

# Simulation of Water Creatures

## A Tutorial by Henry Bush

It has been said that humans are so difficult to replicate because we see them every day: the same must be true of water. In this tutorial I shall endeavour to explain how to overcome some of the difficulties encountered when trying to create a creature that is made of water, and is merged with another object made of water.

1. The first step is to create the objects that will be used. For this tutorial, I shall use a water surface created from a cube and a modified cone as my “creature” (Figure 1). This could be controlled by bones, shape animation, or using any other method.

I have chosen to subdivide the top face of the cube I shall be using, as we shall be adding a displacement map to it later.

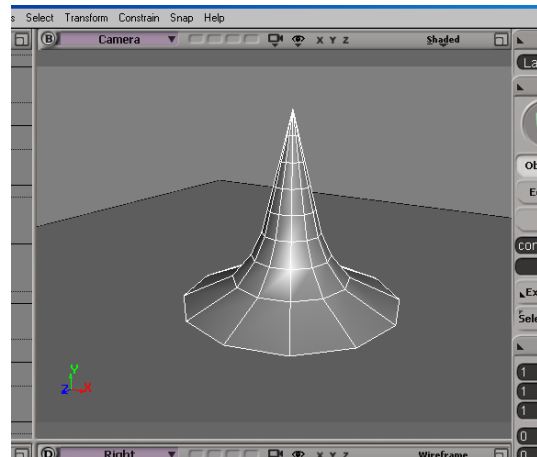


Figure 1

2. The first thing to do is to apply a simple surface shader, the same shader to both objects, as we want them to appear to be one object. I have chosen to use the Cook-Torrance illumination model in this tutorial, with the parameters shown in Figure 2.

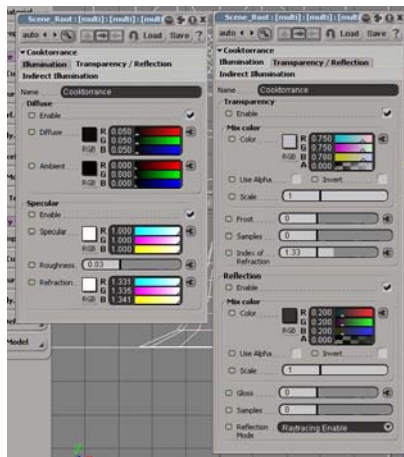


Figure 2

The only difference between this and Phong is in the way that specular highlights are calculated: when the angle between the camera and the surface is very small, the Cook-Torrance model produces larger highlights to emulate the way rough surfaces scatter more light at lower angles. The Cook-Torrance model also takes into account the difference between different colours of light, resulting in coloured fringes around specular highlights. Good results can be obtained using the Phong illumination model, but the specular highlights that are produced by the Cook-Torrance model are, in my opinion, closer to those observed in real water.

3. If you examine a render region, you will notice that it currently shows nothing: this is because our objects have nothing to reflect or refract. It is best at this stage to introduce a “sky sphere”: this is a very large sphere with a constant texture map, used purely to produce reflections. I tend to use a landscape image (in this case <http://negin.org/photo/Kolskiy96/landscape.jpg>), but any image will do.

Alternatively, an environment map can be used: this is applied using an environment shader in the current pass settings. Though it is quicker, the results are less accurate, and thus I have chosen to use a sky sphere in this case.

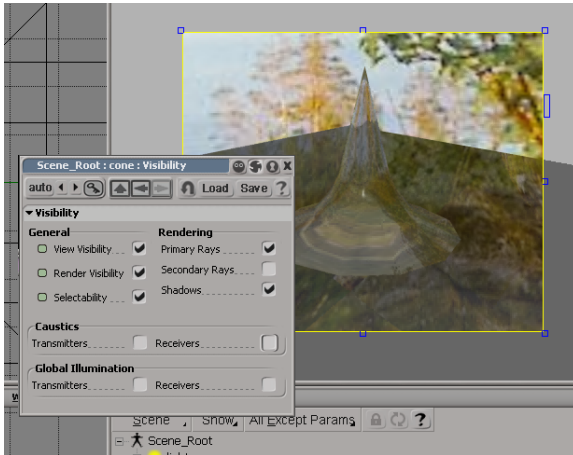


Figure 3

- Our first objective is to merge the cone with the cube. The first thing to do is turn off *secondary rays* in the *visibility* options of the cone (Figure 3).

This is not always necessary, but if the cone (“creature”) will ever move below the surface of the cube then it is: otherwise, a phantom cone will be visible below the surface of the cube.

In order to hide the join between the two objects, we shall fade one object (the cone) out, so that at the interface between the two, it is invisible. There are other methods for getting this effect: one is to use a Boolean operator such as union on the two objects, and then apply a texture to the single object produced. Though this works well in theory, the Boolean operator is not designed to work on the fly (i.e. recalculating for each frame), and frequently fails to produce any result. For this reason I have chosen to fake the effect in a much more reliable way.

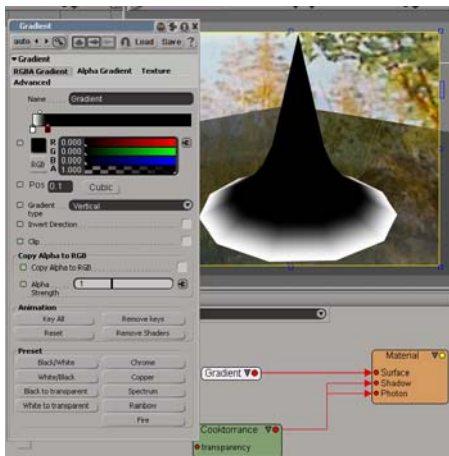


Figure 4

- Open up a render tree for the cone, get a **Texture** → **Gradient** node, and apply a planar *xy* projection to it. Plug it straight into the *surface* of the **material**, so we can see what we are doing. Click on the *White/Black* preset, and adjust the square markers until the effect is similar to that shown in Figure 4.
- We want to fade from water to invisible rather than black to white. Let's plug our **Cook-Torrance** back into the *surface*, and open up its property page. Right-click on the plug symbol next to the transparency *Mix Color*, and select *Blend with... Image*.

This will bring in an image node. We are not interested in this, so delete it. However, the process also brings in an **8-way mixer** node (also available on the **Mixer** menu) with some handy default values. Set *Color 1* to white, unclick *Multiply weight by alpha*, and plug our **gradient** in as the *weight*. This will fade our transparency from its current value (0.75, 0.75, 0.78) to white.

Repeat this procedure for both the *diffuse* and *reflectivity*, but set *Color 1* to be black in these cases. This is because we do not want the cone to react to light at all, but let the cube below reflect. Your render tree should now look something like that shown in Figure 5.

7. You will notice that the join is still very visible: this is because we have not yet faded out the index of refraction. This is slightly trickier than the previous parameters, as it is a scalar and not a colour. For this, we shall use a **Math** → **Change Range** node. Pull one up, and look at the values. This node changes the range of a scalar in a very straightforward way. If we consider our **gradient** (Figure 5), we want the refractive index to be 1.33 (water) where the **gradient** is black (a scalar value of 0), and 1.00 (invisible) where the **gradient** is white (1). Therefore, we change the range from  $0.00 \rightarrow 1.00$  to  $1.33 \rightarrow 1.00$  (Figure 6). Now plug the **gradient** in to the *input*, and the output into the *index\_of\_refraction* of the **Cook-Torrance**.

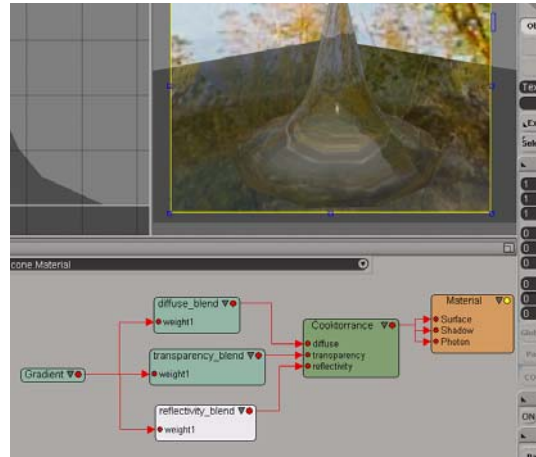


Figure 5

This may not look like the join is completely seamless, but this is probably just as it is the only place that the surface deforms. Let's add some little ripples to it.

8. Open up the render tree again, get a **Texture** → **Fractal** node, apply a spatial projection to it, and plug it in to the *displacement* of the **material**. Though this will work, the outcome is not what we intended: the renderer subdivides the mesh at render time to effect displacement mapping, and it is currently not set up very effectively.

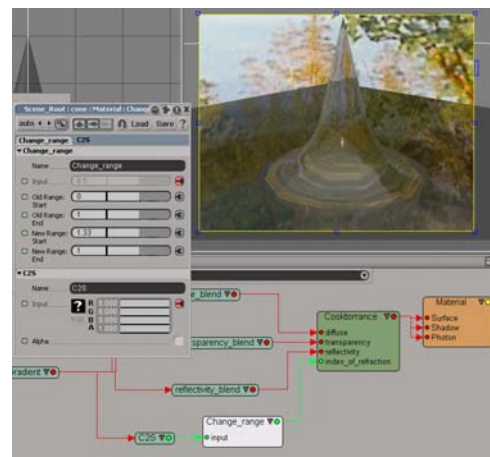


Figure 6

With the cone selected, open up its *Geometry Approximation* box, go to the *Displacement* tab, set it to *Fine*, and click *View Dependent* on. In order to produce a more ripple-like effect, scale down the Fractal node using a **Math** → **Color Math Basic** node. Set it to multiply, plug the **Fractal** into *Base Color*, and set *Color 1* to be 20% grey. Then go in to the **Fractal** node itself, and set *UV remap maximum* to be 3,3,3. This should give a more ripple-like result.\* Apply this shader to the cube as well, to give a seamless rippled surface (Figure 7).<sup>†</sup>

Now that we have some fairly realistic deformation on the surface, we shall further improve the surface shader. In real life, water does not simply reflect 20% of the light that hits it: the amount of light reflected depends upon the viewing angle. If the view

\* If you wish to animate the ripples, I have found that animating the *Time* parameter of the Fractal shader one unit over five frames is about right.

<sup>†</sup> At this stage, it is likely that render times are becoming more lengthy. In order to speed it up, try hiding the cube or disconnecting the displacement maps unless you need them.

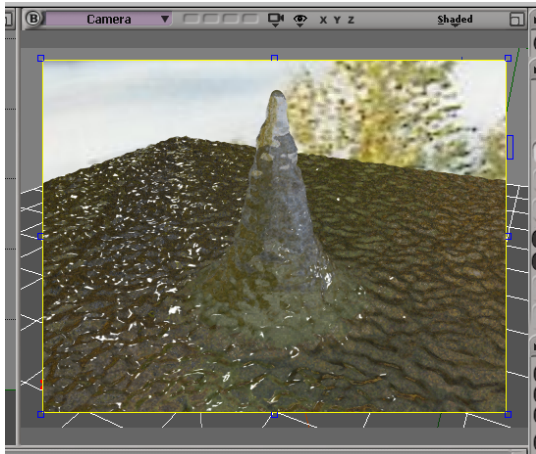


Figure 7

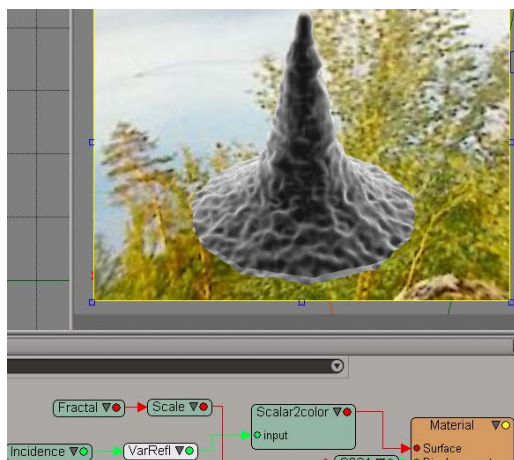


Figure 8

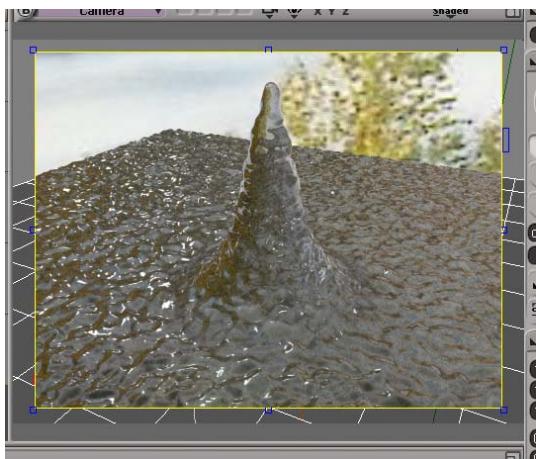


Figure 9

scalar2colour node in to an **Image Processing** → **Color Balance** node. Tweak the blue up a little bit, then plug this into the *Base Color* of your **transparency\_blend**.

direction is perpendicular to the surface plane, little light is reflected, whereas if the view direction is almost in the viewing plane, the surface reflects much more light.

- Open the render tree, get an **Illumination** → **Incidence** node, and plug it into the **surface**. This node (with its default values) gives a value of one when the surface is facing the camera, and zero when the surface is perpendicular to the camera.

In order to get this node to drive our reflectivity, we must first change its range. Get another **Math** → **Change Range** node, and this time set the output range to be the desired range of reflectivity (Figure 8). I have used 1.00 → 0.08, but have a play and see what works for you. Then plug this node in to the *Base Color* of the **reflection\_blend** or whatever you renamed it.

I would highly recommend renaming render tree nodes. This is partly to help you to remember what each one does, but mostly because it saves space in the render tree, which has a habit of getting very crowded very quickly. Also, collapse any nodes that have only one input.

- As the reflectivity of real-life water increases, so the transparency decreases (we can't go creating light now!). Plug your **incidence** shader into another **Math** → **Change Range** node, and set the range from that to fade the *transparency* from 0.75 to 0. Plug this into the *Base Colour* of your **transparency\_blend**.

The more astute amongst you will have spotted a problem. Our transparency used to be slightly blue, to mimic the bluish tinge that water has. As we have just plugged a scalar in as our *transparency*, we have removed that. In order to get it back, plug the *output* from the

Perform operations 9 and 10 with the cube as well, plugging directly into the *transparency* and *reflectivity* of the **Cook-Torrance** (Figure 9).

This gives the basics of the water creature: all that is left is to add more ripples. In Figure 13, I have used a **Texture** → **Ripple** shader to apply some larger ripples over both objects, using a planar  $xz$  projection. Though the effect works ok, the ripples on the cone itself are almost non-existent, as the area of the cone in the  $xz$  plane is very small. At this point it is worth remembering that we have two objects that merge seamlessly, so as to appear as one. Therefore, one solution to the problem is to use different parameters for the ripples on the cone than on the ripples on the cube (Figure 14). This gives passable ripples on both, but a better solution is to use a completely different projection for the ripples on each object: in Figure 15 (next page, including complete render tree), a planar  $xz$  is used on the cube, but a planar  $xy$  is used for the cone, giving much more consistently spaced ripples.

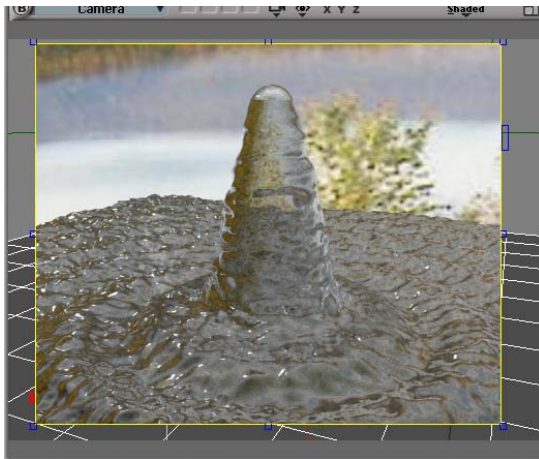


Figure 10

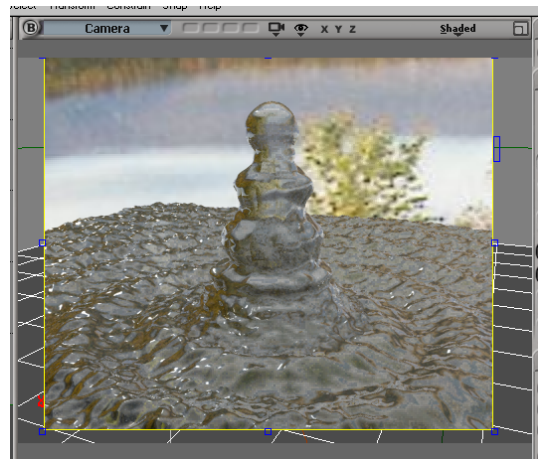


Figure 11

The creation of non-existent, fantastical creatures is one of the many attractions of 3D computer animation. I hope that this tutorial lets the reader's imagination run wild, and realize for everyone to see the creatures that previously only existed in their head.

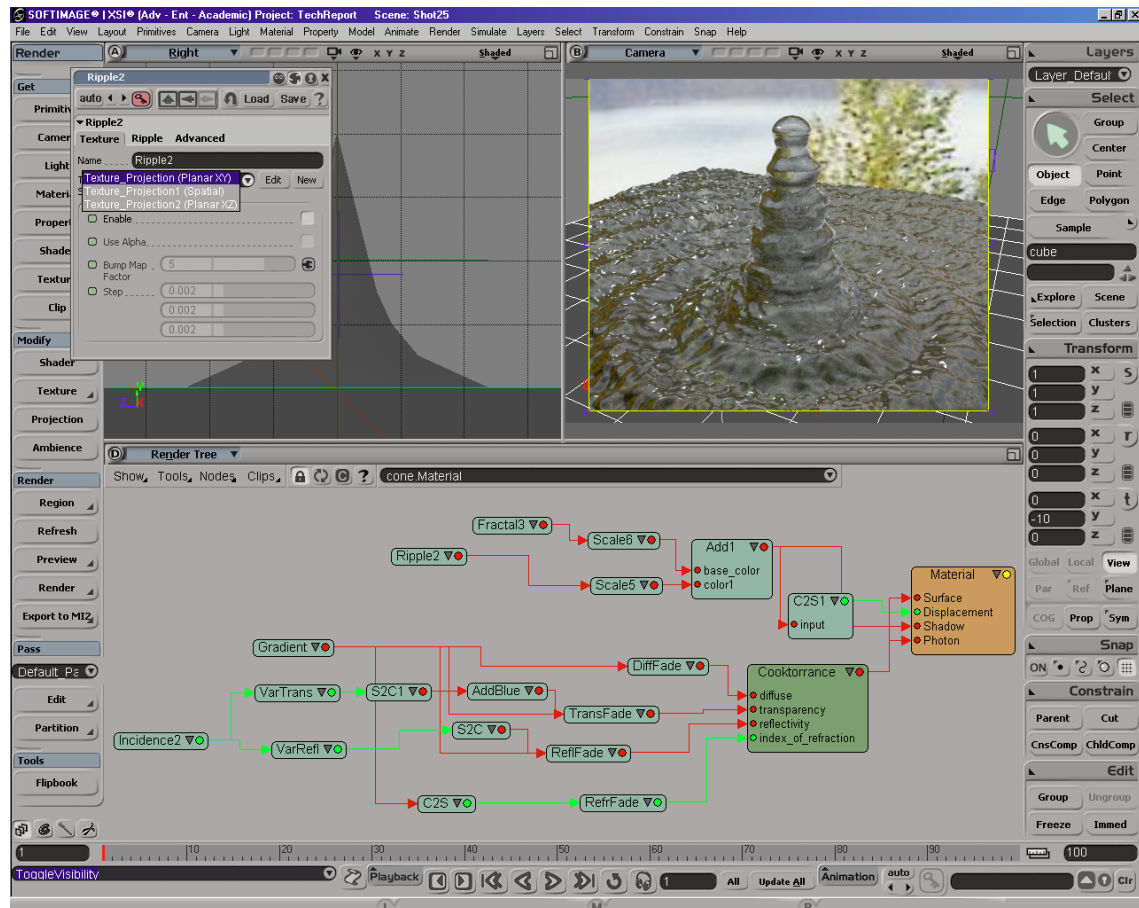


Figure 12

## Bibliography

T. Driemeyer, *Rendering with mental ray*, Springer-Verlag 2000.