FROM THE PUBLISHERS OF VIDEO TOASTER USER MAGAZINE

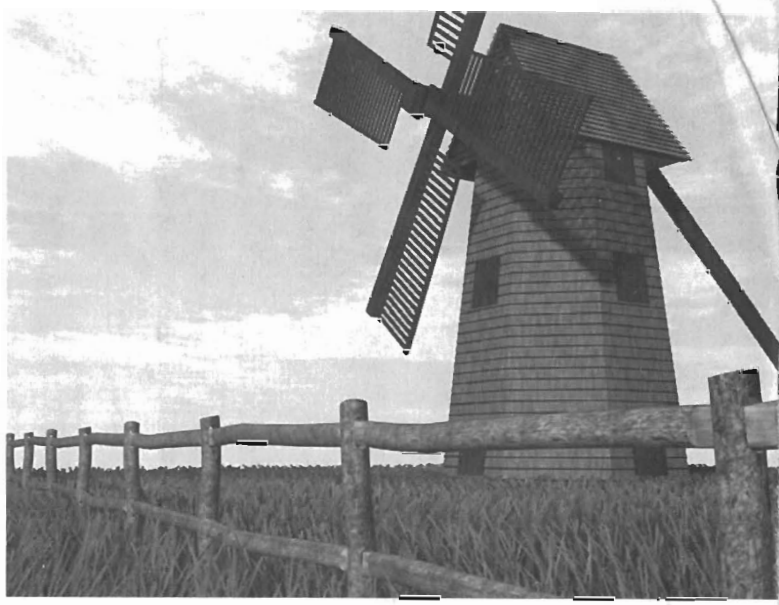
PREMIERE ISSUE

LIGHTWAVE PRO

issue number 1 October 1993

THE NEWSLETTER FOR SERIOUS LIGHTWAVE 3D ANIMATORS

Welcome to LightWavePRO!



I'm a LightWave junkie. I've known it for awhile and have finally come to accept it. And why not? LightWave changed my life. Four years ago, I was working as a purchasing agent for a swimming pool company in Minnesota. I went to an Ami Expo in Chicago and saw Allen Hastings demonstrating LightWave on the not-yet-released Video Toaster. Never had I seen a 3D package on a personal computer like this! I fell in love with LightWave and knew I had to have a Toaster no matter what the cost.

Not long after, I quit my job and started working for a video company where I soon became the LightWave expert. About a year and a half later, I moved out to L.A. to work as a LightWave animator in Hollywood. So it makes sense that I'm interested in LightWave.

Anytime there is an article appearing about LightWave, in any magazine, I make sure to get a copy and read it. The problem is that there is never enough LightWave articles out there to satisfy me.

For a while now, I have felt that there has been a growing need for more LightWave information. The few LightWave articles that appear in magazines haven't been fulfilling my needs, and I was hoping someone would come out with an all-LightWave publication. Enter LightWavePRO.

Little did I know what I was in for.

When Jim Plant, publisher of *Video Toaster User*, contacted me and

continued on page 2

LIGHTWAVEPRO

AN AVID PUBLICATIONS NEWSLETTER

Publisher.....Jim Plant

Editor.....John Gross

Managing Editor.....Angela LoSasso

Assistant Editor.....Josh Moscov

Art Director.....Tom Twohy

Contributing Writers.....Eric Barba

.....Grant Boucher

.....Ken Stranahan

.....Mark Thompson

Editorial Offices:

Avid Publications

273 N. Mathilda Ave.

Sunnyvale, CA 94086

In Issue Number Two:

All about Objects!

- How to Design Objects
- Where to Find Objects
- How to Use Objects

...And Much More!

"Welcome to LWP" from pg 1

told me about the idea for an all-LightWave newsletter, I was very excited. I wanted to be a part of it because it is exactly what I want to read. We hope it is exactly what you want to read as well.

So, what can you expect from LightWavePRO?

A lot.

Each issue will cover different aspects of LightWave. You'll learn tricks and tips from LightWave experts throughout the country. You'll follow along with tutorials that will teach you the ins and outs of your favorite 3D package. We'll have product reviews, current news, and inside information that will keep you up to date with the latest buzz in the world of LightWave computer graphics.

With each new issue, LightWavePRO will concentrate on particular sections of LightWave. Expect future issues devoted entirely to modeling objects correctly or using surfaces, textures and image maps to achieve realism. Or how about an issue that deals with lighting, or new Modeler macros? You'll see all of this and much more in the months to come. LightWave is becoming more and more popular and this publication is going to keep you ahead of the game.

This premiere issue covers the new object deformation features of LightWave 3.0 (or, if you have an Amiga 4000, LightWave 4000). You'll learn all about using Bones and displacement maps to modify your objects in new and interesting ways. By the time you finish this issue of LightWavePRO, you'll be ready to "deform with the best of them."

Remember: We want your feedback. Is there a subject or tutorial you really want to see in a future issue? Drop us a line at our offices or online on CompuServe (send E-mail to either myself at 71740,2357 or to Jim Plant at 72242,1623).

Thanks for joining us for this first issue, and we hope to see you come back for more!

LWP

John Gross
Editor

A Look at Displacement Mapping

By John Gross

In previous versions of LightWave, I used bump maps for wavy or bumpy illusions. Unfortunately, the visual quality just wasn't good enough because the surface was never actually bumpy—it was shaded to appear bumpy. With the introduction of LightWave 3.0, a new tool called displacement mapping allows surfaces to actually change shape during an animation. Need to have waves ripple against the hull of a boat or a flag blow in the breeze? This is where a displacement map can help.

Displacement mapping is actually rather simple to use—you just need to know some basics. First of all, you cannot displace surfaces of an object by themselves; the object must be displaced as a whole. For this reason, the Displacement Map button is located in the Objects panel, not the Surfaces panel.

There are five different ways to displace an object. The first three are different types of image maps—planar, cylindrical and spherical. The other two methods are Ripples and Fractal Bumps. Sound familiar? These are the same options that you had for bump mapping a surface under LightWave 2.0; LW 3.0 has the exact same bump mapping options with the addition of a cubic image map.

For a displacement map to work properly, it's important to have a lot of polygons in your object; otherwise, your displacements will not be smooth. (How often have you seen waves that were pointy?) The best way to prepare an object for displacement mapping is to load it into Modeler and use the Subdivide option (located in the Polygon menu) to create more polygons. The polygons need to be tripled first. If your object is not a flat plane, you will almost always want to use the Smooth Subdivide option.

How much should you subdivide? It depends on how much you are going to displace the object and how close you want to get to it. Usually, some experimentation is required.

Once your object has been subdivided and brought back into Layout, it's time to start displacing. All of the different displacement textures have one thing in common: texture amplitude. This simply refers to how far, in meters, the surface of the object will displace.

After choosing an axis and size for your image, set the Texture Amplitude. For best results, start with low values. You can see the results of the value when the object redraws in the Layout screen. If your amplitude is set too high, instead of proper displacement, your object will look like a rat's nest.

Remember that when you use images for your displacement, the luminance (or brightness) values of the image determine the amplitude of the displacement. Solid white values in the image will displace the object the full amount of the Texture Amplitude value. Solid black values will not displace the objects at all; values in between white and black will displace accordingly. For example, 50 percent gray will displace 50 percent of the Texture Amplitude value.

Ripples and fractal bumps, the other two types of displacement maps, have a few additional values to input. Fractal bumps allow for a frequencies setting which determines how many "patterns" of bumps will be used. I recommend values between 3 and 16. The higher the value, the more "bumpy" your object, but the longer the render time. For fractal bumps, I also suggest starting with a Texture Size value of approximately one-tenth of the object size. To determine this, simply press the Automatic Sizing button while in one of the image map textures; then select Fractal Bumps. The size will carry over.

The Ripples texture has wave values that must be input. Wave Sources is simply the number of centers of waves (a value of 1 will have ripples emanating from one spot only). Wavelength determines the spacing between the waves, while Wave Speed determines how fast the waves move.

If you need to have looping waves moving for an animation, use this simple formula to determine the Wave Speed value: Take the Wavelength and divide it by the number of frames you wish the waves to loop in and enter that value for the Wave Speed. I generally use the object's size for the Texture Size when using Ripples.

Ok, now that you have been introduced to displacement mapping, how can it be applied to your work? To get started, turn the page and follow the tutorials by our outstanding contributing writers Mark Thompson and Grant Boucher.

John Gross is the Editor of LightWave Pro. John currently works as an animator for the NBC television series seaQuest, DSV.

Displacement Mapping: What's It All About?

BY MARK THOMPSON

Along with the new LightWave 3.0 release comes some wonderful new object deformation functions. In keeping with the theme of this premiere issue, this article will focus on one of the truly powerful features of the LW 3.0 release—displacement mapping.

Working in a concept similar to bump mapping, instead of merely giving the appearance of bumps, displacement mapping actually creates them. Or in a nutshell, it is a method of physically deforming an object's geometry based on a textural specification. That texture could either be procedural, such as LightWave's ripple texture, or image-based. There are a couple ways this can be accomplished.

For example, Pixar's photorealistic RenderMan breaks all its primitives (including polygons) into tiny sub-pixel fragments which are individually rendered. In this case, displacement mapping shifts the location of each of these fragments according to the texture.

LightWave, on the other hand, renders polygons as a whole singular entity, and therefore, the smallest unit that can be displaced is the polygon vertices. This method has faster rendering speed, but requires higher polygon densities for detailed displacement maps. Consequently, LightWave is better suited toward larger, less-detailed maps.

For example, you probabaly wouldn't want to use the pits in a golf ball for LightWave displacement mapping, but creating an uneven, rough terrain would be a great application. In fact, watch Steven Spielberg's new TV show *seaQuest, DSV*, and you will see some examples of LightWave-generated terrain that was created using this feature.

Unlike other textures in LightWave, displacement mapping acts upon the entire object rather than a specific surface. At first this may seem like a limitation, but since it is so well-suited as an animation tool, it is better that the whole object is affected. While it certainly can be used like a surface attribute, its object animation capabilities are what really make it shine. I will give two animation examples that would otherwise be rather difficult without displacement mapping.

Displacement Mapped Shark

This first example shows how to make a fish swim. Without displacement mapping, this would be a somewhat arduous chore of setting up multiple morph targets of the fish in various states of body bend. *(Editor's Note: To animate fish, you can also use Bones. For a tutorial, see Eric Barba's* Bones, and a Shark Named Bruce.*)* I have done this with a shark and needed a minimum of four different versions of the object to morph between to create acceptable motion. I might add that a good deal of experimentation was required to get it right.

With displacement mapping, this task is a snap. First, create a vertical, gradient brush map that goes from black to white to black. Then use that image as a planar displacement map on your fish object. Use the texture axis that is perpendicular to the side of the fish. My shark extends along the X axis from tip to tail, so a Z axis map was used. I sized the texture to be just a little bit larger than the object (which was 2.7 meters long). The pertinent settings used were:

Displacement Map: Planar Image Map			
Axis:	Z		
Size:	3m,	3m,	3m
Velocity:	0.13	0	0
Amplitude:	0.25		

The velocity controls how fast your fish will wiggle and the amplitude sets how much it will be displaced in meters. With this done, it is easy to create a school of fish.

Use the new Object Clone feature and make as many copies as you like. Parent each clone to the original and move it to a different location. Then modify the texture center for each displacement map so that all the fish don't swim or wiggle exactly the same way. You'll find that this gradient brush map can be used for a number of displacement applications. Just to experiment, I applied it to the DC-10 object to create the latest in aviation from "Snake Airlines" (see Figure 1).

The Green Hills of Earth

I am quite fond of the next example. It is a simulation of a field of tall grass blowing gently in the wind. This example would be rather difficult to duplicate with most other 3D packages. I dreamed up the technique when an Imagine user on one of the electronic networks queried how to model such a scene.

Grass requires complex modeling. Unlike a neatly mowed lawn, a grassy field cannot simply be mod-

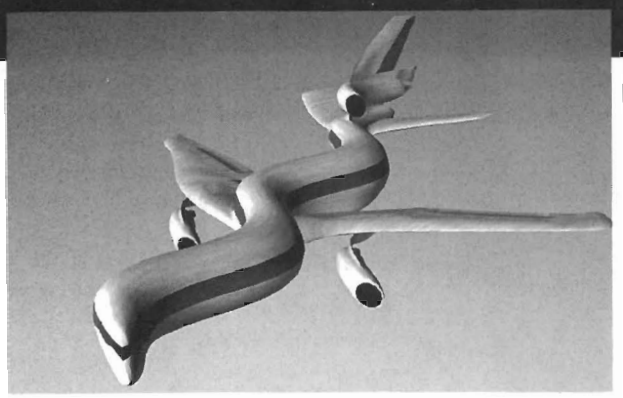


Figure 1

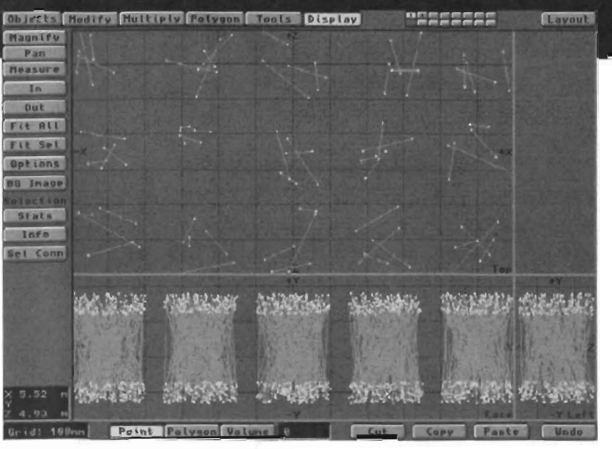


Figure 2

Figure 3

eled with a single, flat polygon with a good grass texture. Individual grass blades really need to be visible.

I have heard of people trying to tackle this problem by using bunches of transparency-mapped polygons, each made to look like a patch of grass. But this technique heavily limits your camera motion (it's very view dependent), and when many transparent surfaces overlap, rendering time increases.

Another method is to model the blades individually as tall, thin pyramids. But considering the number of blades needed to fill a field, this many polygons would rapidly become a burden both in terms of memory usage and rendering time.

The most successful grass simulations I have seen were done by Lucasfilm (now Pixar) back in 1984 using particle systems. Particles have two key advantages over polygons for this type of effect: They render much faster and they consume less memory. Since LightWave supports both one and two dimensional particles (i.e., points and lines), this method seems the most viable.

To model a field of 2D particle grass, go into Modeler and make a single blade by creating a point at the origin and another just above it in the Y direction. The location of the second point will determine the height of your grass (I used 300mm or about a foot). Hitting *P* will create the blade. While your object is still simple, hit *Q* to assign it a surface name.

Now duplicate this single blade into a field. The new Array tool works nicely. Enter a dimension of 100 in X and Z, select Manual offset, and use 100mm for the X and Z offset. This generates 10,000 blades in nice, neat, totally unnatural-looking rows. The new Jitter tool is perfect for adding some randomness to your grass. A jitter of 100mm in X and Z, and 30mm in Y yields a good random look. Now save it, and your model is complete. See Figure 2.

Assigning surface attributes will vary with the look you are trying to achieve. Essentially, textured color and diffuse are all that is needed. In Figure 2, I used a 2mx2mx2m fractal noise texture to modulate diffuse between 20 and 60 percent. A brush map was used for color texturing, but a small (10mm), fractal-noise texture works just as well. However, these settings alone are not enough—the grass will still look two-dimensional. Some shadows are needed.

If you try and trace the shadows, however, you will find that no shadows are cast. That is because 1D and 2D particles have no volume as far as the shadow tracer is concerned. Fortunately, the new shadow mapping feature allows particles to cast a shadow, not to mention a dramatic increase in rendering speed. This just means you will have to illuminate your grass with a spotlight rather than a distant source. At this point, some sort of ground is needed. A big, flat polygon with a texture similar to the grass will do fine or you can make it look more like dirt. Whichever you choose, it will be mostly hidden by the grass. Its main purpose is to serve as a recipient of the grass shadows.

However, what really makes this tall grass interesting is making it blow in the wind. This is where displacement mapping comes in. By using a procedural ripple displacement that starts at the top of the grass and falls off to zero at the bottom, this illusion is made extremely simple and quite effective. The following settings worked well for my example:

Displacement Map: Ripples			
Size:	10m	0.31m	10m
TextureCenter:	5m	0.31m	5m
Falloff:		0	
300% 0			
Amplitude:	0.1m		
Wave Sources:	3		
Wavelength:	2.5		
Wave Speed:	0.025		

These settings cause the top of the grass to move with a rippled wave motion, while the bottom stays completely stationary as if it were being held in place by the ground. (A nice touch would be to construct each grass blade out of multiple segments allowing them to "bend" in the wind, rather than merely tilt as they do with this simulation.) Unfortunately, the falloff parameter for all the textures is linear which means that no amount of grass blade subdivision will allow them to bend. But as you will see, even the straight blades outlined in this simulation look quite good because of their close proximity and random orientation (see Figure 3). And not only does it look good, but it renders quite quickly as well. Figure 3, including the windmill and fence, took just about 10 minutes to render.

A few notes about the grass project. First, the model produced a grass patch that was only 100 square meters. This was fine for my example (which had a constrained camera view), but for larger areas, you may want to either create larger patches or use several smaller ones. Also, as the grass becomes more distant from the camera, the density required for a good illusion is much lower. So when you create your field, you should place denser grass patches in front of the camera, and much sparser patches in the distance.

As you can see, displacement mapping is a truly powerful new animation tool and there are many other uses. When combined with the new Save Transformed feature, it can even be used as an advanced modeling tool.

If you come up with any unique uses for displacement mapping that you would like to share, drop me a line and I might use it (and give you credit) in a future article. Also, if there are any advanced or high-end techniques that you would like to hear about or effects you are having trouble creating, feel free to contact me.

Mark Thompson is president of Radiant Image Productions, a 3D animation and special effects production house.

Send questions or comments to: Radiant Image Productions 51 Derry Street Merrimack, NH 03054 or email to:

mark@westford.ccur.com.

Displacement Mapping

BY GRANT BOUCHER

With the introduction of LightWave 3.0, many new features and tools are available to animators. One such tool is displacement mapping. *(Editor's note: For a brief introduction into the capabilities of displacement mapping, see* A Look at Displacement Mapping *by John Gross)*.

Let's explore some excellent examples of using displacement mapping to create extremely advanced animation techniques that would be difficult, if not impossible, otherwise.

Image Mapped and Fractal Waving Flags

In these examples, we will build a flat plane object with many polygons for use with two different waving flag techniques. The first is a simple, waving ridge using an image map and the second will employ a fractal-noise texture to give us a more organic, windblown effect. We will also expand on these examples with some additional professional techniques.

Building the Flag

In Modeler, create a plane suitable for displacement by using the Box tool in the Object menu. Enter the following settings (in meters) in the Numeric requester:

	Low	High	Segments
X -	0.50	0.50	30
Y -	0.50	0.50	30
Z -	0.00	0.00	1

If your flat plane of polygons is not made of triangles, go to the Polygon menu and select Triple. Everything to be displacement mapped should be made of triangles, and while 1,800 polygons may seem small for those of you experienced with the joys of distorting LightWave objects, this amount works great for these examples. If more polygons are needed, set the segments settings (see first table) a bit higher (be careful not to exceed LightWave's internal object limit of 65,000 polygons per object). Note: If you use too few polygons, you will see the squared-off edges of your displacing polygons. (This is very unprofessional unless it's the effect you want). If you use too many polygons, mathematical, hard images will appear as raised crosses in the Fractal Flag example I detail later. This is analogous to what you see in a Fractal Noise texture map if the contrast setting is too high. At this time, there is no way to adjust the virtual contrast of a Fractal Noise Displacement Map.

While in the Polygon menu, check your normals and use Flip if they are facing away from the camera (the +Z direction). In my test, the polygon's normals were oriented in the -Z direction, or towards the camera, when first created so there was no problem. If you are unsure of whether your flag's polygons are facing in the correct direction, turn on the Double Sided feature in LightWave's Surface panel.

Surfacing the Flag

At this time, you can apply a new surface ("Flag" for example) to our entire object in Modeler or export the new object directly into LightWave and rename the default surface to Flag. Either way, we are ready to surface the object.

Surfacing is easy because the only settings needed are Smoothing and whatever texture or image map we desire on our flag. Load an image or framestore of your company logo or the Video Toaster logo. If you have an image sequence (either framegrabbed live video or a previously rendered animation) available on your hard drive, you can place full-motion video or animations on the flag for very professional and exciting flag effects.

Displacement Mapping the Flag

For this first example of animating the flag, we will use an image map. The image map to use is very easy to build in ToasterPaint (make sure to save your

Displacement Mapping Tips & Tricks

- Displacement maps are not saved as part of an object; they are saved as part of a scene. Use *Load From Scene* if you wish to load an object with a displacement map attached.
- The amplitude of a displacement map is how far (in meters) the object will be displaced from the original *plane* of the object.
- For displacement maps to work properly, the object must be subdivided into many polygons.
- When using images for displacement maps, remember that the luminance values of the image determine the amount of displacement. Bright values are more, while darker values are less. Solid white will displace the full amplitude entered while darker values will displace relatively less amounts. Black will displace nothing.
- Cloning an object will clone any displacement maps associated with that object.
- Displacement maps affect entire objects as opposed to individual surfaces. If you want part of an object to be non-displacement mapped, that part must be saved as a separate object.
- When using gradient spread images for displacement maps, you can easily get by with a one-pixel-wide (or tall) image, assuming that the gradient spread is linearly constant.
- When using gradient spreads as moving maps, make sure to have the ends of the spread match so you do not get a "jump" in the animation.

—J.G.

LWP

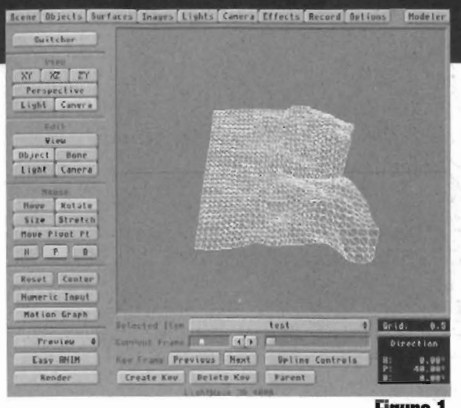


Figure 1

Result of Fractal Bump Displacment Map

Result of Planar Image Displacment Map

object and/or scene file if you have to leave LightWave). Select black and white as the two end colors of the palette range requester in the upper right area of the main control panel. Enter the Transparency/Warp control panel and select a vertical gradient by clicking on the button that looks like a soda can standing up. Leave the gradient highlight bar in the middle and return to the main control screen. Select the Box tool and the Filled tool and draw a square box on the paint screen. You should see it change from black to white to black or white to black to white, depending on your initial settings. If you see one solid color, make sure you are still in Range Mode in the Mode menu.

Cut this brush out with the scissors (box-shaped scissors work best) and save the brush in your LightWave Images subdirectory. If you end up with some black bands around your brush map, use an Image Processing program like Art Department Professional or ImageF/X to crop away the unwanted borders. And while you are there, convert the 24-bit RGB data to 8-bit gray to save disk space and memory usage in LightWave.

Once back in LightWave, (and after reloading your object or scene, if needed) go to the Images panel and load your gradient image. Notice that the thumbnail shows your gradient spread.

Head to LightWave's Object panel and enter the Displacement Map requester by clicking on the T button. Once inside, leave it on Planar Image Map, but select the Z axis for mapping. Select the gradient Image as the Texture Image. Now enter the following settings:

	X	Y	Z
Texture Size:	0.25	1.0	1.0
Texture Amplitude:	0.075		
Texture Velocity:	0.02	0.0	0.0

The Texture Amplitude represents the farthest the object will displace along the Z axis (because we chose the Z axis for the map). The Texture Velocity is the direction and speed the image map travels in relation to the object.

Orient your flag in the Layout to get a good look at it. Make a wireframe preview animation of the first 90 frames or so and notice how simply we created a very primitive, waving flag effect. Render any test frame to see how your texture or image map realistically shifts and changes with the flag.

Anchoring One End of the Flag

The preceding flag is good for a start, but the right and left edge animate freely, as if they are flapping in the middle of space with nothing holding them down—hardly realistic.

Go back into the Displacement Map requester on the Objects Panel and enter the following settings:

Texture Center:	0.5	0.0	0.0
Texture Falloff:	100	0	0

Render a wireframe preview.

We set the center of our texture to the rightmost (+X) side of the flag and told the texture to fall off, or fade away, 100 percent for every meter of distance away from the center of the texture. In other words, the displacement mapping effect gets weaker and weaker the farther we move from the texture center. It is now strongest at the far right +X edge, and is completely gone by the time it reaches the far left or -X edge. Texture falloff is related to the texture center only, not to texture velocity. The ridges continue traveling across the flag, left to right, building in intensity along the way. Add a flagpole or ship's mast and the effect is complete.

Getting Fancy: The Windy Flag

You may still be dissatisfied with the look of the flag, especially if you are one of these so-called "organic" animators, questing to create anything that looks and feels like real Mother Nature (like me). Here are some new settings that should really get you pumping.

Go back to the Displacement Map requester and change the Texture Type to Fractal Bumps. Enter the following settings:

Texture Size:	0.25	0.25	0.25
Texture Velocity:	0.02	0.0	0.01
Frequency:	3		

Your previous settings for Texture Falloff, Texture Amplitude, and Texture Center remain the same and should not have been lost when you changed from Planar Image Map to Fractal Bumps. Create a wireframe preview, but this time edit the flag's rotation to 40 degrees on the Pitch for a better look at the detail. Render a test frame and I think you'll be very impressed with your results.

From previous examples, you know why we used an X velocity, but do you know why I also had you enter a Z velocity? Try a wireframe preview with just the X velocity. You'll notice that the bumps, while very organic-looking, just seem to shift through the flag from left to right, with no wind-like undulation whatsoever. The Z velocity causes the fractal noise texture pattern to travel through the flag in two directions, left to right or X, and from back to front or Z. Since fractal noise is a three-dimensional texture formula, it changes in all three directions randomly. If you force the texture to come towards you, or through the flag material, the changes in the Z direction add just enough variation to make it look like real wind is responsible for the waving flag's motion (See Figure 1). The same principles apply to techniques for ocean waves and fire.

Final Flag Ideas

Any image map can be used for displacement mapping, although only the image's grayscale or luminance data is relevant (i.e., white is greatest amount of displacement, while black is no displacement, and shades of gray ramp in-between). If you are not using an image map as both a texture map and a displacement map, convert the image map to a grayscale, or 8-bit version with your favorite image processing package to conserve memory. You might also want to adjust the contrast or brightness settings while you are there to give a more pronounced or subdued effect to your displacement mapping.

Look into texture map collections like Bill Wiatroski's Textiles from Mannekin Scepter Graphics or Leo Martin's Pro-Textures from Merlin's Software for all sorts of fascinating and unique displacement maps.

LWP

Grant Boucher is a freelance LightWave animator in Orlando, Fla., whose clients include Nickelodeon Studios/MTV Networks and WOFL-FOX 35/Orlando among others.

Swimming Pools For All Occasions

By Grant Boucher

This article will deal with water, specifically with the many different ways swimming pools can be created with LightWave 3.0. We will look at everything from elaborate raytraced versions to quick and dirty wave effects. While not every water possibility is covered here, hopefully, this article will solve many of your most difficult water-related problems.

Building Our Simple Swimming Pool

Make a Box in the Modeler with the follow numeric settings (in meters):

	Low	High	Segments
X	-5	5	1
Y	-5	0	1
Z	-5	5	1

Select the two rightmost bottom points in the Face view (5, -5, -5 and 5, 5, 5) and move them up to about Y = -2. This should create a slanted bottom and sides for the swimming pool. Select the top polygon of the box and cut and paste this polygon into another layer. Apply the surface name of *Water* to this lone polygon. Return to your main pool object, flip all the polygons so all their normals face inward and apply the surface name *Pool Walls* to this object. Export both objects to LightWave as *Pool Walls* and *Pool Water*, respectively.

Surfacing the Pool Walls

The easiest way to create underwater reflections on the pool walls is to use LightWave's underwater texture. For a realistic effect, remember that less light reaches the bottom of the pool then reaches the upper walls. To add a touch of style, you may want to add tiles to the pool walls above and below the water line.

Now, to achieve the darkening effect, we want to use a Falloff of some kind; however, we can't combine the underwater texture as a surface with the tile as a surface. How can we have our reflections and tile them too? Set the color and texture nature of the pool walls with normal surface map settings, while using the underwater procedure with luminosity mapping to create our underwater reflection highlights. Diffuse mapping with falloff can cause whatever surface the pool walls have to get darker the farther down we go. Try the following settings:

Surface Color	0	200	200
Luminosity Map	0%		
Texture Type	Underwater		
Texture Size	10	1	10
Texture Value	33%		
Wave Sources	4		
Diffuse Mapping	0%		
Texture Type	Planar Image Map		
Texture Axis	Z		
Texture Image	A square white brush		
Texture Size	10	5	10
Texture Falloff	0	10	0

Move the pool water object out of the scene temporarily and render a few tests of the pool walls texture. Add your own tiles or texture attributes to the pool walls surface color.

Surfacing the Pool Water: A Quick and Dirty Method

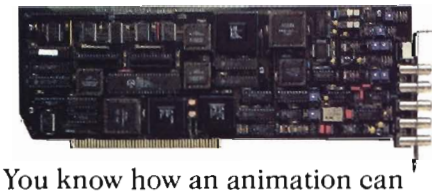
Assuming you're not interested in watching waves lap up and over things (that's the next example), a simple ripple bump map can do wonders for a swimming pool. Enter these settings:

Surface Color	0	200	200
Specular Level	100%		
Glossiness	Maximum		
Reflectivity	33%		
Reflected Image	None		
Transparency	100%		
Color Filter	On		
Smoothing	On		
Bump Map:			
Texture Type	Ripples		
Texture Size	10	1	10
Wave Sources	4		

In the Effects panel, turn Solid Backdrop off so we are using the default environmental settings for Zenith, Horizon, Ground, and Nadir. These will work just fine this example (remember that pool and ocean water get much of their color from the sky above them).

Put your Pool Water object back into place at the top of the Pool Wall and render a test. The pool may not look totally realistic, but it is about as good as

Save Your Animation From Being Eaten Alive.



You know how an animation can take on a life of its own. Sometimes it takes forever. Or it costs too much. Or a tape machine mistakes it for lunch.

The DPS Personal Animation Recorder™ solves these and other animation-production problems. For just $1,995, it gives you the reliability and capabilities of systems costing thousands more.

A plug-in AMIGA® card, the Personal Animation Recorder functions as a single-frame recording deck. With it, you can digitally record your animation onto a dedicated hard disk* and play it back in real time. Which means you can create 3-D animation without the expense and aggravation of tape decks. The Personal Animation Recorder will even genlock to your system.

Because the Personal Animation Recorder operates in a totally digital environment, you won't be bothered with the time base error, jitter, skipped frames, or botched edit points you encounter with traditional animation recorders.

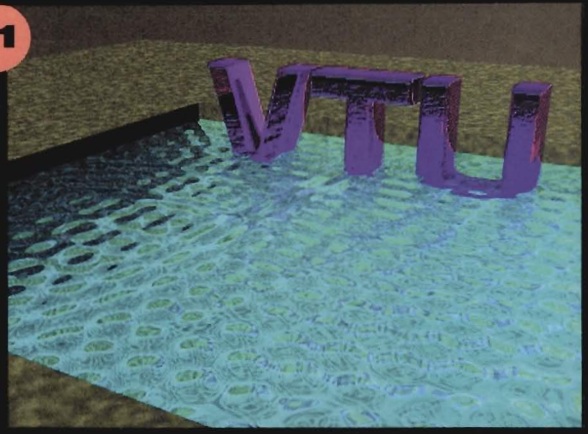
Since your animation is recorded in a component digital 4:2:2 format, you can produce an infinite number of first-generation tape copies. Plus, the Personal Animation Recorder features outputs for true component analog video (Betacam®, MII®), composite and S-Video (Hi8®/S-VHS).

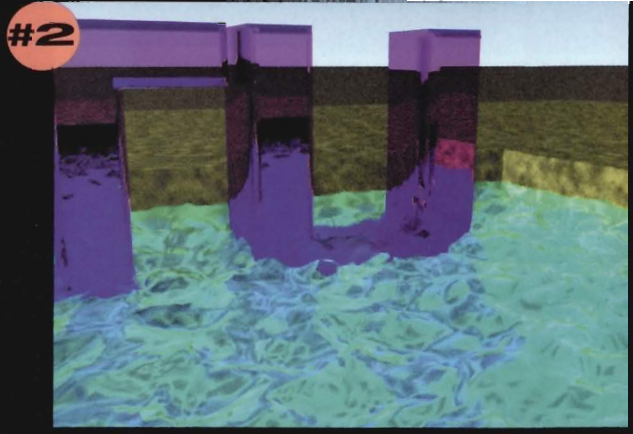
Rescue your productions from the jaws of traditional animation systems. Produce quality animation for a fraction of the usual cost with the DPS Personal Animation Recorder.

DIGITAL
PROCESSING SYSTEMS INC.
If you want to look your best

In the U.S. call (606) 371-5533 Fax: (606) 371-3729 In Canada call (416) 754-8090 Fax: (416) 754-7046

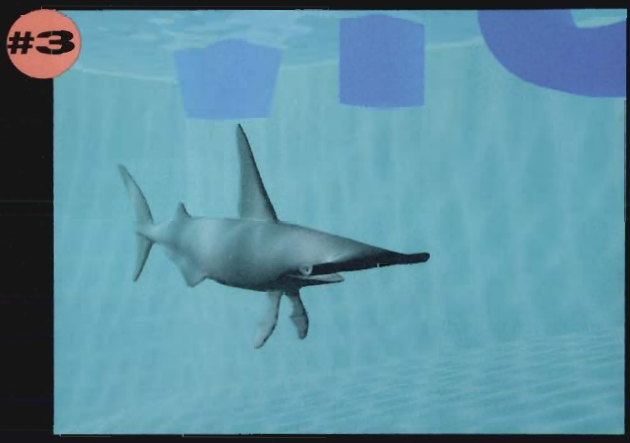
*Hard drive not included DPS Personal Animation Recorder™ is a trademark of Digital Processing Systems, Inc. AMIGA® is a registered trademark of Commodore-Amiga, Inc Hi8® and Betacam® are registered trademarks of Sony Corp. MII® is a registered trademark of Panasonic Broadcast





#1 A Fractal Bump Map applied to the pool water surface. Notice the edges along the pool walls are flat.
—Grant Boucher

#2 A Fractal Bump Displacement Map applied to the pool water surface. Notice the edges are "wavy."
—Grant Boucher

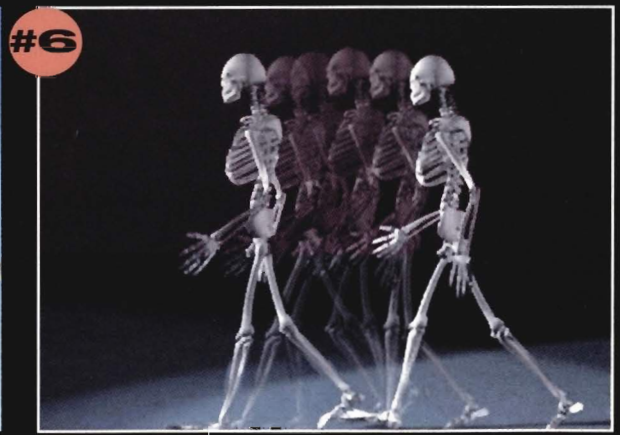


#3 A Displacement Map on the pool surface, plus a Luminosity Map using the same values on the shark skin.
—Grant Boucher

#4 A Fractal Bump Displacement Map applied to the "blowing" grass blades.
—Mark Thompson

#5 A Gradient Image Spread (Displacement Map) applied to the DC-10 object.
—Mark Thompson

#6 Five stages of an object using Bones for a walking cycle.
—Ken Stranahan



#7

#8

#9

#7 A Fractal Bump Displacement Map applied to the Waving Flag object.
—Grant Boucher

#8 A Gradient Spread Image used as a Displacement Map to put waves in a flag. Texture falloff is being used to "anchor" one end of the flag.
—Grant Boucher

#9 A Gradient Spread Image used as a Displacement Map to put waves in a flag.
—Grant Boucher

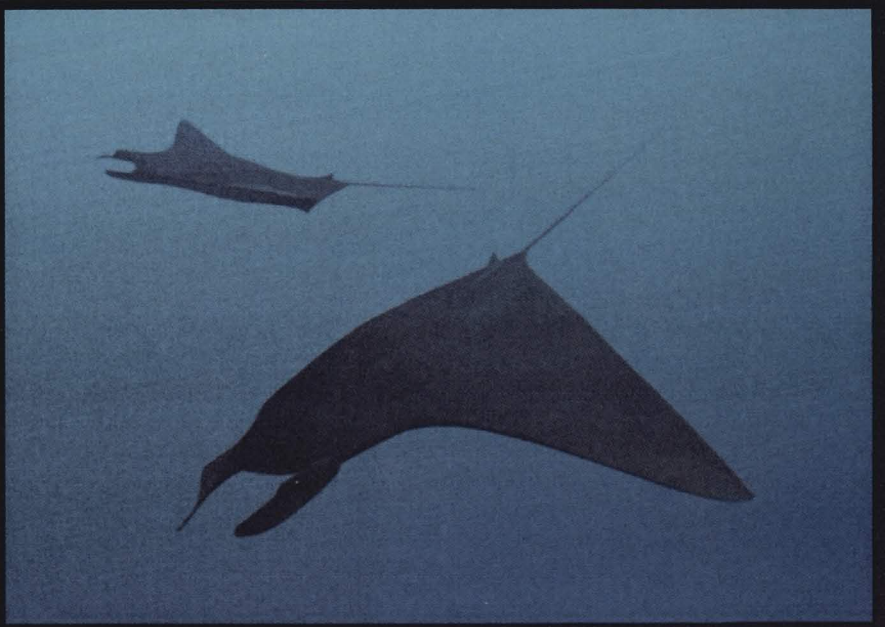
#10 Bones are used in this Shark object to make it swim through water.
—Eric Barba; image courtesy Amblin Imaging

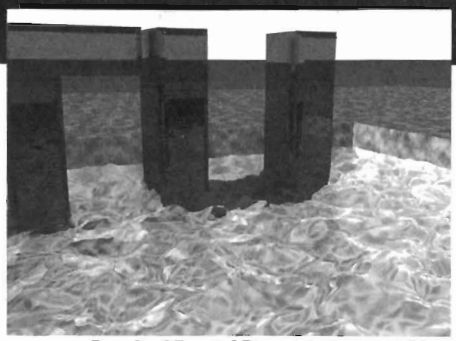
#10

Two bones were used in this object: one to anchor the base, the second to twist the head around.
—John Gross

A Ripple Displacement Map and a Ripple Bump Map were applied to this water surface to achieve realistic-looking waves.
—Richard Payne; image courtesy Amblin Imaging

Bones were used in these Manta objects to make them move realistically through water.
—Eric Barba; image courtesy Amblin Imaging





Result of Fractal Bump Displacment Map

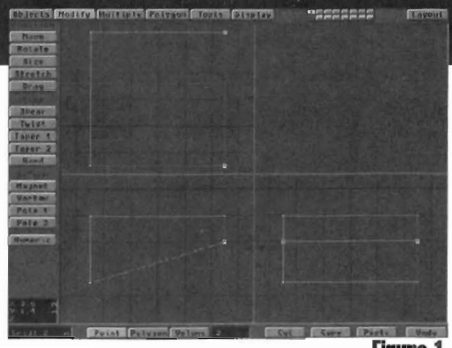


Figure 1

Figure 2

you are going to get without raytracing and/or adding a lot of polygons (see below). You might want to make the Pool Water less transparent (about 75%), if raytracing is totally out of the question.

To Trace or Not to Trace

The above water shortcuts are desirable because they avoid the time-consuming process of raytracing. However, for superior water effects, you should change the Refractive Index of the Pool Water surface above to 1.33 and turn both Trace Refraction and Trace Reflections *on* in the Camera panel. Yes, the rendering time will jump significantly, but the results will be photorealistic, even from close up.

Creating Real Waves

Most animators want real waves—ones that come up and over the sides of the pool and anything in it—that have true depth. Assuming you are either a masochist, a perfectionist, have 40 Toaster 4000s at your command, or are beta testing a Screamer, here's how to do it.

We're going to discuss displacement mapping for wave effects. First you need to either Triple and Subdivide (Faceted) the Pool Water object until it reaches 32,000 polygons, or create a new Pool Water Object that is divided into 180 segments per side and then Tripled into triangles. Replace the old Pool Water object with our new Pool Water32K version and return to the layout. Use the same Pool Water surface settings above, without the ripple bump map. Then proceed to the Objects panel and enter the Displacement Map requester.

Enter the exact same settings for the ripple bump map used earlier in this requester with a Texture Amplitude of approximately 0.066. Render a test or two and see if you like it.

Why doesn't the Ripple displacement map work exactly like the ripple bump map, only taller? Whereas bump mapping fakes the appearance of depth with powerful shading algorithms, displacement mapping actually distorts the object's shape (by moving its vertices) in three-dimensions.

There is one problem: We only want the ripples to distort the polygons in the Y direction, not shift them along the X and Z directions as well. There are two solutions. The first option is to ignore it. Try a Fractal Bump displacement map with a Texture Size of 0.25, 0.25, 0.25 with the same Texture Amplitude as above. This will cause waves to lap around, over, and through any objects which are placed in and around the pool water. However, this technique does not match our underwater surfaces and works better in ocean environments than in closed spaces like pools.

The second option is to render the underwater texture as a normal surface map on a polygon that fills the entire screen, using a separate polygon and scene file. Render a sequence of this texture animating, with white on black surface colors, and save them to your largest hard drive. Then, using the same Pool Water32K object we built before, this time try the Planar Image Map Texture Type (Y axis) in the Displacement Map requester and use the Image Sequence of the ripples we just rendered as your Displacement Texture Map. Now the ripples will only animate vertically and if everything was lined up correctly on your end, the Pool Walls' underwater textures should line up perfectly with those of the Pool Water. Count on rendering times in the hours per frame, but the results should look professional. Save this one for a real moneymaker.

A Compromise Shortcut

Cheating can save you lots of rendering time and might just give you the results you are looking for. Why not use a Pool Water object comprised of 2,000 or 8,000 polygons with the Ripple displacement map procedure above at the same time as adding a Ripple bump map with the same settings to the surface of the water?

Set the Displacement Map Amplitude to around 0.33. The displacement map will give the waves just enough height and depth, while the bump map gives it realistic detail.

Beneath the Waves

What happens when the camera goes beneath the waves? What kind of techniques can be used for underwater realism?

First, the Non-Linear Fog settings in the Effects panel work very well to cloud objects as they move away into the distance. This is more likely to be useful in ocean environments, but a little bit of fog under the pool water might be just what you need.

Set your Fog Color to 0, 200, 200 or your favorite distant water color, and the Maximum Fog Distance to 20 meters. While in the Effects panel, turn on Animated Dither and change the Dither Intensity setting to 4xNormal. This will add a silty look to everything rendered beneath the waves.

Any creatures who are traveling beneath the water will need to have light reflections on them just like the pool walls. Use the exact same underwater texture settings as you used for the Pool Walls surface luminosity, but apply those settings to the surface color.

For example, a shark's skin would have a Surface Color of 0, 100, 100 and an Underwater Texture Color of 200, 200, 200. Make sure you turn World Coordinates on, otherwise, the underwater texture will move with the shark instead of over him. Why not use the underwater texture with luminosity mapping like we did with the Pool Walls? With surface color mapping, the shark will shade naturally based on your light source. If you luminosity map him, his belly will have reflections that are just as bright as his topside.

Diffuse and reflection maps could also be used under the right circumstances. If you want the light reflections to get dimmer as the shark gets deeper in the pool and brighter as he reaches the surface, use the same diffuse map settings as the Pool Walls. Don't forget to turn Double Sided on in your Pool Water surface setting. You won't see the bottoms of the waves without it.

For Crazy People Only

In the previous examples, raytraced shadows cast photorealistic wave shadows on the bottom of the pool and anything swimming beneath the waves. That means you could eliminate the Underwater Texture settings on the Pool Walls object entirely with stunning results. I dare you to try.

It should be obvious that in most cases the ripple bump map versions of pool water should be sufficient. On the other hand, if you have rendering time to burn and need a professional look to your work, add some raytracing. However, it is also possible to physically model real wave effects with LightWave's own built-in procedures. With clever texture mapping of your own, you should be able to improve on the methods described herein.

Grant Boucher is a free-lance LightWave animator in Orlando, Fla. He can be reached on many national 3D electronic bulletin boards or write to him at: LightWave Pro, c/o Avid Publications, 273 N. Mathilda Ave., Sunnyvale, CA 94086.

LWP

Breathing Life into LightWave Objects

by John Gross

Without bones to hold you up, you would lay there like a lump of pudding. Without bones in LightWave, your objects can lay there like, well, objects. LightWave's Bones allow you to reshape an object during the course of an animation without the need for morphing. Bones allow LightWave users to imbue objects with lifelike characteristics, and provide some mini-modeling capabilities in the LightWave Layout screen.

So, how do Bones work? First, of course, you have to start with an object. It's a good idea to use an object whose polygons have been tripled. Bending and twisting an object with non-triangular polygons causes rendering errors to occur as the object's polygons are twisted out of shape.

The whole concept of Bones is rather simple. After loading an object and adding bones, the bones' *rest* positions are determined. Any time a bone is moved from its original rest position, the object (or parts of the object) move along with it. The longer the bone, the more influence it has on an object (in most cases). Bones that do not move help to *anchor* other parts of the object.

To simply demonstrate the power of bones, I will use the Beethoven bust object that is included with LightWave. This object should be loaded into Modeler any non-triangular polygons tripled before modifying it with bones. Once the object is loaded into LightWave, selecting the Beethoven bust as the current object and clicking on the Object panel, then clicking on the Object Skeleton button, will reveal the Bones panel for the bust. The Bones panel may also be accessed by selecting the object, clicking on the Bone button as the current item, then pressing *p* on the keyboard. The Bones panel is where bones are added to an object.

When you first start experimenting, try using only a few bones. We will use two bones for the bust. Once bones are added, they can be named for identification. This is a good idea, especially if your object will have many bones. After naming your bones, it's time to return to the Layout screen to set the bone's rest length, position and direction (which can also be set numerically in the Bone panel).

Upon returning to Layout, a single bone will be seen as a meter-long stick shaped like two pyramids put base-to-base. Don't be confused by the shape. The fat end does not exert any more influence than the skinny end—it's just a reference for the pivot point of the bone. You'll also notice that both the bones are located at the center of the grid and seen with dotted outlines. When a bone has a dotted outline, it is not active and is exerting no influence on the object. Selecting the Bone button (shift-b) allows bones to be edited in the same way objects are, with one exception: Bones have a rest length. Since bones always start out with a one meter rest length, they may be too long or short for some objects. Changing the rest length of the bone determines the range of influence it has on an object. This influence range is a kind of cold-capsule-shaped area around the bone and can be restricted by inputting an influence range value in the Bone panel and selecting the Limited Range button.

After the bone's rest length is adjusted (which can be changed at any time) and its position and rotation determined, two things need to be done. First, pressing *r* instructs LightWave that this position is the rest position of the bone. At this point the bone receives a solid outline indicating it is an active bone. Second, a keyframe should be created for the bone in this position at frame 0. If no keyframe is created, the bone will snap back to its original position; but because you set the rest position, it will start to warp the object.

For our simple bone experiment, the two bones (BottomBone and TopBone) were keyframed at frame 0 with the following parameters:

	Move		Rotate		Rest Length	
BottomBone	-0.4	X	90.0	H	0.8	
	0.155	Y	0.0	P		
	0.0	Z	0.0	B		
TopBone	0.0	X	0.0	H	0.4	
	0.365	Y	-115.0	P		
	0.030	Z	0.0	B		

Notice that the bottom bone is located parallel to the shoulders. This bone acts as the anchor for the base of the bust as the top bone rotates around to tilt the head. In most applications, it would be necessary to have many bones anchoring the base of the statue.

Now, with two bones associated with this object it is easy to see what happens when the top bone is moved around—the head moves along with it. The base of the bust is anchored by the bottom bone and stays pretty much in place. Remember that the longer the rest length of bones, the more influence they will have on the object (if Limited Range is not selected). The top bone can be moved, rotated, sized or stretched and keyframes can be created for the bone to make the head bow and move around. Create keyframes for the top bone with the following Heading rotation values:

Frame #	0	15	30	45	60
Heading value	0.0	45.0	0.0	45.0	0.0

After creating these keyframes for the heading of the top bone, generate a wireframe preview of the 60-

frame animation (make sure to create a keyframe for your camera at a good viewing angle).

It's easy to see that a few bones are all that are needed to give the bust believable character-type motions. Just three to four bones can animate a walking bust—one for each "foot" and a couple for "arms."

To achieve a different level of realism, a LightWave skeleton can be built for an object that more closely represents an actual skeleton. The bones of the skeleton can be parented to one another so that moving one bone will move all the children bones as well. *(Editor's Note: see Eric Barba's and Ken Stranahan's articles in this issue.)*

Of course, character animation is not all bones can be used for. For instance, you may be designing a rocky cliff to be used in an animation. The cliff may need to jut out in certain select areas to support telephone poles. Instead of bringing the cliff into Modeler and modifying it there, you could add a few bones to the cliff object, give them a limited range of influence and set their rest positions in the appropriate places. Moving the bones away from their rest positions will cause the cliff to jut out in the direction of the bone movement.

Here's a few more of the many things you can use bones for: making a cow speak by moving small bones around the lips; making the same cow breathe by having one large bone in the middle of the cow that slightly stretches in width; flapping the muscles of a heart valve; making curtains blow in the breeze; stretching soap bubbles blown out of a mouth; boiling a pot of water; bulging the eyes of a cartoon character; warping a spaceship as it enters warp drive; melting an object into a pool of liquid; and causing a "sinkhole" to appear in a ground plane.

Bones can give remarkable life to your animations. Start off simple and use just a few bones to start deforming your objects. Before you know it, you'll be making full-fledged skeletons right and left.

LWP

John Gross is the Editor of LightWavePRO.

24Tips for working with Bones!

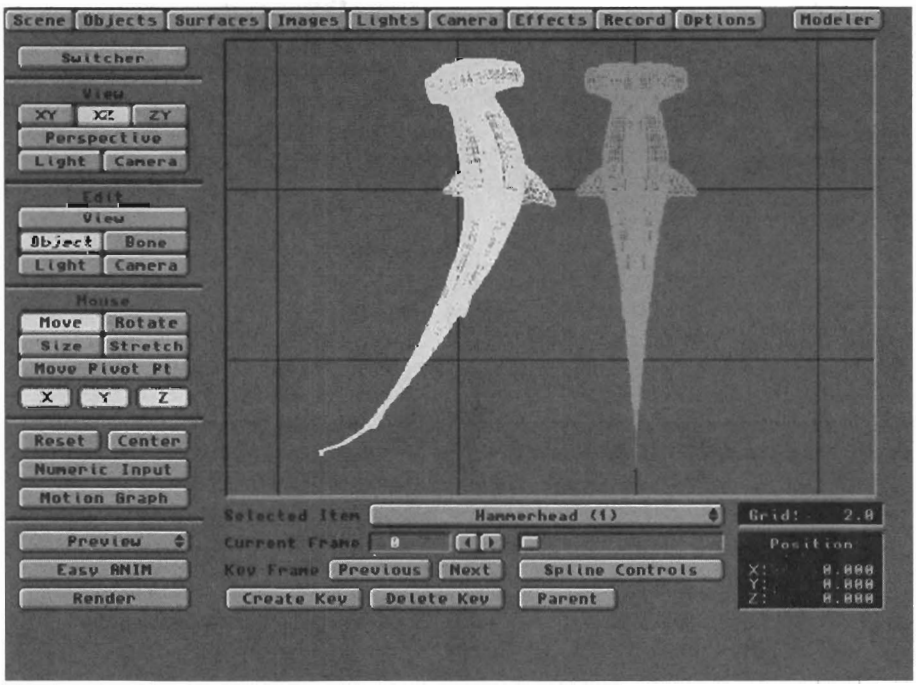
- **Bones affect the points of an object, not the object's polygons. For this reason, it is important to use objects with tripled polygons in order to avoid any rendering errors as the points of a polygon are moved about.**
- **Every bone in an object affects every point in the object. The closer a bone is to any given points, the more those points are affected as the bone is moved. For this reason, placing some bones as "anchors" will cause nearby points to not be affected by the influence of other bones which are farther away. This, of course, changes if you use the Limited Range feature for bones.**
- **The area of influence exerted by a bone is "cold-capsule" shaped around the bone. The skinny and thick ends of the bone do not exert any more or less influence, they are merely there for reference.**
- **The rest length of a bone can be changed at any time. Once changed, the new rest length is the same for all frames of the animation—even if it was changed on the last frame.**
- **After changing the rest length of a bone, there is no need to create a keyframe. New rest lengths are automatically recognized.**
- **Bones always stay with an object no matter where the object is moved. It's as if they were parented, without having to actually parent them.**
- **An object's bones are saved as part of the scene file. They are not saved as part of an object.**
- **Bones may be parented to other bones within the same object. They may not be parented to an object or to another object's bones.**
- **Choosing a bone, and then pressing *p* will present the bone panel and automatically select the chosen bone as the current bone.**
- **It is usually easiest to position all of your bones while they are inactive, create initial keyframes, and then go through the whole bone list and set the rest position for each.**
- **Turning off the Bone Active button in the Bone panel for an object will deactivate a bone until it is selected again.**
- **Do not confuse rest length with size. The rest length determines the area of influence while the size of the bone will actually change the shape of the object. Always change the rest length to set the influence. Sizing a bone is seldom done.**
- **Adding only one bone to an object and not giving it a restricted influence range achieves the same results as if there were no bones. Moving the bone around in this case is the same as moving the object around.**
- **Turning off the Show Bones button in the Options panel will only show the currently chosen bone.**
- **Need to have a lot of bones in your object? Don't worry, there's no limit to the number of bones an object may contain (actually there is a limit—32,000—but who's counting?).**
- **Don't assume that you always need to have bones arranged in a "skeletal formation" for all objects. Sometimes it helps to think of bones as handles sticking out of the back of a puppet. It is also easier to move parented bones if they stay in the same rotation as originally loaded into Layout.**
- **When working with parented bones, always parent all the bones first and then move them.**
- **Moving parented bones is easiest if you turn off the X and Y axes and only move along the Z, then rotate the bone into position. The bones will always move along the Z axis of their parents, so it is easy to move many bones in a string quickly.**
- **Like the grid, bones will never be seen in a rendered image.**
- **To transfer one object's bones to another object, simply replace the first object with the second. The original bones will stay, but may need to have their rest lengths changed to match the new object.**
- **Selecting Limited Range in the Bone panel will cause a bone's influence on an object's points to gradually weaken the farther away the points are from the bone. When you reach the edge of the range value, the bone will have no influence.**
- **Limited Range is best used when using only one or a few bones in an object. If you are using many bones, you will often achieve more realistic motion by not selecting Limited Range.**
- **A bone with a very short rest length (.01 for example) and a limited range will have a spherical influence range, whereas a bone with a long rest length (10 meters for example) and a limited range will have a cold-capsule-shaped rest length.**
- **Modifying an object with bones and then selecting Save Transformed in the Objects panel allows you to save the object in its current state of transformation as a separate object. Different-shaped objects can be saved by saving it transformed at different key frames.**

—J.G.

LWP

Bones and a Shark Named Bruce

by Eric Barba



The Shark on the left has bones, while the Shark on the right does not.

Bones can be used in a number of ways to create motion for many types of effects. In this article I will explain the techniques I used to create the motions for a hammerhead shark for use in the NBC television series *seaQuest, DSV*. I will refer to the shark as Bruce, as this was the name given to the mechanical shark used in the movie Jaws, and I think it's appropriate.

The first step is to model and load Bruce (you can use your own object). I modeled Bruce so he was facing into the positive Z axis (like a vehicle). Next, bones were added. When added, they should point in the right direction at first, so you don't have to rotate them.

Because a shark's movement is most often a side-to-side thrashing, we must first create a bone structure similar to a shark's vertebrae (if it had one). After loading in the bones one at a time, and naming them as we go along. (I named them Shark bone 1, Shark bone 2 etc..), the rest length needs to be set.

I used 13 bones because a very fluid motion was needed, while causing little distortion to Bruce's polygons . The total length of Bruce was divided by 13 to obtain the correct rest length for each bone. As each

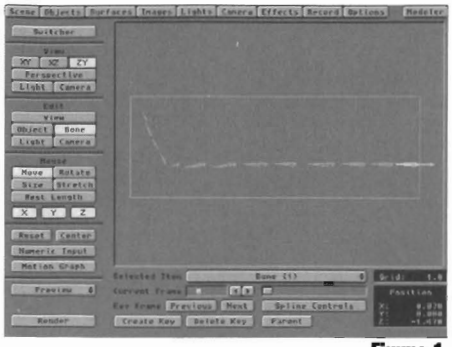


Figure 1

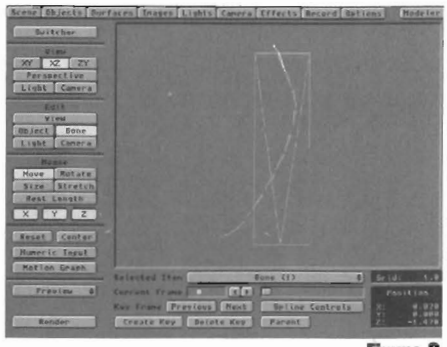
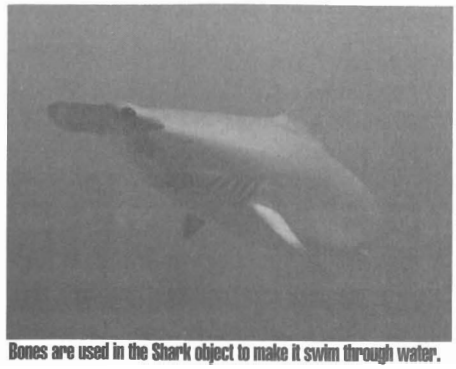


Figure 2



Bones are used in the Shark object to make it swim through water.

bone's length was adjusted, I also parented it to the previous bone (bone 2 to bone 3 to bone 4, etc.). Now you can position them with the tail of the bone pointing toward the tail of the shark.

The simplest way to proceed is to move the first bone into position (all the other bones will remain on top of the first one). The bones need to be lined up along the center of Bruce, both in the Z plane and the X plane. To move the bones along the Z Axis simply disable the X and Y Axis and drag them along. As the bones are lined up closer to the tail, they must be rotated on their pitch to follow the tail properly. As you rotate a bone, any children to that bone will of course rotate as well, so it is very easy to move a bone along the Z axis until its head meets the tail in front of it and then rotate it into place and continue. See Figure 1.

After hitting *R* to set the rest position for each bone, create a keyframe at zero for each bone. Depending on the fluidity of the motion needed, more or less bones may be appropriate. For the LightWave version of Darwin, *seaQuest's* resident dolphin, I have used as many as 24 bones. Other effects can be done with one or two bones, but for fluid character motion, more is usually better.

To ensure good rendering quality, it is also important to note that your object's polygons should be tripled If you don't, you may end up with rendering errors due to non-planar polygons as the object is "boned" about. In cases of severe distortion and or bending, you will also need to subdivide the areas most affected. Of course this will have to be done in Modeler. It was necessary to do this to Bruce, especially in the tail section.

Every bone affects all points in the object (assuming you do not have Limited Influence activated) and the longer the rest length, the stronger the influence. The more bones you have to anchor parts of the object that should not move, the less trouble you will have keeping your object in place. Remember that the shorter the rest length of the bones, and the more bones in the object, the smoother the motion will generally be. You can also limit the range of influence if necessary, but with this many bones in Bruce, it wasn't necessary.

If you do change the bones' range of influence, remember that the shape of the influence surrounding the bone is roughly cold-capsule shaped. Also remember to engage the limited range button in the bone panel. When changing the influence of bones, it generally requires some experimention to get the right look—but this change gives bones much greater flexibility. An example of using a limited range of influence might be if you were going to add bones to the tentacles of a squid. You would not want the bones in the tentacles to affect the main body of the squid or the other tentacles.

The next step was to decide on the motion cycle for Bruce. (I watched actual shark footage to help me prepare). Because we will want a repeatable swim cycle, the first and last keyframes should be very similar (or the same). I didn't want them exactly the same because of the way I keyframed the motion; however, if the last keyframe and the keyframe at zero are the same, and the motion is in the middle of a cycle, (as the tail swings through the Z Axis for example) you should have no problems.

To start the first keyframe, I suggest you develop a sine wave that you will use as a template for the motion. For Bruce, I used Modeler and created a spline that represented the shark's motion at the extreme (i.e., maximum whip). I positioned points that I felt represented the path that Bruce's "vertebrae" would have to take at the extreme, and made them into a spline. I mirrored the spline on the Z axis and then exported them to LightWave.

This template then allowed me to line up Bruce's bones in his first keyframe (see Figure 2). The other keyframes were made by slightly rotating the line of bones along their heading, being careful not to rotate anything else but the heading. The mirrored spline then allowed me to properly line up the bones for the other extreme. The rest was trial and error to control the whip of Bruce's tail and body. I tried to imagine holding a piece of spaghetti and dangling it in the air. The motion the spaghetti strand takes is always naturally fluid, so this made for a good visual model.

With this as a model for smoothness, and using the shark footage, I was able to create the keyframes for a full swim cycle. You will need to select Repeat for each bone motion after you are finished with a complete cycle. This gives you an infinite loop cycle. You might want to save the file as your motion default scene, since bones are not part of an object file (they're saved as part of a scene file). This allows you to Load From Scene to bring in your object with all of its bones intact.

Creating a Motion Path

Next, create Bruce's motion path. Obviously you need to rely on your storyboards or intent of the motion path. To create the path and the illusion of Bruce actually moving through water, I created keyframes at intervals of 30 and Bruce was moved along an equal portion to create the path. After playing with the path to get the right shape, Align To Path can be selected in the object's motion panel. I set Look Ahead Frames to 5. This allowed for a more fluid motion, and a more natural look. In my path, Bruce appeared to anticipate the turns. The Look Ahead Frames may be played with as appropriate to your motion . You could accomplish the same smoothness with keyframes carefully set, but it might take longer. Used properly, Align To Path can be your friend.

If using Align To Path causes your object to face backwards, you've run into an easily solved problem. First load in a null object, save the motion of your object, then load the saved motion file for the null. Then clear the motion for your object, parent it to the null and create a keyframe for the object at zero with the object's heading rotated to 180 degrees. The null will now control the motion of your object, and the object will be pointed to follow your path.

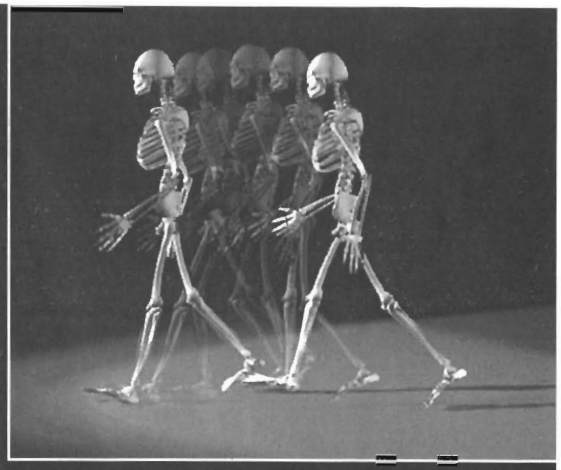
With these steps I was able to create the swim cycle and forward motion for Bruce. It took a little experimenting to get it right but eventually it worked out.

Bones can be a great tool for lots of different effects. With Bones, you have the ability to animate anything from swimming sharks to walking pianos. Load in some bones and start experimenting!

Eric Barba is an animator for Amblin Imaging. He is currently working on Steven Spielberg's television series, seaQuest, DSV.

LWP

Dem Bones!



by Ken Stranahan

By now you've heard of Bones and you just can't wait to make a brigade of ballet dancers leaping across your screen. Well, if you're like me, the first time you use Bones you end up turning your objects into mush. No, Bones isn't easy at first; but with a little work, you can have those dancers doing tour de jetés with realism.

Walk on by

So how do you make a person walk, let alone dance? First, you need to study how people walk. Everybody has a center of gravity. In LightWave terms, it's called a pivot point.

When a person walks, they essentially move their body weight (or center of gravity) forward. Luckily, the feet get smart and prevent them from falling over and landing on their face and the body moves forward in the process. Hey! Grab your camera! You just took your first step!

The key to animating realistic motion is understanding the energy that moves us when we walk. That energy is gravity.

In animation, gravity is an essential component of realistic, human movement. Whether running, jumping or falling, it's important to keep in mind that gravity affects a body in motion. Does the body speed up as it falls? Does it strain from its weight as it moves up stairs, or does it stretch to reach the top shelf? Whether it comes from reading a book such as Edward Muybridge's *Motion Studies* or just watching a videotape, observation is the key to making realistic human motion. So go out and watch people walk. That's really the first step (pun intended).

Skeletós Vertebratus

Now let's move into LightWave. The first thing we have to do is set up the skeleton. Start by loading a human figure. For this tutorial, I will use a one-piece human skeleton object (see Figure 1). If you don't have one, you can make a simple body out of disks. Because we are using bones, there is no need to have separate objects for the arms and legs.

Now, select the Object Skeleton button for your human figure object and add some bones from the skeleton menu. I would suggest adding one bone for each foot and hand, one bone for each arm and leg segment, two shoulder bones, and finally a bone each for the head and torso (perhaps two bones for the torso). Rename the bones something like Left Shoulder, Left UpperArm, Left LowerArm and so on.

When you return to Layout, notice that all of the bones are placed on top of each other in the center of the grid. Before you even think of moving them, first set up the parenting of the bones.

Set up the parents in the obvious order—you know the song "the hand bone's connected to the lower arm bone...the lower arm bone's connected to the upper arm bone." and so on. All of the main parent bones should be parented to the central torso bone.

After parenting the bones you can then move them into position. The trick to positioning the bones is to move the bones that have the most bones parented to them first. For example, since you have a main torso bone, move that into position first. Any bones parented to that will follow. Knowing where to position the other bones is easy. Just look at your body (and a picture of a skeleton wouldn't hurt) to help you understand where the bones should go. Remember to move the main parents first and work down the line.

One common problem is that the parented bones have had their rotations changed by their parents, so they will appear to move in strange directions. They are actually moving relative to their parent's position. The best way to deal with this is to take it one step at a time.

Use the X, Y, Z constraints and the different views until you get it right. This may take some time. A good tip is to move along the Z axis first, because then the bone will move along its parent's length. After the bone is positioned correctly, you should make sure that it will rotate in the correct direction in relation to the body. We don't want any broken arms.

The next step is to change the rest length to fit the body part. It's important that you use the Rest Length and not the Size to affect the bones' final length in the body. Size will affect the object even if you reset the bone. Once the bone is in position, make sure to create a keyframe at frame 0. Always remember to re-keyframe the bone if you need to reposition it.

When all bones are positioned correctly, the final

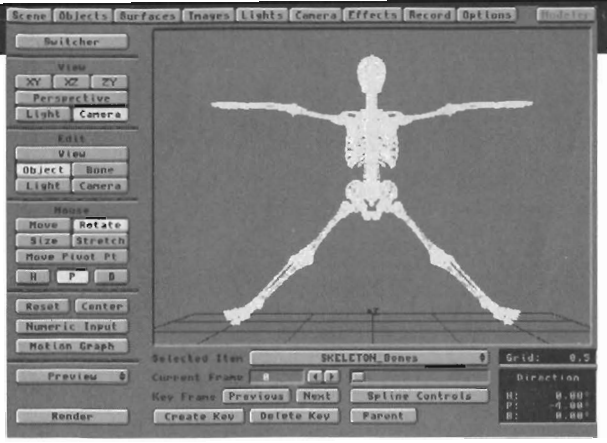


Figure 1

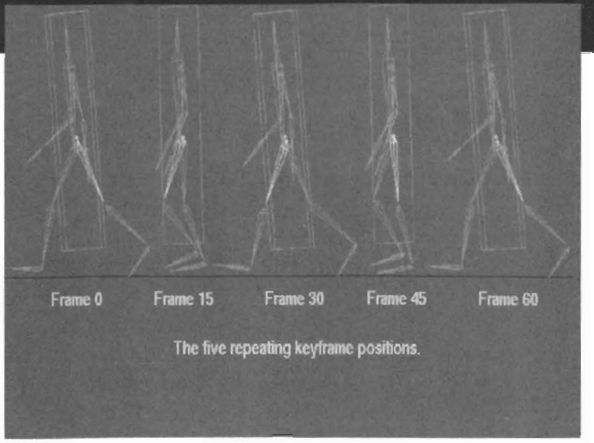


Figure 2

resting position needs to be set for each bone. The rest position is different from a keyframe. The rest position is the position of the bone when it is not affecting the object. As soon as a bone is rotated, sized, or moved from its rest position, the body will be affected. Notice that the bone is drawn with dotted lines until it is given a rest position after which it is drawn with solid lines.

After the bones have been given rest positions and moved a bit, you may notice that the bones can affect more of the object than you originally intended. One way to avoid this is to adjust the bone's area of influence.

This can be numerically adjusted to change the effective range the bone has on the body part. If the legs are too close together, a bone in one of the legs may affect the other leg. The area of influence can then be changed to a smaller area around the bone.

Another way to keep the bone from affecting an area is to add more bones to act as an anchor for part of the body. These bones can be placed and rested, and then never moved. This can also be useful in keeping the limbs from squashing when they are bent, or even sized to emulate muscle flexing. The whole process of adjusting bone influence is no easy task. Spend some time getting used to their capabilities before you get too involved. Once all the bones are positioned and rested correctly, make sure to save the scene. You can then use the Load from Scene function, and load the figure into any animation.

Get up and Go

Now that we have the bones in place, let's make the body move. The first thing to do is to move the center of gravity or pivot point. It should be located in about the middle of the body. After moving the pivot point, there is no need to create a keyframe—it will remember its position. Because the arm and leg bones are parented to the torso bone, it is the obvious place to start our motion. Let's also make the motion a repeating cycle so it's easy to see a wireframe preview. Remember to always try and use as few keyframes as possible.

When the body takes a step it starts falling forward and in turn is moving down and at a forward angle. So now let's simply move the torso bone. Notice that it will move all the bones attached to it. The rear leg bone can be rotated back and the foot placed on the ground. At the same time, the upper leg bone rotates forward to plant the foot. The leg that has the foot planted will have to be moved so the foot is always placed at the same place on the floor. A good trick to keep it positioned correctly is to use a grease pencil and mark the original position right on the screen.

Now that we have a position where the body is in motion, this is a good place to start the cycle. So, let's keyframe all the bones at frame zero. Let's also keyframe them all in the same position at the last frame, 60. Even though the arms and legs will be in the opposite direction, the torso will be in the same position at frame 30—so let's keyframe it for frame 30.

Now, as the body takes its first step it will be planting its foot and the torso will move in an upward direction. When it's been moved up a bit to its apex, the torso should be keyframed in the upper position for frame 15, and again at frame 45. This completes the motion of the torso.

Now all that has to be done is to reposition the arms and legs to finish the motion. Since the legs were at their final position at frame zero, it's easy to switch the position of the legs and re-keyframe them at frame 30. Now that the legs are swinging correctly the walk is starting to come together.

When the body is moving forward, the lower leg bone must rotate back to give the foot clearance from hitting the ground. This too can be keyframed at frame 15 and the other leg at 45. The foot then moves forward to meet the ground. At the same time your arms swing back and forth to counterbalance the motion of the legs. The arms can be keyframed on the same frames as the legs. This will keep the frames that have keyframes on them to a minimum and avoid a confusing scene (see Figure 2).

This first step is just the foundation for the animation. You will have to experiment to get the realistic motion down. This basic motion may sound difficult when reading, but if you follow along and actually try, it will be fairly easy.

Troubleshooting

Unfortunately, you can run into several problems. The first thing to notice is that if you repeat the motion and the body was moving forward, it will jump back to its original position. It's best to make the entire motion without actually moving the body forward. Then, by parenting the body to a null object, you can make the body walk through a scene without having to keyframe the position of each motion separately by simply moving the null object along a path.

If you want a more complex animation there is another problem: Real motion is never really repetitive. There is always a little difference in each movement. To remedy this all you have to do is repeat the keyframes manually. Just go to the motion graph for the object (press *M*, while the object is selected) and take each frame of the first cycle and copy it to each progressive cycle frame.

For example, on a 30-frame cycle, copy frame 15 to 45, 75, 105, and so on. The easiest way to copy the values for one frame is to select the frame on the motion graph, then select Create Key and type in the frame number you want to copy the value to. LightWave will automatically copy the value for the key you have selected into the new key. Once you have your whole cycle in the motion graph, modify each key slightly so there is no true repetitive motion.

Returning to Layout, simply use the null to move the body. You can adjust some of the keyframes of the walking motion to suit each step, even making the person maneuver over or around objects.

After the motion is completed, you may find that the feet may have a tendency to move or bob through the floor. This is because the motion of the keyframes moves in curves. This can be cured by making all the keyframes linear (when the feet are on the ground).

These tips should get you started. Just remember, once you get the basic skeleton set up correctly and saved, it's just a matter of time before you get the motion down. It's not uncommon for an animator to finesse a motion for several days (or weeks!). But once it's done, you can use it in any number of animations. So keep experimenting with Bones and you may be able to create the animations you've always dreamed of.

Ken Stranahan is an animator for Amblin Imaging. He is currently working on Steven Spielberg's television series, seaQuest DSV.

To Get the Most out of LightWave 3D™ You Need

LIGHTWAVEPRO

Subscribe to LIGHTWAVEPRO, the Newsletter for Serious LightWave Animators and receive 12 issues of high-powered features, tutorials and professional tips and tricks.

☐ **#1** I want to receive **LIGHTWAVEPRO**, the Newsletter for Serious LightWave Animators.
Please start my personal subscription for one year (12 monthly issues) at **$72.00**

☐ **#2** I 'm a current **VIDEO TOASTER USER** subscriber, send me **LIGHTWAVEPRO** at the special subscriber's rate!
Please start my personal subscription for one year (12 monthly issues) at $48.00

☐ **#3** I want to receive both **LIGHTWAVEPRO** *and* **VIDEO TOASTER USER**, at the Special Package rate!
Please start my personal subscription for one year (12 monthly issues x2) at **$84.00**

Do you own a Video Toaster? Yes ☐ No ☐

Do you own an Amiga computer? Yes ☐ No ☐

Model: (please specify)_____

☐ **Check Enclosed**

☐ **VISA** ☐ **MasterCard**

Card No. ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Exp. Date ☐☐-☐☐-☐☐ **Signature** _____

NAME .....**TITLE**.....

ADDRESS.....

CITY.....**STATE**.....**ZIP**.....

TELEPHONE ( )

Mail to: AVID Publications 273 N. Mathilda Sunnyvale, California 94086 (408) 366-8220 Fax (408)725-8035

AVID Publications
273 N. Mathilda
Sunnyvale, CA 94086

BULK RATE U.S.
POSTAGE
PAID
CUPERTINO CA.
PERMIT NO.391