Model, Mesh, Material, Shader, Texture

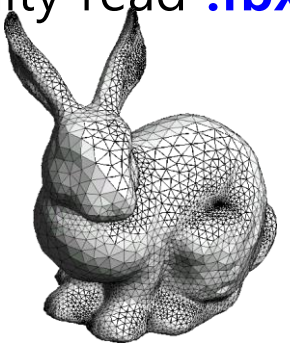
Fall 2023
11/23/2023
Kyoung Shin Park
Computer Engineering
Dankook University

Model, Mesh, Material, Shader, Texture

- In Unity, Materials allow you to specify which Shader to use on a Mesh.
- Shaders perform a set of calculations that tell Unity how to render (draw) your Meshes based on properties specific to that Shader.
- You can apply Materials to make your floor look like it's made of tile, wood, stone, or anything.
- Some Materials use Textures, which are bitmap images (BMP, JPEG, PNG, etc). Unity projects these images on the surfaces of Mesh to achieve a more realistic result.
- The Mesh stores the texture mapping data as *UVs*.
- *UV coordinates* (also sometimes called *texture coordinates*) are references to specific locations on the image.

Model

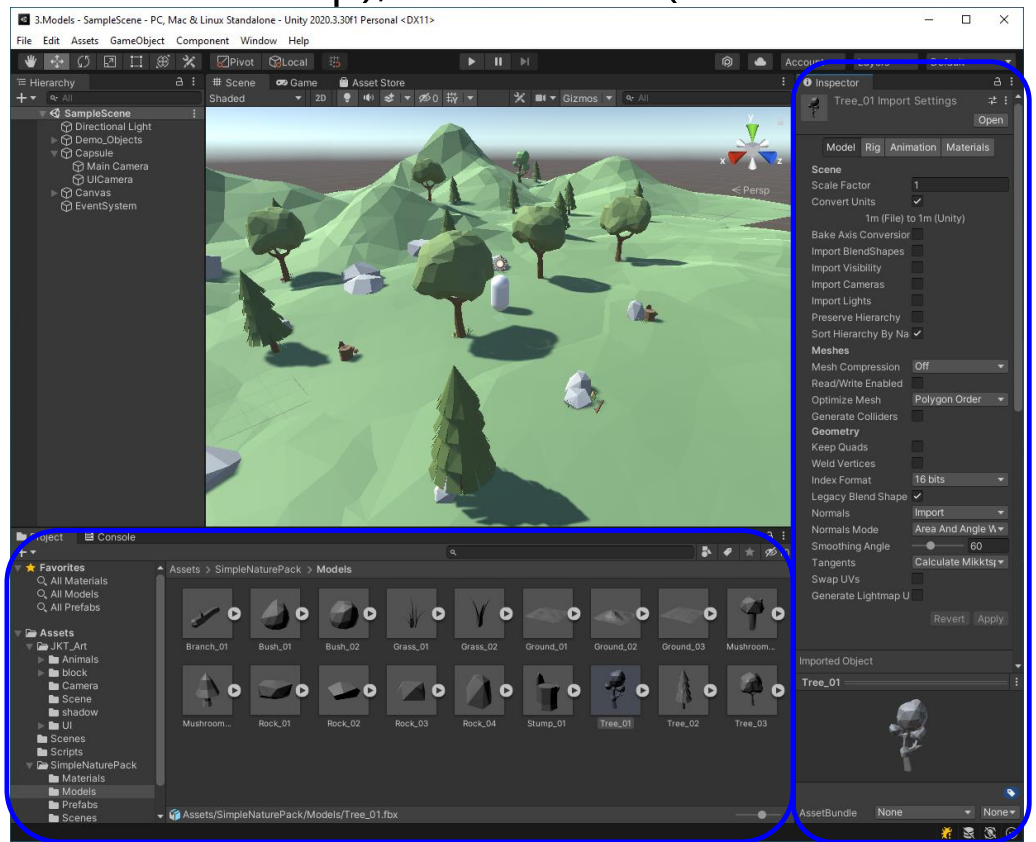
- Models
  - Models are files that contain data about **the shape and appearance of 3D objects**, such as **characters, terrain, or environment objects**.
  - Model files contain a variety of data including **meshes, materials, and textures.**
  - They can also contain **animation** data, for **animated characters**.
  - You usually create models in an external application(such as Maya, Blender), and then **import them into Unity**.
  - Unity read **.fbx**, .dae(Collada), .dxf, .obj standard 3D file formats.



https://medium.com/shader-coding-in-unity-from-a-to-z/rendering-pipe-line-f0471aa0904b

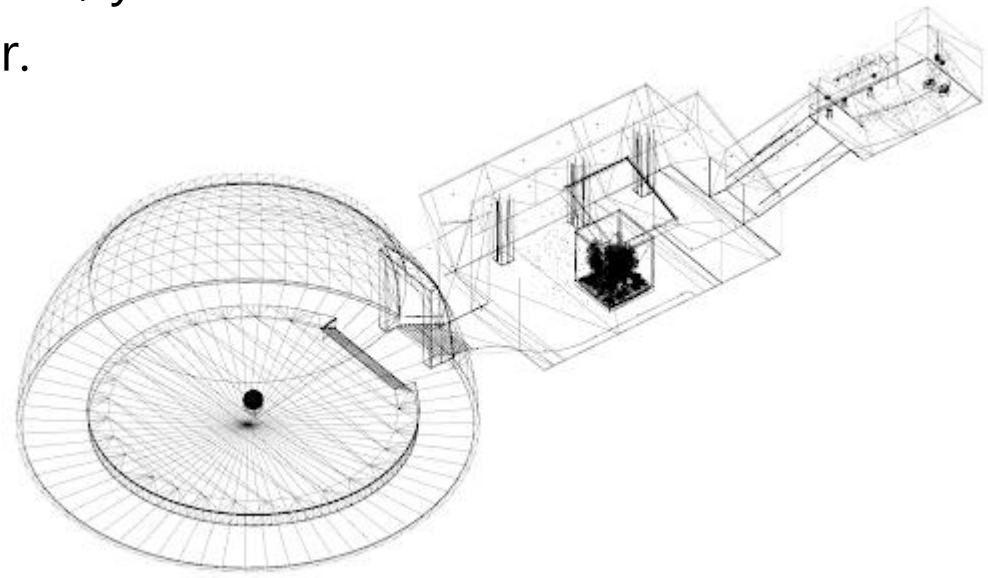
Model

- Model Import Settings window
  - Model inspector window shows the **Model** tab (for 3D model) by default. It also has the **Rig** (for skeleton), **Animation** (for animation clip), **Materials** (for materials and textures) tab.



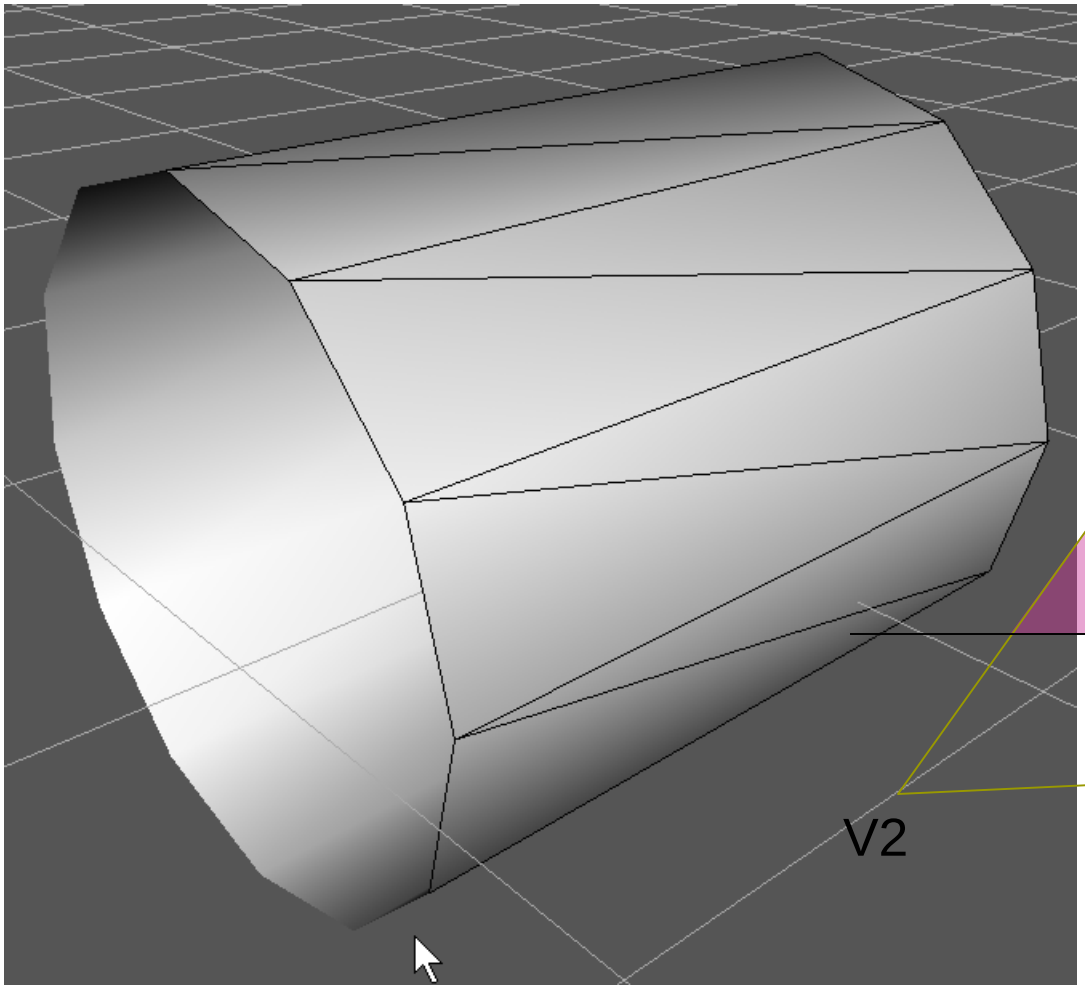
Mesh

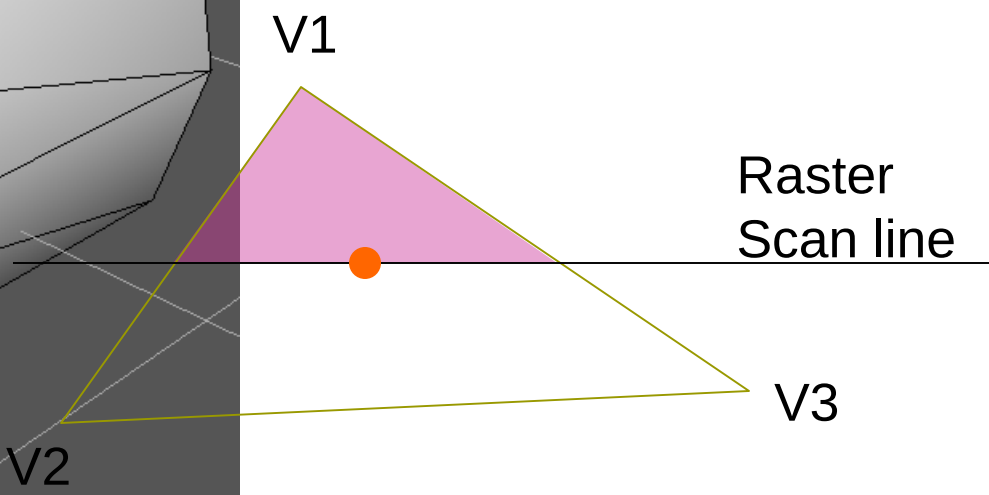
- A **mesh** is a collection of data that describes a **shape**.
  - In graphics, you use meshes together with materials. **Meshes describe the shape of an object**, and **materials describe the appearance of its surface**.
  - In physics, you can use a mesh to determine the shape of a collider.



Wireframe view of the meshes in Unity's 2020.1 HDRP template example project
https://docs.unity3d.com/2019.4/Documentation/Manual/mesh.html

Mesh





Mesh

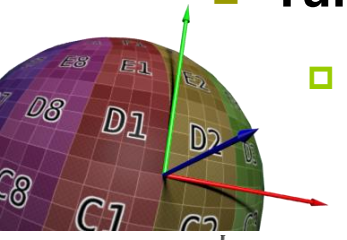
- Unity also supports **deformable meshes**, such as:
  - **Skinned meshes**
    - These meshes work with additional data called **bones**. Bones form a structure called a **skeleton** (also called a **rig**, or **joint hierarchy**), and the skinned mesh contains data that allows it to deform in a realistic way when the skeleton moves. You usually use skinned meshes for skeletal animation with Unity's Animation features, but you can also use them with Rigidbody components to create "ragdoll" effects.
  - **Meshes with blend shapes**
    - These meshes contain data called **blend shapes**. Blend shapes describe versions of the mesh that are deformed into different shapes, which Unity interpolates between. You **use blend shapes for morph target animation**, which is a common technique for facial animation.
  - Meshes with a **Cloth component** for realistic fabric simulation.

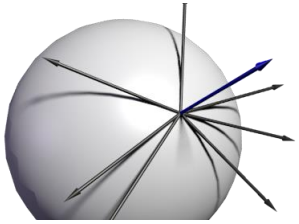
Mesh

- A mesh is defined by these properties:
  - **Vertices** – a collection of **positions** in 3D space, with optional additional **attributes**
  - **Topology** – the type of structure that defines each face of the surface
  - **Indices** – A collection of **integers** that describe how the vertices combine to create the surface, based on the topology
- In addition to this, deformable meshes contain either
  - **Blend shapes** – data that describes different deformed versions of the mesh, for use with animation
  - **Bind poses** – data that describes the "base" pose of the skeleton in a skinned mesh

Vertex Data

- Every **vertex** can have the following attributes:
  - **Position**
    - The **vertex position** represents the position of the vertex in object space. Unity uses this value to determine the surface of the mesh.
  - **Normal**
    - The **vertex normal** represents the direction that points directly "out" from the surface at the position of the vertex. Unity uses this value to calculate the way that light reflects off the surface of a mesh.
  - **Tangent**
    - The **vertex tangent** represents the direction that points along the **"u" (horizontal texture)** axis of the surface at the position of the vertex. Unity uses the w value to compute the binormal, which is the cross product of the tangent and normal. Unity uses the **tangent** and **binormal** values in normal mapping.



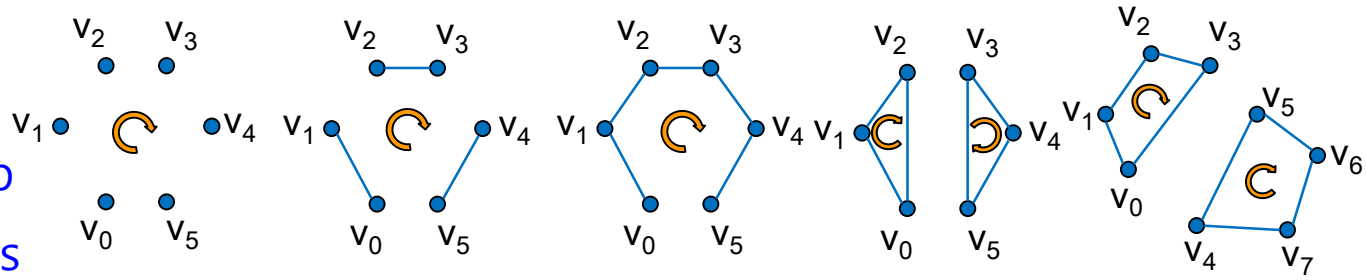


Vertex Data

- **Color**
  - The **vertex color** represents the base color of a vertex, if any. This color exists independently of any textures that the mesh may use.
- **Texture coordinates (UVs)**
  - A mesh can contain up to **8 sets of texture coordinates**. Texture coordinates are commonly called **UVs**, and the sets are called channels. Unity uses texture coordinates when it "wraps" a texture around the mesh. The UVs indicate which part of the texture aligns with the mesh surface at the vertex position.
- **Bone weights and blend indices (skinned meshes only)**
  - In a skinned mesh, **blend indices** indicate which bones affects a vertex, and **bone weights** describe how much influence those bones have on the vertex. Unity uses blend indices and bone weights to deform a skinned mesh based on the movement of its skeleton.

Topology

- **Topology** describes **the type of face** that a mesh has.
  - A mesh's topology defines the structure of the index buffer, which in turn describes how the vertex positions combine into faces. Each type of topology uses a different number of elements in the index array to define a single face.
  - Unity supports the following mesh topologies:
    - Points
    - Lines
    - LineStrip
    - Triangles
    - Quads



  - In the Mesh class, you can get the topology with Mesh.GetTopology, and set it as a parameter of Mesh.SetIndices.

Index Data

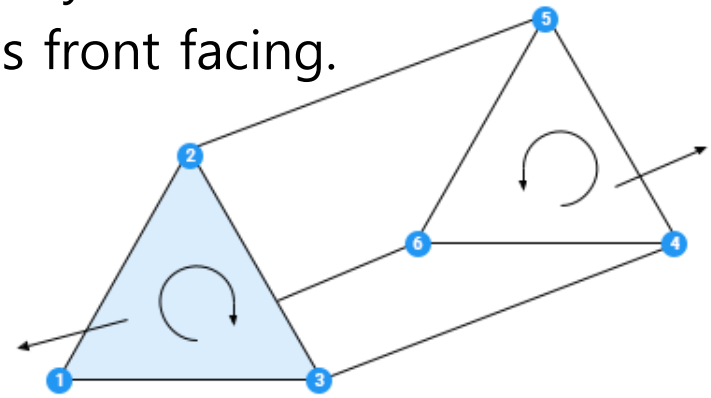
- The **index** array contains **integers** that refer to elements in the vertex positions array.
  - Unity uses the indices to connect the vertex positions into faces. The number of indices that make up **each face depends on the topology of the mesh**.
  - In the Mesh class, you can get this data with Mesh.GetIndices, and set it with Mesh.SetIndices. Unity also stores this data in Mesh.triangles, but this older property is less efficient and user-friendly.
  - **The Points topology doesn't create faces**; instead, Unity renders a single point at each position.
  - All other mesh topologies use more than one index to create either faces or edges.

Index Data

- For example, for a mesh that has an index array that contains the following values: **0,1,2,3,4,5**. If the mesh has a **triangular** topology, then the first three elements **(0,1,2)** identify **one triangle**, and the following three elements **(3, 4, 5)** identify **another triangle**.
- There is no limit to the number of faces that a vertex can contribute to. This means that the same vertex can appear in the index array multiple times. For example, an index array could contain these values: **0,1,2,1,2,3**. If the mesh has a triangular topology, then the first three elements **(0,1,2)** identify **one triangle**, and the following three elements **(1,2,3)** identify **another triangle** that shares vertices with the first.

Winding Order

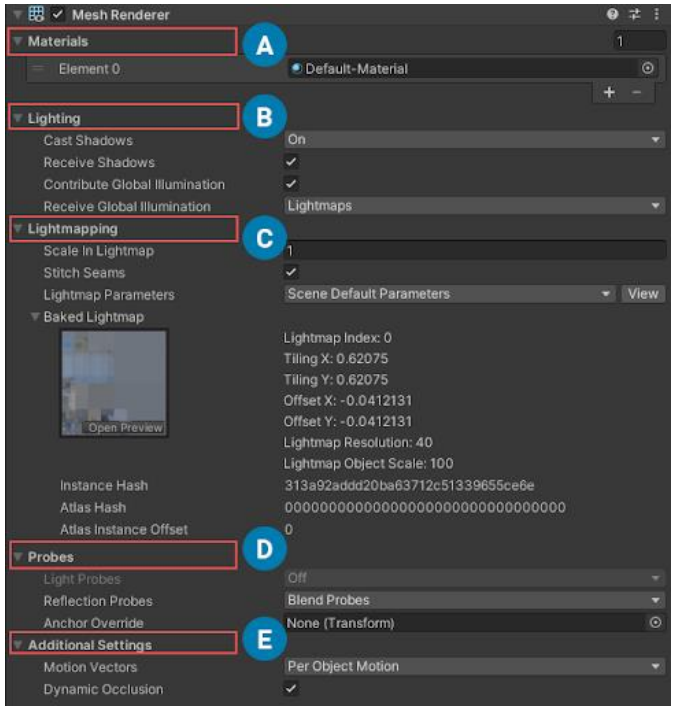
- The order of the vertices in each group in the index array is called the **winding order**.
  - Unity uses winding order to determine whether a face is **front-facing** or **back-facing**, and in turn whether it should **render a face** or **cull it (exclude it from rendering)**.
  - By default, Unity renders front-facing polygons and culls back-facing polygons.
  - Unity uses a **clockwise winding order**, which means that Unity considers any face where the indices connect in a clockwise direction is front facing.



https://docs.unity3d.com/Manual/AnatomyofaMesh.html

Mesh Renderer Component

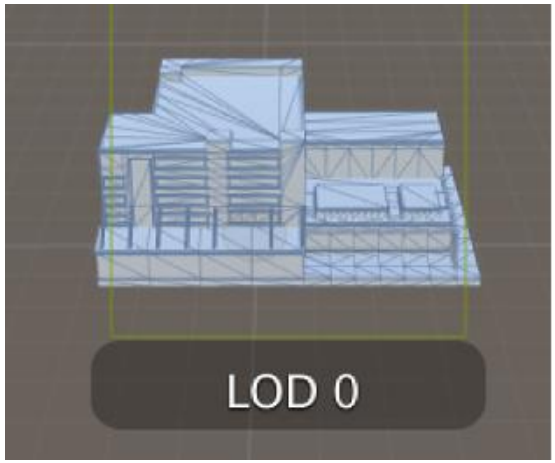
- A **Mesh Renderer component** renders a mesh.
  - It works with a **Mesh Filter component** on the same GameObject; the Mesh Renderer renders the mesh that the Mesh Filter references.
  - To render a **deformable mesh**, use a **Skinned Mesh Renderer** instead.

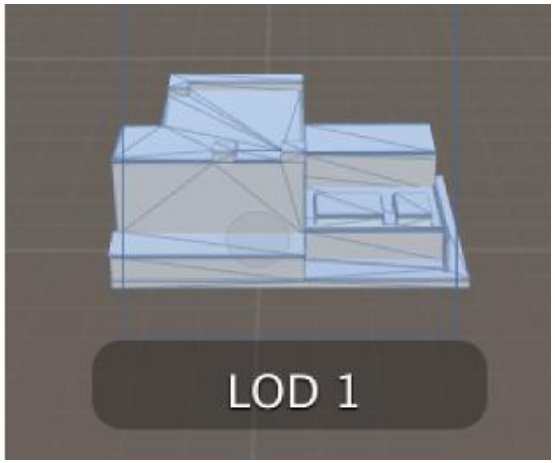


https://docs.unity3d.com/Manual/class-MeshRenderer.html

Level of Detail (LOD) for Meshes

- **Level Of Detail (LOD)**
  - LOD is a technique reduces the number of GPU operations that Unity requires to render distant meshes.
  - When a GameObject in the Scene is **far away** from the Camera, you see **less detail** compared to when it is close to the Camera.
  - By default, Unity uses the same number of triangles to render it at both distances. This can result in wasted GPU operations, which can impact performance in your Scene.





https://docs.unity3d.com/Manual/LevelOfDetail.html

Level of Detail (LOD) for Meshes

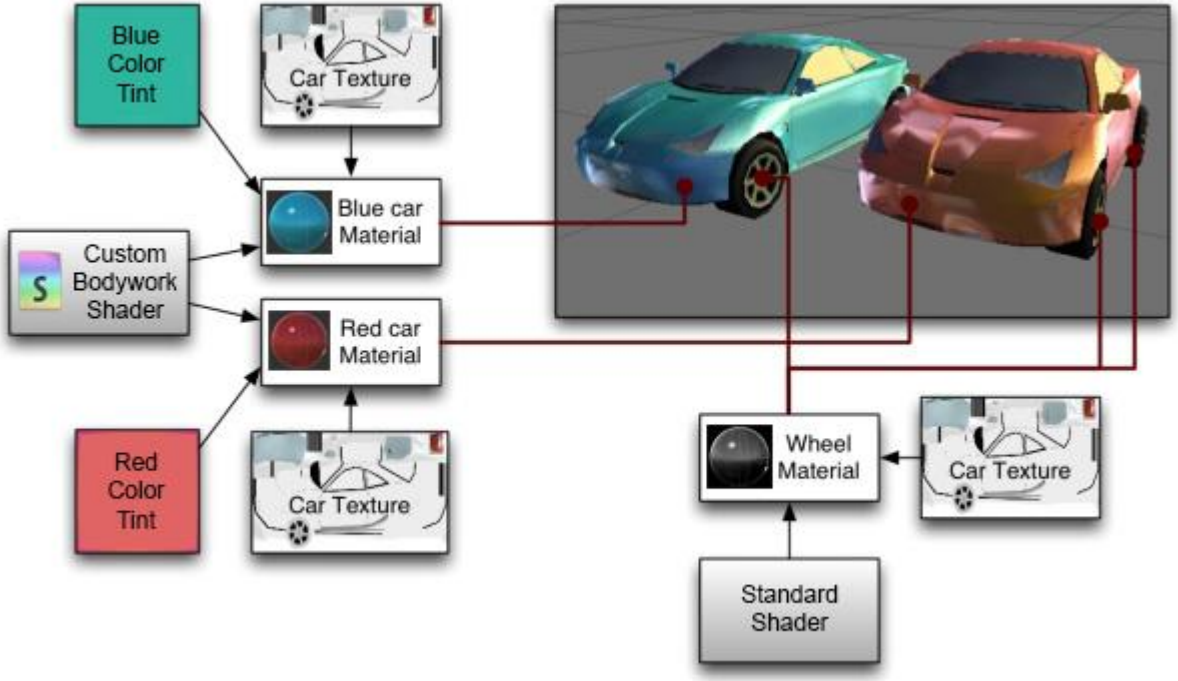
- LOD Levels
  - A LOD level is a mesh that defines the level of detail Unity renders for a GameObject's geometry.
  - When a GameObject uses LOD, Unity displays the appropriate LOD level for that GameObject based on the GameObject's distance from the Camera.
  - Each LOD level exists in a separate GameObject, each of which has a Mesh Renderer component that displays that LOD level.
  - For the very lowest level of detail, you can use a **Billboard Asset**, which Unity displays instead of a 3D mesh. Unity shows and hides these GameObjects as required. LOD levels must be child GameObjects to the GameObject they relate to.

Materials, Shaders & Textures

- Rendering in Unity uses Materials, Shaders and Textures.
  - **Materials** define **how a surface should be rendered**, by including references to the Textures it uses, tiling information, Color tints and more.
  - **Shaders** are **small scripts** that contain the mathematical calculations and algorithms for calculating the Color of each pixel rendered, based on the **lighting** input and the **Material** configuration.
  - **Textures** are **bitmap images**. A Material can contain references to textures, so that the Material's Shader can use the textures while calculating the surface color of a GameObject. In addition to basic Color (Albedo) of a GameObject's surface, Textures can represent many other aspects of a Material's surface such as its reflectivity or roughness.

Materials, Shaders & Textures

- Example of using **3 materials, 2 shaders and 1 texture**.
  - "Red car material", "Blue car material", "Wheel material"
  - Both bodywork materials use the same custom shader, "Carbody Shader". Wheel material use "Standard Shader".
  - Each car body material has a reference to the "Car Texture".



https://docs.unity3d.com/2018.4/Documentation/Manual/Materials.html

Material

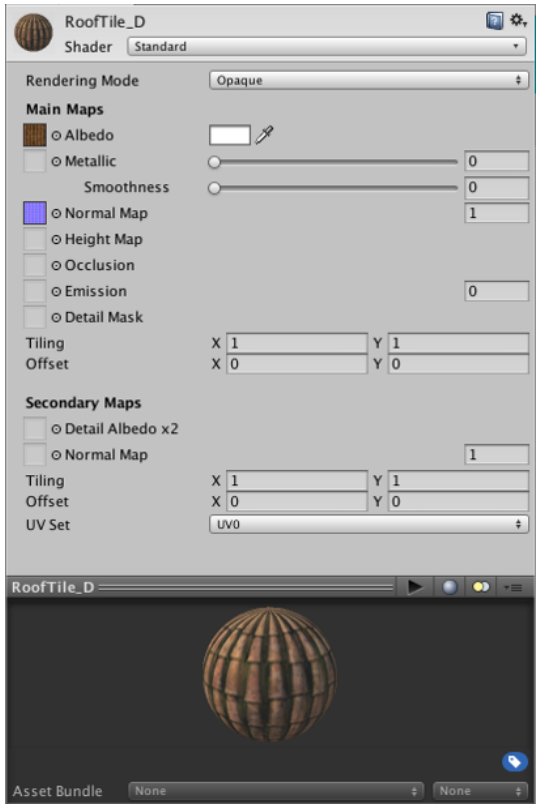
- Materials
  - A **Material** specifies one specific **Shader** to use, and the Shader used determines which options are available in the Material. A Shader specifies one or more **Texture** variables that it expects to use, and the Material Inspector in Unity allows you to assign your own Texture Assets to these Texture variables.
  - A **Material** contains a reference to a **Shader object**. If that Shader object defines material properties, then the material can also contain data such as colors or references to textures.
  - The **Material class** represents a material in C# code.
  - A **Material asset** is a file with the **.mat** extension. It represents a material in your Unity project.

Material

- Assigning a material asset to a GameObject
  - Materials are used by **Renderer components** attached to **Game Objects**, to render each Game Object's mesh.
  - To render a GameObject using a material:
    1. Add a component that inherits from Renderer. **MeshRenderer** is the most common and is suitable for most use cases, but **SkinnedMeshRenderer, LineRenderer, or TrailRenderer** might be more suitable if your GameObject has special requirements.
    2. Assign the material asset to the component's **Material** property.
  - To render a particle system in the **Built-in Particle System** using a material:
    1. Add a **Renderer Module** to the Particle System.
    2. Assign the material asset to the Renderer Module's **Material** property.

Material Inspector

- Material Inspector
  - When you select a **material asset** in your Unity project, you can view and edit it using the **Inspector window**.



https://docs.unity3d.com/Manual/Materials.html

Using Materials with C# Scripts

- All the **parameters of a material asset** in the Inspector window are accessible via script, giving you the power to change or animate how a material works at runtime.
- This allows you to modify numeric values on the material, change colors, and swap textures dynamically during gameplay.
- Some of the most commonly used functions are:

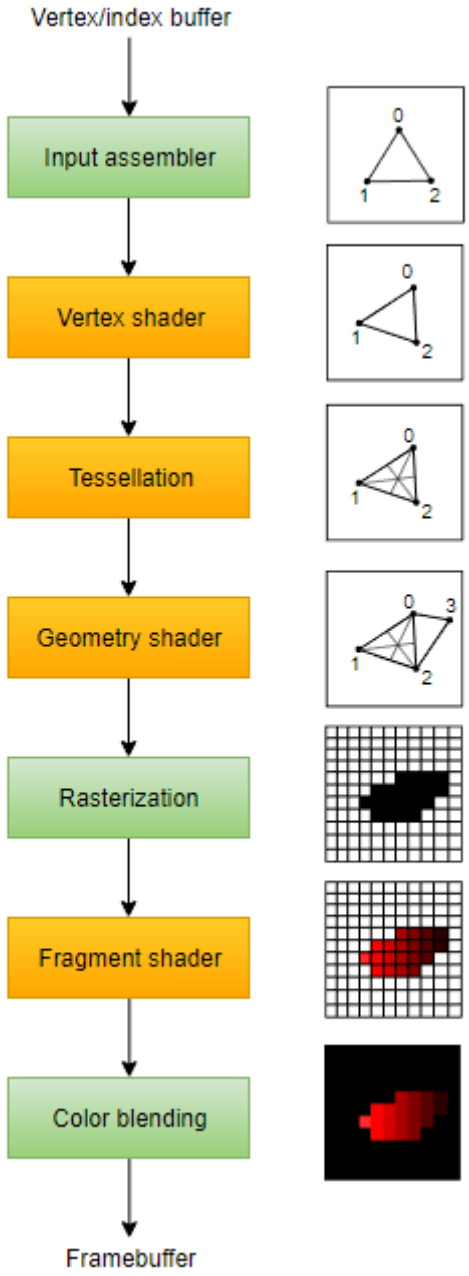
Function Name	Use
SetColor	Change the color of a material (e.g. the albedo tint color)
SetFloat	Set a floating point value (e.g. the normal map multiplier)
SetInt	Set an integer value in the material
SetTexture	Assign a new texture to the material

Shader

- Shader
  - A **Shader** is a **script** which contains mathematical calculations and algorithms for how the pixels on the surface of a model should look.
  - The **standard shader** performs complex and realistic lighting calculations. **Other shaders** may use simpler or different calculations to show different results.
  - Within any given Shader are a number of **properties** which can be given values by a Material using that shader. These properties can be **numbers, colors definitions or textures**, which appear in the Material inspector window.
  - It is possible and often desirable to have several different Materials which may reference the same textures. These materials may also use the same or different shaders, depending on the requirements.

Shader

- Types of shader
  - **Shaders that are part of the graphics pipeline** are the most common type of shader. They perform calculations that determine the color of pixels on the screen. In Unity, you usually work with this type of shader by using **Shader objects**.
  - **Compute shaders** perform calculations on the GPU, outside of the regular graphics pipeline.
  - **Ray tracing shaders** perform calculations related to ray tracing.



https://gamedevbill.com/unity-vertex-shader-and-geometry-shader-tutorial/

Shader

- Terminology
  - **shader** or **shader program** - **a program that runs on a GPU**. Unless otherwise specified, this means shader programs that are part of the graphics pipeline.
  - **Shader object** - an instance of the Shader class. A Shader object is a wrapper for shader programs and other information.
  - **ShaderLab** - **a Unity-specific language for writing shaders**.
  - **Shader Graph** - a tool for creating shaders without writing code.
  - **shader asset** - a file with the **.shader** extension in your Unity project. It defines a Shader object.
  - **Shader Graph asset** - a file in your Unity project. It defines a Shader object.

Standard Shader

- Standard Shader
  - The Standard Shader is designed with hard surfaces in mind (also known as “architectural materials”), and can deal with most real-world materials like **stone, glass, ceramics, brass, silver or rubber**. It will even do a decent job with non-hard materials like skin, hair and cloth.

https://docs.unity3d.com/Manual/shader-StandardShader.html

Standard Shader

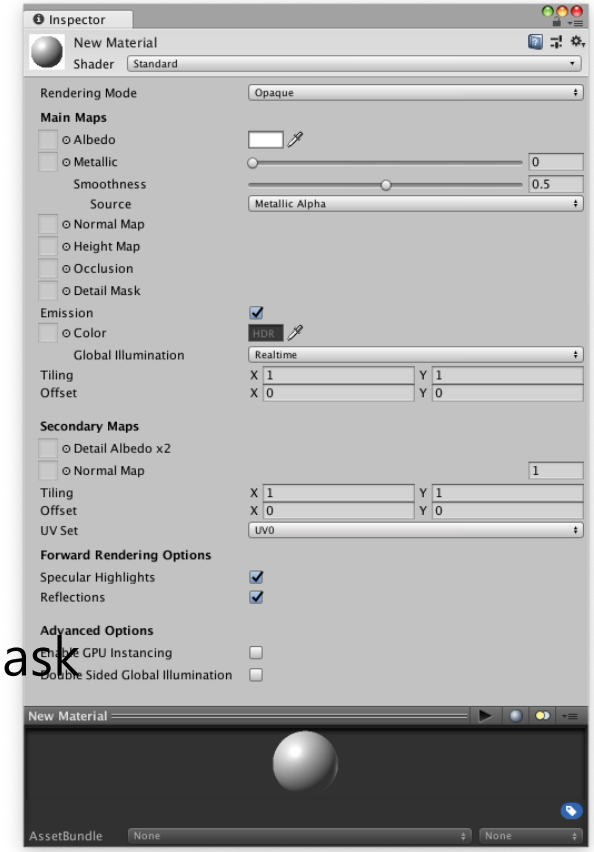
- Standard Shader
  - The Unity Standard Shader is a **built-in shader** with a comprehensive set of features.
  - It can be used to render “real-world” objects such as stone, wood, glass, plastic and metal, and supports a wide range of shader types and combinations.
  - Its features are enabled or disabled by simply using or not using the various texture slots and parameters in the material editor.
  - With the Standard Shader, a large range of shader types (such as **Diffuse, Specular, Bumped Specular, Reflective**) are combined into a single shader intended to be used across all material types.
  - The benefit of this is that the same lighting calculations are used in all areas of your scene, which gives a realistic, consistent and believable distribution of light and shade across all models that use the shader.

Standard Shader

- Standard Shader
  - The Standard Shader also incorporates an advanced lighting model called **Physically Based Shading (PBS)**.
  - Physically Based Shading (PBS) simulates the interactions between materials and light in a way that mimics reality. PBS has only recently become possible in real-time graphics. It works at its best in situations where lighting and materials need to exist together intuitively and realistically.
  - The idea behind our Physically Based Shader is to create a user-friendly way of achieving a consistent, plausible look under different lighting conditions. It models how light behaves in reality, without using multiple ad-hoc models that may or may not work. To do so, it follows principles of physics, including energy conservation (meaning that objects never reflect more light than they receive), Fresnel reflections (all surfaces become more reflective at grazing angles), and how surfaces occlude themselves (what is called Geometry Term), among others.

Standard Shader

- Material Parameters
  - Rendering Mode – Opaque, Cutout, Transparent, Fade
  - Albedo Color and Transparency
  - Specular Mode
  - Metallic Mode
  - Smoothness
  - Normal Map (Bump mapping)
  - Height Map
  - Occlusion Map
  - Emission
  - Secondary Maps (Detail Maps) & Detail Mask
  - The Fresnel Effect



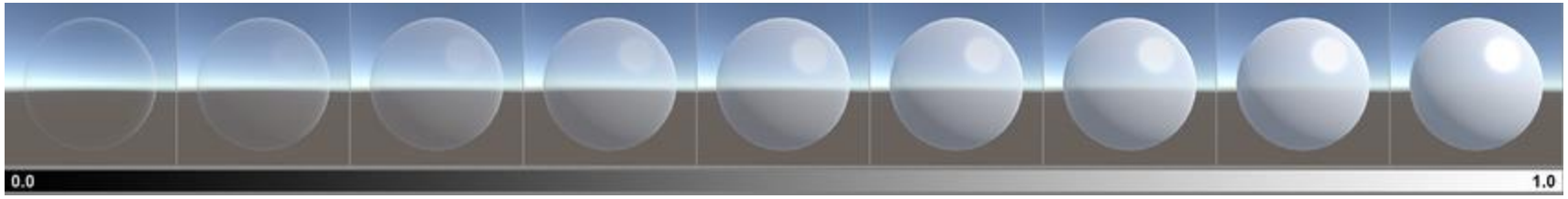
https://docs.unity3d.com/Manual/StandardShaderMaterialParameters.html

Standard Shader

- Rendering Mode
  - **Opaque** - normal *solid* objects with no transparent areas.
  - **Cutout** - create a transparent effect that has hard edges between the *opaque and transparent* areas. In this mode, there are no semi-transparent areas, the texture is either 100% opaque, or invisible. This is useful when using transparency to create the shape of materials such as leaves, or cloth with holes.
  - **Transparent** - rendering realistic *transparent* materials such as clear plastic or glass. In this mode, the material itself will take on transparency values, however reflections and lighting highlights will remain visible at full clarity as is the case with real transparent materials.
  - **Fade** - allows the transparency values to entirely *fade* an object out, including any specular highlights or reflections it may have. This mode is useful if you want to animate an object fading in or out.

Standard Shader

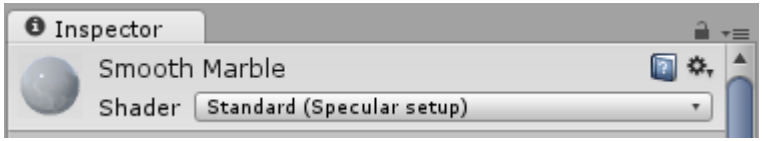
- Albedo Color and Transparency
  - The **Albedo** parameter controls the base color of the surface.
  - Specifying a single color for the Albedo value is sometimes useful, but it is far more common to assign a **texture** map for the Albedo parameter.
  - The alpha value of the Albedo color controls the **transparency** level for the material. This only has an effect if the Rendering Mode for the material is set to one of the transparent mode, and not **Opaque**.



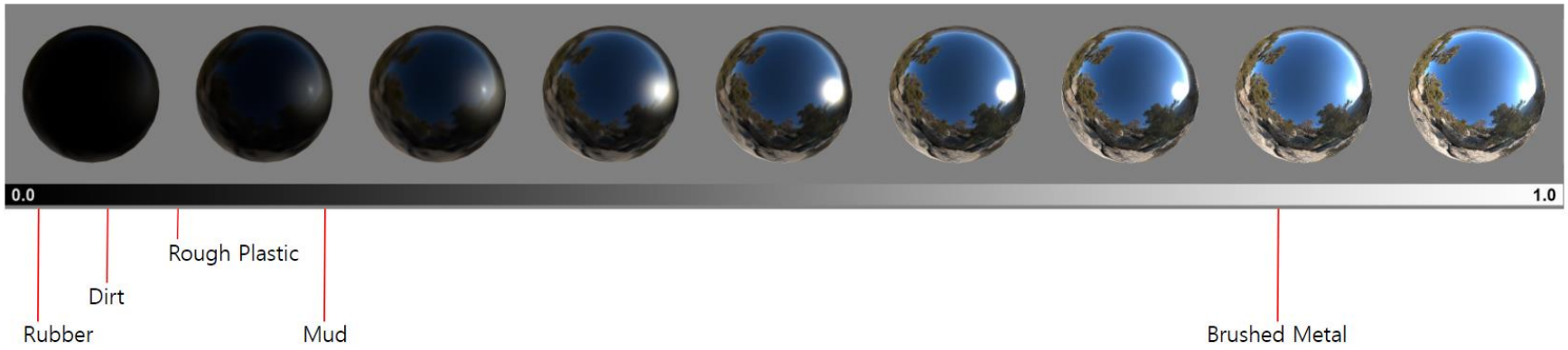
A range of transparency values from 0 to 1, using the Transparent mode suitable for realistic transparent objects

https://docs.unity3d.com/Manual/StandardShaderMaterialParameterAlbedoColor.html

Standard Shader



- Specular Mode
  - The **Specular** parameter is only visible when using the **Specular setup**, as shown in the **Shader** field.
  - Specular effects are essentially the direct reflections of light sources in your **Scene**, which typically show up as bright highlights and shine on the surface of objects.

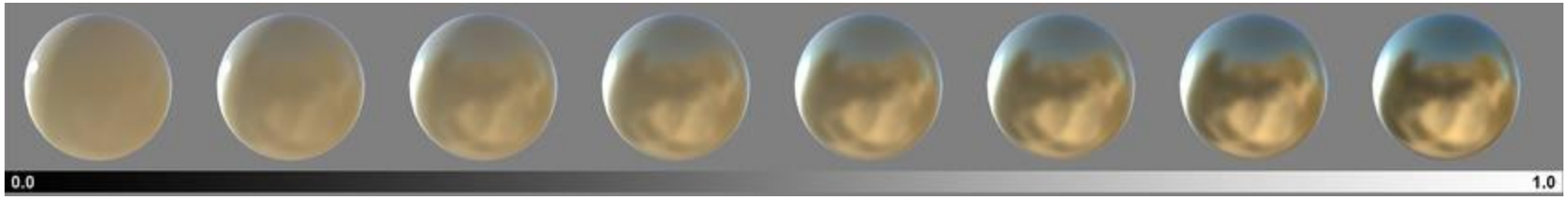


The Specular Smoothness values from 0 to 1

https://docs.unity3d.com/Manual/StandardShaderMaterialParameterSpecular.html

Standard Shader

- Metallic Mode
  - The metallic parameter of a material determines how "metal-like" the surface is.
  - When a surface is more metallic, it reflects the environment more and its albedo color becomes less visible.
  - At full metallic level, the surface color is entirely driven by reflections from the environment. When a surface is less metallic, its albedo color is more clear and any surface reflections are visible on top of the surface color, rather than obscuring it.

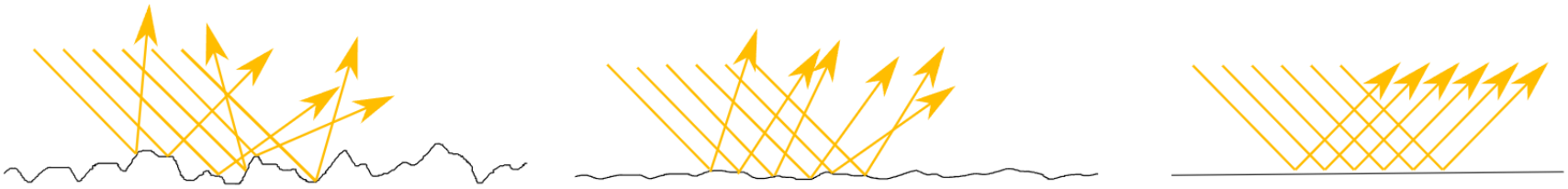


A range of metallic values from 0 to 1 (with smoothness at a constant 0.8 for all samples)

https://docs.unity3d.com/Manual/StandardShaderMaterialParameterMetallic.html

Standard Shader

- Smoothness
  - With a **smooth surface**, all light rays tend to bounce off at predictable and consistent angles. Taken to its extreme, a **perfectly smooth surface** reflects light like a **mirror**.
  - **Less smooth surfaces** reflect light over a wider range of angles (as the light hits the bumps in the microsurface), and therefore the reflections have less detail and are spread across the surface in a more **diffuse way**.

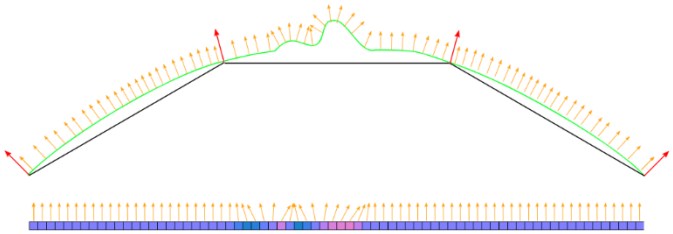


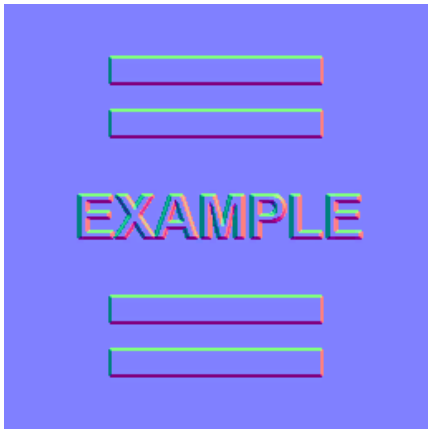
A comparison of low, medium and high values for smoothness (left to right), as a diagram of the theoretical microsurface detail of a material. The yellow lines represent light rays hitting the surface and reflecting off the angles encountered at varying levels of smoothness.

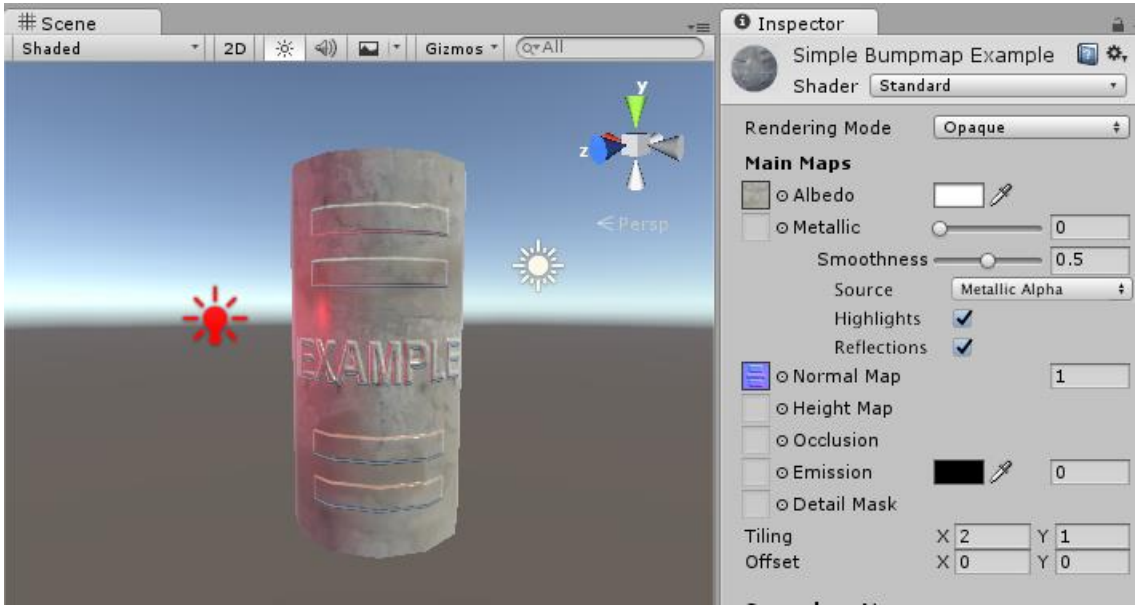
https://docs.unity3d.com/Manual/StandardShaderMaterialParameterSmoothness.html

Standard Shader

- Normal maps (Bump mapping)
  - **Normal maps** are used by normal map Shaders to make low-polygon models look as if they contain more detail. Unity uses normal maps encoded as RGB images. You also have the option to generate a normal map from a grayscale height map image.







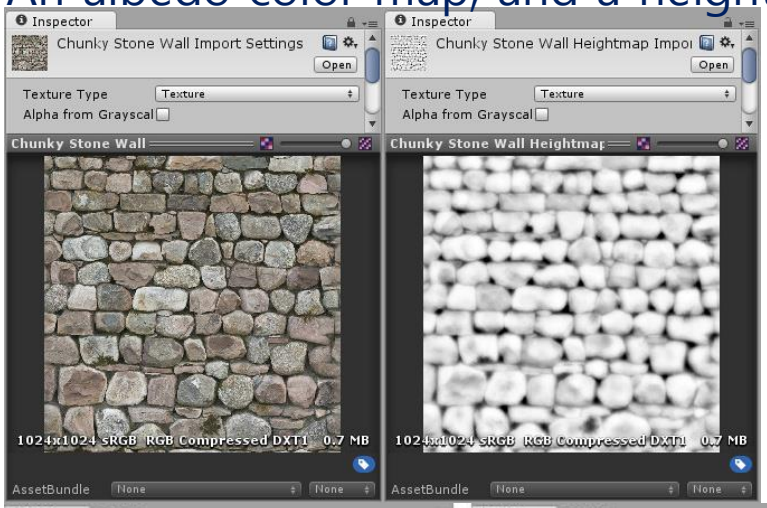
https://docs.unity3d.com/Manual/StandardShaderMaterialParameterNormalMap.html

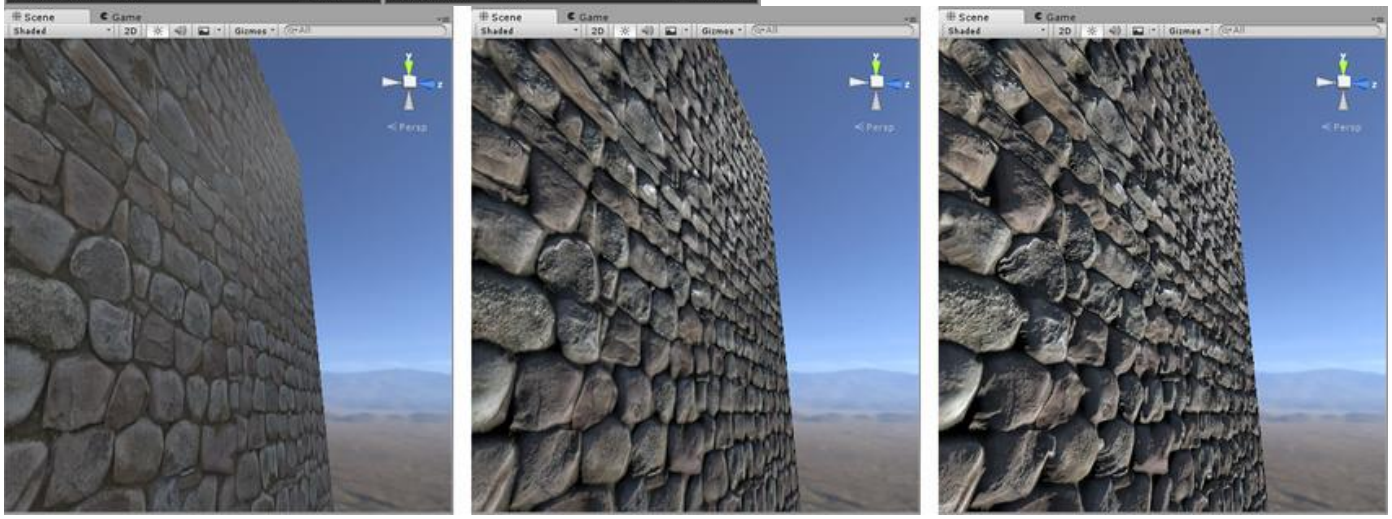
Standard Shader

- Heightmap (Parallax Mapping)
  - Height mapping (also known as parallax mapping) is a similar concept to normal mapping, however this technique is more complex - and therefore also more performance-expensive.
  - Heightmaps are usually used in conjunction with normalmaps, and often they are used to give extra definition to surfaces where the texture maps are responsible for rendering large bumps and protrusions.
  - While normal mapping modifies the lighting across the surface of the texture, parallax height mapping goes a step further and actually shifts the areas of the visible surface texture around, to achieve a kind of surface-level occlusion effect.
  - This means that apparent bumps will have their near side (facing the camera) expanded and exaggerated, and their far side (facing away from the camera) will be reduced and seem to be occluded from view.

Standard Shader

An albedo color map, and a heightmap to match.



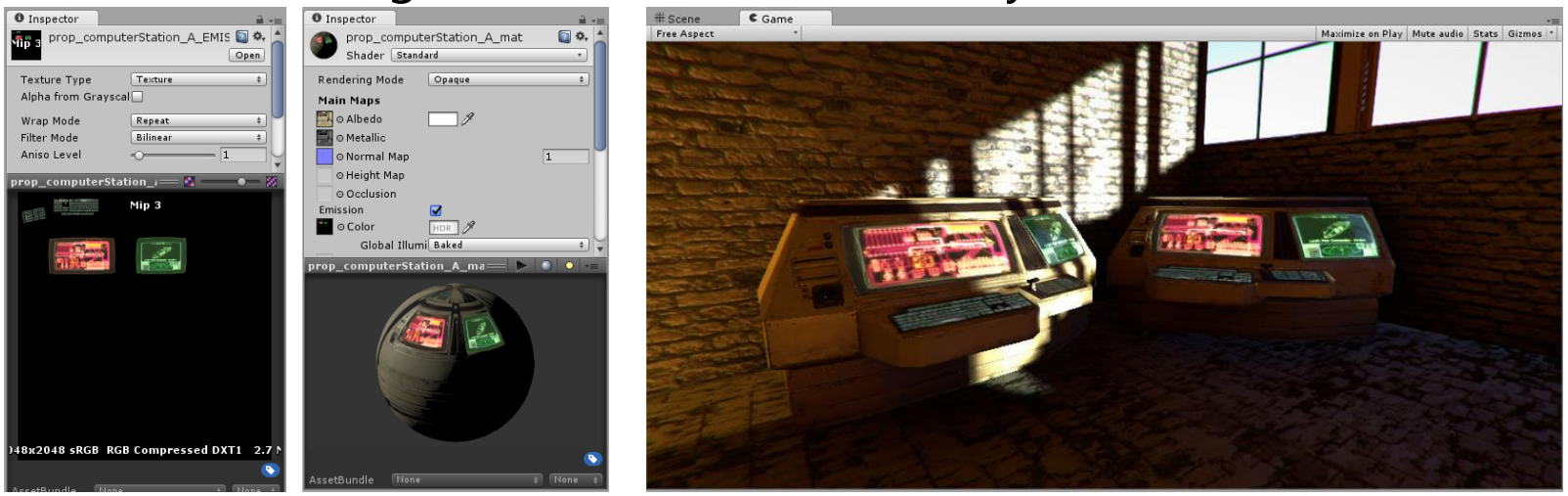


1. No normalmap or heightmap, 2. Normal map, 3. Normal map and heightmap

https://docs.unity3d.com/Manual/StandardShaderMaterialParameterHeightMap.html

Standard Shader

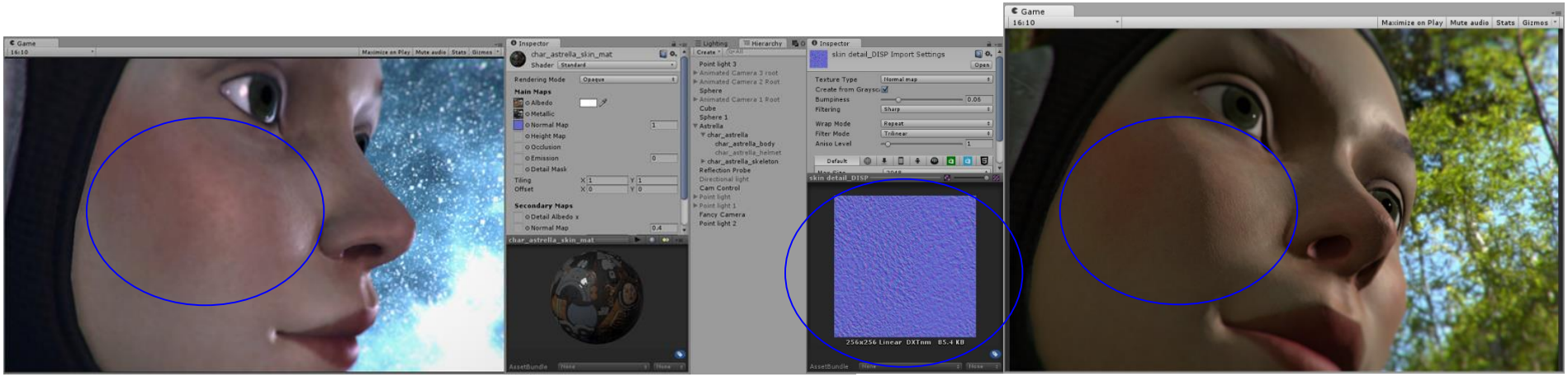
- Emission
  - The Material emission properties control the color and intensity of light that the surface of a Material **emits**.
  - Emission is useful when you want some part of a GameObject to appear lit from the inside, such as **the screen of a monitor**, the disc brakes of a car braking at high speed, or glowing buttons on a control panel. GameObjects that use emissive Materials appear to remain bright even in dark areas of your Scene.



https://docs.unity3d.com/Manual/StandardShaderMaterialParameterEmission.html

Standard Shader

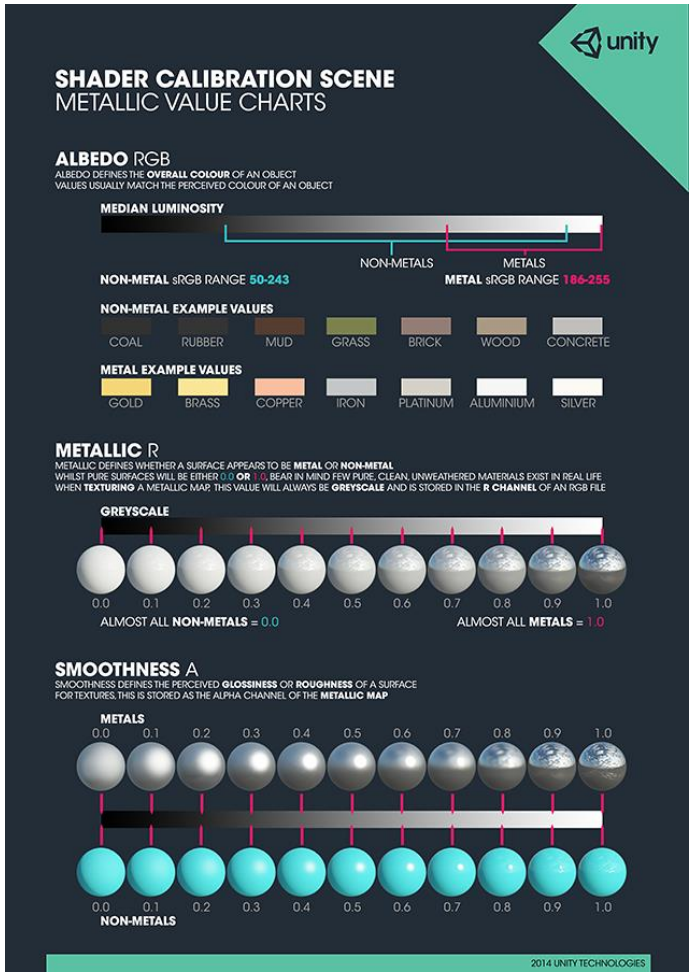
- Detail maps (Secondary maps)
  - Secondary Maps (or Detail maps) allow you to overlay a second set of textures on top of the main textures listed above.
  - You can apply a **second Albedo color map**, and a **second Normal map**.
  - Typically, these would be mapped on a much smaller scale repeated many times across the object's surface, compared with the main Albedo and Detail maps.

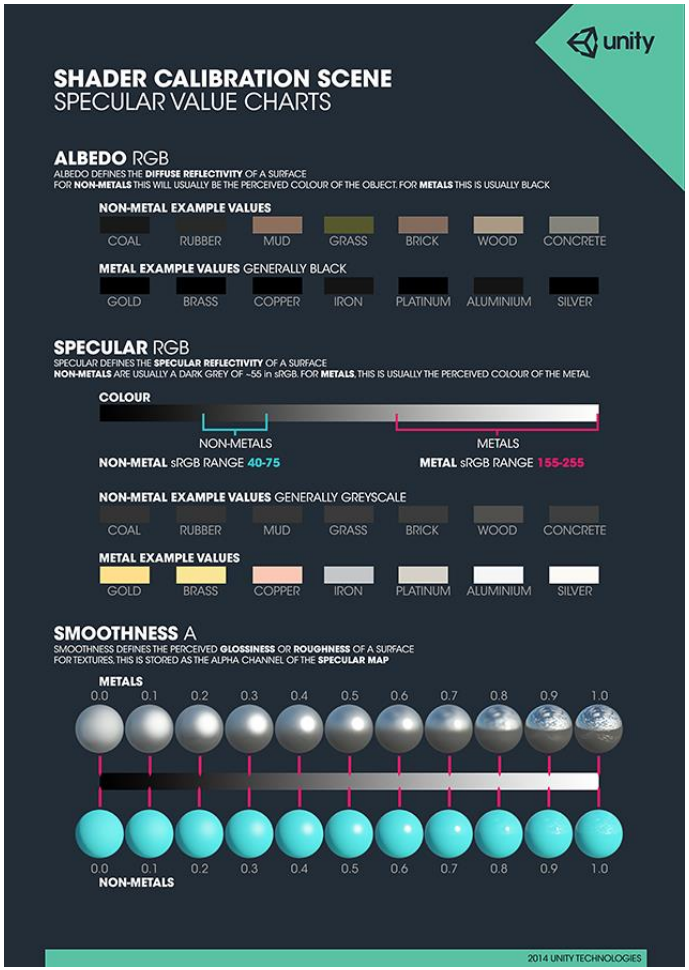


https://docs.unity3d.com/Manual/StandardShaderMaterialParameterDetail.html

Standard Shader

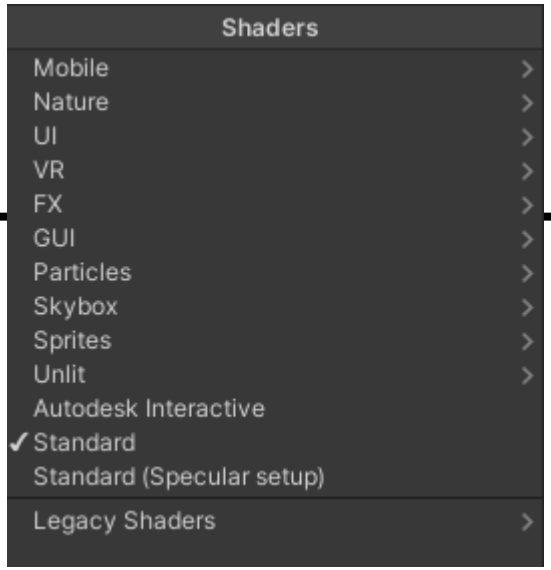
- Material Charts





https://docs.unity3d.com/2018.4/Documentation/Manual/StandardShaderMaterialCharts

Other Built-in Shaders



- In addition to the Standard Shader, there are a number of built-in shaders:
  - FX: Lighting and glass effects.
  - GUI and UI: For user interface graphics.
  - Mobile: Simplified high-performance shader for mobile devices.
  - Nature: For trees and terrain.
  - Particles: Particle system effects.
  - Skybox: For rendering background environments behind all geometry
  - Sprites: For use with the 2D sprite system
  - Unlit: For rendering that entirely bypasses all light & shadowing
  - Legacy: The large collection of older shaders which were superseded by the Standard Shader

ShaderLab

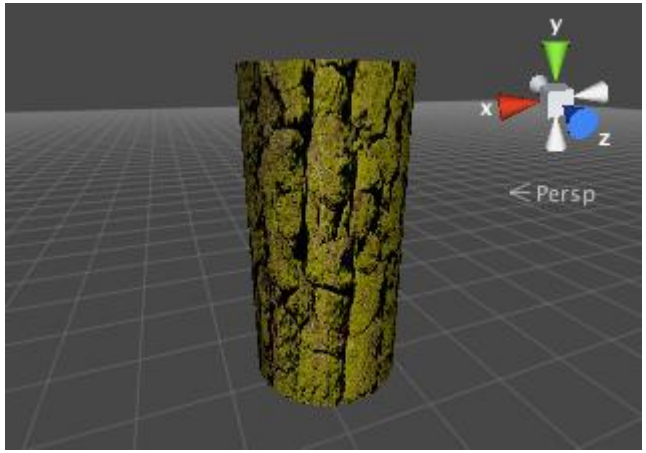
- ShaderLab
  - Properties
  - SubShader
  - Fallback
  - CustomEditor

```
Shader "Examples/ShaderSyntax"
{
    CustomEditor = "ExampleCustomEditor"
    Properties
    {
        // Material property declarations go here
    }
    SubShader
    {
        // The code that defines the rest of the SubShader goes here

        Pass
        {
           // The code that defines the Pass goes here
        }
    }
    Fallback "ExampleFallbackShader"
}
```

Texture

- Texture
  - Normally, the **mesh** geometry of an object only gives a rough approximation of the **shape** while most of the **fine detail** is supplied by **Textures**.
  - A texture is just a standard bitmap image that is applied over the mesh surface.
  - The positioning of the texture is done with the 3D modelling software that is used to create the mesh.



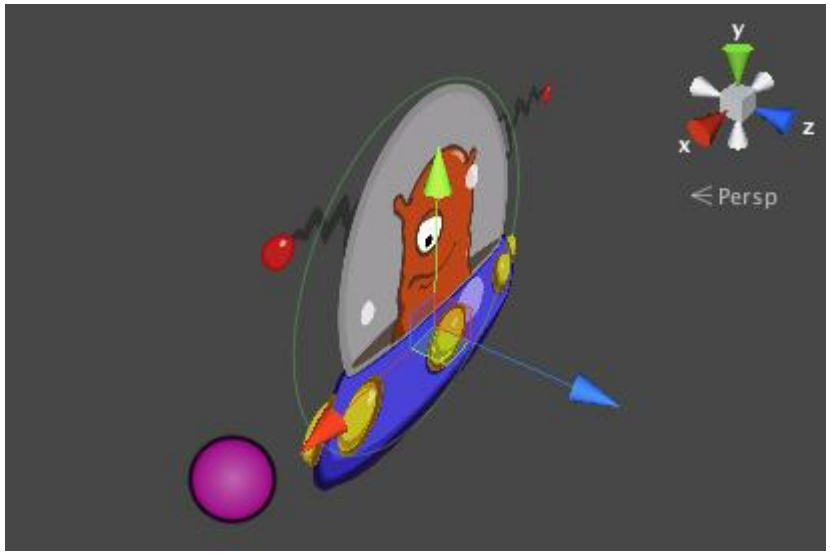
https://docs.unity3d.com/Manual/Textures.html

Texture

- Textures for use on 3D models
  - Textures are applied to objects using **Materials**. Materials use specialized graphics programs called **Shaders** to render a texture on the mesh surface.
  - You should make your **textures in dimensions** that are to the **power of two** (e.g. 32x32, 64x64, 128x128, 256x256, etc.)
  - Once your texture has been imported, you should assign it to a Material. The material can then be applied to a mesh, Particle System, or GUI Texture.
  - Using the **Import Settings**, it can also be converted to a **Cubemap** or **Normalmap** for different types of applications in the game.

Sprite Texture

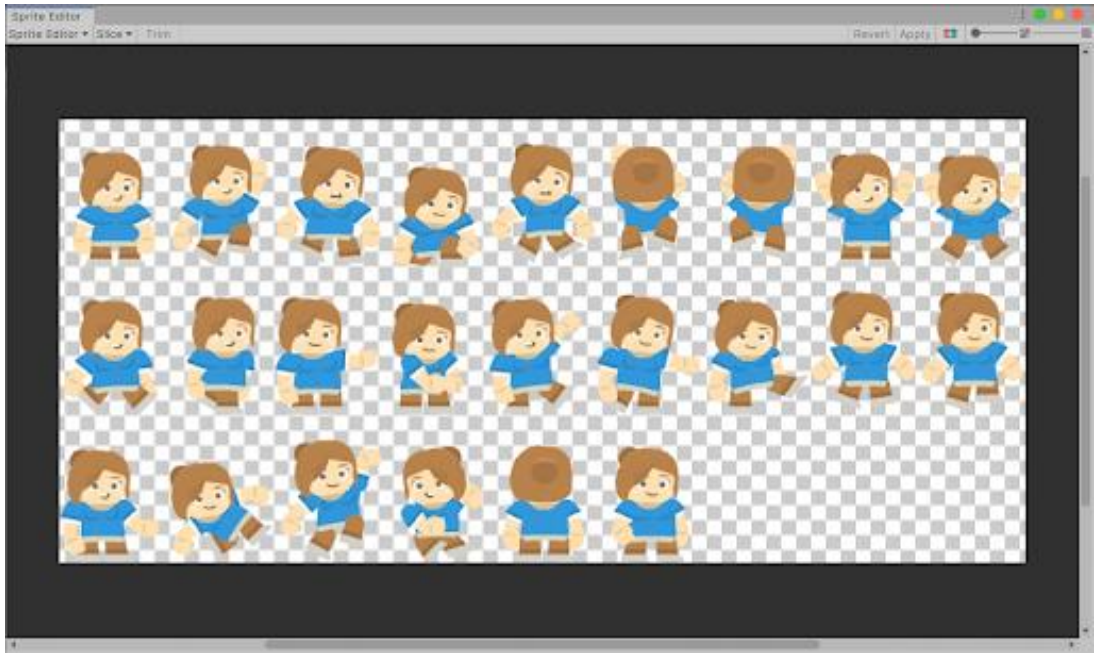
- Sprite Textures for 2D Graphics
  - In **2D games**, the **Sprites** are implemented using textures applied to flat meshes that approximate the objects' shapes.
  - An object in a 2D game may require a set of related graphic images to represent animation frames or different states of a character. Special techniques are available to allow these sets of images to be designed and rendered efficiently.



https://docs.unity3d.com/Manual/Textures.html

Sprite Texture

- In computer graphics or games, **a sprite is a 2D image or animation** that is integrated into a larger scene.
- Originally invented as a method of quickly compositing several images together in 2D video games.
- In general, 2D game figures are all referred to as sprites.



https://learn.unity.com/tutorial/introduction-to-sprite-animations

GUI Texture

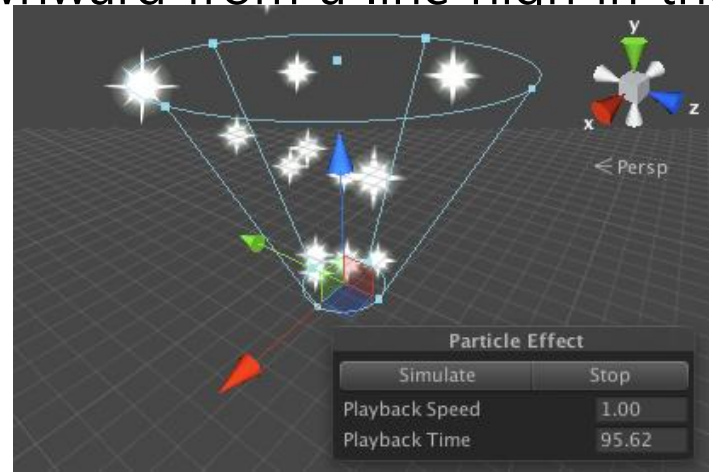
- GUI
  - A game's graphic user interface (GUI) consists of graphics that are not used directly in the game scene but are there to allow the player to make choices and see information.
  - For example, the score display and the options menu are typical examples of game GUI. These graphics are clearly very different from the kind used to detail a mesh surface but they are handled using standard Unity textures nevertheless.

https://docs.unity3d.com/kr/2018.4/Manual/UICanvas.html

Particle Systems

Particles

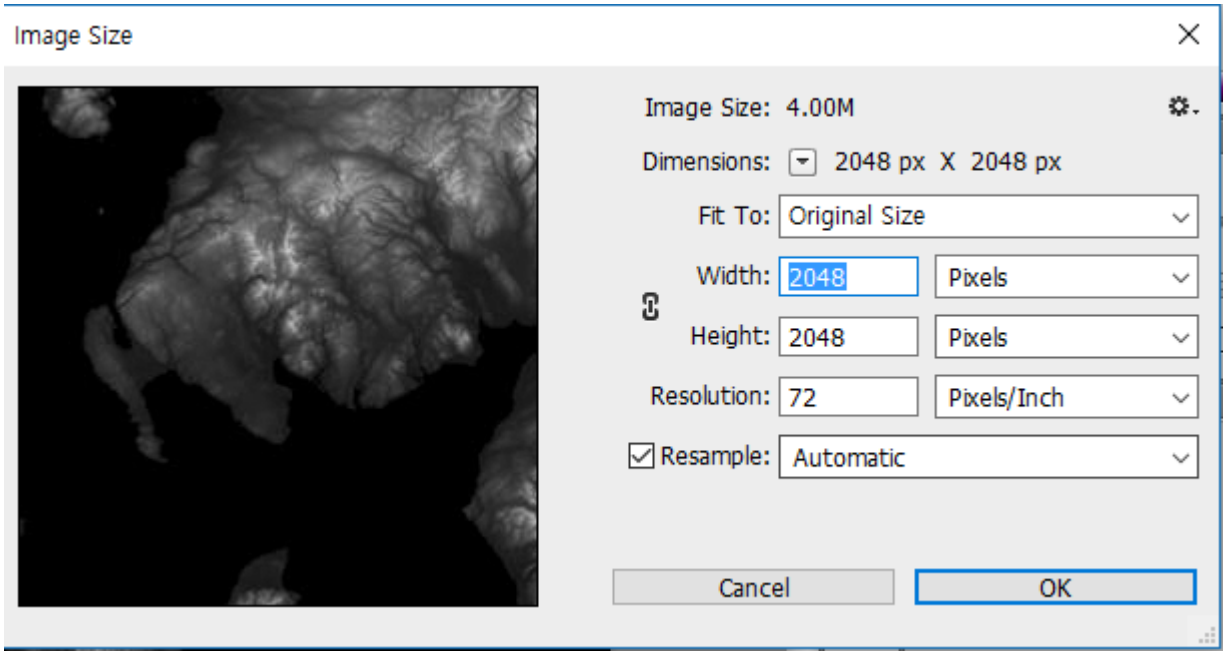
- A particle is a small 2D graphic representing a small portion of something that is basically fluid or gaseous.
- When many of these particles are created at once and set in motion, optionally with random variations, they can create a very convincing effect.
- For example, you might display an explosion by sending particles with a fire texture out at great speed from a central point. A waterfall could be simulated by accelerating water particles downward from a line high in the scene.



https://docs.unity3d.com/Manual/Textures.html

Terrain Heightmaps

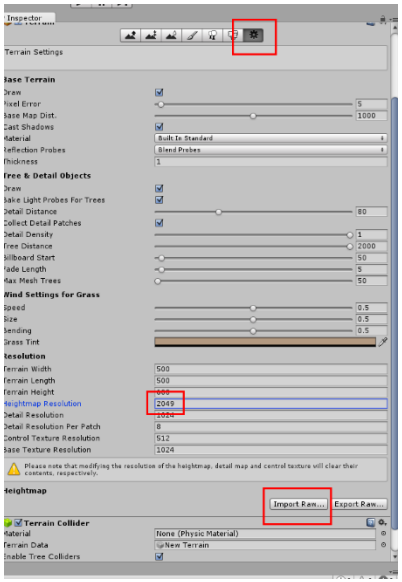
- Terrain Heightmaps
  - Textures can even be used in cases where the image will never be viewed at all, at least not directly. In a greyscale image, each pixel value is simply a number corresponding to the shade of grey at that point in the image (this could be a value in the range 0..1 where zero is black and one is white, say).

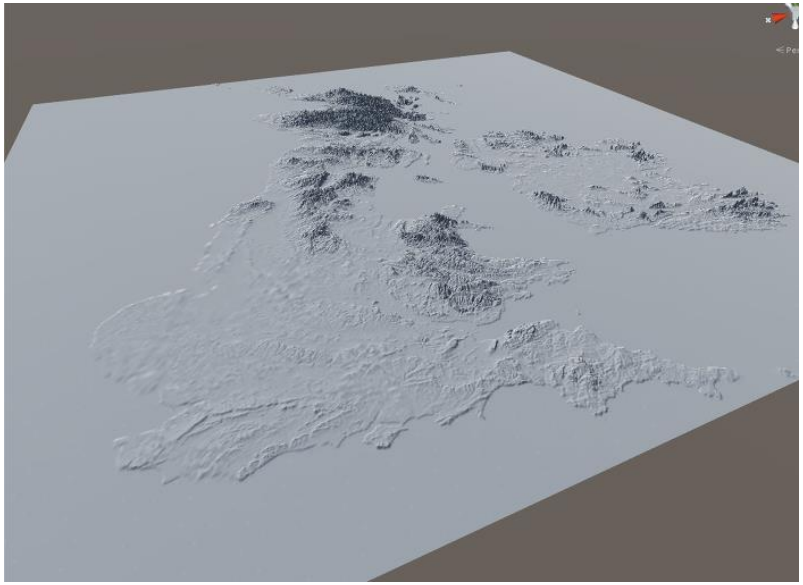


https://chulin28ho.tistory.com/344

Terrain Heightmaps

- Terrain Heightmaps
  - A **terrain** is a mesh representing an area of ground where each point on the ground has a particular height from a baseline. The **heightmap** for a terrain stores the **numeric height** samples at regular intervals as greyscale values in an image where each pixel corresponds to a grid coordinate on the ground. The values are not shown in the scene as an image but are converted to coordinates that are used to **generate the terrain mesh**.





https://chulin28ho.tistory.com/344

Importing Textures

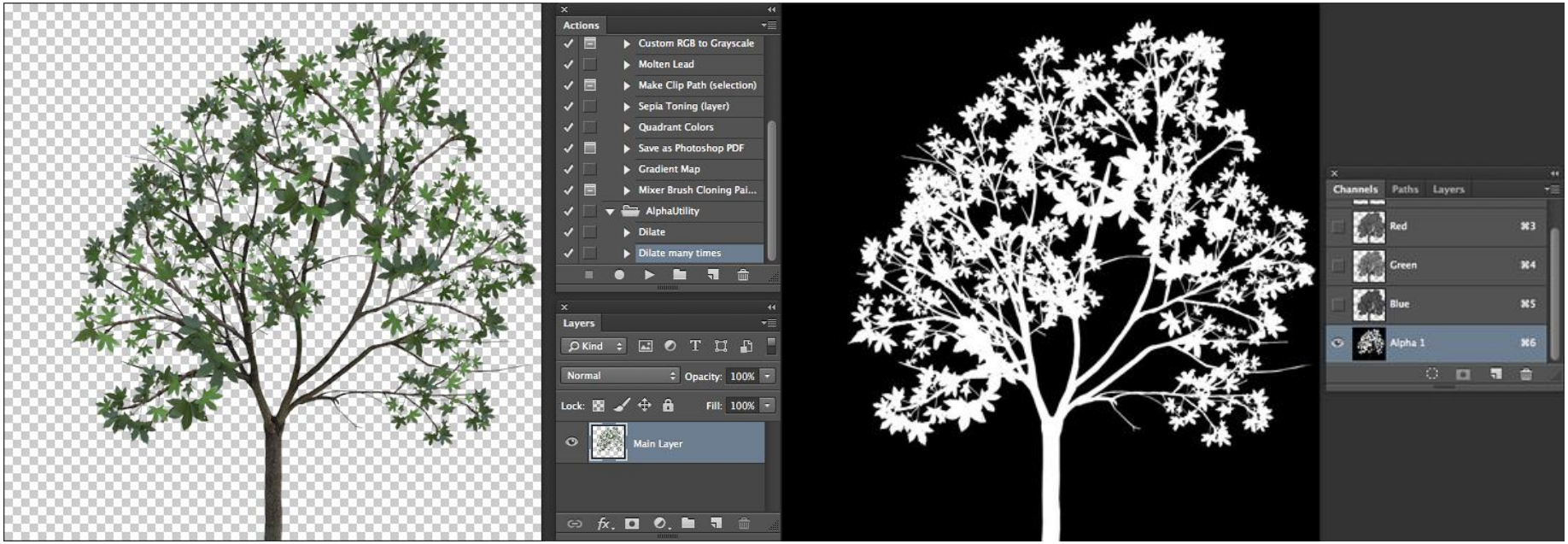
- Importing Textures
  - In a 3D Project, Unity imports image and movie files in the Assets folder as Textures. In a 2D Project, Unity imports image and movie files in the Assets folder as Sprites.
  - To import image and movie files as Textures and Sprites in Unity
    1. Select the image file in the Project window.
    2. In the Inspector, set the Texture Import Settings.
    3. Click the Apply button to save the changes.
    4. To use the imported Assets in your Project:
       - For 3D Projects, create a Material and assign the Texture to the new Material.
       - For 2D Projects, use the Sprite Editor.

Importing Textures

- HDR Textures
  - When importing from an EXR or HDR file containing **HDR** information, the Texture Importer automatically chooses the right HDR format for the output Texture. This format changes automatically depending on which platform you are building for.
- Texture dimension sizes
  - Texture dimension sizes should be **powers of two** on each side (that is, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, and so on).
  - It is possible to use NPOT (non-power of two) Texture sizes with Unity. However, NPOT Texture sizes generally take slightly more memory and might be slower for the GPU to sample, so it's better for performance to use power of two sizes whenever you can.

Alpha Textures

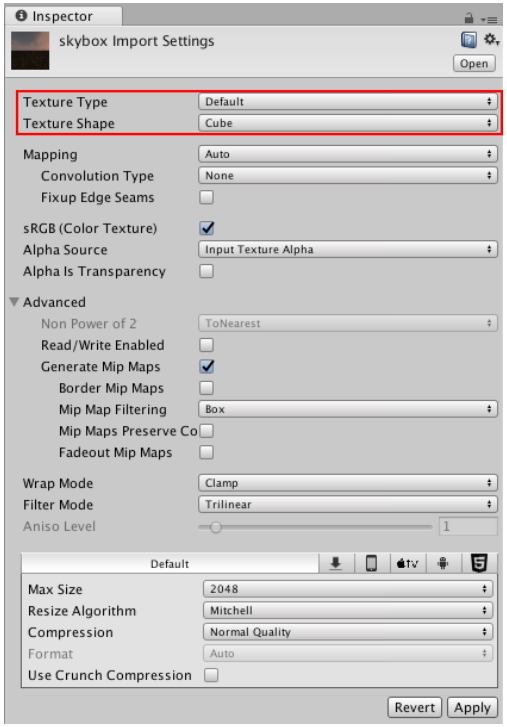
- Alpha maps
  - An **alpha map** is a Texture that contains only alpha information. You can use an alpha map to apply varying levels of **transparency** to a Material.

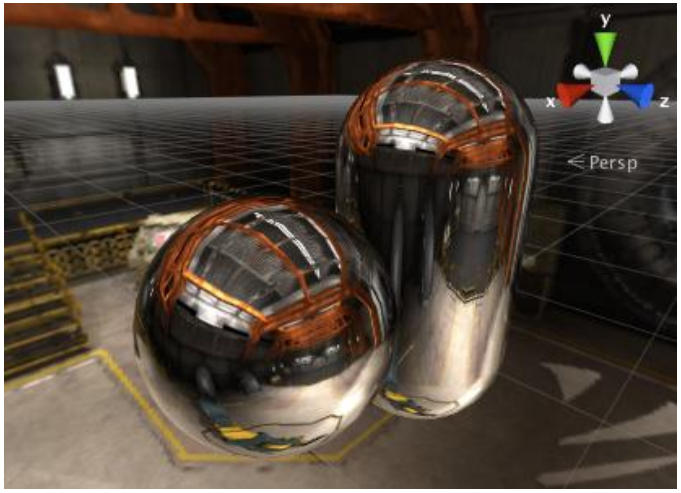


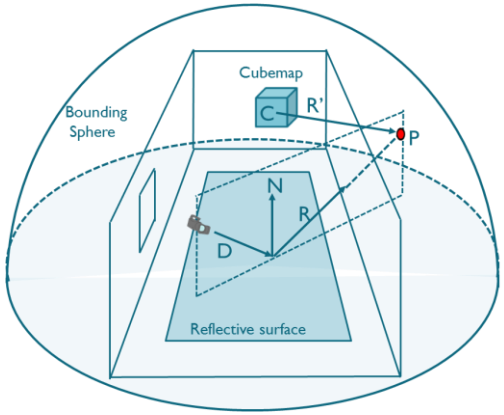
https://docs.unity3d.com/462/Documentation/Manual/HOWTO-alphamaps.html

Cubemap

- Reflections (cubemaps)
  - To use a Texture for reflection maps (for example, in **Reflection Probes** or a **cubemapped Skybox**), set the Texture Shape to Cube.







https://docs.unity3d.com/Manual/class-Cubemap.html
https://community.arm.com/arm-community-blogs/b/graphics-gaming-and-vr-blog/posts/reflections-based-on-local-cubemaps-in-unity