Realistic Water Using Bump Mapping and Refraction

By Ben "DigiBen" Humphrey

* IMPORTANT *

You must have a video card with GLSL shader support to run this.

You will be blown away with this tutorial as we show you how to generate realistic water using shaders. Be sure to put a towel over your keyboard so your drool doesn't get into the keys :) This builds off of our previous tutorial called HeightMap5 (volumetric fog). In the past, to create realistic water that was moving and refracting you had to have a great deal of triangles to simulate the water's wave; however, using normal and dudv maps we can create visually stunning effects that make it appear that we have a highly tessellated surface for the water. In

this tutorial we use a single QUAD (four points) to perform our water effect. The texturing and shaders gives us the effect. There is no water simulation with physics. The idea and mentoring for this tutorial comes from **Michael Horsch** with **Bonzai Software**. Let's go over many of the terms that you will need to understand for this and future tutorials.

* What is Reflection? *

Reflection is just taking the environment and flipping it upside down, clipping it to the surface that we want to show the reflection. In this lesson we only reflect our scene into the water surface.

Reflection in Water

* What is refraction? *

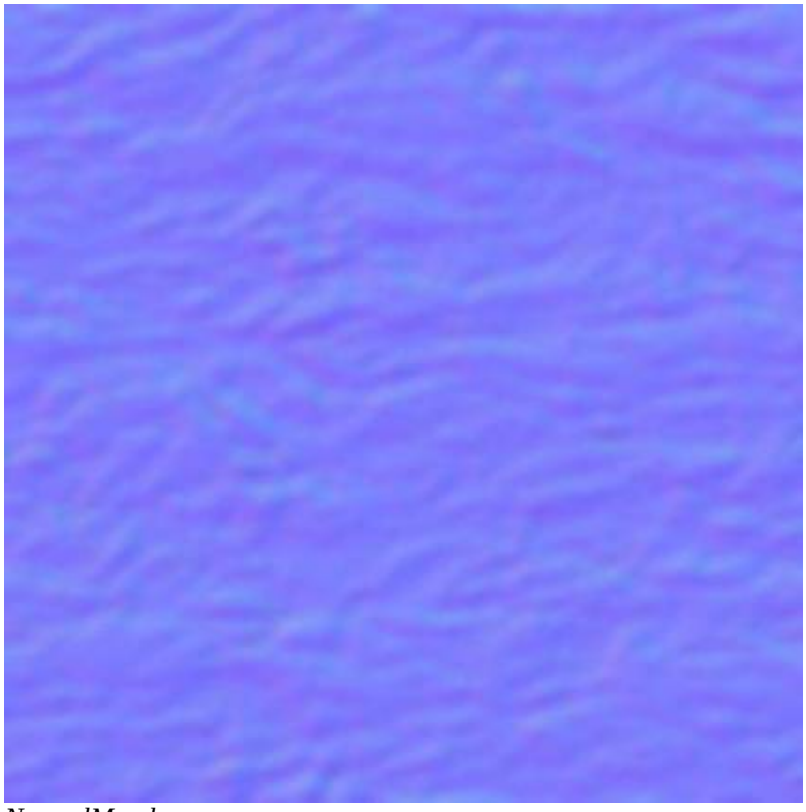
The best way to understand refraction is to run this demo and look at the reflection of the world on the water. Notice that the reflection moves as if there are waves moving it around? This is called "refraction". Basically, you are bending light around. Go to your cupboard and grab a glass. Look through the glass and notice the world reflects funny through the glass. This is also refraction. When dealing with water and glass surfaces you will use refraction to give the appearance of photorealism.

How is refraction done in real-time rendering? People a lot smarter than me created something called a dudv map. Check below for the description on dudv maps.

Refraction in Water

* What is a normal map? *

If you are familiar with bump mapping then you probably have heard of normal maps. Look at normalmap.bmp below and notice the pink, blue and purple colors?



NormalMap.bmp

Each one of those pixels has an RGB value right? Well, instead of using each pixel for a color, it stores a normalized normal from -1 to 1. The normals naturally give the image a purple, pink and blue color. What we do with this image is basically use it to bump map our water to give it the appearance of being highly tessellated and realistic. This realism comes when we use per-pixel lighting on the water. Basically, if we didn't have a normal map we would just depend on the normal of the water surface, which is pointing straight up. The light would look horrible and the realism would be absent. We also use the normal map for our Fresnel term calculations. More on this concept below. In the fragment shader set the normal to vec4(0, 1, 0, 0) and see the difference without the normal map.

There are a couple ways to calculate normal maps. First, you start with the normal image of the water (needs to be seamless or really close otherwise it will show funny seams in the water). I used the ATI's program called TGAtoDot3.exe. You can find it here:

http://www.ragehdtv.com/developer/sdk/radeonSDK/html/Tools/ToolsPlugIns.html

You can use a plug-in for Photoshop that NVIDIA distributes:

http://developer.nvidia.com/object/photoshop_dds_plugins.html

NVIDIA also has a program called "Melody" that loads a high-poly model and a low-poly version. It then creates a normal map from the high-poly version.

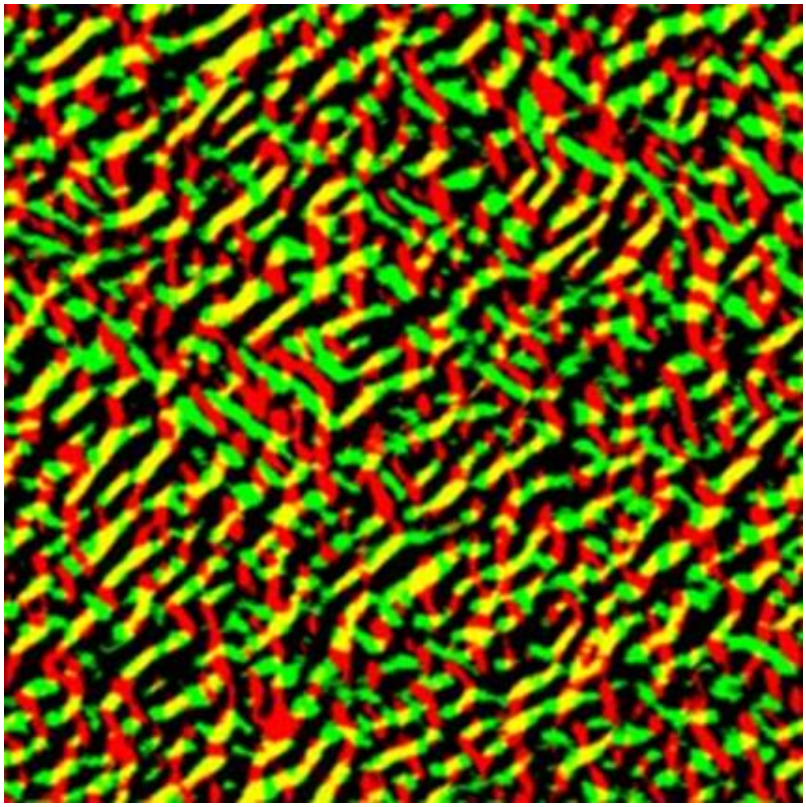
http://developer.nvidia.com/object/melody_home.html

HeliosDoubleSix gives a comprehensive list of links about normal mapping:

http://sv3.3dbuzz.com/vbforum/showthread.php?s=&threadid=77037

* What is a DUDV map? *

Look at dudvmap.bmp below:



DuDvMap.bmp

Notice the funny colors of yellow, red and green? This is called a dudv map. You can probably imagine why it is called that because it's actually the derivative of a normal map. In mathematics the notation for a derivative is du/dv. What this texture does is give us a way to calculate refraction and how the light will react and bend on the object. In our case, our object is the water. Notice how the dudv map looks like waves of water?

I used the TGAtoDUDV.exe program that ATI created. To create the DUDV map you do it from the normal map of the water image, not the original diffuse map. You can download this program on the same page as listed above on ATI's tools page:

http://www.ragehdtv.com/developer/sdk/radeonSDK/html/Tools/ToolsPlugIns.html

* DUDV texture trick for realistic water *

You will probably find that just using the dudv map spit out from the ATI program will give you less than desirable realism for water. To make the dudv texture work well you can do a little trick. Before creating the DUDV map from the normal map generated, create a separate normal image with a little Gaussian blur applied to it. Then use that image to create the dudv map.
Then apply a Gaussian blur to the resulting dudv map so it's not so perfect. This will make the water less "perfect" and sharp looking. Be sure to test different blur values for the effect that looks best with your water image.

* So where is the actual water texture? *

This might sound strange, but we actually don't even use the original water texture that we create the normal and dudv maps from. Can you see why? This is because our water texture is the reflection of the world onto the water. We create a texture on the fly by rendering the reflected world to a texture, then applying that texture to the water surface.

* What is Fresnel term? *

We mentioned above about the Fresnel term (also called Fresnel reflectance). This is a calculation that is used to determine realistic light reflecting off of a surface. In the 19th century there was a French man named Augustine Jean Fresnel (1788-1827) who was a pioneer in studying light and it's reflective nature. He developed a formula to calculate reflection of light that will
simulate realism with light reflections. You might have heard of a Fresnel lens before. It originally was used as a lens for lighthouses. Now days projectors use Fresnel's discoveries to project light.

Anyway, enough history. In our tutorial we don't use the full equation for Fresnel reflectance, but it can be approximated with the following equation:

Fresnel = 1 - SurfaceNormal . IncidentLightVector;

This means that we subtract the dot product of the normal and the incident vector (reflected light vector). If you are unfamiliar with an incident vector, this vector is the vector that "bounces" off the surface when hit. For example, if bouncing ball hits a surface at 45 degrees, it will most likely bounce off the surface at a 45 degree angle (assuming no other gravity and physics calculations). That bouncing back vector is the incident vector. So when the light hits on the surface, we need to find the reflecting vector coming off the surface. This is necessary for finding
the Fresnel term.

Notice that it also takes the surface normal. For our water, since we only have a flat surface we will want to use our normal map values for each pixel to approximate the appropriate light reflections.

If you comment out the Fresnel reflectance calculations in our shader you can see the necessity for using Fresnel term. The contrast looks horrible and not very realistic.

* The use of a depth map *

We can calculate the fogginess of the water by the use of a depth map. We render our current view to a texture using just the depth values in the z-buffer. We use these depth values to further add realism to our water lighting and reflection. The depth buffer will tell us how far away the pixel is from the camera. We can use that value to make certain areas of the water darker and certain areas lighter. The depth texture is created by just the world and the terrain, not the water. This way deeper parts of the water will show that. It's a great way to handle the fogginess of the water.

Since we already calculate our own fog we don't actually need the depth map, but I chose to use it to show you how to calculate fog this way. I also added it because it makes the surface a lot darker, which looks better in my opinion.

* What is bump mapping? *

This tutorial uses bump mapping to make the water look tessellated. If you have a low-poly model and you want to make it look like it's more detailed you can create a normal map to give it that effect. If we use per-pixel lighting then the model will have incredible detail and will light the model as if there is extruded detail. To do bump mapping we need to do some calculations in "tangent space".

Bump-Mapped Water

* What is tangent space? (also called texture space) *

You may have heard of tangent space before, also sometimes referred to as texture space (by NVIDIA). Each vertice basically has it's own axis, with the normal pointing up as the Y axis and a tangent and bi-tangent creating the X and Z axis respectively. This XYZ tangent-space axis is calculated for every vertice that will be bump mapped.

The reason why we need to convert to tangent space is because we use a normal map that stores normals from only one view (the view it was created). When the world and objects in the world are transform, rotated and scaled, it doesn't line up with the normals in the texture map.

Think of it this way: you create a normal map of a 3D character facing forward, but when the character is moving in tons of different directions you need to either rotate the normals in the texture map every time you rotate the character, or you just convert the vertices, light and camera position to tangent space. Can you see why it's also called "texture space"?

In this tutorial we don't focus on calculation the tangent space for vertices because we only have one flat quad that has a tangent-space axis of:

```
tangent   = (1.0, 0.0, 0.0, 0.0);
normal    = (0.0, 1.0, 0.0, 0.0);
biTangent = (0.0, 0.0, 1.0, 0.0);
```

However, we do need to calculate tangent space for the view and light vectors. In code we do it like this:

```
viewTangetSpace.x = dot(viewDirection, tangent);
viewTangetSpace.y = dot(viewDirection, biTangent);
viewTangetSpace.z = dot(viewDirection, normal);
viewTangetSpace.w = 1.0;

lightTangetSpace.x = dot(lightDirection, tangent);
lightTangetSpace.y = dot(lightDirection, biTangent);
lightTangetSpace.z = dot(lightDirection, normal);
lightTangetSpace.w = 1.0;
```

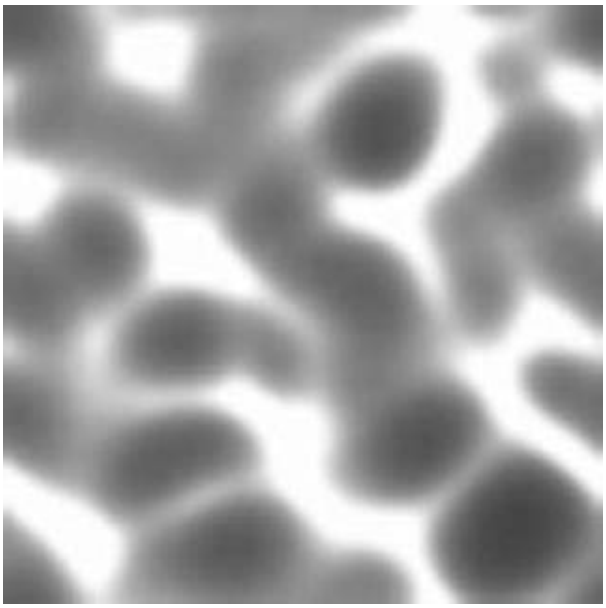
Now when we calculate our bump mapping depending on our light and view, it will already be aligned with the normal map's original axis.

* BiTangent verses BiNormal *

Apparently BiTangent and BiNormal have been used synonymously as the same thing, which isn't true. Though BiNormal is used all over the place, many have pointed out in the graphics world that it is incorrect. BiTangent should be used when speaking about tangent space in 3D. We will use BiTangent instead of BiNormal for this tutorial to try and correct this minor mistake.

* What are caustics? *

Caustics are the effect of texturing objects underwater to give the illusion of light reflecting/refracting through the surface of the water. We have a few dozen textures that simulate this effect. The textures are multi-textured onto the surface of the terrain. We only show them when we are under the water.



* SUMMARY - How does the water effect work? *

In summary, let's explain in a nutshell how our water effect works. We have a single QUAD across the terrain for our water. Every frame we render to three textures (reflection, refraction and depth). Then in our shaders we tile the normal and dudv map, while incorporating the correct equations to add all of the texture maps together. We then add specular lighting and voila!

By viewing this tutorial we assume that you have looked at the previous tutorial HeightMap5, as well as our beginner GLSL shader tutorials. The files that you will want to look at for this lesson is Main.cpp, Main.h, Water.cpp, Water.h, Water.vert and Water.frag. We added CShader.cpp and CShader.h to this project, but there is nothing new in those files from the beginner GLSL shader tutorials.

CONTROLS:

The controls are just like the last tutorials. The camera is moved by the mouse and the arrow keys (also w, a, s, d). The mouse buttons turn detail texturing on/off as well as switch to wireframe. The fog in the water can be changed by the + and minus keys. We also added these keys:

F1 - Makes the water appear farther away by shrinking the water texture
F2 - Makes the water appear closer by enlarging the water texture
F3 - Makes the water move faster

F4 - Makes the water slower

***** Be sure to use the F-keys to get some amazing effects. *****

www.GameTutorials.com