VIA University College

Graphics & Audio

Making games look and sound good

Unity Graphics Overview

Learn the basics of rendering in Unity

Lighting

Looking at the light components and the Light Window in Unity

Modelling

Learn about meshes, UV mapping, 3D modelling and more

Shaders

A look at shaders and how they relate to materials, textures and meshes

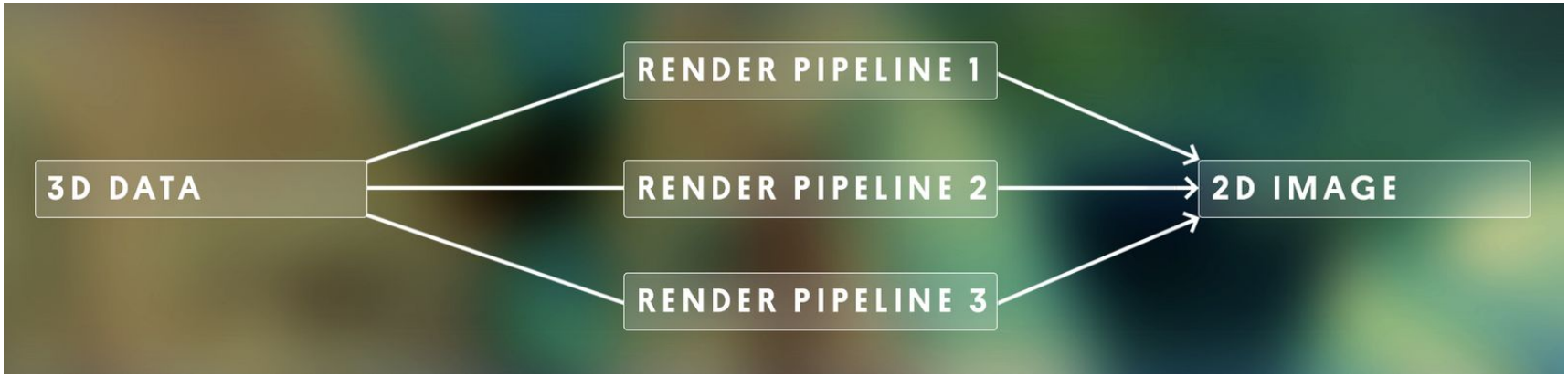
Audio

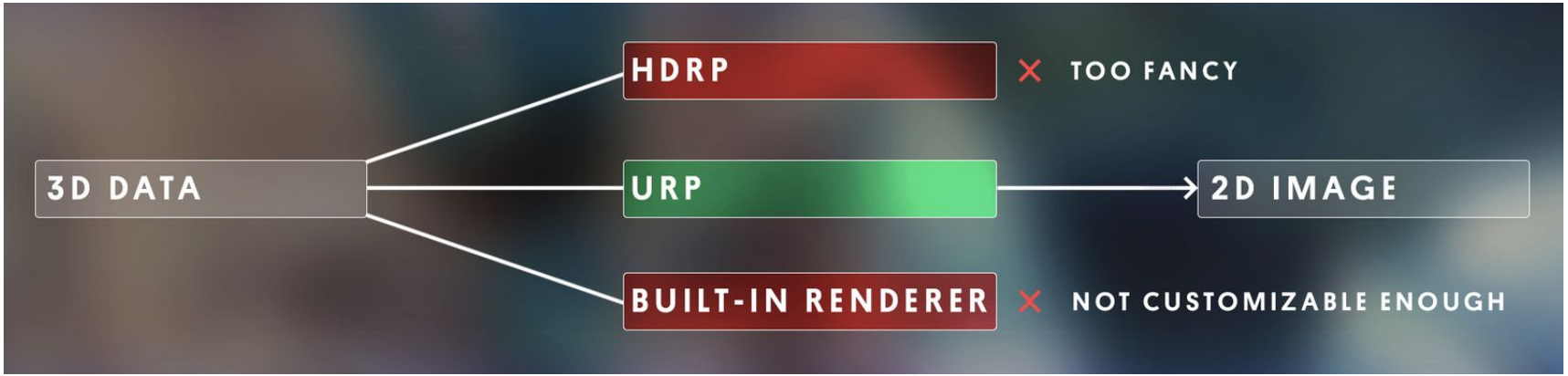
AudioClips, AudioSources, AudioListeners and how to control them

What is Rendering?

Render Pipelines

Unity Graphics Overview





HDRP Example

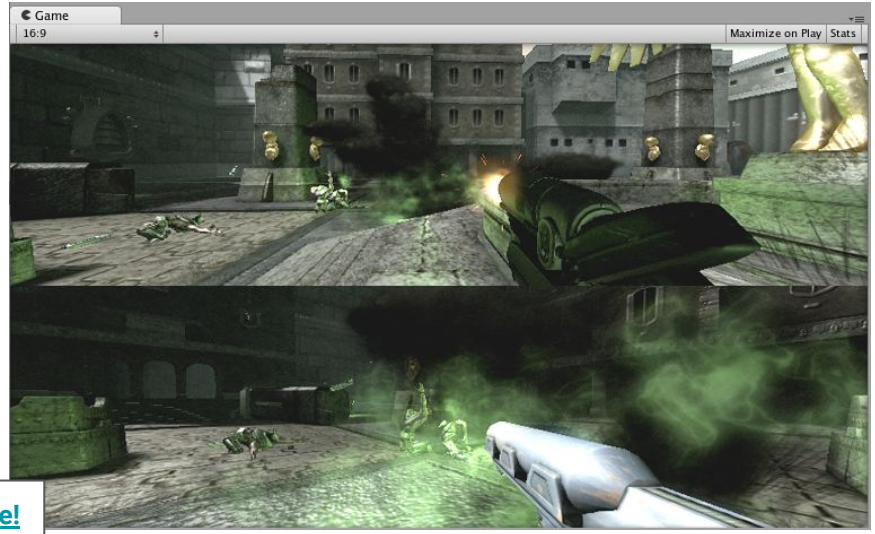
Camera

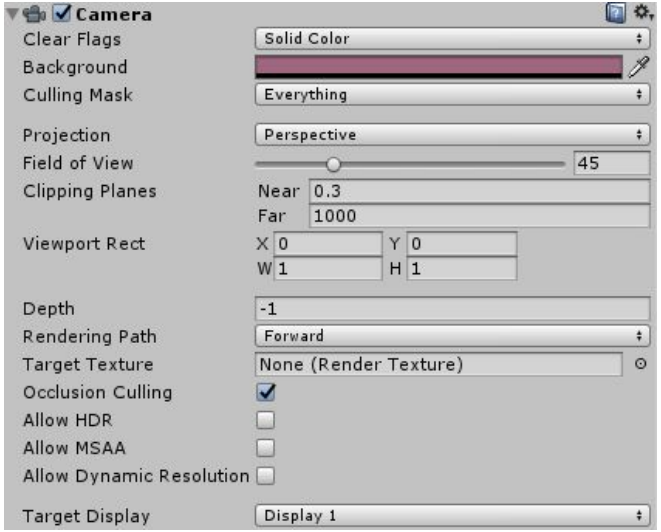
Unity Graphics Overview

Cameras are the devices that capture and display the world to the player

You can have an unlimited number of cameras in a scene

They can be set to render in any order and at any place on the screen



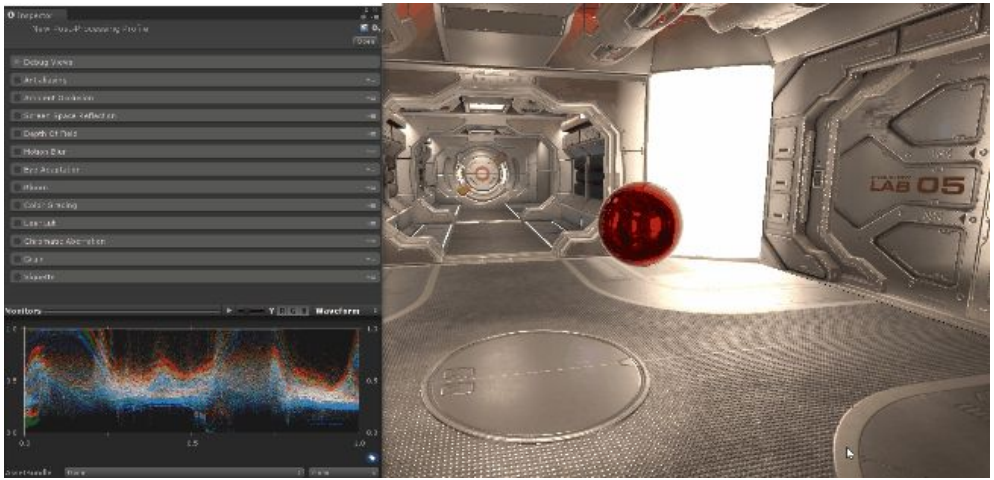


Learn more!

The Post Processing Stack

Image Effect (Shaders) combined into a single post-processing pipeline

Get it via the Unity Package Manager (only usable for Built-in renderer)



Example effects:

Antialiasing

Ambient Occlusion

Fog

Depth of Field

Motion Blur

Bloom

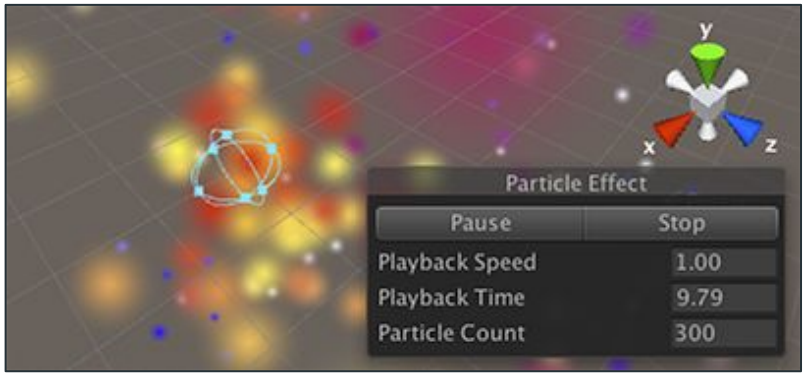
Color Grading

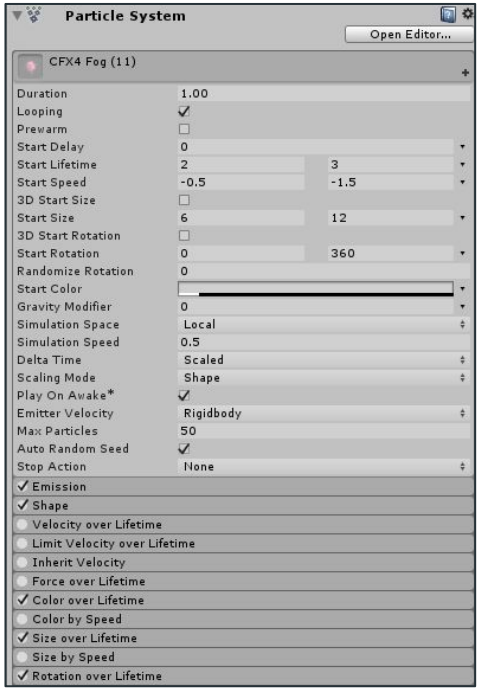
Particle System

Particles are simple **2D or 3D** entities that are displayed and moved in great numbers through a particle system.

Explosions, rain, snow, fire, dust, glitter, level ups...

Find some at the Asset Store!





Light Settings

Window > Rendering > Lighting Settings

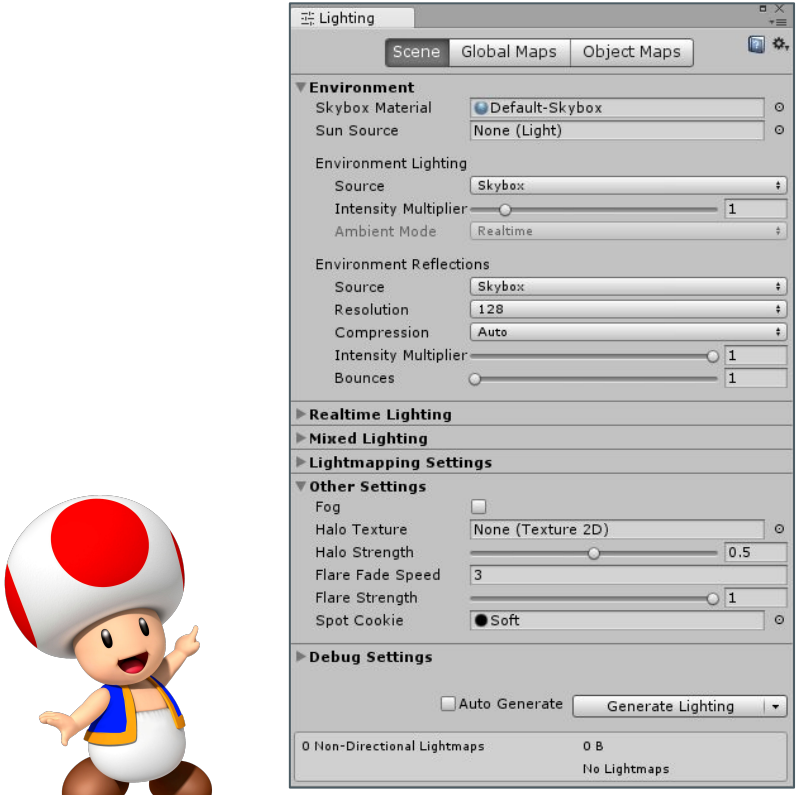
Skybox

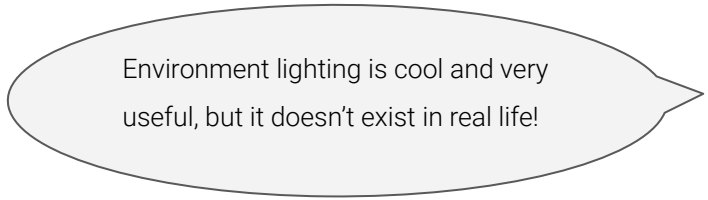
Environment/Ambient lighting

Fog

Light Explorer

Realtime & Mixed Lighting

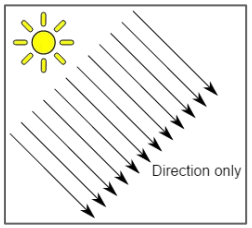




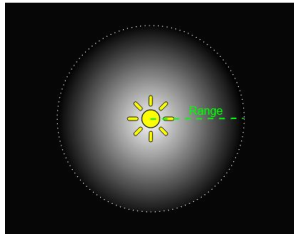
Learn more!

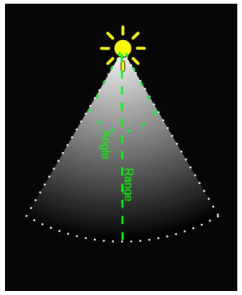
Light Components

Directional Light



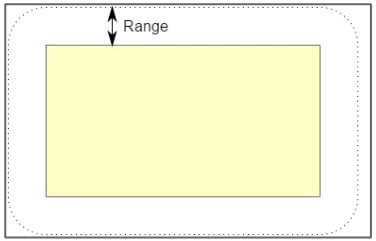
Point Light





Spotlight

Area Light

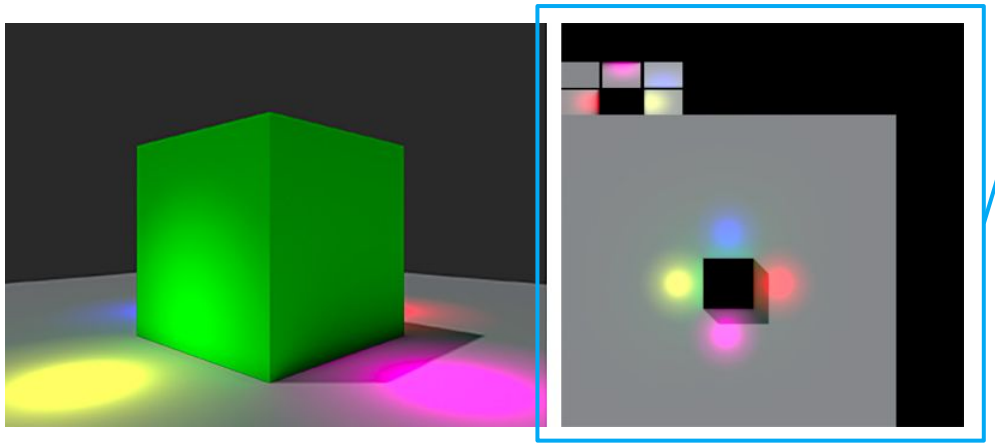


Light Baking

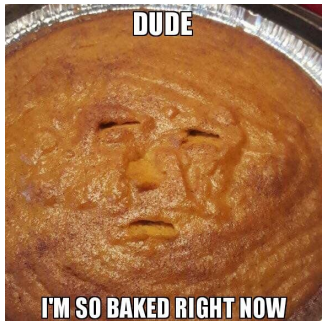
Realtime lighting can be **expensive**!

When baking a lightmap, the effects of light on static objects in the scene are calculated and the results are written to textures which are overlaid on top of scene geometry to create the effect of lighting.

A GameObject needs to be static in order to take advantage of the precomputed and baked lighting. Moving Objects can take advantage of **probes**

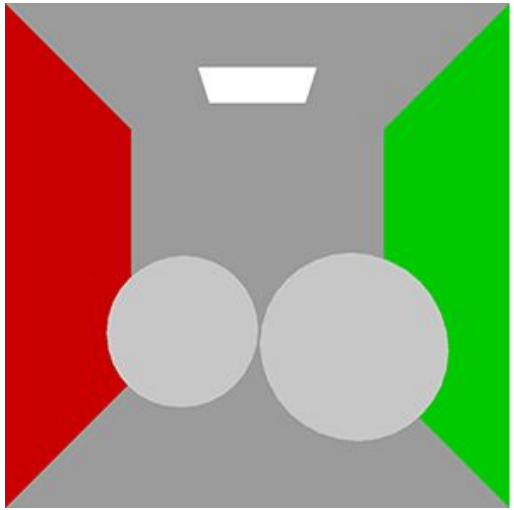


Baked Texture!

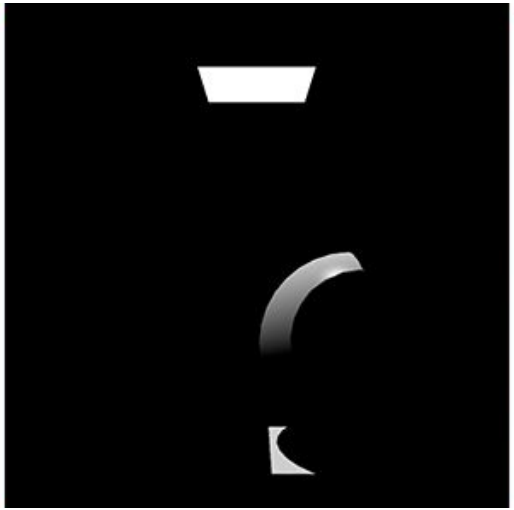


Global Illumination

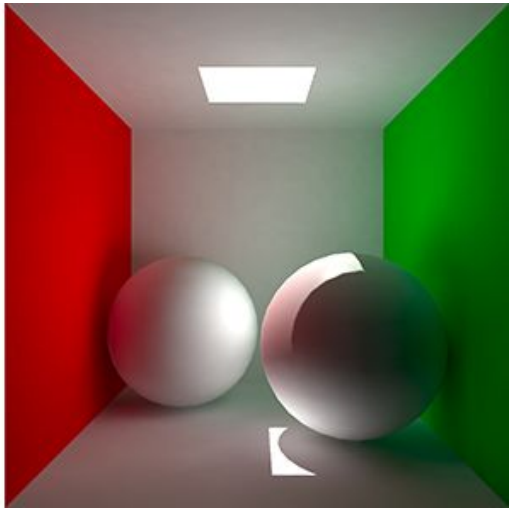
A general name for a group of algorithms used in 3D computer graphics that are meant to add more realistic lighting to 3D scenes by adding indirect lighting.



No Light



Direct light



Direct + indirect light

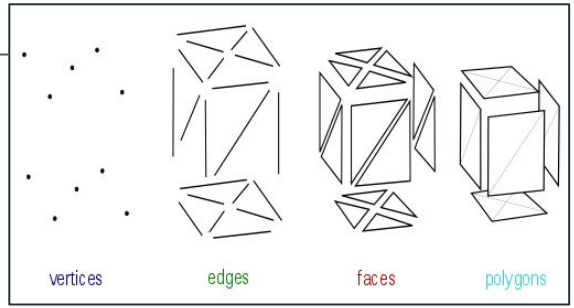
Mesh

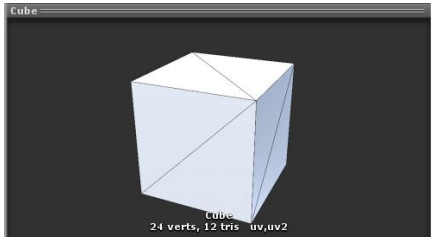
Modelling

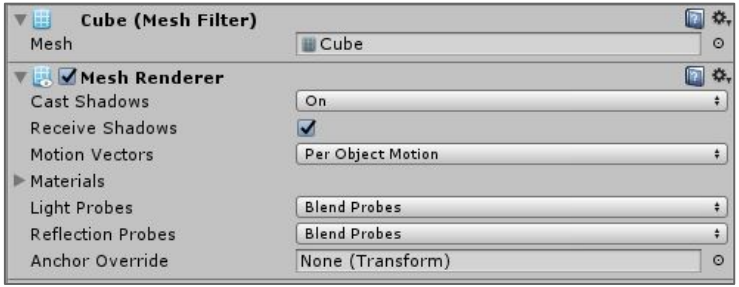
A (polygon) mesh consists of **vertices, edges and faces**

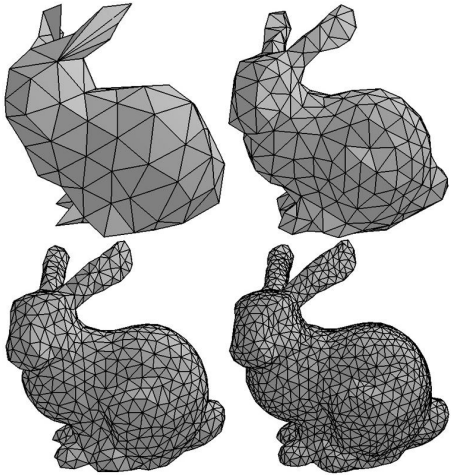
The mesh is the 3D skeleton of your GameObject

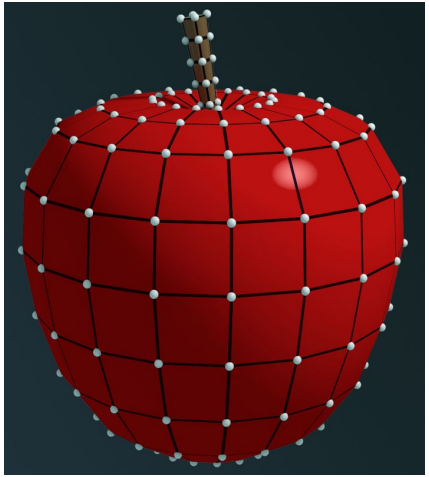
Mesh Filter and Mesh Renderer components



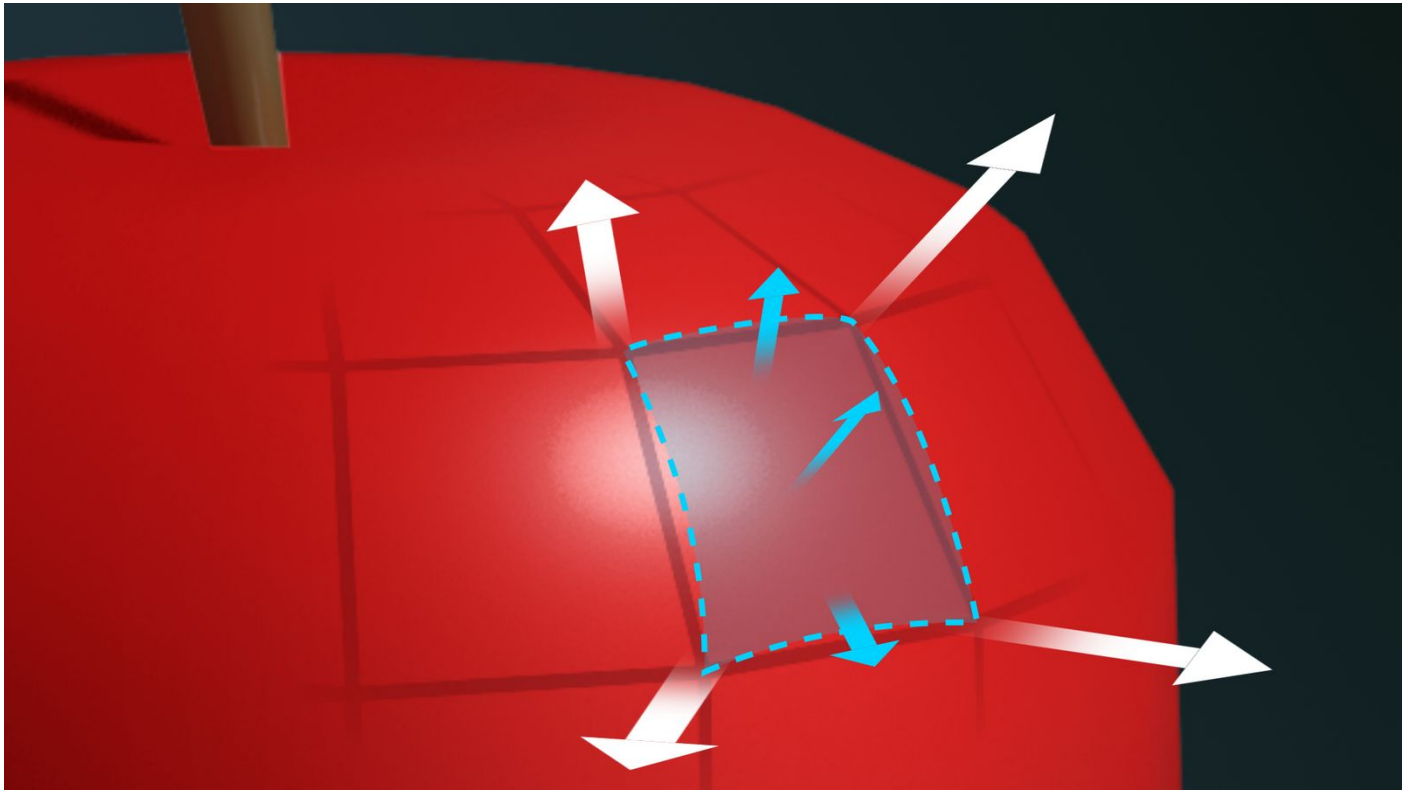






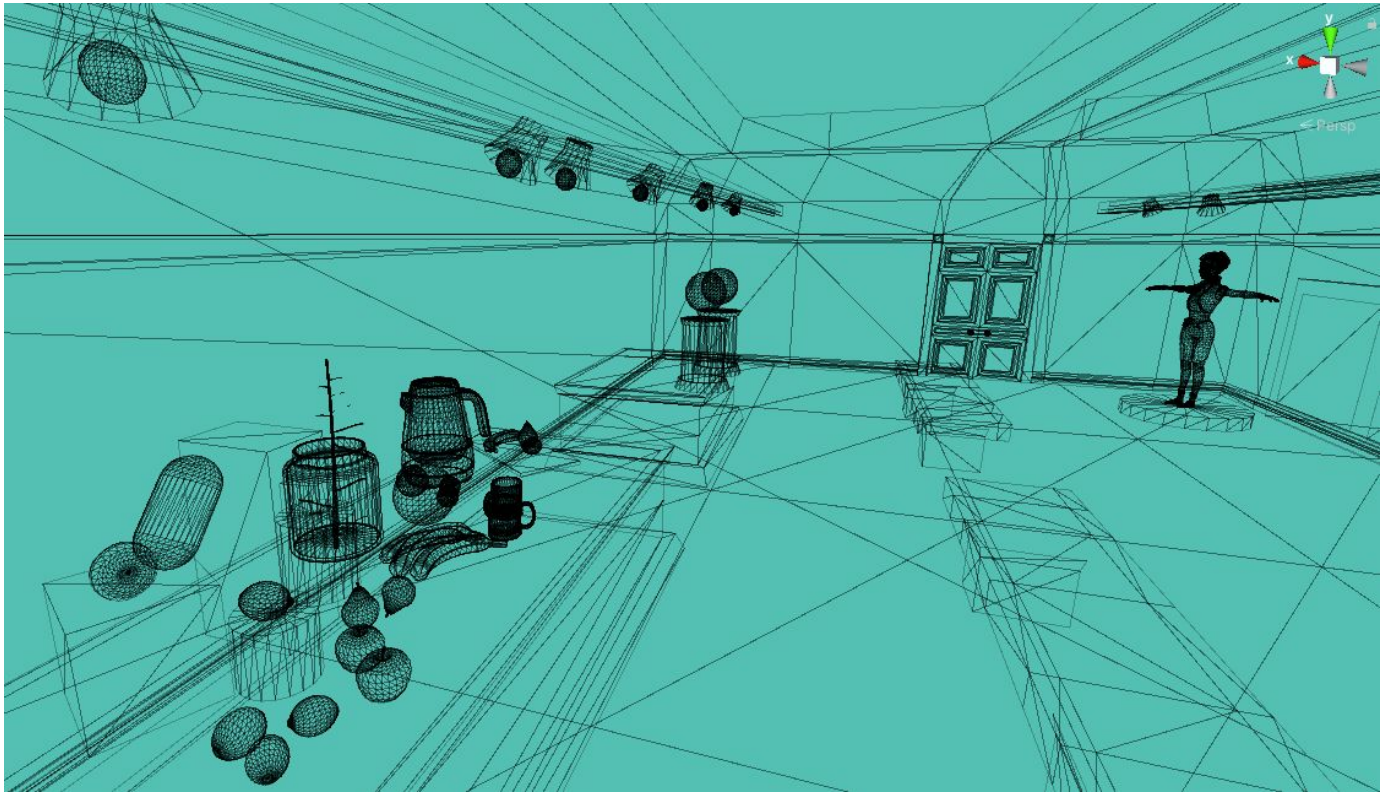


Normals



Scene View - Rendering Mode

Modelling



3D Models

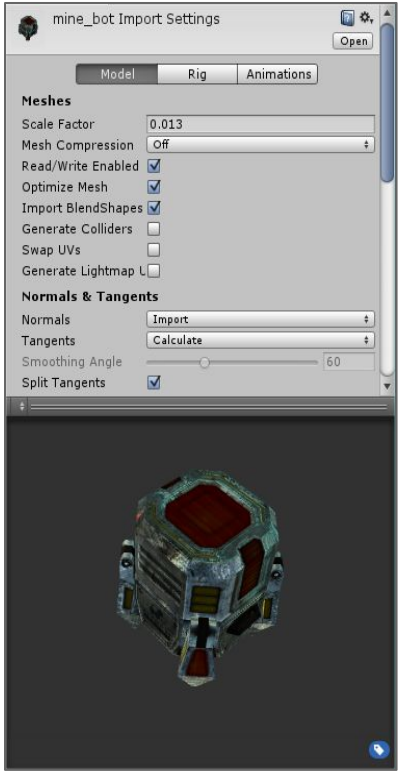
Modelling

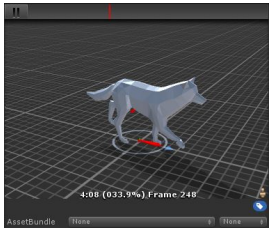
A model (in game development) consists of:

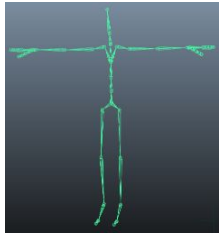
- One or more **meshes**

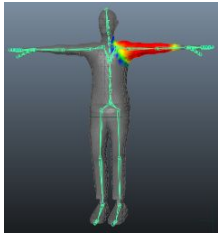
Sometimes it also has:

- A material
- Custom UV map
- A rig and animations
- Other fancy stuff...









Learn more!

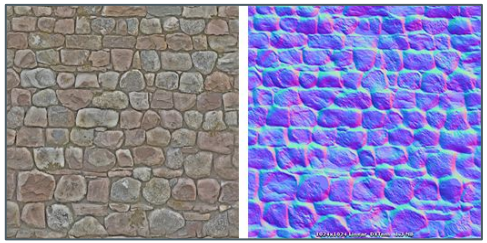
Material

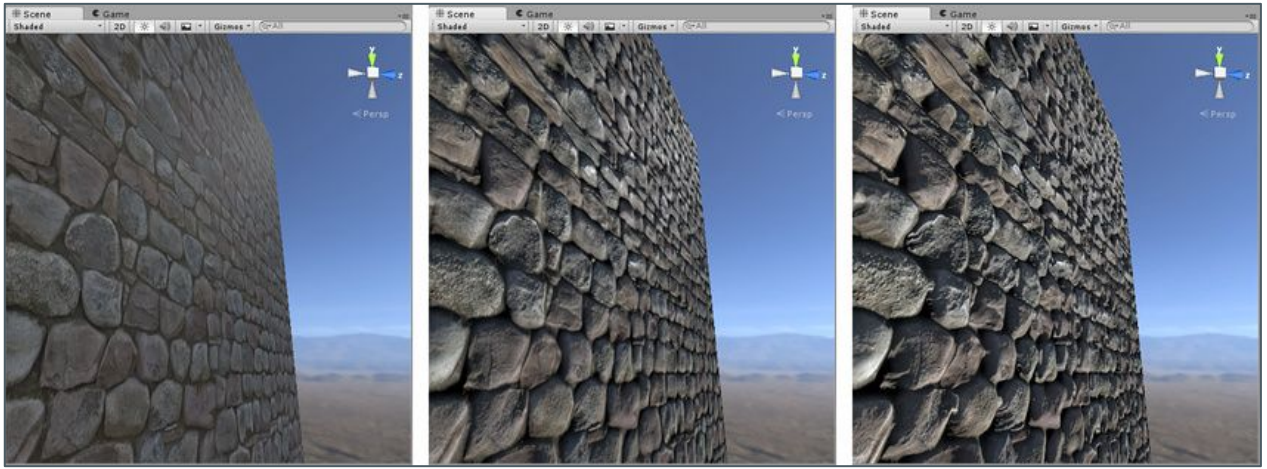
Modelling

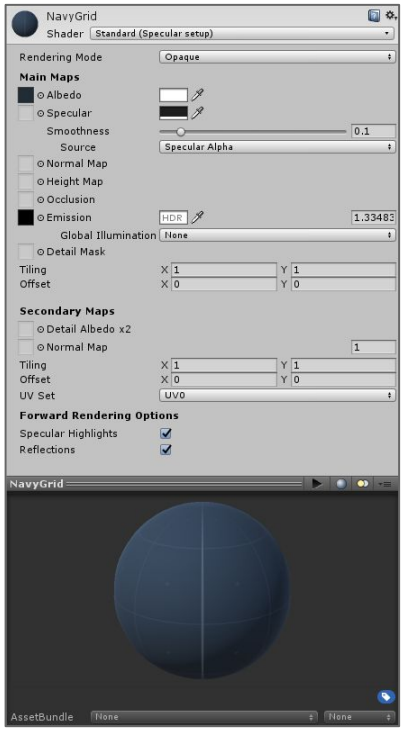
Materials are assets that control the visual appearance of meshes

Can’t modify standard material!

Albedo, textures, emission, tiling...





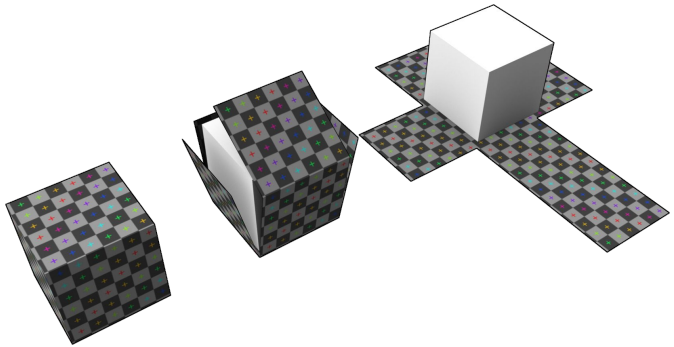


UV Mapping

Modelling

The process of projecting a 2D image to a 3D model's surface for mapping textures

Done with an external 3D modelling software



Importing

3D models are created with software such as Blender, 3DS Max & Maya.

There is two approaches to importing:

1) Import exported 3D files: .FBX, .3DS, .obj
2) Auto-convert proprietary application files: .Blend, .Max

Instead of learning 3D modeling, you can find models online

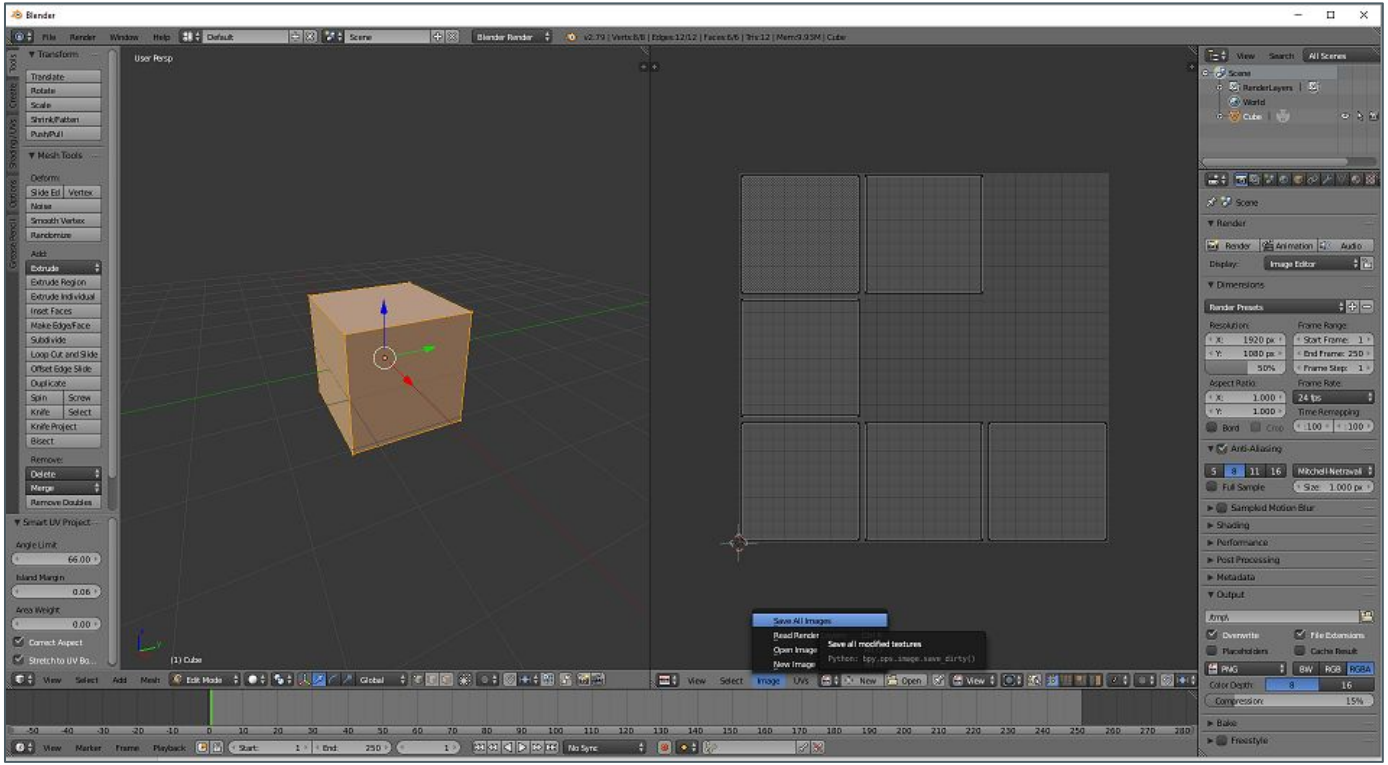
Blendswap, Sketchfab, the Asset Store, etc.

When importing, Unity automatically creates a model prefab 

Learn more!

Blender

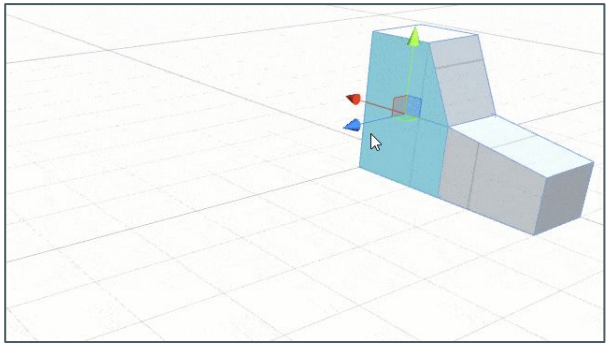
Modelling



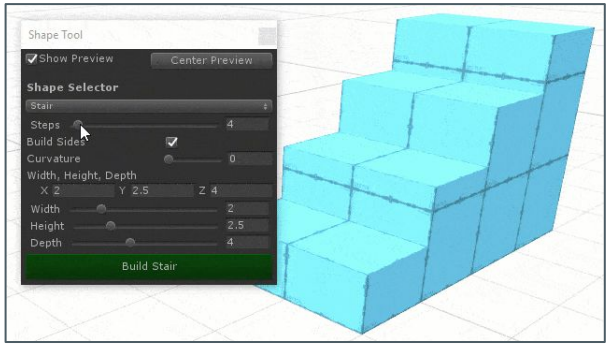
Probuilder

Modelling

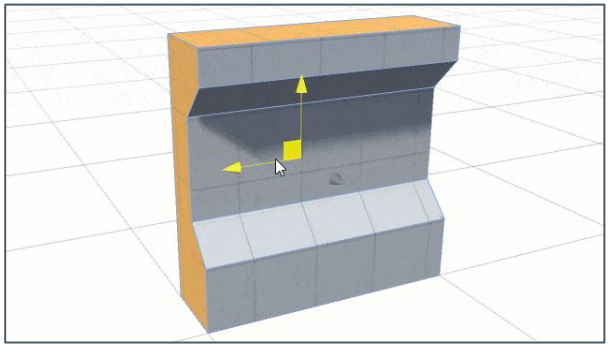
Extrude and insert



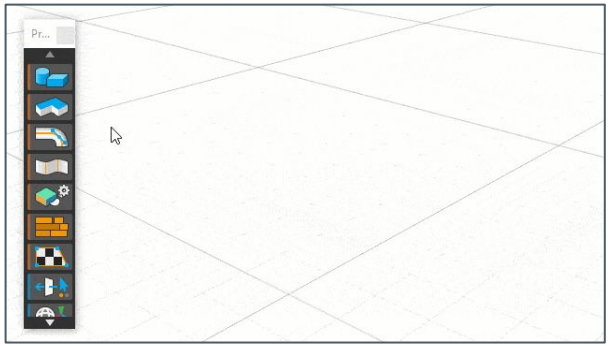
Procedural shapes



In-scene UV Controls



Indefinitely editable shapes



install with

Unity Package Manager

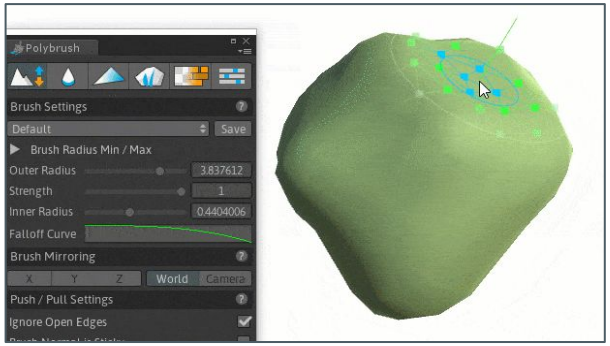
Demo

PolyBrush

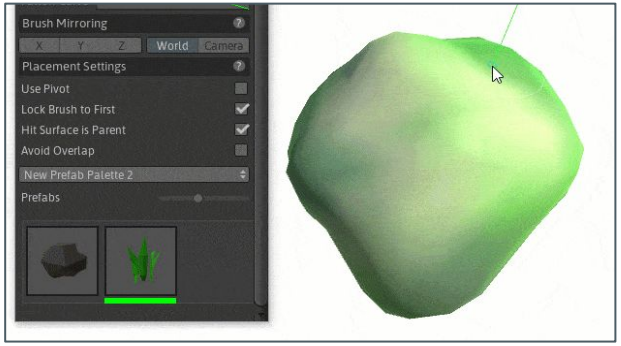
Modelling

Blend Textures

Paint Vertex Colors



Scatter Detail Meshes

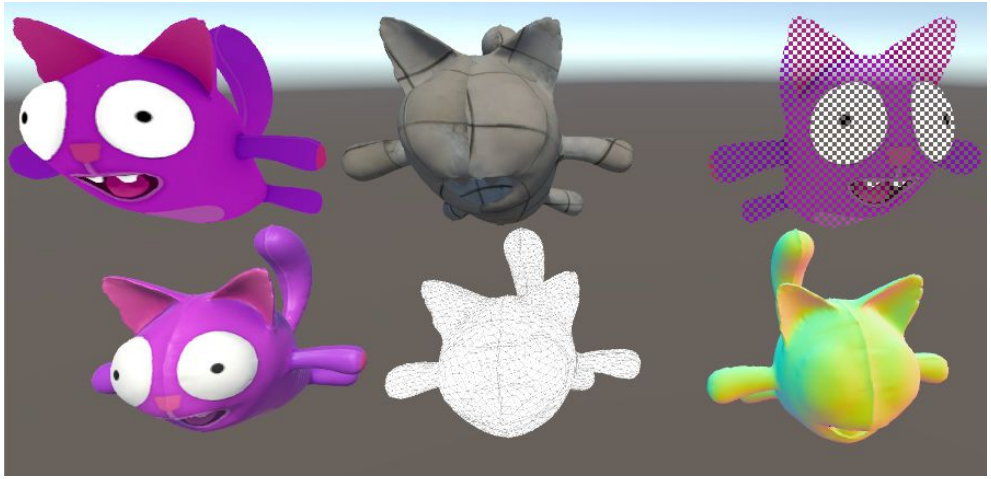


Sculpt Geometry

install with

Unity Package Manager

Shaders



(a) Albedo

(b) Warped diffuse

(c) Ambient cube

(d) (b) + (c)

(e) (a) * (d)

(f) Specular

(g) Rim lighting

(h) Specular + Rim Lighting

(j) Final result

Shaders

Fixed Pipelines vs Shaders

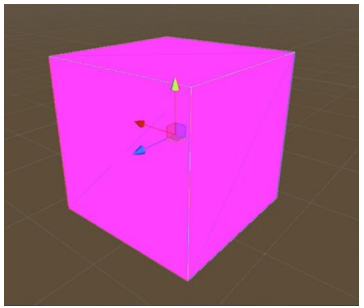
What are Shaders?

Shaders are SCARY!

Weird terminology (Rasterization, Stencil Buffer, Cull...)

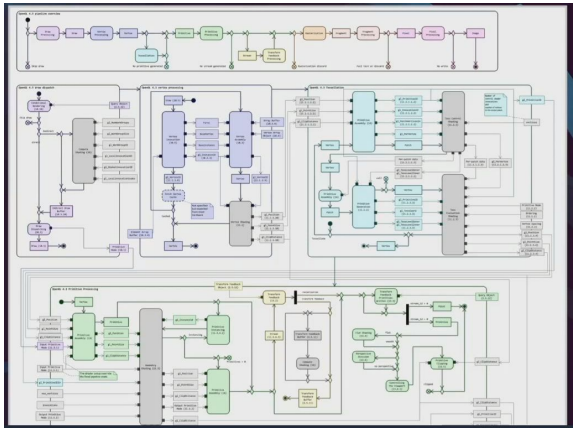
Unfamiliar math and data

Programming for the GPU? Parallel computing with thousands of cores!



$$I_{uniform} = \underbrace{I_a k_a}_{ambient} + \underbrace{I_p k_d (L \bullet N)}_{diffuse} + \underbrace{I_p k_s (R \bullet V)^n}_{specular}$$

$$I_{specular} = I_p k_s \left(\frac{2\underbrace{\{(1-t)(L \bullet N_1) + t(L \bullet N_2)\}}_{Same\ as\ Gouraud}\underbrace{\{(1-t)(V \bullet N_1) + t(V \bullet N_2)\}}_{Same\ as\ Gouraud}}{\|(1-t)N_1 + t\,N_2\|^2} - \underbrace{(L \bullet V)}_{Constant} \right)^n$$

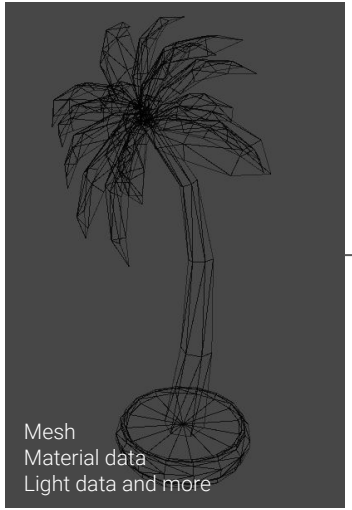


What are Shaders

A program that tells a computer to render something in a specific way

Originally used for the production of appropriate levels of light, darkness, and color within an image (**shading**)

Input



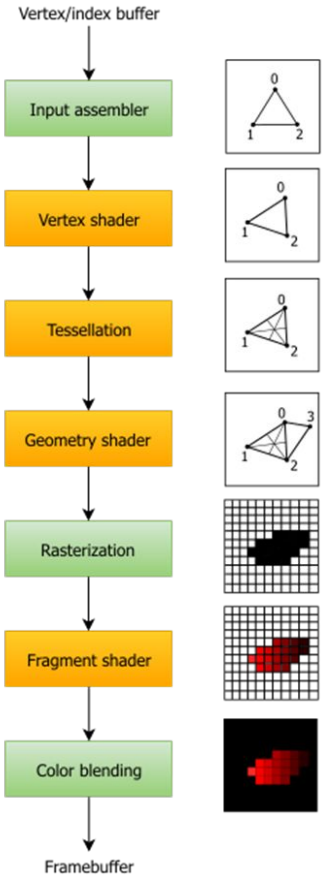
Shader

```
Shader "Custom/NewSurfaceShader" {
    Properties {
        _Color ("Color", Color) = (1,1,1,1)
        _MainTex ("Albedo (RGB)", 2D) = "white" {}
        _Glossiness ("Smoothness", Range(0,1)) = 0.5
        _Metallic ("Metallic", Range(0,1)) = 0.0
    }
    SubShader {
        Tags { "RenderType"="Opaque" }
        LOD 200

        CGPROGRAM
        // Physically based Standard lighting model, and
        #pragma surface surf Standard fullforwardshadows

        #pragma target 3.0

        sampler2D _MainTex;

        struct Input {
            float2 uv_MainTex;
        };

        half _Glossiness;
        half _Metallic;
        fixed4 _Color;

        UNITY_INSTANCING_BUFFER_START(Props)
```

GPU

Image

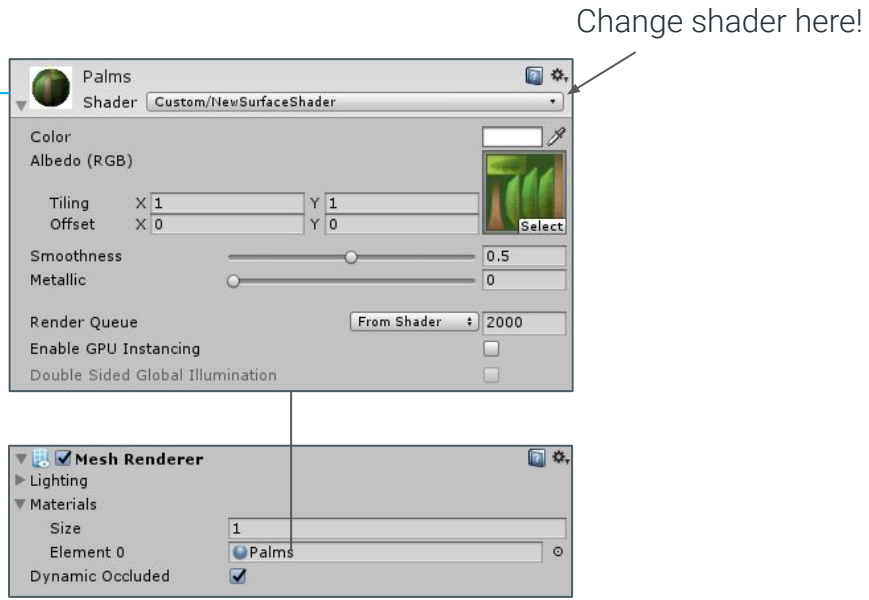
Materials & Shaders



Materials & Shaders

Materials are the UI for interacting with the properties of shaders

```
Shader "Custom/NewSurfaceShader" {
    Properties {
        _Color ("Color", Color) = (1,1,1,1)
        _MainTex ("Albedo (RGB)", 2D) = "white" {}
        _Glossiness ("Smoothness", Range(0,1)) = 0.5
        _Metallic ("Metallic", Range(0,1)) = 0.0
    }
    SubShader {
        Tags { "RenderType"="Opaque" }
        LOD 200

        CGPROGRAM
        // Physically based Standard lighting model, and
        #pragma surface surf Standard fullforwardshadows

        #pragma target 3.0

        sampler2D _MainTex;

        struct Input {
            float2 uv_MainTex;
        };

        half _Glossiness;
        half _Metallic;
        fixed4 _Color;

        UNITY_INSTANCING_BUFFER_START(Props)
```



Why Program Shaders?

Harness the power of the GPU!

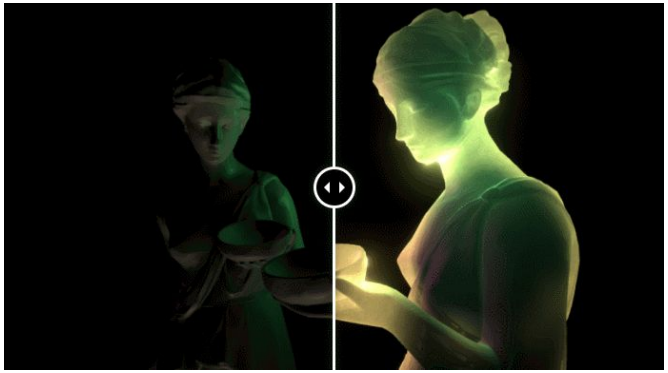
Why Program Shaders?

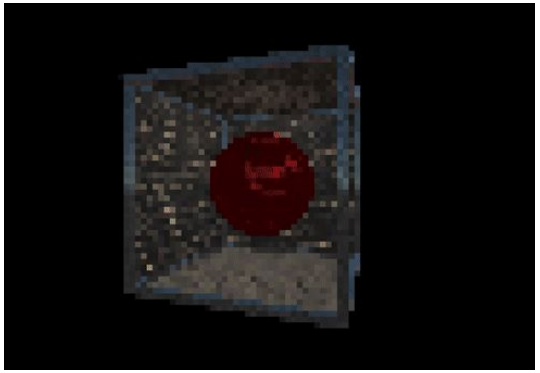
Complete control over how your game looks - Unity only have general purpose shaders

Valuable skill that not a lot of programmers possess

They are fun to play around with!

They are not THAT scary!





Why Program Shaders?

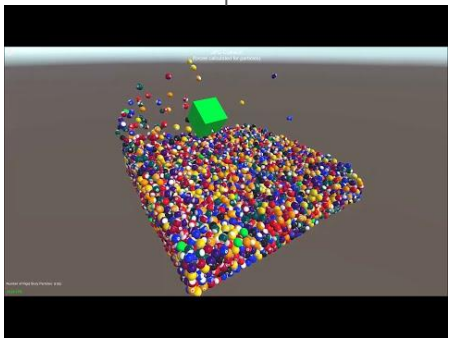
Shader Templates in Unity

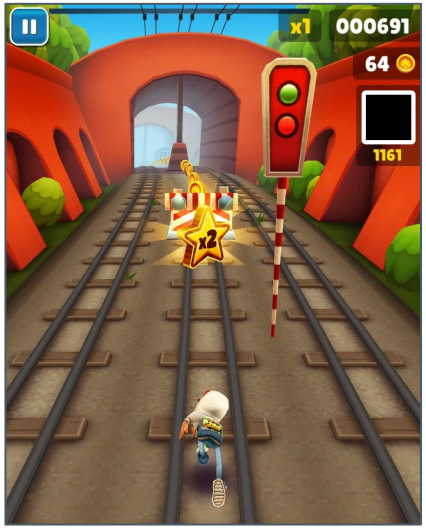
Surface Shaders

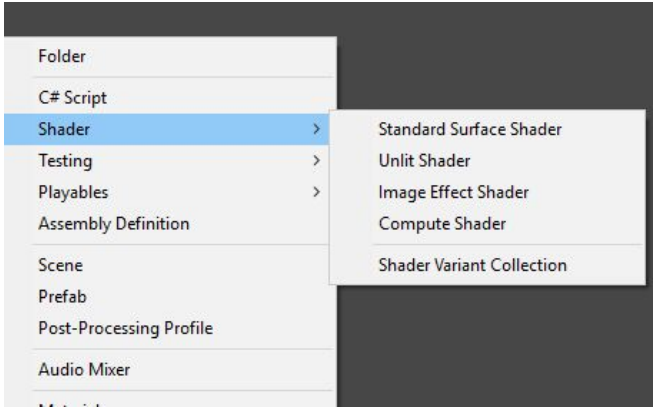
Unlit Shader

Image Effect Shader

Compute Shader







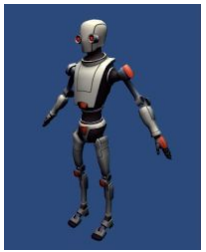
Unlit Shader

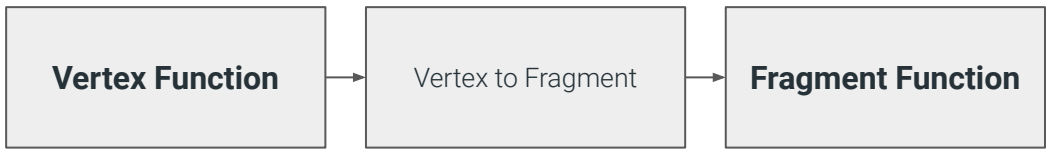
Build everything ground up

Manipulate vertices and pixels

Vertex & Fragment Shaders

```
fixed4 frag (v2f i) : SV_Target
{
    fixed4 col = tex2D(_MainTex, i.uv);
    UNITY_APPLY_FOG(i.fogCoord, col);
    return col;
}
ENDCG
```





Surface Shader

Surface shaders do a lot **automatically**

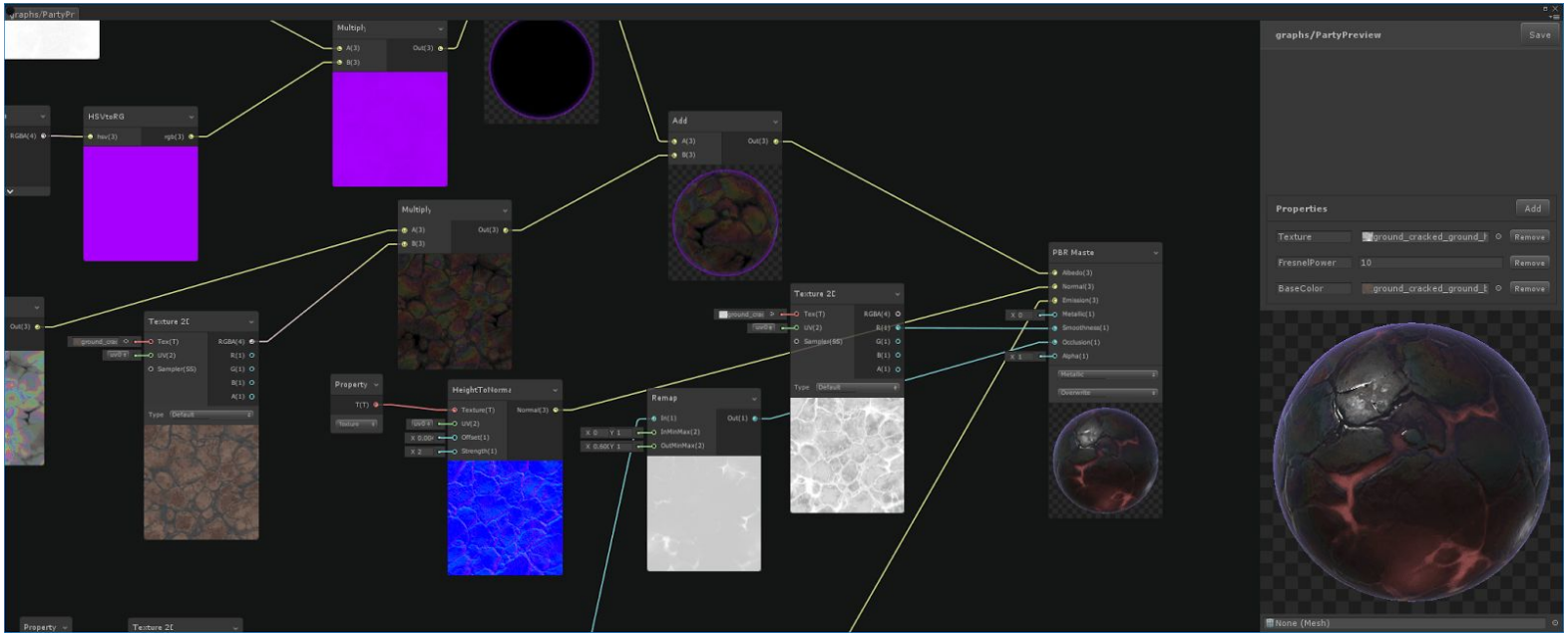
- Light
- Shadows
- Reflections
- Probes

```
void surf (Input IN, inout SurfaceOutputStandard o) {
    fixed4 c = tex2D (_MainTex, IN.uv_MainTex) * _Color;
    o.Albedo = c.rgb;
    o.Metallic = _Metallic;
    o.Smoothness = _Glossiness;
    o.Alpha = c.a;
}
```

Returns various material properties

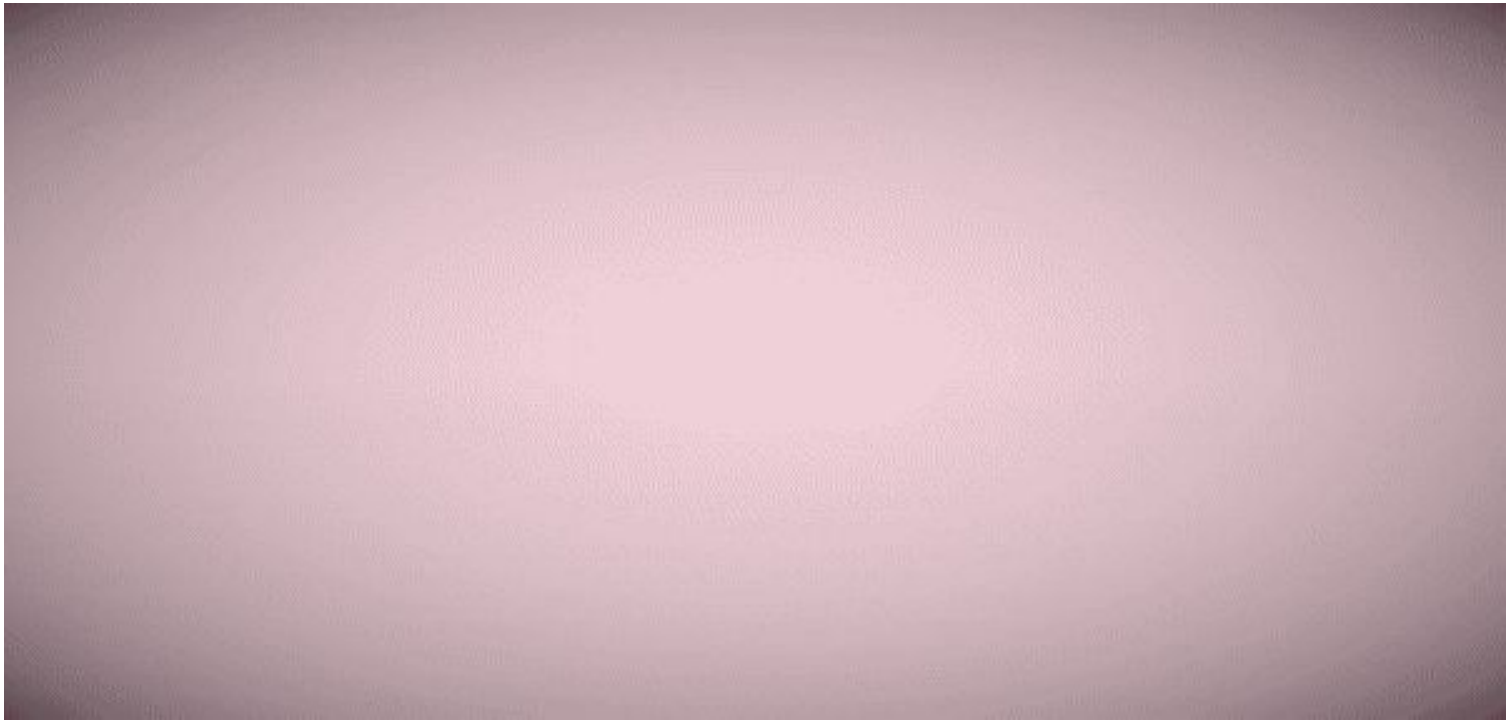
Shader Graph

Shaders

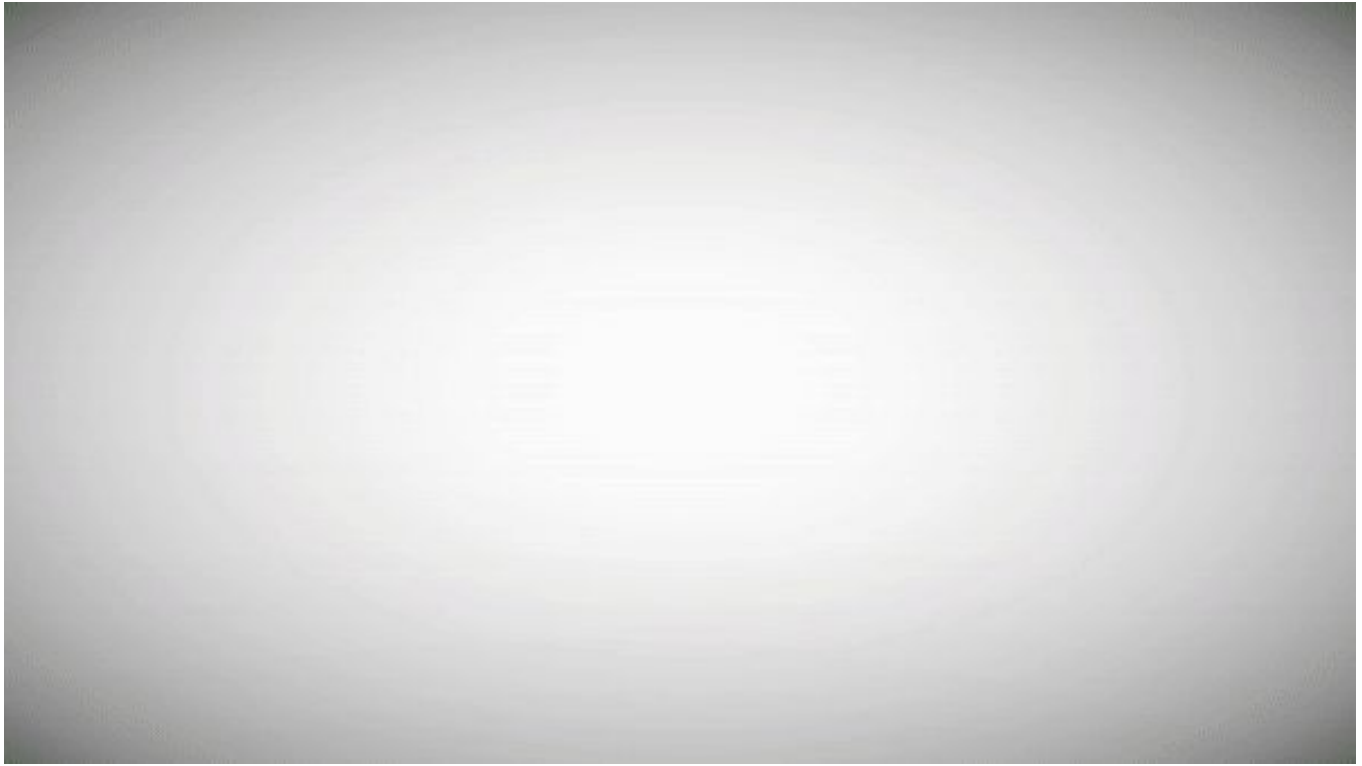


[Basics of Shader Graph]

Shader Demo

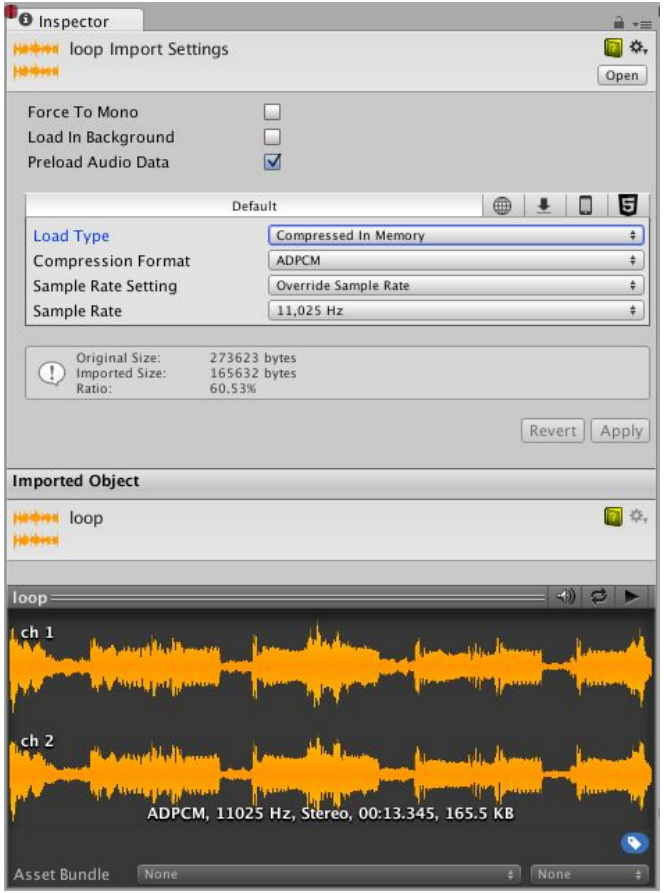


Shader Demo



Audio Clip

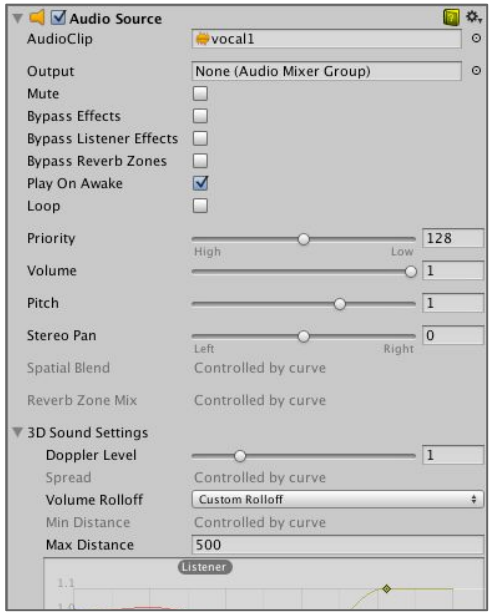
- An asset containing audio data.
- File formats supported in Unity:
  - .aif, .wav, .mp3 and .ogg
- Load type, compression format, sample rate, etc…
- Preview window
- Free resources:
  - freesound.org (SFX)
  - bensound.com (Music)
  - 99sounds.org
  - HQ free Sound FXs from GDC
  - Or use the asset store and google!



Learn more!

Audio Source

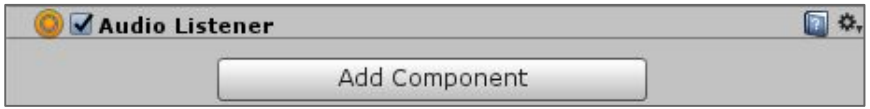
- “Audio Source” is a component in Unity.
- If you use it on a GameObject, you turn it into a controllable **speaker**
- The component can’t do anything without an assigned Audio Clip.
- It is used like a controller for starting and stopping clips, as well as modifying audio properties on it.
    - Play On Awake, Loop, Volume, Pitch, etc...

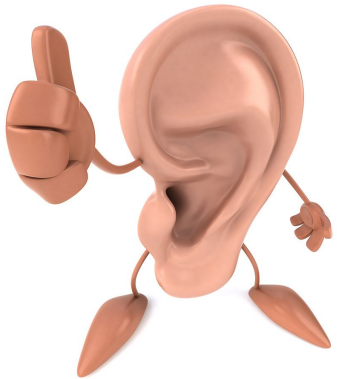


Learn more!

Audio Listener

- The Audio Listener component is the ears in your game.
- It receives input from Audio Sources that are in the scene.
- The component has no settings, since all of the audio's behavior is handled elsewhere (Audio Source & Audio Mixer).
- Attached to the Main Camera by default
- Always only have one listener in your scene at a time!

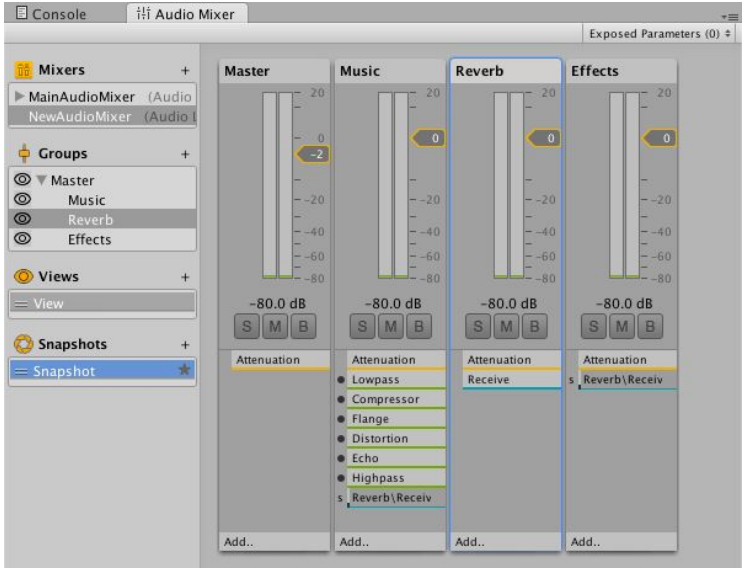




Audio Mixer

- The Unity Audio Mixer is an asset that allows you to mix various Audio Sources, apply effects to them, and perform mastering.
- The asset is configured through the Audio Mixer view.

- Grouping and serialization of signals
- Audio Effects
- Snapshots
- Exposing AudioMixer parameters
- Routing signal through multiple mixers



Scripting

Audio

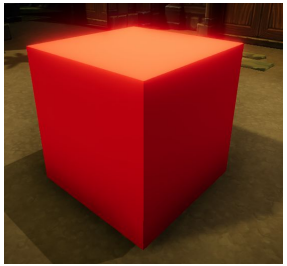
- You can play a single audio clip using Play, Pause and Stop.
- Multiple audio clips can be played on one audio source using PlayOneShot.
- You can play a clip at a static position in space with PlayClipAtPoint.
- Control volume and pitch to adjust sound effects slightly.
- Learn more in the documentation.

```
private AudioSource source;
public AudioClip shot1;
public AudioClip shot2;

void Awake() {
    source = GetComponent<AudioSource>();
}

void Update() {
    if (Input.GetKey("f")) {
        source.PlayOneShot(shot1,1f);
    }
}
```

```
void OnCollisionEnter(Collision coll) {
    source.pitch = Random.Range(0.75f, 1.5f);
    float hitVol = coll.relativeVelocity.magnitude * 0.2F;
    source.PlayOneShot(shot2, hitVol);
}
```

Learn more!

Exercises



Exercises are available on itslearning along with an environment unitypackage.