Day 7

Taraash

Contents

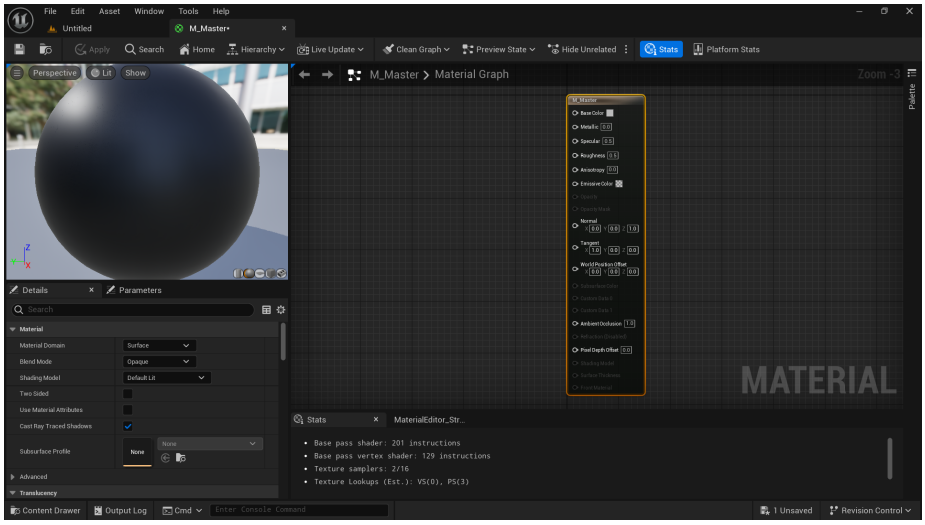
Introduction

We just talked about materials in day 7, we didn't do too much, so let's quickly get started.

Materials

I talked about materials in my notes for day 2 (which you can find here). I'll continue from there. Let's create a new material in Unreal Engine by right-clicking in the content browser and selecting `Material`.

Our goal is to make a master material, which we can use to create different material *instances* which we'll drag and drop onto our meshes.



Let me first break down the material editor.

1. The big window in the first quadrant is the graph editor, this is where we will use nodes and logic to determine how our mesh will look. (Find the `Palette` option on the far right to see all the nodes available to us.)
2. The second quadrant is the preview for our material. Take a look at the bottom left to change this preview mesh (from the default sphere to say, a cube, or a plane.)
3. In the third quadrant we'll find the details panel. If we click a node it'll show us details for that, and if nothing is clicked it'll show us the details for the material itself.

(You could change the background (or the environemnt the preview mesh is currently in) by going to `Window` > `Preview Scene Settings` and expanding the `Environment` section. You can also change the lighting settings here.)

Color spaces

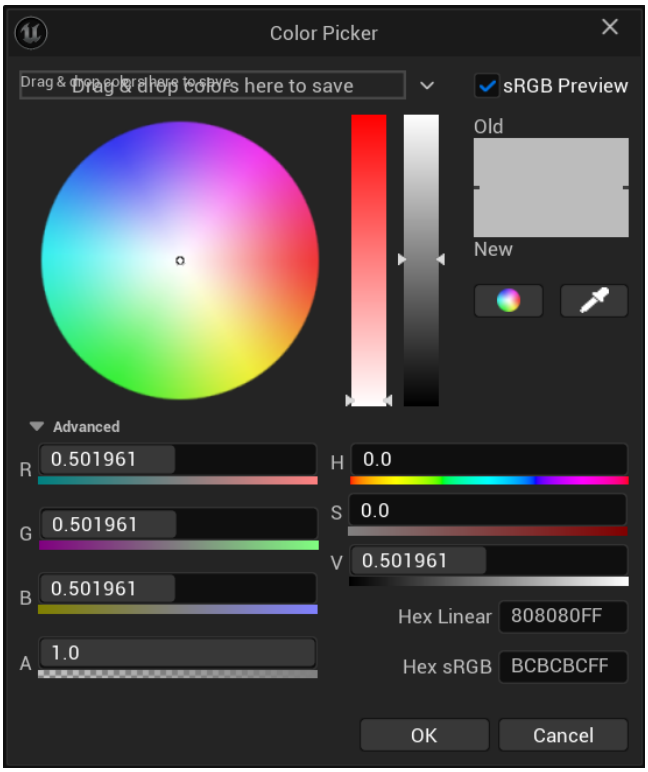


Figure 1: Unreal's color picker

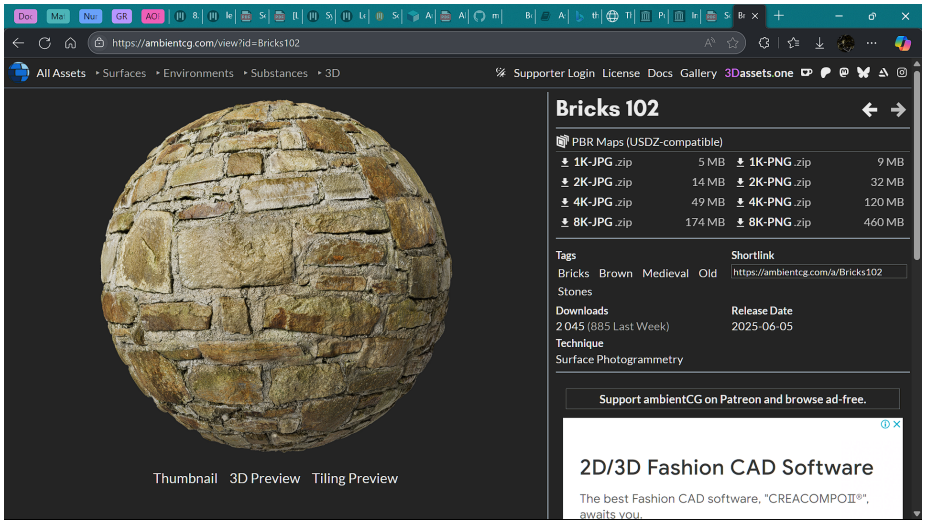
Let's talk a little bit about color spaces. You'll see two different columns in the above screenshot, one for `RGBA` and one for `HSV`.

RGBA R is the red channel, G is the green channel, B is the blue channel and A is the alpha channel (which determines how transparent the color is). Each of these channels can have a value between 0 and 1 (which are then mapped to 0 to 255).

HSV H is the hue, S is the saturation and V is the value (brightness). The hue can be between 0 and 360 degrees, while saturation and value can be between 0 and 1. Saturation can be effectively though of the radius (with white as the center, in the color wheel) and changing the hue will change the angle of the color in the color wheel.

Creating a material

To get a realistic material, we need to use a texture. I will download one from AmbientCG.



I'll get the 4k JPG version. (4k is usually the best option, but you can choose a lower resolution if you want to save on performance.)

Let's see what we got

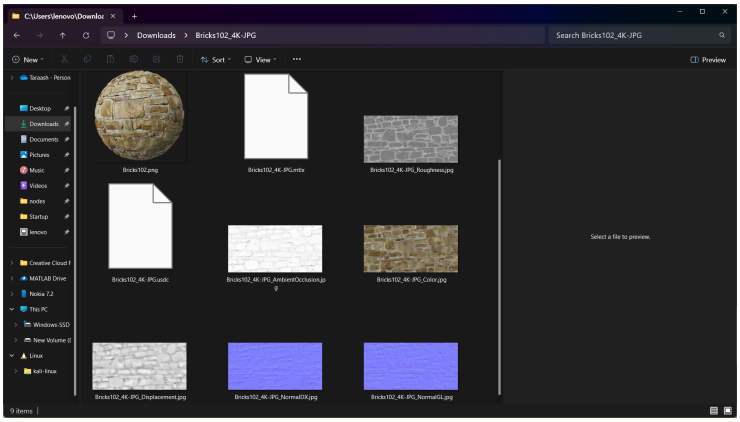


Figure 2: Extract and save the textures somewhere

1. The `Brick102` is just a beauty render, what we see in the preview.
2. The `.mtlx` and `.usdc` files are actual materials. (The first is MaterialX, usually seen with `.obj`'s and the second is USD (Universal Scene Description), which is a file format used by Pixar.) We'll be creating our own material, so we won't be using these.
3. The rest are our actual textures, in Unreal Engine, I'll create a new `textures` folder and import these there. (Note that we have two normal maps, DX is DirectX (Microsoft's graphics API) and GL is OpenGL (Khronos Group's graphics API). It doesn't matter which one you use, I'll be using the DX one.)

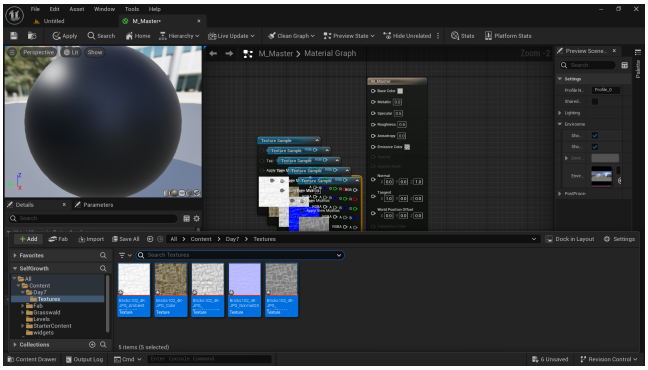
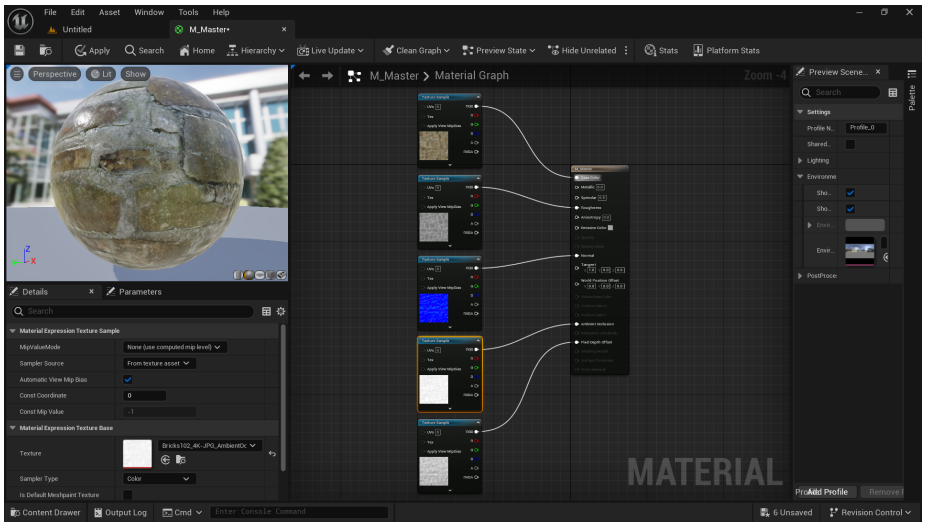


Figure 3: Import and drag and drop them into the material editor.

(Use `Control+Spacebar` to get the content drawer and naviagte to where you imported the textures.)

Click on the new nodes that we created by dragging the textures and plug them into their corresponding inputs in the big material node. (If you're unclear what that texture is, you can click on the node and in the details panel it'll show you the name of the texture.)



(To align these nodes I selected them and right-click > align nodes > align left and distribute vertically.) You'll also note that we don't really have a slot for the displacement texture, we're not really advised to use that in Unreal Engine (since that requires our mesh to be dense enough for the displacement to look good—bogging down our computer). I just plugged it into the `Pixel Depth Offset` input in the screenshot, but I'd recommend not using this map.

Our texture already looks pretty good, but I'd like to have some controls over this, in particular, changing the exposure (or strength), contrast and min/max values of the input. Before we talk about this, I wanted to quickly mention some very handy keyboard shortcuts in the material editor. (Pressing x while right-clicking anywhere will create a new node there)

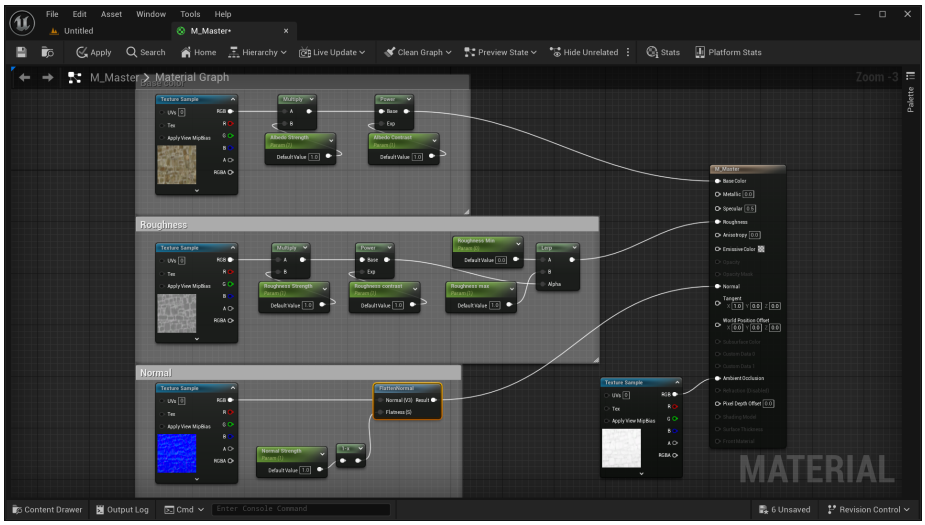
Keyboard shortcuts

Hold down on the key and right-click to create its corresponding node.

1. S for scalar parameter
2. 3 for three vector (1 for a float, *which is not a parameter*)

3. M to multiply
4. E to power
5. O to invert
6. L for lerp
7. U for texture coordinate

A "parameter" is exactly similar to variables in blueprints, these are the value that will be exposed so you could easily change these (without having to open up the material).



Take a look at this.

1. For the base color, I added a simple multiply and exponent node. Second input to both of which I connected to a scalar parameter (whose default value should be 1). In the exponent node, with our exponent less than 1, the difference between the high and low values of your base (base color, here) is decreased and vice-versa for when the exponent is greater than 1.

2. Let's talk about the roughness map. This is a grayscale image, and hence it's R, G and B channel all give exactly the same information. Under the hood, when you plugin in three-vector (which RGB is) into an input which expects a scalar (or a 1-vector), it automatically averages the values (if we plug in $\vec{V}$, it's converted to $\frac{V^0+V^1+V^2}{3}$ where I'm using the index notation). Doing a tiny bit of math will show you that for grayscale images, RGB, R, G, and B channels all give the same information.
   I'll talk about the `Lerp` node in the next section.

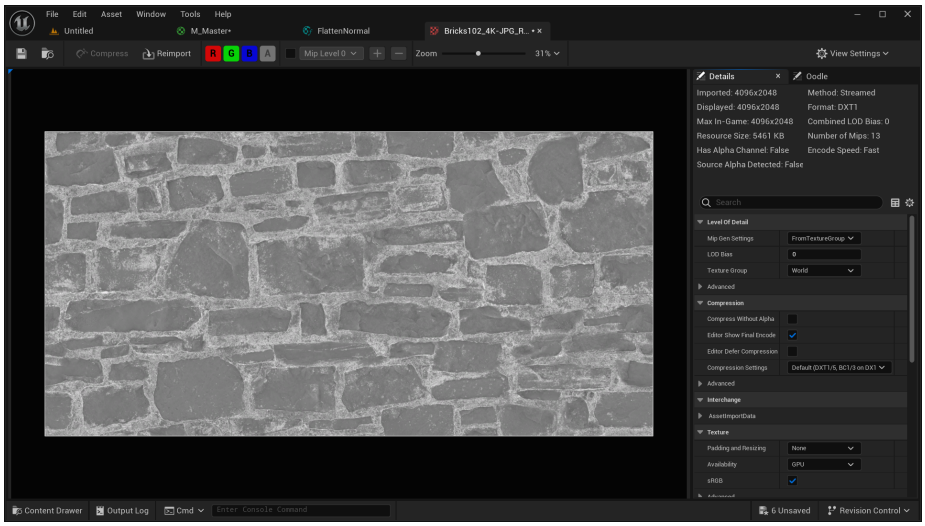


Figure 4: Opening up the texture by double clicking, try pressing on the R, G and B channels

3. For the normal node, I used a `FlattenNormal` node with an invert node since we'd like it so when we increase the parameter's value, the strength of the normal increases (and not decreases, which would've happened if we didn't use the invert node). Fun fact the `FlattenNormal` is a `Material Function` which you can open by double-clicking. You'll find it too uses a `lerp` node, hopefully, after the following disucssion of the lerp node you'll understand what this does.

Another interesting observation for those who are... interested, these texture maps 4096×2048 and not a square (2048×2048 or 4096×4096). Unreal says the textures should always be a power of two (which we have here), but having them as squares is not necessary! As long as your dimensions are a power of 2, you're good to go.

The lerp node

It follows the following simple formula

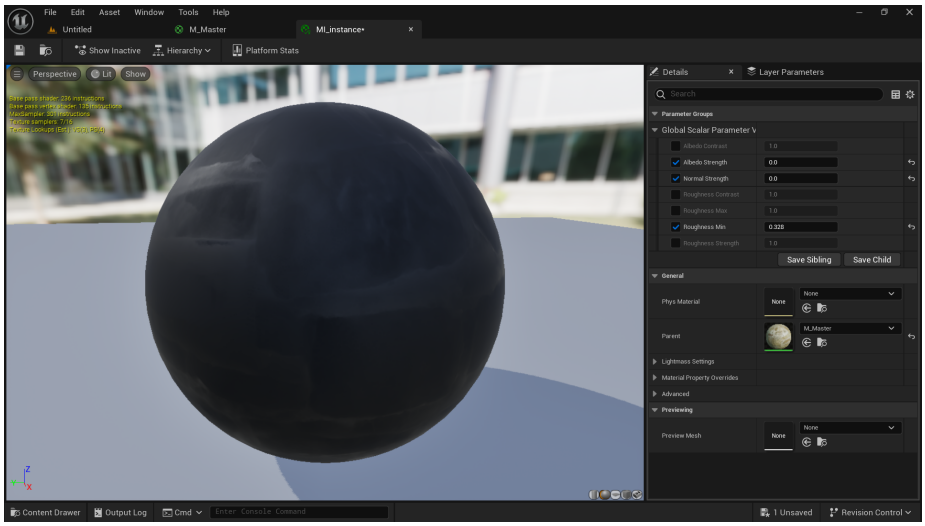
$$\text{Output} = A \cdot (1 - \text{alpha}) + B \cdot \text{alpha}$$

(this *alpha* parameter doesn't need to be between 0 and 1, it can go beyond and the formula still holds.)
If the input are textures, this operation is done pixel-wise, and that's very powerful! Lerp is short for linear interpolation, which the above formula also shows.

To put is simply, values closer to 0 will now be closer to A and values closer to 1 will now be closer to B.

Material Instances

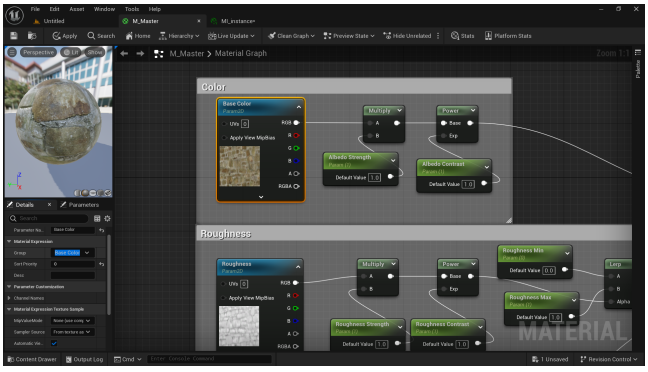
These are the actual 'materials' that you'll drag onto your mesh. It is not recommended you directly drag on a `Material` onto a mesh as that's ineffecient. You should always right-click on a `Material` and click the `Create material instance` option and rename it. This allows you to change the *parameters* we set in the material graph before without having to open up the node graph. Let's take a look:



On the left we'll find all the parameters we made before, I've set the albedo (color) and normal strength to 0 so we can just see the roughness map and how changing those min/max values affects the result. (Increasing the roughness min will make only the shinier parts of our mesh rough.)

I would like to organize these parameters I see, and would also like the option to switch the texture maps I used. To convert any (viable) node to parameter all you need to do is right-click and hit `Convert to parameter`. I'll do this for our base color, roughness, normal and ambient occlusions maps.

Organization



Select a parameter, and in the details panel find the `Group` setting, double click and add a relevant name. This creates a group which we can access from the drop-down menu directly on the right of the selection for other parameters.

The `Sort priority` determines the order in which parameters in a group are displayed, the lower, the closer to beginning it is (if 0, it'll be at the top)

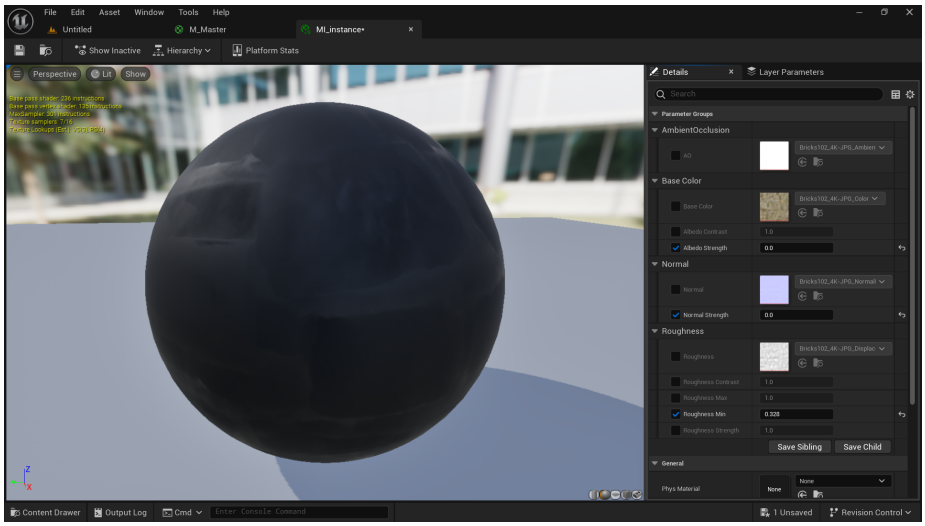
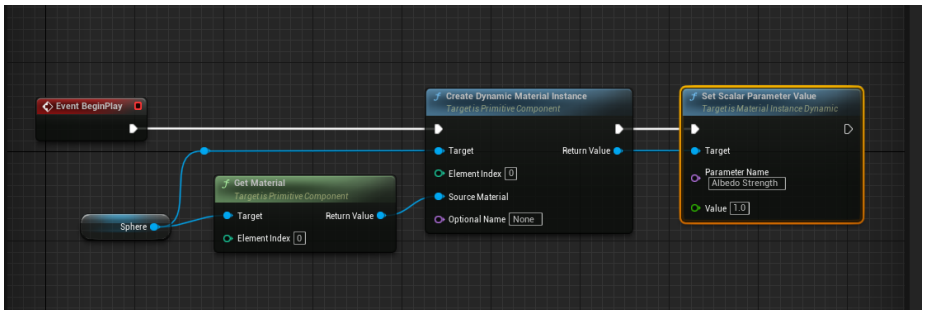


Figure 5: Organized the Material Instance for ease of use

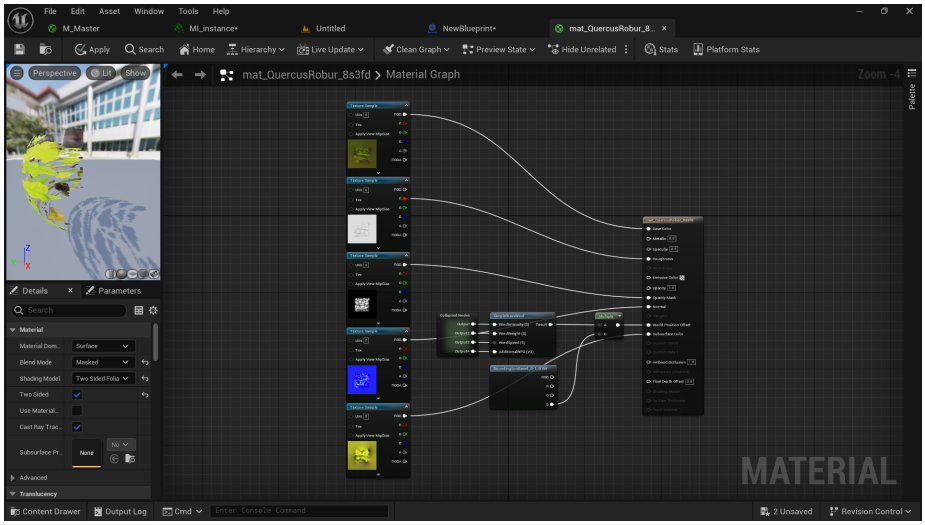
If you'd also like to sort the groups themselves, you'll have to rename them such ('Base color' → '1. Base color', 'Normal' → '2. Normal').

Changing a parameter with Blueprints



Take a look at what this does, on BeginPlay we're getting the material assigned to `Sphere` (which I plugged in to be the material instance we created) and coverting it to a `Dynamic Material Instance`. We then wish to change a parameter, which we can by using the `Set scalar/vector/texture parameter` node. (You can right-click on a parameter in the material instance editor and copy its display name to be certain you've got the right thing.)

Wind Animation



I wont't talk about this much here, since it really isn't needed. Find the assets from Grasswald and plug in the maps you get. A new one will be the Translucency maps, that goes in the Subsurface input (this determines how the light penetrates/changes color when it goes inside the surface).

(In the material details, check `two sided`, set the `shading model` to `two sided foliage` and the `blend mode` to `masked`.)

World Position Offset As the name suggest, this allows you to offset the mesh vertices from the material.

The `SimpleGrassWind` node does exactly what it says, determines each vertex offset to try and simulate simple wind. I converted all of its inputs to parameters, selected those 4 new nodes and collapsed them to a single one (right-click and find the option that does this) just to keep things tidy.

Plugging this in directly would make the root of the grass mesh also float, which we don't want. To fix this we get the `BoundingBoxUV` node which gives us the coordinates for the meshes' bounding box. We're multiplying the WPO (which is essentially a set of 3-vectors) with the Blue channel of these coordinates. We're doing so since Blue corresponds to the Z-axis and this value will be 0 at the bottom (the root, which we don't want to move, no offset) and 1, which'll be the top.

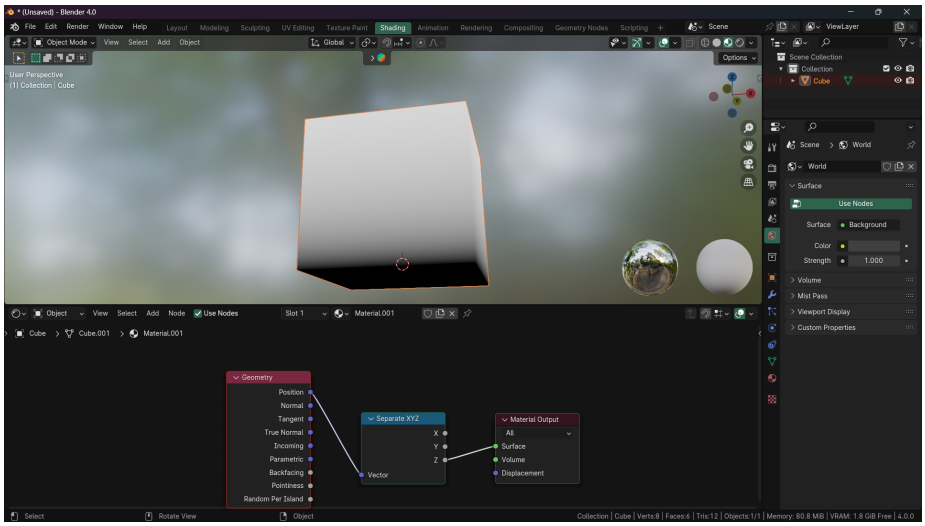


Figure 6: The gradient looks like this

I hope that makes it clear, multiplying this gradient with our world position *offset* would return 0 at the root, effectively fixing it in place. The last thing I'll mention are LODs.

Level of Details (LOD)

In the grass asset from Grasswald, you'd see multiple `.fbx` files, many ending with lod0, lod1, lod2...etc. Whenever your character is close to an object, you'd like it to be at its highest quality, but when we move away, we don't need all that detail, it'll just slow things down. To fix this, we switch the object to a lower level of detail, and we can do this multiple timings, depending on the asset we have and the distance of the character. When importing this grass asset, you'll find a pop-up menu (in Unreal Engine), find and check the `Import (mesh) LODs` option (it should be checked by default) to be able to use this optimzation trick in your game. Unreal will automatically switch to lower LOD if we are far away.

Finish

And we're done! I hope materials are now somewhat clearer. In the end I talked about a specialised example of foliage, you can skip this right now if it doesn't make sense. I also very quickly mentioned the `UV coordinates` (the `TextCoord` node in Unreal, which you can place with `U+right-click`), but I couldn't cover them here, we might talk about these later, I'll refer you to Pwnisher's video on material which was a source of reference for this class. With that being said, if

you've got any doubts (I know assignment 2 is leaving many people stumped!), please let me know.