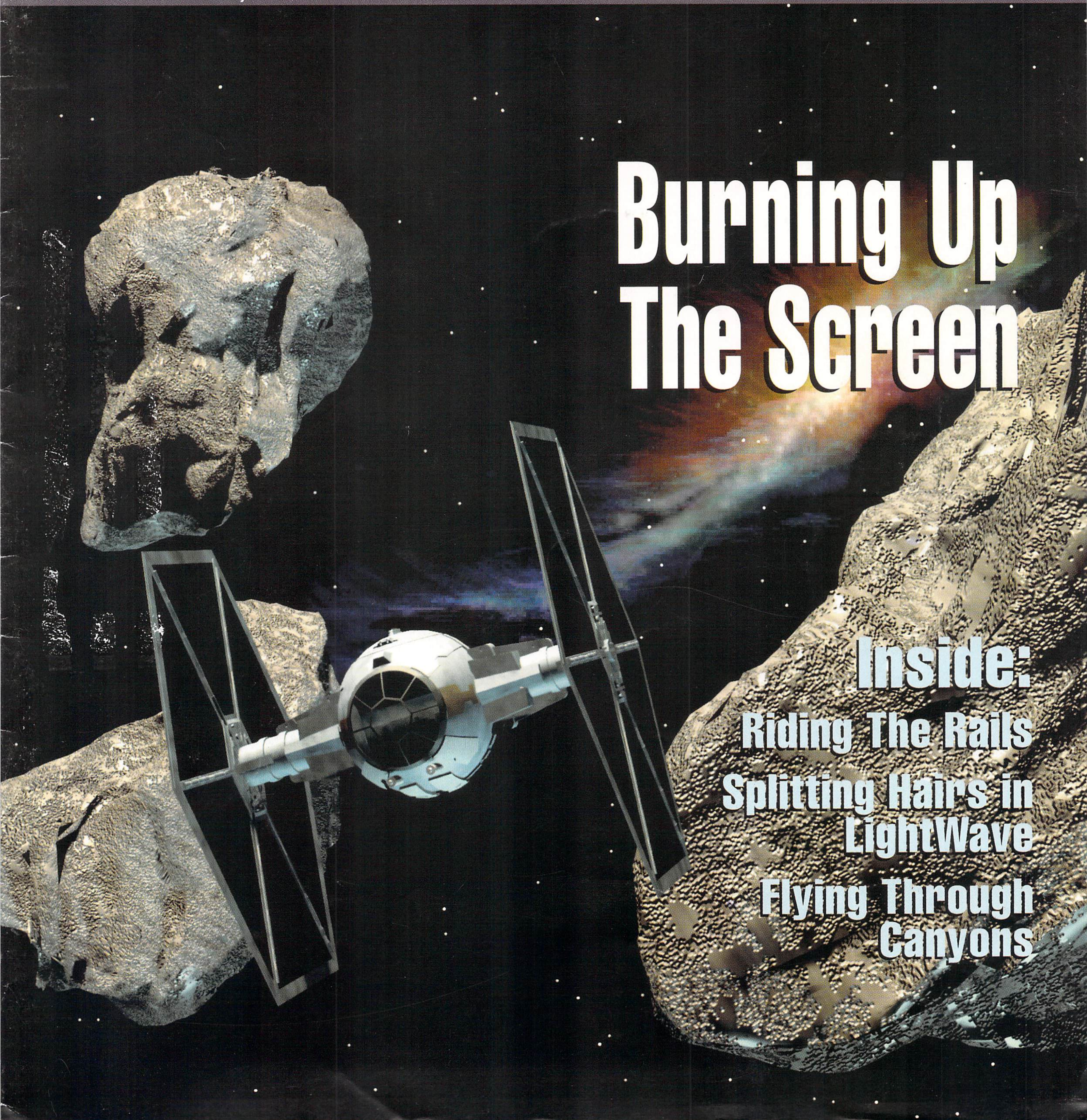
THE NEWSLETTER FOR LIGHTWAVE 3D ANIMATORS

LIGHTWAVEPRO

an Avid Media Group, Inc. newsletter

Volume 3 Issue Number 1 • January 1995 • $8.00



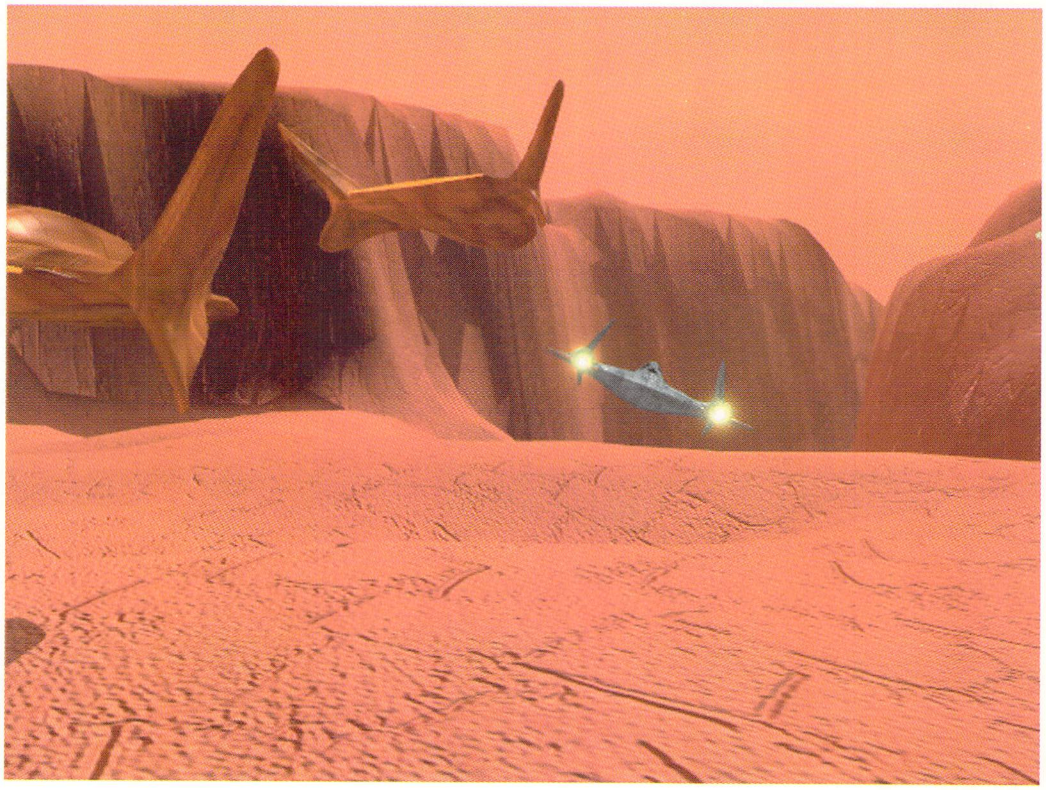
Burning Up The Screen

Inside:

Riding The Rails

Splitting Hairs in LightWave

Flying Through Canyons



The Chase

A rendering of spaceships chasing through a displacement-mapped canyon, taken from an intro sequence by Zapa Digital Arts for a soon-to-be-released game.

Compro Games Copyright 1994

Rail Extrude Table

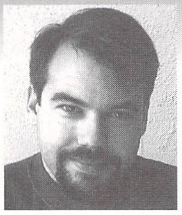
The legs and scroll work of the glass table in this image were created by using Modeler's Rail Extrude features.

Image by Arnie Boedecker Copyright 1994

Fireball

Adding a small cluster of lens flares to the tip gives the fireball added menace and destructive power.

Image by Skyvision Entertainment Copyright 1994



EDITOR'S MESSAGE

by John Gross

With the beginning of a new year, it is traditional to reflect upon the old and look forward to the new. With that in mind, let's cover some of the latest LightWave news and talk about the future.

Video Toaster Expo

Over the years, the Video Toaster has given birth to a slew of third-party products (not to mention a few magazines). So many, in fact, that it has now given birth to its own expo. In mid-December, Avid Media Group hosted the first Video Toaster exposition in Universal City, Calif.

Frankly, I was amazed at the number of people that showed up for this event. There was such a great turnout that the show hours were even extended on the first day to provide attendees enough time to see everything.

There were a lot of great products at the show, which I am sure you will hear about in future issues of *Video Toaster User*. LightWave was certainly well-represented at the show, running on Amigas, PCs and SGIs. Both Carrera and Aspen were there showing ScreamerNet rendering on their Alpha-based systems. Radiosity Software was there showing WaveMaker 2.0 (formerly published by Axiom Software), which includes new elements, improved features and a new Image Factory function. Dynamic Realities was showing a beta version of Impact! (which started life as Newton's Law). It has a nice-looking interface, and if it lives up to its feature list, it should be a great program. One of my favorite software products was Hester and associates' Plug-Ins and Go macros for LightWave. Not only are there a ton of macros for Modeler, but there are macros for doing a lot of other things, such as calculators for just about everything under the sun, including render time calculation and tire rotation/speed calculators. All in all, there are about 150 macros. Expect more information on these and other products as they are released into the market. (Actually, as of this writing, WaveMaker 2.0 is released and selling—contact your local dealer or Radiosity Software at 612-787-0855.)

My vote for Favorite Celebrity Attending the Expo goes to Dick Van Dyke,

see Editor's Message, page 9

TABLE OF CONTENTS

4 Great Balls of Fire
by Colin Cunningham
There are a million different ways to create realistic fire and effects in LightWave. Here are just a few used to burn up the screen on *RoboCop: The Series*.

6 Flying Through Canyons
by Nir Hermoni
Learn how to produce a detailed polygon mesh terrain and assign convincing paths to the objects zooming through a desert landscape.

8 Fur and Hair
by Gonzalo Garramuno
For years, computer artists struggled to create realistic fur and hair through bump maps and textures with little success. Since the arrival of LightWave, all that has changed.

10 LightWood in LightWave
by Dan Ablan
Time is running short and the task of generating realistic textures on deadline is growing more difficult by the minute. With the help of LightWave, TPaint and this tutorial, your problems are solved.

12 Popular 3D Rendering Algorithms
by William Frawley
Part one of a series begins with an exposition on the geometrics of 3D space and framebuffers before examining the concept of Z-buffer rendering.

14 Riding the Rails
by Arnie Boedecker
The discussion of LightWave's underappreciated tools continues with this tutorial on creating a glass table object with the Rail Extrude tool.

16 Digital Cinematography
by John F.K. Parenteau
In this month's column, take a look at some specific examples of shots that are fairly difficult in live action, but quite possible in CGI.

LIGHTWAVEPRO
an Avid Media Group, Inc. newsletter

Editor	John Gross
Managing Editor	Jim Plant
Editorial Coordinator	Douglas Carey
Art Director	Helga Nahapetian Taylor
Art/Production Coordinator	Kristin Fladager
Production	Sergio "Berimbau" Miller
Associate Editors	Joan Burke
	Corey Cohen
Circulation Director	Sherry Thomas-Zon
Circulation Assistants	Tracy Ann-Sparks
	Debra Goldsworthy
Contributing Writers	Dan Ablan
	Arnie Boedecker
	Colin Cunningham
	William Frawley
	Gonzalo Garramuno
	Nir Hermoni
	John F.K. Parenteau
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)
Printed in the USA© 1995 Avid Media Group, Inc.
Are you interested in writing for *LIGHTWAVEPRO* or submitting images? If so, contact us at our offices or electronically.
Avid Media Group, Inc. its employees or freelancers are not responsible for any injury or property damage resulting from the application of any information in *LIGHTWAVEPRO*.

LIGHTWAVEPRO (Vol. 3, No. 1); (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: This month's cover image was created by Amblin Imaging's Greg Teegarden. The asteroid textures and the nebula were hand-painted in Photoshop. The building of the objects and the scene generation took approximately five hours to complete.

Great Balls of Fire

Hellfire & Brimstone, LightWave-style

by Colin Cunningham

Jerry Lee sure had the right idea, pounding away on his piano and then burning it to the ground for an encore. While that cat could torch a stage without batting an eyelid, mastering flames on the CG front is tricky enough to stop even The Killer dead in his tracks. There are a billion different ways to create realistic fire and heat effects in LightWave, so I'll primarily cover a few techniques used to burn up the screen on *RoboCop: The Series*. They may not be suited for all situations, but hopefully they'll inspire all you virtual arsonists out there to create your very own completely non-lethal pyrotechnics.

In episode 17, "Heartbreakers," our metal hero must battle corruption at the highest level and recover the stolen prototype for a weapon called the Heartbreaker, a microwave-emitting gun that can cause instantaneous heart attacks in its victims. The climax of the show has RoboCop being blasted by the device, an effect I was asked to develop. Since the beam of the gun is invisible (it's just a large microwave oven), I was halfway done already. I began work on the main effect: making Robo's body armor glow red-hot as he is bombarded by the deadly rays. After piling through a dozen harebrained schemes, including actually torching Robo actor Richard Eden (a suggestion not appreciated by the producers), I finally settled on using my old friend the lens flare. Though this was met by dirty looks and verbal abuse from my FX buddies, I thought it would work just fine. There is a saying I'm sure you've all heard at one time or another, "When in doubt, use lens flares," and for this particular effect, that adage couldn't be more true. Though impressive, lens flares tend to be overused, especially the kind that make your eyes water, with lens reflections and random streaks firing out of every corner. While lens flare abuse should certainly be punishable by death (or, more fitting to this article, being torched), I found flares to be a key element in simulating fire FX. When used creatively, they can add subtle realism to most scenes.

Glowing Armor

My first few attempts at making a nice glow effect involved positioning some flares over Robo's body armor. Just as my co-workers suspected, the results looked like lens flares, each of them clearly visible. The flares also lacked the concentrated, uniform glow we desired; dissolving the glow 50 percent blended them together nicely, but the overall glow was too soft and not intense enough. The solution rested in the way the flares were spaced (Figure 1). Glow A consists of five flares arranged in a circular pattern; each flare is set at 100 percent glow with 0 percent dissolve. As you can see, each of the five flares is quite noticeable. Glow B, however, is made up of 13 flares arranged in a similar but tighter pattern. The flares range from 100 percent glow at the center to 40 percent glow and 20 percent dissolve at the outer edge of the circle. By adding more flares and placing them closer together, we allow the central glows to bleed together, forming a uniform and more intense glow. Using this cluster technique, it's possible to create complex shapes made entirely of lens flares.

There was more to the shot than I had first thought, however. Robo's armor had to gradually glow red-hot as the camera tracked him stumbling backward into a wall. While it sounds like a difficult task, LightWave's new Background Layout Preview function simplified the process. I needed to lock the flare cluster to his chest as he stumbled around, so after generating a background preview from the footage of Robo, a null object was stuck to the center of his chest, traveling with him every step of the way. I simply made keys for the null every three frames or so, fewer when necessary. Once that was complete, I parented seven lens flares to the null (five for the body, two for the helmet) and arranged them in a tight cluster pattern (Figure 3). To conclude, I set up dissolve envelopes for the flares to make them fade in gradually. Aside from a few minor adjustments every few frames, the flares moved with the null and the glow looked as if it was locked to Robo's chest as he struggled. As far as coloring goes, try setting the flare color to R 255 G 154 B 0 and turn on Central Glow and Red Outer Glow in the Lens Flare menu (Random Streaks and Outer Ring should be turned off). The combination of the two colors adds a realistic, fiery look to the flare and works nicely (see the color pages for the results).

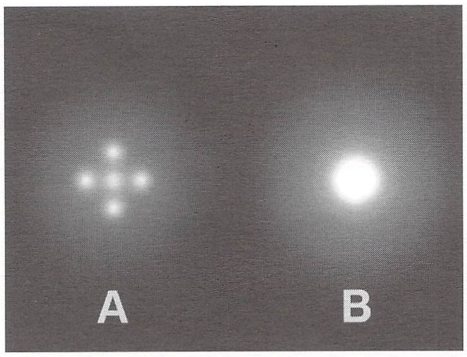


Figure 1: To avoid seeing individual flares (A), pack them tightly so their central glows "bleed" together (B).

Heat Ripples and Hotspots

Before moving on to creating actual flames in LightWave, there are a few more things we need to know about lens flares. The one problem I ran into when using flares was that the screen became too washed out and bright as more flares were added. Figure 2 shows two different approaches. Glow A is one lens flare set at 125 percent glow. Although the central hotspot is the correct size, it's not intense enough, and the haze around the glow completely washes out that half of the screen; it looks more like a

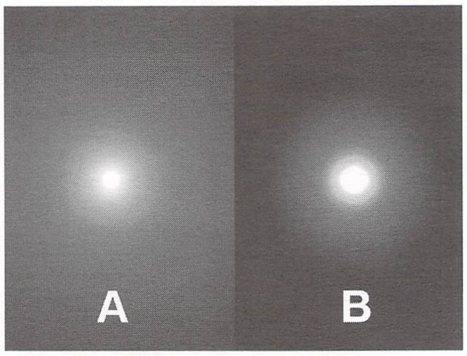


Figure 2: While single, large flares wash out the screen (A), many clustered flares produce a more intense glow without excess haze (B).

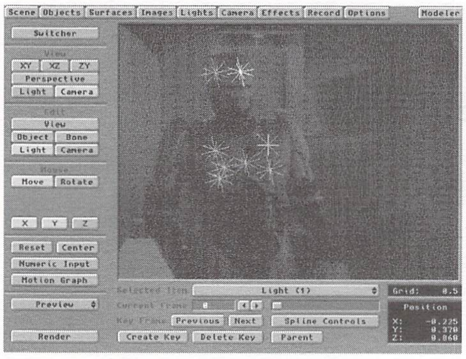


Figure 3: Parenting a lens flare cluster to a null object allows for more control when rotoscoping to background footage.

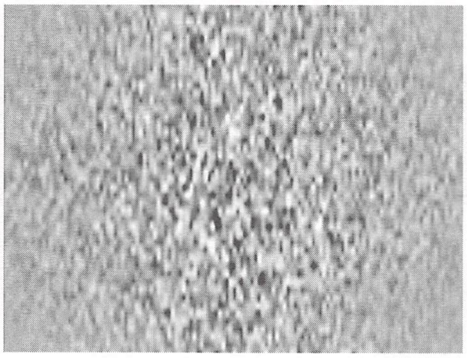


Figure 4: Use the default surface setting to better visualize your travelling bump map before using refraction.

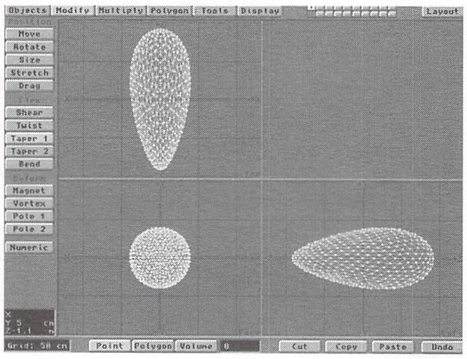


Figure 6: Taper the ball by a factor of 0.5 along the Z axis to get the general shape for the fireball.

light seen through fog. Glow B, however, is comprised of five tightly knit flares with 40 percent glow. This gives us an equally sized but more intense hotspot without the unnecessary haze that accompanies large lens flares. The desired effect is up to you, but hopefully you now have a better idea of how to manipulate lens flares and coax them into doing your bidding.

Although the flare method looked just fine, there still seemed to be something missing from the shot: Robo's armor didn't really look, well, hot enough. I thought nothing more of it and went to see *True Lies* with the rest of the department. We went for a few drinks after the show and talked about a few scenes that caught our fancy, particularly the ones with the Harrier jets. The heat signature they emitted really gave the impression that hot exhaust was blasting out of the jets. It was this element that could give my scene more realism; with this in mind, I awoke the next morning and got right to work adding heat ripples to RoboCop's glowing body.

The first stop was Modeler, where I built a box 2.692 meters wide by two meters high by 0 meters deep (see last month's article "Interactive Refraction" for more details). Name the surface of the box "Screen" by hitting "q" for Change Surface (Polygon menu) and save this object as "screen." Hit "q" again and rename the box surface "Screen.REF." Save this copy of the object as "Screen2" before exiting to layout.

Position the camera at 0,0,0 and make a keyframe for it by hitting return twice. Next, load in the "Screen" object and position it at X 0, Y 0, Z 3.2 before making a keyframe. The next step is simple enough: for my scene I loaded the background footage of Robo as an image sequence and planar-mapped it onto the screen along the Z axis (don't forget to hit automatic sizing). Set the luminosity to 100 percent and the diffuse value to 0 percent. Now comes the interesting bit. Load in the "Screen2" object and position it away from the first object at X 0, Y 0, Z 3.1 (remember to make a keyframe). Enter the Surfaces menu and set the "Screen.ref" surface as follows:

Color: Doesn't Matter
Specularity: 0%.
Transparency 100%
Refractive Index 1.25

Go into the BUMP MAP menu and set the values as follows:

Size X 0.021 Y 0.045 Z 0
Texture Falloff X 50 Y 30 Z 0
Texture Velocity X 0 Y 0.035 Z 0
Amplitude 325% Frequencies 1

Exit into the Camera menu and select Trace Refraction before rendering. Simply put, as the bumps move up the invisible refraction screen, they distort the image sequence on the screen behind just as heat ripples distort things behind them (Figure 4). A texture falloff was added because RoboCop happened to be standing center-frame and I wanted the heat signature to fade off gradually away from him. Because we had a Screamer at our disposal, I wasn't too concerned about rendering time (it turned out to be a couple of minutes per frame), but you may want to try faking refraction as mentioned in Dan Ablan's "Faking Refraction" (September 1994). I received better results using refraction, but not everyone can afford to wait an hour for a render.

Fireballs-a-Plenty

Now comes the part where we actually destroy things. At this point, Robo's armor is glowing more intensely by the second and it looks as if he'll be nothing more than a hunk of burnin' scrap if something isn't done fast. Just when things appear bleak, Robo reverses the polarity of the Heartbreaker beam, blasting a huge fireball across the room that incinerates

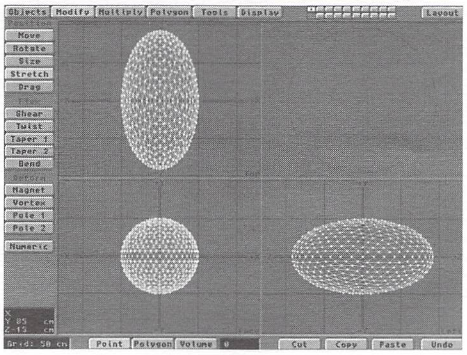


Figure 5: Stretch the ball by a factor of 1.84 along the Z axis

everything in its path, including the bad guy. Some real pyrotechnics were actually set off during filming, but the fireball itself was to be a LightWave effect. This was by far the easiest and most enjoyable part of the scene.

While in Modeler, select the Ball tool in the Objects menu and create a Level 3 Tesselation with the default radius (hit "n" to enter numeric values for tools). Once created, use the Stretch tool (Modify menu) to stretch the ball by a factor of 1.84 along the Z axis (Figure 5), again using "n" to enter numeric values. Next, use the Taper 1 tool (Modify menu) to taper the object by a factor of 0.5 along the Z axis; make sure the sense is set at "-". We've just created the general shape for our fireball (Figure 6). Hit "q" and name the surface "Fireball-OUT." Now select all the polygons in the object by holding down the right mouse button and lassoing the entire object; you can make sure all polygons are selected by hitting "" for Select All Connected. Copy and Paste the object in the same layer and while the polygons are still selected, Size the object (Modify menu) by a factor of 0.95. Keep the polygons highlighted and hit "q" to rename the smaller object "Fireball-IN." Although we now have two fireballs (one inside the other), save the layer as one object called "Fireball" before exiting to Layout.

Although the shape is definitely important, it's the surfacing of an object like this that can determine its success. That said, load the fireball object and enter the surfaces menu. I achieved a nice effect by surfacing as follows:

Fireball-OUT

COLOR 226 225 155
ADDITIVE On
LUMINOSITY 0% with GRID
Texture as follows:
FALLOFF X 0, Y 0, Z 85
VALUE 90%
LINE THICKNESS 2
DIFFUSE 0% with GRID
Texture as follows:
FALLOFF X 0, Y 0, Z 90
VALUE 100%
LINE THICKNESS 2
SPECULARITY 0%
TRANSPARENCY 4% with FRACTAL NOISE

see Great Balls of Fire, page 9

Flying Through Canyons

by Nir Hermoni

For a recent assignment, I needed to produce a spaceship chase through Martian terrain and canyons. The main problem in this type of project lies in positioning the objects at proper positions in relation to the polygon mess, er, mesh. A detailed landscape is made of thousands of polygons, and with no hidden line removal in Layout, it's very easy to place the objects or camera under the ground level by mistake, and not notice it until a render comes by.

Painting a Landscape

So how do you create a detailed polygon mesh terrain and assign convincing paths to the objects zooming through it?

Start by drawing the landscape in a paint program, using only grays. You could use a color image, but LightWave only looks at the luminance (bright/dark) values of the image when using it as anything other than a color map. Plus, a grayscale image will save memory.

The image is going to represent the landscape, as seen from above, so we can use it as a displacement map in LightWave. In the final image, black represents the lowest parts of the terrain and white the highest. I find DeluxePaint's Shade tool a valuable aid in this procedure.

- In DeluxePaint, set up a range of grays between black and white. Fill the screen with the Black color from the range. Using a fairly large brush, draw with the left and right mouse buttons alternately to darken or lighten the areas under your brush. Use finer brushes to add details.

If you don't want to paint, Vista Pro 3.0 allows you to save its data as a color map. In Vista Pro, choose the "Alt->IFF" option under the "ImpExp" menu to save the current landscape as a color map. Load this color map to any paint program or image processing program and change the colors to a range of grays from black to white. Then save the image.

Messing with Meshes

The next step is to enter LightWave and build our terrain object.

- In Layout, enter the **Images** panel and load up your terrain image.
- Enter Modeler and select **Box** from the Objects menu.
- Select **Numeric** and leave all the values at their default except set both Low and High Y values to 0. Select **OK** and then select **Make** (Return) to make a flat 1-meter-square polygon.
- Choose **Triple** (**Polygon** menu) or press T to convert this four-sided polygon to two triangles.

If we built the mesh out of four-sided polygons, they could become degenerated in the displacement process and produce rendering errors.

- Make sure the polygons are facing up by selecting them and pressing **Flip** (**Polygon** menu or f).
- Click on **Array** (**Multiply**) and input 100 for X and Z, leaving Y set to 1. Make sure the Offset is set to **Automatic**.

(You may use smaller numbers if you are short on memory, but you sacrifice from the final mesh detail by doing so.)

After a while, a mesh made of 20,000 polygons will appear.

- Immediately press (m) to merge the overlapping points. Choose Automatic from the requester and select OK. Modeler will inform you that 29,799 points have been eliminated.
- Change the surface name of the mesh to your desired name (Surface or q), and save it by exporting it to Layout with a name you wish.
- If you wish, use the Center macro to center the terrain on the axes.

The next step is to use the terrain image we created earlier as a template in order to create the motion path of our ships.

- Still in Modeler, add the terrain map as a background image using modeler's BG Image (Display menu) function. Choose Automatic Size and Y axis to fit it to the landscape in the Top view.
- Press 2 to go to the second layer. We will now create a curve that will be the flying path of the ship as seen from above.
- Choose Points under the Polygon menu. Looking closely at the image in the top view, follow the canyons (or ridges) while clicking on key locations with the right mouse button to create a logical path.
- Press Ctrl-p to convert the individual points into a curve.
- Move the curve in the Face or Left view so that it is a suitable height above the plane. This will be the average height your ship will cruise above ground level.

Figure 1 shows a curve in layer 2 above a landscape in layer 1 with a sample background image in the background.

- Click Freeze (Tools or Ctrl-d) and then export the newly made polygon to Layout. Exit Modeler.

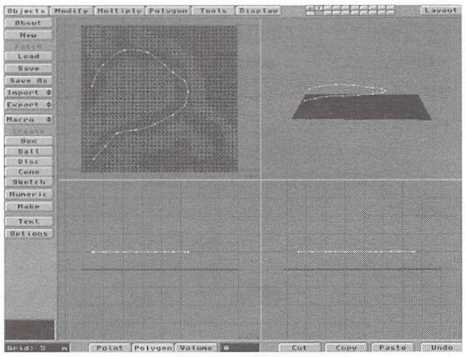


Figure 1: A curve was drawn over a displacement image to get an accurate path.

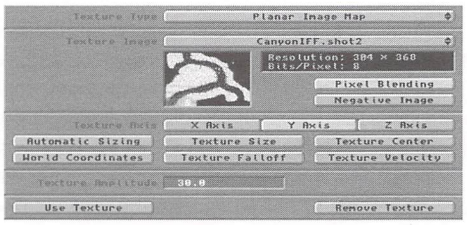
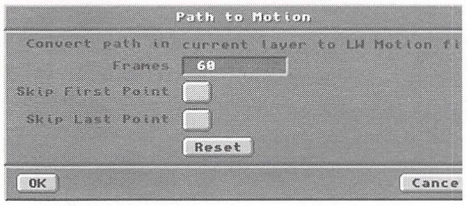

Texture Type: Planar Image Map
Texture Image: CanyonIFF.shot2
Resolution: 384 x 368
Bits/Pixel: 8
Texture Amplitude: 30.0

Figure 2: The Displacement Map panel

Displacing the Landscape

- Choose the terrain as your current object and press (p) to display the Objects panel. Click on the T button next to Displacement Map to enter the displacement map panel (Figure 2).



```
Frames          60
Skip First Point
Skip Last Point
```

Figure 3: Modeler ARexx macro "PathToLWMotion.lwm"

- Choose Planar Image Map for the Texture Type and select the terrain image created earlier in the Texture Image pop-up menu. Click Y axis to map the image on the top of the object, and Automatic Sizing so the texture will fit the plane perfectly. The Texture Amplitude value determines how high the highest parts of the landscape (corresponding to the white parts of the image) will be in meters, so you can set this to what you wish. I used 30.
- Record the values for Texture Size and Texture Center. We will use these numbers to apply identical mapping to the path polygon so it will conform to the landscape.
- Click on Use Texture to keep the settings.

Creating the Path

- Enter the Displacement Map texture panel for the path polygon, but this time do not click on Automatic Sizing. Instead, put the numbers you recorded earlier in the Texture Size and Texture Center and use the same amplitude as you did for the landscape. This will change the height of the points that make the polygon in the same manner it displaced the scape polygons. Select Use Texture to return to the Objects panel.
- Select Save Transformed and save the path polygon under a different name.

Next, we are going to go back into Modeler and create the path that the ships will be flying through.

- Load the transformed polygon and select Remove (Polygon menu or k) to remove any faces and keep only the points that form the polygon.
- Choose the points in the direction you want the ship to travel and then press Ctrl-p to make a curve out of them.

The points that make the curve represent key frames in the final motion. To make the job easier, remove most of the points while keeping the most important ones intact. Be selective—the points that you keep should preserve the shape of the original curve. After making the curve, save it, just to be on the safe side. If you do not remove unwanted points, they will be calculated in the motion file.

- Execute the macro called Path To Motion (PathToLWMotion.lwm). This will transform the curve we have to a LW motion file.

The macro will ask for the number of frames the motion will last in Layout, and if you wish to ignore the first and last points of the curve (Figure 3). This may be handy if the curve has control points at its beginning or ending. The macro will then save a LW motion file based on the shape and direction of the curve.

- Go back to Layout and clear the polygon we used to create the motion path from the scene.
- Load your spaceship of choice. Press (m) to enter its motion graph window, select Load Motion and load the motion we just saved.

Most of the work is done. All that remains is to adjust the ship's rotation to fit the path. **Align to Path** in the Motion Graph may be good enough, but if it's not, change the rotation manually for each key frame. This is the reason we deleted points from the curve—to keep the number of key frames as small as possible. Also, you may add variations to the height the ship is flying above ground level. It is now the same height the original curve was above the plane. Vary it slightly in different key frames to add "natural noise" to the scene. For example, when the ship goes over a hill, it should stay high for a while before coming down back to its average height.

Multiple Cameras

In the suggested scene, we have spaceships zooming through narrow canyons, chasing one another. But how do we know that the chase is accurate, and the ships are actually seeing the enemy ship they are chasing after?

The solution: To view the scene from an object's point of view, use a reference light as your "eye." Here's how:

- Add a light. Make its intensity 0, so it doesn't affect anything in the scene. Parent the light to the object in question and then choose Light View. Next, move and/or rotate it so it represents the object view.

This could be the cockpit of an airplane, near the head of a horse, or who knows where for a floppy disk. Don't forget to set a keyframe at frame 0 for the light.

The light will follow the parent (the object) everywhere, and you can even change this view's "zoom factor" by making the light a spotlight and varying its **Spotlight Cone Angle** parameter.

These reference lights make great "virtual cameras" to better understand the motion in a scene. When you're finished creating the scene, you may remove all the "view" lights, or choose to leave them. If the light's intensity is 0 and it has no lens flare, it won't affect the final image or rendering time in any way.

One final tip: LightWave knows there is a Help key on the Amiga keyboard, and supports it extensively. Pressing it presents a list of all the keyboard shortcuts that are available for the current panel. I suggest trying this on every single panel you come across. Many hidden gems can be found by doing so.

LWP

Nir A. Hermoni has been doing Amiga graphics since 1989. He now works for ZAPA Digital Arts LTD in Israel, creating LightWave visuals for computer games and television. Nir can be reached on the Internet as zapa@zeus.datasrv.co.il.

Getting Started

The following ARexx script is useful when beginning new projects. I like to keep all my project-related files in one location in order to minimize the mess when dealing with a large number of files, and to help in case I'd like to move projects from one computer to another. I added the line "Assign LW: work:toaster/3d/scenes" to my s:user-startup file. Work:toaster/3d/scenes is, of course, the place I keep my projects in.

The ARexx script creates a "parent" directory with the name you specify.

Inside it, "Objects" (for objects) and "Images" (for brush maps) directories are created, each with its own "Archive" subdirectory for test or unused objects and maps. A "Frames" directory is created for the animation frames LightWave will generate. "Temps" is where I store motion graphs, envelopes and surfaces that are relevant to the scene, and "Scenes-archives" is for all the scene files I create while working on the project. Just save the script as "mdlw.rexx" (Make Dir LightWave) in the Rexx: drawer, and type "rx mdlw <<dir name>>" at a shell prompt to get the job done.

```
DIRNAME = arg(1)

if DIRNAME = '' then do
   say ' '
   say 'Create directories for LightWave, by Nir
   Hermoni 1994 (C)'
   say 'LW: should be assigned to your
   :toaster/3d/scenes directory.'
   say 'Usage: rx mdlw <directory name>'
   say ' '
   exit
   end

address command
   'makedir lw:'DIRNAME
   'cd lw:'DIRNAME
   'makedir lw:'DIRNAME'/Images'
   'makedir lw:'DIRNAME'/Images/Archive'
   'makedir lw:'DIRNAME'/Objects'
   'makedir lw:'DIRNAME'/Objects/Archive'
   'makedir lw:'DIRNAME'/Frames'
   'makedir lw:'DIRNAME'/Temps'
   'makedir lw:'DIRNAME'/Scenes-archives'
   say 'Directory structure prepared, have a
   ray-traced day'
   say ' '
   exit
```

Fur and Hair
The LightWave Solution

by Gonzalo Garramuno

Although 3D computer graphics can reproduce reality with great accuracy, there are still some things that present a great challenge to the animator. Animals and human beings, for example, are one of the most difficult things to re-create on the computer because they require a great amount of detail. Since viewers are familiar with both people and animals, they expect to see wrinkles and individual hairs when close enough. For years, computer artists struggled to create realistic fur and hair through bump maps and textures, with little success. But, if you have seen some of the new television commercials, like the ones featuring a group of polar bears, you might have noticed that this problem seems to have been solved.

However, you might not be aware that LightWave can represent hair and fur in a way very similar way to that seen in those commercials. As a matter of fact, it has had that option since version 1.0, although, as you will see, it could not easily be implemented until version 3.0.

Enter Particles

Since its conception, LightWave has been able to draw particles (one-point polygons) and particle lines (two-point polygons). The latter are the ones we will be talking about here. If you think about it, fur and hair could easily be represented by a group of lines (hundreds or thousands) coming out from the surface (skin) of your object. But how can you put a line that conforms to the shape of your skin? By using booleans. However, the process needs to be repeated for hundreds or even thousands of lines, which makes the task impossible to do manually.

Since version 3.0, LightWave's Modeler has added the option of macros, which are ARexx programs that can tell Modeler to perform the same actions that you could do by clicking, only much faster and with any number of repetitions. Therefore, in order to have Modeler create fur and hair, what we need to do is write a macro. We do not need to start from scratch, as we may find some other macro that performs some similar action to what we are looking for. Go into Modeler, draw a ball in the center of the screen and choose the Random Surface Points macro. (This is accomplished by going into the Objects menu, clicking on the Macro button and keeping the left mouse button pressed while going up or down until you find it.) After you execute it, just press OK and see what happens. The macro performs a boolean function on your object and the result is a group of points that are laying exactly on the surface of your object. When I first tried the macro, I thought it was pretty useless and that it should be modified to do something more useful. I quickly realized that it could be changed pretty easily to allow the creation of hair. What we need to do is copy and size those resulting points, and then make a polygon with the new points and the old ones to get the lines that are coming out from the surface of the object. The macro included here does exactly that, and it is basically a modified version of the Random Surface Macro (whose file is named prick.lwm) included with LightWave.

Typing the Macro

In order to type in the macro, you need to be able to use an ASCII text editor. The Amiga computer has two text editors in its operative system (ED and Memacs). Refer to your manuals for information on them and the operative system. You can also use a commercial text editor like Cygnus ED or The Edge! or any word processor that allows you to save the documents as plain ASCII text.

You might want to load the Random Surface Points macro first and add the modifications instead of typing everything from start. If so, look for the file named "prick.lwm" in the "Arexx_Examples/lwm" of your Toaster directory.

After typing everything, save the file under a new name (like "Hair.lwm") in that same directory. Then go into Modeler, and use the "Add Macro" to add this new macro to your list. After that, you are ready to use it. If you get an error message while running it, return to your text editor and check the line where the error appeared. You probably mistyped something there or before.

Using the Macro

The usage of the macro is fairly simple.

Copy all the polygons that you want to have hair on top of into an empty layer. Then run the macro. The macro will pop up a requester with a number of options:

- Maximum number of points refers to the maximum number of hairs that you might end up with. To create a realistic effect, you should try to set this value pretty high (2000-15,000 is a good range) but keep your memory capacity in mind.
- Create Lines allows you to create the hairs. If not selected, the macro will act just like the old Random Surface Points macro.
- Number of Segments refers to the number of segments per hair. If more than one, it will allow your hairs to bend more smoothly if you use any tool on them.
- Line Size is the length of your hair. It works by using a scaling percentage. Values between 1.05 and 1.3 are usually OK. Values smaller than 1 will make your hairs go into your object. If using more than one segment, the sizing value will be performed for every segment each time.
- Center works in conjunction with Line Size and allows you to set the center where the sizing is going to occur. By default, it will find the center of your object by averaging the position of its boundary points. Therefore, the default values are usually OK.
- Line Angle allows the hairs to bend in one direction. Just set the maximum angle of bending.
- Rotation Center works in conjunction with Line Angle and allows you to set the center where the rotation is going to be performed. By default, it will find the center of your object by averaging the position of its boundary points. Therefore, the default values are usually OK.
- Rotation Axis works in conjunction with Line Angle and allows you to set the axis on which the bending is going to occur.
- Jitter Amount and Jitter Type allow you to add some random variation to the hairs. The amounts are unit numbers that you can select. The defaults values are 1/10 of your current object's size, which is usually OK.
- Put Results on Original Layer copies the hair on top of the polygons. Otherwise, they will remain in another layer.

If the results of the macro are not to your liking, simply click on Undo and try again with other values. If you get bad results with several sets of very different values, you may have mistyped a line in the macro.

Just go over the listing again and check the lines very carefully. Changing the names of variables or operations, or even placing periods where they shouldn't be, can render a macro useless.

After the macro finishes, you will probably need to do some work yourself, like erase some hairs that might intersect with an ear, or use the different Modeler tools to "comb" your hair.

When you go into Layout, if you have LightWave 3.5 you might want to try different settings for the Particle/Line Size. To do that, go in the Objects menu, select the hair object (or the one that has the hair on it) and change its value.

I should mention that it seems that LightWave seems to use a different algorithm (or a bug?) for shading particles, which makes the hair stand out a little, even if it shares the same surface as the "head."

Using shadow maps or raytraced shadows can help to avoid this problem a little.

Included on this month's disk is the complete, typed macro. Simply copy it to the appropriate directory and run the "Add Macro" macro to add it to your list.

LWP

Gonzalo Garramuno is a 21-year-old animator trying to make a living in Argentina. Since the salaries for animators are extremely low in his country, he has been forced to work as an editor, video operator and CG operator, and as a teacher to younger animators. He can be reached by e-mail at ggarramuno@houseware.satlink.net or by surface mail at Rosario 414, 3rd Floor, Buenos Aires, Argentina.

Editor's Message

continued from page 3

who is an avid LightWave/Toaster fan and knows his stuff.

Star Trek: Voyager

By the time this issue sees print, you may have seen the pilot and/or first few episodes of the new Star Trek series, *Star Trek: Voyager*. One of the interesting things about *STV* that you may or may not know is that there are a good number of LightWave effects in the pilot and upcoming episodes. As a matter of fact, three of the six *Voyager* shots in the opening sequence for the series use the LightWave-modeled *Voyager*. See if you can figure out which ones are the real model and which ones are the LightWave model. Some of the other LightWave-generated effects for the pilot include stars, planets, badlands and a galactic wave.

Let me tell you, there has been a lot of blood, sweat and late, late nights put into the CGI version of the Voyager and related effects. At press time, we were just at the tail end of delivering effects for the pilot, and I am very interested in seeing how everything comes out (I am also very tired!). Hopefully, we will be doing a large number of effects shots for future episodes of the series.

Multi-Platform LightWave

Work is progressing nicely on the porting of LightWave (as seen at the Expo) and NewTek says that the multi-platform version of LightWave 4.0 will be shipping sometime in the first quarter of '95. There had been talk about shipping before the end of 1994, but obviously that didn't happen.

Here's the problem with announcing shipping dates: Prospective buyers demand to know when their product is going to ship, but it is extremely difficult to predict when a product is going to be completed. So difficult, in fact, that companies have been off by years! This is not a dilemma unique to NewTek. Every hardware/software company known to humankind has found itself in this situation at one time or another.

I think NewTek should limit itself to talking in quarters regarding the shipdates of every product. It gives the user an approximate date while not tying NewTek down to a specific period. (It also gives manual writers time to finish their work!)

LWPRO

As *LWPRO* gains in popularity, there has been some talk lately around the Avid offices of turning it into a full-fledged magazine. What do you think? We'd love to hear your comments on the subject. Contact us on-line or by fax to let us know what you think.

New Year's Resolution

Finally, my New Year's resolution for '95 would have to be 2048x1536.

John Gross
Editor

Great Balls of Fire

continued from page 5

Texture as follows:

SIZE X 0.07, Y 0.08, Z 0.6
VELOCITY X 0, Y 0, Z -0.3
VALUE 100%
FREQUENCIES 6
CONTRAST 2.0
EDGES Transparent
EDGE THRESHOLD 1.0
SMOOTHING On
DOUBLE SIDED On

Fireball-IN

Everything is identical to Fireball-OUT except the following:

COLOR 251 45 0

The GRID textures for LUMINOSITY and DIFFUSE have a value of 85% and 100%, respectively, and both have a TEXTURE FALLOFF of 62% on the Z axis.

The reason for the Grid textures was to make the Fireball gradually fade out toward the tail so the shape of the object isn't evident (the transparent edges helped as well). Though the Fireball looked pretty good on its own, I parented a small cluster of lens flares at the tip to give it a furniture-destroying glow (see the color pages). If all goes well, your fireball should look like a fireball, complete with traveling flames. Using this as a template of sorts, you can modify the shape or surfacing of the object to come up with some interesting effects of your own.

I'm sure that with some practice and plenty of experimenting, the fire department will come a knockin' in no time. As for me, Robo blew the bad guy to hell and I'm still trying to regrow my eyebrows.

Cluster lens flares together for a more uniform, intense glow. Use many small flares packed together instead of one large flare to avoid excess glow washing out the scene.

Create two 2.692m x 2m screens in Modeler, Screen1 and Screen2. Position Screen1 to fill the entire camera view and position Screen2 slightly closer to the camera. Map image sequence onto Screen1 and make Screen2 refractive with a bump map traveling up it.

Create a Level 3 tesselation and stretch and taper one end. Name the surface Fireball-OUT and select all polygons. Then copy, paste and shrink the object slightly. Finally, name the smaller object Fireball-IN and save the two as one object. Experiment and HAVE FUN!

LWP

Colin Cunningham is enrolled in the infamous classical animation program at Sheridan College in Canada. He is currently working on a 3D cartoon short that he calls "a cross between Evil Dead *and* Itchy & Scratchy. *Cunningham can be reached at (905) 338-8033 at any time, because Sheridan students don't sleep.*

LightWood in LightWave

by Dan Ablan

If you read *LWPRO* on a regular basis, it's safe to assume that you are well-versed on the capabilities of this great 3D program. It is probably also safe to say that you know how to import surfaces and textures into your animations from all types of sources. Imagine, though, that you are new at all of this, and only wanted to use the system and software you paid for. Why should you have to purchase extra textures if you just spent so much money on your system? On the other hand, what if you aren't new at all, but still have to meet a deadline, and there is no time to generate realistic textures? Well, here is a simple tutorial that works well for both new users and the experienced.

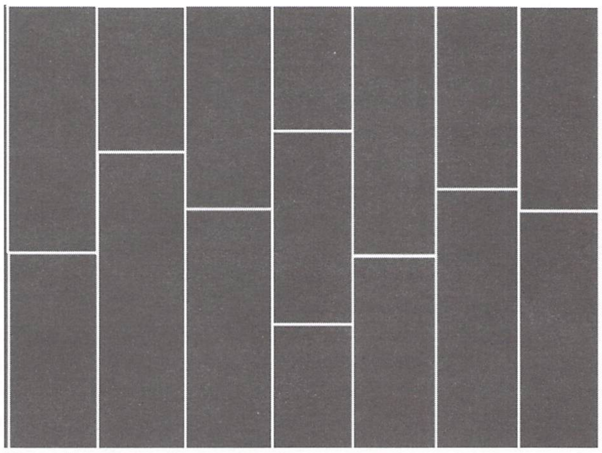


Figure 1

Two Steps

The credit for this idea should go to my good friend Arnie Boedecker, who created a fantastic living room floor with this technique. I took it one step further, using the final image as a background for a logo job. Perhaps you can use it for something else, maybe paneling a wall or a log cabin. All you need is ToasterPaint, LightWave, and a little time.

We're going to take that familiar lightwood surface and use it to make a random, wood-slat floor (Figure 1). Start by loading the **Get Small** project. If you're not running with a lot of RAM, close CG and LightWave if they're open. Enter TPaint. The first task is creating a black and white image that has two purposes. The first is to help line up our wood floor slats, and the second is for a bump map. Select an unfilled rectangle as your drawing tool, with the smallest circular or square brush. Draw out a long vertical rectangle, so you have a white outline on black. Continue drawing out vertical rectangles, but make their vertical positions vary. You can use the **Set Grid** feature of TPaint to help keep even width. Figure 1 shows what the final image bump map image should look like.

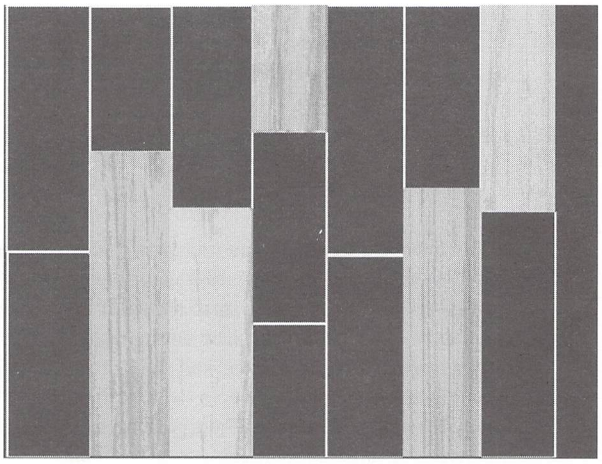


Figure 2

Remember, this will serve as an outline for your floor and where the wood slats will be, so play around with the randomness of the lines. Once you're happy with what you've done, **save** it as an RGB image, as LightWood_Bump. Now, with this image still loaded, hit the **J** key to jump to TPaint's other screen. Load the **lightwood** image from the **Wood** directory. This is the part that gets somewhat tedious, but it's worth the effort. Begin to cut different parts of the lightwood image and TXMap it down.

When you cut out a section of the lightwood image, or the whole screen, make sure you **Copy This Brush**, under the **Brushes** menu. This will hold what you've cut in a temporary buffer. Use the **J** key to jump back to your other screen with the black and white bump map image. Under **Options**, using the right mouse button, select **TXMap**. Choose a filled rectangle as your drawing tool and the smallest square as your brush (not the pixel brush). Go to any one of the black and white rectangles and TXMap your image into just one of the areas (Figure 2). Cut different areas of the lightwood image and shrink, flip, darken and lighten to create random pieces of wood. Don't be afraid to cut an area, stretch it out, and then cut it again. The more ways you manipulate the lightwood image, the more random it will be. **Save** this as LightWood_Tall. Hit the **J** key again to jump back to the original lightwood image. Cut out another area, copy the brush, and TXMap it into place. Darken or Lighten this one a bit. Continue this process, filling up each space on your bump map, until you have what looks like random pieces of wood, like a floor. Remember to be creative and flip your brush, stretch it, or crunch it to vary the texture. In addition to darkening and lightening, add a little color of white, or brown to a few of the areas for more randomness. Figure 3 shows the final wood image in TPaint. Each seam should match the black and white image you created earlier. Again, save the image.

Into LightWave

Once you've made your bump map and wood floor, close TPaint and enter LightWave. Here's a tip: You can quit and close TPaint all in one move. Hit the left shift and the tilde key all at once (Tilde is the key just left of the number 1 key). The first thing you need to do is make a one-sided polygon in the shape of a rectangle in Modeler.

Make sure your surface is facing the camera, name that surface something like WoodBKD, and

Export this as a new object, saving it as LightWoodBKD_obj. Go to your images panel and load the two Toaster Paint images you just created, LightWood_Tall and LightWood_Bump. To help save memory in LightWave, you may want to change the LightWood_Bump image to two colors.

Under the surface panel, select the WoodFloor surface, and click on the **T** button next to **texture color**. Use a planer image map, on the Z axis. Click automatic sizing and turn off antialiasing. Antialiasing is turned off in this panel, because we'll turn it on under the camera panel for rendering. If both antialiasings are turned on, the images can become blurry. Click on Use Texture. Set your **specularity** level at about 70 percent. **Glossiness** is High. The next thing to do is create the separation between the slats of wood. Since the slats of wood were placed down using our bump map as a template in paint, the seams will match up evenly with the bump map in LightWave.

Click the **T** button next to **Bump Map**. Also set planer image map, Z axis and automatic sizing. Set the amplitude to -55 percent. Click **use texture**. Move your camera view so the flat polygon

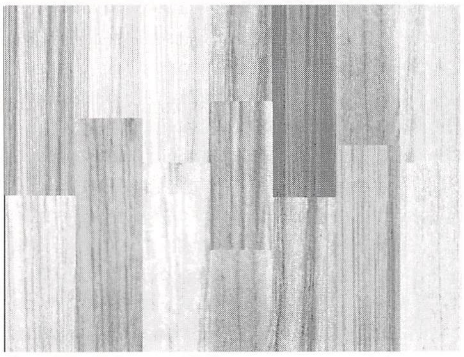


Figure 3

(LightWoodBKD_obj) we just created is filling up the whole screen.

Now render the image to see how it looks. You may want to vary the amplitude for the bump map and adjust the specularity a bit. If you like what you see, **save all objects** and save the scene. Go a step further now by changing the light source. Make the distant light a spot and change the color to a soft amber or pale yellow. Doing this will warm up the image. Remember, LightWave defaults its light color at white. Look around—the light around you is never pure white. Angle down from the top left, so that not all of the area is hit by the light (Figure 4). Turn on

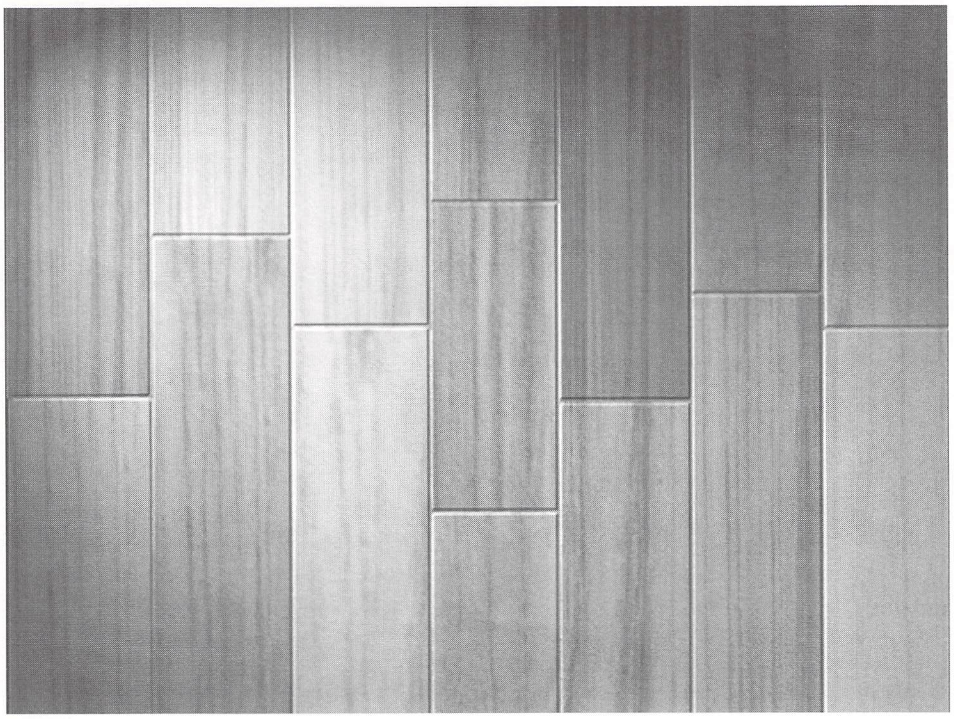


Figure 4

Trace Shadows and render in Medium Resolution with low antialiasing.

It will take a little time to render, but you can then save the image and load it back into LightWave, using it as a **background image.**

This is a fairly simple technique, but very effective. You can use it as a background image, like I do, or repeat it for a flooring like my friend Arnie does. If you were going to use it as a floor, consider making it glossy, and set a reflection level of about 15%. Turn on **Trace Reflection** and you'll see some nice results. You might want to make the images (LightWood_Tall and LightWood_Bump) seamless, so you can repeat them over a large area.

By taking existing images already in your system and spending a small amount of time manipulating them, you can create truly realistic textures in LightWave. For example, although the ever-familiar Verde-Pompeii Marble image is nice, it's overused. I've seen it used time and time again in animations, CG pages, etc. But you can still use it... just take it one step further. If you were making a floor for a living room scene, and had absolutely no time to get any textures, and only had what your system came with, you could use the Verde marble image. Color that floor grey. Then, diffuse the Verde Pompeii image, as a planer image map — there you go, grey marble. You may even notice it looks a bit like carpet. Bring it into Art Department Professional and reduce the colors. Back in LightWave, use it as a bump map with diffusion, for some random bumps. Try coloring that LightWood image white, yellow, green or whatever with the same diffusion method. Or, use the bump map example on the Verde Pompeii image for a marble wall look.

Sometimes not having all the great texture packages and imaging programs is a good thing, because you're forced to use your imagination to make the most out of what you have. Possessing that ability is a great asset in any situation.

LWP

Dan Ablan is animator/owner of AGA in Chicago and animates with LightWave. Recent projects include work for Kraft and The Dial Corporation and trailers for Star Trek: Deep Space Nine. *Dan is also co-founder of the Chicago LightWave Association.*

He can be reached at (312) 239-7957 or by e-mail at dma@mcs.com.

In the February Issue:

Next month's issue of *LIGHTWAVEPRO* features part two of December's "LightWave 101: Methods of Modeling," plus a tutorial on proper land vehicle animation, which includes how to have tires spin at the correct speed.

Rendering Algorithms
Part 1: The Theory of Z-buffers

by William Frawley

While many types of rendering algorithms exist, those explained in this two-part series are some of the more popular ones. Each of the different methods described has advantages and disadvantages, which I assume might explain why LightWave programmers Allen Hastings and Stuart Ferguson incorporated aspects of both Z-buffer and ray tracing techniques. And one is still hard-pressed to find much, if any, commercially available 3D software incorporating radiosity techniques.

This study of rendering algorithms won't necessarily be dealing with the specifics of LightWave's implementation, but some of the rendering techniques incorporated by this and most other 3D software may prove of interest to those who are curious about what's happening after they hit Render. I know I was—hence this study. But before we begin to look at the mechanics of Z-buffers this issue, and ray tracing and radiosity next month, a brief exposition on the geometries of 3D space and frame buffers is definitely in order.

The 3D Universe

Common to most image synthesis software, objects exist relative to an origin in three-dimensional object space represented by some coordinate system, usually Cartesian (XYZ). Much like a camera taking a photograph, the rendering of these 3D objects projects them toward the eye or **camera view** onto the two-dimensional **screen space**, where the total area covered by the view is dependent on the **view angle** (see color image 3D Object Space). In order to reduce the chaos, not to mention time, of rendering an infinite amount of space, **object space** is bound on its sides by **boundary walls** and near and far planes called **hither and yon planes**, respectively. These **clipping planes**, as they are called, constitute the boundaries for the **viewing volume**. Any object inside this volume potentially makes it into the final image, and any object or part of an object outside this volume gets "clipped," hence the term clipping planes. Usually, the **viewscreen** (where the final image is drawn) serves as the hither clippingplane as well.

Finally, for rendering purposes it is common to have the Z-axis oriented perpendicular to the viewscreen and pointing away from the camera into the object space—similar to LightWave's implementation. As we shall soon see, the significance of this orientation method becomes apparent when dealing with Z-buffer rendering algorithms. However, understanding the nature of Z-buffers requires a brief explanation of frame buffers.

Frame Buffers

Simply put, a **frame buffer** is a specialized piece of memory storage hardware whose memory locations are arranged in a gridlike pattern. A frame buffer's main objective is to "buffer" one "frame" of video, each memory location roughly corresponding to one pixel on screen. Therefore, for a full-color frame buffer, each memory slot would contain one or more numbers representing a certain color value for that pixel. Alternatively, the number could be an index or pointer to a reduced bitplane **color map** or look-up table. (Unfortunately, space does not permit an in-depth discussion of color maps.) Suffice it to say that a typical frame buffer represents a full-screen image. However, this memory array concept may also be creatively used to store numbers representing something completely different.

Z-buffer Rendering

The basic concept of Z-buffer rendering is quite simple. Two buffers are used for this technique. One is a frame buffer that will eventually contain the final rendered image, while the other buffer used contains the **z-depths** or distance of the nearest visible surface along the Z-axis as seen from the corresponding pixel of the viewscreen. This is why the axis orientation is transformed with the Z-axis pointing away from the screen. It would be helpful for understanding the entire process if we break down the algorithm itself into a series of simple steps.

First, once the three-dimensional model is constructed, including the final positioning of the camera and lights, the computer starts with the first screen pixel and decides if anything should be drawn there. It selects an object and mathematically determines if that object is visible from the current pixel center. If it is visible, the next step is to ascertain whether it is the closest object thus far encountered at those XY coordinates. It does this by checking the Z-buffer, which, in addition to the image buffer, had been previously cleared to some default value prior to rendering. For the image buffer, that value would have been the background color, and for the Z-buffer, the initial value in each memory location would have been set to the z-distance of the yon clipping plane, the farthest possible point in object space. If the current visible object's surface distance is closer to the eye than the current value in the Z-buffer for that point, that object's surface distance or z-value replaces the one currently in the Z-buffer, signifying that this new object is the closest one visible in the camera view thus far. Finally, the color shading of the closest visible point on the surface of this object at this point on the screen is determined from the object's surface properties and light source characteristics and entered into the image buffer. This completes the process for one pixel or point making up the screen. The next pixel in line repeats this entire sequence of steps for the current object. Once the entire screen is completed for this

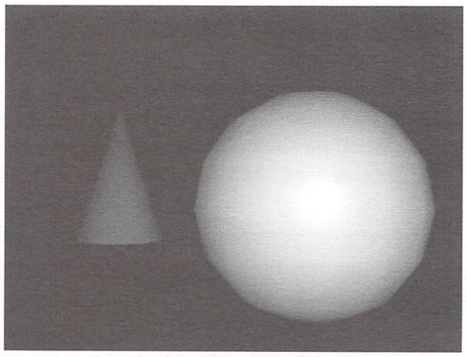


Figure 1: If translated to a luminance map, a typical Z-buffer might look like this. The lighter values represent z-values closest to the camera view. See color image "Z-buffer " for the scene layout from another view.

object, the next object in the scene is considered, and, as before, checked pixel by pixel via the Z-buffer for visibility. Wow, try doing this by hand!

If you watch LightWave's rendering screen, you'll notice the image generally being built up one object at a time. The accuracy of the image is constantly changing as closer objects eventually replace more distant objects in the frame buffer. Hence, the erasability of the buffers proves invaluable to this algorithm's method. In other words, it is inconsequential as to what order surfaces (or points of a surface) are chosen to be processed. In the end, after all surfaces have been processed, it is the closest object seen at each pixel that will remain in the Z-buffer, ready to be shaded for the final image (see color image Z-buffer).

Interestingly, although the Z-buffer is not intended to represent any kind of an image—only z-values—because of its implicit nature, if translated to a color map it will probably exhibit a certain likeness to the final image nonetheless (Figure 1).

Z-buffer Shadows

At the expense of time and memory, the Z-buffer algorithm technique can produce rudimentary shadows. However, the process does not take into account transparent/translucent objects and requires extra memory storage for each light's shadow buffer in addition to the normal image and Z-buffers. Another requirement relates to the type of light source that can be used to produce shadows. As you've probably seen in LightWave's Lights menu, only directional Spotlights work with shadow mapping, because each light must render its own Z-buffer, called the **shadow** or **illumination buffer**, for the scene. Considering the nature of the geometry involved, Directional and Point lights will not work for this method.

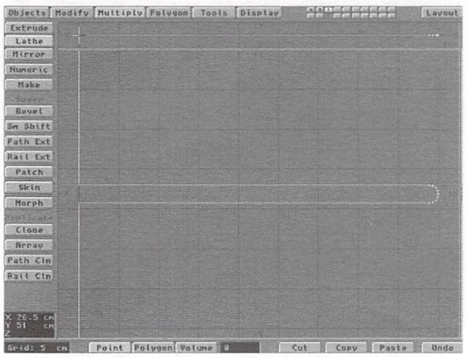


Figure 2: For Z-buffer shadows, all surfaces and objects not directly seen by the light source, such as the sphere and back surfaces of the box and cone, would be excluded from the light source's shadow buffer. Only the z-distances of the nearest visible surfaces would be included.

Shadow buffers enter into the proverbial equation at the shading stage for each visible object's surface point. Before the normal image is rendered, the scene (or frame for animations) is rendered from the point of view of each light, creating its own Z-buffer of any visible objects. Therefore, if an object's surface makes it into the light's shadow buffer, it can be considered to be illuminated by that light source. If not, that object must not be illuminated by the light, or is obscured by some other object, and thus in shadow (Figure 2). Once the shadow buffers for each light source are rendered and saved in memory, normal rendering begins for the camera view image.

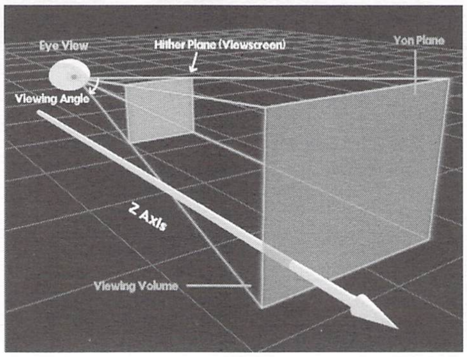


Figure 3: An overview of 3D object space. All objects within the viewing volume would be potentially visible from the viewpoint of the camera; all others outside would be clipped.

When a visible point on a surface is encountered, the shading procedure begins. From the view of each light source, the pixel containing the surface point in question is computed. Then the z-distance from that pixel of the light's view to the object's surface point is calculated. If that z-value equals the value in that light's shadow buffer, the surface point must be receiving illumination from that light source, and is added into the shading of that surface point in the final image buffer. If the distance is greater than the value in the shadow buffer, it must be farther than the nearest object, and is thus in shadow. Therefore, no shading for this light is added into the final image. The algorithm then proceeds to the next light source (if any) for computation of any additional shading information for that point in the image, repeating the entire procedure as before. As you may have guessed, the extra time and memory required to store shadow buffers are directly proportional to the number of shadow-casting lights in the scene.

Z-buffer Pros and Cons

When all you require of your images is simplicity, Z-buffers can be quite fast. And if available memory is a concern, pure Z-buffer algorithms allow a nearly un- limited number of objects within your scene, because each pass through the Z-buffer calculation requires only one object in memory at a time. This is contrary to other algorithms such as ray tracing, which requires all objects within the scene to be held in memory simultaneously. Another advantage owing to the singular object algorithm includes the ability to add independent plug-in modules or subroutines to handle the rendering of different types of modeled objects, such as polygons, curved-surfaces or fractal objects. Once the information from one object is written to the picture and Z-buffer, each new pass can be handled by a new program catering to a new kind of primitive. Theoretically, a whole library of plug-ins could be used for multiple object types.

On the other hand, Z-buffers have problems or are extremely inefficient in dealing with transparency and reflections. With the transparency subroutines, the sorting loops needed become too slow to be efficient. Similarly, pure reflections become impossible, except with time-consuming subroutines using mirrored textures. Other possible problems with Z-buffers include aliasing, motion-blur and good shadows, but, as we witnessed, these can be overcome with some crafty programming workarounds.

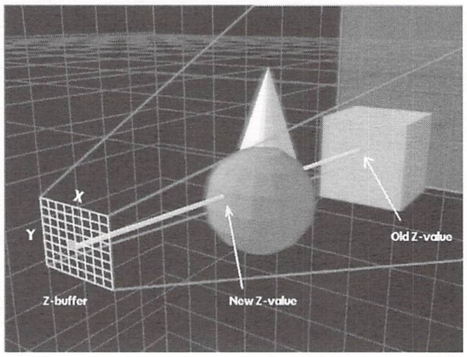


Figure 4: Each time through the loop, a new object's distance from the screen is checked against the value in the Z-buffer. If it is closer than the one currently there, it replaces the old value. This is the heart of the z-buffer algorithm.

What's Next?

Next month we'll take a look at an algorithm familiar to almost everyone acquainted with 3D synthesis: ray tracing. Though this technique entails more realism than Z-buffers, the improvement comes at the expense of increased rendering time.

We'll also examine a method of preprocessing the shading calculations that, when used in concert with a good rendering algorithm, can produce some of the most superb images in 3D rendering today. Because of the intensive number-crunching involved with radiosity, most 3D animators have to be content with simulating this supra-realistic effect. Sigh. Oh well—tune in next month and we'll grapple with it anyway.

LWP

William Frawley is president of Ecliptic Arts, a developing 3D animation and special effects production house. To ensure he has no free time whatsoever, he is also currently the author of a monthly graphics column for an Amiga-related publication called Amazing Computing, *a part-time sales jockey for a local wine and beer establishment, and an enthusiastic philosopher and mystic with a hankering for good pizza. Send questions or comments to Ecliptic Arts, c/o William Frawley, 315 W. Fifth Street, Muscatine, IA 52761.*

Riding the Rails

A Rail Extrude Tutorial

by Arnie Boedecker

I have found that the best way to begin to use and understand Modeler is to master a few functions at a time. Some functions will be used on nearly every project, while others will rarely be touched. (Anyone out there a big Quantize user?) Obviously, the tools used largely depend on the project at hand.

A number of months ago, I stumbled across a project in which I needed to construct a glass table with a detailed metal bottom (see color pages). I knew that in order to build such an object, I would have to dive into some previously unused tools. What I found was Rail Extrude.

Rail Extrude, in my opinion, is a tool that hasn't received the coverage it deserves. I've found it to be extremely useful in creating wrought iron gates and similar detailed objects.

Operation: Glass Table

To construct the glass table object, only a handful of tools are needed, including Lathe, Mirror, Clone and, of course, Rail Extrude.

The first step is to create the glass tabletop. For mine, I wanted the circular glass slab to have slightly curved sides, for a more elegant, less boxy feel.

- In the Face view, plot a few points to make up a curve (or better yet, use the side of a disk), and move them out some distance on the X-axis.

I moved mine about 45 cm, which provided an accurate measurement from the center of the table out to its edge.

- Create two points on the Y-axis that line up with the top and bottom points of the curve.
- Select the points in a clockwise fashion and create a polygon (p) to lathe into the tabletop (Figure 1).
- Using the Lathe tool (Multiply), lathe the polygon around the Y-axis, using Numeric to set the number of sides to 100 to ensure a very smooth table edge.
- Select all of the top flat polygons and use the polygon Merge function (Polygon menu or Z), so that the top consists of one polygon rather than 100. Then do the same for the bottom polygons.
- Select the top and bottom polygons, and use the **Surface** button (**Polygon** menu or q) to label their surface as Glasstop.T&B. Label the side polygons Glasstop.Sides.
- Move the entire object up so it is not sitting on the 0 Y-axis. I moved mine approximately 35 cm.
- Save the object as TableTop.

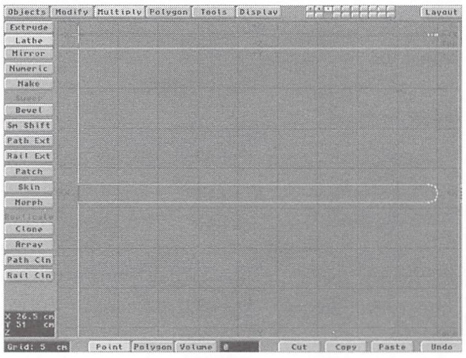


Figure 1

I used different surface names so that I would be able to give the sides a smooth surface while keeping the top and bottom flat. Had I tried to smooth the GlassTop without assigning different surfaces, the sides would have smoothed into the top and bottom polygons, resulting in a very unnatural look.

The next step is to create the detailed table bottom. This starts off with the top support upon which the table top will rest.

- In the Face view, make a flat 2.4 cm disc with 16 sides (Objects/Disc/Numeric) and move it out a shorter distance than the edge of the table on the X axis (I used 32.5 cm). Next, lathe the disc around the Y-axis with 100 sides to complete the support.

Next, it's time to create the curved metal legs.

- In a new layer, with the support ring just created in a background layer, draw out a curve using the Sketch tool (Objects) in the side view. My curve had a slight S-shape to it, as needed for the table's design. The curve should be drawn from the edge of the support ring to the "ground" (0 on Y). Hit return to make the curve.

In the Face view, I shifted the top few points to the left to give the curve a desired path of the table leg. Also, I had to be certain that the top point of the table leg curve intersected the top support in the background layer. Once created, there would not be a gap in between the legs and the top support. This curve will serve as our "rail" in our Rail Extrude operation (Figure 2).

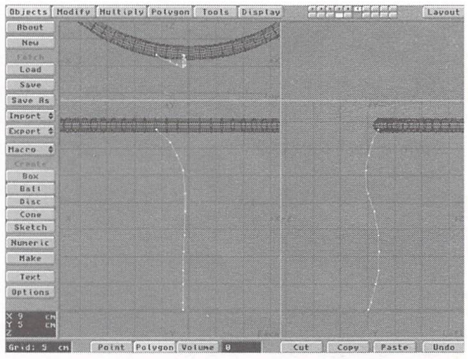


Figure 2

Riding the Rail

- In a new layer, with the "rail" in a background layer, create a 1-cm disc to use for the table leg. After moving the disc to the top of the curve, rotate it in all three views until it is perpendicular to the start of the curve (Figure 3).
- Select **Rail Ext** (**Multiply** menu) and choose **Uniform Knots** set at 20, and **Oriented**.

When attempting your rail extrusion, if it seems to be heading in every direction and completely out of the

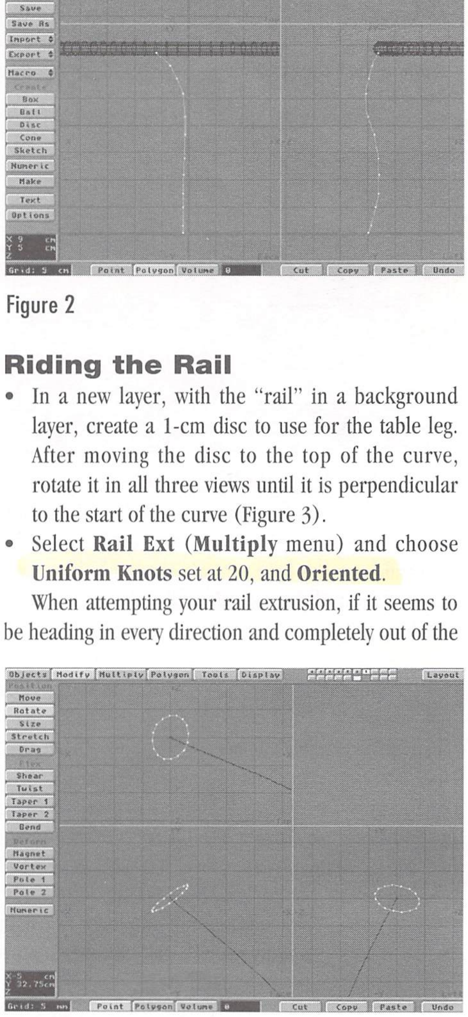


Figure 3

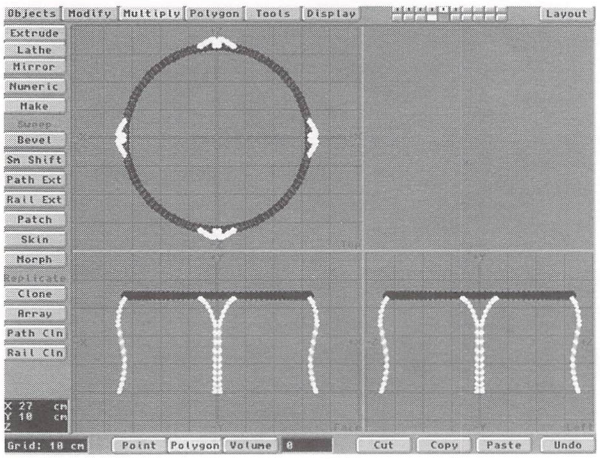


Figure 4

rails path, chances are the curve was drawn backward. To correct this, simply click **Undo** to erase the incorrect rail extrusion, move to the layer with the "rail" in it, and press (f) to flip the direction of the curve. Then return to the layer with the leg disc in it, make sure the "rail" is in the background, and try again.

If, once the disc extrudes correctly, you are dissatisfied with its shape, click **Undo** again to get rid of it, move a few points of your rail until it is more the shape you are trying for, and then try the rail extrude again. It took me a few attempts before I was satisfied with my table leg extrusion, so don't be afraid to shift a point here or there.

The extrusion just completed is only half of the complete table leg.

- To complete the leg, mirror (Multiply) the object just created on the Y-axis in the Face view.

The mirrored objects should just barely touch each other, giving the impression that they are connected somehow. This completes one of the four table legs. Now the table leg needs to be cloned three more times around the table.

- Hit the Clone button (Multiply) to bring up the Clone requester. Enter three for the number of clones and 90 degrees for the rotation on the Y-axis. All other settings should be left at the default. Click OK, and the leg will be cloned three times at equal spacing around the table (Figure 4).

Thanks For Your Support

The next step is to create the connecting leg supports. Since these supports have a perfectly smooth arc to them, the best way to create them would be to use the same lathing method used for the top support.

- In a separate layer, make a flat 1.1 cm disc in the Face view and move it out about 32.5 cm on the X-axis. Again, lathe this disc on the Y-axis with 100 sides.
- In the top view, with the table legs in the background, move the entire ring that was just created until it intersects two of the table legs correctly (Figure 5). The ring can be moved or rotated from the front most edge 90 degrees to align it in place. Move the ring down and align it from all three views to get it to intersect two of the legs. For those with LightWave 3.5, this becomes an easier task while using the visibility tools (Display menu).
- Once you are certain the intersection is correct, cut away all of the unnecessary polygons (highlighted in Figure 5), until you're left with one-fourth of the original ring arching between two table legs.
- As before, clone this piece three times around the Y-axis, with a 90 degree rotation value, to complete the leg support, connecting all four legs.

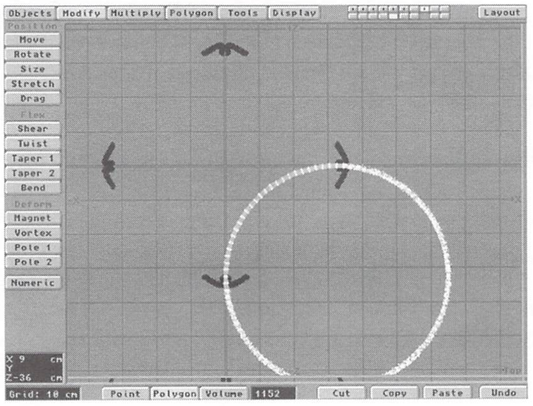


Figure 5

Add Elegance

The final step in creating the table bottom is to make the S-like details between each table leg, which gives the table its elegant appeal. This could be done simply with rail extrusion, as well.

- In a new layer, draw out an S-shape using the Sketch tool in the Face view, and hit the return key to make the curve.

I adjusted the curve's points as needed to make sure the curve was as smooth as possible. I also made sure the curve touched the top support, so the S-details would be connected to the rest of the table.

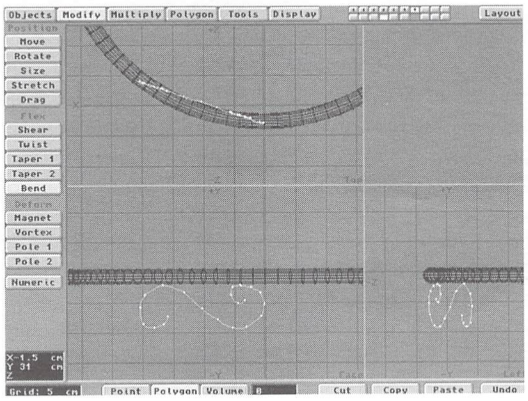


Figure 6

- Using Rotate (Modify), rotate the curve somewhat in the top view, so that the curve follows the arch of the top support, to which it would be connected. Make sure that one end of it is aligned with 0 on the X axis (Figure 6).
- Once satisfied with the curve's position and shape, move to another layer and create a flat 7 mm disc in the face view. Again, with the S-detail curve in the background, move and rotate the 7 mm disc in all three views until it sits perpendicular to the start of the S-detail curve. It may be necessary to flip the direction of the curve, as mentioned earlier.
- Once the disc is in place, use the rail extrude function to create the S-detail.

To create rounded-looking ends, I beveled the polygons on each end of the S-detail.

- Next, as with the table legs, mirror the S-detail, so that the edges just touch each other.
  - To create the other detail, clone the S-detail objects, just as before, with the number of clones being three, with a 90-degree rotation around the Y-axis.

With the S-detail objects in the foreground and the table legs in the background, you can see that they are on top of each other. To correct for this, we simply rotate the S-detail objects 45 degrees around the Y-axis. It is better to use the numeric input requester in this case, so that there is no mistake of things being off-center.

- Now that all of the table bottom pieces are created, it's just a matter of cutting and pasting the legs, the top support, the leg supports, and the S-detail objects into a single layer. Merge points (Tools) and assign the surface name Tablemetal to the whole object, then save it as TableBottom.

Final Touch

You can now load your objects into Layout, surface them and render away.

I gave a metal surface to Tablemetal, and turned smoothing on. I gave Glasstop.T&B a glossy glass surface, and left smoothing off. Tabletop.Sides also received a glossy glass surface, but smoothing was turned on. I also gave it a bluish surface color and turned on **Color Filter**, to give the sides a tinted look.

This project was beneficial in that it helped me become more acquainted with the rail extrude function and the Clone requester. I realize that I am constantly finding new uses for previously avoided tools.

Learning to use the many tools in Modeler, even those rarely used, can provide a new outlook on what can or cannot be modeled, and with new tools like Metaform, it seems as though anything is possible.

LWP

Arnie Boedecker is president of Imagi•Nation Enterprises in Illinois. He can be reached at 603-D WatersEdge Drive, McHenry, IL 60050, (815) 385-8198.

Digital Cinematography

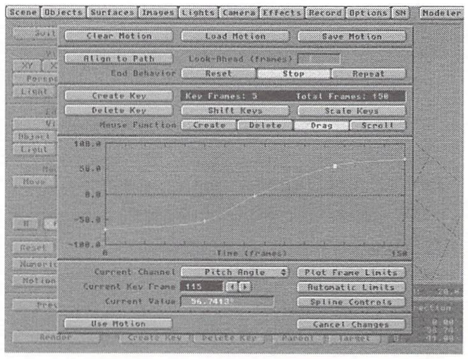
by John F.K. Parenteau

Last month we discussed many of the principles behind good camera motion, including some of the theories involved in various styles. If any piece of information is important to remember, keep in mind that camera motion is as much a character as the actors on the set. What you choose to show and not show makes all the difference in the world, as does how you show it.

Imagine a movie shot entirely from one wide angle. Though it has been done, it isn't an interesting way to use the medium. That's what plays are for. This month, let's take some specific examples of shots that are fairly difficult in live action, but are quite possible (I didn't say easy) in CGI. Though I describe a specific object, feel free to use anything available. It's the method used to shoot the object that's important in these exercises.

Quite frequently, the most complex moves for an animator involve a complex or violent motion of the object, camera or both. There are two methods to make this type of shot possible. Let's first address a shot with complex object motion and a relatively stationary camera. In this scene, we are looking up as a helicopter comes crashing down toward us. We will need to whip pan with the ship as it slides by the camera and, as the helicopter comes to a hard stop, we need to stop as well. The problems we will run into in this scenario are largely with the camera. As the ship starts in the distance, we have little or no camera motion. We will need to quickly ramp up motion to pan rapidly with our ship, then slow that motion down abruptly as the ship crashes. The trick is maintaining a consistent spline while still keeping up with the action. I always find it handy to rough out the move as best as possible early, including trying to manually track the camera. Start with a rough positioning for the move of the helicopter. Slide it up on the Y-axis a good distance, pitching it down toward the ground. Create a keyframe at frame 0. Now move the helicopter to the ground and create a keyframe at frame 150. (This is just an estimate of length at this point. Until we actually work out the entire move, it will be difficult to tell what frames are necessary to make it realistic.) Now, display frame 75, showing the middle position of the helicopter move. Place the camera beside the ship and create a keyframe at frame 75. Without changing the frame you are on, also create a keyframe at frame 0 for the camera. Now, move to frame 0 and tilt the camera up to the helicopter, creating a new keyframe for the camera at frame 0. Since we have roughly determined that our animation will end on frame 150, move to this frame and tilt the camera down to view the helicopter on the ground, creating a keyframe for the camera here. You may end up discarding most of the keyframes, but you need to start somewhere. Consider this your pencil sketch.

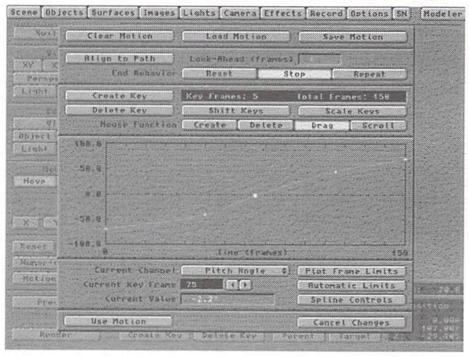
Of course, if you were to preview these motions, the effects would most likely be quite random and highly unacceptable. The camera will overshoot the ship in the beginning, leading it too much in its effort to reach the middle keyframe. As the helicopter drops by the camera, we will lag behind it a bit. This is what I mean by competing splines. Though the motions are simple, the actions and settings for each keyframe differ enough to change the individual characteristics of the motions as they relate to each other. Considering the drastic difference in the two motions in this example, with the helicopter dropping from the sky while the camera is just tilting to follow, it is quite obvious how the splines do not match up. But even the most subtle motions may not match either. Pay close attention to the previews to note differences in the motions. Sometimes it is even worth the time to create a wireframe preview for more specific information to view. Adjustments are not necessarily a mathematical process, but most likely a visual one. The significant issue to take into account is the condition of the spline of each object. Since neither the camera nor the ship is connected in any way, their splines are being affected independently by the keyframes that make them up. For this reason, it is important to create keyframes on the same frames for both camera and object. Even if slight differences in keyframe positions exist, the subtle effects these differences have on the motions will prevent a smooth comparison between the two motions.



Look-Ahead (frames)
End Behavior: Stop
Key Frames: 5 Total Frames: 150
Mouse Function: Drag
Current Channel: Pitch Angle
Current Key Frame: 115
Current Value: 56.7413°

Figure 1

If you do create a preview, you'll notice that the camera moves much quicker than the ship, panning off then back on at the keyframe positions. No matter how hard you try, the splines will always react differently if the motions are even slightly different. In this case, the ship is dropping quickly through frame as well as rotating and pitching violently. The camera is only pitching, thus creating a much less complex spline curve. The next step we take is placing additional keyframes between our existing ones. I've estimated a keyframe position of 75 and 115. Though this is not exactly a precise split between the other keyframes, they are placed where the camera is farthest out of position. Pitch the camera back to frame the helicopter in center frame again at each of these keyframes. Make sure to create keyframes at these



Look-Ahead (frames)
End Behavior: Stop
Key Frames: 5 Total Frames: 150
Mouse Function: Drag
Current Channel: Pitch Angle
Current Key Frame: 75
Current Value: -2.2°

Figure 2

new positions for the helicopter as well. Now create a preview. Though the ship is in frame for most of the shot, it is no longer a smooth tilt. Examining the camera motion graph (Figure 1), we can see how the pitch spline is not an even progression. By evening this up, we can smooth out the unacceptable bumps in the tilt (Figure 2). Creating a new preview, the camera rushes through the tilt a bit more, actually tilting off the copter partially. This may or may not be a problem since perfection in a camera move isn't necessarily a desired effect. As an object moves quickly by camera, it is natural to lead the object rather than lag behind and miss the action. By inputting some bank in the keyframes for the camera (Figure 3), we can enhance the randomness of the shot.

Though this may be acceptable for this application, it may not be for others. It is important to be

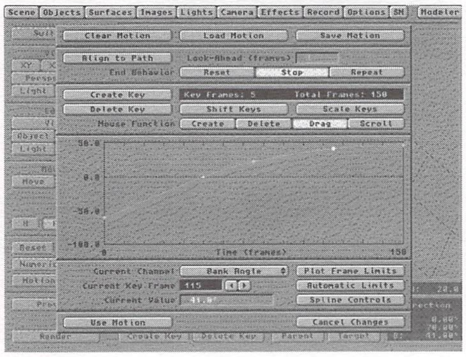


Figure 3

able to create a smooth move as necessary. The smoothest method of creating a tracking camera motion is by targeting the camera to the object. One of the newest projects here at Amblin Imaging is the new *Star Trek* series premiering in January, *Voyager*. We have been working out the move for the final jump-to-warp shot of the show. Not to give too much away, but the producers wanted to track a specific area on the ship as it came toward us, pivot on this area and then watch it jump to warp from behind. We had 11 seconds for the shot. That may seem like a long time, but to move a huge ship from offscreen, toward camera and by camera, then jump to warp, takes a long time. I actually attempted the simple method as described above with our helicopter, using individual splines for the camera and the ship, but found the complex pivot couldn't be ironed out. Instead, I created the motion for the *Voyager,* then saved the *Voyager* motion and applied it to a null object. By targeting the camera to the null, I would always track the object perfectly, since the null has the same exact motion. Why not track the ship itself?. The easiest reason is that I wanted the ship to enter the frame. By targeting the actual ship, I would never be able to pan the camera off at the start of the shot. I also wanted to track a specific piece of the ship's anatomy (not the bridge), and these areas did not happen to fall at the pivot point of the object. Needless to say, targeting the ship doesn't allow for any modification. By targeting a null with the same motion, I can modify various keyframes while still maintaining roughly the same motion. In the first keyframe I shifted the null object so the *Voyager* was offscreen. As the ship moved toward the camera, I shifted the null object on the X and Z axes to maintain a desirable camera view of the ship. I don't want to give too much away, but the shot would not have been possible without this function. Let's examine our helicopter scene with this new wrinkle.

Starting with our old scene, target the camera to the object. As you run through the anim now, the ship is always perfectly in frame, with its pivot point carefully placed in the center by LightWave. Though our move is a perfect pan now, it doesn't allow for much variation or adjustment. Now save the motion of the ship and apply it to a null object and re-target the camera to it. It is important to remove any rotational information from the null object. This will not affect the camera, but it will inhibit your adjustment of the object, since LightWave confuses axes when the rotation has been greatly affected. Move to each keyframe and use the numeric input to zero out the rotation. Also, remove the bank on the camera if you applied any in the previ-

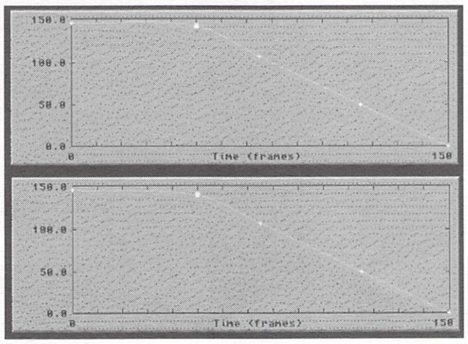


Figure 4

ous exercise. (The pitch is disabled by the target function.) At first, nothing has changed, but the first frame needs adjustment. Drop the null object on the Y axis, pushing the ship to the top edge of frame rather than the center. Move to the second keyframe on the null object and enter your motion graph for the null. As you examine the Y axis, the spline between the first and second keyframe actually backs up, moving up to prepare its acceleration for the subsequent keyframes. By dragging the second keyframe down to prevent this, you will create a smooth tilt for the camera. Drag the second keyframe down just enough to create as straight a line between the first and second keyframes (Figure 4). Now use this motion and preview your animation. Note how the ship starts out of frame but enters smoothly as the camera catches up. At the end of the animation, I slipped the null back on the Z axis to center the ship. Viewing the motion graph once again, look at the Z axis and adjust the second-to-last frame to create a smooth transition between this and the last keyframe (Figure 5).

Though the examples above are good learning tools, keep in mind that each animation is drastically

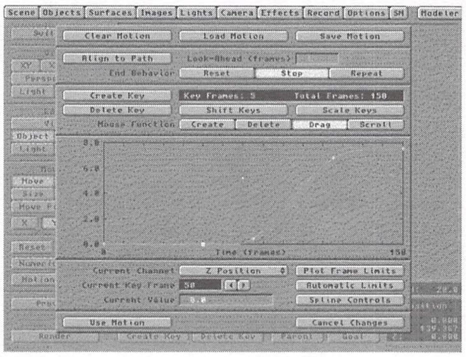


Figure 5

different. It is best to follow these basic rules for complex motions:

If both the camera and objects have motion, create keyframes for both in the same positions. For example, if you have animated your object and are now applying motion to the camera (though the camera motion doesn't require as many keyframes), put keyframes at the same frames as the object. This will help create similar spline attributes between keyframes for both. Use the motion graph religiously. Pay attention to overshoots on all axis. If the camera starts from a dead stop, holding for a second or two then moving, check the second keyframe to make sure none of the splines have backed up to start the move. This always produces an unwanted glitch in your motion (Figure 4).

Ignore the velocity graph for the camera if the camera is targeted to a null. Since several axes are ignored when the camera is targeted, the velocity graph is usually pretty scary. Pay closer attention to the null object velocity graph since it contains the master motion.

If the object is going to stay in frame during the entire animation, parent the null to the object rather than loading its motion. Since the null is parented to the object, the slight adjustments left and right and up and down to frame the object will require very little change in motion, and thus a simple spline. Imagine a null with the motion of an object loaded on it. Not only does the spline have to move from great distance to great distance, but it also has to make the new adjustments for proper framing. By parenting the null to the object, the null has no motion as it moves through space with the object. Thus, any minor framing adjustments you make create a spline with minimal change.

Try a few whip pans with both methods to see how they look. Remember that camera movement is not a science, but an art form. There is no right or wrong, only your perception of good or bad. Creativity is what most clients pay for, and that creativity is what gives you your own personal style.

LWP

John F.K. Parenteau is vice president and general manager of Amblin Imaging, whose CGI work can be seen regularly on seaQuest DSV.

NAB MultiMedia World

CONFERENCE: APRIL, 9-13, 1995 • EXHIBITION: APRIL 10-13, 1995
LAS VEGAS HILTON • LAS VEGAS, NEVADA

Helping Business Profit from Multimedia

The Interactive Multimedia Association and the National Association of Broadcasters have assembled the best minds in multimedia to help you take advantage of current and future multimedia opportunities. NAB MultiMedia World offers strong, practical advice on integrating multimedia into your business and how to make it profitable!

THE EXHIBITION

Explore the industry's most comprehensive showcase of multimedia technology, products and services from more than 100 of the world's leading multimedia developers and manufacturers.

THE CONFERENCE

Gain the knowledge and the skills you need to compete and perform in multimedia, video production and broadcast communications.

ADDED BONUS!

Maximize your production or animation system's performance in the ALL NEW Light Wave 3D training classes and *Video Toaster User* Pavilion, #117.

For More Information

CALL
1-800-342-2460

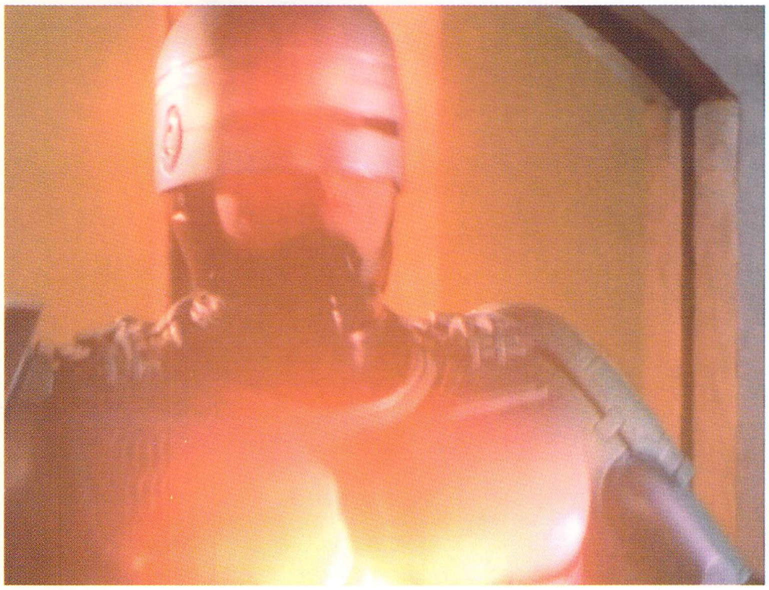
FAX-ON-DEMAND
301-216-1847
CALL FROM THE HANDSET OF YOUR FAX MACHINE

INTERNET
<register@nab.org>

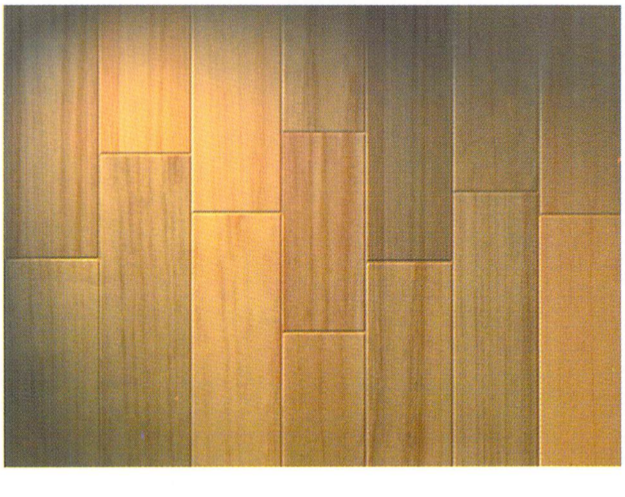
REGISTER TODAY!

LWP

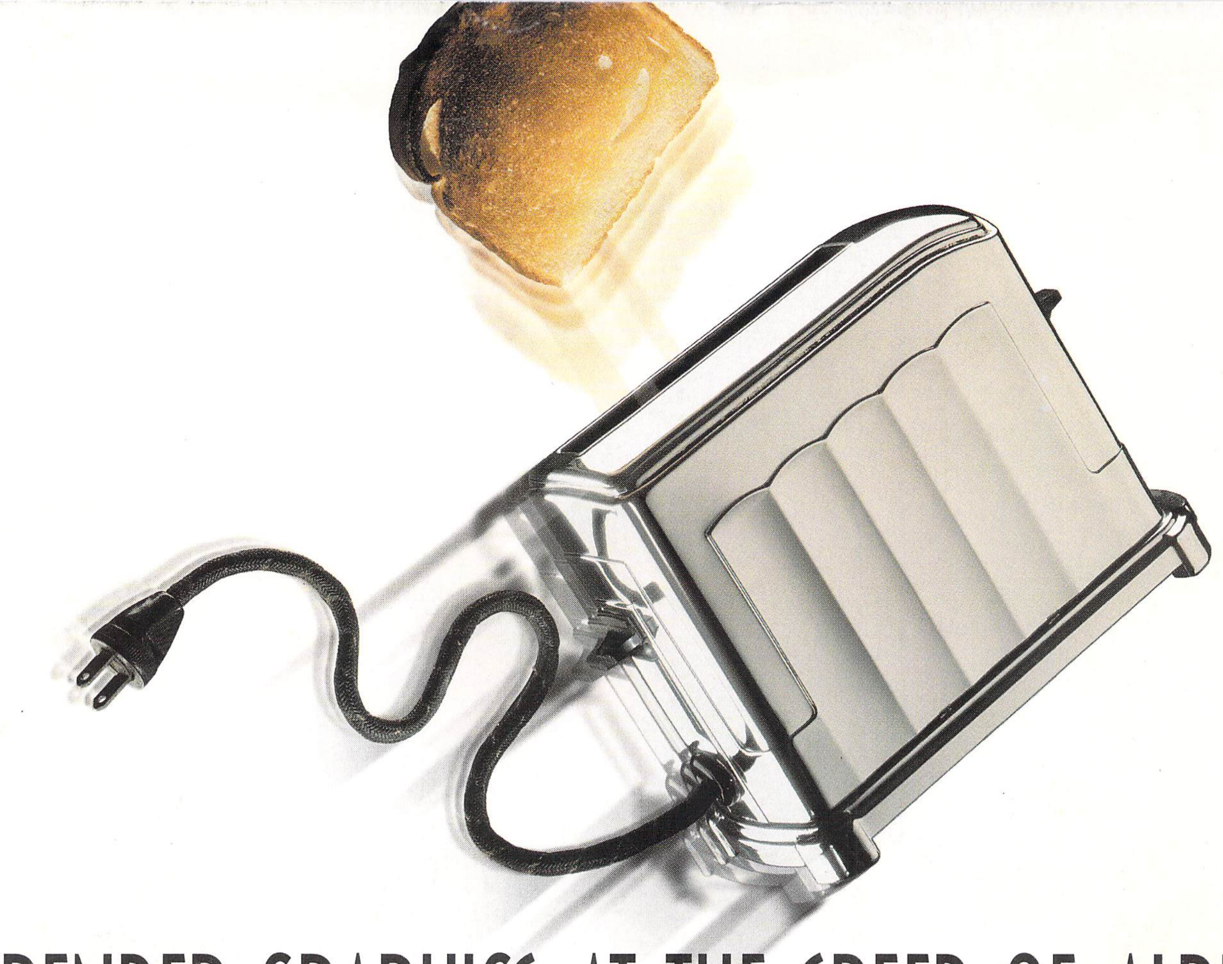
Robo Explode
Lens flare clusters and the fireball object make an explosive combination.
Copyright Skyvision Entertainment 1994



Robo Glow
Carefully placed lens flares give the impression of glowing steel while refractive heat signature adds a subtle touch of realism.
Copyright Skyvision Entertainment 1994

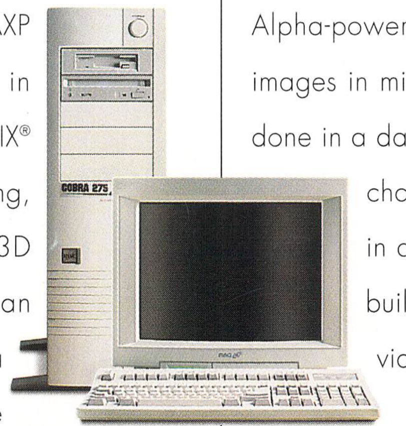


Light Wood
The final result of using our wood creation techniques.
Copyright Dan Ablan 1994



RENDER GRAPHICS AT THE SPEED OF ALPHA ON A COBRA AXP 275 WORKSTATION.

ALPHA GENERATION™ There's no better way to burn graphics in LightWave™ or other applications. Introducing the Carrera Cobra AXP 275, the workstation leader in price and performance. Run UNIX® and Windows NT™ frame rendering, animation, multimedia and 3D graphics applications faster than you've ever seen on the power of a 275MHz Alpha™ processor—one of the 64-bit RISC rockets from Digital Semiconductor, a Digital Equipment Corporation business. With the blistering performance of an Alpha-powered Cobra, you'll generate digital images in minutes instead of hours. You'll get more done in a day. Maybe even get home on time for a change. And the Cobra AXP 275 comes in a variety of configurations, loaded with built-in PCI SCSI-2, PCI Ethernet, PCI video, PCI and ISA slots, CD-ROM, and more. Call or E-mail us for details. Then get ready for a workstation that really cooks.



CARRERA Computers, Inc.

23181 Verdugo Dr., Building 105A, Laguna Hills, CA 92653 • 800-576-7472 • e-mail: CARRERA1@DELPHI.COM