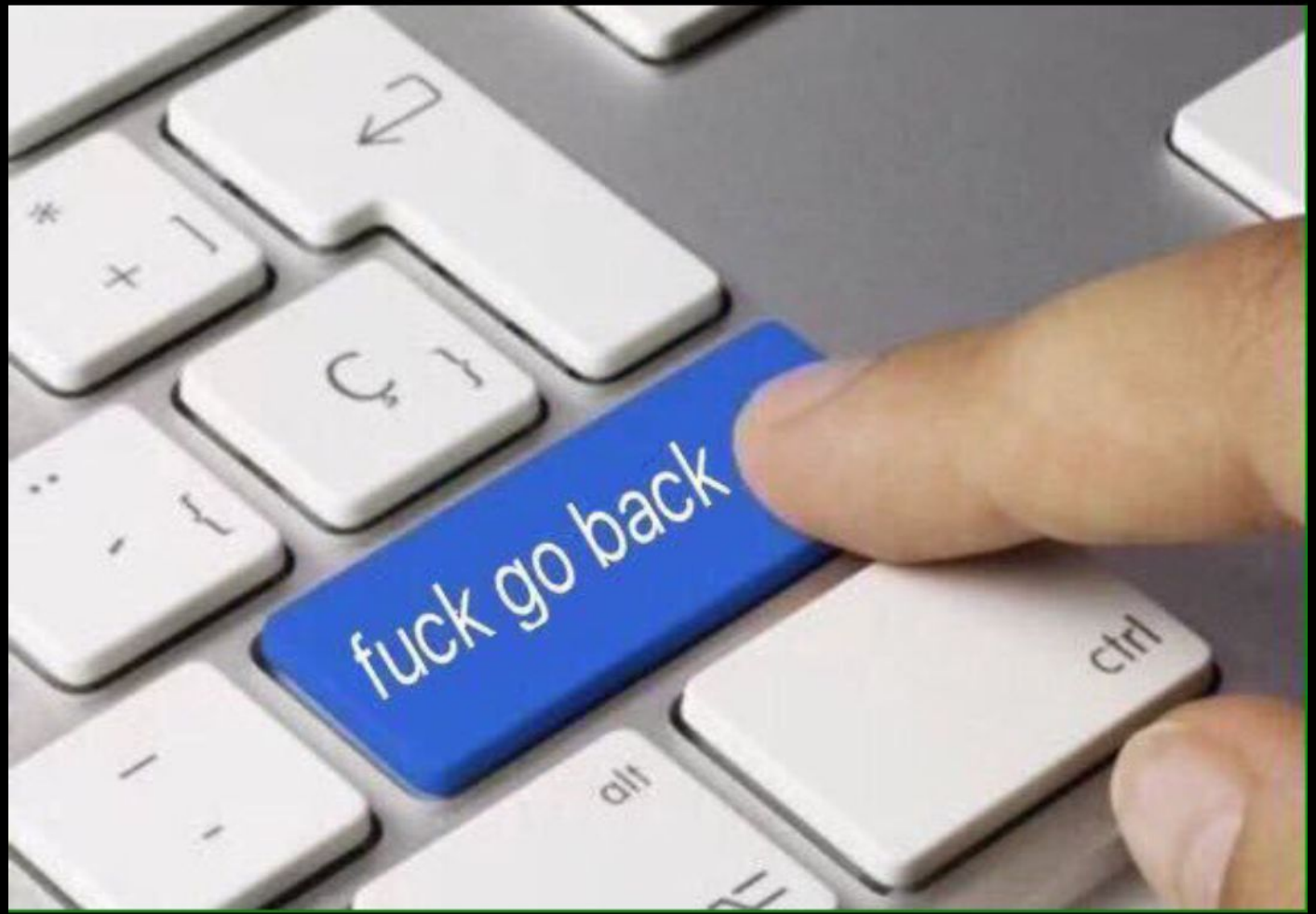
$$L_o(x, \omega_o, \lambda, t) =$$

$$L_e(x, \omega_o, \lambda, t) + \int_{\Omega} f_r(x, \omega_i, \omega_o, \lambda, t)\, L_i(x, \omega_i, \lambda, t)\, (\omega_i \cdot n)\, d\omega_i$$

Emissive radiance

Bidirectional Reflectance Distribution Function

Incoming radiance


fuck go back
ctrl

Getting into Blender

What is Blender?

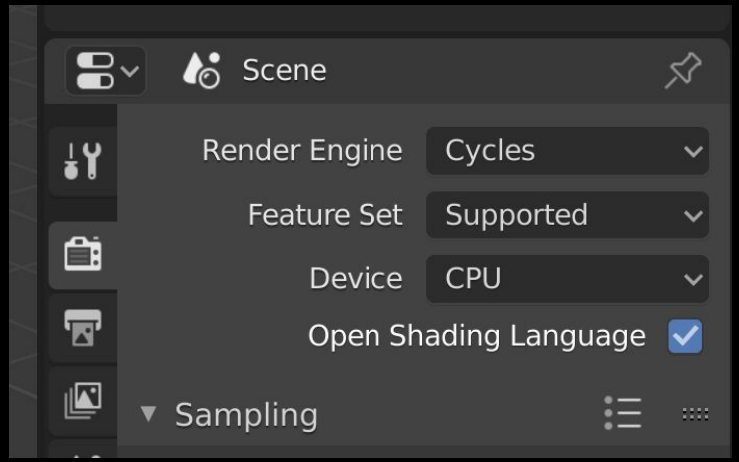
“Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, video editing and 2D animation pipeline”

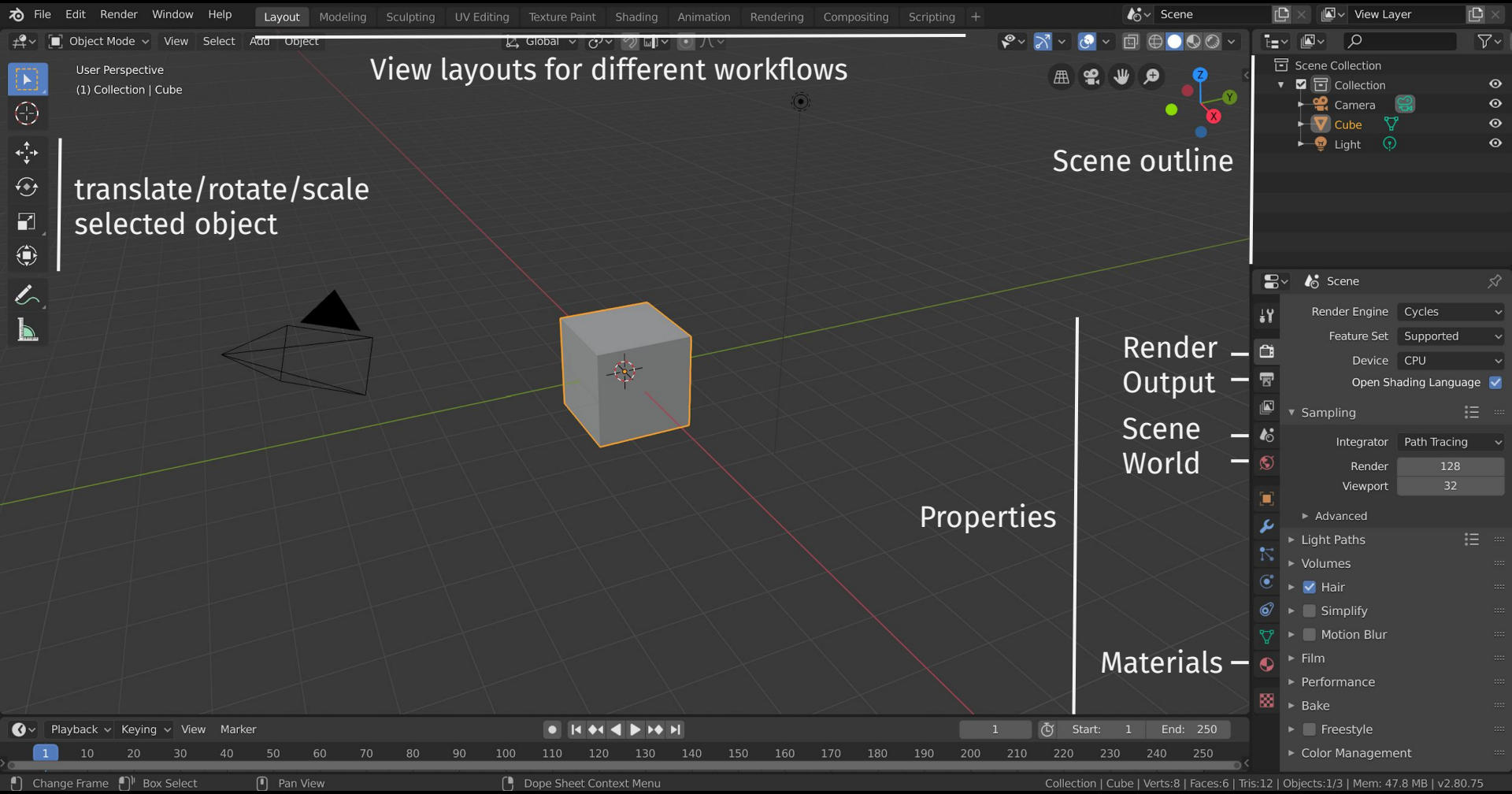
https://www.blender.org/features/

Getting into Blender

Ok, let's get into some Blender

First, switch to cycles, using cpu, enable OSL

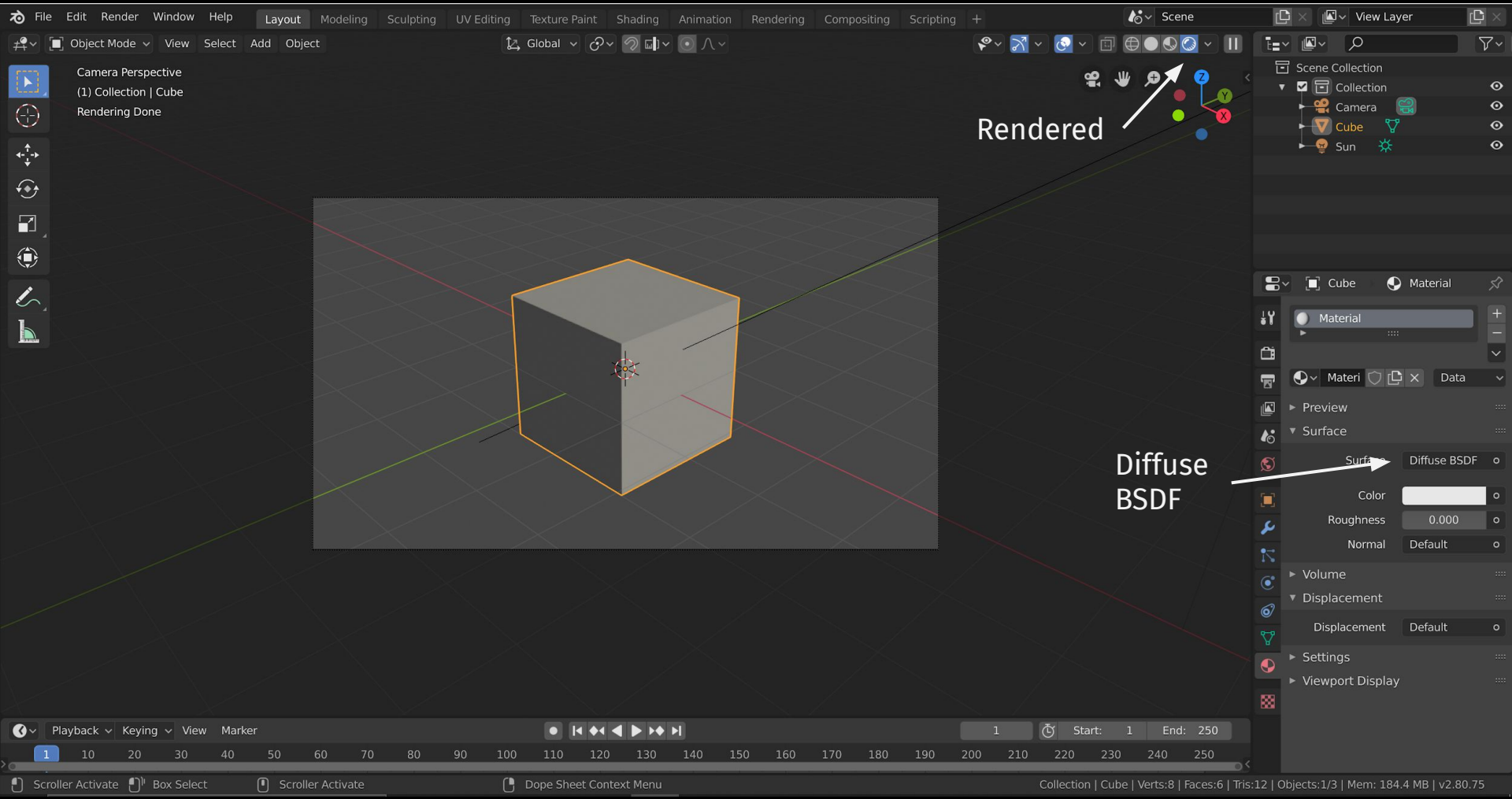



File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Scene
View Layer
Object Mode View Select Add Object
Global
User Perspective
(1) Collection | Cube
View layouts for different workflows
translate/rotate/scale
selected object
Scene outline
Scene Collection
Collection
Camera
Cube
Light
Render
Output
Scene
World
Properties
Materials
Render Engine Cycles
Feature Set Supported
Device CPU
Open Shading Language
Sampling
Integrator Path Tracing
Render 128
Viewport 32
Advanced
Light Paths
Volumes
Hair
Simplify
Motion Blur
Film
Performance
Bake
Freestyle
Color Management
Playback Keying View Marker
Start: 1 End: 250
1 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250
Change Frame Box Select Pan View Dope Sheet Context Menu
Collection | Cube | Verts:8 | Faces:6 | Tris:12 | Objects:1/3 | Mem: 47.8 MB | v2.80.75

Getting set up

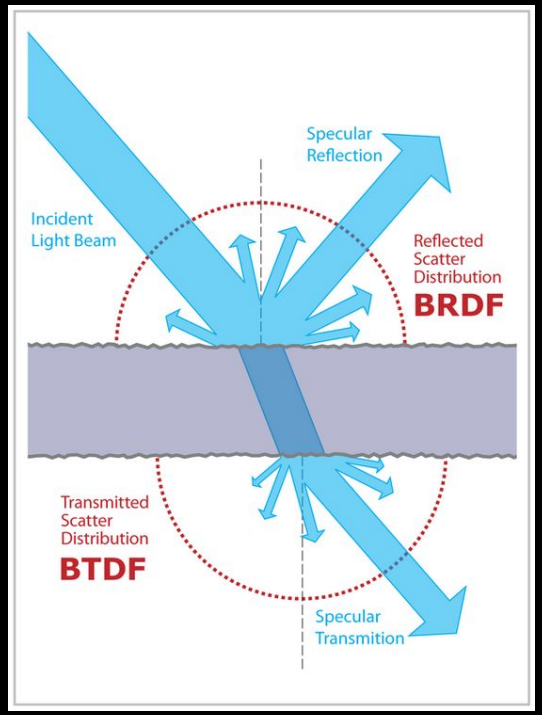
The default point light is sort of shit, let's add a sun light and switch the viewport shading into *rendered* mode.

Then click on the cube, go to the materials properties panel, then under the surface section switch the surface BSDF to the Diffuse BSDF


File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Scene
View Layer
Object Mode View Select Add Object
Global
Camera Perspective
(1) Collection | Cube
Rendering Done
Rendered
Scene Collection
Collection
Camera
Cube
Sun
Cube Material
Material
Materi
Data
Preview
Surface
Surface
Diffuse BSDF
Diffuse BSDF
Color
Roughness 0.000
Normal Default
Volume
Displacement
Displacement Default
Settings
Viewport Display
Playback Keying View Marker
1
Start: 1
End: 250
1 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250
Scroller Activate
Box Select
Scroller Activate
Dope Sheet Context Menu
Collection | Cube | Verts:8 | Faces:6 | Tris:12 | Objects:1/3 | Mem: 184.4 MB | v2.80.75

BSDF?

A BSDF, while not well defined, is basically a combined BRDF and BTDF that defines how photons are transmitted through the volume as well as how they are reflected



A Simple Diffuse BSDF

As we saw earlier, we can define a simple diffuse BRDF as a constant percentage of the incoming irradiance, allowing us to compute the final color like so:

$$BRDF = (surface\ albedo)\ /\ pdf$$

A Better Diffuse BSDF

Blender uses the popular Oren-Nayar Diffuse BRDF, which better approximates the falloff for the contribution of light sources as their angle becomes shallower

$$L_r = \frac{\rho}{\pi} \cdot \cos\theta_i \cdot (A + (B \cdot \max[0, \cos(\phi_i - \phi_r)] \cdot \sin\alpha \cdot \tan\beta)) \cdot E_0$$

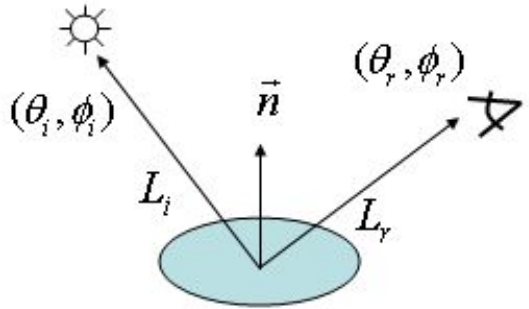
where

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

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

$$\alpha = \max(\theta_i, \theta_r),$$

$$\beta = \min(\theta_i, \theta_r),$$

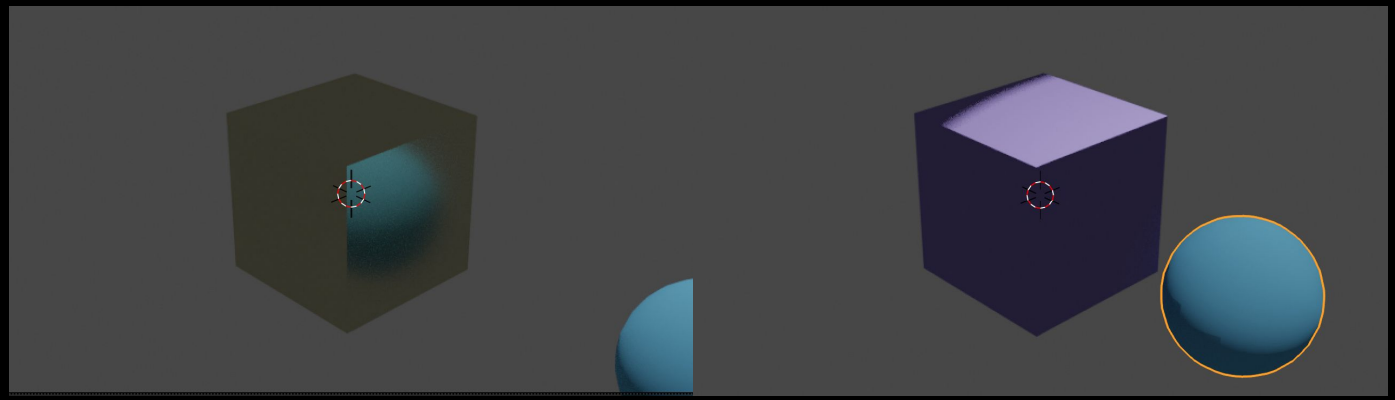


Playtime

Play around with the different BSDFs!

Try Glossy, Toon, Glass, and Subsurface scattering

Some of the models also have different implementations

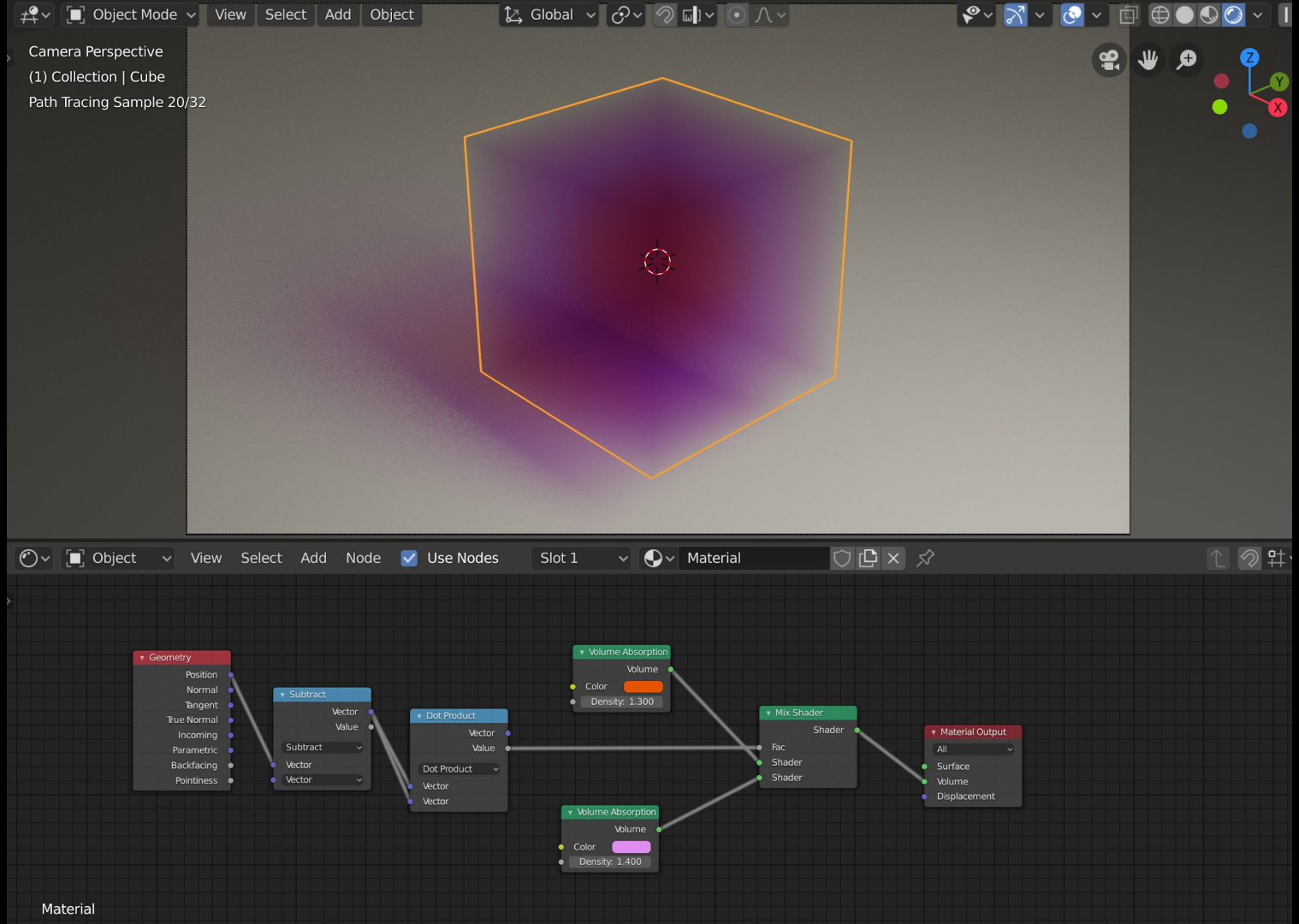


Shader Graph

What if we want to define more complex materials?

Shader graphs can be used to mix and modify colors and materials to make complex surfaces using textures, normal maps, etc

Switch to the shading tab with your material selected to see the shader graph


Object Mode
View
Select
Add
Object
Global
Camera Perspective
(1) Collection | Cube
Path Tracing Sample 20/32
Object
View
Select
Add
Node
Use Nodes
Slot 1
Material
Geometry
Position
Normal
Tangent
True Normal
Incoming
Parametric
Backfacing
Pointiness
Subtract
Vector
Value
Subtract
Vector
Vector
Dot Product
Vector
Value
Dot Product
Vector
Vector
Volume Absorption
Volume
Color
Density: 1.300
Volume Absorption
Volume
Color
Density: 1.400
Mix Shader
Shader
Fac
Shader
Shader
Material Output
All
Surface
Volume
Displacement
Material

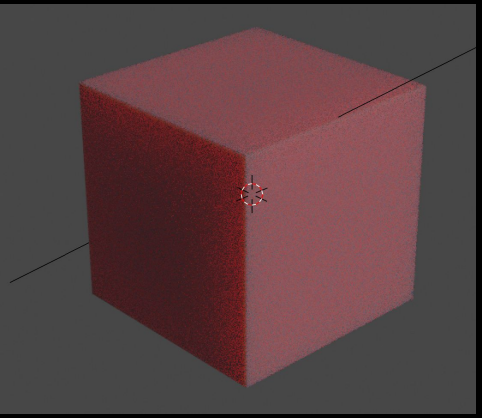
Shitty DIY Combined BSDF

Let's make our own BSDF to define a skin-like material!

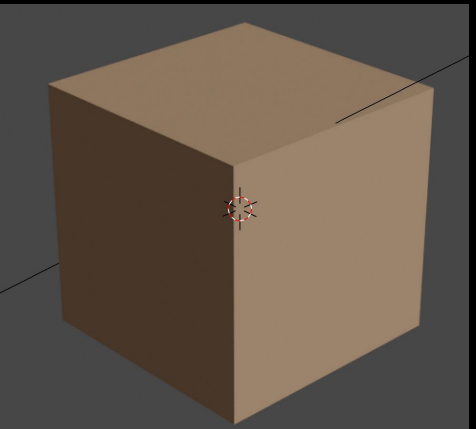
The default subsurface scattering BSDF assumes the object's surface and interior is all one color, but that's not true for our skin, which has a tan (at least for me) surface color and red subsurface color

Work time—what can we mix/add together to create this kind of material?

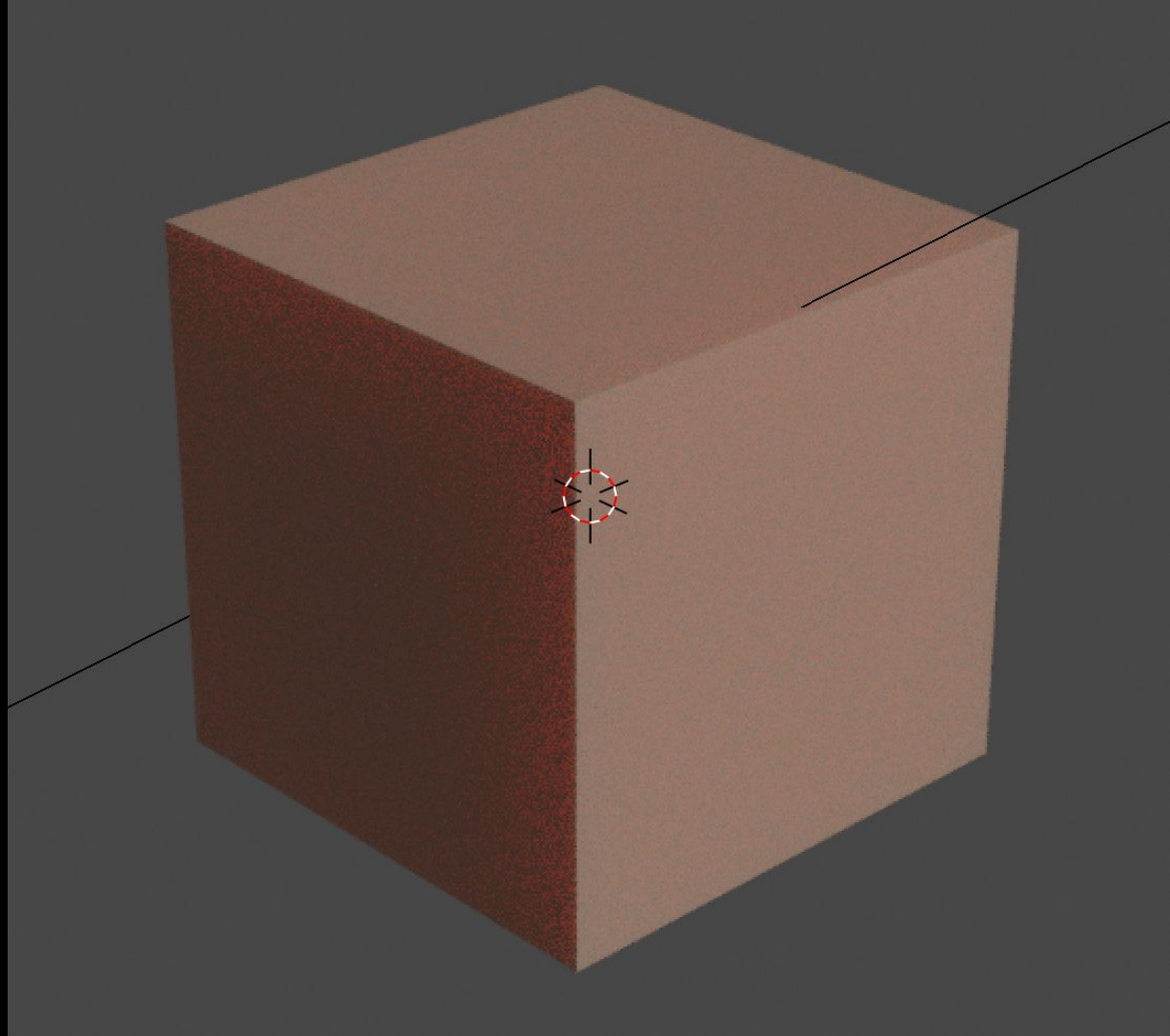
Subsurface Scattering



Mixed w/ Diffuse BSDF



=

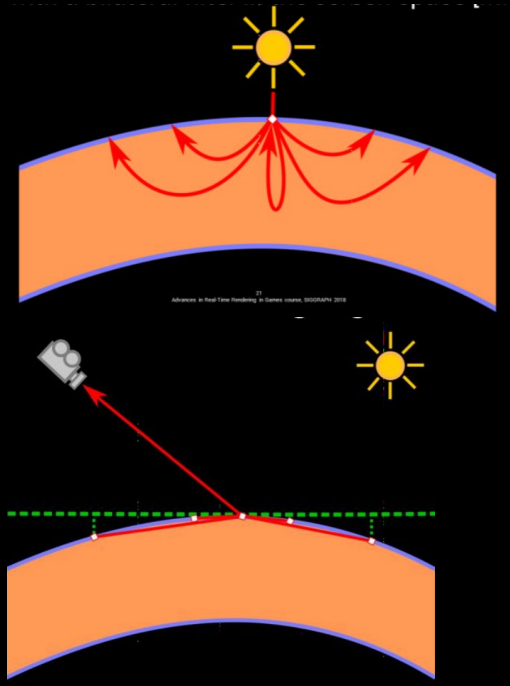


The BSSRDF (optional)

You may have noticed that the subsurface scattering material isn't a BSDF

Subsurface scattering is estimated using a Bidirectional Scattering-Surface Reflectance Distribution Function (!)

Integrates over incoming radiance of surrounding area as well



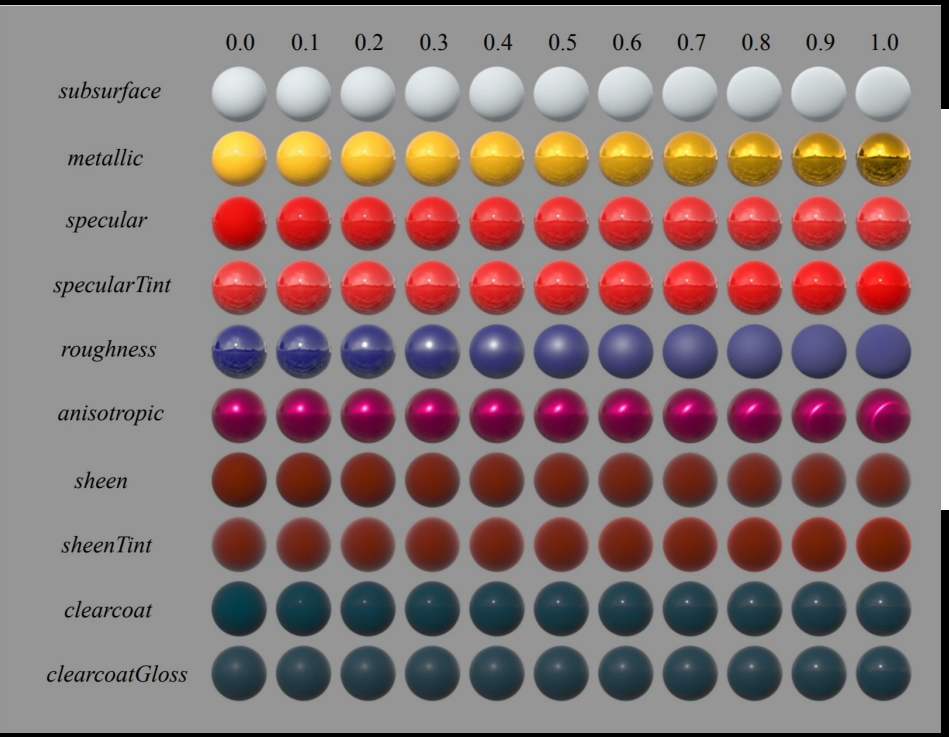
Photos from *Efficient Screen-Space Subsurface Scattering…* | Advances in Real-Time Rendering, SIGGRAPH 2018

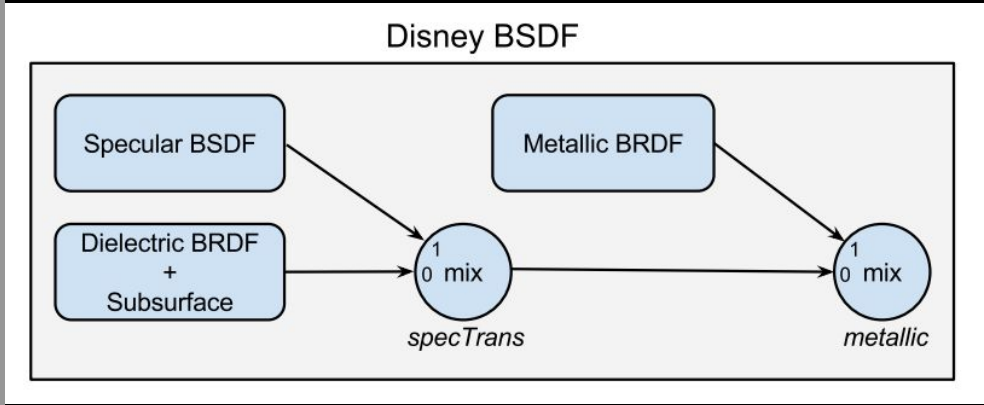
The Disney “Principled” BSDF

Is there a universal model we can use to model nearly all glossy/smooth/scattering surfaces?

Brent Burley at Disney Animation Studios came up with the “Principled” BSDF as a universal and easy-to-use way to describe most surfaces an artist would ever want. The model is now almost standard in nearly all renderers

Combines a lot of previous work on approximating surfaces, a lot of modifications to existing models

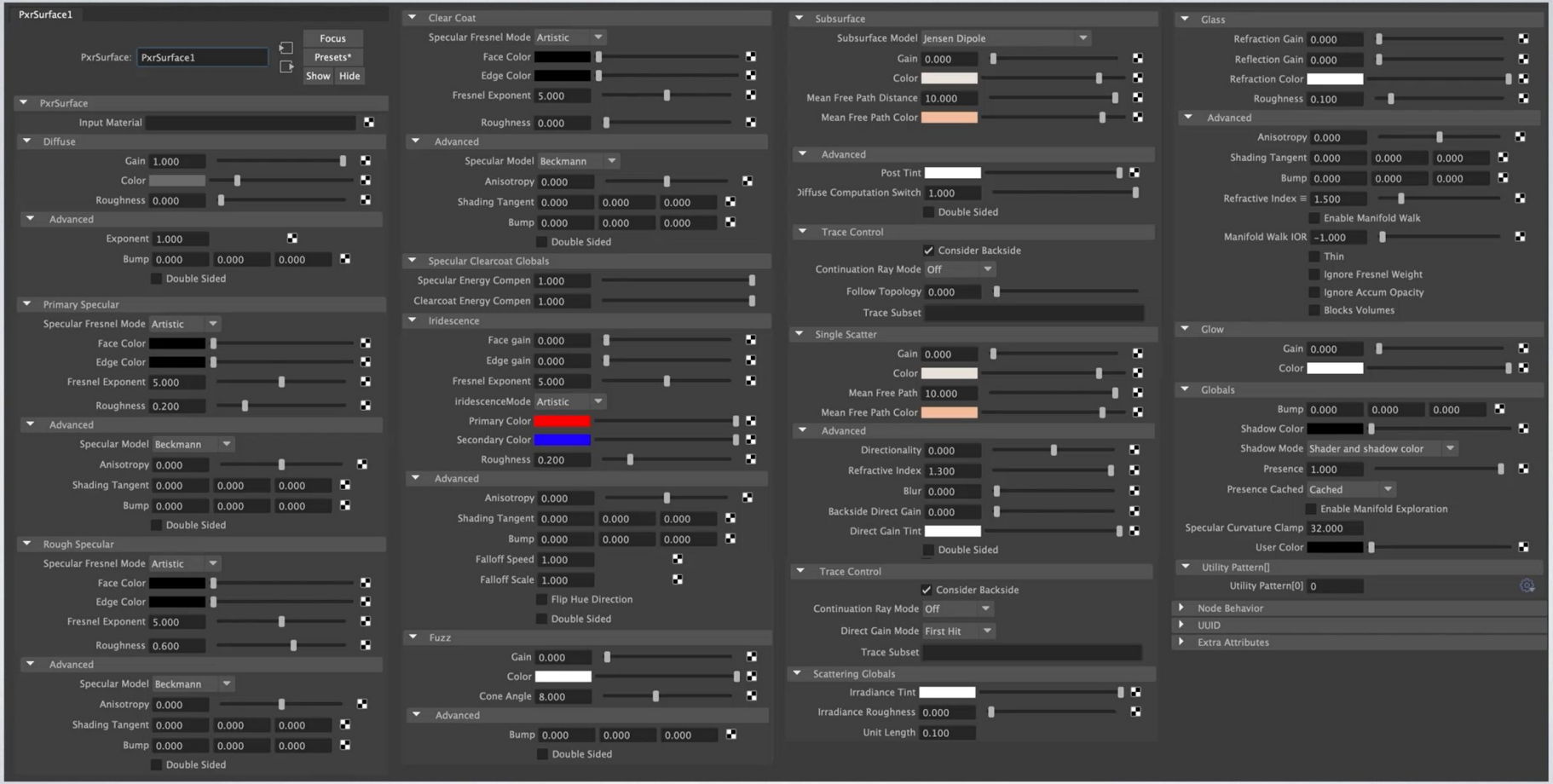




They aren't shown but the clearcoat and sheen lobe are both additive

Extending the Disney BRDF to a BSDF with Integrated Subsurface Scattering | SIGGRAPH 2015, Brent Burley

“That’s not a knife… That’s a knife!” Mr Crocodile Dundee



Shader Graph and OSL

OSL is a shading language originally developed by Sony Pictures Imageworks to describe *“materials, lights, displacement, and pattern generation”*

Used in a wide variety of animation and VFX, including most recent Marvel films and Pixar films.

Also is node based, and Blender’s nodes can be converted to OSL nodes under the surface by checking the “Open Shading Language” Box in render properties

OSL Example


```
shader simple_material(
    color Diffuse_Color = color(0.6, 0.8, 0.6),
    output closure color BSDF = diffuse(N))
{
    BSDF = Diffuse_Color * diffuse(N);
}
```

More in a nice tutorial here:

https://thhube.github.io/tutorials/osl/osl.html

For the Future

Take 168!

A lot of this presentation is based off a GTC 2020 presentation on Pixar's XPU rendering pipeline, it's pretty beginner friendly and I would recommend checking it out:

https://developer.nvidia.com/gtc/2020/video/s22266

Ravi Ramamoorthi's youtube channel

acmurl.com/graphics-interest

Slide References

Oren-Nayer reflectance model
https://en.wikipedia.org/wiki/Oren%E2%80%93Nayar_reflectance_model

Unity SSS implementation
http://advances.realtimerendering.com/s2018/Efficient%20screen%20space%20subsurface%20scattering%20Siggraph%202018.pdf

Principled BSDF
https://blog.selfshadow.com/publications/s2015-shading-course/burley/s2015_pbs_disney_bsdf_slides.pdf