Unreal Engine 3 Series
The Material Editor

written by David Lally | davidmlally@gmail.com

This document will cover the basic workflow of importing a model into the Unreal 3 Editor from Autodesk's Maya. In this overview we will be using the Unreal 3 Editor, Autodesk Maya (2008), Feeling Software's Collada Exporter for Maya, Crazy Bump, and Photoshop. See below for download links to the plug-ins or applications that may not be available to you.

Feeling Software's Collada:
http://www.feelingsoftware.com

Crazy Bump:
http://www.crazybump.com/

The following overview will be structured as a tutorial for taking a provided model of an apple from Maya, and setting up its materials for real-time usage inside of the Unreal Editor's Material Editor using their node based system. For starters, it would be a good idea to find some photographic reference for the look we'll be attempting to achieve. Below is a sample image of a well-lit apple, demonstrating a range of different effects we will attempt to simulate (Fig 1.0).

Fig 1.0 – A photo of an actual apple that will be used as reference.

When developing any type of shader, whether it's offline or real-time rendered, it is a good idea to identify the physical properties you'll be trying to achieve. Take a careful look at how light effects your object directly, as well as indirectly. Note the subtle color differences as well as the sharp details most evidently seen in the highlights of an object. With these ideas in mind, let's take a look at the provided photo-real texture that we will be using as a base throughout the tutorial (Fig 1.1)

Fig 1.1 – The base diffuse texture provided with the tutorial, taken from a photograph. The speckles have been removed digitally.

As you can see, the speckles on the apple have been digitally removed because we will attempt to layer them on separately once we're inside of unreal. This will give us an extra level of customization once we're in the engine as we will then be able to modify and customize the base color of the mesh independently of the speckles and/or bruises. I have provided a black and white image named "speckles.tga" in your tutorial folder.

Now, before we export our mesh out of Maya, lets run our base photo through Crazy Bump in order to generate some additional maps that we'll be wanting to work with such as the normal, specular, and ambient occlusion maps. Load Crazy Bump and choose to load a photograph from your files. Load in the provided "apple_diff.tga" and you will be brought to a screen that looks like (Fig 1.2):

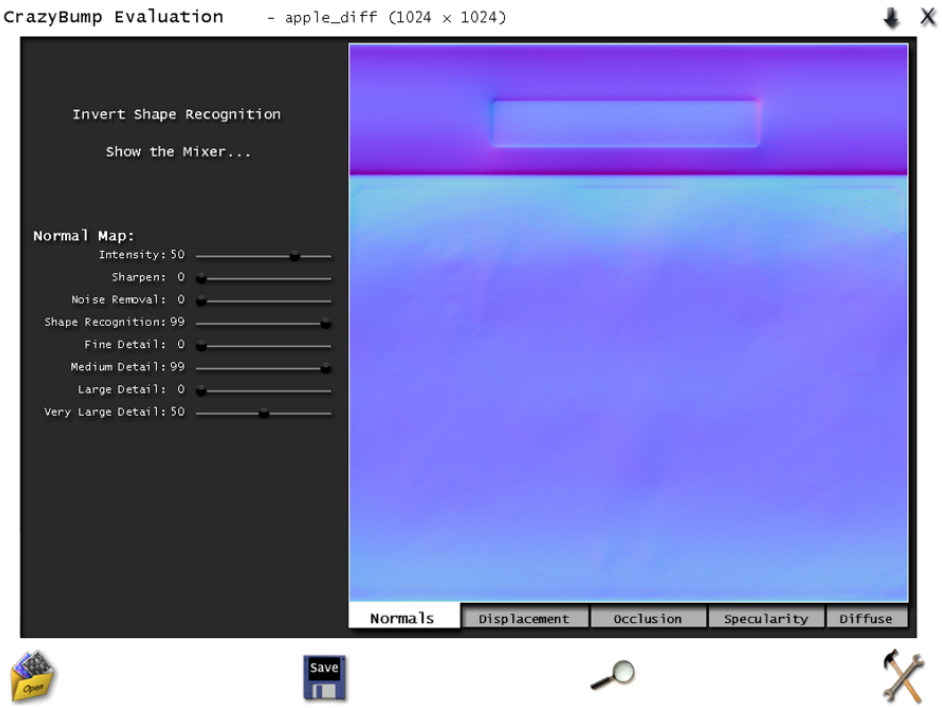

Fig 1.2 – Crazy Bump Screen (Notice the tabs for the different maps you can easily edit inside of Crazy Bump)

As I mentioned, we're interested in exporting a normal, occlusion, and specularity map out of Crazy Bump (we have our diffuse), so take some time to judge the amount of detail we'll be needing based on our original reference photograph. I recommend a high intensity, a slight sharpen, an increase in the fine detail for our small bumps, and increase on the large detail. This should give us the amount of subtle detail we need for the apple's surface. Once you're happy with your maps, click the save (disk icon) button and export all of your maps to a familiar project folder. If you didn't export as .TGA files, be sure to convert your images to .TGA for implementation to Unreal. You can do this by re-saving the images inside of Photoshop. You should now have the four image files seen below (Fig 1.3):

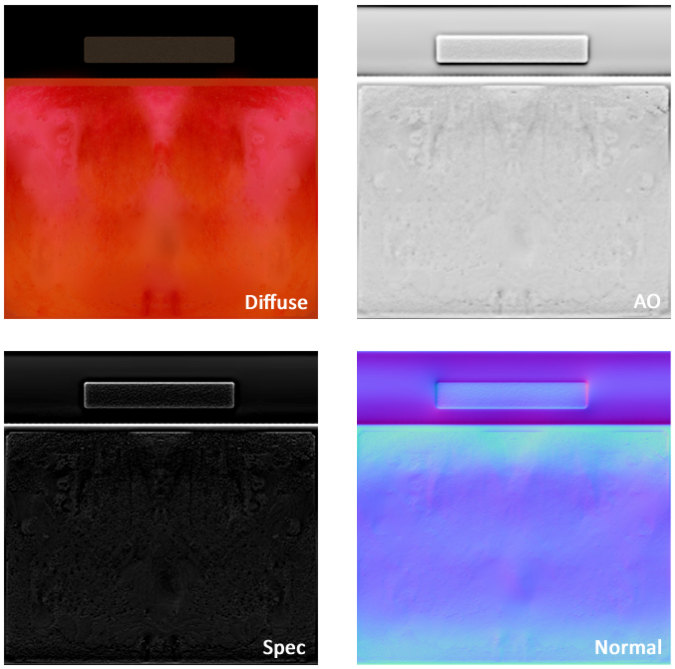

Fig 1.3 – Maps exported from Crazy Bump (Diffuse, Ambient Occlusion, Specularity, and Normal)

With all of our maps ready, lets export our mesh out of Maya using the COLLADA plug-in. Load up Maya and import the provided "apple.obj" file by going to File > Import (Fig 1.4). Notice that .obj files maintain their UV Layout which was made prior to the development of our diffuse texture (Fig 1.5). Because of this, be sure not to edit the UVs unless you're prepared to run the updated diffuse map through Crazy Bump once more.

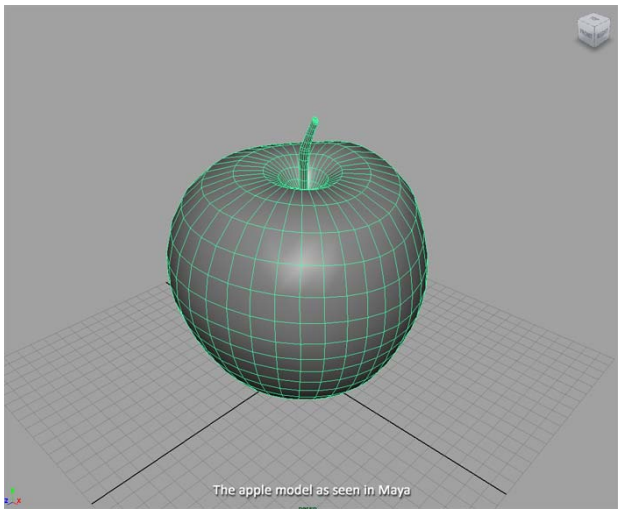

Fig 1.4 – The apple model as seen in Maya

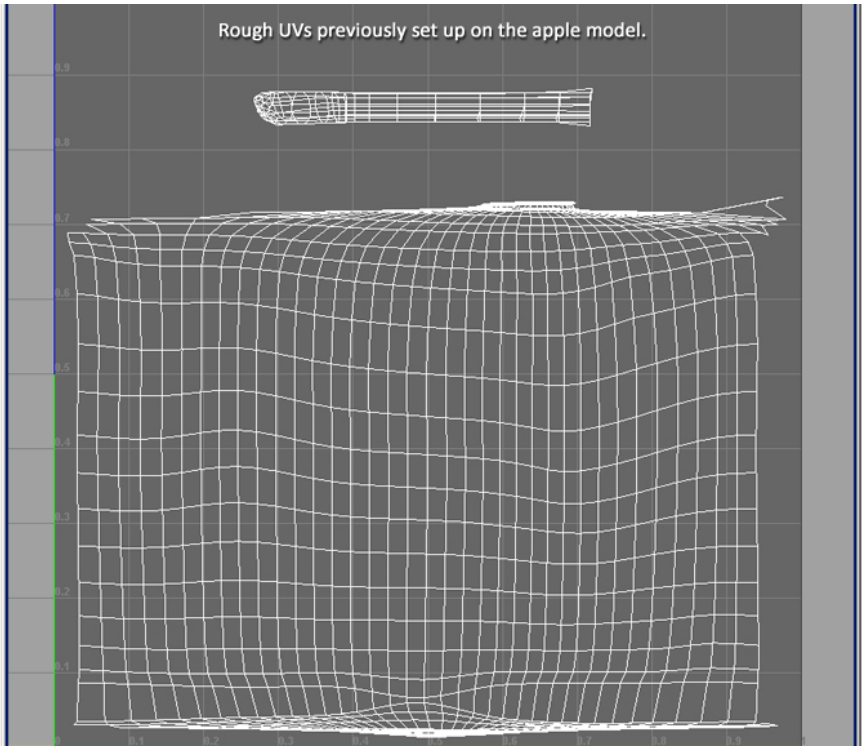

Fig 1.5 – The apple model has some rough UVs that are laid out to work with our various texture maps.

By now you should have downloaded COLLADA for Maya from Feeling Software's website. After doing so, go to Window

> Settings and Preferences > Plug-in Manager and select the check box for “Loaded” next to COLLADA.mll. If you don’t see the COLLADA.mll file, you can browse to it using the button at the bottom of the window. With this plug-in activated, we can now use COLLADA to export a .DAE file that will work perfectly inside of the Unreal Engine. We can do this by selecting our apple’s mesh, going to File > Export Selection and selecting “COLLADA Exporter (.dae, .xml)” as our file type. Be sure to save the .DAE file in a recognizable area so you can locate it once we’re inside of Unreal!

Now that our apple model and all of our textures are ready to go, we can move right into the Unreal 3 Editor and begin setting up our real-time materials for the apple for use inside the game engine. Load a new file in the Unreal 3 Editor.

At this point you can load a pre-existing level into the engine or create your own level. Either way, we just need a testing ground for viewing our apple model/shader. I recommend creating your own level as many of the preset levels will have post effects applied to the world which will alter the appearance of our end product.

Once you have a level selected, open the “Generic Browser” and set up a new asset folder that will contain our models, shaders, and textures throughout the tutorial. You do this by going to File > New in the Generic Browser and setting up the new folder as scene below (Fig 1.6):

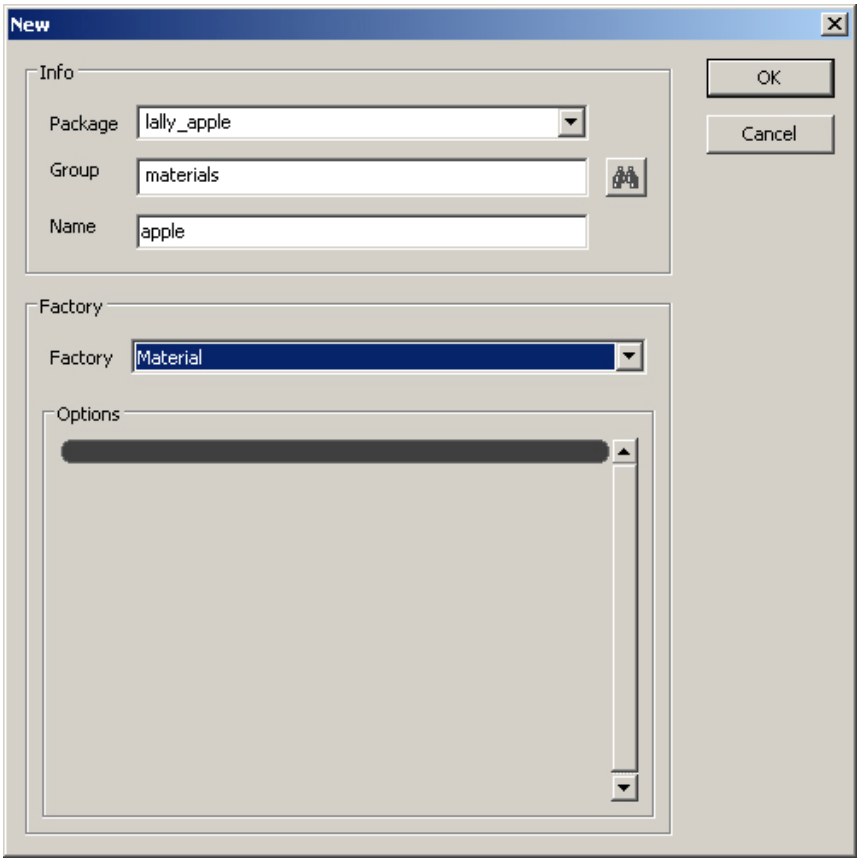

Fig 1.6 – The dialogue box for setting up a folder in the generic browser.

Once you have this new folder created, you’ll be able to go to File > Import and bring in any outside materials. Do this to import the .DAE file we exported out of Maya, as well as all of our .TGA texture files. Now that our assets have been imported, double click on our apple model (this is probably barely visible in the thumbnail for the time being). Double clicking on this model will open the Static Mesh Editor. While in the Static Mesh Editor go to Collision > Auto Convex Collision to set up some automated collision geometry for our apple. This will allow it to be imported into the

environment as an error-free static mesh (all static meshes require some collision). When you're done adding the collision, exit out. The collision will save automatically. Now, in your editor viewports, click an area/object where you would like to spawn your apple and right click. In the right click menu, go to Add Actor > All Templates > Static Mesh. If you have the apple model selected in the Generic Browser, the right click menu will already display the name of your model (Fig 1.7). Add the Static Mesh into the world.

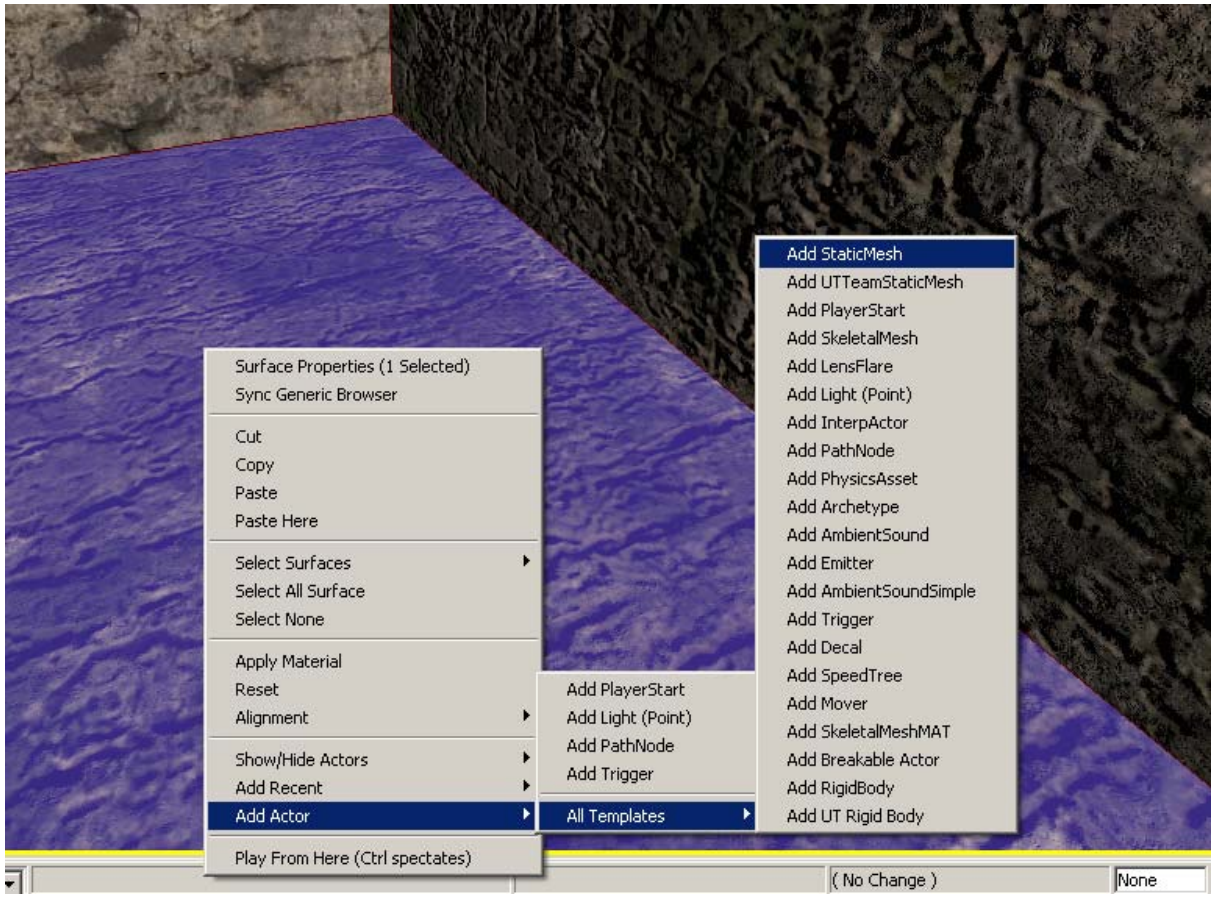

Fig 1.7 – The right click menu for adding a static mesh into the environment.

Our apple now appears in the game world. If you apple didn't come into the Generic Browser with a material, right click an empty area in your Generic Browser and create a new material. You can assign this material by clicking on the apple's static mesh in the viewport, right-clicking, and going to Materials > and assign the material that you have selected in the Generic Browser. You should see the generic grey checkerboard shader on your apple in the viewport if you didn't already. This will now be the shader that we customize in order to reach our desired apple effect. Before continuing, be sure that there is a light in your scene. If there isn't a light, hold the "L" key and left click somewhere in the environment to quickly generic a point light. Position the light so that the apple is well lit.

Double-click on our new shader in the Generic Browser in order to open the Material Editor, the area where we'll be doing the majority of our work. You will see our "PreviewMaterial_1" node that contains all of our major inputs for our shader including Diffuse, Specular, Normal, etc. –some of which should sound familiar as we exported maps from Crazy Bump with the same names! Instead of just plugging in our generated maps, we will be doing some creative methods of implementation in order to reach our desired apple shading effect.

From the Material Expressions list, locate the “Texture Sample” node and drag it onto the workspace. Double click that “Texture Sample” node while in the workspace and notice that there is a field in your properties tab for designating the file path for the texture. Using the magnifying glass icon, you can search for our diffuse texture in the Generic Browser window. When you find the diffuse texture, select it and return to the properities tab in the Material Editor. Now click the “green arrow” icon that will input your Generic Browser selection. We now have our diffuse texture! Drag the black tab from our Texture Sample node into the Diffuse tab on our PreviewMaterial_1 as seen in the image below (Fig 1.8):

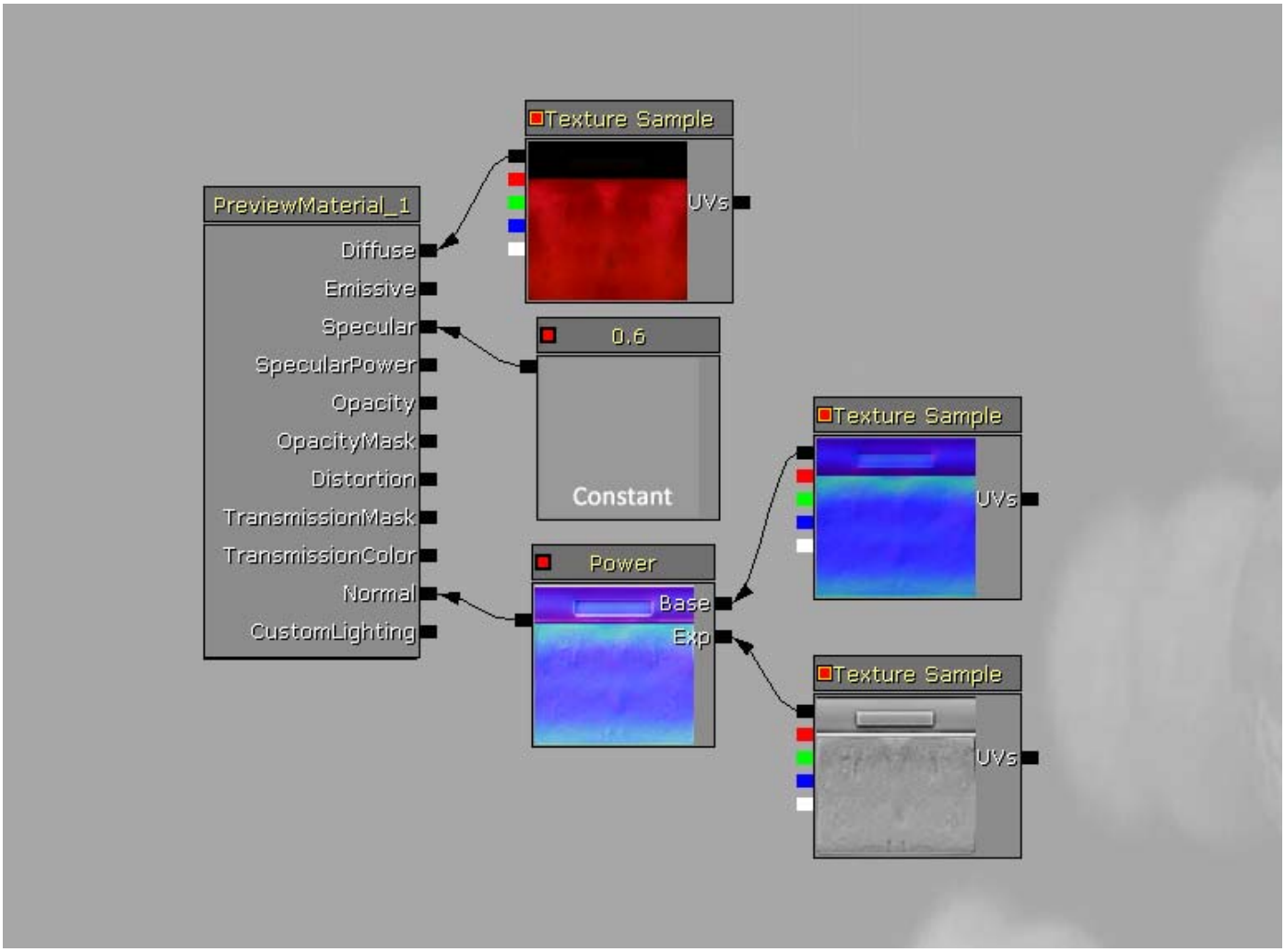

Fig 1.8 – A basic collection of nodes to apply diffuse, specularity, and normal mapping to our shader.

As seen in the image above, repeat the “Texture Sample” process by grabbing two more nodes and plugging in our normal map texture and our ambient occlusion texture. As seen in the above image, we’re also using a “Power” node and a “Constant” node in our workspace which you can drag from the Material Expressions list as well. The Power node will amplify the effect of our normal map according to the black and white values present on our ambient occlusion map. You can hook the normal, AO, and power node up exactly as seen in the above example. When you’re done, connect the updated Power node into the Normal tab of PreviewMaterial_1. Also note that we’re using a constant of 0.6 on our Specular tab in order to control the way light hits our apple’s surface. When your connections match those of Figure 1.8, you will be able to select the left most green check box icon (Fig 1.9) to update the shader in the viewport.

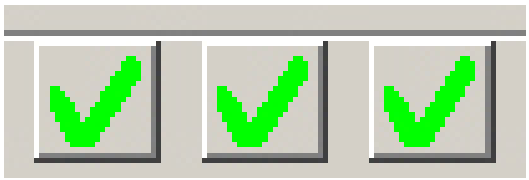

Fig 1.9 – Green Check Boxes (Assign material to the world, create a Fallback, and Regenerate Auto Fall Back)
Select the left most check mark (assign material to the world).

You should see the feedback of your shader working in the viewport, even without rebuilding the lighting within the scene, etc. Now lets take it one step further and add our Fresnel effect by taking a Fresnel from our Material Editor and driving it into the Emissive tab of our PreviewMaterial_1. The Fresnel will help us get that shine across the apple, though it will look awkward at first. You apple is likely to look something like this at this point (Fig 2.0):

Fig 2.0 – The apple shader with an erroneous Fresnel along the edges.

So with this set up, we can see that the apple appears to have a decent amount of detail with the beginning of a specular highlight that is close to the original reference photograph. Unfortunately, because of specular constant of "0.6" that we have plugged into the "Specular" channel, we're getting a very unnatural and seemingly smoky glow around the edge of the apple. We will fix this in a section of the next step.

In this image, you can see that there are quite a few nodes piped into the Emissive channel, namely a Fresnel effect which is actually the desired effect we're shooting for. The Fresnel effect will allow us to get that nice light wrapping around the edges of the apple and we can set a color bias so that the edges are no longer a grey tint (Fig 2.1):

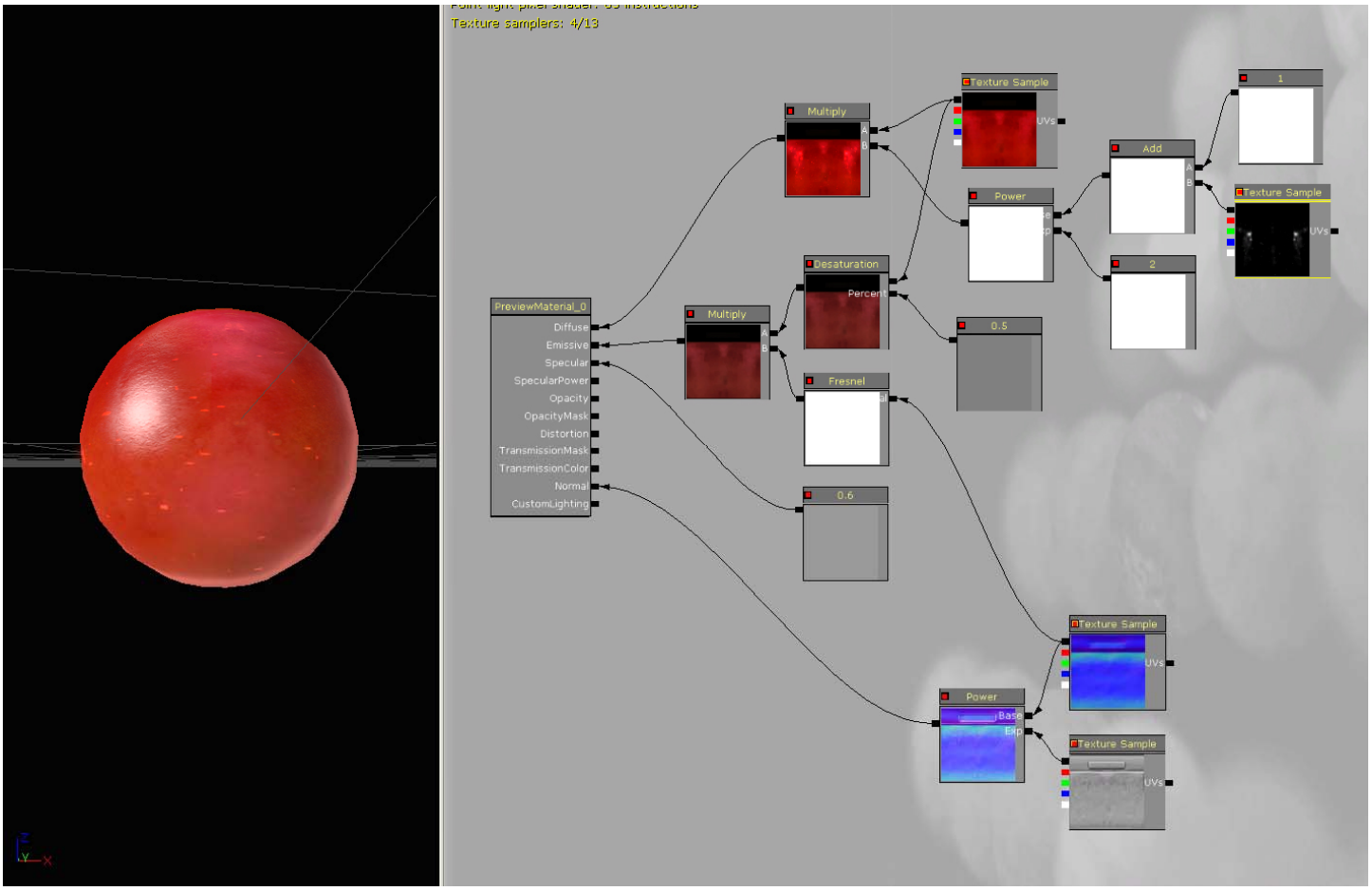

Fig 2.1 – The Emissive channel recieves the fresnel effect through a multiply node (multiplying the fresnel against a desaturated version of the original texture map).

With only that emissive channel change to the model, you should receive a look that is close to this (Fig 2.2):

Fig 2.2 – Our apple with the added fresnel effect in the emissive channel.

You'll now notice that there is something else going on in Figure 2.1. We're using a separate image of black and white speckles to simulate speckles on our apple to be layed on top of our original apple skin texture. So why are we doing this? Well, we could very well just keep all of this on the same texture to get a similar effect on our apple. However, by separating the layers, we now have much more control over the individual images that can be manipulated individually in real-time! That means we can edit the hue, saturation, contrast, or any other element of these images to get unique effects such as the apple slowly turning brown and rotting or the speckles beginning to glow unnaturally. Below is the result of adding the speckles normally as seen in figure 2.1 (Fig. 2.3):

Fig 2.3 – The apple with the speckles added as a separate layer.

Now, here is an example of a method of manipulating just the speckles in order to make them green as if the apple was beginning to unnaturally rot away. Notice that because we piped in a greyscale image for the speckles, we can multiply that same image by any 3-vector constant (simply meaning a constant with R, G, and B channels) and change our speckles to any hue we'd like. From here, you can edit the Power node and additional constants to reach the desired effect (Fig 2.4):

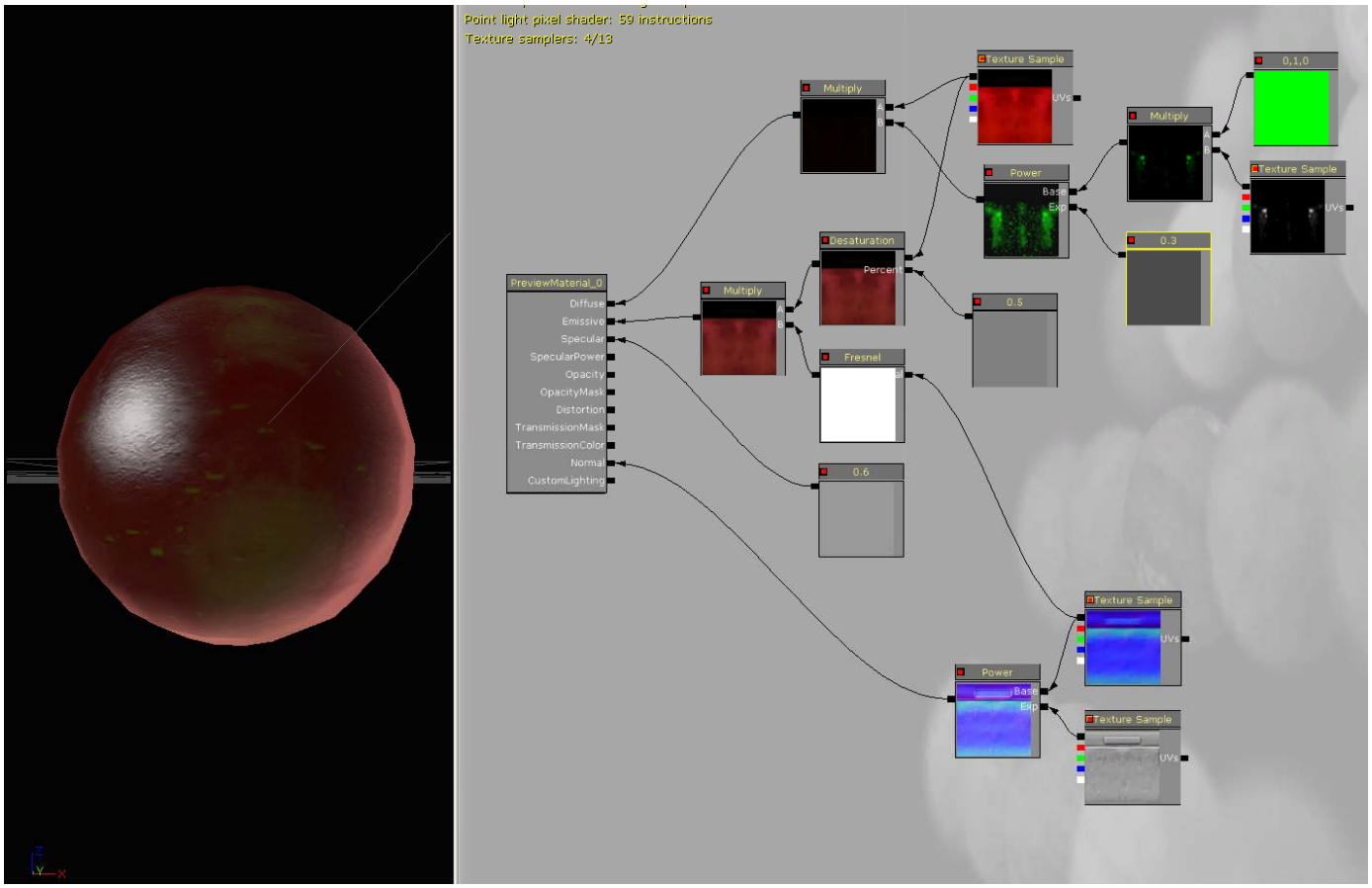

Fig 2.4 – The speckles are multipled by a 3-vector constant to change them to green.

Because the green speckles are a bit extreme, lets alter them to become something more realistic like a rotting tan color. By changing our three-vector constant to more of a dirty brown color and adding the two layers instead of multiplying them, we get a fairly believable look that can be animated if we so choose (Fig 2.5 and 2.6).

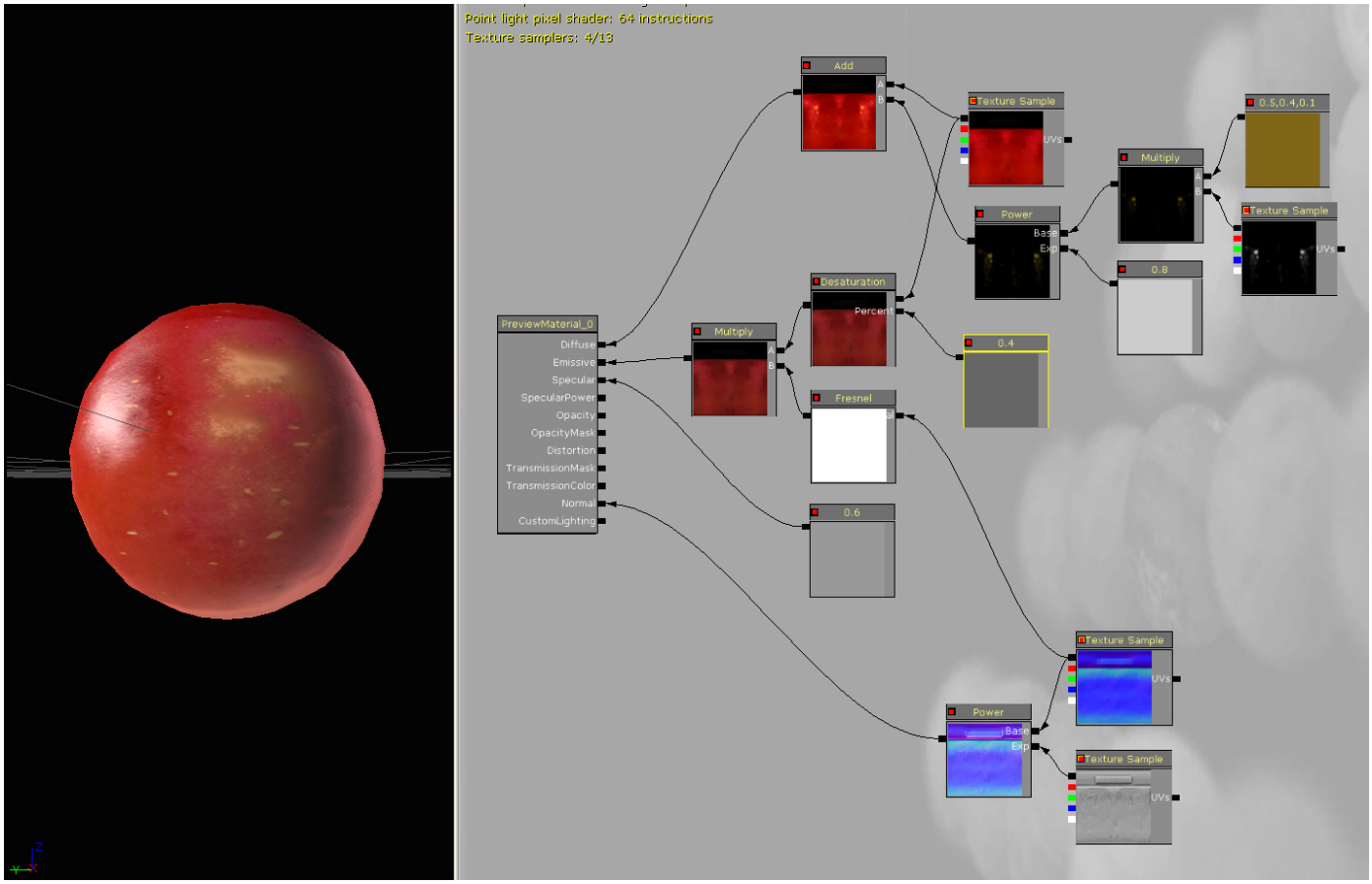

Fig 2.5 – Our speckles and bruised areas modified to be a tan color to simulate rotting. This effect could then be animated for a gradual transition over time.

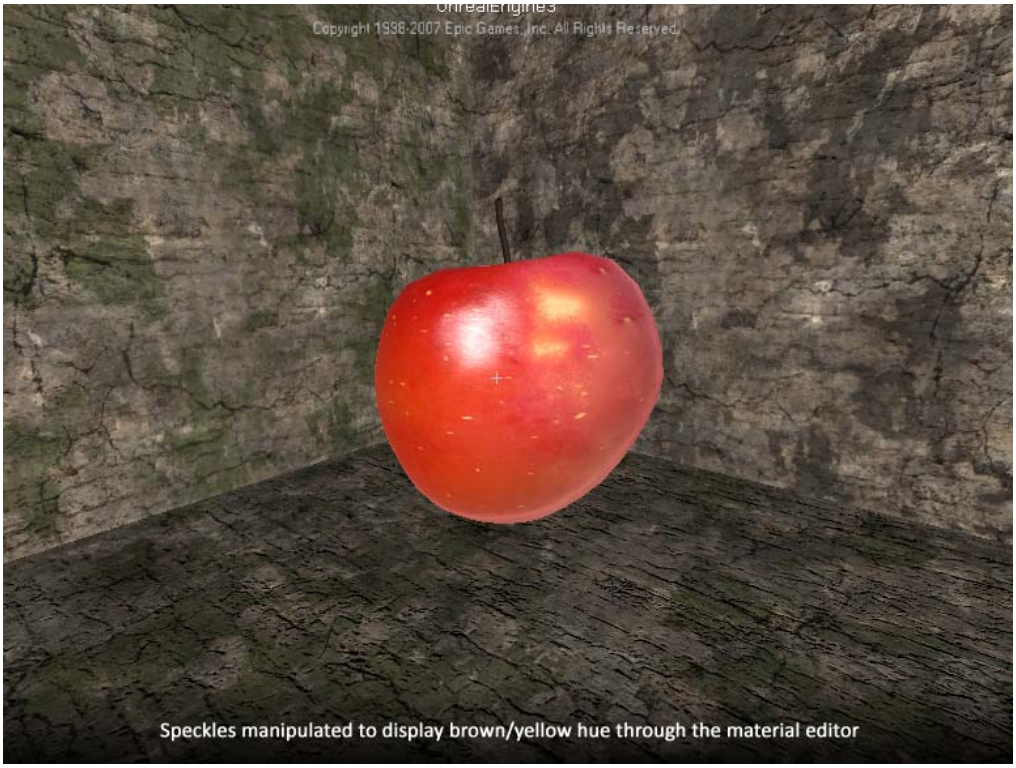

Fig 2.6 – The brown, rotten speckles in our test level during game play.

Now, obviously this effect isn't only for an apple and can be applied to any type of model for any type of look you're going for. You're likely to use different maps with separate layers for character set ups, and maybe even environments. Think space marine with glowing lights throughout his suit that need to strobe on and off. With this type of set up, our nodes can also be accessed through UnrealKismet, the node based scripting language, to trigger these events at specific instances in time or upon interaction. Here are some additional links to reference materials you may find useful for Unreal's Material Editor:

List of Material Expressions (Hourences):
http://book.hourences.com/tutorialsue3matexpressions.htm

Unreal Developer Network (Epic Games):
http://udn.epicgames.com/Three/MaterialsTutorial.html

Material Basics (Waylon):
http://waylon-art.com/LearningUnreal/UE3-11-MaterialBasics.htm

Real Time Rendering (Akenine-Moller, Haines, Hoffman):
http://www.realtimerendering.com/