Re-creating the World: An Overview

In this chapter, I introduce some of the main themes that come up throughout the book, and give a brief description of what you can expect in later chapters. Then I move on to some important general-purpose tools in Blender that can be of great use in modeling and animating certain physical phenomena. Some of the features covered here, such as node-based materials, are things you will use in conjunction with advanced physical simulations. Other features will come in handy when full-fledged physical simulations would be unnecessary or inappropriate. Some of the techniques I describe here use common tools such as textures and modifiers in ways you might not have thought of.

1

Chapter Contents

Re-creating the Physical World with Blender

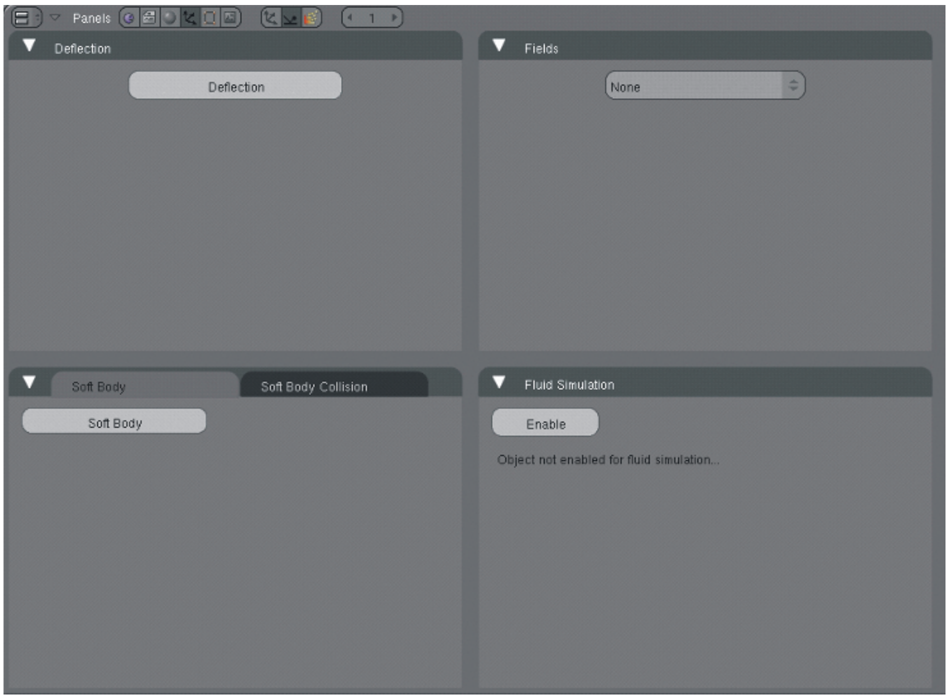
If you're reading this book, you probably already know that Blender is a powerful tool for creating 3D imagery and animation. You probably also know that it can be difficult to find good documentation on a lot of Blender's advanced functionality. There is of course the official Blender wiki, which is an invaluable resource, and numerous excellent online tutorials. Nevertheless, getting good, thorough information about many of these features can be a challenge. Unfortunately, this means that many of Blender's most exciting features tend to be underused; artists want to create, not spend their time searching for tutorials and documentation.

I'd like to change that with this book, by giving a complete introduction to Blender's intermediate to advanced functionality. I hope that this book will help artists become familiar enough with this functionality to comfortably incorporate these features into their creative workflow. Unlike in my previous book, *Introducing Character Animation with Blender* (Sybex, 2007), which has readers model, rig, and animate a character over the course of the book, there's no single, overarching goal here. This book, like the physical world it aims to simulate, is a bit of a hodgepodge. Each chapter addresses a specific aspect of Blender's functionality. There's not much dependence between chapters, so you don't need to read the chapters in strict order. Regardless of the order you choose to approach it in, I hope that this book helps unlock some of the mysteries of Blender's advanced functionality and gives Blender artists some powerful new tools to create their worlds.

Blender's Physical Simulation Functionality

Blender has a variety of tools for simulating physical phenomena. For Blender users who have not yet had the opportunity to study these tools closely, some of them can be a bit confusing. Not all the effects discussed in this book are located in the same place in Blender, and not all are activated in the same way. Getting them to interact in the ways you want them to requires a certain degree of understanding what's going on. In particular, this book is geared toward animators who want to incorporate physics simulations into actual animated scenes. This is not at all difficult to do, but it may sometimes require stepping out of your comfort zone. For example, although the Blender game engine and its integrated Bullet physics engine is widely used by Blender game creators, that area of Blender's functionality remains unexplored by many animators. One of the things you will learn in this book is how to access the various physics-related tools in Blender and put them to work to create the animations you are after.

The main focus of this book is on the features found in the Physics Buttons area (Figure 1.1) and the Particles Buttons area (Figure 1.2), and on the Bullet physics engine, which is accessed through the Blender game engine. The parameters for the game engine are set mostly in the Logic Buttons area (Figure 1.3).



Object not enabled for fluid simulation...

Figure 1.1 The Physics Buttons area

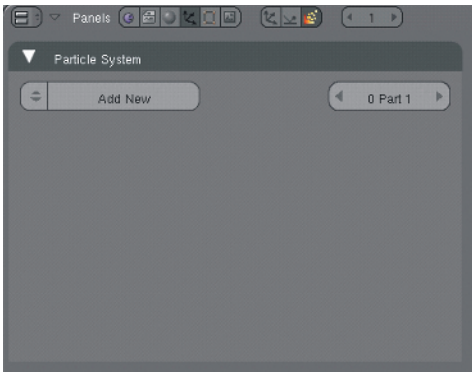


Figure 1.2 The Particles Buttons area

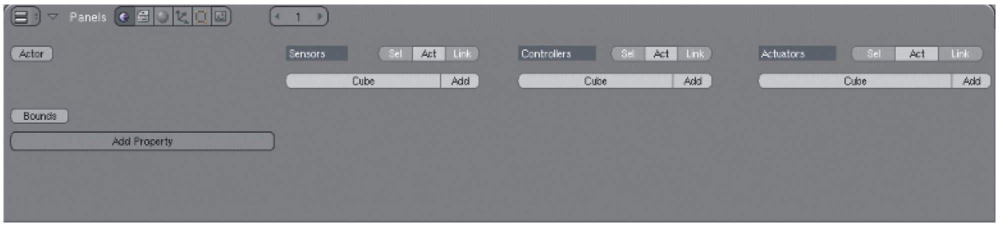


Figure 1.3 The Logic Buttons area

The simulation features available in the Physics Buttons area are the following:

Particles Particles are used to simulate very small real-world objects whose collective behavior can be calculated by the simulator. Particles are often used to simulate smoke and fire, by assigning translucent halo materials to each particle. Particles can also be used to simulate swarm behavior—for example, in the case of insects—and Blender 3D objects can be treated as constituents of a particle system by use of the Duplivert tool. Parameters relating to the speed, direction, life span, and other traits of particles can be set. Using "static" particles, the trajectory of each particle from the emitter is also modeled in the 3D space, to simulate strandlike real-world objects such as hair. Numerous options exist for controlling the behavior and appearance of static particles. In this book, both ordinary and static particles are covered.

Soft Body This simulates the behavior of materials that are flexible, and are visibly deformed by the forces acting on them, such as friction, gravity, and others. In Blender, soft bodies' vertices can move relative to each other, but their underlying structure does not change. Depending on settings, soft body simulation may be appropriate for modeling rubber, gelatin, sheet metal, cloth, and other materials. As you will see in Chapter 4, soft bodies can also be used with static particles to control hair, which also exhibits soft body behavior.

Fields and Deflection These settings are pertinent to both particles and to soft body simulation. Fields are options for external forces that can be made to act on other objects that have soft body properties or particle systems active. Among the fields available are curve guide fields, wind, vortex, and spherical fields. Each of these field options is discussed throughout the course of this book.

Fluid Simulation The Fluid Simulation panel contains settings pertinent to fluid simulation, and any objects that will play a roll in the fluid simulation, including any obstacles, inflow or outflow objects, and the object that acts as the domain for the simulation and marks off the spatial area in which the simulation will take place.

These features are all in some way connected to the internal structure of objects. In soft body simulation, the mesh structure is affected; in fluid simulation completely new meshes are constructed on the fly; and in particle simulation mesh structure is bypassed entirely in favor of a different approach to calculating the location and relationship between points.

For forces that operate on objects and enable rigid bodies to interact as they would in nature, with mass, gravity, friction, and collisions accurately modeled, it is necessary to use the Bullet physics engine, a component of Blender's game engine.

In this book, I show examples of all of Blender's various physical simulation functionalities, and give examples that I hope will encourage you to come up with creative and innovative approaches to using the simulators to get the effects you want. Particles, soft bodies, fluids, and rigid bodies are all included, as are the various deflection properties and forces that enable other objects to interact with these simulations. These topics make up the lion's share of what this book is about. Although numerous tutorials and examples can be found online for most of this functionality, these features have not been well documented in book form until now, and I hope to provide a coherent overview of all of Blender's physical simulation functionality. To take advantage of Blender's rigid body physics capabilities, it is necessary to do some work in the game engine environment. Because this book is written primarily with animators in mind, I don't assume that you have experience with that, and include a brief overview of how to get what you need out of the game engine.

The Science of Simulation

The main objective of this book is to give CG animators a complete understanding of the various physical simulation tools available in Blender, to enable them to realize their creative ideas. For this reason, the book is "artist-oriented" and is intended to be only as technical as is necessary for artists to understand what is possible and how to achieve it with the available tools. Nevertheless, Blender being an open source project as it is, there is another audience I would like to reach with this book, and that is the audience of technically-oriented readers who may be inspired to take an interest in the area of physics-based animation research or, if already engaged in such research, may be motivated to use Blender in their work or to contribute to Blender's code base. For those people, I hope that this book can serve as a way to become quickly familiar with the Blender approach to physical simulation, and as a demonstration of what Blender is capable of at present. People who are aware of recent developments in physical simulation will quickly identify numerous areas in which Blender could be extended and improved.

Physical simulation for 3D animation is an active field of research. Although it is a subfield of general physics simulation, the requirements and objectives of 3D animation are different from those for other kinds of physics simulation. Simulations used for engineering purposes, for example, must be able to compute real-world effects of physical situations with great accuracy, and they don't have to be especially fast. When planners of a skyscraper use fluid simulations to evaluate how the structure will hold up to wind, it is worth a considerable amount of time and money to ensure that the results in terms of forces and stress are as physically accurate as possible down to very small details, and at the same time the visual aspect of this simulation is likely not to be of interest at all. An example of a powerful open source package for computing these kinds of simulations is the OpenFOAM tools from OpenCFD. On the other hand, simulations for animation must be visually convincing, and computationally efficient enough to be carried out in a reasonable time frame on an appropriate budget, whereas very little depends on precise accuracy.

As Kenny Erleben and his coauthors amusingly observe in *Physics-Based Animation*, another difference is that the "artists and creative people" who use these simulations have "very little respect for the real world." Animators are constantly creating situations that would be impossible or highly improbable in the physical world, and it's important that the tools they use offer them the ability to create what they want to create, regardless of its physical authenticity. This is why CG armatures for character animation are so different from human skeletons; in Blender it's possible to stretch, bend, and swivel bones into any kind of position you like. A physically accurate representation of a human would not allow this. In the area of physical simulation for CG animation, the line between flexibility and robustness on the one hand and verisimilitude on the other is a key consideration. As in other areas of Blender, very high levels of accuracy are not the objective. In the same way that Blender's modeling functionality is not oriented toward advanced computer-assisted design (CAD) work, the approaches taken to physical simulations are intended to be useful to artists, not aeronautics engineers. Not to say that CG animators are not rocket scientists, but indeed, their needs in terms of fluid dynamics simulations are quite distinct!

If you're interested in the technical underpinnings of the physical simulations described in this book, a good place to start reading more is the home page for the Bullet Physics Library, at www.bulletphysics.com. In the forum area of that website you'll find a special area dedicated to links, research papers, libraries, demos, and discussions of Bullet and other physics software. For technically inclined readers, the previously mentioned book, *Physics-Based Animation* by Kenny Erleben, Jon Sporring, Knud Henriksen, and Henrik Dohlmann (Charles River Media, 2005), provides an excellent overview of many of the main methodologies used in physics simulation for CG. The book goes into considerable detail about kinematics, interaction of multiple rigid bodies, soft body and fluid simulations, and collision detection, among other subjects.

Nonsimulation Tools and Techniques

As you'll see throughout this book, there are cases when physical simulation can be demanding in terms of time and computing resources, and cases when the results are not exactly what you have in mind for your effect. For these reasons, it is important to have a solid understanding of your other options for creating physical effects. Even without dipping into sophisticated simulation methods, you'll find that there is an awful lot in the toolbox of Blender's standard features. A good understanding of materials, textures, and modifiers will enable you to think creatively and to approach each simulation challenge with freedom and flexibility. The remainder of this chapter is a tour of tools and techniques that have not been covered in much depth in print in the past. None of the techniques touched on in this chapter are crucial to understanding the rest of this book, but I think that people who want to get the most out of Blender will find them well worth checking out.

Using Materials and Textures

This book is about approaches to simulating physical phenomena. To do this, you can use a variety of tools to calculate the internal and external forces that are acting on objects and that determine the movements and the structural behaviors of the objects. But modeling the movement and deformations is not the whole story. No matter how realistically the objects in your scene collide, and no matter how convincingly the liquid in your fluid simulation flows and splashes, no one will be convinced if everything is the same opaque, dull, gray material. Blender has a powerful system of materials and textures that can be used to create stunning effects such as the mountain in Figure 1.4. Because this book is for intermediate to advanced users of Blender, it's assumed that you know the basics of using materials and textures, but there are a few techniques and functions that are either relatively new in Blender, or else are specific enough in use to be worth describing here in detail.

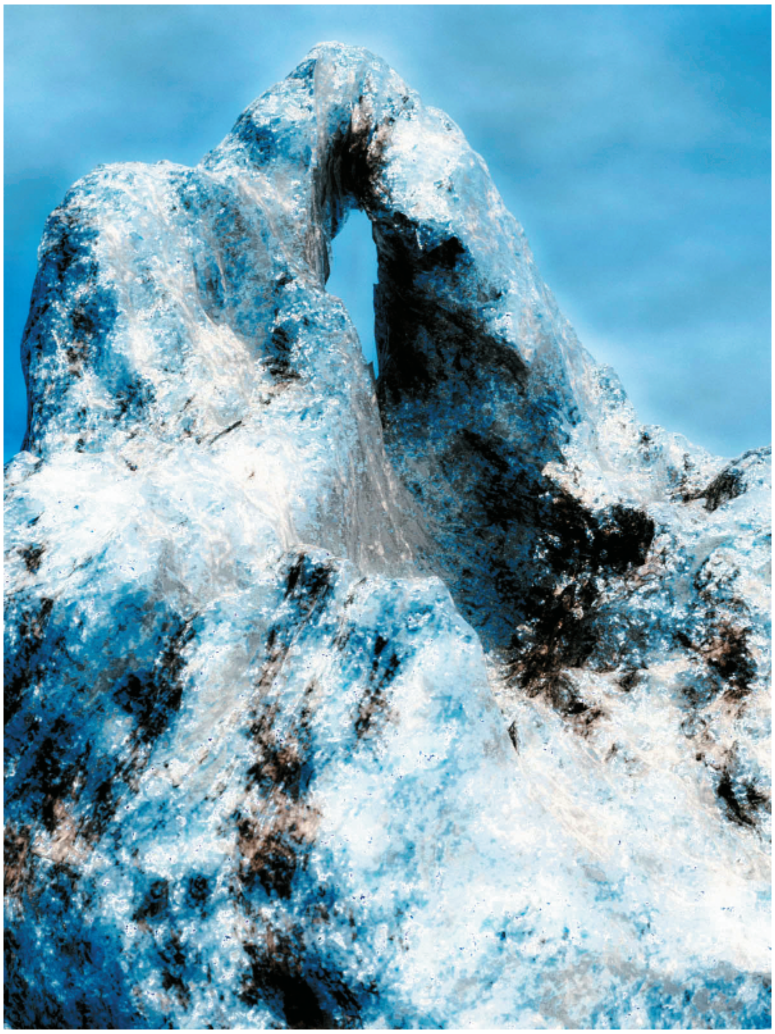


Figure 1.4 *Mount Reckon* by Robert J. Tiess is an excellent example of the potential of Blender's procedural textures.

Hot Lava with Material Nodes

The nodes system greatly increases Blender's power and flexibility in creating materials. By using nodes, it is possible to mix materials with different qualities in a wide variety of ways. In this example, I demonstrate how to use the system to combine two materials to achieve a simple but effective lava effect.

A notable feature of flowing lava is that it is composed of two very different main "materials" (of course, in reality, the material is the same, just in a different state). There is the flowing, molten part of the lava, which is liquid and glows from heat, and there is the portion of the lava that has been exposed to cold air and hardened, which is a dark-colored rocky crust. In Blender, it is possible to create these two materials separately and then combine them by using nodes. To create this material, follow these steps:

1. Delete the default cube and add a UV sphere with default values (32 segments, 32 rings, radius 1). Set it to have an active material called Material.
2. Select Nodes in the Links And Pipeline tab of Material, as shown in Figure 1.5. This makes Material a node material. If you are not used to working with node materials, it is easy to get confused, so it helps to keep track of which materials need to be node materials and which materials do not need to be. In most cases, materials that you use as inputs into node materials should not be node materials themselves. You can always tell which materials are node based and which are the basic kind in the material list, because each node material has an *N* before its name. Open a nodes window now and you will see the setup in Figure 1.6. What you're looking at are two nodes: an Input material node and an Output node. The Output node represents the final appearance that Material will take on. At present, the Input node is not linked to any material, as indicated by the red Add New button. Click Add New to create a new material node. The node's header bar will now read MatNode.

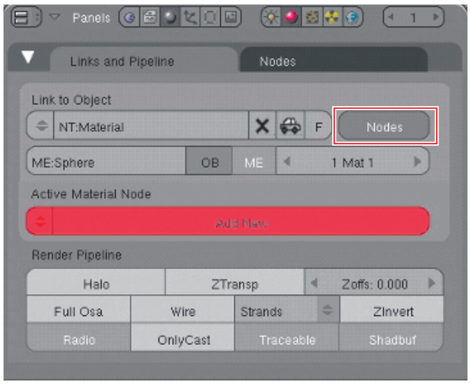


Figure 1.5 Activating Nodes on Material

3. Add a second material node to the node setup. Do this by pressing the spacebar while in the Nodes Editor window and choosing Add > Input > Material, and then selecting Add New from the highlighted drop-down list on the node. The new node will be connected to the previous node by default. Hold the left mouse button (LMB) and drag your mouse across this connection to sever it, as shown in Figure 1.7. Rename MatNode and MatNode.001 to **Rock** and **Magma**, respectively, and rename the main node material from Material to **Lava**.

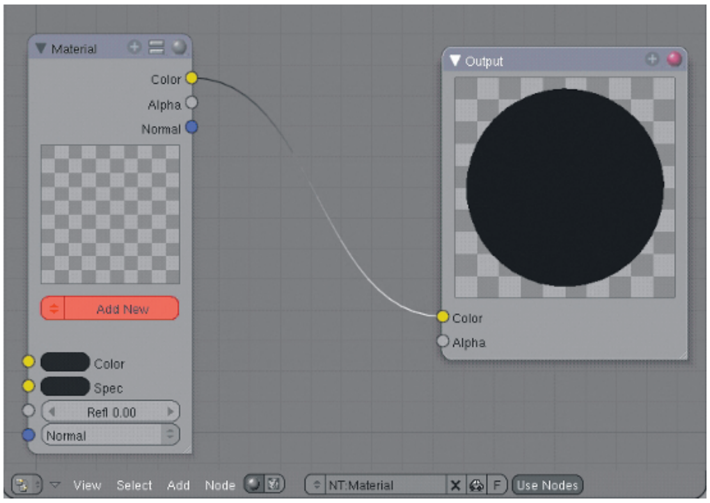


Figure 1.6 Nodes window

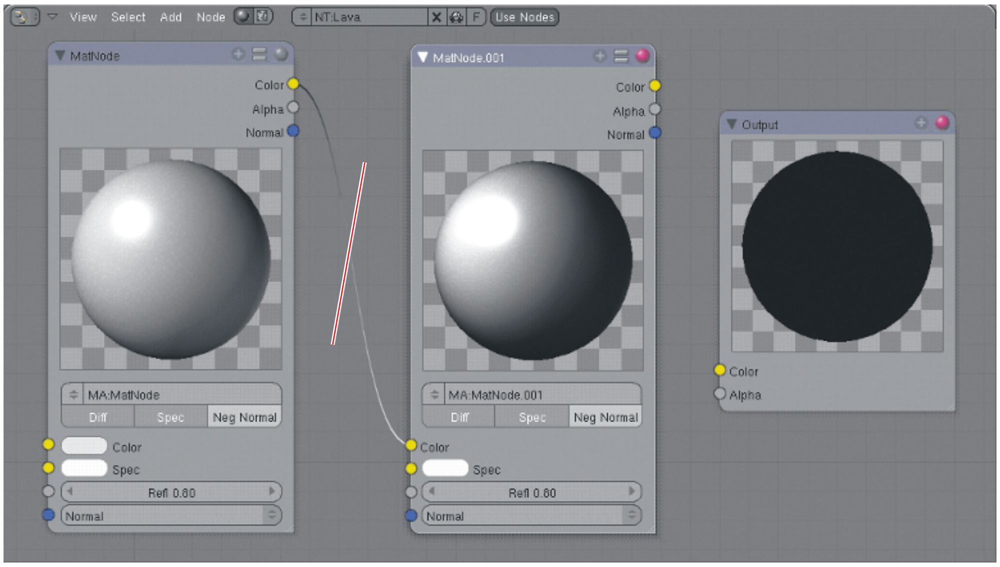
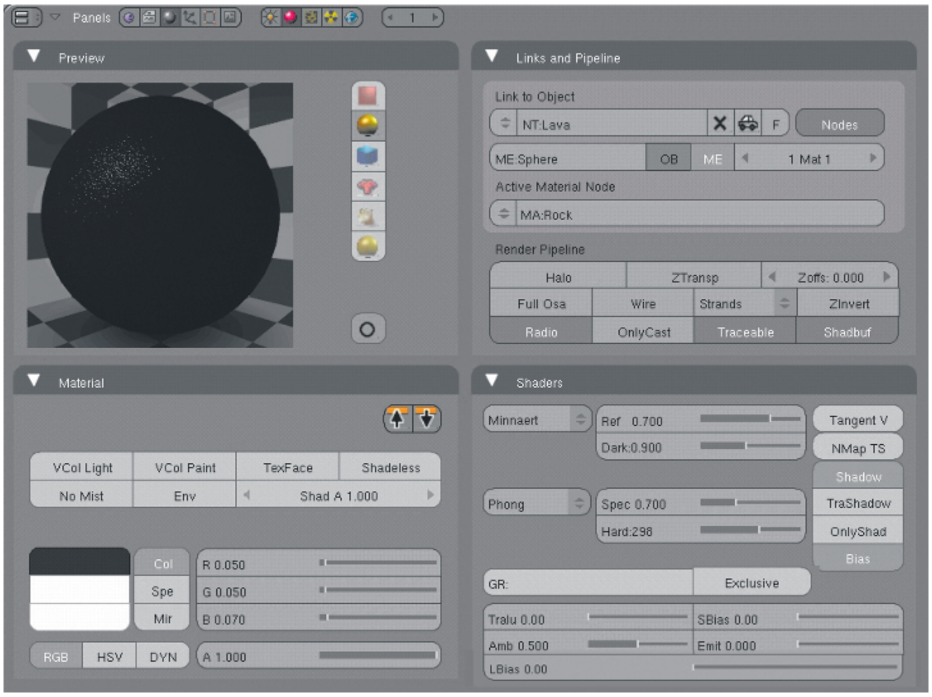


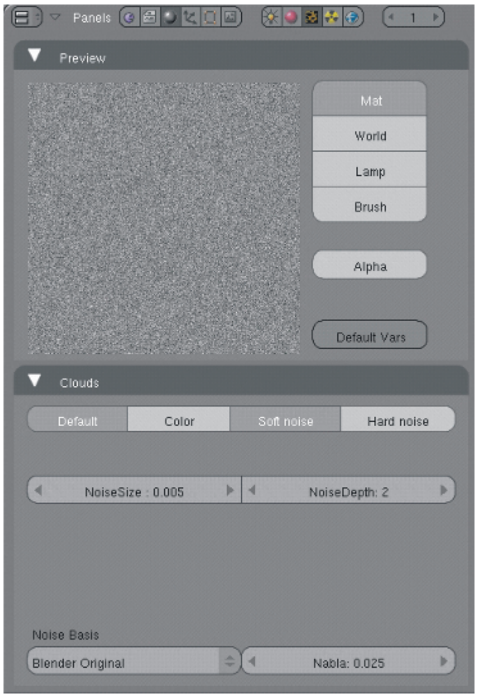
Figure 1.7 Two new input materials

4. Select the Rock material node and set its material values by using the Material buttons as you would for any ordinary non-node material, as shown in Figure 1.8. Note that when you select an input material node in the Nodes Editor, the settings for that material appear in the Material buttons window. Add a Clouds texture set at a very small noise size, as shown in Figure 1.9, and map the texture to Nor on the Map To tab as a bump map to give roughness to the surface of the rock. Then select the Magma material node and edit the material settings as shown in Figure 1.10. Note the high Emit value for the glow. For the orange color texture, use a soft Clouds texture with default parameter settings.



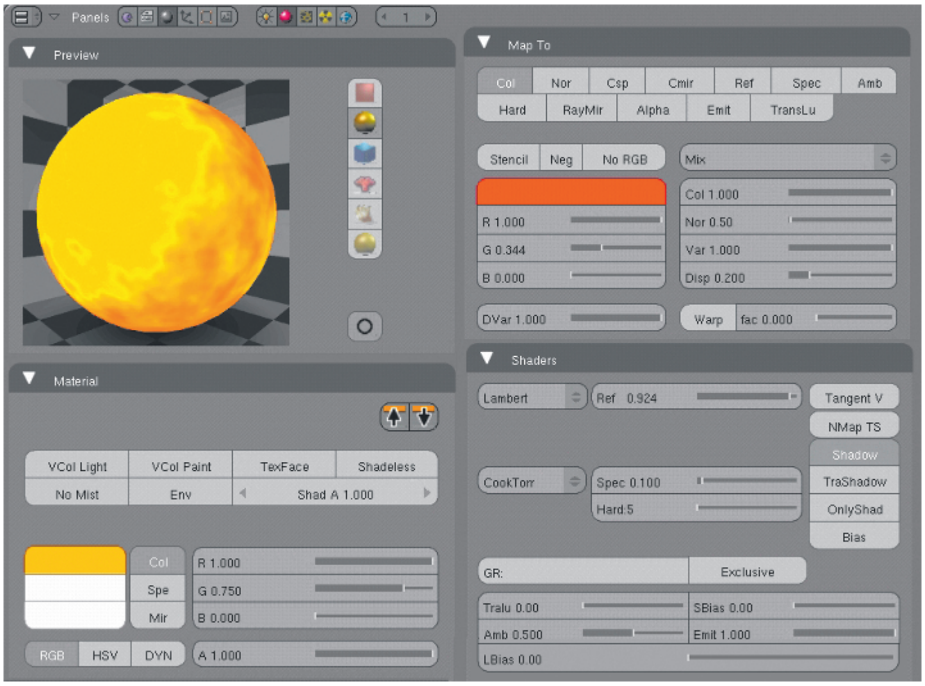
Link to Object: NT:Lava
ME:Sphere — 1 Mat 1
Active Material Node: MA:Rock
Render Pipeline: Zoffs: 0.000
Shad A 1.000
Col R 0.050
Spe G 0.050
Mir B 0.070
A 1.000
Minnaert — Ref 0.700, Dark:0.900
Phong — Spec 0.700, Hard:298
GR:
Tralu 0.00, SBias 0.00
Amb 0.500, Emit 0.000
LBias 0.00

Figure 1.8 Material settings for Rock



NoiseSize : 0.005
NoiseDepth: 2
Noise Basis: Blender Original
Nabla: 0.025

Figure 1.9 Roughness texture for the rock



Shad A 1.000
R 1.000
G 0.750
B 0.000
A 1.000
Mix
R 1.000
G 0.344
B 0.000
Col 1.000
Nor 0.50
Var 1.000
Disp 0.200
DVar 1.000
Warp fac 0.000
Lambert Ref 0.924
CookTorr Spec 0.100
Hard:5
Tralu 0.00
SBias 0.00
Amb 0.500
Emit 1.000
LBias 0.00

Figure 1.10 Material settings for Magma

5. You now have the two basic materials completed in your node setup. What remains is to mix them. To do this, add a Mix node by pressing the spacebar while in the Nodes Editor and choosing Add > Color > Mix. Create connections between nodes by LMB-dragging your mouse from one node's socket to another socket. To mix the two materials, draw a connection between Rock and the Color1 socket on the Mix node, and a connection between Magma and the Color2 socket. Draw a connection between the Color sockets on the Mix node and the Output node, as shown in Figure 1.11. As you can see, the output material is now a mixture of the two input materials. The mix factor of the Mix node is 0.50, which means that the two materials are mixed at a uniform level of 50/50 across the entire object. It is necessary to input a mix factor that will give a more-appropriate mix effect.

6. Add a new Input node in the same way you did for the previous two material nodes, except this time select Texture instead of Material for the node type. In the drop-down menu on the node, select Add New to create a new texture, and name the texture **Mix**. Make this texture a hard Clouds texture with the values set as shown in Figure 1.12. This texture will provide a pattern of black-and-white (that is, 0.0 and 1.0) to replace the uniform 0.50 that's the default for the Mix node.

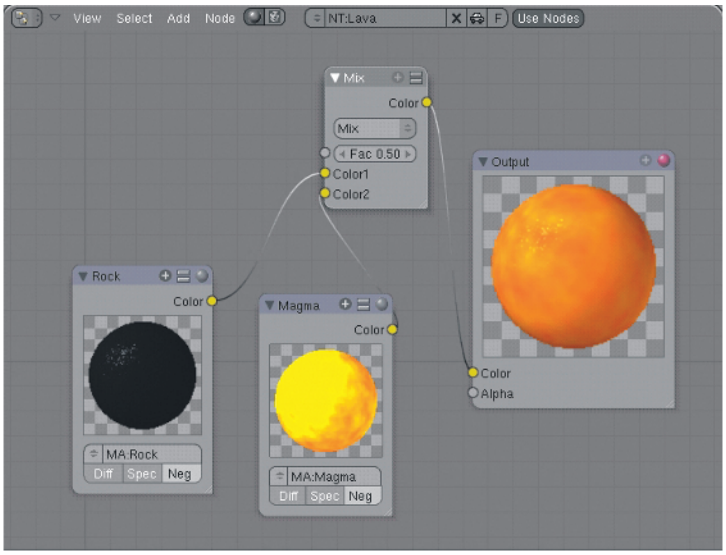


Figure 1.11 Mixing the materials

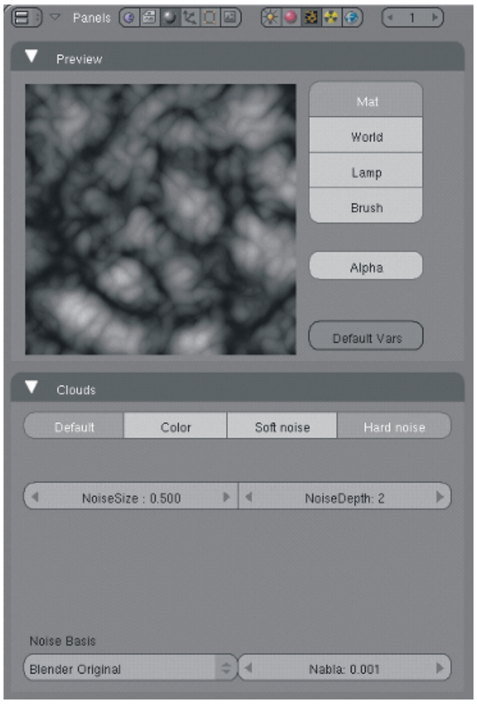


Figure 1.12 A hard Clouds texture for Mix

7. Add an RGB Curves node by pressing the spacebar and choosing Add > Color > RGB Curves. This will enable you to tweak the contrast and brightness of the factor value going into the Mix node. Connect the Mix node to the RGB Curves node, and the RGB Curves node to the Fac connection point on the Mix node, as shown in Figure 1.13. All that remains is to tweak the RGB curve to yield the mix pattern that looks best. To get the correct black-and-white pattern, reverse the direction of the RGB curve so that it goes from upper left to lower right, and then edit the shape of the curve itself. The more gentle the curve, the smoother the transition between the two materials will be. Points can be added on the curve simply by clicking the LMB, and deleted by clicking the button on the node itself marked with an *x*.

8. Finally, add a Displace modifier to the object with the settings shown in Figure 1.14. Type **Mix** in the Texture field to use the texture you just created to determine the displacement. After you've done this, set up your lights and take a render. The finished material should look something like Figure 1.15.

If you have trouble getting the object lit the way you see it here, check the file lava.blend on the CD for the book to see how I've done the lighting.

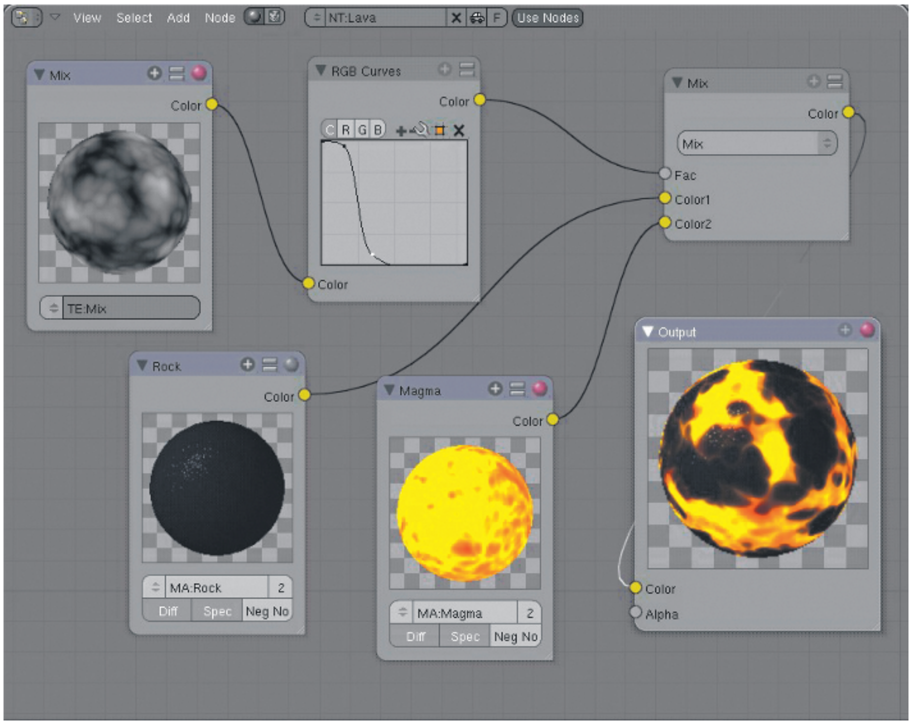
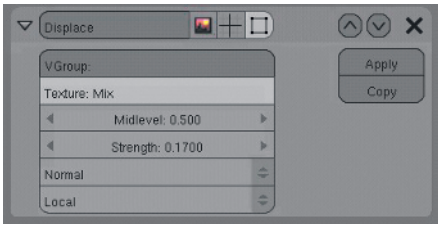


Figure 1.13 Finished node setup for lava



Displace

VGroup:

Texture: Mix

Midlevel: 0.500

Strength: 0.1700

Normal

Local

Figure 1.14 Displace modifier

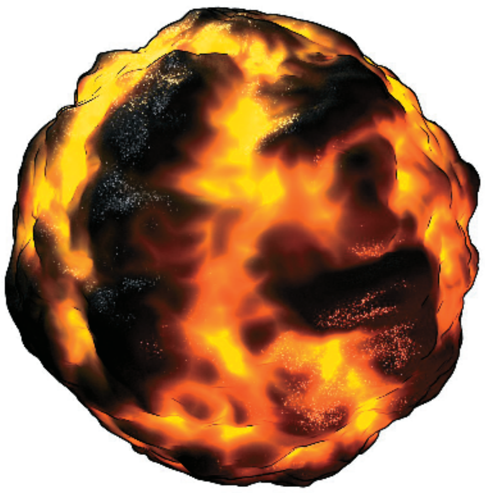


Figure 1.15 Finished render

As you can see, being able to mix different materials and textures in this way is a powerful way to create complex materials that more faithfully represent the complexity of nature.

Transparency and Subsurface Scattering

In Blender, there are three kinds of transparency: Z transparency, ray transparency, and the environment (Env) material setting. In fact, only the first two are true transparencies in terms of the Blender 3D environment. The Env setting creates an area of alpha 0 transparency in the rendered image itself and is used primarily for compositing. In Figure 1.16, you can see an example of the three types of transparency together. The knight piece on the left has the Env option selected, as shown in Figure 1.17, and the area defined by its outline is alpha 0, with the world background visible in the render. This material does not have any transparency option selected, and its true material alpha value is 1. The Env option is not gradated in any way; if it's on, the object and everything behind it will be cut out of the final render. Note that things that are in front of the object are not cut out, such as the middle horse's nose in the image. This option is useful for compositing.

The middle knight uses Z transparency, with the settings shown in Figure 1.18. Z transparency takes two values into consideration: the material's alpha value and the Z values of the points in the scene. The alpha value of a material is a measure of how opaque the material is. A 0 (zero) alpha means the material is completely invisible; a 1 alpha means that the material is entirely opaque. The Z value is the distance of a point in space along the camera's local Z axis; that is, the distance from the camera. Z transparency works very much like layer transparency in a 2D image-manipulation program, using the Z value to determine the ordering of the layers. In the image, the middle knight's alpha value is set to 0.2.

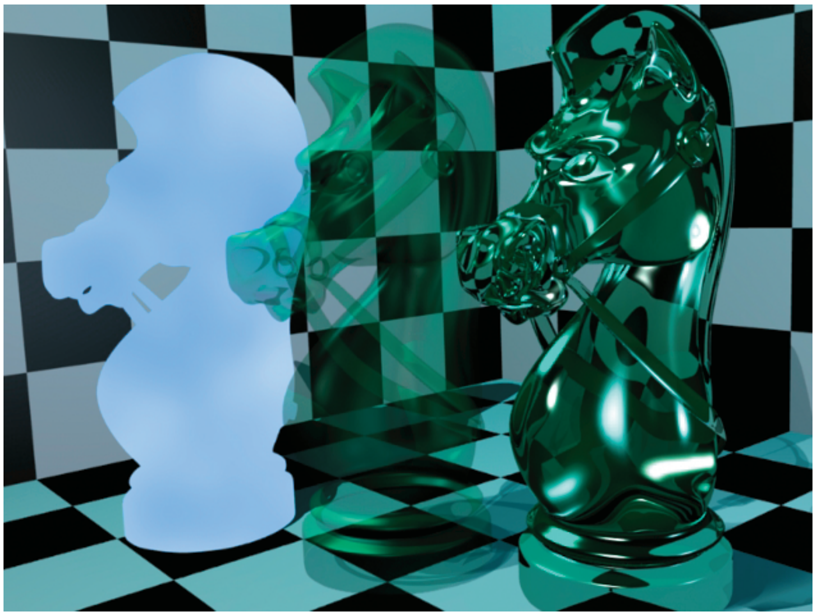


Figure 1.16 The knight piece on the left has the material setting Env selected. The piece in the middle has ZTransp selected, and the one on the right has RayTransp selected.

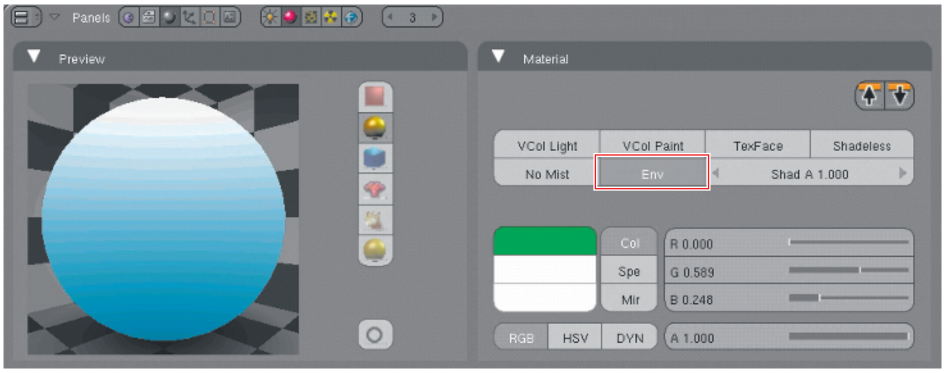
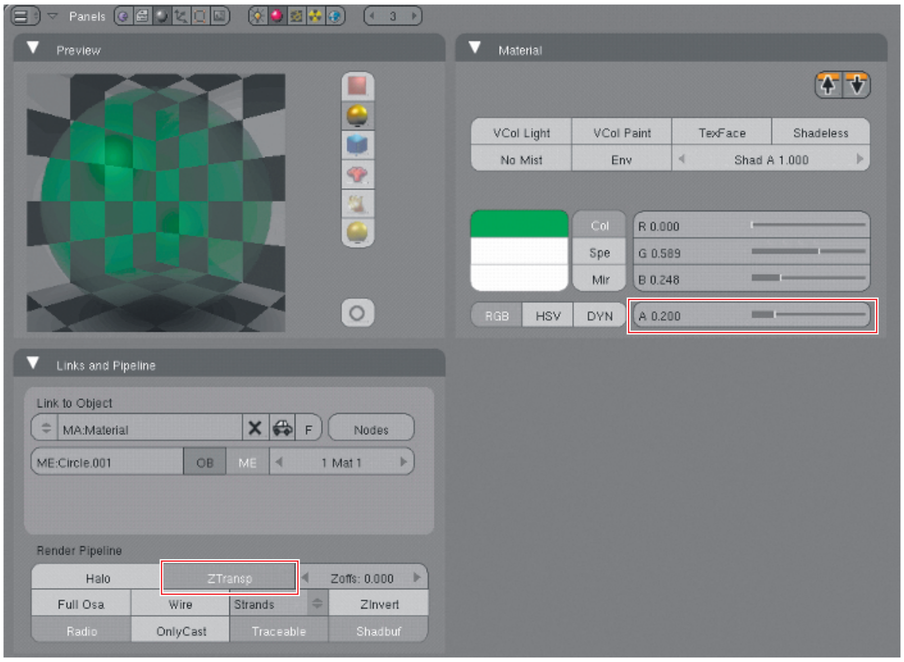


Figure 1.17 The Env option

The knight on the right also has an alpha value of 0.2, but with ray transparency, with values shown in Figure 1.19. As you can see, the effect is very different from the middle knight. Whereas the middle knight looks ghostly, the third knight looks like it is made of a real-world transparent material. The difference, as you can see, is mainly in the way the light from the background is distorted as it passes through the object, as it would be in real life. This distortion depends on the index of refraction (IOR) of the material. Denser materials have higher IORs. Air has an IOR of approximately 1 (1 is the IOR of a perfect vacuum), water has an IOR of about 1.3, glass has an IOR of somewhat higher, between 1.5 and 1.8, and diamond has an IOR of about 2.4.



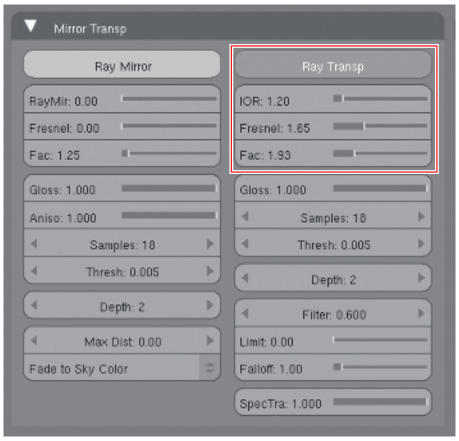
VCol Light | VCol Paint | TexFace | Shadeless
No Mist | Env | Shad A 1.000
Col | R 0.000
Spe | G 0.589
Mir | B 0.248
RGB | HSV | DYN | A 0.200

MA:Material | F | Nodes
ME:Circle.001 | OB | ME | 1 Mat 1

Halo | ZTransp | Zoffs: 0.000
Full Osa | Wire | Strands | ZInvert
Radio | OnlyCast | Traceable | Shadbuf

Figure 1.18 Z transparency values

To represent a material's IOR, it is necessary to use ray tracing to follow the path of the light. Using ray tracing will make render times longer, but in cases like these it is necessary. The Depth value on the Ray Transp panel is set at 2 by default. This means that the ray calculation will pass through two surfaces. If there are more than two transparent surfaces for the light to pass through, the subsequent surfaces will appear black. To calculate ray information passing through more surfaces than this, you must raise the Depth value. Again, this will increase render times, so it should be done only when necessary.



Ray Mirror | Ray Transp
RayMir: 0.00 | IOR: 1.20
Fresnel: 0.00 | Fresnel: 1.65
Fac: 1.25 | Fac: 1.93
Gloss: 1.000 | Gloss: 1.000
Aniso: 1.000 | Samples: 18
Samples: 18 | Thresh: 0.005
Thresh: 0.005 | Depth: 2
Depth: 2 | Filter: 0.600
Max Dist: 0.00 | Limit: 0.00
Fade to Sky Color | Falloff: 1.00
SpecTra: 1.000

Figure 1.19
Ray Transp values

Transparent Shadows

Ray shadows cast by transparent objects are transparent. However, in order for them to appear as they should, the material on which they are cast needs to be set to take transparent shadows. This setting is found on the shaders tab, as shown in Figure 1.20. You can see how this affects the shadow that falls on the checkerboard material in Figure 1.21.

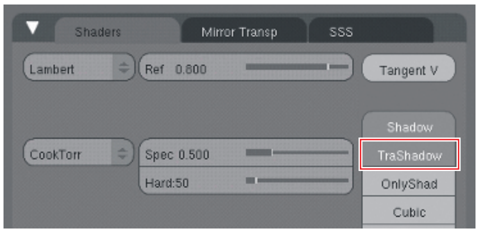


Figure 1.20 TraShadow option for receiving transparent shadows

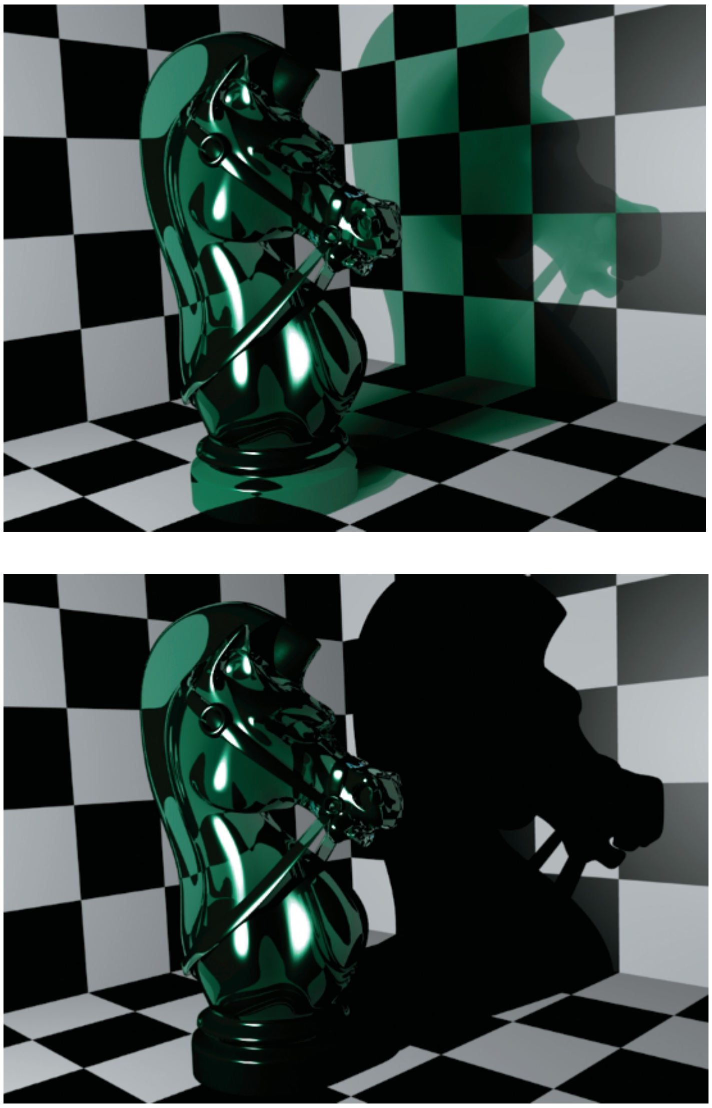


Figure 1.21 In the first image, the black- and- white checkerboard materials are set to take transparent shadows; in the second, they are not.

If you want to avoid ray tracing, transparent shadows can also be produced by using a spotlight with the Irregular shadow buffer type and setting a <1 value for Shad A in the shadowing object's Material tab. In upcoming versions of Blender, the new Deep Shadow Maps functionality coded by Joe Eagar as part of the Google Summer of Code will offer significant speed and quality benefits, as well as transparence, to non-raytraced shadows.

Subsurface Scattering

When light strikes an object, it can be reflected in a variety of ways. Mostly, the way it reflects depends on qualities of the surface of the object. In Blender, the diffuse and specular shader settings determine how a material reflects light (other settings, such as ray mirroring, may also play a part). With some materials, however, calculating how light behaves on the surface only is not enough. For materials such as wax, skin, most vegetation, jade, milk, and many others, a very small amount of surface translucency influences the way light reflects. Although most of the light reflects back from the surface, some of the light penetrates the surface of the material and is reflected back through the material in a scattered state, diffusing some of the light. Although this effect is subtle, it is present on many of the things that humans are best at recognizing, such as food, animals, and other people. For this reason, a lack of subsurface scattering can be a dead giveaway that an image is CG. When used well, it can enable you to achieve extremely realistic organic materials, as in Enrico Cerica's image in Figure 1.22.

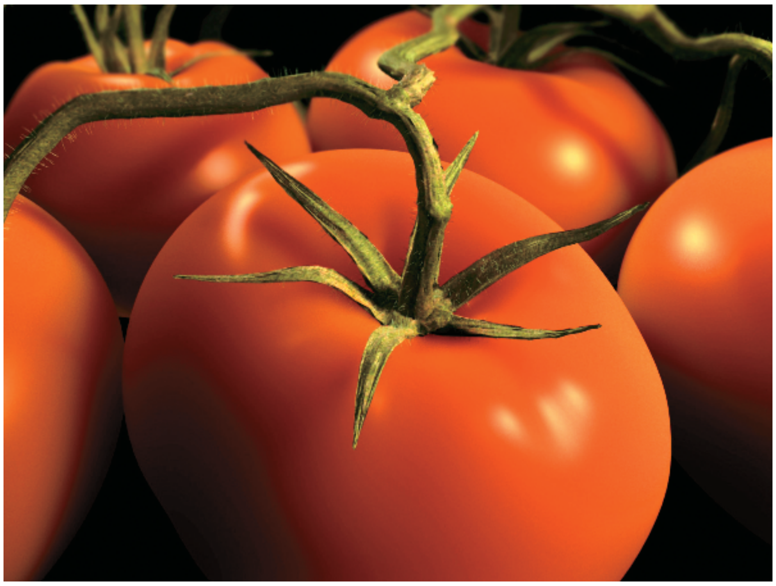


Figure 1.22 Enrico Cerice used subsurface scattering to achieve photorealism in this image.

To enable subsurface scattering, click the Subsurface Scattering button on the Subsurface Scattering (SSS) tab in the materials buttons area, shown in Figure 1.23. There are a handful of preset materials: skin (two different ones), marble, whole milk, skim milk, potato, ketchup, cream, apple, and chicken. (I guess the developer who set up the presets was eager to get to lunch!) You can select one of these and then modify the values to create a Custom setup.

The settings for subsurface scattering are as follows:

Scale indicates the scale of the objects that the material is applied to. It is used to calculate the size of the blurring radius for the material. To calculate the scale you should use, divide 1 by the number of millimeters you want a single Blender unit (BU) to represent. If you want a Blender unit (BU) to represent a meter, the Scale value should be 1/1000 = 0.001. If you want a BU t to represent 2 centimeters, the scale is 1/20 = 0.05. If a BU is a millimeter, the scale is 1.0. It is important to pay attention to this, because correct scale is the most important factor in getting your materials with SSS to look realistic.

Radius R, G, and B are the blurring radii for Red, Green, and Blue. These values determine the distance that different colored light scatters under the surface of the material. If you've ever shone a flashlight through your hand in the dark, you have seen that the red in your hand scatters more than the other colors.

IOR is the index of refraction, or the degree to which light is bent when it travels through the material. As mentioned in the previous section, higher IOR values indicate higher density. In the case of subsurface scattering, the effect is subtle, because no actual rays pass through the material. In most cases, you can get away with setting the IOR to approximately the value of water, 1.3.

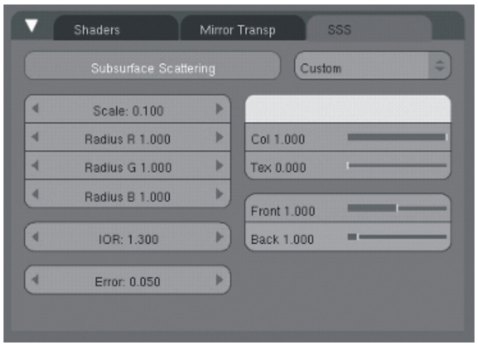
Error controls the precision of sampling. Higher error values will speed up rendering but can result in visible artifacts. You can set it as low as 0.02.

Col determines how much the base color of the material is affected by the diffuse color set in the color picker above the Col field.

Tex controls the degree to which textures on the material are also blurred. 0 will leave the textures unaffected by the scattering.

Front is the weighting factor for front scattering. Front here means pointing toward the camera. Front scattering shows as diffusing of the light and blurring of shadows on the surface of the material.

Back is the weighting factor for back scattering. Back scattering shows up as light passing through from behind the material.



Scale: 0.100
Radius R 1.000
Radius G 1.000
Radius B 1.000
IOR: 1.300
Error: 0.050
Col 1.000
Tex 0.000
Front 1.000
Back 1.000

Figure 1.23 The SSS panel

Figure 1.24 shows three objects with materials that differ only in their subsurface scattering. The knight on the left has no SSS, the knight in the middle has strong front SSS with weaker back SSS, and the knight on the right has strong back SSS with weak front SSS. Of course, for more-convincing materials, you would also want to adjust the shader values appropriately; wax has a different specularity from jade. This example is mainly to highlight the differences in the subsurface scattering effect itself.

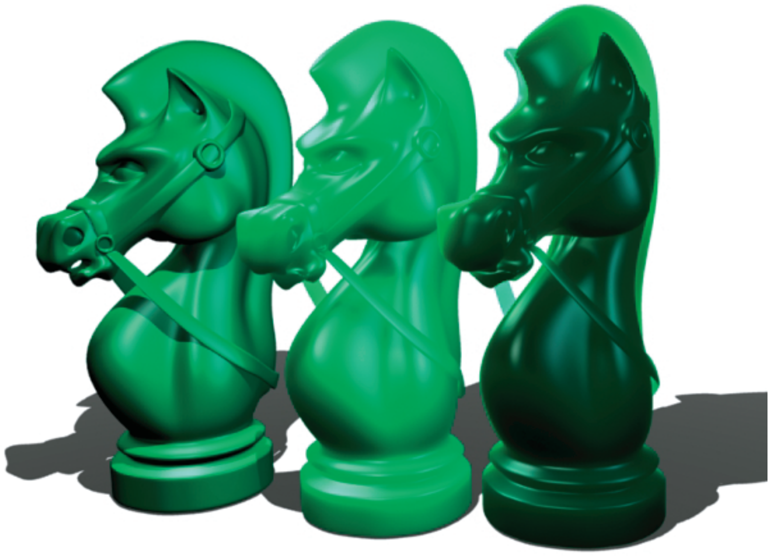


Figure 1.24 Three materials: no SSS, strong front SSS, strong back SSS

The algorithm used in Blender for subsurface scattering was originally presented at SIGGRAPH '02 in the paper "A Rapid Hierarchical Rendering Technique for Translucent Materials" by Henrik Jensen and Juan Buhler. You can find the original paper at http://graphics.ucsd.edu/~henrik/papers/fast_bssrdf.

Sky Maps

If you want to make outdoor scenes, you need to create a sky background. There are a variety of ways to go about modeling clouds and other features of skies. Mostly, though, it is enough for the sky to be a background, and in this case the simplest and most common approach to making skies is to use an image texture called a sky map. I won't discuss how to make sky maps here. You can make them by taking photographs, or by using sky and landscape creation software such as Terragen. Blender user M@dcow has created a fantastic repository of free sky maps at http://blenderartists.org/forum/showthread.php?t=24038 that you can use in any way you like. The sky maps used in the various examples in this book were taken from that resource.

The two sky maps shown in Figure 1.25 are included on the CD accompanying this book as files angmap12.jpg and sky_twilight.jpg. As you can see in the figure, sky maps can come in several forms, which require different mapping methods to accurately fit into a scene. The rectangular sky map shown in the figure is intended to be mapped onto the top half of a sphere. Angular maps appear as highly distorted sphere-shaped reflections, and represent the entire visible background in all directions.

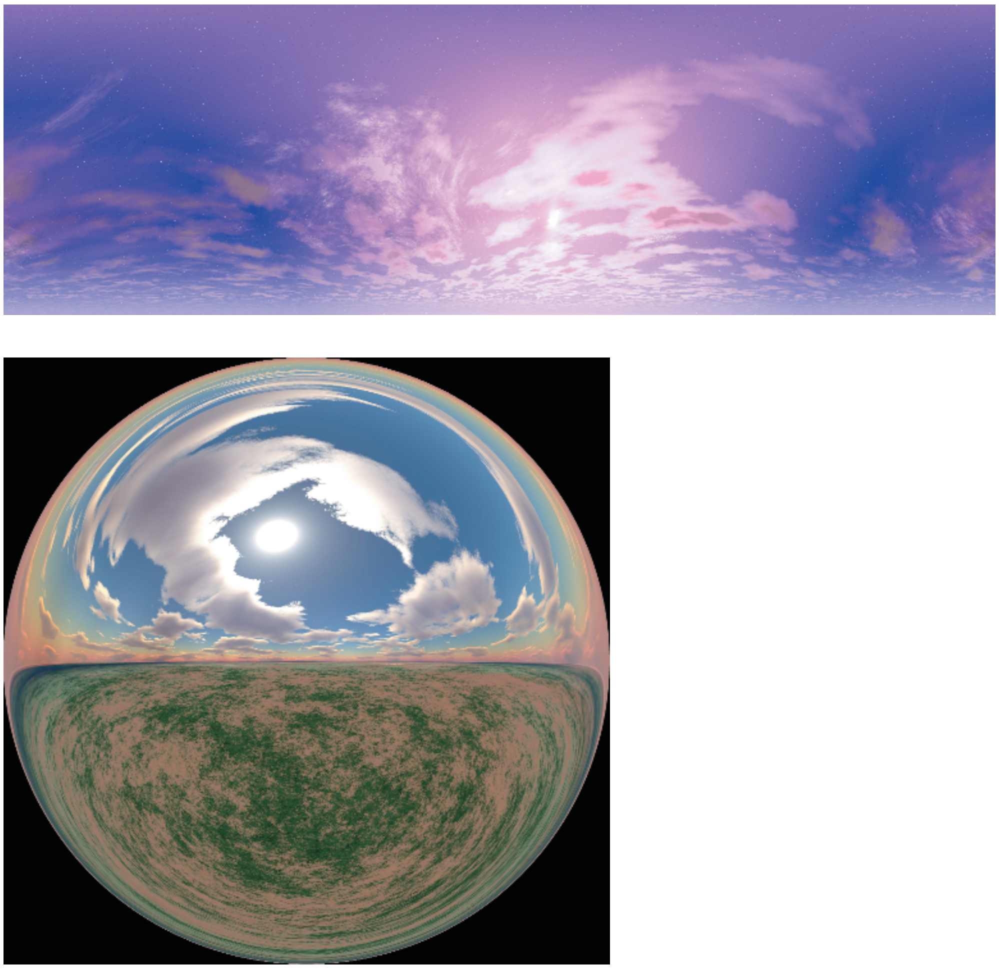
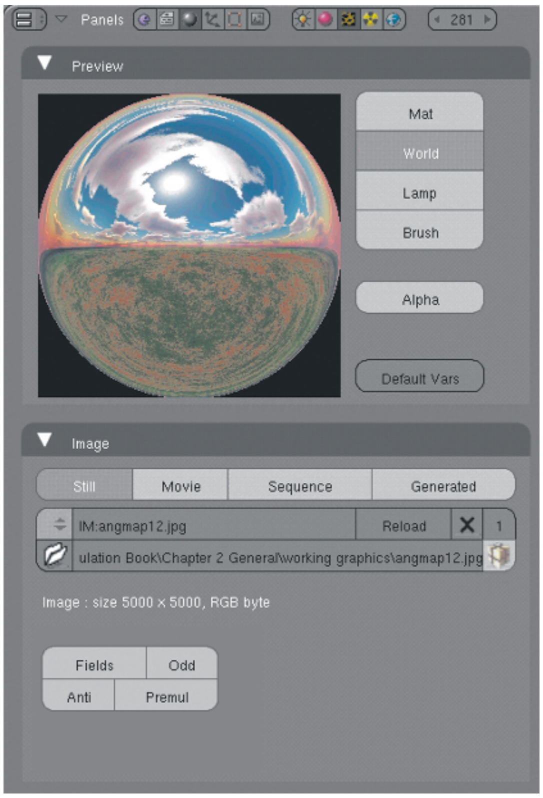


Figure 1.25 A sphere map and an angular map

Using these sky maps is simple. The sky map is added as an image texture to the world in the same way that an image texture is added to a material. The texture buttons panel should look something like Figure 1.26. The world buttons should look like Figure 1.27. Make sure the options Hori, Real, and AngMap are selected. Hori is the option for mapping the image with respect to the horizon. Real means that the actual 3D space horizon is used, as opposed to the center of the camera's current view. Selecting this option ensures that the view of the sky will shift correctly as the camera moves or rotates. AngMap maps the coordinates of this specific type of image to the 3D world. When you use a spherically mapped image, select the Sphere mapping option, as shown in Figure 1.28.

Note that in the case of the Sphere mapped image, only the top half of the background space is mapped with the image. The remainder is the default blue. The spherical mapping describes a hemisphere extending from the horizon to the zenith in the 3D space. Because this map is all sky, it is assumed to end at the horizon. If you use this mapping, you must set up your world to include or conceal the horizon, just as in real life. If you want to have the horizon in view, it may be better to map the sky image onto a mesh dome or tube, so that you have more control over where the horizon is in the camera view. It is also possible to turn the camera's clipping value up very high and extend a plane so that it nearly meets the horizon, and then to match the gap of background color to the color of the plane. The AngMap option is necessary if you want the entire background, ground and all, to be included in the sky map. For example, if you are animating an airplane dogfight, you want your camera to be able to move freely and have an accurately mapped image background visible from any angle. It should be noted that the motion of the "ground" part of the image will not necessarily match up with the motion of real 3D objects on the ground if the camera is moving, making them appear to slide across the terrain, so this is most useful as a backdrop for flying objects. As with everything, remember that different shots may have different setup requirements.



IM:angmap12.jpg

ulation Book\Chapter 2 General\working graphics\angmap12.jpg

Image : size 5000 x 5000, RGB byte

Figure 1.26 Texture buttons for a sky map image texture

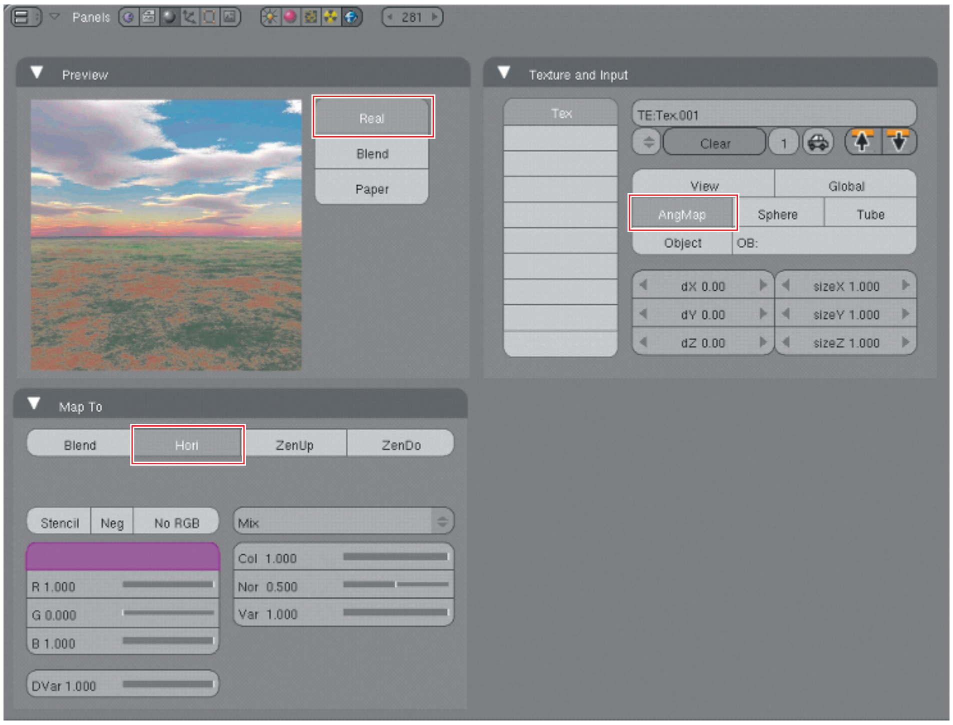


Figure 1.27 World buttons for an AngMap sky map

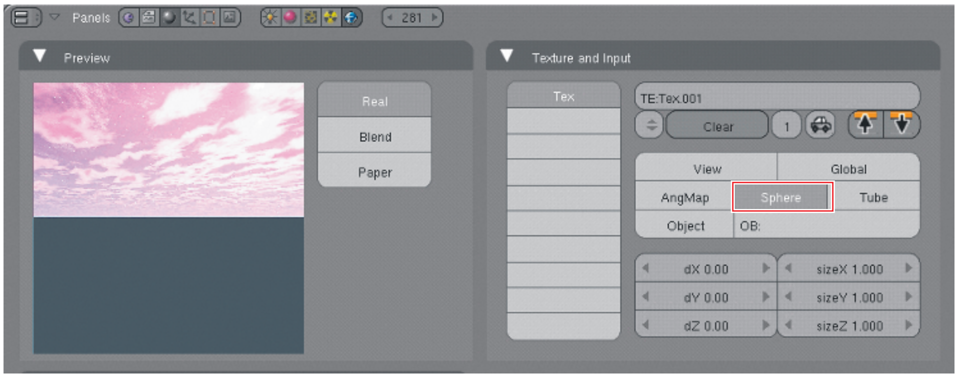


Figure 1.28 World buttons for a sphere sky map

Faking Physics with General Tools

In this section, I describe techniques I've come across for achieving various physical effects without using actual simulations. Some of these techniques are based on methods that can be found in some form or another on personal websites or in threads on BlenderArtists.org, but they are all approaches that I think deserve a wider audience. Not only are the methods themselves useful, but the tools you'll use to set them up are generally applicable.

Robert J. Tiess

The artist who created *Haiku* and *Mount Reckon*, Robert J Tiess has been a Blender user since he discovered the software in 2003, and has gained a reputation in the Blender community for his stunning, thought-provoking images. His lush, richly textured character illustrations were featured in *Introducing Character Animation with Blender.* His work covers a wide variety of styles, and he is constantly exploring new ways to use Blender to the fullest in expressing his own artistic vision. His enthusiasm for Blender comes through clearly in his comments about the software and its community. He writes, "Blender's thriving community, amazing artists, gifted and generous coders, supporters, forum moderators, and documentation writers, all make it so much less a piece of a software and far more of a generally positive and empowering phenomenon that is truly global in scope and as universal in appeal, adaptability, and usability as it gets in the technology world."

Modeling Bodies of Water by Using Modifiers and Textures

In Chapter 5, "Making a Splash with Fluids," you'll see how to work with the Blender fluid simulator. It is a common misconception that this is the best way to represent water and liquids, and in many cases this is not the most direct way to get the effect you want. Other methods for creating fluid effects include displacement textures, as used in Figure 1.29 to excellent effect.

In this section, I show you an approach for representing the behavior of large bodies of water in a way that animates convincingly. I also show a simple way to make the surface of the water interact with an object floating in the water. The methods I present here draw on several methods described in Colin Litster's excellent ocean tutorials, which you can find at his web page, at www.cogfilms.com. The method I present here is somewhat simplified, but it will give you a good sense of the main ideas of the approach. I highly recommend a visit to Colin's website to see the rest of his tutorials.

To create this effect, follow these steps:

1. Delete the default cube and add a plane in the top view (NUM7). Scale the plane to 30. Press the W key, select Subdivide Multi, and subdivide by 70 subdivisions. The result is a subdivided plane, as in Figure 1.30. With the whole mesh selected, click Set Smooth in the Links And Materials tab.

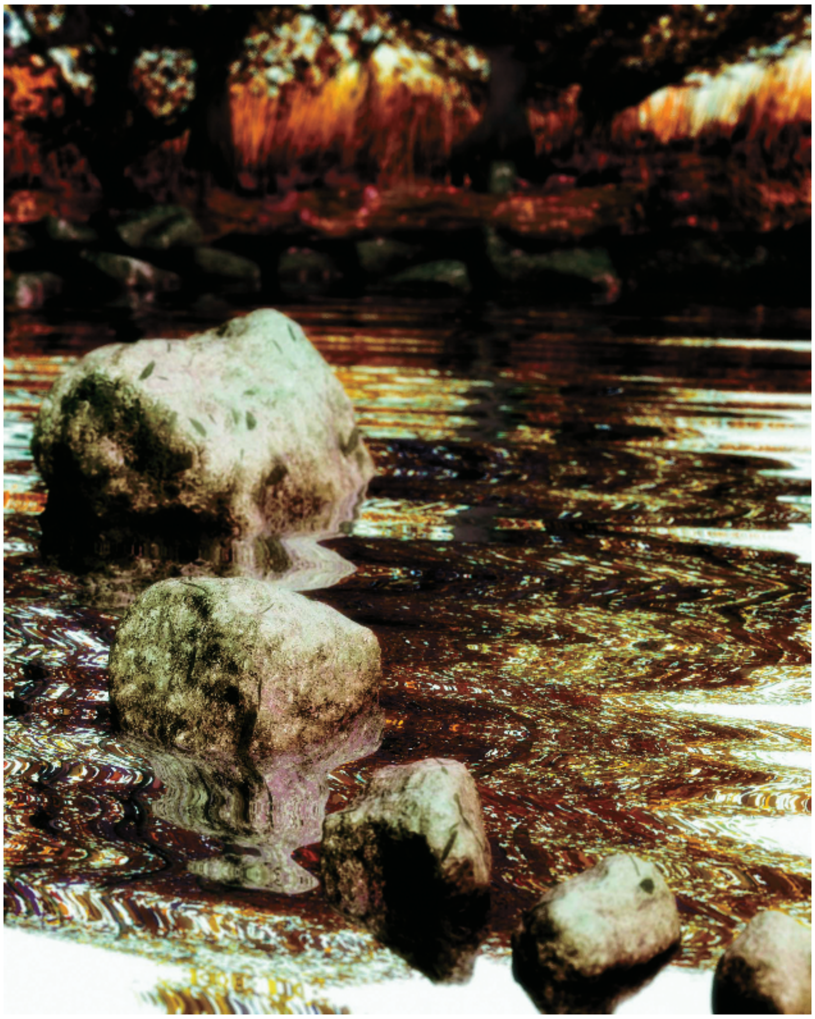


Figure 1.29 *Haiku* by Robert J. Tiess uses displacement textures to model the wave pattern on the surface of water.

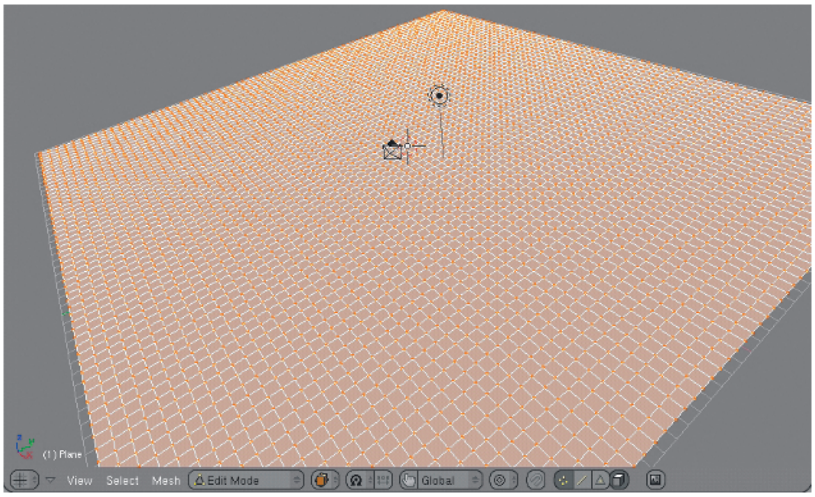
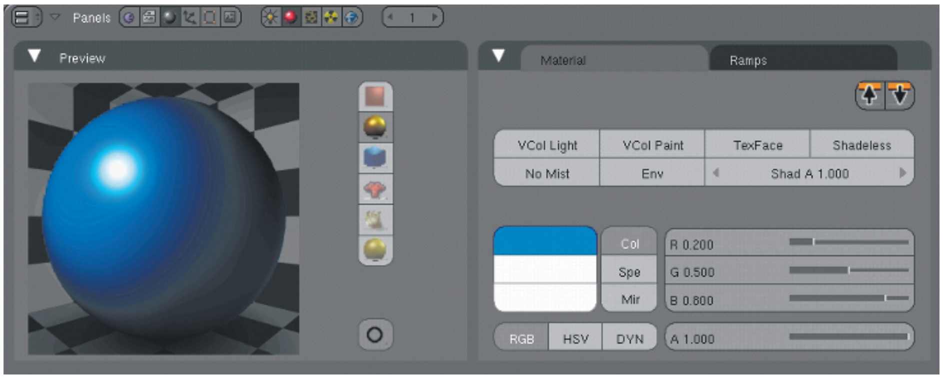


Figure 1.30 A subdivided plane will be the ocean surface.

2. Give the plane a material, as in Figure 1.31. Choose a base color for the water similar to the one shown, and increase the specularity slightly from the default. Set up your lights and camera along the lines shown in Figure 1.32. The camera view should be filled by the water surface. I have one lamp placed off to the left of the camera and one placed over the far corner of the plane in front of the camera. Place your lamps in similar positions. Turn off shadows for the lamps.
3. With the plane selected, add two Wave modifiers, with the settings shown in Figure 1.33. The first wave will move along the Y axis, from one edge of the plane to the other. The other wave will move along the Z normal of the plane, creating a swelling effect. Using the two Wave modifiers together will help make the motion less obviously regular. Note that each modifier can be set to display in render, 3D, and edit mode view. For now set the modifiers to display in render and 3D views. Throughout the tutorial, I've switched the 3D view display on and off. Also, add a Subsurf modifier with the settings of Levels: 2, Render Levels: 3. Make sure that the Subsurf modifier is at the bottom of the modifier stack. The Wave modifiers should be acting on the subsurfaced mesh, not the other way around. I've set the start time for the waves at –200, so that the waves will be in full effect by frame 1.



Shad A 1.000
R 0.200
G 0.500
B 0.800
A 1.000

Figure 1.31 A base material for the water

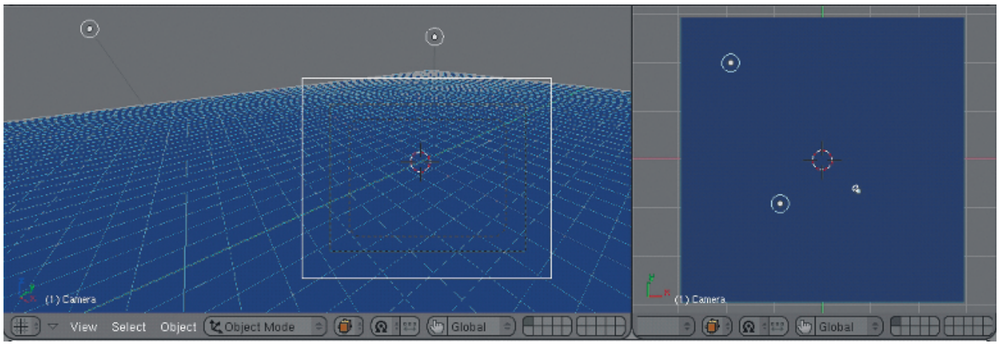
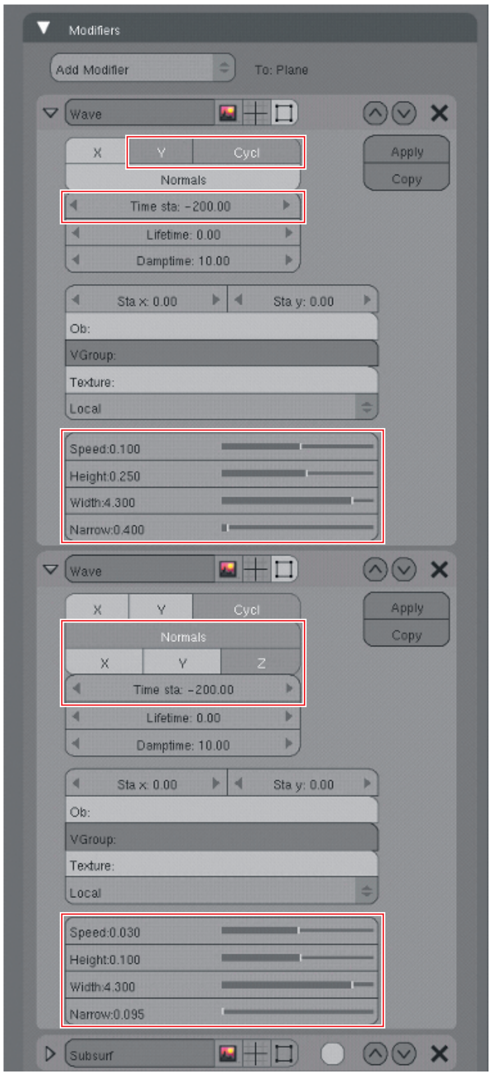


Figure 1.32 Camera and lights setup


Modifiers

Add Modifier · To: Plane

Wave
X · Y · Cycl
Normals
Time sta: -200.00
Lifetime: 0.00
Damptime: 10.00
Sta x: 0.00 · Sta y: 0.00
Ob:
VGroup:
Texture:
Local
Speed:0.100
Height:0.250
Width:4.300
Narrow:0.400

Wave
X · Y · Cycl
Normals
X · Y · Z
Time sta: -200.00
Lifetime: 0.00
Damptime: 10.00
Sta x: 0.00 · Sta y: 0.00
Ob:
VGroup:
Texture:
Local
Speed:0.030
Height:0.100
Width:4.300
Narrow:0.095

Subsurf

Figure 1.33
The modifier stack

4. Everything else to be done for the ocean surface effect from here on involves adding textures and texture effects. First, add an Empty object at the center of the plane. Snap the cursor to the plane by pressing Shift+S and selecting Cursor To Selected. Add the empty by pressing the spacebar and choosing Add > Empty, and then press Alt+R to clear the rotation of the empty (this is not necessary if you add the empty in top view). Create the first texture for your ocean surface material and map it by using the Empty object. Make the texture a soft noise Clouds texture with settings as in Figure 1.34 and name it **Displacement**. Map it to Nor and Disp with the values shown in Figure 1.35, and set the Disp value in the texture to 0.3. This will add one more level of displacement waves to make the movement of the surface even more uneven. Scale the Empty up a factor of 3 by pressing the S key and 3. Run a test render and make sure your surface looks something like Figure 1.36. If not, back up a bit and figure out where yours is different.

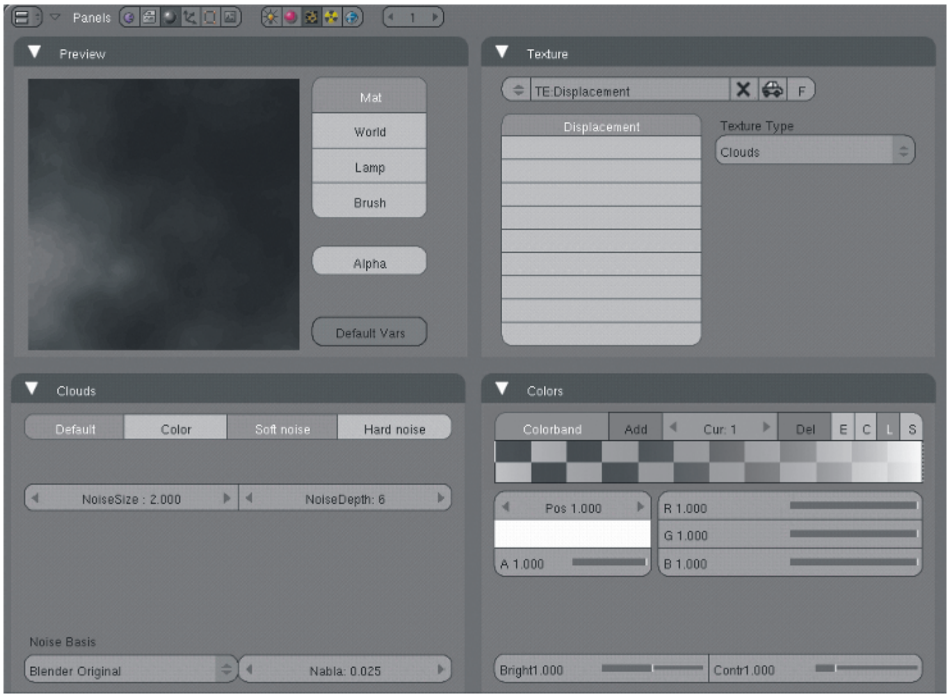

Preview
Texture: TE:Displacement
Texture Type: Clouds
Clouds: Default, Color, Soft noise, Hard noise
NoiseSize : 2.000
NoiseDepth: 6
Noise Basis: Blender Original
Nabla: 0.025
Colors: Colorband, Add, Cur: 1, Del
Pos 1.000
A 1.000
R 1.000
G 1.000
B 1.000
Bright1.000
Contr1.000

Figure 1.34 Displacement texture

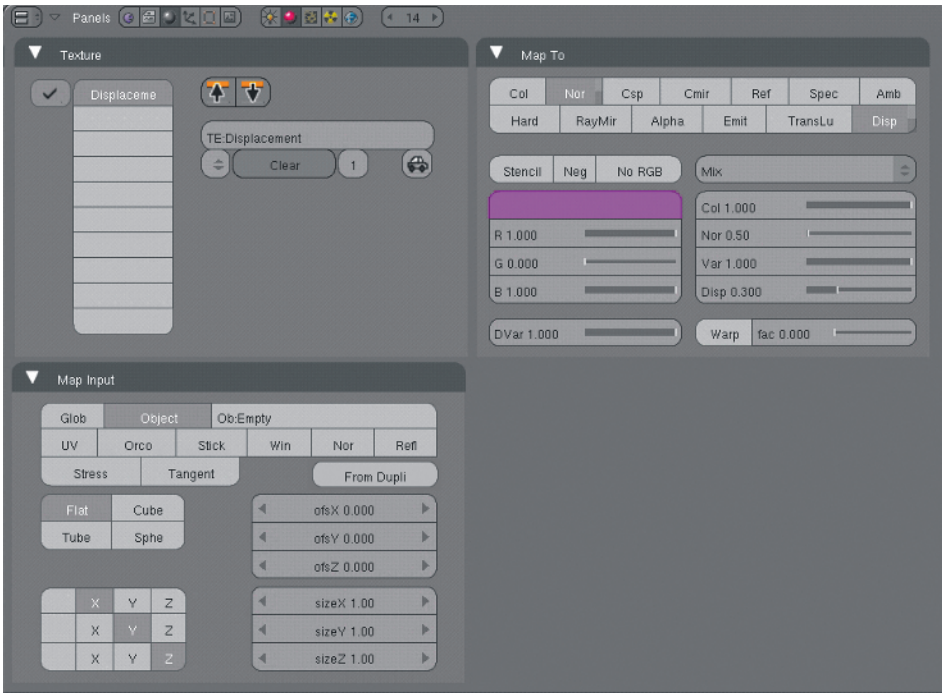

Texture: Displaceme
TE:Displacement
Clear 1
Map Input: Glob, Object, Ob:Empty
UV, Orco, Stick, Win, Nor, Refl
Stress, Tangent, From Dupli
Flat, Cube, Tube, Sphe
ofsX 0.000
ofsY 0.000
ofsZ 0.000
sizeX 1.00
sizeY 1.00
sizeZ 1.00
Map To: Col, Nor, Csp, Cmir, Ref, Spec, Amb, Hard, RayMir, Alpha, Emit, TransLu, Disp
Stencil, Neg, No RGB
Mix
R 1.000
G 0.000
B 1.000
DVar 1.000
Col 1.000
Nor 0.50
Var 1.000
Disp 0.300
Warp fac 0.000

Figure 1.35 Texture mapping settings for the Displacement texture

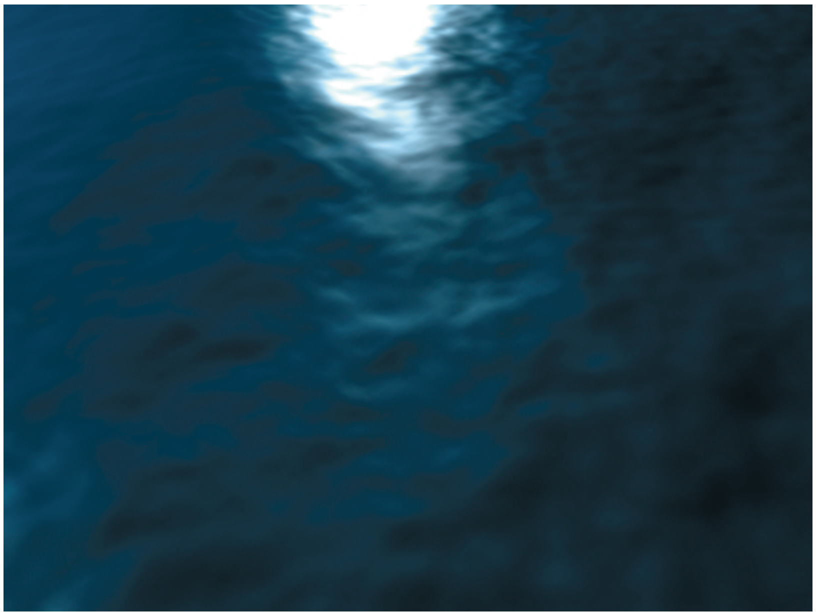


Figure 1.36 Test render

5. These texture-based waves will be animated by keying the movement of the empty. Key two points for the empty and set the empty's Ipos as shown in Figure 1.37. Select the curves, and choose Curve > Extend Mode > Extrapolation in the header menu to make steady slopes. The empty should move along X and Y axes only, and be sure to keep the slopes gradual so that the empty doesn't move too fast.

6. Make a texture to represent the small wavelets on the surface of the water, and call it **Small-waves**. This texture will be a hard noise texture, as shown in Figure 1.38. The mapping values for the texture are shown in Figure 1.39. Note that Size is set to 10.0 for all dimensions, to make these wavelets small. The Map To value is Nor. Change the Nor value in the Map To panel to 3.0. When you take a test render, you should see something like Figure 1.40. Once again, the placement of the lights will make a lot of difference in exactly how your render turns out. Experiment a bit with this.

If you find yourself having difficulty getting things to look as I have them, study the lighting setup in the corresponding file ocean.blend on the CD.

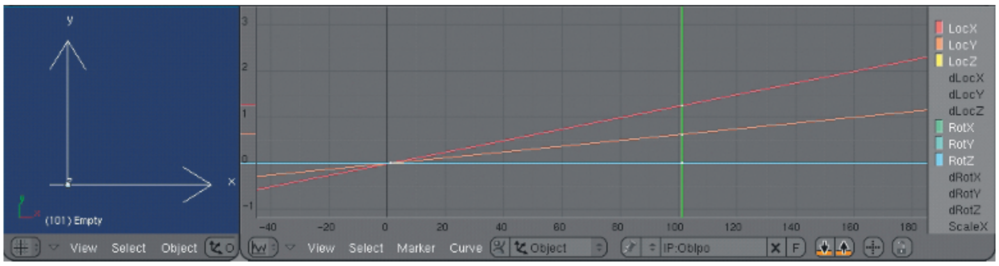
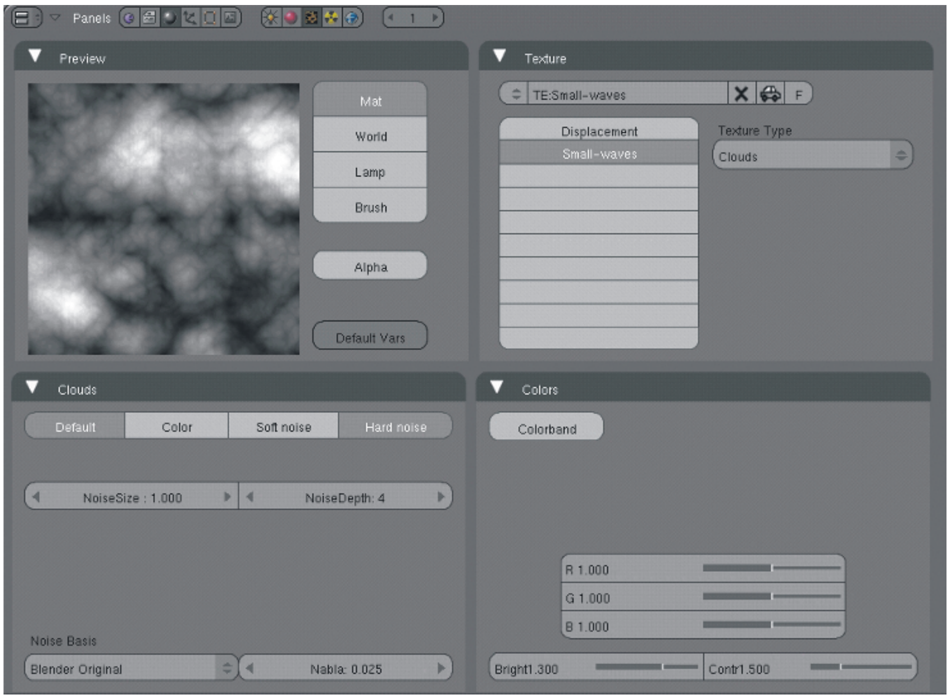


Figure 1.37 Ipos for animating the empty's movement



TE:Small-waves

Texture Type: Clouds

NoiseSize : 1.000

NoiseDepth: 4

Noise Basis: Blender Original

Nabla: 0.025

R 1.000

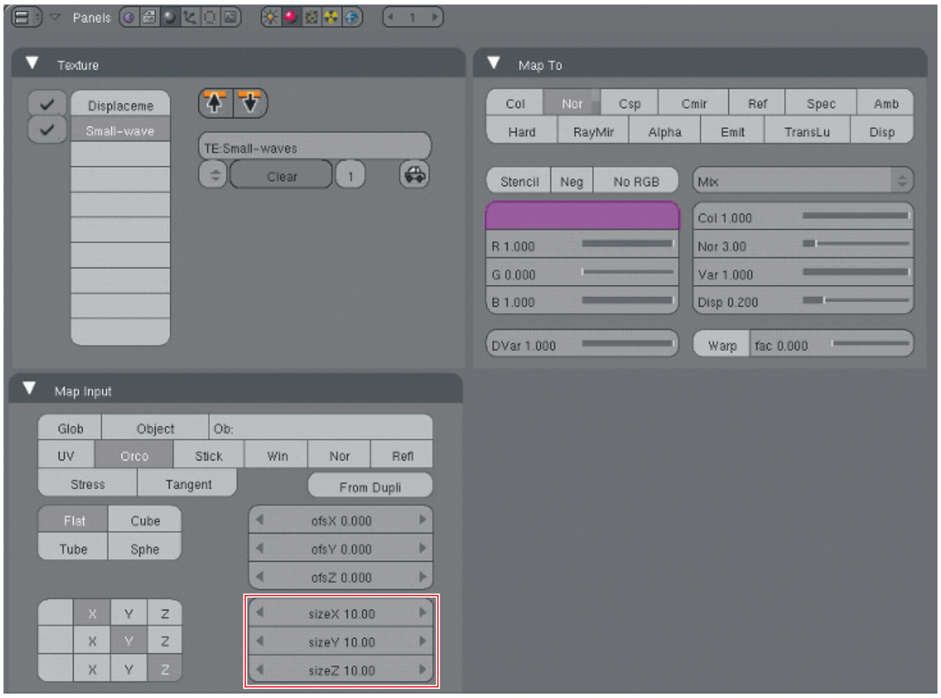
G 1.000

B 1.000

Bright1.300

Contr1.500

Figure 1.38 Small-waves texture



TE:Small-waves

Clear 1

Mix

R 1.000

G 0.000

B 1.000

Col 1.000

Nor 3.00

Var 1.000

Disp 0.200

DVar 1.000

fac 0.000

ofsX 0.000

ofsY 0.000

ofsZ 0.000

sizeX 10.00

sizeY 10.00

sizeZ 10.00

Figure 1.39 Small-waves texture mapping

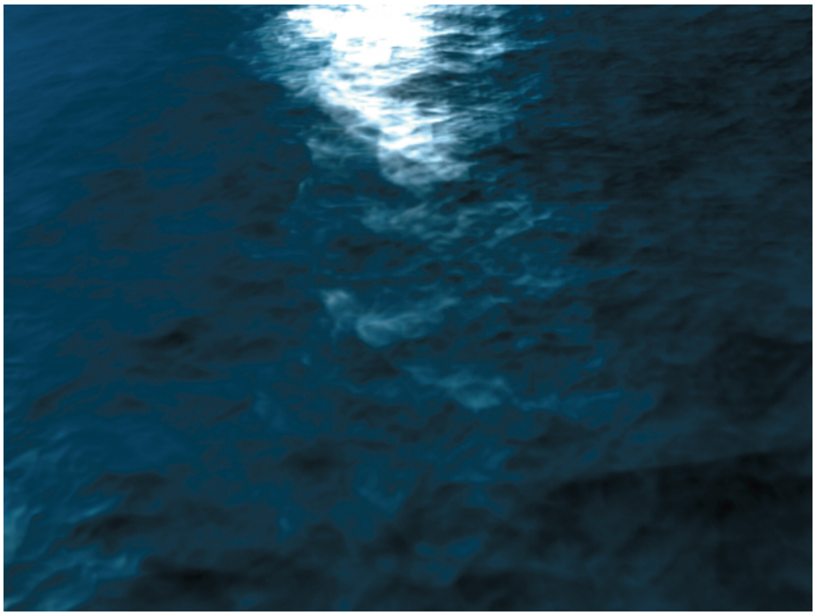
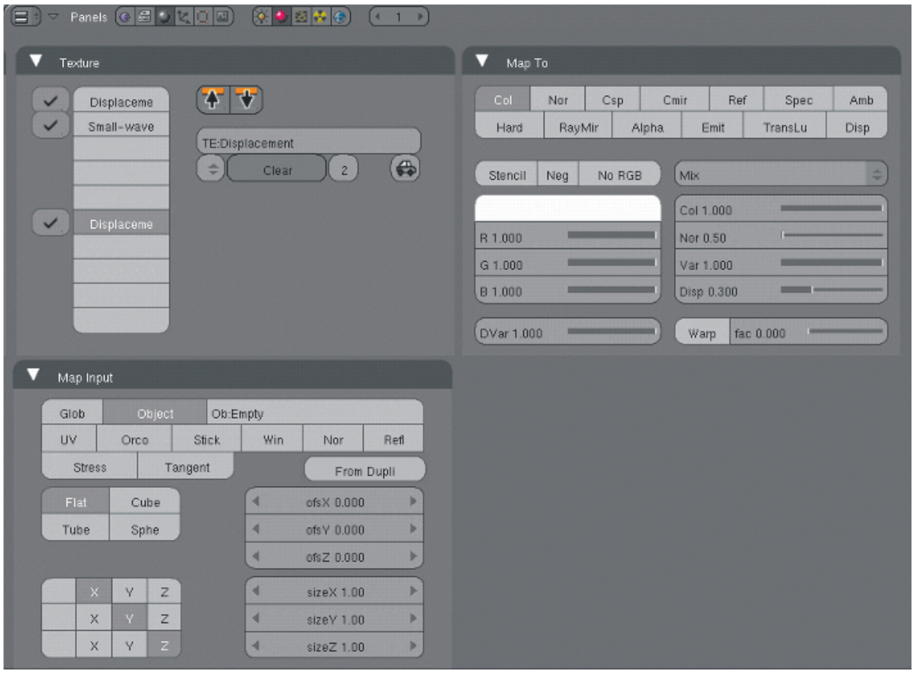


Figure 1.40 Test render

7. For the foam on the waves, use the same texture as you used for the displacement. Do this by selecting channel 0 (the top channel) and pressing the Copy To Clipboard button (the left-hand button of the two highlighted buttons in Figure 1.41). Then select channel 5 (the sixth one down, as shown in the figure) and press the Copy From Clipboard button. Map this texture to color, and make sure the R, G, and B values are turned to white. A test render yields something like Figure 1.42.



TE:Displacement
Mix
R 1.000
G 1.000
B 1.000
DVar 1.000
Col 1.000
Nor 0.50
Var 1.000
Disp 0.300
Warp
fac 0.000
Ob:Empty
ofsX 0.000
ofsY 0.000
ofsZ 0.000
sizeX 1.00
sizeY 1.00
sizeZ 1.00

Figure 1.41 Using the Displacement texture as a color map

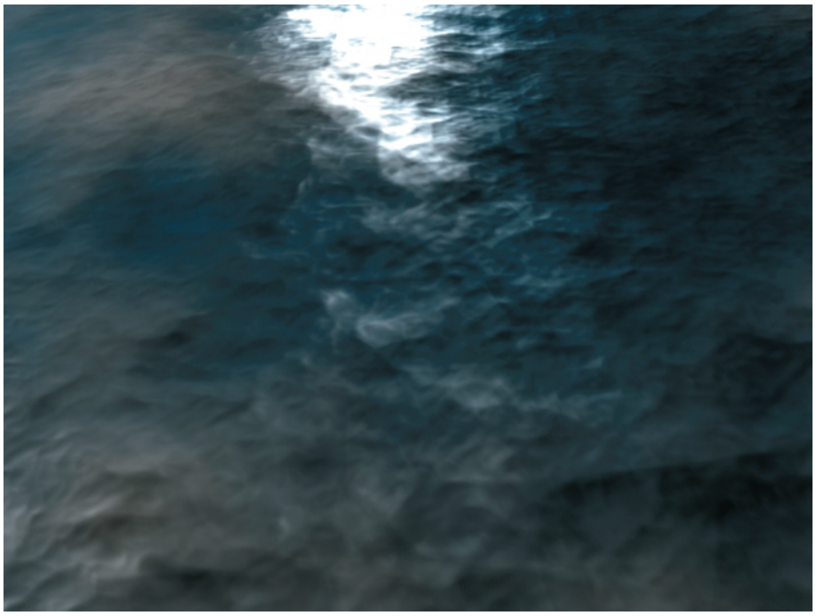


Figure 1.42 Test render

8. Following the same procedure as step 7, copy the Displacement texture from channel 5 to channel 6. With channel 6 selected, go into the texture buttons. This time, you will create a separate texture by pressing the 3 highlighted in Figure 1.43 to make this texture a single- user texture. The 3 represents the number of "users" of the texture, which in this case means the number of channels that the texture is associated with. Clicking this button separates the texture you are working with from the texture associated with the other channels, creating a new, identical texture. You can edit this texture now without affecting the other channels. Rename this texture **Crests** and adjust the brightness, contrast, and colorband as shown in Figure 1.44. Map this with the Empty object also, but select Emit as the Map To value. Set DVar to about 0.359, as shown in Figure 1.45. This texture channel will give a faked translucency to higher points of the textured displacement.

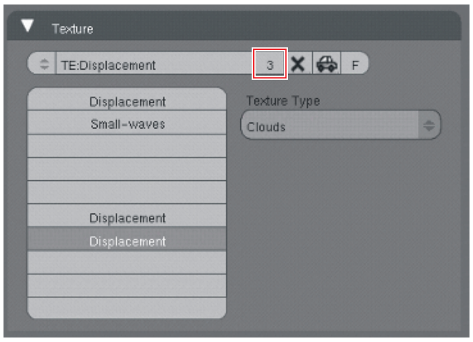
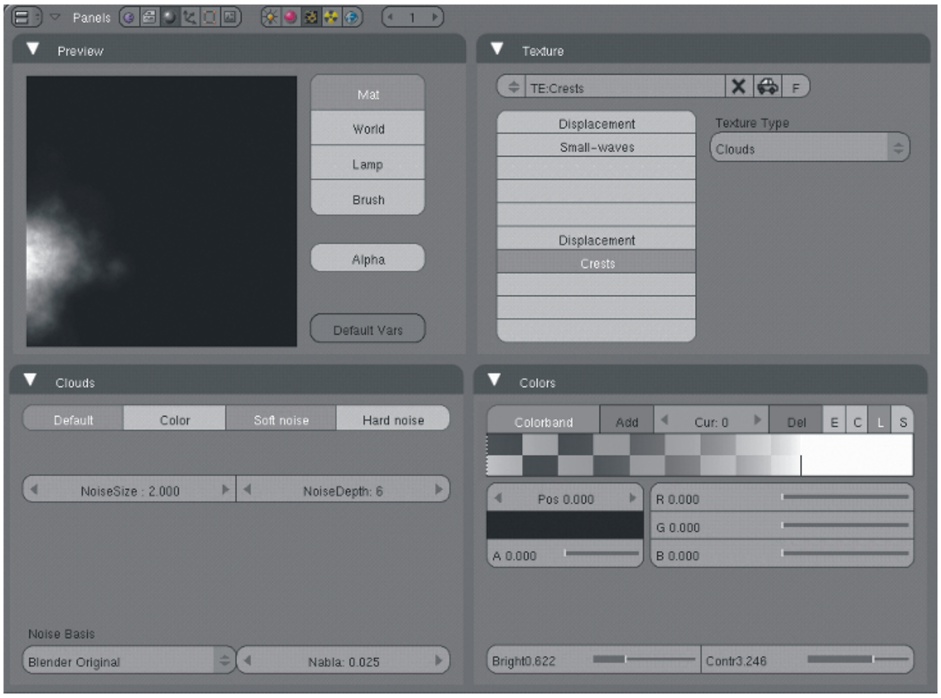
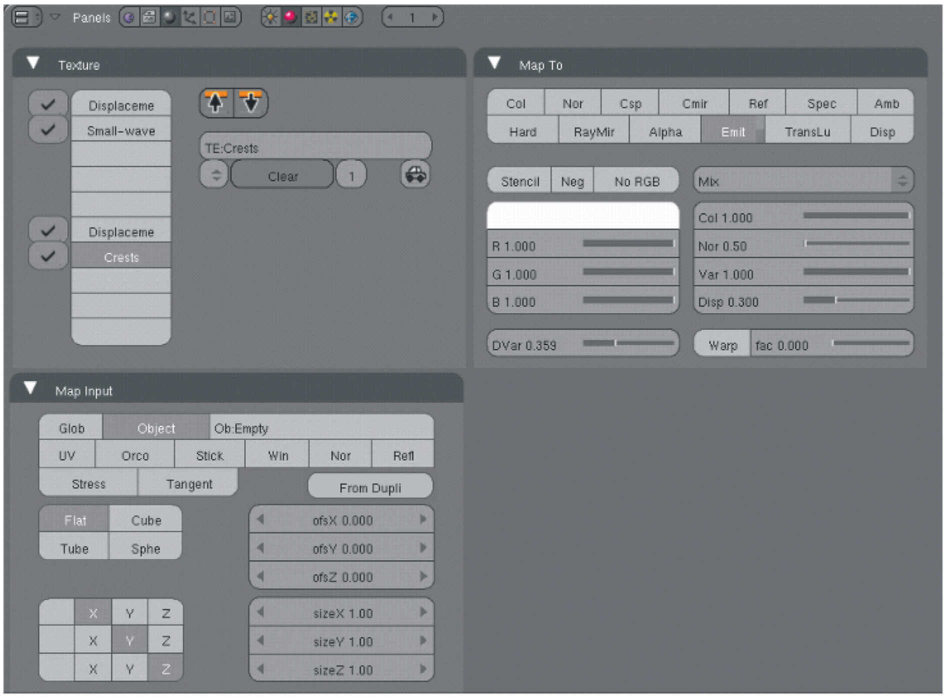


Figure 1.43
Press the button displaying the number of users to make a single-user copy of the texture.



TE:Crests
Texture Type Clouds
NoiseSize : 2.000
NoiseDepth: 6
Noise Basis Blender Original
Nabla: 0.025
Cur: 0
Pos 0.000
A 0.000
R 0.000
G 0.000
B 0.000
Bright0.622
Contr3.246

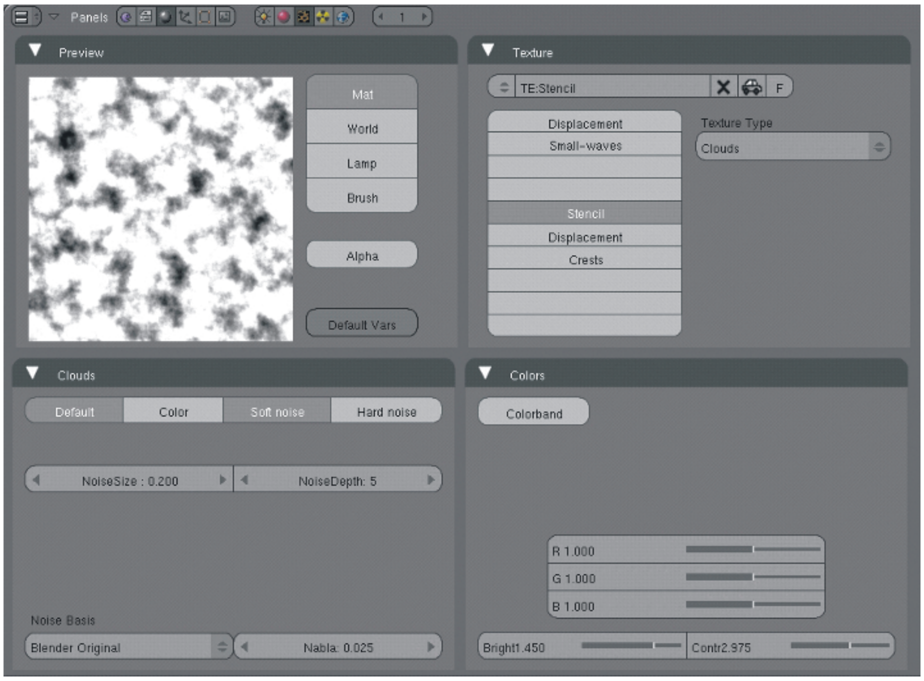
Figure 1.44 Crests texture



TE:Crests
Mix
R 1.000
G 1.000
B 1.000
DVar 0.359
Col 1.000
Nor 0.50
Var 1.000
Disp 0.300
fac 0.000
Ob:Empty
ofsX 0.000
ofsY 0.000
ofsZ 0.000
sizeX 1.00
sizeY 1.00
sizeZ 1.00

Figure 1.45 Crests texture mapping

9. Select channel 4 in the Texture tab of the Material buttons and click Add New to add a new texture. Name this texture **Stencil** and set its values as shown in Figure 1.46. Select Stencil in the Map To tab. This will prevent the foam and crest textures from matching the displacement too perfectly. A test render should look like Figure 1.47.



TE:Stencil
Texture Type: Clouds
NoiseSize : 0.200
NoiseDepth: 5
Noise Basis: Blender Original
Nabla: 0.025
R 1.000
G 1.000
B 1.000
Bright1.450
Contr2.975

Figure 1.46 Stencil texture

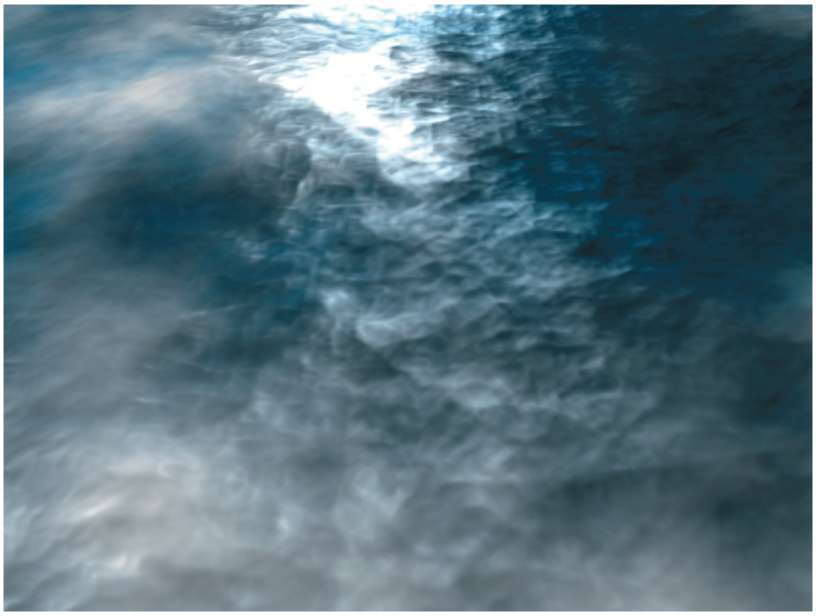


Figure 1.47 Test render

10. Key the movement for the small waves as shown in Figure 1.48. Note that this is a Material Ipo and that the index of the Ipo to the right of the drop-down must match the channel of the texture, counting from top to bottom and beginning at 0. The second channel down, therefore, is channel 1. This wraps up the water itself. If you like, render out a few seconds of animation to see how the movement looks. You might want to do this without oversampling by deselecting the OSA button and at a small size, such as 25 percent, to make it quicker.

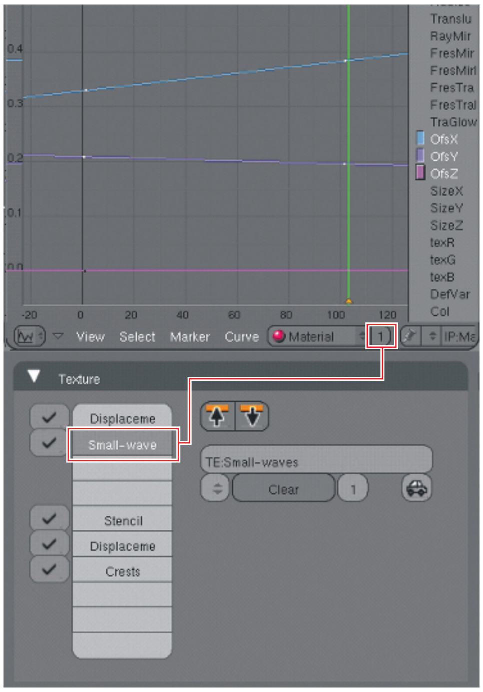


Figure 1.48 Keying the offset for channel 1

11. When you're satisfied with the water, set the modifiers on the plane not to display in the 3D view mode, as shown in Figure 1.49, so that the plane appears perfectly flat. Snap the cursor to the plane again, go into top view, and add a torus by pressing spacebar and choosing Add > Mesh > Torus. The default settings and size are just fine as they are. Add a white and a red material to the object, as shown in Figure 1.50, to make a simple life ring.

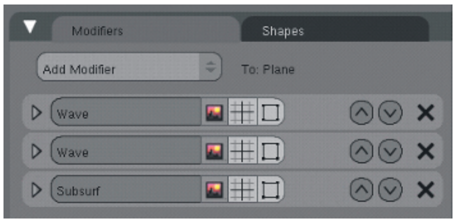


Figure 1.49 Modifiers set to display only when rendered

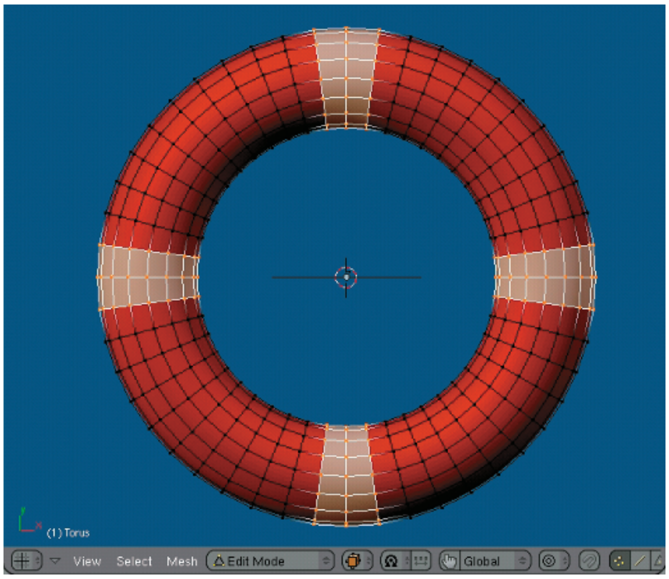


Figure 1.50 A life ring

12. In Object mode, move the torus up slightly, as shown in Figure 1.51. With the torus selected, Shift-select the plane and enter Edit mode with Tab. Select three vertices in the plane in places that correspond to points on the ring, as shown in Figure 1.52. Press Ctrl+P to make the vertex parent. Set the modifiers on the plane to display in the 3D view again and preview the animation by pressing Alt+A. The life ring should follow the motion of the displaced surface, floating gently on the waves.

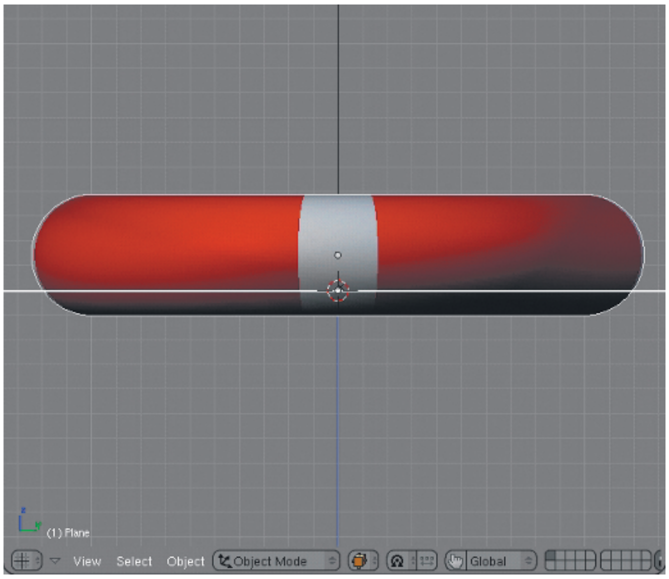


Figure 1.51 The life ring's basis floating position

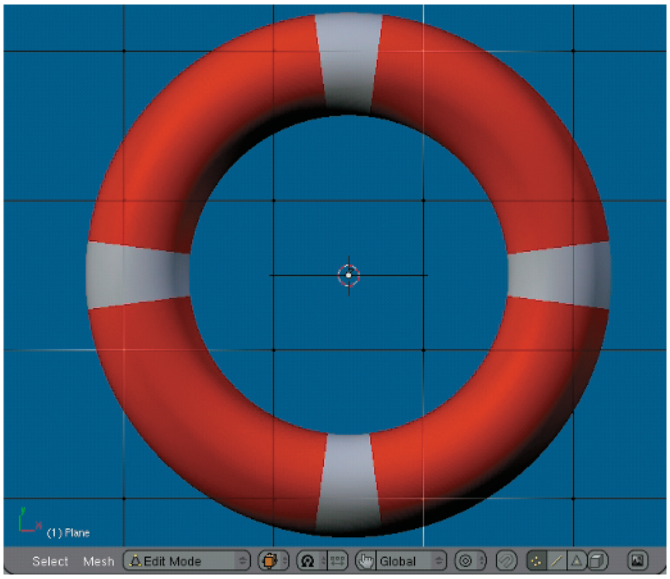


Figure 1.52 Three vertices selected for vertex parenting

13. Run another test render. This time you should see something like Figure 1.53. The water and the ring look okay individually, but the point where they meet is not convincing at all. The line is too abrupt, and there is no sign that the ring is having any effect on the water. Water should deform slightly at the points where it touches an object, as a result of surface tension.

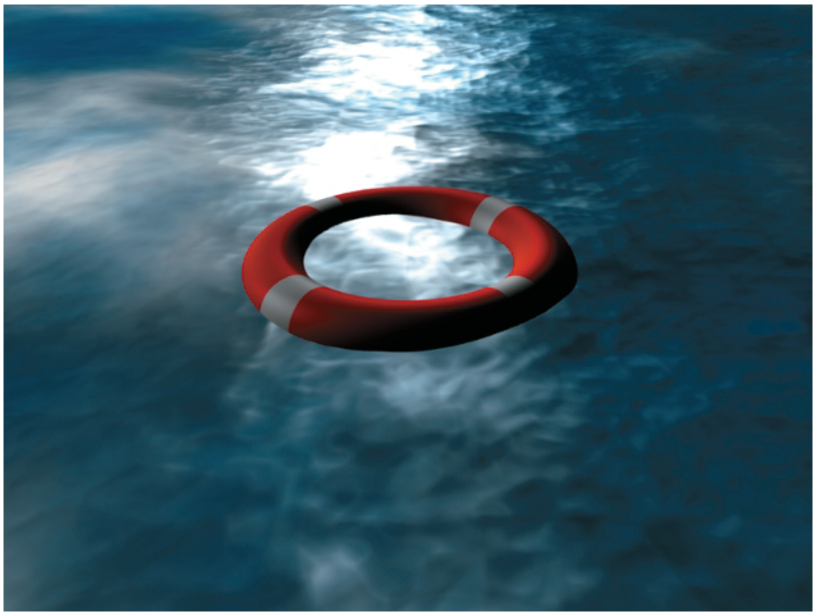
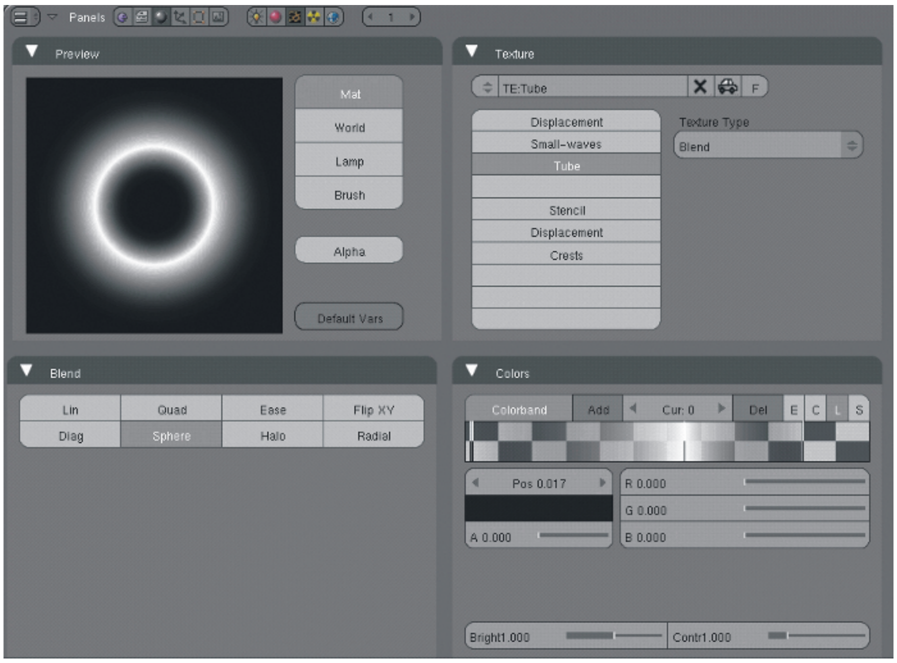


Figure 1.53 Test render of the life ring

14. Add another texture in channel 2 called **Tube**. The Tube texture will be a Sphere Blend type texture. You will also need to add an extra index to the colorband by clicking Add. Make the new color black with alpha 0 and arrange the colorband as shown in Figure 1.54. Set the Map Input value to Object and type **Torus** in the field, as shown in Figure 1.55. This will make this texture follow the movement of the life ring object. Also, adjust the size values in the Map Input tab to 0.50 in all dimensions. In the Map To tab, select color and RayMir. Push the R, G, and B values up to make white. Setting the texture to map to RayMir will enable ray mirroring on the surface of the water, but only in the area directly around the life ring. Using ray tracing will slow down your renders, so you can disable this if you want to speed things up, but it looks good. To enable the ray mirroring, be sure to select Ray Mirror in the Mirror Transp tab, but leave its RayMir slider value at 0.00.



TE:Tube

Texture Type: Blend

Cur: 0

Pos 0.017

A 0.000

R 0.000

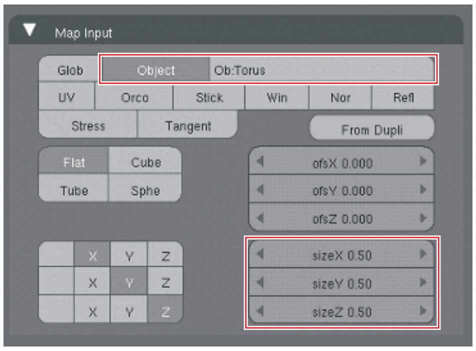
G 0.000

B 0.000

Bright 1.000

Contr 1.000

Figure 1.54 The Tube texture



Ob:Torus

ofsX 0.000

ofsY 0.000

ofsZ 0.000

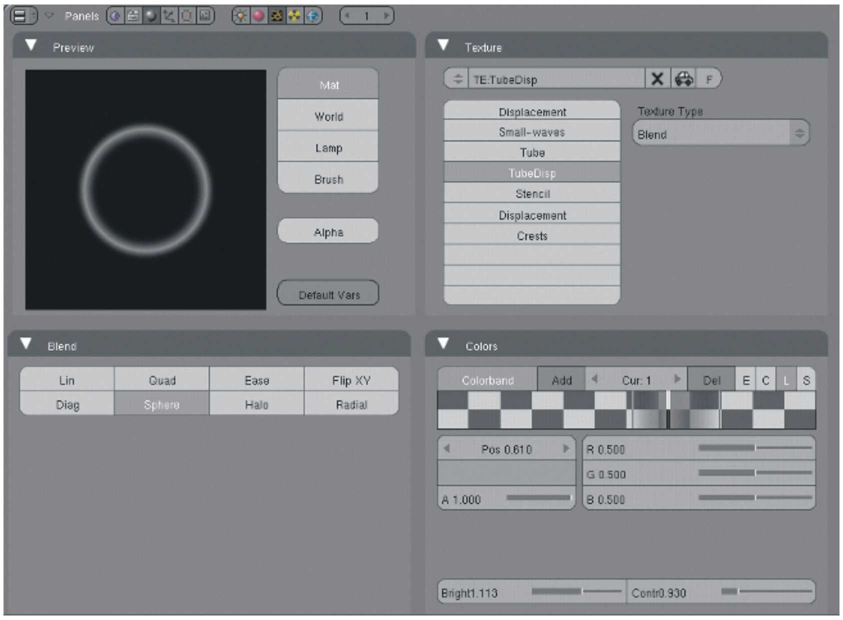
sizeX 0.50

sizeY 0.50

sizeZ 0.50

Figure 1.55 The Tube texture Map Input tab

15. Finally, create the displacement texture for the surface tension on the life ring. Duplicate the Tube texture and place the copy in texture channel 3 by using the Copy To and Copy From Clipboard buttons as you did previously. In the Texture buttons, click the 2 next to the TE drop-down to make the texture a new single-user texture, and rename the texture **TubeDisp**. Set the values as shown in Figure 1.56. Set the mapping values as shown in Figure 1.57. The Map Input values are the same as they were for Tube.



TE:TubeDisp

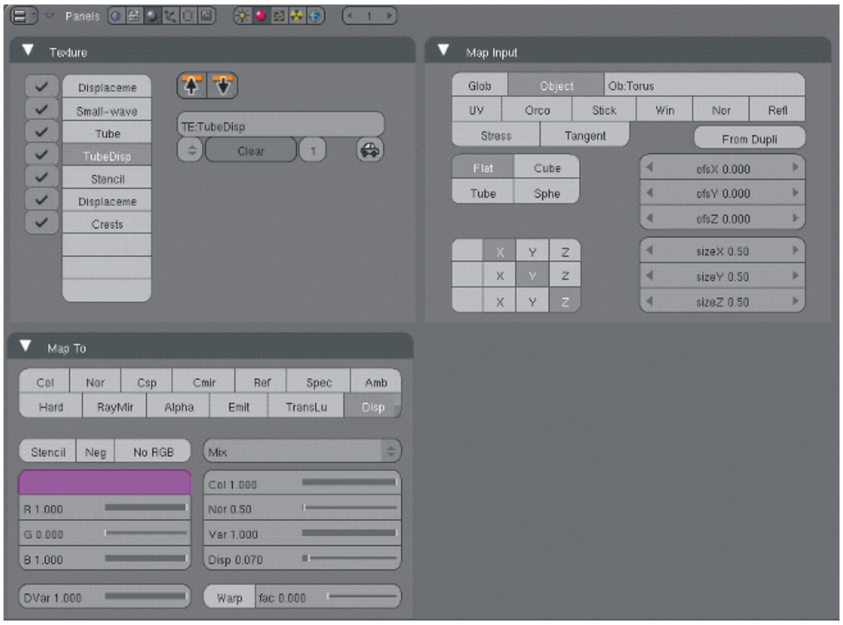
Texture Type: Blend

Cur: 1

Pos 0.610 | A 1.000 | R 0.500 | G 0.500 | B 0.500

Bright 1.113 | Contr 0.930

Figure 1.56 The TubeDisp texture



TE:TubeDisp

Ob:Torus

ofsX 0.000 | ofsY 0.000 | ofsZ 0.000

sizeX 0.50 | sizeY 0.50 | sizeZ 0.50

Mix

R 1.000 | G 0.000 | B 1.000 | DVar 1.000

Col 1.000 | Nor 0.50 | Var 1.000 | Disp 0.070

fac 0.000

Figure 1.57 The TubeDisp texture mapping

16. To see what's happening to the surface of the water, try a test render with the torus placed on a separate layer, out of view. You should see results along the lines of Figure 1.58. Placing the life ring back in the picture where it belongs will give you a final rendered effect like that in Figure 1.59. Render out an animation to see the effect in motion.

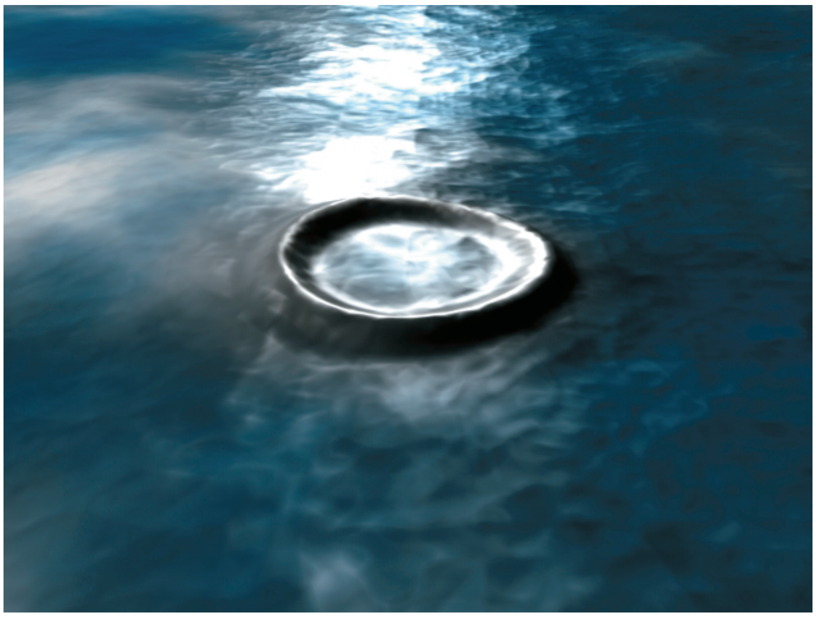


Figure 1.58 Surface tension displacement without the life ring

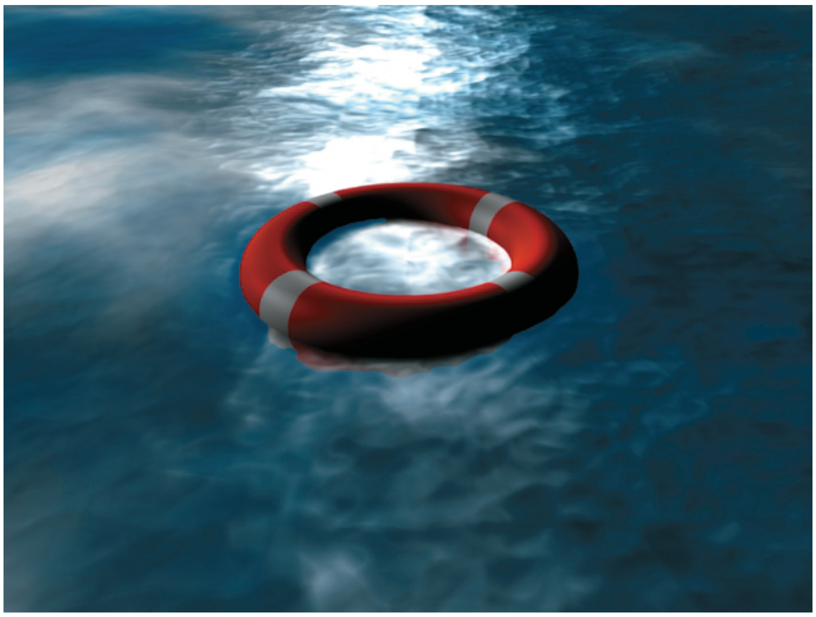


Figure 1.59 The full effect with surface tension and reflection

Faking a Cloth Flag by Using a Displacement Modifier

In Chapter 3, "Getting Flexible with Soft Bodies and Cloth," you'll see how to use soft bodies and how to achieve convincing cloth effects. For animating a flag waving in the wind, there is a simple approach that does not require any actual simulation, but rather uses Blender's displacement modifier. To create the effect, follow these steps:

1. Model the flag and the flagpole. The flagpole can be a simple cylinder scaled appropriately. The flag is a plane, scaled to the appropriate dimensions and subdivided to about the level shown in Figure 1.60. You can press the W key, select Subdivide Multi, and set a factor of 45 to get a good level of subdivision. Activate a Subsurf modifier and click Set Smooth. Create materials for the pole and flag and set the material settings for color and specularity to whatever you think looks good.

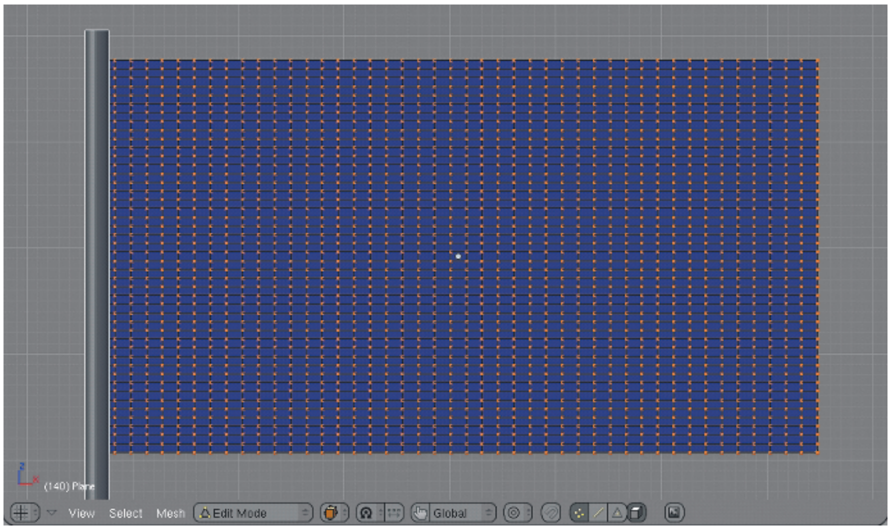
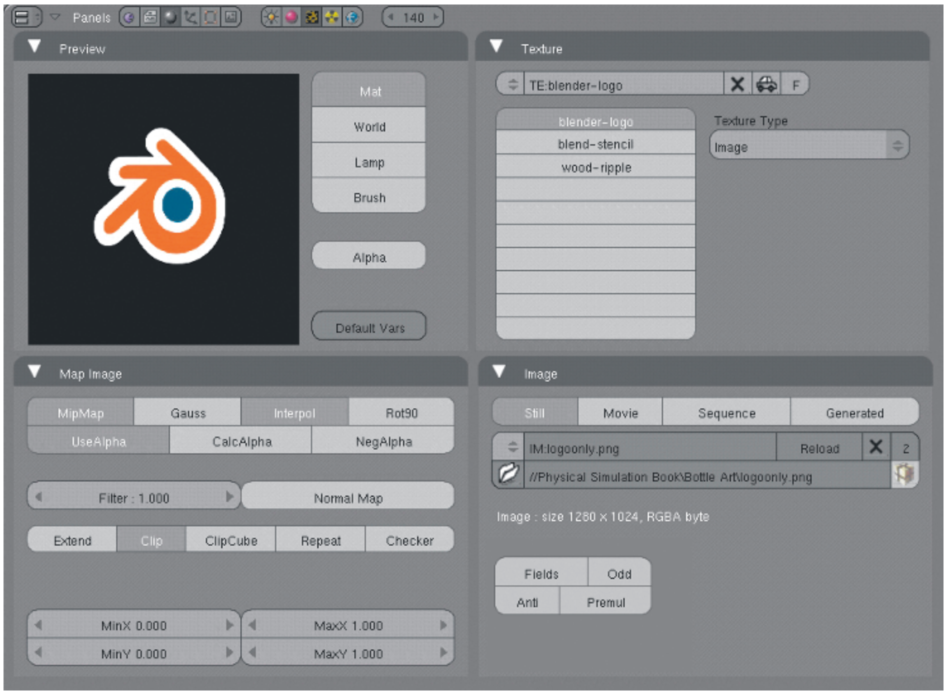


Figure 1.60 The subdivided flag mesh

2. In the Texture panel of the Material buttons, click Add New to add a new material. This will create a new texture in the topmost texture channel in that panel. As I mentioned previously, this is channel 0.

   Add an image texture with the logo image (`logo.png`) from the CD, as shown in Figure 1.61. You need to UV-map this texture to the flag, or else the texture will not behave correctly when the surface of the flag displaces. To do this, access UV Face Select mode with the flag selected, select all faces by pressing the A key, and then open a UV/Image Editor window and unwrap the flag by pressing the E key. Open the logo image in the UV/Image Editor and position the flag and logo appropriately, as shown in Figure 1.62. Map the texture to the Col value in Map To, and in Map Input make sure that UV is selected.



TE:blender-logo

blender-logo
blend-stencil
wood-ripple

Texture Type: Image

Filter : 1.000

MinX 0.000 MaxX 1.000
MinY 0.000 MaxY 1.000

IM:logoonly.png
//Physical Simulation Book\Bottle Art\logoonly.png
Image : size 1280 x 1024, RGBA byte

Figure 1.61 The blender-logo texture

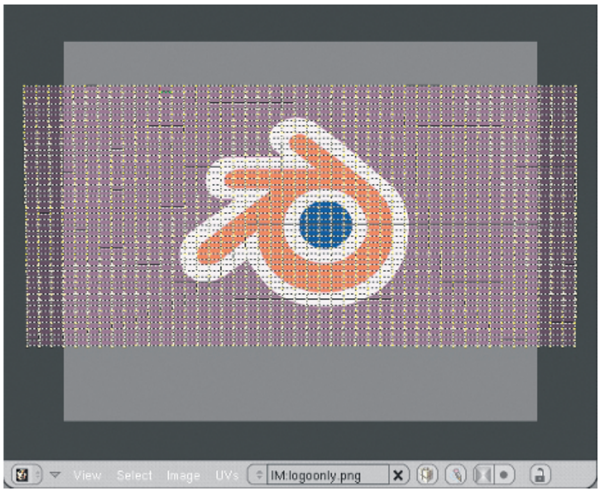


Figure 1.62 UV Mapping the blender-logo texture

3. Create a vertex group for the flag mesh by clicking New in the VertexGroups buttons, and name the vertex group **PoleDamp**. In a moment, you'll use this vertex group to determine the intensity of the displacement modifier that will produce the flag's ripples. Because the flag should be fixed at the pole, the intensity of the ripples will be zero at the pole. You can assign the vertex group weights by hand in Edit mode, or you can do this by weight painting, as shown in Figure 1.63. The part of the flag that meets the pole should be weighted 0, and most of the flag should be weighted 1. The yellow and green transition area in the figure indicates where the weight should be gradated.

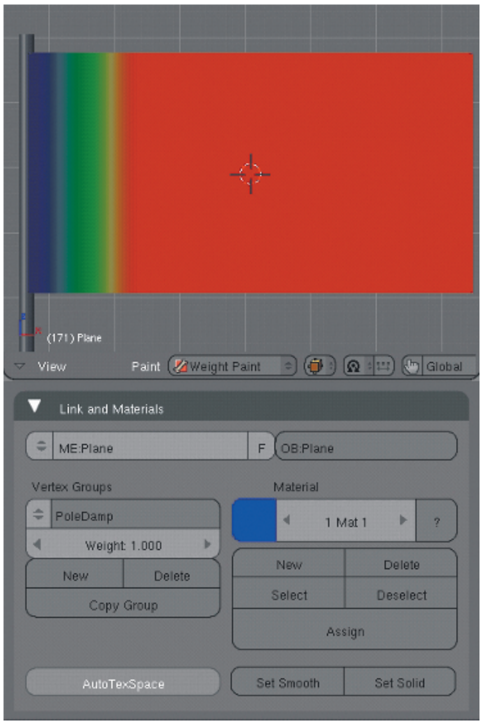


Figure 1.63
Weighting the vertex group

4. Create the texture that will be used to displace the ripples in the flag. Select channel 2 (the next channel down, below the channel with blender-logo) and click Add New. Name the texture **wood-ripple**. This texture will not be used directly on the material, but will be used to control a Displace modifier, so make it inactive on the material by deselecting the channel as shown in Figure 1.64. For the ripple pattern, select a Wood texture type and set the values shown in Figure 1.65.

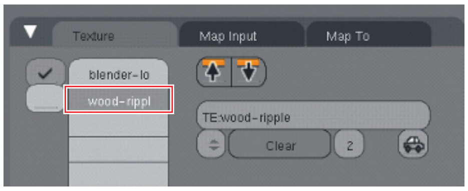
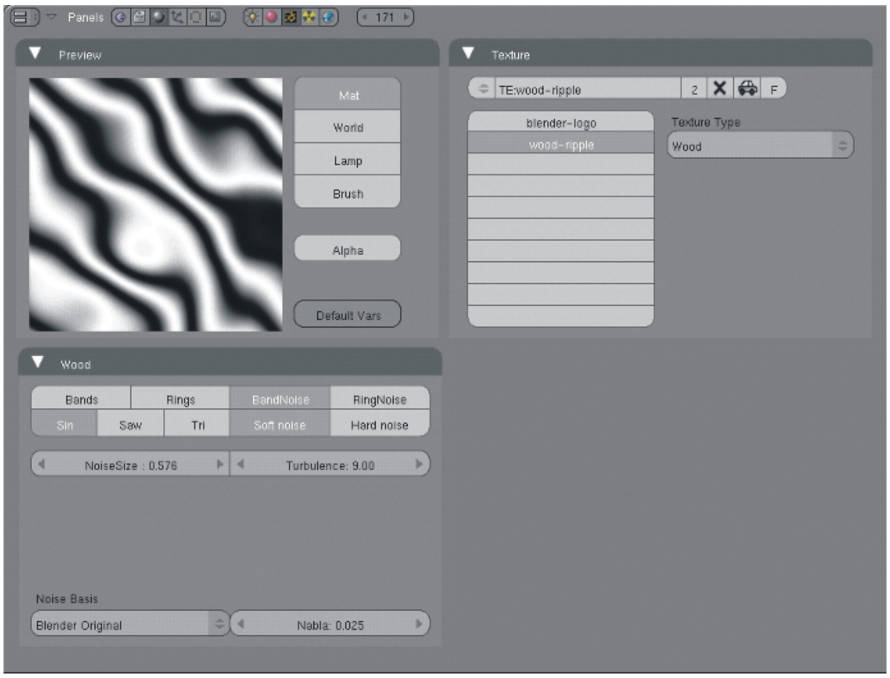


Figure 1.64
Deselect the second texture channel

5. The Displace modifier will be animated by means of an empty. Snap the cursor to the Flag object and add the empty. Clear the rotation on the empty by pressing Alt+R and scale the empty up by a factor of 3.5, so that it looks as shown in Figure 1.66.

6. Now it's time to add the Displace modifier for the flag mesh in the Modifiers tab, with the settings shown in Figure 1.67. For VGroup, type the name of the vertex group you set up in step 3, and select Object from the texture coordinates drop-down menu at the bottom of the panel. Type the name of the empty from step 5 in the Ob field. Leave Midlevel at 0.5, and set the Strength at around 0.2. You can see the effect of the modifier in the 3D window.



TE:wood-ripple 2

blender-logo
wood-ripple

Texture Type: Wood

NoiseSize : 0.576
Turbulence: 9.00

Noise Basis: Blender Original
Nabla: 0.025

Figure 1.65 The ripple texture

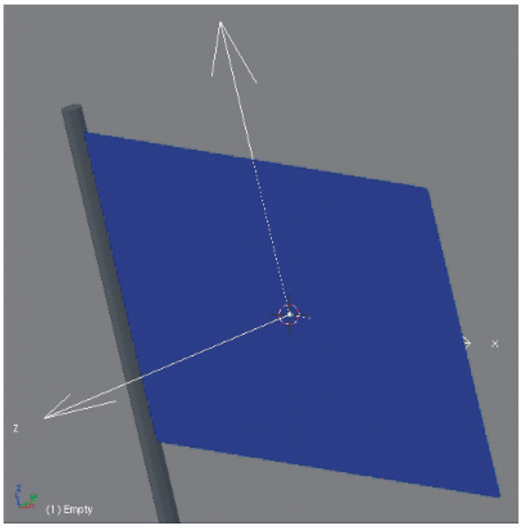
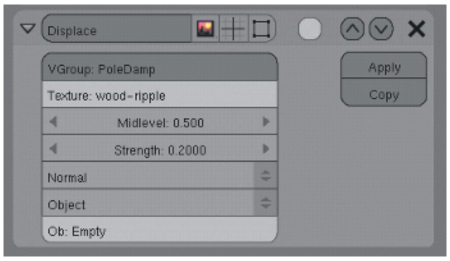


Figure 1.66 Adding an empty



Displace

VGroup: PoleDamp
Texture: wood-ripple
Midlevel: 0.500
Strength: 0.2000
Normal
Object
Ob: Empty

Figure 1.67 The Displace modifier

7. The last thing that needs to be done is to add motion by animating the empty. Add a LocRot keyframe at frame 0 with the empty at its original position by pressing the I key. Advance 50 frames and translate the empty along its X axis to about the edge of the flag (it doesn't need to be perfect), and then add another LocRot keyframe. Select the LocX Ipo in the Ipo Editor and change its Extend mode to Extrapolation by choosing Curve > Extend Mode > Extrapolation in the header menu. The Ipos should look as shown in Figure 1.68.

8. If you render a still now, it should look something like Figure 1.69. (As you can see, I've added a sky map for the background here, as discussed previously in this chapter.) Render a test animation to make sure that the level of displacement is right and the speed looks as you want it. If you need to adjust the speed of the flapping, adjust the LocX Ipo for the empty.

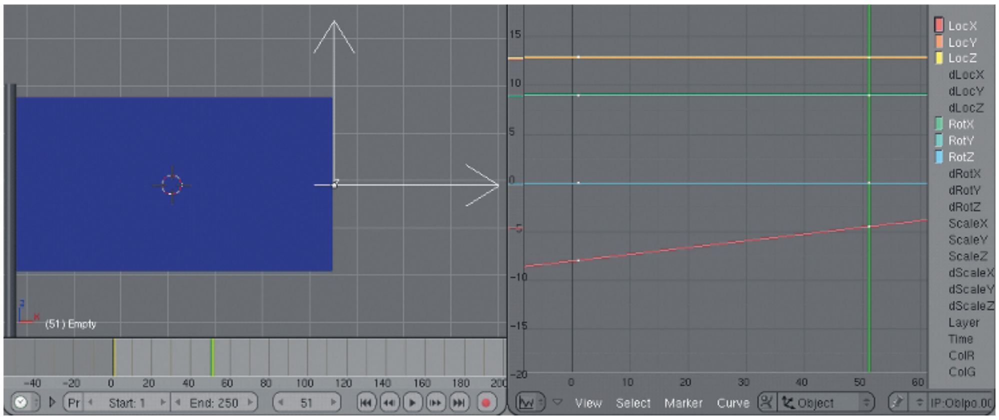


Figure 1.68 Keying an Ipo for the empty

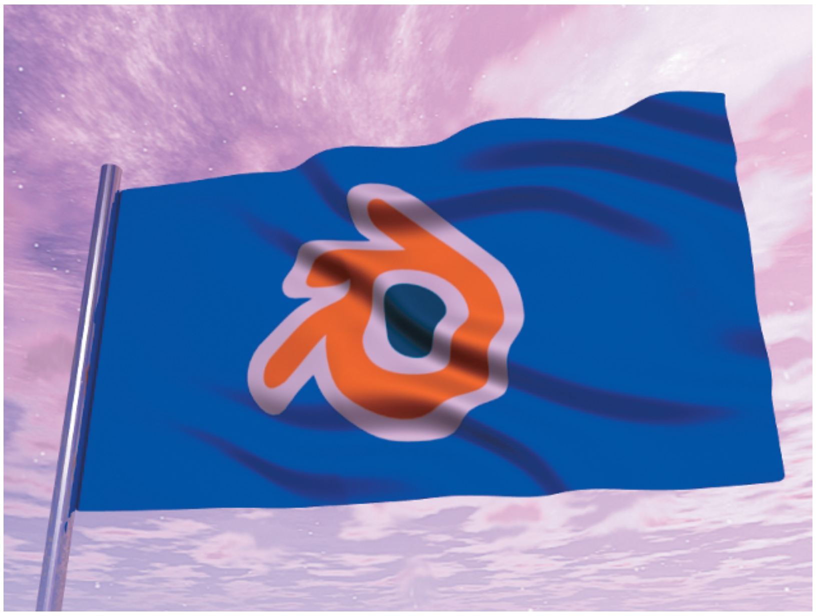


Figure 1.69 Flag with displacement texture

Creating a Poseable Spring by Using an Array Modifier, Shape Keys, and PyDrivers

A recurring concept in the area of physical simulation is the behavior of springs. Springs are designed to be flexible and to extend a certain distance relatively easily. As a spring extends, internal forces build up that compel the spring to compress. Eventually (assuming the extending force does not break the spring), the spring bounces back into a compressed position. A spring has a natural point of equilibrium, and if it is pushed by an external force in either direction out of this position, its internal forces will push back and forth with decreasing energy until the spring returns to its position of equilibrium. Many physical forces can be well described as behaving analogously to springs.

Several of Blender's various simulators model something like spring behavior, and later in the book I discuss how this results in the physical effects that you use in your animations. But what about an actual spring? Aside from the issue of simulating the internal forces of a spring, there is the question of how to model, rig, and deform the 3D object itself. It is not entirely trivial to do this, and the approach I describe takes advantage of a variety of useful features of Blender and shows how these can be used together to achieve a simple and elegant solution for a rigged spring. First, I show how to model the spring itself by using the Array modifier in conjunction with driven shape keys (thanks to Blender user mexicoxico for his post on BlenderArtists.org, outlining this approach). Next, I'll show how a simple armature can be added that uses a PyDriver to control the deformation of the spring in an intuitive way.

Later, in Chapter 3, this spring will also come in handy as a nice example of the basic mechanics of soft bodies and one of the ways they can be used to control the movements of non-soft- body objects.

Modeling the Spring Mesh

Begin in front view (NUM1). Make sure the cursor is exactly in the center. To do this, first select the default cube, press Shift+S, and select Cursor To Selected. Then delete the default cube, add a plane by choosing Add > Plane, and subdivide once, as in Figure 1.70.

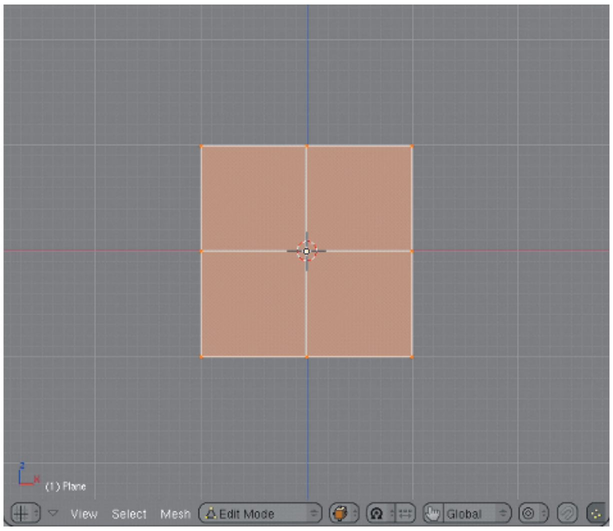


Figure 1.70 Subdivided plane

Select the leftmost and lowermost vertices as in Figure 1.71 and delete them, to result in a square exactly one Blender Unit high, as in Figure 1.72. In Edit mode, select all by pressing the A key, and move the square 3 Blender Units in the positive direction along the X axis (that is, to your right). You can do this by pressing the G key followed by the X key, and holding Ctrl while you move the square. Looking at the situation from top view (NUM7), it will look like Figure 1.73. It's important to do this in Edit mode, so that the object center remains where it is.

Press the period key to toggle the rotation pivot to the 3D cursor (you can toggle it back to the median point by pressing Shift+comma). In top view with all the vertices selected, press the E key and select Region to extrude the face, followed by the R key to rotate the extruded face. Press the Z key once to force rotation around the global Z axis, and input the value –30 so that the extruded face rotates as in Figure 1.74.

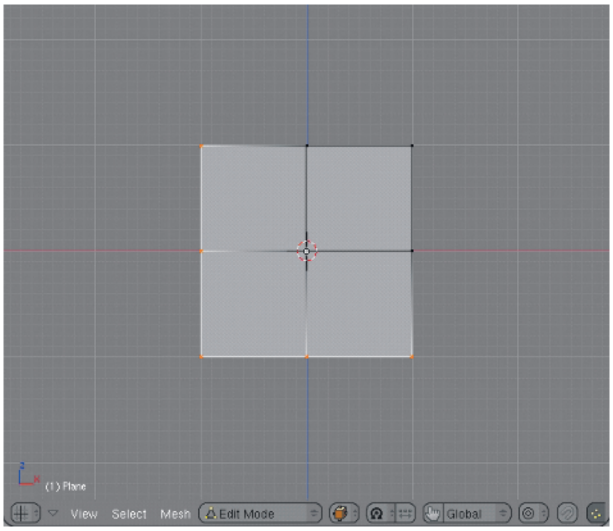


Figure 1.71 Select and delete these vertices.

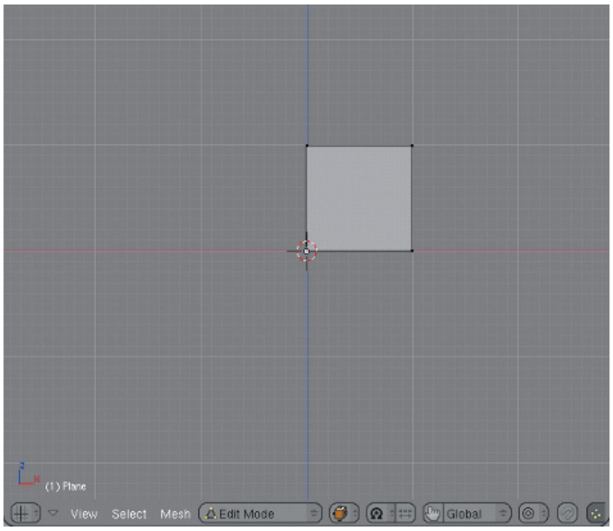


Figure 1.72 A one-BU-high square, in front view

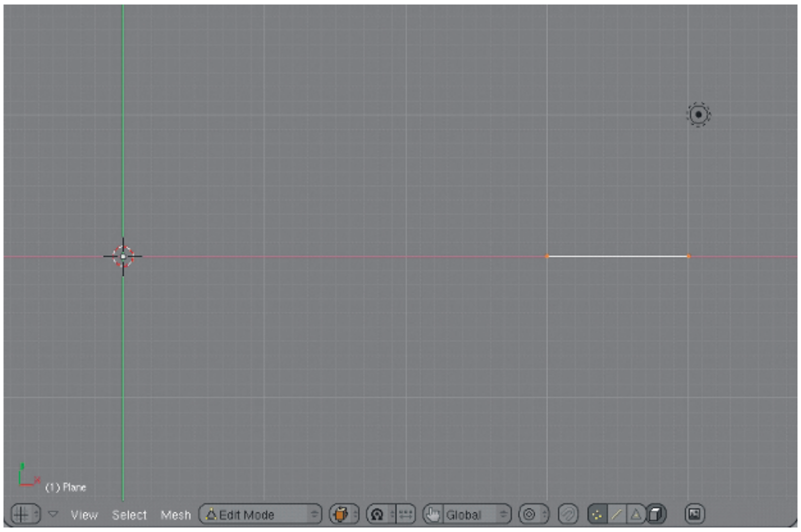


Figure 1.73 Moving the square in Edit mode, top view

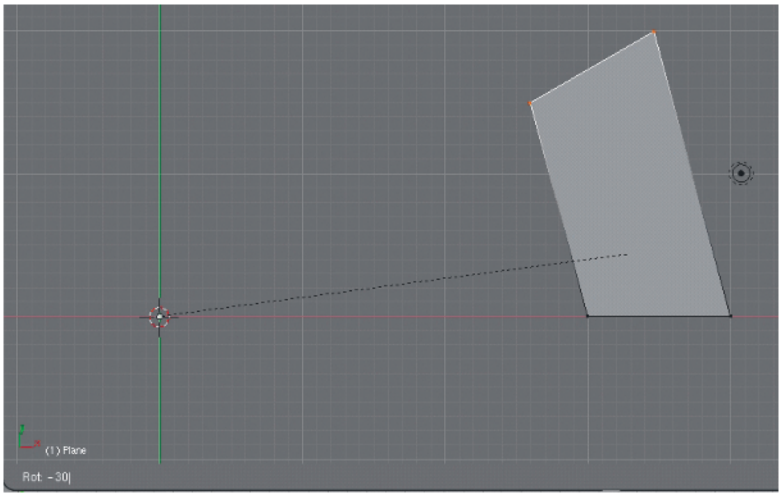


Figure 1.74 Extrude and rotate the face –30 degrees, top view.

Enter Face Select mode either by using the header menu or by pressing Ctrl+Tab+3. Press Z to go into transparent view and select and delete the faces shown in Figure 1.75.

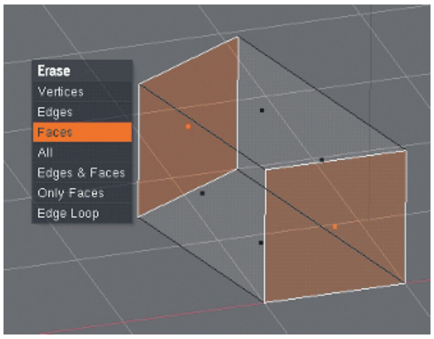


Figure 1.75 Delete the end faces.

In top view, enter Object mode and once again ensure that the cursor is on the center of the object. Press the spacebar to add an empty. Select the mesh, and press Ctrl+A to apply the current scale and rotation to the object, as in Figure 1.76.

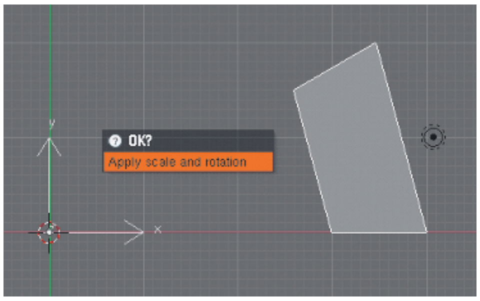


Figure 1.76 Add an empty and apply the scale and rotation to the mesh object.

Now things begin to get interesting. With the mesh selected, in Object mode, go to the Modifiers panel in the Edit buttons and select an Array modifier. The Array modifier creates a sequence of copies of the original object that can be manipulated in various ways. One of the ways to manipulate the array is to use an object to determine how the coordinates of each instance of the object in the array differs from the previous instance. To do this, deselect Relative Offset and instead select Object Offset. This will make the offset between instances of the array dependent on a separate object. The object to use for this is the empty, so fill in the Object Offset Ob field with the name of the empty, which is **Empty**, as in Figure 1.77.

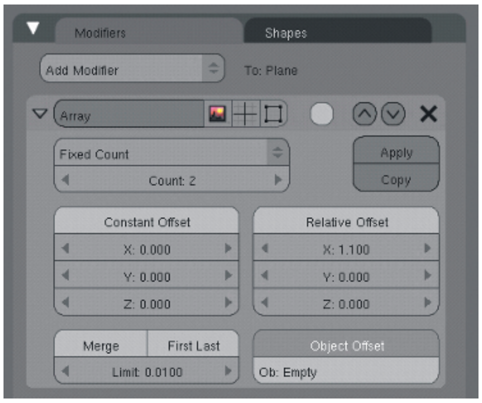

Modifiers
Shapes
Add Modifier
To: Plane
Array
Fixed Count
Count: 2
Apply
Copy
Constant Offset
X: 0.000
Y: 0.000
Z: 0.000
Relative Offset
X: 1.100
Y: 0.000
Z: 0.000
Merge
First Last
Limit: 0.0100
Object Offset
Ob: Empty

Figure 1.77 Array modifier

Select the empty in top view, rotate it by pressing the R key, and input the value –30 degrees. Rotating in this way means that each subsequent instance in the array will be offset from the previous instance by a –30 degree rotation. At the moment, because Count is set at 2 by default, there are only two instances in the array, so you should be seeing something like Figure 1.78.

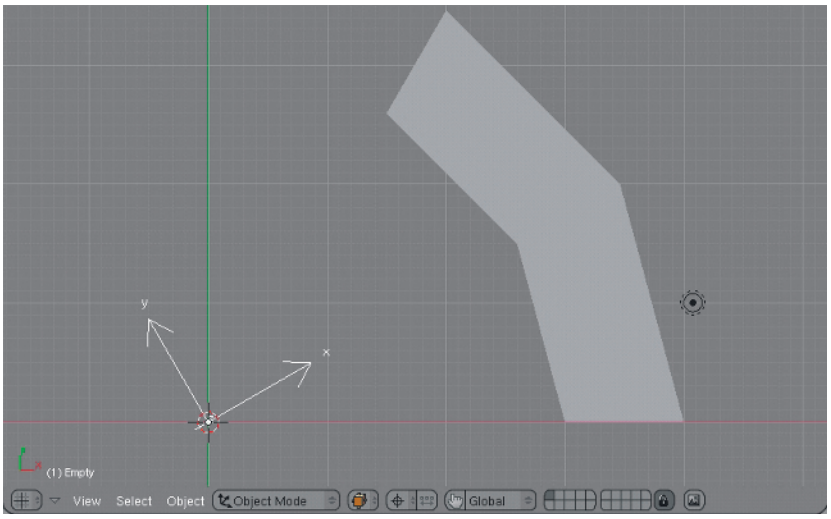


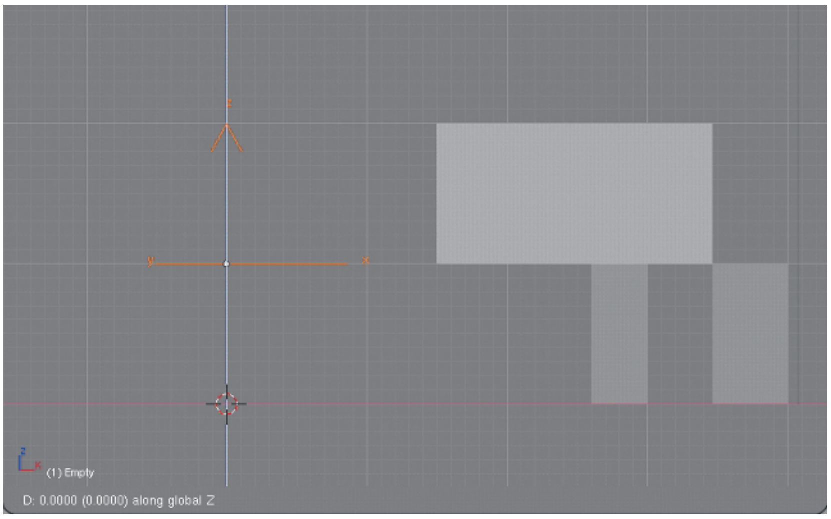
Figure 1.78 Rotating the empty

Setting Up the Shape Keys

This offset will account for the shape of the spring around its central axis. However, the spring also needs to be offset upward in the direction of the axis. Furthermore, simply offsetting the mesh with the Array modifier will not result in an unbroken coil, but in a lot of small chunks stair-stepping around the axis, which is not right for a spring. In addition to this, the deformation of the spring along the Z axis must be possible to animate.

In Blender, deformations that can be animated as increasing and decreasing linearly in intensity are easy to achieve with shape keys. To set up the necessary shape keys, follow these steps:

1. By pressing the G key and then the Z key, and holding Ctrl to constrain the movement to discrete increments, translate the empty exactly one Blender Unit up along the Z axis. This will cause the second instance of the array-modified mesh to offset upward along the Z axis also, as shown in Figure 1.79.
2. With the mesh object selected, go to the Shapes tab in the Edit buttons, and click Add Shape Key. This creates the basis shape key. Click Add Shape Key one more time to create a deform shape key called Key 1.
3. With Key 1 selected in the panel, enter Edit mode. The idea is to create a shape key such that the end of one array instance meets the beginning of the next, creating an unbroken coil. To do this, it is necessary to select only the vertices shown in Figure 1.80, and translate them directly upward along the Z axis one Blender Unit (hold Ctrl while translating, to constrain the movement), as shown in Figure 1.81.
4. In the Modifiers tab, toggle Merge on in the Array modifier panel, select Set Smooth on the mesh, and add a Subsurf modifier set to Levels: 2 and Render Levels: 3. As you can see in Figure 1.82, the spring is taking shape.



D: 0.0000 (0.0000) along global Z

Figure 1.79 Translating the empty one BU up along the Z axis offsets the array.

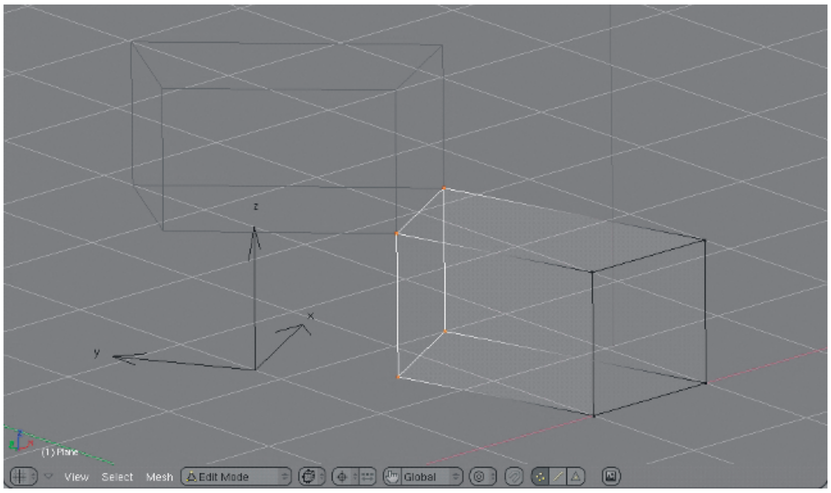


Figure 1.80 These vertices will be edited to create the shape key.

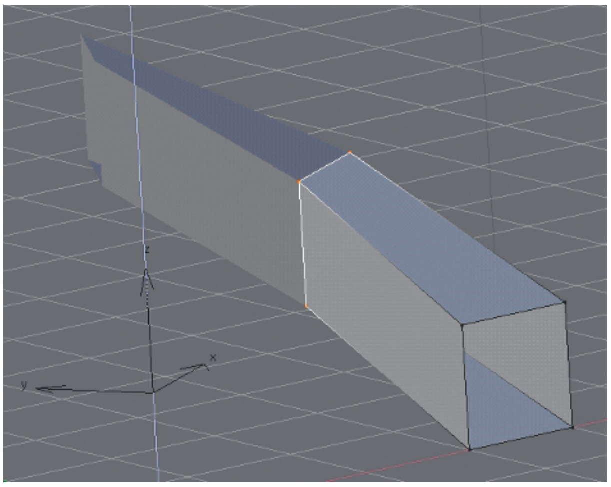


Figure 1.81
Translate the vertices straight up to meet the bottom vertices of the array-modified instance.

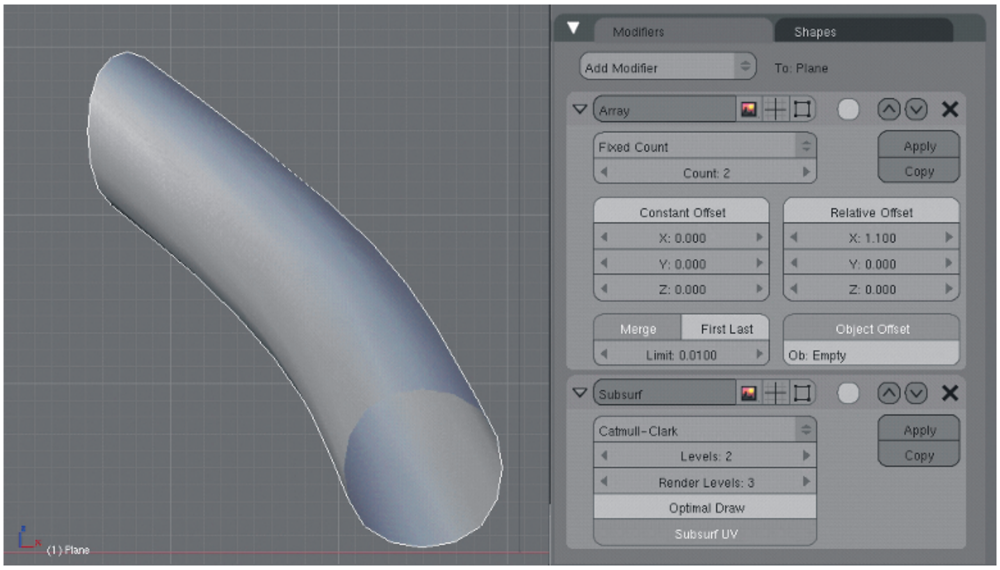

Modifiers | Shapes
Add Modifier | To: Plane
Array
Fixed Count
Count: 2
Apply
Copy
Constant Offset | Relative Offset
X: 0.000 | X: 1.100
Y: 0.000 | Y: 0.000
Z: 0.000 | Z: 0.000
Merge | First Last | Object Offset
Limit: 0.0100 | Ob: Empty
Subsurf
Catmull-Clark
Levels: 2
Render Levels: 3
Optimal Draw
Subsurf UV
Apply
Copy

Figure 1.82 Adding a Subsurf modifier

5. The shape will be driven by the location of the empty. To set up the shape key driver, open an Ipo Editor window with the mesh selected and choose Shape from the header drop-down list. Press Ctrl+LMB to create an Ipo curve, and then again to add another vertex on the Ipo, along the lines of Figure 1.83. It doesn't really matter where the vertices are located; you'll be adjusting them manually in a moment.

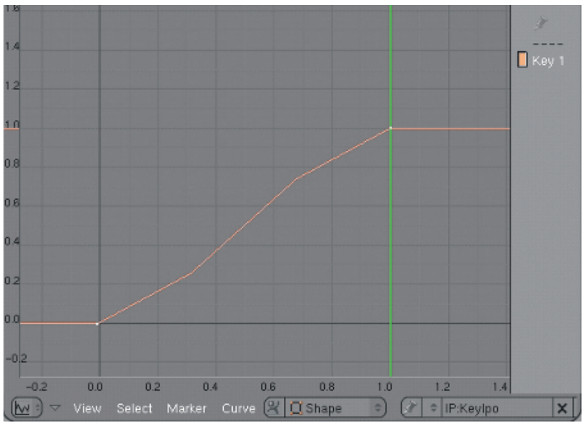
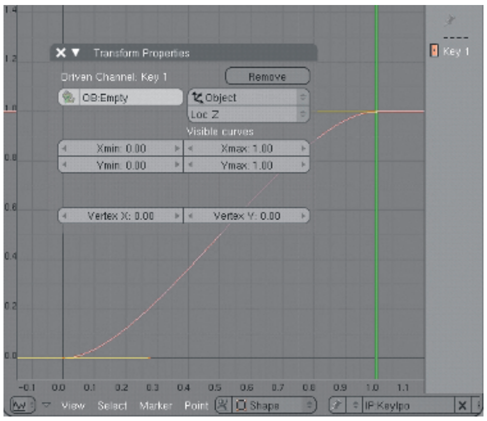


Figure 1.83 Placing vertices on the Ipo

6. Press the N key to bring up the Transform Properties dialog box in the Ipo Editor. Click Add Driver and type **Empty** in the OB field. In the drop-downs to the right of this field, select Object and LocZ. In Edit mode, select the control points on the curve one by one and make sure that they are set at Vertex X: 0, Vertex Y: 0, and at Vertex X: 1, Vertex Y: 1, as shown in Figure 1.84.

7. Tab out of Edit mode, and from the header menu choose Curve > Extend Mode > Extrapolation. This results in the Ipo becoming a straight diagonal line.

8. Go back to the Array modifier on the mesh. Set Count to 250. Your spring is now fully formed, and you can control its degree of extension by translating the empty along the Z axis, as shown in Figure 1.85. The most natural range is to have the empty between the 0.1 and 1 points on the Z axis. You can see and set the Transform Properties for the empty in the 3D viewport by pressing the N key.


Transform Properties
Driven Channel: Key 1 | Remove
OB:Empty | Object | Loc Z
Visible curves
Xmin: 0.00 | Xmax: 1.00
Ymin: 0.00 | Ymax: 1.00
Vertex X: 0.00 | Vertex Y: 0.00

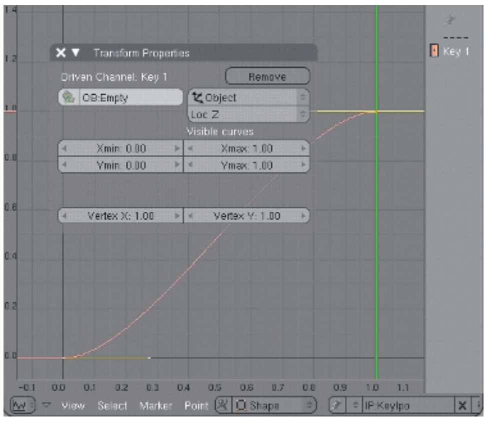
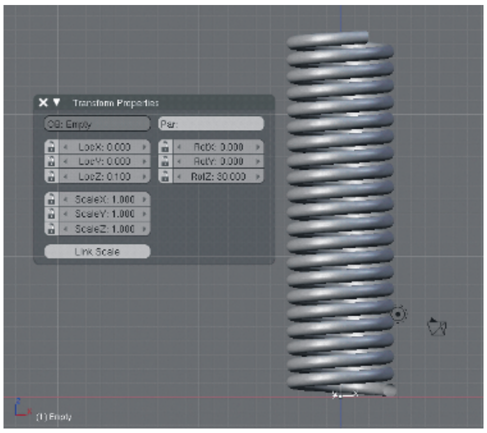
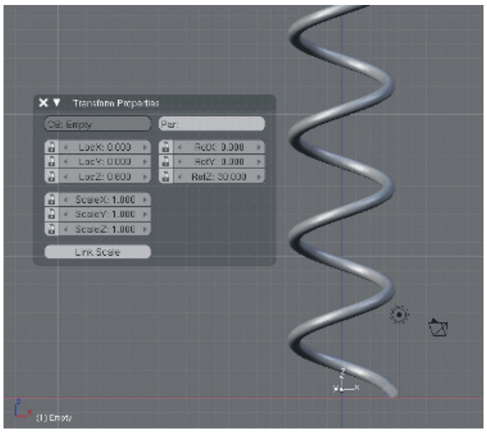

Transform Properties
Driven Channel: Key 1 | Remove
OB:Empty | Object | Loc Z
Visible curves
Xmin: 0.00 | Xmax: 1.00
Ymin: 0.00 | Ymax: 1.00
Vertex X: 1.00 | Vertex Y: 1.00

Figure 1.84 Ipo transform properties


Transform Properties
OB: Empty | Par:
LocX: 0.000 | RotX: 0.000
LocY: 0.000 | RotY: 0.000
LocZ: 0.100 | RotZ: 30.000
ScaleX: 1.000
ScaleY: 1.000
ScaleZ: 1.000
Link Scale


Transform Properties
OB: Empty | Par:
LocX: 0.000 | RotX: 0.000
LocY: 0.000 | RotY: 0.000
LocZ: 0.600 | RotZ: 30.000
ScaleX: 1.000
ScaleY: 1.000
ScaleZ: 1.000
Link Scale

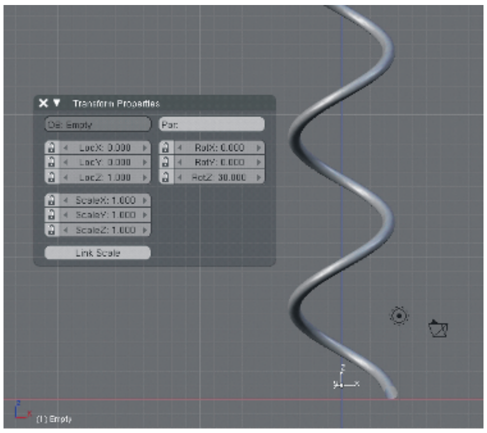

Transform Properties
OB: Empty | Par:
LocX: 0.000 | RotX: 0.000
LocY: 0.000 | RotY: 0.000
LocZ: 1.000 | RotZ: 30.000
ScaleX: 1.000
ScaleY: 1.000
ScaleZ: 1.000
Link Scale

Figure 1.85
The extension of the spring is controlled by translating the empty.

Rigging the Spring

Although it is now possible to control the position of the spring by using a combination of object transforms on the spring itself and on the empty that is driving its deformation, doing so would be unnecessarily complicated. Intuitively, for a spring like this, it is desirable to simply have two control points, one at each end of the spring, that can be used to pose the spring directly. This can be done by using a simple armature to control the position and rotation of the objects, and to drive the movement of the empty by means of a PyDriver.

To create the armature, once again go into Object mode and ensure that the cursor is snapped to the location of the original mesh's center, which should also be the location of the empty. Press the spacebar to add an armature. Select X-Ray in the Editing Options area in the Armature tab of the Edit buttons. Press the G key and the Z key and hold Ctrl to move the tip of the bone directly up the Z axis 25 Blender Units, as shown in Figure 1.86. Extrude by pressing the E key and draw the tip of the second bone another 25 Blender Units up along the Z axis, as in Figure 1.87. With this bone selected, press Alt+P and select Clear Parent.

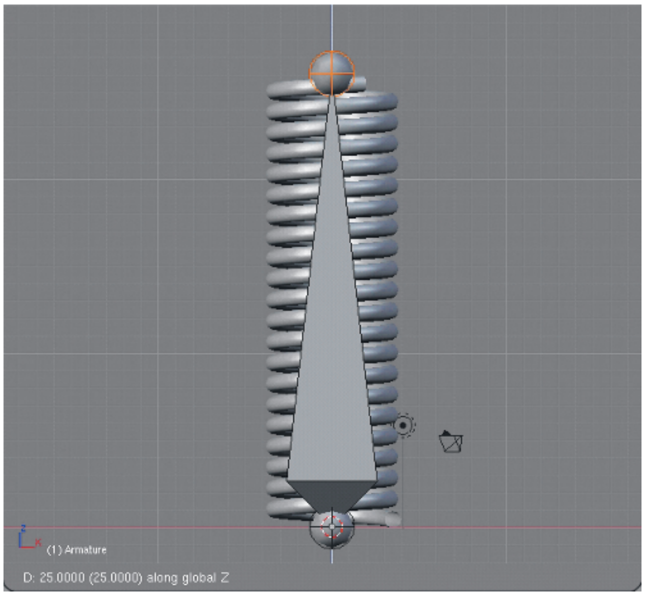


Figure 1.86 Adjusting the size of the first bone

Go into Pose mode. Select the top bone, whose name is Bone.001, and then Shift-select the bottom bone, called Bone. Press Ctrl+I to add an inverse kinematics (IK) constraint to the selected bone. The bottom bone, Bone, should turn yellow.

Now bone-parent both the spring mesh object and the empty to Bone. To do this, select the mesh, Shift-select the empty, and then select the armature, which will appear in Pose mode. Select Bone, press Ctrl+P, and select Make Parent To Bone.

To make the armature's appearance more intuitive, it will be useful to create a custom bone shape to represent the ends of the springs. I used an extruded 12-vertex circle for this.

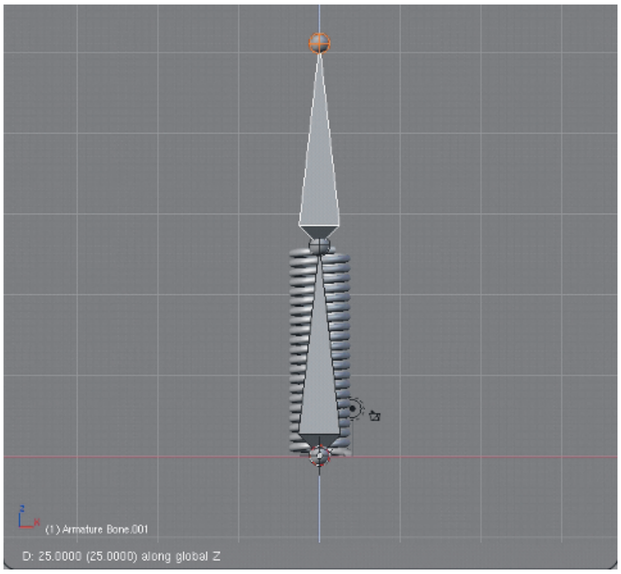


Figure 1.87 The second bone

With the armature selected in Pose mode, go into the Armature Bones tab in the Edit buttons and type **Circle** into the OB field for each bone. Then go to the Draw tab in the Object buttons and select the Wire draw type.

You'll want to adjust the size of the Circle object so it displays sensibly. When you change the size or rotation of the object, press Ctrl+A to apply the scale and rotation, to have the object display with the new values in the armature. Your rig should look something like Figure 1.88.

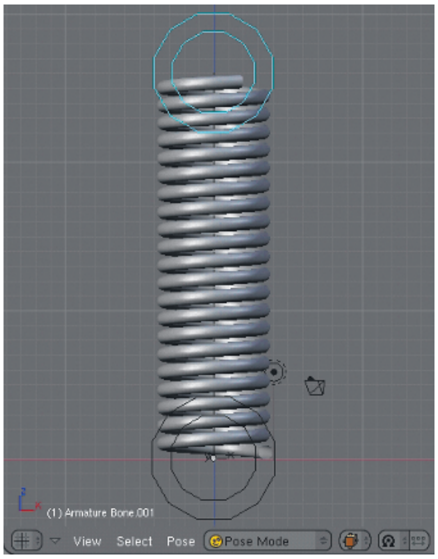


Figure 1.88
The rig so far

The Circle object itself you can hide in the Outliner by toggling the visible, selectable, and renderable icons off, as in Figure 1.89.

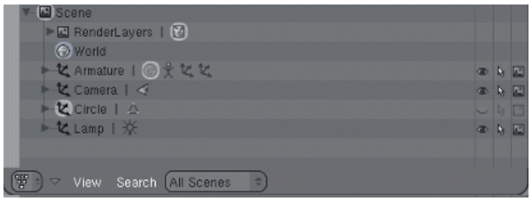


Figure 1.89 Controlling visibility, selectability, and renderability in the Outliner

Setting Up a PyDriver

You can now pose the armature, and the spring and empty will rotate around to follow the endpoints of the armature as they ought to. However, the spring does not extend to follow the endpoints, because the empty is not being translated along its Z axis.

You might consider using a stretch bone in some way, but this is likely to add undesirable side effects of distorting the mesh. Using some combination of copy location constraints with very small or precisely adjusted influence values may also be possible, but it is not practical in this case. The most simple and straightforward approach is to use a PyDriver to drive the Z location of the empty as a small fraction of the vector distance between the two bones' roots.

If you have some experience with Python scripting in Blender, you will find PyDrivers straightforward. If not, it is probably best to study some Python programming and familiarize yourself with the basics of Python scripting in Blender before diving into PyDrivers. The remainder of this section assumes that you have some basic knowledge of this. Unfortunately, it's beyond the scope of this book to go into depth on this background information. The information I present here won't be crucial to anything else in the book, though, so feel free to skip it if you don't feel ready for it.

Unlike with ordinary scripts, you are limited to a single line of code. Also, PyDrivers can use a shorthand form that is not available for ordinary scripting to access several common data types. These shorthand forms are as follows:

- ob('name') to access the object named name
- me('name') to access the mesh named name
- ma('name') to access the material named name

In this case, the values we're interested in are found in the Armature object, so the first of these shorthand forms will come in handy to access that object.

An object of class Armature has a Pose object associated with it, accessed by the .getPose() method, which in turn has a set of PoseBones corresponding to the bones of the armature. The .bones attribute of the Pose is a dictionary keyed on the name of the bone, which returns a PoseBone object. These PoseBone objects, intuitively, represent the bones of the armature when it is being posed. The location of the PoseBone is the location of the bone in Pose mode. Furthermore, each PoseBone has an attribute head and a tail, and each has a set of coordinates associated with it. The location of the base of the bone is the head, so this is the value of interest here.

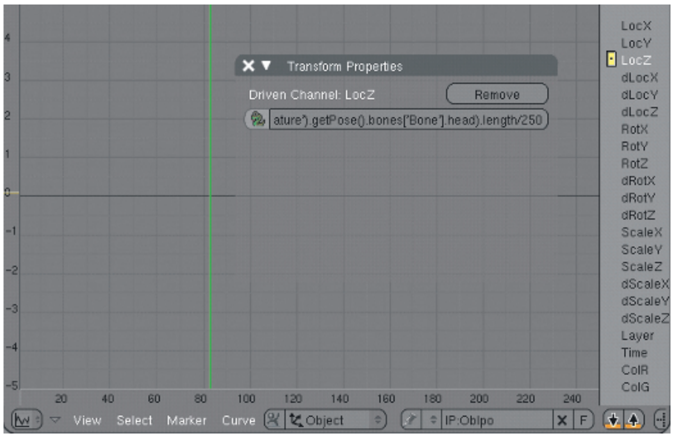
Python in Blender has access to a math utilities library that enables it to do vector math simply, so finding the distance between two 3D points is simply a matter of subtracting one from the other.

Finally, because the rest position of the spring has the empty at location 0.1 BU and the distance between the bones at 25 BUs, and the empty's location should change proportionately to the distance between the bones, it is possible to calculate the position of the empty as the distance between the two bones, divided by 250.

Putting all that together yields a single line of code that will serve as the PyDriver for the local Z axis location of the empty:

```
(ob('Armature').getPose().bones['Bone.001'].head-
ob('Armature').getPose().bones['Bone'].head).length/250
```

Setting up the driver is simple. Select the empty and select Object from the header drop-down in the Ipo Editor. Select LocZ from the list of Ipos along the right side of the Ipo Editor window. Press N to show the Transform Properties dialog box. Click Add Driver, and then click the Python snake icon to the left of the OB field. In the new field that appears, type the preceding code in a single unbroken line, as shown in Figure 1.90.


Transform Properties

Driven Channel: LocZ

```
ature').getPose().bones['Bone'].head).length/250
```

Figure 1.90 PyDriver

After this is done, the spring is fully rigged. You can play around with posing it by moving either end of the armature and seeing how it behaves, as in Figure 1.91. In Chapter 3, you'll revisit this rigged spring to see how a simple soft body simulation can be used to control the spring's behavior.

PyDrivers

Blender uses the Python language for scripting, and a great deal of additional functionality can be implemented by using Python scripts. Another way that Blender uses Python is in PyDrivers, which enable an Ipo curve to be driven by any value that can be expressed in a single line of Python code with access to the Blender Python applications programming interface (API). This means that rather than having a one-to-one correspondence with the value of another Ipo, as ordinary Ipo drivers do, an Ipo can be driven by much more sophisticated operations on multiple input values. In the example in the text, an Ipo is driven by an operation on the vector distance between two bones.

It is beyond the scope of this book to give a thorough introduction to Python or to object-oriented programming (OOP), but people who have some experience with the ideas behind OOP will find PyDrivers fairly easy to pick up after a bit of studying the Blender Python API.

Accessing an object's attributes or calling class methods on an object is done by appending the name of the attribute or the method to the end of the name of the object, separated by a period. In the case of calling a method, the method call ends with a set of parentheses enclosing the arguments. If there are no arguments, as in the case of most getter methods, the parentheses are empty.

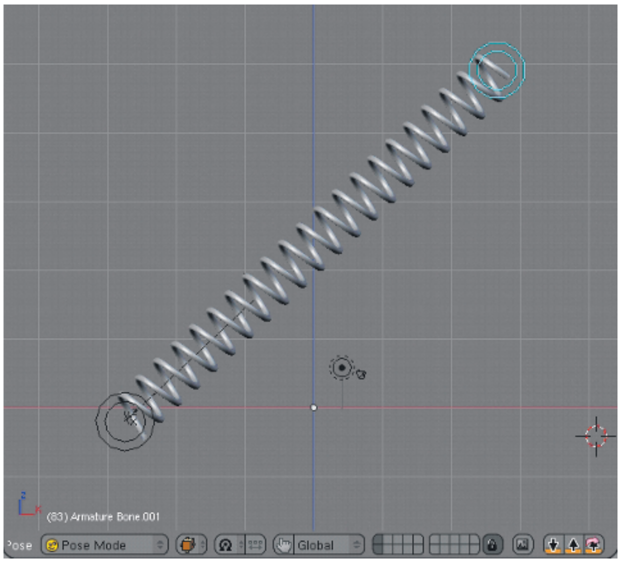


Figure 1.91 The finished, poseable spring

In this chapter, you saw several interesting ways to use general-purpose tools in Blender to represent various physical phenomena. Some of the tools you may have already been familiar with, but I hope that you've had your imagination stimulated to think of new and creative ways to use these tools to achieve the effects you want. In the next chapter, you'll look at Blender's powerful new particle system, and the discussion of physical simulation will begin in earnest.