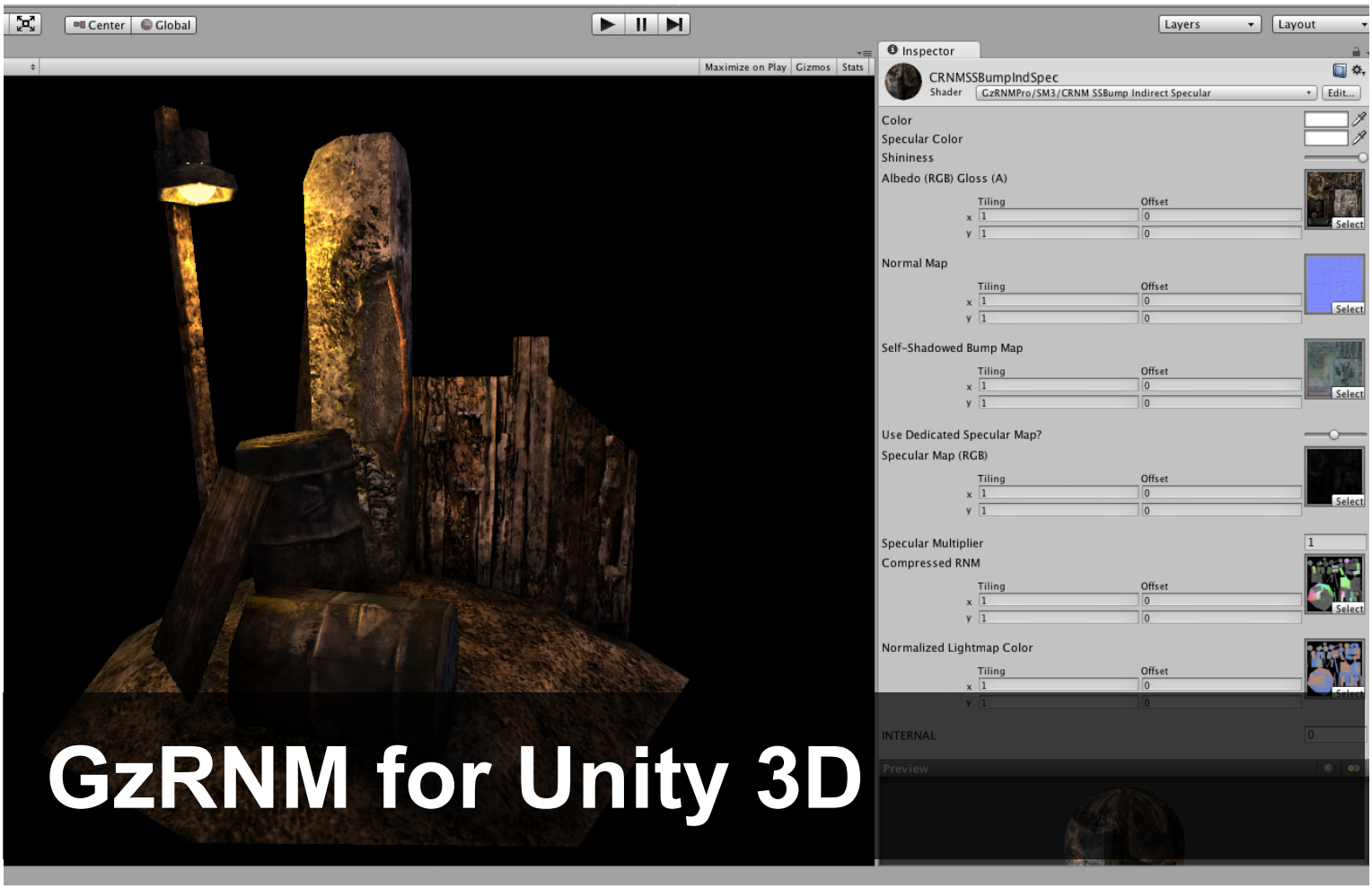

GzRNM by Jonathan “Geenz” Goodman - Documentation and tutorial asset by Wes McDermott

Table of Contents

GzRNM Documentation

What is Radiosity Normal Mapping?

Radiosity Normal Mapping (RNM) or Directional Lightmaps as it's sometimes referred to, is a shading technique that was developed by Valve and used in Half Life 2[1]. It is a process in which bump maps (normal maps) are combined with pre-computed radiosity lighting to achieve realistic surface illumination. Whereas Normal mapping is typically computed one light at a time, RNM effectively bump maps with respect to an arbitrary number of lights in one pass. GzRNM is a set of shaders and utilities that bring this same functionality to Unity 3D in a convenient and easy to use workflow.

In this document, we'll cover the usage of GzRNM, the process behind creating the RNMs in a 3D program, using the built in tools for creating CRNMs as well as converting a normal map to an SSBump map.

The GzRNM Package

Once you've downloaded the GzRNM package, you'll add it into your Unity project just as you would any package by using "Import Package." Once the package has been added to your project, you'll find several folders, which contain the shaders, GzRNM Utilities, RNM Basis Maps and a sample scene.

To apply one of the shaders to your project, you simply need to create a material or use an existing material and select one of the GzRNM shaders from the drop-down menu as shown in Figure 1.

[1] For more information on Valve's RNM process, visit:

http://www2.ati.com/developer/gdc/D3DTutorial10_Half-Life2_Shading.pdf and http://www.valvesoftware.com/publications/2007/SIGGRAPH2007_EfficientSelfShadowedRadiosityNormalMapping.pdf.

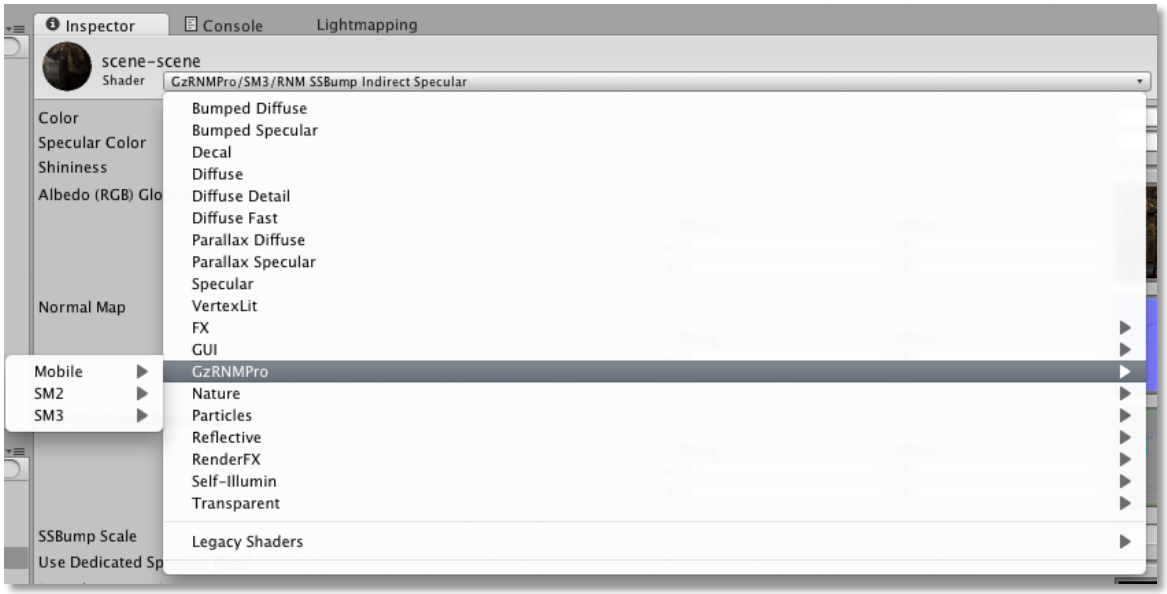

Figure 1 GzRNM shaders

The GzRNM shaders are divided into 3 categories, which indicate the graphics hardware they require. For instance, the SM2 set are for Shader Model 2.0 platforms, where as SM3 require Shader Model 3.0 support and Mobile are for mobile platform use. In the Shader Reference section of the documentation, you can see a complete list of each shader.

GzRNM gives you the ability to create a special type of compressed RNM format, which is referred to as CRNM. CRNM contains a composite of the 3 RNM images and store light intensity information in one map instead of 3 RNMs. It's important to stress that CRNM only store light intensity. If you'd like to use light color as well, you'll need to also supply a normalized lightmap color map to the shader. Although, this highly compressed color map can be left out, as it's not required for the CRNM shaders. GzRNM Pro also contains the GzCRNM SSBump shader, which is the preferable shader for mobile platforms such as iOS and Android.

Depending on which shader you decide to use, you will also need to supply either a set of 3 RNM component maps or a CRNM map, an SSBump map and or Normal map, as well as a Normalized Lightmap Color map and Specular map. Below you can see the different types of maps that are used by the shaders as shown in Figures 2, 3 and 4.

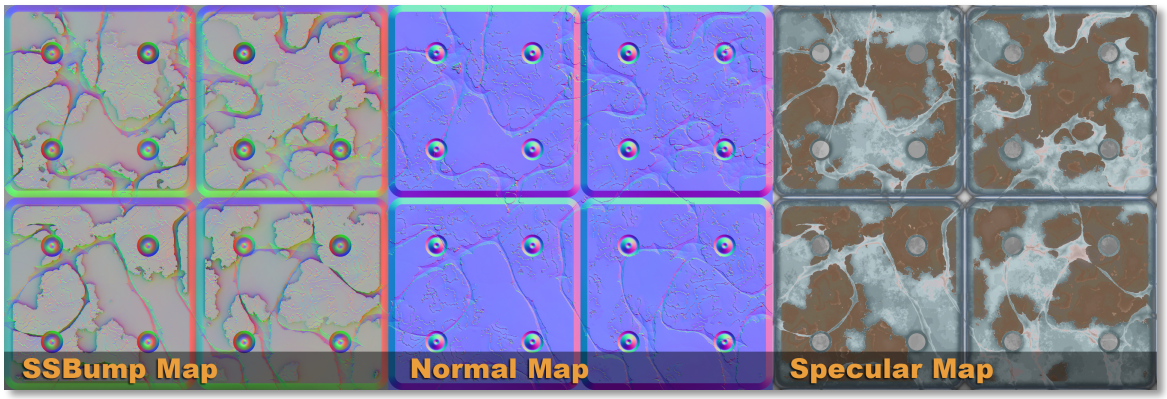

Figure 2 SSBump Map, Normal Map and Specular Map

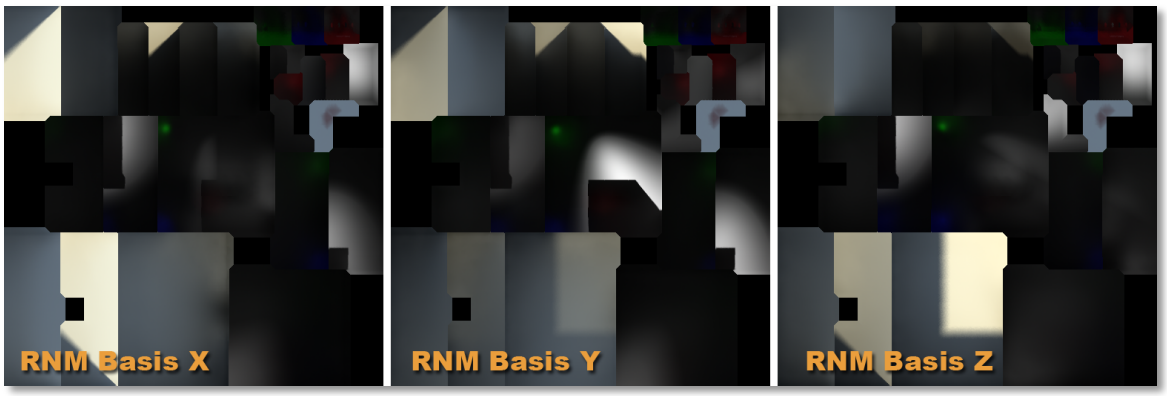

Figure 3 RNM Basis Maps

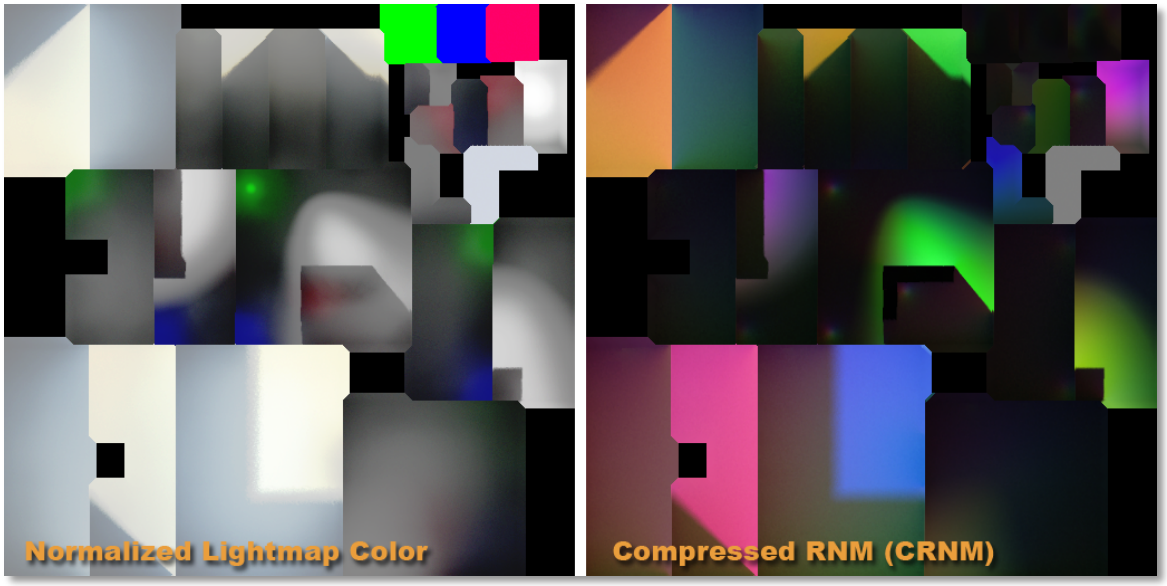

Figure 4 Normalized Lightmap Color and CRNM

Over the next sections, we'll take a look at what these specific maps are, how they are created and how the shaders use them.

Creating the RNM Component Maps

The RNM component maps are rendered in your 3D program. In RNM lightmaps, light values are computed for each of the three vectors in regards to a specific basis i.e. the bumpBasis denoted by Valve into three textures. Basically, each map represents what the lighting should look like from the surface normal of a polygon when it points in one of three bumpBasis [X, Y, Z] directions as shown Figure 5.

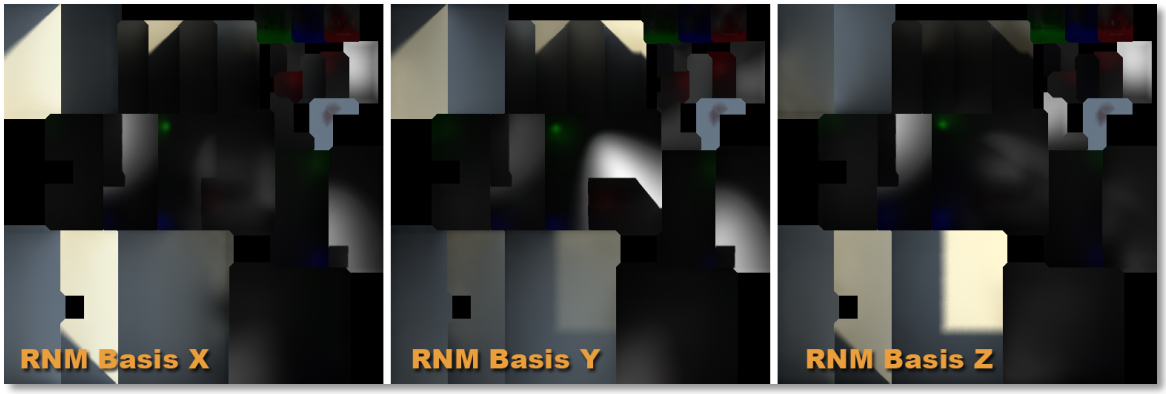

Figure 5 RNM Basis Maps

In order to transform your mesh's surface normals in tangent space to the bumpBasis, you'll use the RNM Basis maps that can be found in the Snow Cat Solutions folder in your Unity project as shown in Figure 6. Each of the three images corresponds to one of the bumpBasis directions through which RNMs should be generated.

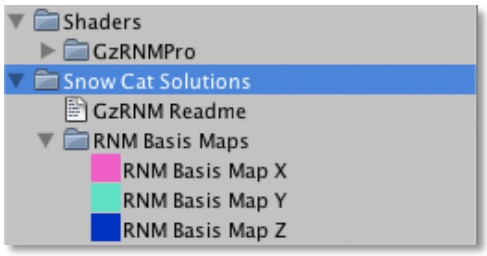

Figure 6 RNM Basis Maps are located in the Snow Cat Solutions folder.

Using the basis textures is simple enough; you simply supply the texture as a normal map using the primary UV of your mesh in your modeling application.

Working with the Correct UV Set

When assigning the RNM Basis textures, it's **VERY IMPORTANT** that you set the UV Map for each RNM Basis texture to your primary UV set and not your secondary lightmap UV set. If you use the wrong UV Map when assigning the RNM Maps, you will not have the correct basis baked into your lightmaps. When it comes time to baking the maps, you'll then bake the RNM maps from your lightmap UV set as you normally would bake a lightmap as shown in Figure 7.

Apply RNM Basis Maps = Primary UV set - Bake RNM Lightmap = Lightmap UV set

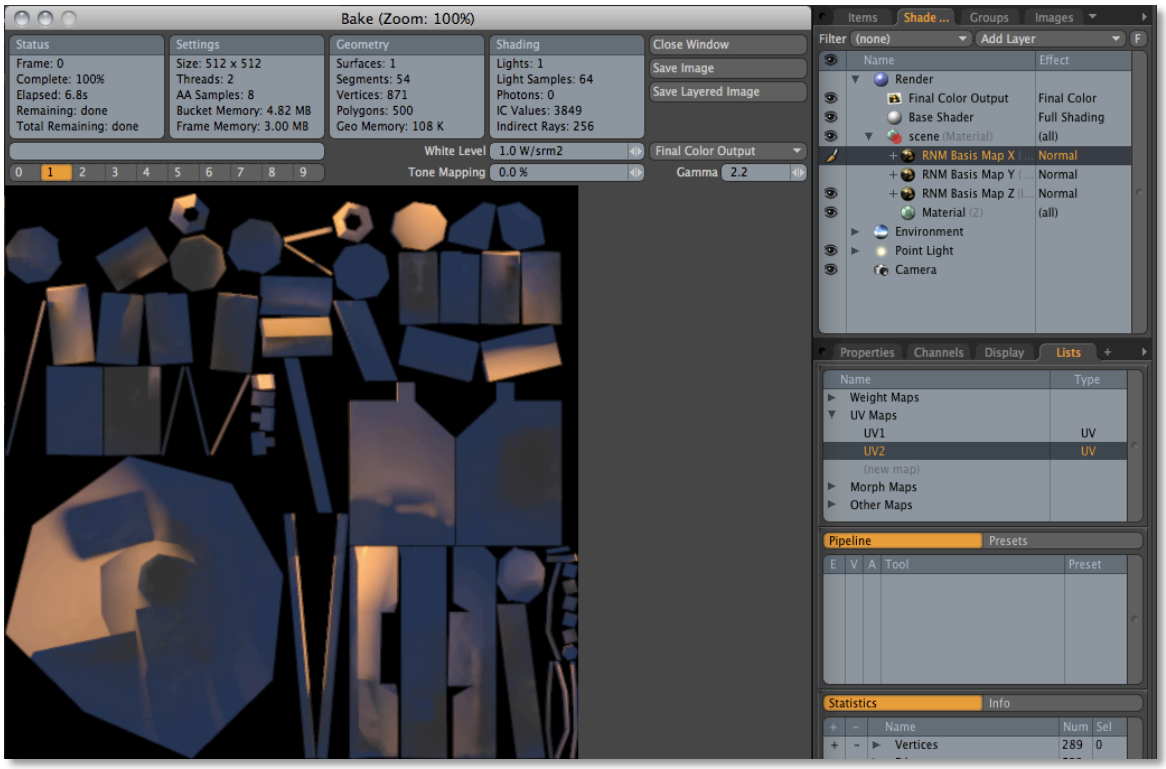

Figure 7 Assign RNM Basis to Primary UV set and bake RNM maps from lightmap UV set

Now, let's take a practical look at generating RNMs in a 3D application. Although this documentation uses Luxology modo and Autodesk Maya for the demonstration, this workflow can easily be adapted to any 3D application.

Rendering RNMs using Luxology modo

To render the RNM maps in modo, you'll need to properly setup the RNM basis maps. First, you need to import the RNM Basis Maps into the Clip Browser that are supplied with GzRNM and are found in the Snow Cat Solutions folder as mentioned above and shown in Figure 8.

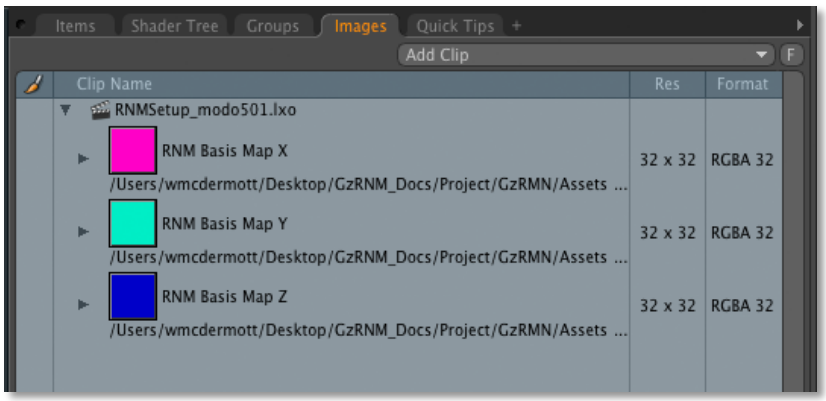

Figure 8 The RNM Basis textures are loading into the Clip Browser

Next, you'll add the RNM Basis Maps to the Shader Tree so that they are applied to the material of your object. You will then set each texture layer's Effect to Normal and click the Eye Icon to the left to the texture to disable it. Remember, you will need to set the appropriate UV Map for each RNM Basis texture as shown in Figure 9.

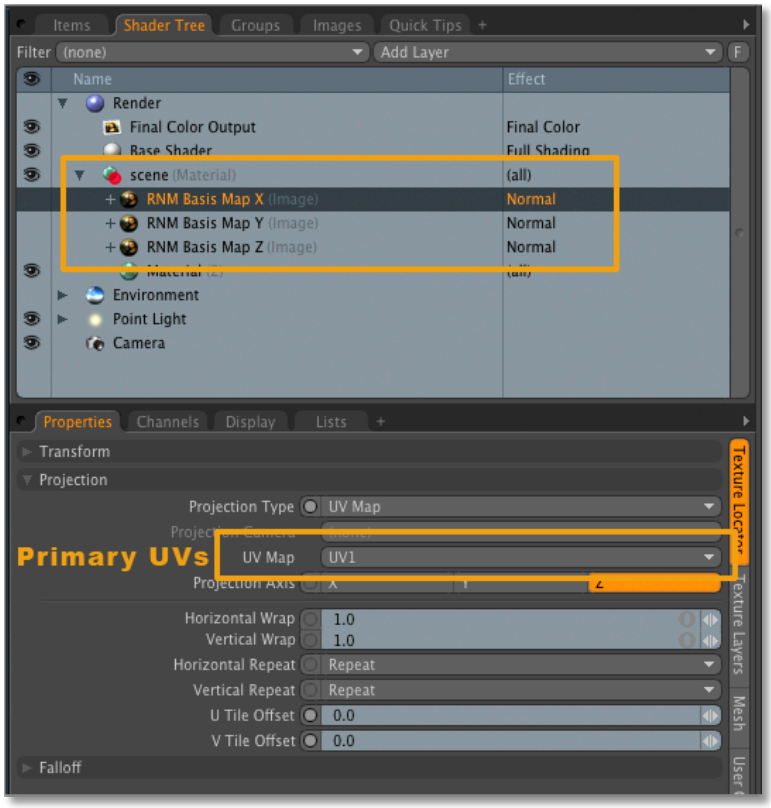

Figure 9 The RNM Basis textures are assigned to the primary UV set.

To render the RNM, make sure that you have a "Final Color Output" Render Output in your Shader Tree, select the lightmap UV set from the Lists > UV Maps viewport, enable the Eye Icon on the "RNM Basis Map X" texture and select **Render > Bake To Render Outputs** in the main menu. Be sure to set your Render setting resolution to a square, power of two texture size, such as 512x512 or 1024x1024.

It is recommended that each of the RNMs be saved in the EXR format so that the shaders will have full access to the scene's luminance. Again, before baking, remember to select the lightmap UV set as shown in Figure 10, as this is an easy step to overlook and a common place for a mistake to occur.

You now need to bake each of the RNM Basis Map components in the same manner. You can also, at this stage render, a color lightmap as this will be needed with the CRNM shaders, however, if you use GZCRNMMerger, a color CRNM will be generated for you automatically and you don't bother with the extra render.

Note: If you render a color lightmap, you will need to normalize the color. This can be done easily in Photoshop by just running **Image > Auto Color.*

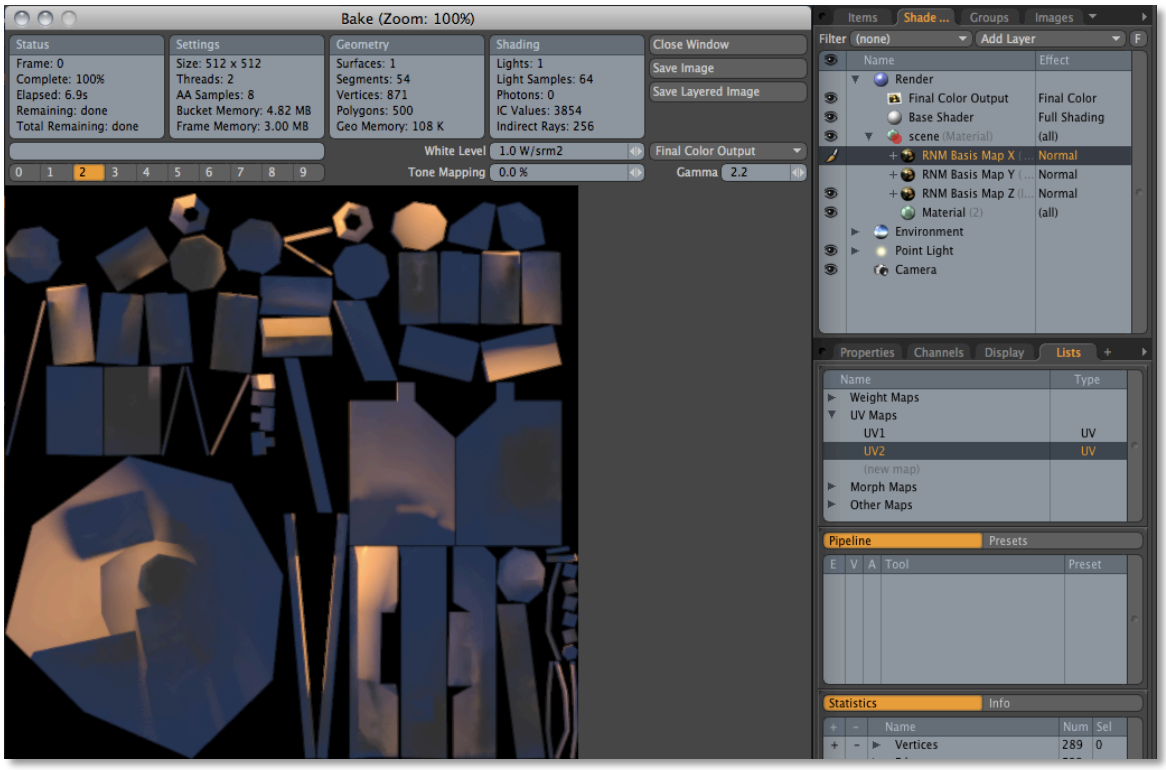

Figure 10 The RNM map is baked from the second UV set or lightmap UVs.

Rendering RNMs using Autodesk Maya

To render the RNM maps in Maya, you'll need to properly setup the RNM basis maps. Open the Hypershade and create a mia_material_x node and name it "RNM_X." In the Overall Bump channel of the mia_material_x node, click the add texture button and add the "RNM Basis Map X" texture as shown in Figure 11. That RNM Basis Maps can be found in the Snow Cat Solutions folder of your Unity project.

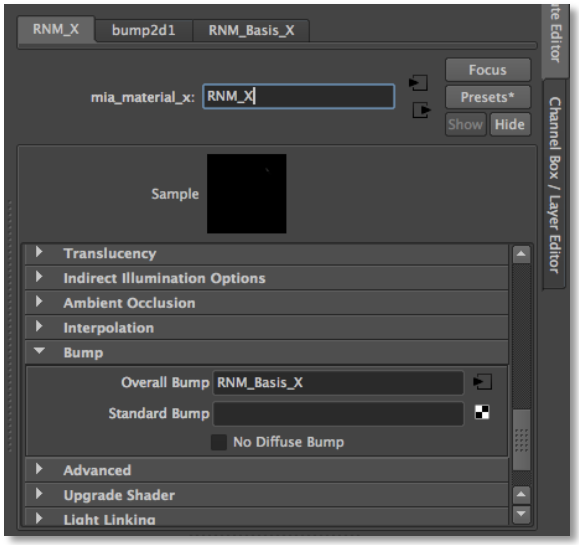

Figure 11 Add the RNM Basis X map to the Overall Bump Input

Set the bump2D node for the mia_material_x shader to use Tangent Space Normals as shown in Figure 12.

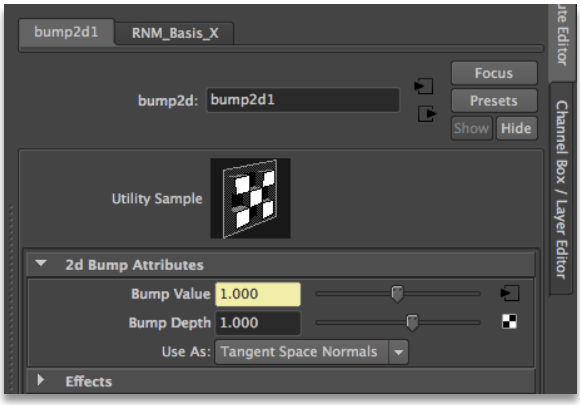

Figure 12 Set Use As: to Tangent Space Normals

Next, assign the mia_material_x shader to your object, set the Color on the shader to a middle-grey value and set the reflection to 0 as shown in Figure 13.

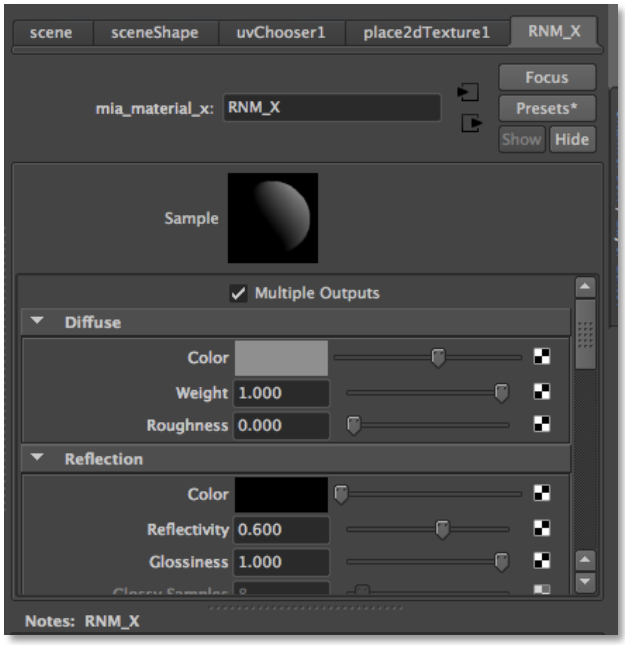

Figure 13 Set the color to mid-grey and remove the reflection values.

Now, you need to make sure that this RNM Basis map is using the correct UV set. To do this, use the **Relationship Editor > UV Linking > Texture Centric** to set the RNM Basis texture to the primary UV set as shown in Figure 14.

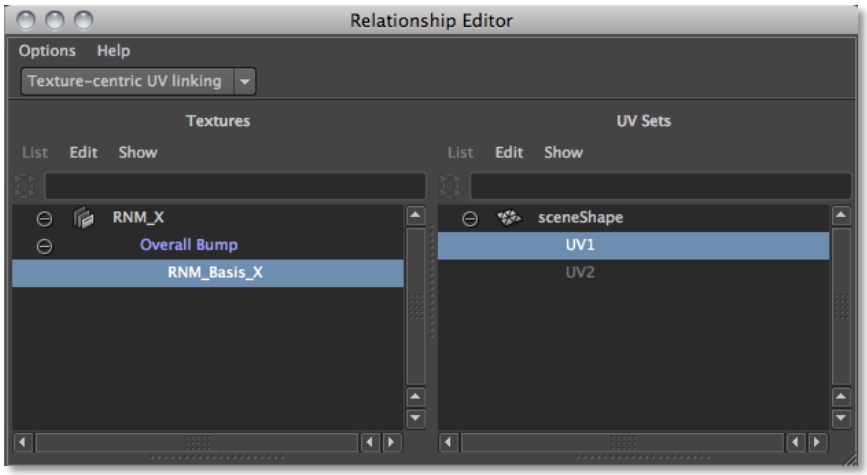

Figure 14 Set the RNM Basis map to use the primary UV set

Now, you're ready to bake the RNM map. To bake the map, go to **Lighting/Shading > Batch Bake (mental ray)** and open Batch Bake with Options. In Figure 15, you can see the options to use.

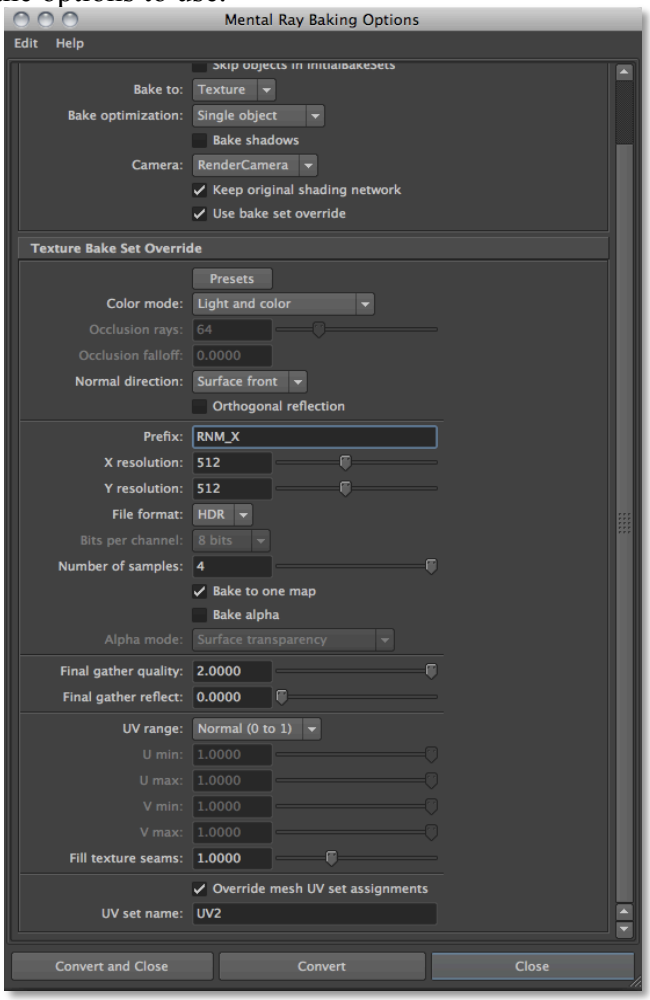

Figure 15 Use Batch Bake to bake the RNM map

With Batch Bake, you can override the mesh UV assignment. Here, we need to set the bake to the lightmap UV set by checking "Override mesh UV set assignment" and typing in the name of your lightmap UV set. Now, you can click the convert button to bake the RNM X map. You should set the file format to HDR, so that the GzRNM shaders will have full access to your scene's luminance. However, you'll need to save this file to an EXR file in Photoshop before it's imported into your Unity project.

From this point, you just need to duplicate the mia_material_x shader two times, apply the other 2 RNM Basis maps to each new shader and bake. Again, be sure that the UV assignments are set correctly for the RNM Basis maps and when you bake the actual RNM map. In Figure 16, you can see the shader graph for the mia_material_x shader.

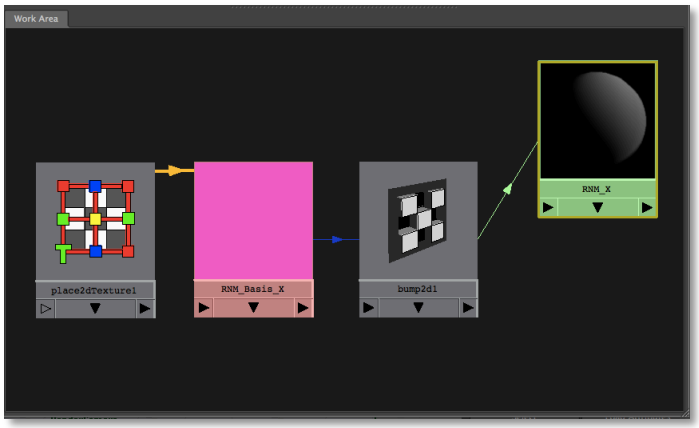

Figure 16 The mia_material_x shader graph

Creating CRNMs using CRNM Merger

If you plan on using one of the CRNM shaders, you can use the CRNM Merger utility to automatically create the CRNM and CRNM normalized color. Once you have rendered the 3 RNM component maps and imported them into your Unity project, you can run the CRNM Merger by going to **Utilities > GzRNM > CRNM Merger**. A window will appear that will allow you to select the appropriate RNM component as shown in Figure 17. Once you've added the RNMs, click "Create" to generate the CRNM and CRNM Color files. These new files will be saved to the same directory in your Unity project that the RNM component files are located.

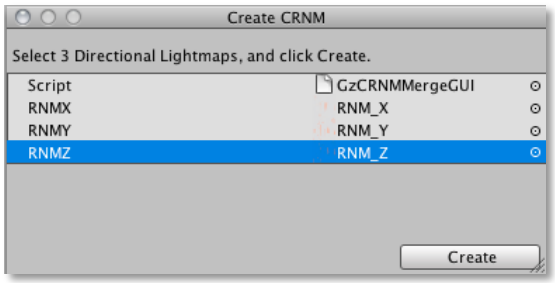

Figure 17 Use CRNM Merger to create a CRNM from the 3 RNMs.

Creating SSBump Maps using SSBump Converter

You can use the SSBump Converter utility to automatically generate an SSBump Map for you project. Simply go to **Utilities > GzRNM > SSBump Converter** to bring up the utility window and select a Normal Map or Direct-X style SSBump Map to convert as shown in Figure 18. If the Normal Map was generated using SSBump Generator, select the toggle box, "Made in SSBump Generator." Click "Create" to generate the file. The new SSBump will be saved to the same directory in your Unity project that the Normal Map is located.

If you would like to manually create an SSBump Map for more advanced control, you should http://ssbump-generator.yolasite.com/.

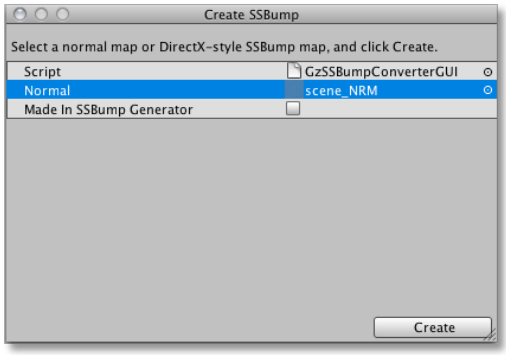

Figure 18 Use SSBump Converter to convert a Normal map to an SSBump map

GzRNM Shader Reference

Shader Parameters

SSBump Scale:

Scales the SSBump map effect. This may need to be increased to achieve desired look.

Use Dedicated Specular Map:

This setting fades between either a specular map stored in the diffuse map's alpha channel, or a dedicated colored specular map.

Specular Multiplier:

The parameter multiplies the specular effect by the user-defined value.

GzCRNM Shaders

CRNM

GzCRNM:
Compressed directional lightmaps that store light intensity information in one lightmap instead of 3 separate RNM component maps

GzCRNM Directional Specular:
Allows for specular shading from dynamic light sources.

GzCRNM Indirect Specular:
Allows for dynamic specular highlights from dynamic lights, and has UDK-like specular for everything else.

GzCRNM SSBump:
RNMs that support self-shadowed bump maps.

GzCRNM SSBump Directional Specular:
Allows for specular shading from dynamic light sources. Uses SSBump and Normal Map

GzCRNM SSBump Indirect Specular:
Allows for dynamic specular highlights from dynamic lights, and has UDK-like specular for everything else. Uses SSBump and Normal Map

Mobile

GzCRNMSSBump Diffuse:
Compressed directional lightmaps that also take advantage of Valve's self-shadowed bump maps in a really lightweight shader. Good for iOS and Android platforms.

RNM

GzRNM:
Basic RNM shader

GzRNM Direct Specular:
RNMs that support specular shading from dynamic lights.

GzRNM Indirect Specular:
Directional lightmaps with UDK-like specular.

GzRNM SSBump:
RNMs that support self-shadowed bump maps.

GzRNM SSBump Directional Specular:
Allows for specular shading from dynamic light sources.

GzRNM SSBump Indirect Specular:
Allows for dynamic specular highlights from dynamic lights, and has UDK-like specular for everything else.

Alpha-Cutout and Transparent

Each shader has a transparent and alpha-cutout version.