THE NEWSLETTER FOR LIGHTWAVE 3D ANIMATORS

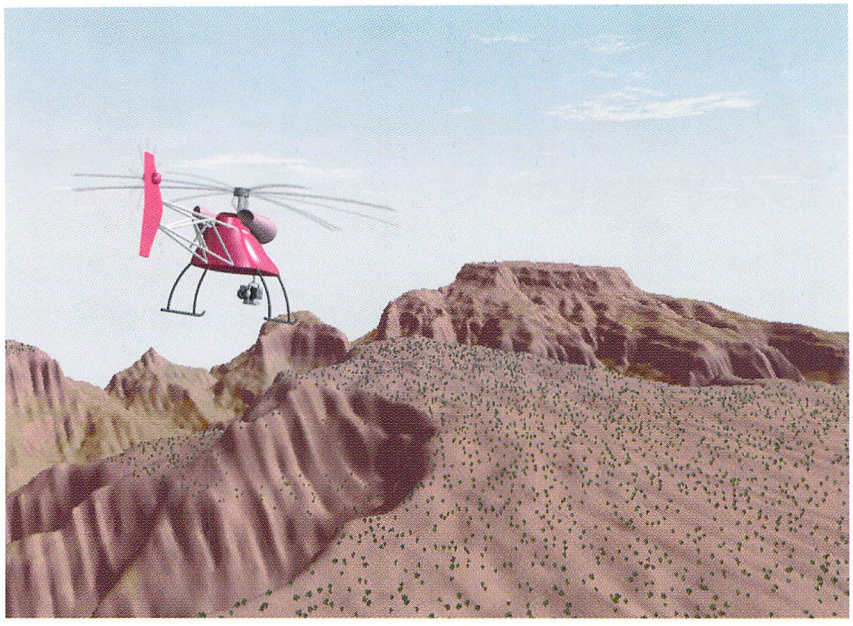
LIGHTWAVE™ PRO

an Avid Media Group, Inc. newsletter

Volume 3 Issue Number 4 • April 1995 • $8.00

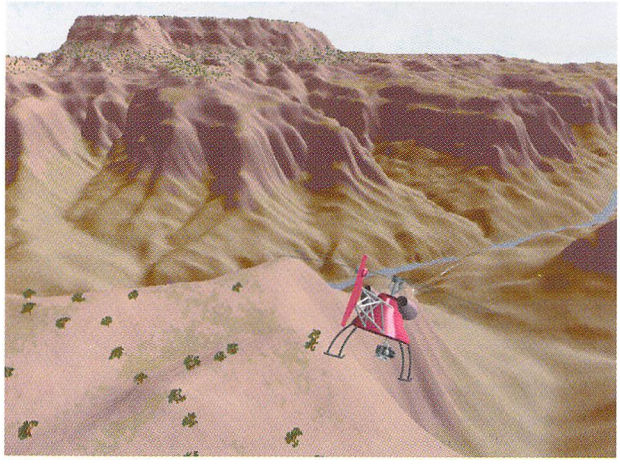
Mastering Metaform

Inside:



Canyon Run 50

The canyon footage in the final animation frame 50 was created in Scenery Animator and brought in as a background sequence. See "Tearing Through Canyons," page 4.

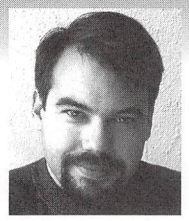


Canyon Run 297

A spline patch of the Grand Canyon was used in creating frame 297 of the final animation. See "Tearing Thru Canyons," page 4.

Dice

The dice in this scene were created by beveling a cube and then metaforming. The numbers were added with the Boolean Subtract tool. See "LightWave 101," page 16.



EDITOR'S MESSAGE

by John Gross

By the time you read this, it should have already happened. I know you hear that a lot when you are reading articles that were written before an event actually transpires, but this time I'm pretty sure it has happened.

I'm talking about the shipment of LightWave 4.0. First slated for distribution at the end of December 1994, and then again in the first quarter of 1995, LightWave finally shipped on April 7 (OK, that's only a little bit after the first quarter). While this is considered a "pre-release" version of LightWave, it is for all practical purposes so close to the final shipping version that the only additions are a few little things that didn't make it in time. Of course, since I am writing this before April 7, I don't actually know yet what those few things are. For all I know, LightWave will actually be done by then (it is very close).

Now you know what one of the problems is when writing about impending software shipments. The only information we, as writers, have is (1) information released from the manufacturer, (2) information released from the programmers themselves, and (3) rumor, conjecture, heresay and other non-confirmed sources. While I have, in many cases, mentioned and written about when the newest version of LightWave would ship, I have always based what I was saying on information retrieved from sources 1 or 2 above.

I have heard of and from people who have been very upset because LightWave didn't ship when originally forecast (end of December). I have also heard from people who have said that they have "lost business" because LightWave didn't ship then. They had needed specific features that would be included in LightWave 4.0 for a project they wanted to do (or had even bid on!).

Am I missing something here? Does it seem rational to depend on a product that has not even shipped in order to procure or complete a job? Perhaps I have lost touch with the "common LightWave masses" because I happen to be included among the number of fortunate users who get to see and use beta copies with new features.

Even so, I can't believe that anyone can be angry with NewTek because they didn't ship a product when originally planned. Disappointed, perhaps,

see Editor's Message, page 18

TABLE OF CONTENTS

LIGHTWAVEPRO
an Avid Media Group, Inc. newsletter

Editor ..... John Gross
Managing Editor ..... Jim Plant
Editorial Coordinator ..... Joan Burke
Associate Editor ..... Corey Cohen
Art Director ..... Helga Nahapetian Taylor
Art/Production Coordinator ..... Kristin Fladager
Production ..... Sergio "Berimbau" Miller
Circulation Director ..... Sherry Thomas-Zon
Circulation Assistants ..... Kris Nixon
..... Debra Goldsworthy
Contributing Writers ..... Dan Ablan
..... Wayne Cole
..... William Frawley
..... David Jones
..... Enrique Muñoz
..... John F.K. Parenteau
..... Earl Terwilliger
Group Publisher ..... Michael D. Kornet

Editorial Offices: Avid Media Group, Inc.
273 N. Mathilda Avenue, Sunnyvale, CA 94086
Telephone (408) 774-6770; Fax (408) 774-6783
John Gross can be reached electronically at:
jgross@netcom.com (Internet); 71740,2357 (CompuServe)

Are you interested in writing for *LIGHTWAVEPRO* or submitting images? If so, contact us at our offices or electronically.

LIGHTWAVEPRO (Vol. 3, No. 4); (ISSN 1076-7819) is published monthly by Avid Media Group, Inc., 273 N. Mathilda Ave., Sunnyvale, CA 94086-4830. A one-year subscription (12 issues) in the U.S. and its possessions is $48 (U.S.); Canada/Mexico, $60 (U.S.); overseas, $84 (U.S.). To subscribe, call toll-free 1-800-322-2843. Allow 4 to 6 weeks for first issue to arrive. Second-class postage rate paid at Sunnyvale, CA and additional mailing offices. POSTMASTER: Send address changes to *LIGHTWAVEPRO*, 273 N. Mathilda Ave., Sunnyvale, CA 94086-4830.

About the cover: The dice in this image were constructed using a beveled box and Metaform. The numerals were created with a Boolean subtract. For more information, see this month's "LightWave 101" column. Image copyright 1995 David Jones.

Tearing Through Canyons

By Wayne Cole

As 3D artists and animators we are constantly immersed in time and space. If you ask animators how they view a scene in its first inception, they usually admit that it is visualized as if they were in the scene in actual 3D space—not as a viewer apart from the scene seeing it on a movie screen, TV screen or printed page. That process starts when the translation from mind to form begins. We have to make sure things look good both spatially and temporally within the confines of a 4x3 or letter-box view frame. We have to ensure that our objects/actors appear to move naturally in virtual space (they are only electrons, after all). Even the dancing gas pumps and credit cards have to look "natural," and move with believable displacements for the amount of time we view them.

And there is the "other" space and time demanding an animator's attention: "How long is this gonna take to render?" and "Do I have enough [insert electronic media of choice] to store/archive these frames?" After all, you don't want to lose the frames of the animation—essentially your generation 0 master—until the program airs, the customer pays you, your production is done, your master survived the duplicating process, and all the duplicates are ready to ship.

We may spend a great deal of time and energy worrying about rendering time and media space, but because we are so saturated with 3D thinking, we often overlook 2D cheats that can save a bunch of time. Nir Hermoni's "Flying Through Canyons" article (*LIGHTWAVEPRO*, January 1995) reminded me of a similar project I did over a year ago. My approach, however, used 2D animation, combined with a feature of LightWave I have heard some people call a "quirk," in order to get around my inability to quickly model and paint a realistic "impressive landscape."

Stuck in 3D

The concept for the project was that a Remotely Piloted Vehicle (RPV) had to appear on screen in a form that suggested an engineering wireframe drawing. Then it had to materialize into its "finished form" in mid-flight through an impressive landscape. The RPV, by the way, was a small helicopter with a video camera mounted on for remote viewing/taping operations.

The RPV was going to be fairly easy to build since it was a mechanical object with many standard geometric shapes—things like cylinders, rectangular solids, and tubes with bends in them. Luckily, LightWave had just shipped with spline patches, so complex fuselage curves were easy to form. The landscape, at first, appeared to be another matter.

Stuck in my 3D mode, I thought about large spline patches, displacement maps...I probably played with at least four different ideas for generating this landscape. Meanwhile, the nervousness factor went up with each idea deemed impossible within the allotted time and budget. I was so entrenched in the 3D mindset that, for two days, I had the answer staring me right in the eyes and I didn't see it.

Rocket Science to the Rescue

One idea I tried used Scenery Animator to create a landscape DEM that I would convert to a LightWave object using Interchange. That part worked great. I had a wonderful spline patch of the Grand Canyon. All I had to do was to surface it to look like a real landscape in LightWave and then fly the RPV through it. I tried planar mapping expanded versions of Scenery Animator's map image of the landscape after snapping it from the Scenery Animator control screen. I tried building a convincing texture map in Brilliance. I tried selecting polygons in the mesh and applying different textures for water, sand and vegetation. All of which looked like what a dog leaves near hydrants.

Then it hit me. I remembered from back in my rocket scientist days that all those wonderful six-degree-of-freedom Link simulators for the space shuttle and other high-performance aircraft just gyrate around in front of a movie! At the same time, I remembered the little "quirk" in LightWave that, at the time I discovered it, I thought was a restriction instead of a useful feature.

The Immovable-Movable Background

If you load an image into a LightWave scene to use as a background image, the image is always dead-center in front of, and perfectly square in relation to, the camera view, no matter where you point the camera. So the solution to my landscape problem became quite simple: use a Scenery Animator sequence as a LightWave background image sequence, then wiggle the RPV around in front of this animated background to mimic appropriate flight attitudes, just like the simulators do.

One of the benefits of choosing this method is that the final LightWave animation rendered faster than it would have if a real 3D landscape had been used instead. All LightWave had to do was plonk an image from the background image sequence in the yon plane (see "Rendering Algorithms Part 1: The Theory of Z-buffers," *LIGHTWAVEPRO*, January 1995), then render a single object—the RPV.

While making the Canyon Clip I used Scenery Animator 4.0 to generate a landscape movie. This process will work with any other scenery generation tool, like Vista Pro or World Construction Set, as long as it results in the ability to save individual RGB images to be used for the background sequence. In Scenery Animator I chose to use a DEM for a real-life place rather than using a fractal-based imaginary landscape. I picked the Grand Canyon. (Well, the requirement *was* an impressive locale.)

- In Scenery Animator's Project menu, load a DEM file to define the landscape through which you wish to fly. Then, using the various buttons on the main screen, set up the attributes that Scenery Animator uses to "paint" the landscape (Figure 1).

  You can control things like the minimum elevations

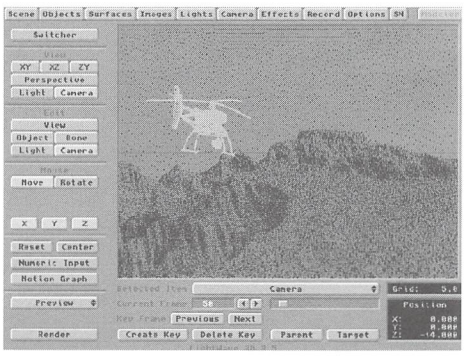


Figure 1: Scenery Animator Main Screen

at which snow and rock appear and the maximum elevation at which vegetation appears. There is a button to bring up a Sky Control panel, where you can define the cloud parameters for your particular scene. Water and tree characteristics are also definable. Finally, the direction and horizon angle for the sun must be entered.

Make a conscious and intelligent decision about light direction. Remember, you will want the lighting to favor the 3D object while creating enough landscape shadows to contribute to the feeling of depth. So avoid things like full backlighting unless you really want a silhouette scene. Above all, note the angle of the light in the landscape scene, because you need to be sure that the LightWave scene lighting will be consistent. If the landscape shadows go one way and the 3D object shadows go another, your final product will not look right.

- The next step in constructing a realistic-looking landscape involves defining colors to be used for water, rock soil, sky and vegetation. In Scenery Animator, you can control all these characteristics from the control panel that comes up when the Screen button on the main control panel is selected. You have full flexibility to make the colors as real or surreal as you want.

Exercise care in selecting colors in any RGB-based program, since your ultimate output will most likely become, at some point, composite video. Do not choose extremely saturated blues, greens, reds or whites unless you do not care about chroma crawl and exceeding "legal" NTSC color values. Even with careful color selection, it doesn't hurt to run the frames produced with your landscape generator through ADPro's (or another image-processing package's) Broadcast Limit filter just to be sure.

- Now that the landscape is characterized, set up the flight path of the camera through the landscape. In Scenery Animator, for example, the Map button on the main screen takes you to the camera control panel (Figure 2). The program offers a map of the landscape showing the camera location and its field of view within the area defined by the loaded DEM landscape. There are buttons to control your view of the map on this panel, as well as requesters where you enter the camera location, attitude and lens values. You also set the keyframes for camera motion and lens focal length. After you have defined the camera motion and are satisfied with it based on previews run from Scenery Animator's main screen, it's time to generate the movie in front of which your object will appear to fly.
- At this point, from Scenery Animator's Anim Mode menu, save the animation as IFF24 frames. These frames will form the image sequence for the LightWave animation background movie. Though

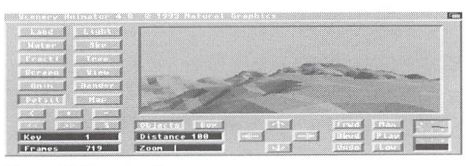


Figure 2: Scenery Animator map screen

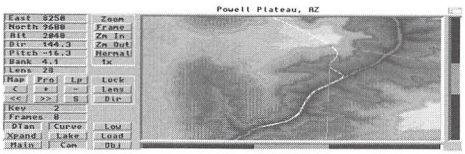


Figure 3: Layout showing a wireframe of frame 50 with background image.

the description sounds long, the entire process took me approximately two hours as a first-time user of Scenery Animator, and I suspect the time would be about the same to set up the same animation within other landscape generation packages.

Different packages have different standards for naming frames of an animation, and LightWave is no exception. Whatever landscape generator you use, you'll need to be sure that you label the frames it generates in LightWave-compatible form before loading them into LightWave. The landscape image names should look like "FlyThruxxx," where "xxx" is a three-digit number matching the LightWave animation frame in which you wish to have that specific landscape image used as a background. Note that all the images in the sequence should also be in the same directory.

The Flight Plan

Now comes the fun part (i.e., the LightWave part):

- Load the flying ship object that you've so painstakingly modeled (or so painlessly lifted from a commercial or public domain object set) into Layout.
- Click on the Load Sequence button (**Images** panel), and when the file requester comes up, select one of the images from the generated landscape fly-by. Be sure to delete the numeric portion of the image name, because Layout will automatically append the number of the frame it is currently rendering to the image name you enter in this requester when it looks for an image in the sequence to place in the background.
- To tell LightWave to do just that—use the sequence for a background—select the sequence in the **Effects** panel's **Background Image** pop-up requester. Now you are ready to begin setting up the motions of the object, light and camera.
- From the **Options** panel, select the **Show BG Image** button. As you advance and reverse the frames using the **Current Frame** slider, arrow buttons or requester, you will see a "pencil sketch" version of the background image (like that shown in Figure 3) change to the one that matches the frame currently designated in the frame requester.
- Use the **Move** and **Rotate** tools with your object selected to position it appropriately for the particular background image. You can even use a wireframe preview to check the motion of the object in front of a pencil-sketch version of your background movie.
- Another time-saving preview trick available in LightWave 3.5 and greater involves the ability to save and load wireframe previews. Use an empty scene with only the background image loaded and **Show BG Image** (**Options** panel) enabled. Select the **Preview** pop-up and make a **Bounding Box** preview of this scene—essentially, the pencil-sketch version of your movie. Now, from the **Preview** button pop-up, save your wireframe preview.
- Go to the **Options** panel again and enable the **Preview** button for the **Layout Background**. You'll find the Layout screen's Current Frame slider is much more responsive as you move from frame to frame. If, after looking at a preview with the object(s) in it, you don't like the object motion, simply reload the background-only preview, reposition your objects and make another preview. Then save the preview again if you have more objects to load and position.
- You can repeat this operation for each object in the scene. If the scene has many objects that require critical positioning with respect to the background, this method saves a lot of time in laying out the scene.

When positioning the object, remember that the scenery tilts and moves for the camera's apparent motion. So if your camera is set up like a chasing plane's view, be sure to have your object lead the camera through a motion. For example, if the background sequence does a sharp tilt to the left, it indicates the camera—er, chasing plane—is banking hard right. Assuming it is following the flying object, that object should have entered its hard right bank before the chase plane. It may even have started its roll-out by the time the chase plane starts its bank into the turn.

Light positioning is also important. In the turn above, the angle at which the light hits the object must change by the same degree of turn your background movie indicates the flight path took. You have the option of moving the light or parenting the camera to the object, keeping the light stationary and rotating the object through the turn. Remember, no matter how the camera moves, the background image will stay directly in front of it. So, to get the realism of the chase, move the camera toward and away, side to side and up and down in relation to the object.

If you made similar perturbations to the camera's position within the landscape generator animation, you now get the feel of real flight, with altitude, attitude and heading changes. And it looks like you are actually following the flying object rather than being rigidly—and unrealistically—attached to it. You don't even have to remember all the little zigs and zags you put into the landscape generator animation. Simply coordinate the LightWave camera motion and object motion to the major direction changes in the landscape animation.

Well, that's it! Just set the frame count for the animation and start the rendering. You'll be surprised how fast it goes with a background image sequence landscape instead of a real 3D landscape. See the color pages for an idea of what the result looks like.

The Cons

Are there drawbacks to this method? Of course. Your object can't cast shadows on the landscape. But by proper management of light direction and camera angles, those shadows, if they existed, would not be visible anyhow. Another potential disadvantage is that the object can't fly behind an element of the land-

see Tearing Through Canyons, page 17

Making LightWave Sparkle!

A Particle Animation Lesson

by Enrique Muñoz

LightWave has always had the ability to do realistic animations. With versions 3.0 and 3.1 we were introduced to advanced features like Lens Flares and Displacement Mapping. With version 3.5 we've gained Metaform. Yet even with the upcoming 4.0 version, this program still lacks something—Particle Animation. This is where MetroGrafx's Sparks comes in. In this tutorial we will focus on two effects from the popular television series *Babylon 5*: Explosions and Water Fountains.

Setting Up

The software I am using is LightWave 3.5 and Sparks 2.174. This is important because this version of Sparks has several bugs in it (call MetroGrafx for an upgrade), and I want to make sure everything you do comes out the same way I did it, regardless of these glitches. I'll assume that you have had some previous experience with both programs, so if you have trouble keeping up, refer to your manuals.

To get started, we need some base objects: one single-point polygon for our water fountain project (name it FountainParticle.lwob), a spaceship to explode (a good one would be the spacefighter object that comes with LightWave), a particle cloud for the explosion and a fountain object to put our fountain in.

Pyromania

We'll start with the *B5* explosion.

- Go into Modeler and **Load** (**Objects** panel) in your spaceship. Run the **Macro** (**Objects** panel) Fragment.lwm (This macro comes with Sparks). For input values, see Figure 1. You might be wondering why I chose such a low number of subdivisions. You must remember that each axis of subdivision gets multiplied by the other two axes, for a total of 64 in our example. (We get 25 because that was the proximity where the spacefighter was when the macro did the Boolean operations.) Another thing about our settings—Fragment ordering doesn't mind if you have Random ordering selected (like we do).
- The base name for the fragment objects is SF-Fragments. After running the macro you should end up with 25 objects (if you are using the spacefighter). Load

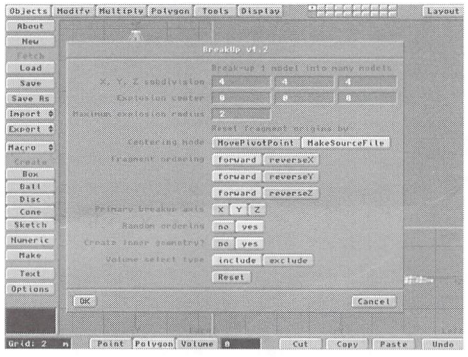


Figure 1: The BreakUp macro used to slice up your object.

Sparks and enter the appropriate values for each of Sparks' fields. For Particle Quantity enter 25 (or any other number you get after running the macro). For Start Position enter 0 for all axes. For Velocity (m/sec) enter 12. For % Velocity Variation enter 30. For Ground Plane enter -500. We are putting such a high and negative number for this field because we don't want our particles (the ship fragments) to collide with the "Ground." For Gravity (m/sec²) clear whatever entry is on there and leave it empty.

- Next we must fix our Set Angles. This requester is used to input the directions in which we want our particles to go. For the purposes of this tutorial the particles should move in a spherical manner. In the Plan View (Set Angle) set Heading to 0 and 360 for

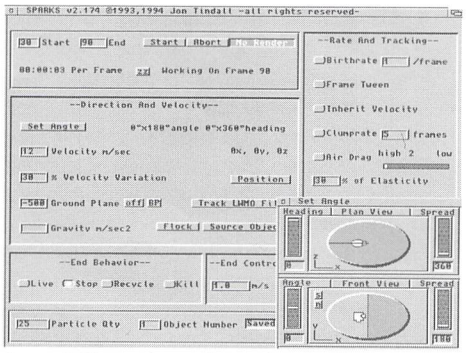


Figure 2: The Sparks interface with our explosion settings.

Spread. In the Front View (Set Angle) set Angle to 0 and Spread to 180. Set the Start and End Frame to 30 and 90, respectively. (We chose 30 for our first frame so you can have some time to fly your ship before it explodes.) After all this typing and tweaking your screen should look something like Figure 2.

- Under the Control menu select No Move. Under the Scene menu choose Select Object. Click on the button labeled Add and a file requester will come up. Select the directory where your object files were saved and while holding down the shift key select each object and press OK. Select the button labeled Align to Path. Here's the tricky part. We are going to set mathematical equations for our Rotation Evaluations. Under the Evaluate H, P and B fields enter the following:

  Evaluate H.....h[i]+10
  Evaluate P.....p[i]+10
  Evaluate B.....b[i]+10

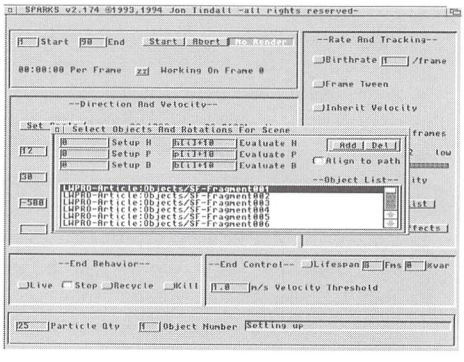


Figure 3: Spark's Select Objects requester showing the mathematical equations for our rotating ship fragments

Your screen should look like Figure 3. Close this window. Under the Scene menu choose Save Scene. Save your scene in the directory of your choice and name it B5Explosion.scn.

- One other thing we must do to our setup is parent it to a null object, which will help us keep all the elements of our explosion (like lens flares, particle clouds, etc.) together when we go back into LightWave. Do this by selecting Parent Object from

the Scene menu. The last thing to do under this menu is select the Particle Size; in this case, select Medium. Now hit Start, and after a couple of minutes, load your scene into LightWave. You might want to set up fade envelopes for the objects. They weren't set up within Sparks because I usually cut to another scene after exploding a ship (for more drama).

- The final elements to add are lens flares, particle clouds, and surfacing of the inner geometry of the ship and the particle cloud. Use a dark red fractal noise texture to make the inner geometry look like burnt metal. I will leave the setup of the lens flares and particle clouds to you since I know different animators have their own look and taste for explosions. (For more information on explosions, check out past issues of *LWPRO.*) Select a good camera position and watch the magic. Wow!

Getting Wet

This next tutorial involves building a pretty complicated fountain. Luckily, I've made it as simple as possible.

- First of all, if you're proceeding straight from the other tutorial, you might want to select New from the Project menu so that we start out with no default settings selected. Go to the Set Angles requester and in the Plan View set Heading to 0 and 360 for Spread. In the Front View (Set Angle) set Angle to 86.5 and Spread to 6. Adjust the Start and End Frames to 1 and 180, respectively. Set End Behavior to Recycle. Set your particle quantity to 180.
- Click on the Effects button, which will bring up a requester. Even though there are a lot of options here, the only thing we have to concern ourselves with is Bounce Probability. Set this field to 5%. Click on Accept.
- Under Rate And Tracking select Birthrate on and input 3. For Velocity (m/sec) input 3. Type 20 for the % Velocity Variation field. For the Ground Plane input 1.1. For Gravity (m/sec^2) input 4. Finally, click on the Position button and type 1.45 on the Y axis. Leave the other two axes (X and Z) at 0. Your screen should look something like Figure 4.
- Under the Control menu select No Move. Under the Scene menu choose Select Object. Click on the button labeled Add and a file requester will come up. Enter the directory where your object, FountainParticle.lwob, is located and select it. Close this window, select Save Scene (under the same menu) and save the scene in a directory with the name Central Spout.scn. Don't forget to parent your particles to a null object and select a size for them (preferably medium).

The weird thing about doing some particle animations in LightWave is that when you try to recycle particles, you get this long streak when they go back to their starting positions. *[Edior's note: This is due to a bug in LightWave when using Particle Blur and repeating particle motions.]* The way to remedy this is to set up a fade envelope so that the particles are dissolved at the point where they are going back to their starting positions. Follow these steps:

- In Sparks, select Setup Fade under the Scene menu. This will bring up a requester. For the Active and Inactive Value we want to input 0% and 100% dissolve, respectively. We only want one frame for Fade In and Out Duration. The last thing is End Offset, which corresponds to our Birthrate. Since our Birthrate is three per frame, we input the same amount for End Offset.

That takes care of one part of our fountain. Now for the other part—the side angle spouts. Again, we must start from scratch to make sure that there aren't any variables we didn't consider.

- Go to the Set Angles requester and in the Plan View set Heading to 0 and Spread to 5. In the Front View (Set Angle) set Angle to 65 and Spread to 5. Set the Start and End Frames to 1 and 180, respectively. Adjust End Behavior to Recycle. Set your particle quantity to 120.
- Under Rate and Tracking options select Birthrate on and input 3. For Velocity (m/sec) input 3. Type 20 for the % Velocity Variation field. For the Ground Plane input 0.2. For Gravity (m/sec2) input 7. Next, click on the Position button and type 1.45 on the Y axis. Leave the other two axes (X and Z) at 0. Set your End Behavior to Recycle. Now go to the Effects requester and input 5 for the Bounce Probability. Leave everything else at its default and click on Accept. Your screen should now look like Figure 5.
- The last thing we have to do to our setup is select No Move from the Control menu. Under the Scene menu select your object, FountainParticle.lwob, with the Select Object option. Save your scene in the directory of your choice with the name SideAngleSpout.scn. Again, we are going to want fade envelopes for our particles so that we don't get that long streak when we recycle them (use the same settings as our previous example). The next thing on the agenda is to set our Particle Size. Set this to anything you want, but I recommend Medium.

Putting It All Together

The only thing we have to do now is put the scenes that we created in Sparks together.

- First, select **Add Null Object** and load in the scene Central Spout.scn. From the **Objects** panel select **Load from Scene** and load Side Angle Spout.scn. Do this last procedure three more times and rotate each corresponding null object (the parent to our particles) a multiple of 90 degrees, while parenting them to the first null object. Next, load in your surrounding objects (like your fountain model, and maybe something like a park surrounding objects), set your correct lighting, and BAM!—a pretty killer scene.

Pitfalls

The main problem that might arise during these tutorials is memory limitations, RAM- and hard drive-wise. It took 9MB of hard drive space to save the completed scene file with all five of our scenes together (one of our central spout scene and four of our side angle spout). You should be running your machine with at least 18MB of RAM (what LightWave professional doesn't?) or else you won't be able to render it out.

You might be wondering why we went through this whole process for the explosion when there are other programs that will do this for you (like PowerMacros). The only problem with these utilities is that you don't get all of the various options that Sparks provides. An example of this is if you want your explosion parts to be affected by factors such as wind and gravity.

In any case, I've included the Sparks setups for the Fountain and Explosion (for you lazy folks or people without the time to follow this article) on the *LWPRO* disk so that all you have to do is select Load Project from the Project menu.

LWP

Enrique Muñoz spends his time listening to the all-time greatest band, Pearl Jam, while he works on various television commercials (and hopefully on an upcoming feature film project) as the senior animator for Digital Imaging, based in Ontario, Calif.

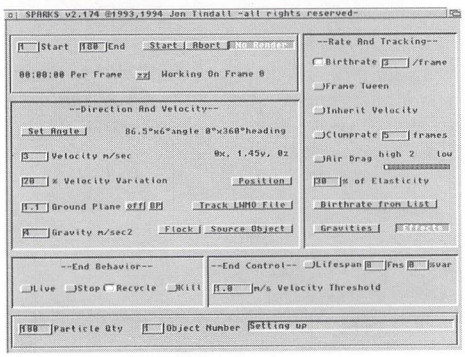


Figure 4: The Sparks interface with our central spout settings.

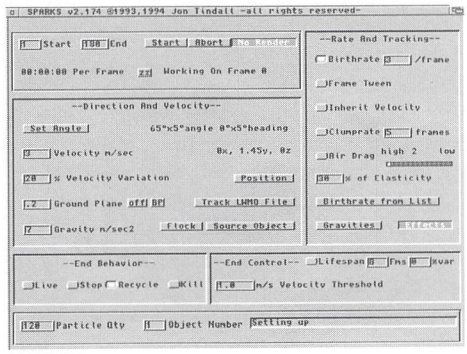


Figure 5: The Sparks interface with our side angle spout settings.

Have Starfield, Will Travel

by William Frawley

Anybody out there still relying on the "Random Stars" object for their space scenes? If so, you need help. Dr. Starfield to the rescue! I mean, really, we've all seen them—shining examples of rich, cinematic starfields in movies such as *Star Wars*, *Star Trek* (all seven), and more. Heck, even the Universal logo background looks pretty impressive! In the vacuum of space, there's not much standing in the way of you and the glorious bounty of millions of visible stars. Here then, let the Doctor fill the required prescription for respectable starfields: caffeine, LightWave, good tunes and...more caffeine.

What Are Particles Again?

Just in case you've recently been revived from a successful cryogenics experiment, the basis of starfields in LightWave involves the use of particles, or more specifically, single-point polygons. This special type of polygon has the characteristic of being completely dimensionless, which causes it to uniformly render at the same screen size regardless of its distance from the camera. Created by converting a point into a polygon, particles possess all the surfacing capabilities of polygons, but because of the dimensionless nature of points, contain none of the spatial properties of polygons. Incidentally, two-point polygons, or lines, behave similarly as particles, except lines have a second dimension. Since version 3.5 of LightWave, users have been able to control the rendered screen size of both particles and lines: small, medium or large. This means that each particle renders as one pixel, a 3x3 pixel array or a 5x5 pixel array, respectively (similarly, lines are drawn as one, three or five pixels thick). This ability to control the size of rendered particles will play an important role later on in our endeavor for enhanced realism.

A Starfield of Dreams

OK, so now you're ready to get serious. You're in Modeler and you're wondering how many licks does it take to reach the center of a TootsiePop—uhh, I mean, how many particles does it take to make a convincing starfield? Well, since we'll be constructing a fully functional, spherical starfield to be used in scenes where camera movement is quite dynamic (a lot of panning, tilting and warping), I've found that approximately 57,000 is the minimum for a nice plenteous look. (A dense starfield is one thing; we'll see later that a resplendent starfield takes multiple objects and surfaces.) However, you must consider your memory situation. On an 18MB T2000, this many particles uses about 75 percent of the available space, leaving only enough room for about another 20,000 polygons for a successful render. You can reduce the amount of particles needed if you're camera doesn't pan much by slicing the starfield in half and discarding all the particles behind the camera. Or parent those remaining particles to a Null and use the camera's motion file for the Null's motion file. Consider this option as well: If you do have enough RAM, you can again slice the field in half, but this time take those particles and add them to the remaining particles (rotate 180 degrees) in front of the camera, thereby increasing the starfield's density twofold.

With an idea of how many particles you can comfortably get away with on your machine, there's just one more thing to consider before we create the starfield: its size. Again, consider what the camera's motion will be. If it's warping through space, you might want to size the starfield as large as possible to reduce the apparent motion of the more distant stars. Otherwise, if the starfield is too small, all stars will show motion, destroying the illusion of the galaxy's expansiveness, and in no time you'll be at the Outer Rim with no stars in sight (we'll cover special star columns for warping a little later). On the other hand, if the camera merely needs to pan or remain static, pivoting motion won't divulge the stars' actual distance from the viewer. Therefore, keep these points in mind when using the appropriate particle-creating macro. [At press time, I'm still working on "Starfield.lwm," a starfield/particle macro that is much faster and more appropriate for our purpose than the "Point Distributions" macro. Look for it on a future *LIGHTWAVEPRO* disk.]

- For now, enter Modeler, select **New** (N, **Objects** menu) and use the "Point Distributions" macro to create 10,000 particles. Set Falloff Towards to "Center" and Density Distribution to "Constant," leaving everything else at their defaults. This process could take about 10 minutes, but once it's done, these particles will serve as "seeds" for building most of our starfield. Enter Polygon selection mode (space bar) and rename the **Surface** (q, **Polygon** menu) to "Stars 1" or something similar. Export this first object to Layout, saving it with some name like "Stars1.10K" where the 10K signifies that this object is made up of 10,000 particles.
- Now **Copy** (c) these particles, enter layer 2 (2) and **Paste** (v). Next we need to randomly shift the positions of the particles so they don't overlap with those in layer 1. Because the radius is only 1 m (suitable for our purpose), **Jitter** (J, **Tools** menu) the points of the object by .1 m (5-10 percent of the radius) on all axes (Figure 1). Rename the surface (q) for these particles "Stars 2" and again **Export** to Layout with a name of "Stars2.10K."

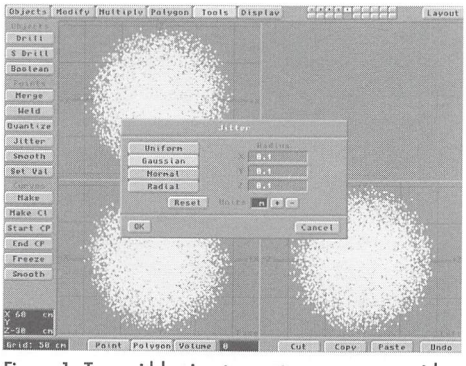


Figure 1: To avoid having to create any more particles than necessary with the "Point Distributions" macro, construct a percentage of the total number, then copy this object to other layers. Use the Jitter tool to randomize the locations of the cloned particles.

- Repeat this procedure three more times so that there are 10,000 particles in each of the first five layers, each with distinctive surface names and saved as separate objects.
- Now enter layer 6, and again use the "Point Distributions" macro to create 5,000 more particles (same parameters as before). Rename this surface "Stars 6" and Export/Save as "Stars6.5K."

- For the last step, enter layer 7 and create 1,000 particles, rename the surface "Stars 7" and Export/Save as "Stars7.1K."
- Enter Layout, shut down Modeler and **Reset** the camera to the origin. **Create** a keyframe (Return) for the camera in its new position. Since we constructed multiple star objects earlier, we can now assign each group of particles a different particle size to give the starfield the appearance of depth. Without this feature, each star looks as though it was at the same distance as its neighbor. Pretty boring, eh? Therefore, make the Particle/Line Size (**Objects** panel) of Star objects 1 to 4 "small," that of Star objects 5 and 6 "medium," and that of Star object 7 "large." Since each particle won't be affected by a light source, you can save some rendering time by turning off each star object's "Self Shadow," "Cast Shadow" and "Receive Shadow" options.
- In the **Surfaces** panel, set each object's surface **Luminosity** to 100% and **Diffuse Level** to 0%. Change the color of each surface to include some yellow, blue and red varieties. By varying the Surface Color and Luminosity of each surface, you can fine-tune the subtleties of the starfield to suit your particular taste.
- Next, turn **Ambient Intensity** (**Lights** panel) and **Light Intensity** to 0% and make sure to set the **Antialiasing** (**Camera** panel) to at least **Low**. Before you do the test render, do a **Save All Objects** (**Objects** panel) to record the surface settings and then **Save Scene** (**Scene** panel).

From now on you can load this starfield into any of your space projects by doing a very handy Load From Scene (**Objects** panel). With that business out of the way, try a test render and compare it with the Actual Stars object (Figure 2).

To increase the richness of the field even more, try further subdividing each object into multiple surfaces back in Modeler. I'm sure you can figure out how to do this simple task. You'd then have multiple particle sizes with multiple surface attributes. The Milky Way never looked so good.

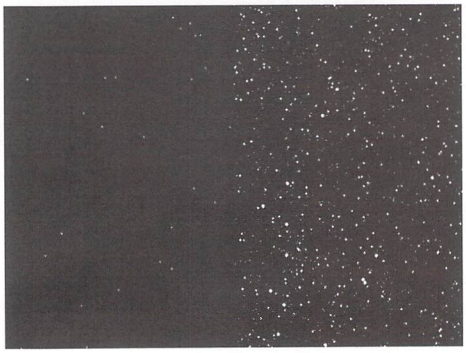


Figure 2: With a little effort, you can overcome the "girlie-starfield" syndrome. On the left is the Actual Stars object (≈1500 particles); on the right could be your own 56K (memory permitting) award-winning starfield.

Yes, My Sun

A good starfield wouldn't be worth its weight in hydrogen if it didn't have some local stars asserting their presence in your scene. This is where lens flares really shine (pun intended). Depending on how you want a local star to look, you'll want to use at least two flares for the actual star and another for any associated glare or lens spikes (star filter). For example, Figure 3 shows the settings I used for each of the Seven Sisters shown in the color image "Pleiades."

For each nearby sun, I used three flares. One was used as a faint blue flare and to produce lens spiking using the Star Filter option (**Flare Options** panel). The other two flares were colored white and used to produce the actual star. By combining two or more flares with similar attributes, the edges of the central hotspot will be more pronounced and the overall glare surrounding the lens flare will be reduced. Next, in order to vary the apparent size of each star, I used the Fade With Distance option so I wouldn't have to con-

Figure 3: Three lens flares were used for each of the Pleiades suns in the related color photo. Shown are the settings used for one blue lens flare acting as the flare and lens spike element. The other two identical white flares were used as the basis for the sharply defined star element.

cern myself with each flares intensity. Simply set a Nominal Distance, parent all the flares for each star to a null object, and move the null to the desired distance. For a really cool effect, if you happen to be animating the scene (I do a lot of still images), use Grant Boucher's "Random Envelope Generator" macro to create an envelope for each flare's intensity channel. This will give each local star its own pulsating look (Hint: Keep the envelope peaks fairly moderate).

In "Pleiades" I added a little extra eye candy in the form of dust clouds surrounding various members of the open cluster. Using OpalPaint (or any good 24-bit paint application), I simply painted a series of images using a combination of airbrush and watercolor tools to blend the blue into the black background and vice versa. After saving the color version of each image, I converted each one into a grayscale negative, making sure to boost the blue channel considerably more than the red or green in order to heighten the contrast between the dust and the background. This step ensures that, when applied as a Transparency texture map, the blue region (now represented by black in the negative) will be fairly opaque; hence, more visible (Figure 4).

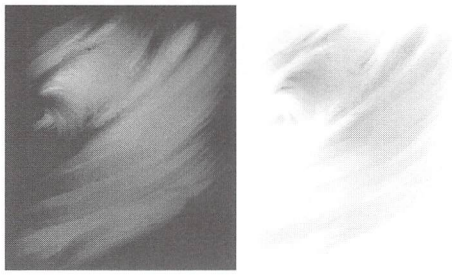


Figure 4: A good 24-bit paint program makes adding subtle details to your scene easy. In this case, the color (shown in black and white) image on the left was texture-mapped to a plane for the dust cloud's color, and then converted to a grayscale negative for the transparency texture map (right).

Ahead, Warp Factor 9

So now you want to warp, do ya? "I'm givin' it all she's got Captain! I can't give any morrrrrrrre!" Well, if you plan on testing the limits of your inertial dampening system, a different kind of starfield—a kinder, gentler starfield—is required. For straight-ahead warping through space, it's best to construct a columnar-shaped starfield; a tunnel full of stars, more or less. To preclude excessive redundancy on creating a warp column, I refer you to Mojo's excellent treatment of this subject in his article "Simple Space Stuff: Part I" (*LIGHTWAVEPRO*, August 1994).

Once you understand this concept for creating a nice warp column (don't forget to use Particle Blur, in the **Camera** panel), let me take it one step further by

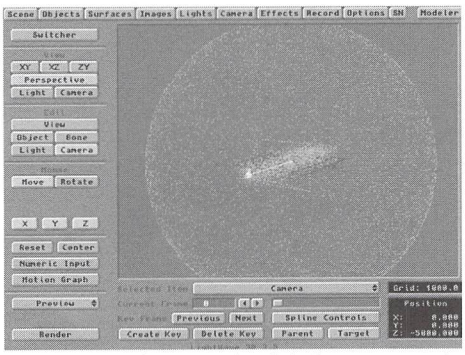


Figure 5: To get that DS9-end-credit look, more accurately reflecting the principle of parallax, the camera travels through an elongated star column, which is in turn surrounded by a much larger spherical shell of particles. (The Spherize macro was used on a Point Distributions-created starfield.) This gives the illusion of extremely distant stars in the background as the closer ones warp by.

see Have Starfield, Will Travel, page 17

Digital Cinematography

by John F.K. Parenteau

For quite a few columns now, I've focused on showing you how to re-create real-world lighting in LightWave. Though we have worked in general terms, the actual fact is that producing a realistic effect relies on much more than convincing lighting. A poorly modeled object or low-resolution textures can detract so much that even the best lighting will never look in the least genuine. At Amblin Imaging, the world of CGI effects is usually limited to the realm of the fantastic. Effects on *seaQuest*, though theoretically re-creating an environment we have all seen before, underwater, take a stylistic approach to this world. In truth, at the depth the submarines move, you would never see much more than the glow of lights within inky blackness. So authenticity is stretched for the sake of exposure and broadcast requirements. Yet, as we began work on the pilot of *Star Trek: Voyager*, we quickly came to realize that the stylistic would no longer apply. We were now dealing in a much more realistic world.

I know you're saying, "Realistic? It's in space, for crying out loud!" But space or not, *Star Trek* has existed for many years under conventions and practices people have become quite comfortable with. Though it may seem the wave of the future, and the most logical direction to take, CGI has been relatively non-existent in the *Trek* realm. Except for a few rare effects, this universe has existed solely in physical or dimensional models. Yet the future could not be held back, and Paramount (in very large part due to David Stipes, one of *Voyager*'s effects supervisors and a LightWave fan) commissioned Amblin Imaging to create a computer-generated *Voyager* model to complement the new dimensional model. In the early plan, the CGI model was intended for extremely limited use, in cases where the motion-control rig was busy or specific shots when the *Voyager* was required to tumble or appear out of control (a motion-control nightmare). At the start of effects production, only two *Voyager* shots for the pilot were commissioned as CGI shots.

The Motion-Control Way

As many of you may already know, motion-control rigs have been utilized for years to produce effects shots ranging from the battles in *Star Wars* to *E.T.* riding across the moon. The basic process is quite simple, though limiting: The model is connected to a rod that supports it suspended in the air. The camera moves on a motorized, computer-controlled rig. By programming the camera to move past the ship, the illusion of the ship flying by camera is created. But there are inherent problems in this simple process.

First, since the camera is connected to a crane or boom arm, it cannot move around the object completely without seeing its own support system. The answer to this problem was to move the model in sync with the camera, allowing for additional, yet still limited, banking or other movement.

Second, to support the ship, a rod is connected somewhere on the ship. Though the mount can connect in several places on the model, it is nonetheless immovable during shooting. In addition, though the mount is painted to fade into the background, if the camera moves into a position in which the mount is in front of the model, the finished shot will show a hole in the model, in the shape of the rod, as it "punches" through the image. Thus, regardless of the mobility of camera and model, a motion-control rig can never truly move in a 360-degree path around its object.

Third, the models are commonly shot against black, with backgrounds matted in later. Most moderately detailed models must be in the four- to six-foot size range. Unless the motion-control rig is extremely large, and the background very wide, it is nearly impossible to make a model of this size appear tiny in frame. Shots of a ship receding in the distance are difficult, and when required, are often post-production tricks.

Fourth, since the rig is computer-controlled and highly accurate in its programmed move, it does not move at actual real-time speed. Averaging one frame per second, the camera moves slowly over the model, regardless of the speed of the final shot. If the ship were to pass over camera very fast, the camera would still move at one frame per second to photograph the shot, and compensate by moving a greater distance per frame. Motion blur, the smearing effect an object has on film as it speeds by camera, must be applied in post-production and lacks realism.

Of course, motion-control operators will point out that all the problems above can be worked out, and in truth, they are right. The corrections, however, take time and money. In the realm of CGI, these "problems" don't even enter the equation. Since the model exists in a 3D space within the computer, we can move it as far away as necessary, spin it, tilt it or fly around it. Motion blur is simply a button, and with the right amount of rendering power, just a few more minutes per frame. As we proceed into the first season of *Star Trek: Voyager*, these facts are becoming increasingly evident to the producers and the special effects staff. Though the model is still a major part of effects production, the world of CGI is slowly but surely becoming an important part of the show.

The CGI Way

No, it isn't that easy. No matter how much we all feel CGI is the answer, it is still a complex and difficult process at times. Several people (including David Jones, Bruce Hall, John Gross, Tony Stutterheim and Eric Barba) spent many weeks completely re-creating the physical model (what *Star Trek* calls its "real" model) in Modeler.

As a kid, I spent many an afternoon tearing apart perfectly good model kits to create my new and much more fantastic design. I remember how easy it was to grab a small piece from another model and place it on my ship to add relief. These small objects, called "nurnies" (a Ron Thorntonism), could have been the death of us on *Voyager*! Building a model from scratch can be much easier, since you can determine what detail and where to place it. In matching a dimensional model, the 100 frivolous bumps a model builder decides to throw on at the last minute can cause a CGI modeler to go mad! Our animators spent many days studying the model, photographing it and videotaping it, all with the sole thought of exactly matching an existing model. Though the model will continue to be improved and enhanced, the final product should match the original almost precisely. The best part of the CGI ship is that somewhere, hidden inside a room, is a small plaque with the names of the people who gave their all to make this *Voyager* come to life.

Modeling wasn't the only battle to be fought as we started up on *Voyager*. Texturing played an important role in creating a convincing model. Hours were spent creating bump maps, specular maps and diffuse maps in an attempt to give our CGI ship a realistic feel. After the long and painstaking process, the *Voyager* visual effects department told us it was all wrong! We quickly came to realize that we had textured our model to look as realistic as possible, when the dimensional model we were matching wasn't real at all. For example, careful consideration had been placed in making the textures clean and smooth, when in truth, the paint that had been applied to the dimensional model had left minute streaks over the surface—streaks we failed to match! Many issues arose that showed our staff the complexity of duplicating an existing model. On *seaQuest*, all the ships existed as CGI first. No practical model was ever created to match. Even the toymakers were required to contact us for designs since none had actually been done. The *Voyager* had been fabricated out of plastic components, carefully airbrushed and decaled with painstaking detail. It was important for our staff to approach the surfacing not from the standpoint of creating a realistic look, but rather a look that matched the dimensional model.

Lighting the Way

Practical lighting quickly became an issue as we placed CGI lens flares to match the fiberoptic lights designed within the dimensional model. Minute details became major issues. For example, the glowing panels on the front of the warp engines are created on the physical model by placing a small light inside a translucent shell. The small red light illuminates the panel, creating the soft glow through the milky white surface. On the CGI model, we aren't required to use a light to create the same effect. Instead, we simply apply luminosity to the outer panel. Though this seemed the easiest answer, it proved to be another detail overlooked. As I examined shots of the dimensional model provided by the *Voyager* crew, I noticed light kicking off the surface of these translucent panels. Though the panels appeared red from the light inside, the gleam of light off the outer surface was white. In truth, this is an unwanted effect of the actual model, but it was one that would make our CGI model stand out to the trained eye. By applying some diffuse value to the luminous panel, a similar effect was created. Many weeks of testing were required to achieve the look that the *Voyager* effects staff had become accustomed to in the past.

With dimensional model photography, the ship is shot with darkened windows, no warp engine glows and no practical lights. Each of these items are shot as separate passes and re-composited in the on-line bay to allow for the greatest control over each element. Though a similar process could be taken with the CGI model, Dan Curry, the visual effects producer, began requesting we provide a fully composited shot rather than individual passes. Careful consideration had to be taken to set values for windows, lights and glows to provide a properly balanced final image.

The final battle that continues to be fought is matching model photography lighting. David Stipes spent hours discussing with me the methods in which the dimensional model is lit. A key light is placed in an optimum position, usually to provide a shadow edge on the camera side of the ship. Light is bounced off a white card placed above the model. Other fill lights are placed on the side and bottom of the ship, with roughly the same value to provide an even fill across the shadowed areas of the model. Initial approaches to lighting the CGI *Voyager* appeared to be easy, but as we all know, nothing is actually easy!

Though the simple lighting setup worked well for the practical model, the CGI model was not flying a similar flight pattern. The reason for a CGI model, initially, was to provide moves unavailable to the motion-control rig. This often means that the basic lighting setup can no longer work to provide an acceptable look. Yet, since we are working in a computer, certain rules that apply in the real world are not even considerations for us. For the shot in the pilot that we call "The Wave Slap," when the *Voyager* is struck by a wave of energy (courtesy of Grant Boucher) and flung across the galaxy, we were faced with one of our most difficult lighting tasks. It was necessary to bring light from the back of the ship in increasing intensity, to accentuate the violence of the vessel being struck. Though it would seem simple to place lights behind the *Voyager*, if you recall the shot correctly, the ship was lifted back end first and tumbled forward. By simply placing lights behind the ship, the lighting may appear correct for the first few frames, but as the ship is raised, shielding itself from the lights, a huge shadow would be thrown across the saucer. In motion-control work, there would be little to do to solve this problem. In CGI, I simply applied keyframes to the lights, lifting them up to match the angle the ship had been tilted. This technique allowed the light to continue to reach over the edge and light the saucer. Just as the *Voyager* exits frame, the keylights are almost directly above the ship, rather than behind the wave.

It has never been said that motion-control photography is easier than CGI. The greatest complaints are that CGI cannot look as real. After several episodes of *Star Trek: Voyager*, the discussion on the Internet still rages as to which ships are CGI and which are models. Though it would be nice to be confused with reality instead, at least it's a step in the right direction.

By the way, for those of you who may be unsure which is which, there is a dead giveaway to the CGI *Voyager*. Whenever you see it from the rear, the LightWave *Voyager* will have lit windows in the very back end of the ship (below the shuttle bay). The real model doesn't have this, as there was no way to snake a light back there.

LWP

John F. K. Parenteau is one of the vice presidents of Amblin Imaging and CGI effects supervisor for seaQuest DSV.

LightWave-generated Voyager Footage (as of early-March)

Opening Sequence

Three of the six opening sequence shots use the LightWave-generated *Voyager*. The other three shots use the practical model. All of the background elements were generated by Santa Barbara Studios using Wavefront. The three LightWave-generated *Voyager*s are:

- The first shot of *Voyager* flying by sun
- The third shot of *Voyager* flying through space fog
- The last shot of *Voyager* flying by planet and jumping into warp

"Caretaker" (pilot)

- All Badlands and vortex footage
- All galactic wave footage
- Voyager getting hit by galactic wave
- Planets, stars and sun for all planet shots
- Blue anamorphic flare elements in transporter beam in/outs
- Alien fractal elements used in "Caretaker" beam-out effects
- Final jump-to-warp shot

"Parallax"

- All *Voyager*-at-warp shots (including stars and warp stars)

"Phage"

- Five shots of *Voyager* in asteroid lined with mirrors. The actual model was used in the foreground and LightWave-generated reflections of the CGI *Voyager*, alien ship and phaser beams were used for the backgrounds.

"Eye of the Needle"

- Micro wormhole shots seen on the *Voyager*'s viewscreen

"Emanations"

- All shots of *Voyager* and ring planet/asteroids together.

General

- Many stock shots of *Voyager* flybys at warp and at impulse speeds

Fractal 3D Objects

by Earl Terwilliger

Now that our favorite rendering and modeling software package will soon be available on other platforms, our jobs as animators and artists will be made a bit easier. We can choose the platform where we already have most of our other necessary software for video or graphics production (if it is not an Amiga), eliminating lots of swapping data back and forth. I am looking forward to using LightWave on the PC since I do most of my graphics work on that platform.

One package that runs in the PC DOS environment that I have found of particular value for creating natural objects such as trees, plants, leaves and animal "critters" for use in LightWave is called LPARSER. The program, written by Laurens J. Lapre, is free and can be found in the graphics section of the SIMTEL collection of PC software. (SIMTEL is available on CD-ROM or through Internet access.) We probably overlook the "gems" to be found in these channels of software distribution because their value to us is not marketed in the usual way. "Hidden treasures" such as LPARSER may go unnoticed.

Modeling Objects Found in Nature

I like to look at nature, and when I am capturing a scene from it, I seek to replicate it as closely as possible. To accomplish this, I could use one of a number of good natural landscape generators, such as Vistapro or World Construction Set. These programs use "fractal" math to generate images (bit maps) resembling natural landscapes, complete with rocks, trees and bodies of water. Even in LightWave, fractal math can be used to create several natural-looking surface textures from what LightWave calls "Fractal Noise." (Benoit Mandelbrot coined the name "fractal.")

In brief, fractals are geometric shapes that can be subdivided in parts, each of which is (at least approximately) a reduced-size copy of the whole. Fractals are generally self-similar and independent of scale. Although there are mathematical structures that are fractals, the term also describes many real-world objects—such as clouds, mountains and coastlines—that do not correspond to simple geometric shapes.

These fractal-generated natural landscapes work well as backgrounds for many projects. However, one problem with the generated bit maps is that the "objects" in those images, such as the trees, can't be individually manipulated. There are techniques in LightWave for this, but what if I want a tree to move, or some other object to move behind or around it? The best solution is to have a model of the tree object. Hopefully, the landscape generators we have today will be, in the near future, updated with more 3D modeling capabilities. Today, though, LPARSER can help.

What Is LPARSER?

LPARSER allows you to produce trees, plants, "critters" and other shapes found in nature in a 3D format, which you can import into LightWave. The program is based on l-system fractal math (very similar to the math used in the fractal math landscape generators). There are many sample objects provided in the LPARSER package that you can readily use. However, if you want to make your own objects, you will need to learn the l-system commands implemented by LPARSER. The documentation supplied with this software will show you the l-system command syntax it implements, but that is not enough to teach you how to use it. For that, I highly recommend the book mentioned in the LPARSER documentation, excerpted as follows:

"The implemented l-system is based on the one described in the book 'The Algorithmic Beauty of Plants' (ABOP) by P. Prusinkiewicz and A. Lindenmayer (this is where the "l" from l-systems come from). If you want more information on making your own l-systems, you'll want to check out this book. A lot can be done by changing the l-systems supplied with the parser and seeing for yourself what changes in the final form."

Basically, an l-system, or Lindenmayer system, is formal grammar for generating "strings." These strings are a collection of rules. Recursive application of these rules to the initial string results in a string of "fractal" structure. When this string is interpreted as a set of graphical commands, the results can be displayed. As evidenced by the LPARSER output, these l-systems generate realistic-looking plants and other natural objects we see around us.

Object Loading in LightWave

Although LPARSER can create objects in several output file formats, only one of them can be used by LightWave. We are most interested in the output file format called DXF. These DXF-formatted output files can be loaded via TIO into LightWave's Layout. The DXF object file format is one that was designed for use in a popular PC package called AutoCAD. TIO is the module "inside" Layout that comes into effect when a non-LightWave-formatted object is being loaded. Layout will load LightWave objects directly and invoke the TIO module to convert several other formats, including AutoCAD DXF, Sculpt 3D, Swivel 3D, 3D Studio and Wavefront. You don't have to do anything special—just click on Load Object in the **Objects** panel and specify your object file name. TIO will try and determine the object format type. If it does not readily detect the object type it will ask you via a requester to select the object format from a list of supported types. (Syndesis wrote the TIO routines for NewTek and has updated them. Hopefully, we will see them implemented in the newest version of LightWave.) Modeler will only load native LightWave-formatted objects, but once an object is imported via TIO into Layout, it can be further imported into Modeler.

Running LPARSER on the PC

The LPARSER program runs as a command line-driven PC DOS application. LPARSER needs to know the name of the input file with l-system commands to parse and any parameters needed to specify the output file type. For example, to parse the sample fern plant object, you would use the following PC command: lparser -3 fern.ls.

This would invoke the LPARSER program, which would read the l-system commands in the fern.ls file and generate the output DXF fern object in a default file called output.DXF. The -3 parameter tells LPARSER to create a DXF file with 3DFACES. Objects made with 3DFACES are formed with true polygons of three to four vertices. DXF objects generated with this parameter are imported into Layout via TIO and render well. If given the -d parameter, LPARSER will also create DXF-format objects, but they will be formed by two-point "polygons" that are actually just lines. These objects will import into Layout via TIO but do not render with the desired results. Though LightWave can give "thickness" to a line (a two-point polygon), a line really has only one dimension, length.

To have a real surface requires two dimensions of length and width, and thus a minimum of three points. (There are, however, uses for two-point polygons. Mark

Thompson showed us how to construct a great-looking grassy field from two-point polygons in his article from the premiere issue of *LWPRO*, October 1993.) Again, for the objects we are creating via LPARSER to look real, we need them to consist of true polygons.

There are other LPARSER command line parameters mentioned in the documentation that can improve the results of the final object rendering depending on which rendering package is used. (Before LightWave we had other choices!) As can be expected, the rendered results greatly depend on the object being rendered, too. For example, certain object forms may get too "thin," and there is a parameter to vary this option. There is also an option to "connect" shapes that make up "branches" or "trunks" of trees and plants. We all do some experimentation in LightWave to get things just right, and we will need to do some experimentation with the parameters in LPARSER as well.

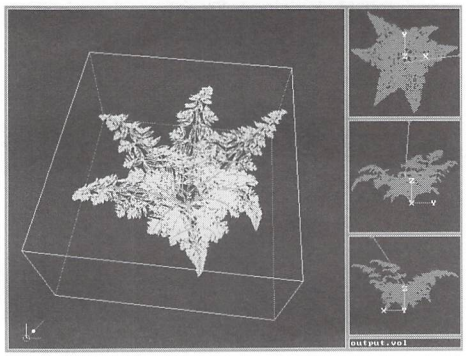


Figure 1

Converting the LPARSER DXF File on the PC

A problem with some of the files created by LPARSER in the DXF 3DFACE format is that they are quite large and take a long time for TIO to import. (Actually, this stems from the design of the DXF 3DFACE file format.) Another time-consuming task is checking for duplicate points or vertices that may be shared by multiple polygons (i.e., "merging points"). To save conversion time on my Amiga, I wrote a PC program called DXF2LW, which will do the conversion from DXF to LightWave and "merge points" to reduce their number. It also chooses a set of surface attributes preferable to TIO's. (DXF2LW, along with its C source code and documentation, is included on the April '95 *LIGHTWAVEPRO* disk.)

Though TIO will import the LPARSER DXF objects, you might like the DXF2LW conversion program better because it runs on the PC and will free up the Amiga. Also, you might find the DXF2LW surface attribute defaults more suitable than those from TIO.

An LPARSER Sample Object

Figure 1 shows a screen from LVIEWER, a companion program included with the LPARSER package. This image shows the results of the sample fern object after being "parsed" or generated by the LPARSER program via the command shown above. (The LVIEWER program will also view or display 3DStudio 3DS binary objects, and 3DStudio objects are importable by TIO into LightWave.) When the objects generated by LPARSER are imported into LightWave and rendered, the results are great! I'll let you judge for yourself. Figure 2 (the fractal tree image in the color pages) shows the results of LightWave rendering a rather simple scene. This scene has an LPARSER-generated tree with some LightWave-generated clouds and hills in the background. The tree object was converted from the DXF LPARSER output format to LightWave format by DXF2LW.

Changing Surface Attributes

After loading or importing one of these LPARSER objects into LightWave and before rendering, you may want to make a few surface parameter adjustments. Actually, this holds true for any DXF object imported via TIO or converted by DXF2LW, since there are not really any surface characteristics besides color that will carry over to the converted object. Both TIO and DXF2LW make somewhat arbitrary but logical selections as to several of the surface attributes available in LightWave that are not present in the DXF-format specifications.

However, both TIO and DXF2LW carry over the color specified for each surface from the original DXF-formatted object file. The following list shows the surface attributes each program selects:

	DXF2LW	TIO
Surface Color	same as DXF	same as DXF
Smoothing	On	Off
Diffuse	100%	0%
Double-Sided	Off	On

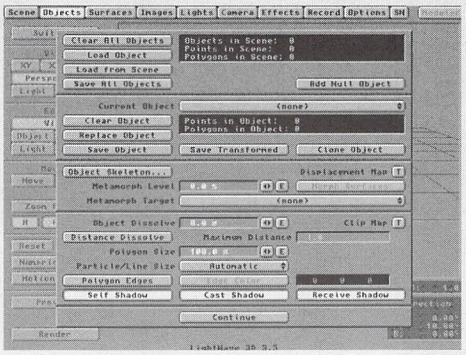
Both DXF2LW and TIO change the axis on which the object is aligned. DXF2LW accomplishes this by exchanging the Y and Z axis point coordinates so the object will be oriented along the Y axis instead of the Z axis. The object is thus "up and down" versus "lying on its side."

Double-sided polygons increase rendering time, so double-sided is turned off by DXF2LW. The object will still render properly because DXF2LW automatically checks polygon orientation. This software uses some rather simple math to determine which plane a polygon can be viewed from and whether the polygon is clockwise or counterclockwise. All of the counterclockwise polygons are "flipped" to verify that they are clockwise in orientation. This flipping ensures that their "surface normals" are all aligned the same way and visible to the camera in their viewable plane. The DXF2LW program displays statistics about the polygons and their orientation as it converts the DXF data to LightWave format.

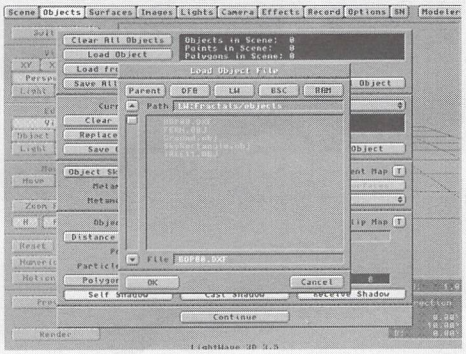
TIO activates **Double Sided** (**Surfaces** panel). Since this function increases rendering time somewhat, you may want to turn it off. Before you render any object with **Double Sided** off, import the object into Modeler and make sure all the polygons are aligned (or flipped) in the proper direction.

As mentioned above, the color scheme for a DXF object is carried over by both my conversion program (DXF2LW) and TIO from the original DXF object. In the equipment originally used to print CAD (computer-aided design) drawings, a different "pen" was used for each color. (Remember, the DXF format is mostly used in

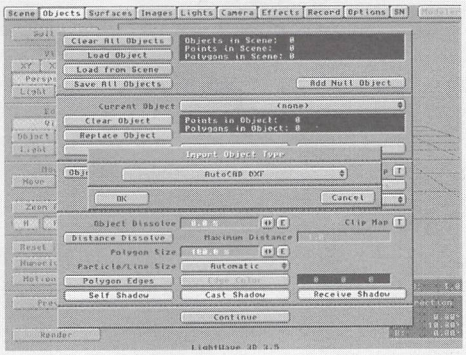
How to Import a DXF-Formatted Object



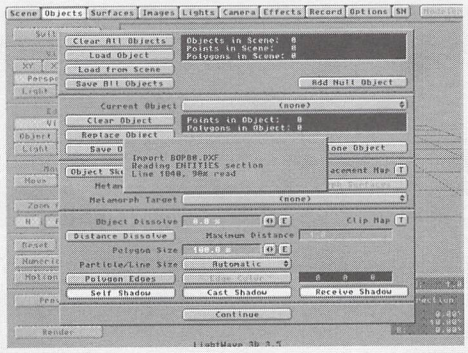
Choose Load Object and select your object's name from the file requester.



Choose Objects from the main Layout screen.



Specify DXF as the object type by using the Up or Down arrows.



As TIO imports and converts the object, it will display the conversion progress.

see Fractal 3D Objects, page 18

Simple Refraction

Creating a Realistic Magnifying Glass

by Dan Ablan

When people think of refraction, most don't consider it a part of everyday 3D jobs. But, if you look around you, many things refract. Incorporating this quality into your animations will bring you one step closer to achieving total realism.

Ever since I fell into 3D animation, I've been like a sponge. Everything having anything to do with 3D, whether high-end or low-end systems, intrigues me—especially when it comes to LightWave. There have been a few animations that, though relatively simple, had some ideas that were so effective, one would wonder how you could overlook such a great idea. With this tutorial, you'll create a magnifying glass, but you could use the following steps for eyeglasses, telescopes, binoculars, etc.

LightWave is great for many reasons, but one that I particularly admire is its ability to mimic real-world properties. This magnifying glass idea has always been in the back of my mind, and finally, one day, I decided to try it. In about five minutes, I had the look I was after. Later, I worked a little harder and longer on the idea, and thanks to LightWave, made a pretty convincing magnifying glass. As with just about anything in LightWave, by setting up objects, images and lights as you would in a real setting, you can pull off what the *Revolution* tape called "image miracles." Enough jabber, though—on with the tutorial.

For the magnifying glass, let's begin in Modeler.

Making The Glass

Whenever you model anything, it is always best to have that thing right in front of you, or at least a photograph of it. You may know exactly how something looks, but once you've modeled it, for some reason, it just doesn't look right. When you actually get a hold of that thing you modeled, you see the very subtleties that make that thing unique. It's the same for a magnifying glass. Yes, I know exactly how it should look, but, inevitably, it just doesn't look right if I build it from memory. By having a real magnifying glass in front of me, I can see the proportions, surfaces, and most importantly, how it reacts to its surroundings. Since I've built a magnifying glass already, you don't have to run out and get one for this tutorial.

- Clear out Modeler by clicking **New** (**Objects** menu or N). The first thing to do is build the glass. If you had that magnifying glass in front of you, you'd see that the glass portion is convex, like a contact lens. This is very important to model correctly because its shape, with refraction and the right surface properties, determines how realistic it will look when rendered. I wanted the glass to curve outward, so I chose to use the **Magnet** tool.
- First, create a disc in the Face view. Use the numeric requester and enter the following settings in **mm**:

Sides	40
Segments	1
Axis	Z
Center	0, 0, 0
Radii	1.5, 1.5, 0

Click **OK**, then the (a) key to fit all views.

- In another layer, create a box, using these settings with the numeric requester:

Low	-2.5, -2, 0
High	2.5, 2, 0
Segments	X = 20, Y = 20

The box will be used to create an even template to make the disc malleable.

- Go back to layer one and select layer two (the box layer) as your background. Pull out the face window view for a larger work area. **Extrude** the disc just 1 mm and then center it on the Z axis. (If you have not changed Modeler's config file, your F2 key should be center on one axis macro.)
- Choose **Boolean** (B) from the **Tools** menu and click **Intersect**. If you come up with extra points and your disc is not flat, delete the points that are off of 0, on the -Z axis.

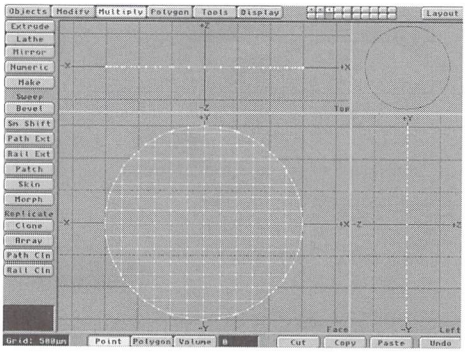


Figure 1

In a minute or so, you should have a disc that is made up of even polygons in the X and Y axes (Figure 1). The reason it's done this way that because by tripling and subdividing just a disc, you'd get a mess of uneven polygons, and it would be hard to use, and...well, it's a mess. Anyway, once the disc is made, save it as "disc flat." You won't need to use it like this in Layout; we're saving it just in case.

- Under the **Modify** menu, select the Magnet tool. With the left mouse button, drag out to cover the whole disc (Figure 2). Next, with the right mouse button, from the top view, drag out just a little, and you'll see the disc begin to curve outward. The larger the magnet area

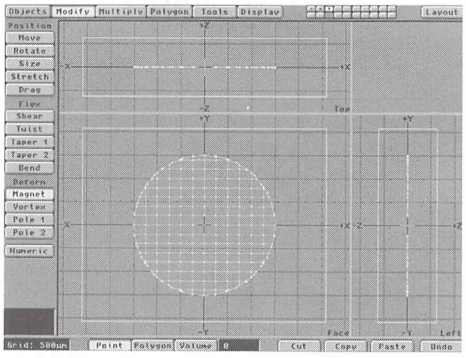


Figure 2

(left mouse), the larger the influence (right mouse) will be. You should end up with a disc that looks something like Figure 3. Save it as "disc magnet." Again, you won't need this particular piece in Layout, but if you screw up, or crash, it's much easier to reload this object, and later delete it, than to go through the steps again.

- Now, give the disc the surface name "glass." Then, using the **Mirror** tool under the **Multiply** menu, mirror the curved disc against itself so you have a two-sided curve (Figure 4). **Merge** points (m) to get rid of any duplicate points. **Save** this as "mag.glass."
- Now you need to build the edge of the magnifying glass that holds the lens in place. In a clean layer, make a disc using the numeric requester and these settings in **mm**:

Sides	40
Segments	1
Axis	Z
Center	0, 0, 0

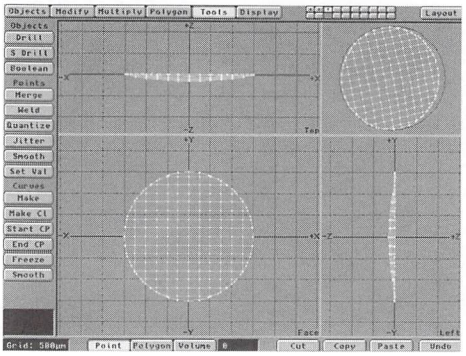


Figure 3

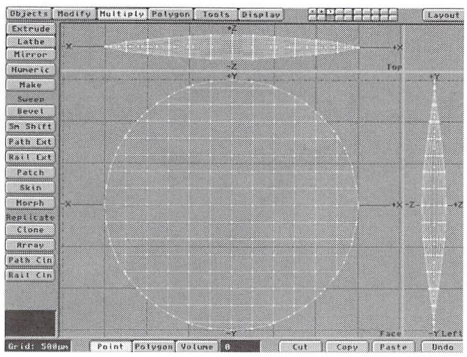


Figure 4

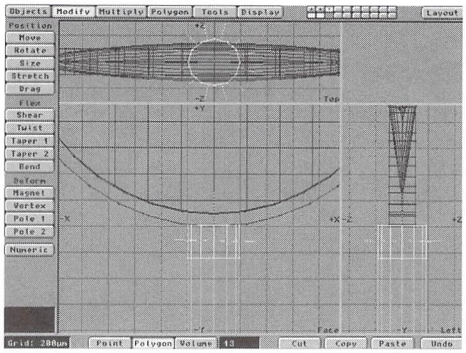


Figure 5

Figure 6: Using the proper refraction settings, a magnifying glass enlarges what's behind it.

Figure 7: As the magnifying glass gets closer to the object, it magnifies less, as would a real magnifying glass.

X	1.6
Y	.6
Z	0

- Copy this disc to another layer. Select **Size** (H) from the **Modify** menu and click **Numeric** (n). Enter **Factor** of 0.94 to scale it down just a bit. Next, **Extrude** (**Multiply** menu) it .2 mm on the Z axis. Finally, center it on the Z axis.
- Go back to the layer that has the larger disc and put the smaller extruded disc in the background layer. Select **Boolean** from the **Tools** menu (B) and hit **Subtract**. When the hole has been cut in the larger flat disc, **Extrude** it .2 mm on the Z axis. Center it on the Z axis, and give the hole a **Surface** name of edge. Finally, save it.

Now all that is left is the neck and handle. You can model these in the same step, by making one disc and surfacing two different areas. I made mine look like a fancy wood-type handle, but for now, just make a disc.

- Select **Disc**, and using the numeric requester, enter these settings in **mm**:

Sides	12
Segments	2
Bottom	-4.7
Top	-1.59
Axis	Y
Center	0,-3.145,0
Radii	0.195,1.555,0.18

- Using the right mouse button, select the center points with the lasso feature. Move these points to about -1.84 mm to create the neck area of the handle (Figure 5). Next, select just those polygons using the lasso tool again (right mouse button) and surface them as "neck." Surface the remainder of the handle as "handle." Save this, too.
- Now it's time to put it all together. The best way to join the pieces is through the Boolean operations. However, you can copy and paste all items to the same layer. Once you have all the pieces together, hit the (m) key to merge points. Export this object to layout, saving it as "MagnifyGlass."

Making It Work

Surfacing this thing is relatively simple. Place your favorite wood surface or image around the handle and use a silver/metal surface for the neck and edge. The glass is also easy. Use these settings:

Texture Color	172, 187, 200
Diffusion	95
Specularity	75
Glossiness	High
Reflective	5
Reflected Image	Fractal Reflections
Transparency	95
Refractive Index	1.55
Smoothing	On
Max Smoothing Angle	10

- Under the **Camera** menu, turn on **Trace Refraction**. Set the render resolution to Low for quick render tests. Refraction is the key here. As the magnifying glass is closer to the camera and farther away from another object, it will appear very magnified, as in the color image (Figure 6). I chose to just use a scan of a $5 bill as an object. You could use a scan of a page of words, a table with many objects spread across it, or a newspaper.
- As the magnifying glass gets closer to the object, it doesn't magnify as much (Figure 7). If you have a magnifying glass near you, look through it, and move it toward and away from a piece of paper on your desk. See how it reacts? It's the same in LightWave. The only difference is that you can't start a campfire in LightWave with a concentrated light source.

Refraction is interesting. To make it work properly in LightWave, you usually need more than one refraction index on the same object. Light enters one part of the object, refracts, travels through another part, and leaves through yet another. Realistically, you should have a refraction setting for each part, but this magnifying glass isn't quite thick enough to warrant that. If you were modeling a glass of water, however, you would set refraction for where the light enters, the water, and where the light leaves. [Ed itor's note: For a more detailed description of refraction properties, see Mark Thompson's "Understanding Refraction," *LWPRO*, January '95]

Other Ideas

I recently saw a great animation from a company in Europe. Called "Invisible Man," it was black and white and had a terrific idea. In one part of the piece, the invisible man pulls a pair of glasses out of a desk drawer and proceeds to put them on. You (the camera) are watching this move from the character's perspective. When he brings the glasses up to his face (camera view), you see the objects on the desk in front of him enlarge and slightly deform, just as if you were actually putting on a pair of glasses. The above steps could be used the same way for building a pair of glasses, or a telescope. Or how about a pair of binoculars? Even if you only do logos with LightWave, why not have a magnifying glass travel across the logo, instead of a typical glint of light?

With your eyes open, watching real-world properties, plus a basic understanding of LightWave, you can create those "image miracles." It's those little differences that will really make your animations stand out.

LWP

Dan Ablan is a LightWave animator for AGA, based in Chicago. He can be reached at (312) 239-7957 or via Internet at dma@mcs.com.

LightWave 101

Holding Curves With Metaform

By David Jones

Welcome back to "LightWave 101"! It's been awhile since this column has appeared in *LWPRO* due to the fact that Taylor Kurosaki, who used to write it, is no longer able to do so because of his busy schedule with Naughty Dog, a computer game development company. I'll be taking over the reins.

This month's installment will cover the new object-modeling tool Metaform. In case any readers have been living under a rock or out exploring uncharted desert regions for the last six months, I'll give a short description of this powerful feature:

Metaform, a tool found in Modeler's **Subdiv** requester (**Polygon** menu), allows you to create a primitive object, or metabox. Then, by simply pressing a button, Metaform turns this primitive angled shape into a smooth, organic model. Now the problem is that if you only create a primitive in the general shape and size of what you want the final object to look like and press the button, you will most likely end up with an amorphous blob instead of the nice organic model you expected. Don't worry—this happens to just about everyone the first few times they experiment with Metaform. So don't panic.

When LightWave 3.5 first arrived, I was very excited about Metaform. After playing with it for several hours, though, I came to the conclusion that Metaform was going to be useful for making blob-type objects but not much else. It could not possibly replace spline modeling for building complex-looking objects like automobiles and airplanes, which require precise curves. There just wasn't enough control. Well, let me tell you that I could not have been more wrong. Not only can Metaform produce just about any smooth object you can think of, but it can also do it quickly, and is nowhere near as complicated as modeling with curves or splines.

Before using any modeling tool, know the requirements that need to be met for it to function correctly. Metaform, thankfully, has only a few prerequisites that need to be met for it to work properly. Let's go over them:

The main requirement is that your primitive object—or metabox, as I like to call it—consist sentirely of three- or four-sided polygons. Metaform will not work with polygons that have been created using more than four points and doesn't like triangles (three-sided polygons) as much as four-sided polygons. Although it will function with them, it has a tendency to cinch up in the place where you have triangles. So remember, always try to use quads (four-sided polygons) if possible when making your primitive.

The only other requirement for Metaform is that your object be closed up on all sides. You should not leave any open ends in the geometry of the primitive or else strange things can happen when you Metaform. Other than that, only the general rules apply. These include not having any duplicate points or polygons; therefore, use **Merge** points (**Tools** menu) and **Unify** polygons (**Polygon** menu) before you Metaform. Additionally, the object should be tripled before rendering.

That's about it for requirements. There's nothing better than the hands-on approach, so let's get started on the tutorial.

- Go into Modeler and select the Box tool (Objects). Hit (n) for numeric and accept the default values by clicking OK. Now hit return to create the box.
- Press the (a) key on the keyboard to autosize the display. Then go to the Polygon menu and click on the Subdiv (Subdivide) button. You'll see the requester shown in Figure 1.

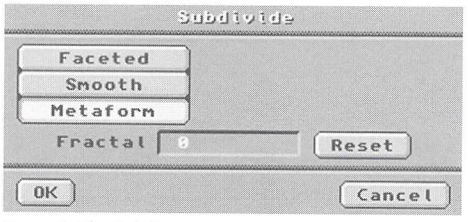


Figure 1: The Subdiv requester

- If **Metaform** is not already highlighted, select it by clicking on it. Then click on OK. Your cube has gone through a one-level transformation with Metaform and should now look like Figure 2. Click **Metaform** two more times. Your cube has turned into a sphere.

This is as good a place as any to stop and think about what I was discussing earlier. Let's review for a moment. You made a box, then hit **Metaform**. The cube started collapsing in on itself and got rounder with each subsequent metaform until finally becoming a sphere. However, what if my intention was to make a box with rounded edges, like dice (something with tight curves and closely rounded edges)? Now, pay close attention, because this may seem trivial, but it opens up many exciting possibilities. If you can control how tight the metaformed curve will be, just about anything can be created with it.

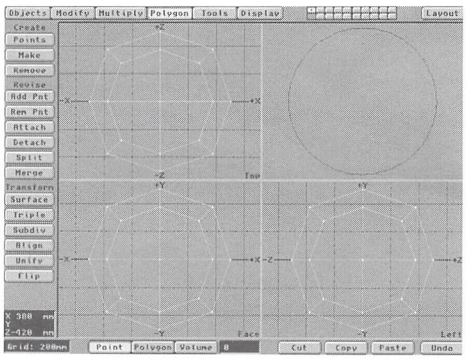


Figure 2

- Go into another layer, build the default box again and autosize it with the (a) key. Now select the **Multiply** button at the top of the screen, find **Bevel** and click on it.
- In the Bevel requester, enter an **Inset** of 35 and a **Shift** of 0. Make sure to set your Units to mm. Your requester window should look like Figure 3. Select **OK** and watch as Bevel adds a new set of polygons on the same plane as the originals, but inset 35 mm toward the center.

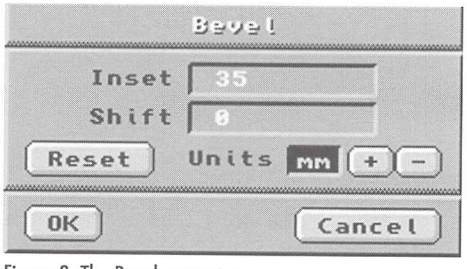


Figure 3: The Bevel requester

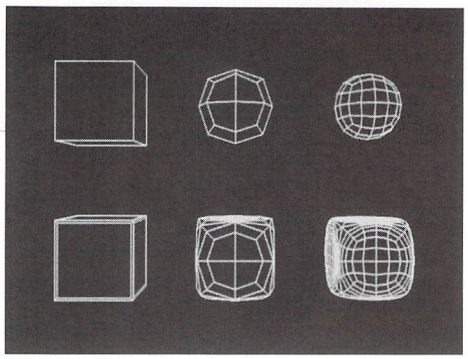


Figure 4: The top sequence shows a normal cube metaformed into a sphere. The bottom sequence shows how beveling the edges of the cube first helps contain the metaform.

- Now **Metaform** the primitive again, making sure no polygons are selected. Notice how the edges of the box hold their shape and don't collapse inward as much as they did before. Metaform the object two more times and it should look like Figure 4.
- Once this has been completed, hit (d) on the keyboard to bring up the display options and select a **Moving Solid** preview. Rotate the object around in the preview window and examine the edges. Observe how they are nice and round but still have tight curves. This time, when you metaformed the object, it didn't turn into a sphere. The reason is that the new polygons added to the box by the bevel operation acted as restraining polygons; thus, Metaform kept the curves it created tight.

What this should tell you is that if you want to make a tight curve, add extra polygons where you want the resulting curve to be tight. Metaform looks at the space between the polygons, and when it subdivides them it creates new polygons at the halfway point. If you already have polygons that are close together in the primitive, Metaform still adds new geometry there, but it won't pull it in and curve it as much.

I hope that this simple tutorial was helpful in illustrating how to hold curves with Metaform. Although beveling is only one way to add extra geometry to a primitive, it is used frequently and seems to work well. Remember that the best way to learn a new tool is to use it, and this advice is especially true with Metaform. So what are you waiting for? Start modeling!

LWP

David Jones is an animator for Amblin Imaging currently working on Star Trek: Voyager.

Tearing Through Canyons

continued from page 5

scape. You can avoid this problem in a couple of ways. The easiest is to run an object that would believably be part of the landscape (a tree, for example) between the camera and the flying object every now and then. Another way, which is a topic for another time, would be to use the same image sequence as the **Foreground Image** sequence and a companion **FG Alpha Image** sequence to "manifest" those portions of the landscape behind which you want to fly into the foreground.

Close Enough for...

This method of using a 2D background movie to simulate flying through a 3D terrain may be eschewed by the 3D purist simply because it uses 2D animation. But the fact remains that this method can actually reduce both rendering time and the total time for completion of the project. It is also true that you can get a more realistic-looking landscape in a shorter period of time than if you try to create a spline patch landscape and then paint it. But the real advantage is that when you hear "And make it fly through a killer landscape," you know the landscape is actually going to be the easiest part.

The next time you have a difficult 3D problem, don't let any potential 2D solutions go unconsidered.

LWP

Wayne Cole is the proprietor of Infinity Heart Enterprises in Santa Barbara, Calif. He can be reached at (805) 964-9540, or via CompuServe at 76370,621.

Have Starfield, Will Travel

continued from page 9

offering this additional tip. As seen in the end credits for *Deep Space Nine*, the principle of parallax provides a more accurate representation of what warping through space might look like. This means that as we are traveling forward (or any direction, for that matter), the closest stars will appear to move the greatest, causing the longest blurs, and the most distant stars might not appear to move at all. Two possible solutions exist to mimic this phenomenon. Either build a completely realistic, physically accurate spherical starfield, as we did earlier (scaled extremely large), to act as a mock galaxy, or create a "shell" of particles surrounding the warp column to act as a static background wall of stars (Figure 5). In both cases, make sure to set a Distance Dissolve (**Objects** panel) for the star objects you'll be warping through. Aren't space scenes fun?!

Docking Bay

Here are some astronomical constants that might be useful:

1 astronomical unit = 1.5 x 1011 m
1 light year = 9.4605 x 1015 m
1 parsec = 206,265 A.U. or 3.262 light years
Milky Way radius = ≈55,000 light years
Number of stars in the Milky Way = >200 billion

Next time, we'll tackle some of the more interesting stellar phenomena, such as globular clusters, spiral galaxies, black holes, comets and solar flares. Until then, keep looking up!

LWP

Send questions, comments or frozen pizzas to Blue-Line Imaging, c/o William Frawley, 315 W. Fifth Street, Muscataine, IA 52761.

Editor's Message

continued from page 3

but not angry. If you are depending upon materials that do not exist yet, you are making a very unwise business decision. A product will ship when it's done (and sometimes a bit before!).

Now that I've had my little tirade, I think it would be nice to tell you about some of the new features in LightWave 4.0. This, of course, is just a quick list. We'll go into much more detail in the next and upcoming issues.

The most exciting thing about the new LightWave is definitely its ability to allow third-party manufacturers to supply plug-in capabilities. Layout will provide for displacement, image post-processing, motion and shader filters. Note the word "shader": these filters are much more advanced than simple textures. Modeler allows support for plug-in macros and custom tools.

Layout's Objects panel now allows you to make an object unaffected by fog or unseen by ray tracing. Both options can be very handy for composition work. Bones now allow for a minimum and maximum influence range.

It seems each new version of LightWave introduces something you can't see how you ever did without. Version 2.0 brought us automatic texture sizing. 3.0 introduced us to macros. And 3.5 brought us Metaform. I believe 4.0's must-have feature may very well be its surface samples. This feature gives you the ability to quickly render out small tests of a surface as many times as needed. On the Amiga, they render out in rows on your selected display device. On non-Amiga versions, they render out in a vertical "filmstrip" mounted on the edge of the Surfaces panel. By the way, every once in a while, a LightWave option comes along that just doesn't seem to make sense to some people. (An animator I know used to wonder why anyone would want a "Clear Light.") The Surface Sphere Diameter may be such an option. Defaulted to 1.0, it lets you change the relative size of the sample spheres that you can render. The sample sphere doesn't actually render out any larger, but you will notice a texture change if you adjust to this value. Remember to give this option a value close to the size of your surface, or you won't get accurate-looking results. I've made the mistake myself of not seeing my texture on the sample sphere when I was trying to map a planet, but instead looking at a 1-meter sphere.

Layout now supports Flyer clips. This should be a welcome addition for those with a Flyer edit suite.

Layout's Camera panel has been revamped as well, with the addition of custom pixel aspects and a field-of-view readout. I just recently used the custom pixel aspect feature to complete a cinemascope film teaser for the movie CutThroat Island (check out the end of the preview for the LightWave stuff). There are now frame boundaries in the camera view that dynamically change whenever you adjust resolution and pixel aspects. I'm sure you'll agree they are a great addition.

Modeler now sports a few more layers and (ta da!) multiple undo/redo. You can even adjust the number of undo/redo operations available. There are a few new jitter and array options and Metaform now allows you to use a smoothing angle to "hold" sharp edges.

The units field in all requesters is now absent. Instead, you can just type in the abbreviation of the unit you wish to use and Modeler will automatically convert it to the units of measurement you have set in the Display options. Along with this new feature comes another valuable one: the ability to perform basic mathematical functions in the numeric field of requesters. Simply type a formula such as "21.527m + 19.34in" and Modeler will insert the answer in the field when you hit return. Very nice.

Add in the ability to view solid, static models and save separate layer, zoom and window positions onto numeric keys, and you round out some of Modeler's new capabilities.

As I mentioned above, we'll be covering these and more new features in upcoming issues.

Oh, by the way, if for some reason LightWave hasn't shipped yet, forget I said anything.

LWP

Fractal 3D Objects

continued from page 13

CAD-type PC programs.) That is why both TIO and DXF2LW use a surface name beginning with "PEN" for each color or surface. There is a maximum of eight surfaces/colors used in the conversion as follows:

Surface Name	Color	R	G	B
PEN0	Black	0	0	0
PEN1	Red	255	0	0
PEN2	Yellow	255	255	0
PEN3	Green	0	255	0
PEN4	Cyan	0	255	255
PEN5	Blue	0	0	255
PEN6	Magenta	255	0	255
PEN7	White	255	255	255

If a color "number" in the DXF object is higher than seven, seven is subtracted from it until it fits into the above color scheme when converted by the DXF2LW program. TIO also fits the DXF object colors into the above scheme. DXF2LW was written to support only the DXF "entity" called 3DFACE. While this conversion program works for objects generated with the proper LPARSER command, it does not work on all DXF-formatted object files. TIO supports more of the DXF format than DXF2LW does. The appendix of the LightWave manual contains more details on the TIO conversion program, the DXF format and other formats supported by the TIO program.

Some of the sample "critter" or "animal" objects look good with their LPARSER-generated color settings. However, the plant objects will need to be changed. For plants, trees and leaves generated by LPARSER, it's best to change yellow to brown and all other colors to a shade of green. This step might save you the time of rendering in Layout or selecting polygons in Modeler to see what colors/surfaces are attached to what polygons.

As you might guess, there are different surfaces (or sets of polygons) generated for the "branches" and "leaves." For some of the "flower" objects, you will need to experiment with the colors, as these objects look more realistic with more colors than just browns or greens. If you take the time to learn the LPARSER-implemented l-system commands for creating your own objects, it will be easier to originally select the color and thus the surface you want for each object part. When constructing your own LPARSER objects, you have control over the parts of the object being created. These parts can be given different colors and thus treated as separate surfaces once imported into LightWave.

Also, if you are loading more than one object that was originally in DXF format, remember to change the surface names to something meaningful before loading a second object. If you load more than one object with these same (DXF default) surface names, the objects will share these names and their individual surfaces cannot be changed independently.

Another surface attribute that needs to be changed is the diffuse setting. It will be set to 0% by TIO and should be set to 100% or somewhat less for each surface to be seen. I'll leave it up to you to adjust any of the other attributes to create the effect you want.

Object Sizing

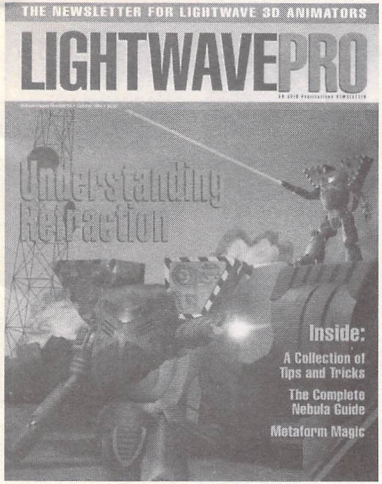
The size of the DXF object may not quite be what you would expect after it is imported into LightWave, because the unit of measurement is not carried over by the conversion process. The original object might have been created in inches or feet. When the object is imported into LightWave, the unit of measurement is assumed to be meters. To fix this problem, it can be resized in Layout, rescaled in Modeler or converted to a proper size using the Object to Metric macro (MetricScale.lwm) in Modeler.

Should You Be Using LPARSER?

Some of the sample LPARSER-generated objects do have a high point/polygon count, which increases rendering time, but I think the results are worth it. By changing the original commands in the LPARSER input file you do have some control over the size of the objects created. Even if you do not take the time to learn the l-system commands needed to make your own original objects, the samples provided may be a welcome addition to your collection. If you don't have access to a PC to generate these sample objects, I have converted them all into LightWave format for you. They should all be available on the next LIGHT-ROM CD from Amiga Library Services.

LWP

Earl Terwilliger can be reached on the Internet at aisterwi@acs.eku.edu or on CompuServe at 70575,1330.



TO ORDER WITH YOUR VISA OR MASTERCARD CALL TOLL FREE!

1(800) 322-2843

LIGHTWAVEPRO

THE NEWSLETTER FOR LIGHTWAVE 3D® ANIMATORS

Make a Smart Investment

Subscribe Today!

Canadian/Mexico: Add $US12
Overseas: Add $US36
Allow 6-8 weeks for delivery.

Make checks payable to LIGHTWAVEPRO.
Prepayment required on all overseas orders.

YES! Enter my one-year (12 issues) subscription to LIGHTWAVEPRO at the Special Introductory Rate of only $48—that's 50% off the cover price!

Name_____

Address_____ Apt. #_____

City_____ State _____ Zip _____

☐ Payment Enclosed ☐ Bill Me

BFQLW

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

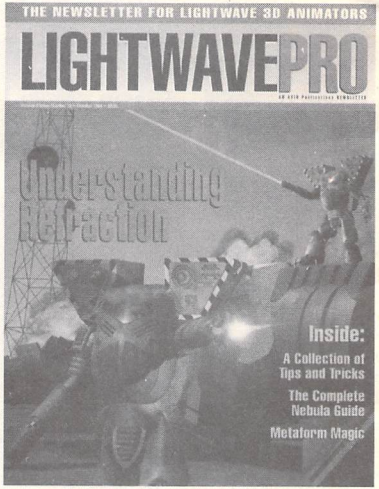
BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 2263 SUNNYVALE,CA

POSTAGE WILL BE PAID BY THE ADDRESSEE

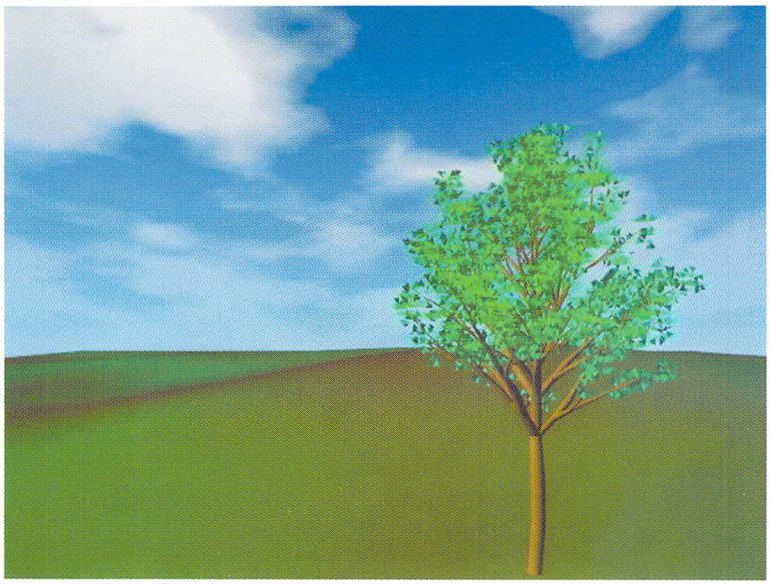
LIGHTWAVEPRO

273 North Mathilda Avenue
Sunnyvale, CA 94086-9313



TO ORDER WITH YOUR VISA OR MASTERCARD CALL TOLL FREE!

1(800) 322-2843



Fractal Tree

A LightWave rendering of an LPARSER-generated 3D fractal tree. See "Fractal 3D Objects," page 12.

Image copyright 1995 Earl Terwilliger

Simple Refraction

Using the proper refraction settings, a magnifying glass enlarges what's behind it. See "Simple Refraction," page 14.

Copyright 1995 Dan Ablan

Pleiades

A total of 56,000 particles comprise this starfield, 40,000 of which are set to render Particle Sizes as "Small," 15,000 "Medium" and 1,000 as "Large." Lens flares were used as local suns and dust clouds were planar mapped for enhanced effect. See "Have Starfield, Will Travel," page 8.

Copyright 1995 Bill Frawley

See What Happens When Everyone Puts Their Heads Together.

Raptor 3 is twice as powerful as anything you've seen before.

Sink your teeth into Raptor 3 with Alpha 21164 microprocessor, the most powerful Windows NT workstation and CPU combination in the world. With Alpha 21164, Raptor 3 is now twice as fast as any Raptor to date. For Lightwave users, that means this one machine can take bigger bites than ever of your 3D rendering tasks. What's more, Raptor 3 is processor independent. In other words, you switch CPUs, not computers, when you upgrade. Or start out with another RISC microprocessor like MIPS R4600 or R4700 and upgrade later. Turn your attention to the motherboard and you'll find it delivers unparalleled I/O capability with 4 PCI slots, 2 ISA slots and twin SCSI ports, making it easy to use Windows NT's disk striping feature that can double hard disk performance. And 8 SIMM sockets provide capacity for up to one gigabyte of main memory. Best of all, Raptor 3 is from DeskStation Technology. The company that helped give birth to the 3D animation rendering industry and continues to reinvent it with an insatiable appetite for research, development and innovation. The result? For 3D rendering, Raptor 3 with Alpha 21164 is head and shoulders above anything else.

Raptor 3™

For more on Raptor 3 at twice the speed, call (800) 793-3375

DeskStation Technology