HDRI 3D

LIGHTING THE WAY FOR DIGITAL ARTISTS

ISSUE #6

Copyright 2005 Columbia Pictures
Copyright 2005 Columbia Pictures
Copyright 2005 Columbia Pictures

Stealth Busts Off the Frame

INSIGHTS & TECHNIQUES from Professi

Issue 6 $10.95 USD $13.50 Canada

From the Editor

Dariush Derakhshani
Editor-in-Chief

SOAPBOX KOOSH

I don't see why I can't walk down a dimly lit hallway and take a few steps down into the cellar to access my utility application icons like disk defrag and the like on the workbench. Or take the small hidden door into the secret den behind the cellar's pegboard to get to my finance applications like Quicken and do my tax receipts. Or perhaps you would rather take a short walk in the grass to your shed to find your drive defrag icon. Take the spiral stairs up to get to your office and word processing apps.

Why can't we take game mods and apply them to our OS directly? If we ask our video cards to display these stunning 3D environments when we go out to shoot up some disgruntled monsters in our games, why can't we ask for the same environments for our desktop experience? Right now we are limited to a flat desk. On that desk sit our icons. We click and we go. But how far are we from the day when the desktop is a full 3D space? How much longer is it till we can mod it anyway we want using built-in 3D in the OS? Aside from the fact that I forget where I put things all the time, it would be a hoot. I imagine opening a CG roll top desk to get to the drawers in my desktop office scene to access my articles in Word doc format. Another drawer and I have my book's doc files. Close the roll top desk, and off through the door to the game room where I can launch CounterStrike and get my ass shot off by my friends online.

This level of computing is essentially here, though perhaps not as widespread as need be. 3D is slowly creeping into the OS desktop, and is seen in some OSX setups, as well as the much-anticipated Longhorn version of Windows. These are, of course, much more limited than my musings here, but really how long will it take for an OS desktop experience like this to become popular? All we need is an OS-based 3D engine like those found in popular 3D games and we're all set (for the most part). Before long, mods will be everywhere and you'll be able to set up your system to be a house, a compound, a dark dungeon, a cave, a shrub maze, or anything else you can think of to create for yourself or download off the Internet. We just have to see it here and there, and I doubt it will be long to spread long and wide.

Next to speech recognition, which has to be the next major input advancement in computing, 3D space is the king of what's to come next in our everyday computing.

Now on to the magazine! I am thrilled to have *Stealth* on our cover this issue. Having heard a lot of the work Digital Domain has done for this movie, I knew it would be a good thing to get a scoop on it, so thanks to Digital Domain for their help. It's exciting to see houses like CafeFX (last issue's *Sin City* cover) and Digital Domain and others we have been speaking to (like Imageworks, DreamWorks, and Pixar, to name but three) becoming very excited to share their work with us and help us broaden people's awareness of workflow and procedure. It just affirms my belief that our field is still very excited about education and sharing information. Seeing everyone at SIGGRAPH just cements that feeling even more.

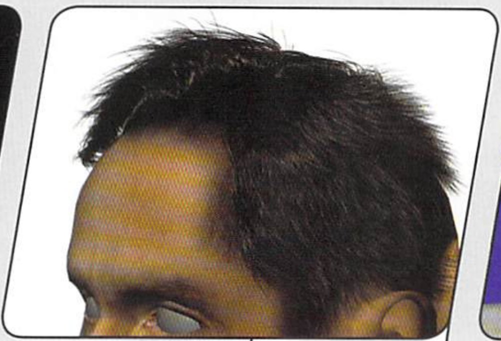
Speaking of SIGGRAPH, there were some nice things to see, but the two that stood out most for me were two different displays from NEC and BrightSide. Both of their separate technologies bring high dynamic range color depth and rich blacks to flat panel displays. It's very intriguing what they're cooking up, and I hope to have my greasy hands on them soon to play with and report on. I think this is an important step in display technologies, to allow us to see high dynamic ranges and colors that match what we intend to output. Having a good display is important, and having used Dell's stunning 24" widescreen LCD and Apple's beautiful 23" Cinema Display, I'm eager to see the next step in full force and out there in the market for everyone, though I figure that will be a while. Keep an eye peeled.

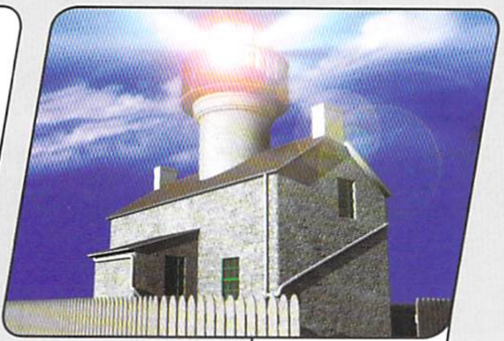
So well anyways, in this issue we have a nice spread on RenderMan and MTOR from our friend Peter, which I'm jazz hands about right now. There is so much work done in RenderMan, it's always nice to get more about it out there for people to read. We'll also be checking out Pixar's RenderMan for Maya plug-in in the next issue. Having had a look at it up at Pixar recently, I can tell you its pretty damned cool. It's completely integrated into Maya's UI, so there is no need for an intermediary step (like MTOR, RAT, etc.). I think that's a bit of a coup; it brings RenderMan into the reach of small boutiques that work on short term and fast turn-around projects that don't have the facility for RenderMan shader writers and RenderMan ProServer. And man, the fur it renders is beautiful. More next issue.

Boaz is taking the next step in his mental ray series, and how to use Nuke to composite lighting passes together. The same workflows carry over to Shake and other node-based systems, and covers some solid concepts in layered rendering. Jake gets further in his Normal Mapping exercises using LightWave. If you're into gaming, this is hot stuff. We get to see some hair work in SOFTIMAGE|XSI and more on MEL scripting in this issue. William "Proton" Vaughan answers 20 questions on LightWave you know you need to have answered, and Brad gives us another informative LW Essentials. We also have an animated tattoo done with Paint Effects and a lot more, so dig in and keep letting us know what you like to see in HDRI 3D.

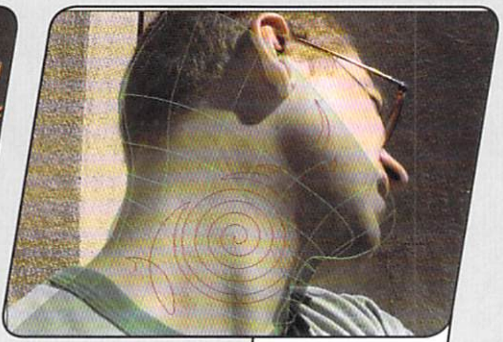
IN THIS ISSUE

PAGE 12


PAGE 23


PAGE 39

PAGE 51


PAGE 72

tutorials

Resource files and selected screen shots for this issue are posted at **www.hdri3d.com/resources**

contents

HDRI 3D

Issue #6

PUBLISHER
Alice Edgin alice@hdri3d.com

EDITORIAL DIRECTOR
Charles Edgin charles@hdri3d.com

EDITOR-IN-CHIEF
Dariush Derakhshani dariush@hdri3d.com

EDITOR
Brad Carvey bradcarvey@hotmail.com

EDITOR
Raffael Dickreuter raffael@hdri3d.com

TECHNICAL EDITOR
Philip Duncan philipd@hdri3d.com

MANAGING EDITOR
Amber Goddard amberg@hdri3d.com

ART DIRECTOR
David S. Wilkinson david@sabredesign.net

LAYOUT/DESIGN
Sabre Design artists@sabredesign.net

CONTRIBUTORS

Stephen Burns	*chrome@ucsd.edu*
Brad Carvey	*bradcarvey@hotmail.com*
Jake Carvey	*jake@spinoffstudios.com*
Raffael Dickreuter	*raffael#xsibase.com*
Peter Hartwig	*peter@idiotbarn.com*
Eric S. Keller	*kellerrific@yahoo.com*
Aaron Kent	*kindofshiftless@yahoo.com*
Boaz Livny	*info@visionanimations.com*
Ron Magid	*magron2@yahoo.com*
Peter Ratner	*ratnerpj@jmu.edu*
Robin Scher	*robin@uberware.net*
William "Proton" Vaughan	*Proton@spinquad.com*

ADVERTISING SALES 1-702-990-8656
advdmg@dmgpublishing.com

SUBSCRIPTIONS & BACK ISSUES

Online Store	www.dmgpublishing.com
Email	mag@hdri3d.com
Phone	1-702-990-8656 or 1-877-303-4629
Fax	1-702-992-0471

CHANGE OF ADDRESS newaddress@hdri3d.com

HDRI 3D MAGAZINE
(ISSN 1551-689X)
is published bimonthly by
DMG Publishing
2756 North Green Valley Pkwy #261
Henderson NV 89014 USA

A Step Closer to Believable Digital Acting

Raffael Dickreuter
Editor

For the past few years, we have only seen rather small improvements in terms of tools for creating believable character animation. The holy grail, the face, remained mostly untouched. Although we were able to see breakthrough performances by characters such as Gollum and Yoda, it still seemed out of reach for the large majority of CG content creators.

At Siggraph this year in Los Angeles, besides the new SOFTIMAGE|XSI 5.0, Softimage previewed a new software called FaceRobot. It basically minimizes the tedious and complex process of rigging a face to a few minutes for a basic setup. Demo videos created in cooperation with Blur Studios, well known for its breathtaking game cinematics, gave us a glimpse of where this could go. FaceRobot basically lets you select certain key points on the face, and the software will rig the face for you within a few minutes and is supposed to work on almost any human-like face.

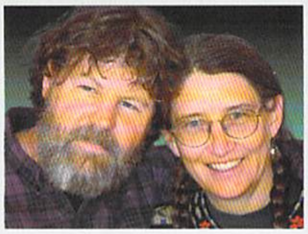
Obviously, the tool goes much further than the common practice of Blend Shapes or basic muscle systems; instead, it seems to create that fleshy look and lets you add all the subtleties that we are looking for in a face to see a lifelike performance. It also seems to handle extreme expressions, such as screaming or eating, really well. Softimage is secretive about the technology, price, and release date; however, this software could be a big step towards real digital acting. Maybe it will open the door to having many complex characters with facial rigs that are produced by a smaller team in a much shorter amount of time than ever before.

To get digital acting to the next level and make today's digital performances much more believable, such a tool will be key. Until now, the process of rigging many complex faces seemed too time consuming and expensive to be taken on by many smaller studios; it was just a task for larger production houses with the means and time to invest in it.

Softimage was one of the first companies to ever use IK. Back in the early 90s, when *Jurassic Park* wowed audiences around the globe, Softimage was leading the industry to a new area. It's hard to predict if this could happen again with FaceRobot over ten years later, yet it is an exciting step further and might give the industry the momentum that is needed to move on to the next level. This is probably just the beginning.

Celebrating Our 25th Anniversary
T. S. Computers 11300 Hartland St. N. Hollywood CA 91605
818-760-4445 www.vgn.com

STUFF



By Brad Carvey
Electrical Engineer, 3D Animator
New Mexico

If you are a graphics artist and don't have a PSP, buy one. That's the conclusion I came to recently after buying a PSP "Play Station Portable." I have not played any games on the PSP. I suspect that at some point I may play games, but that's not why I bought the PSP.

Recently, while at a large family gathering, I was able to talk to some relatives that I rarely see. The standard questions usually start with something like, "what have you been doing." I have a cel phone with a 128 MB memory card, so I showed a few animations that I just happened to have on the phone. Everyone was very impressed. It was painful for me, because I was not prepared to show stuff and I did not have anything exciting on the phone.

To avoid getting caught in that situation again, I decided to buy a PSP to show demo materials and see if it was as good as I thought it would be. The PSP display is great. I have been converting demo material to PSP format using a free product called PSP Video 9. I have also downloaded large images to the PSP. I created an image of a composite I did for *Kate and Leopold*. In the image, I placed the raw background, foreground, and the color corrected composite. I was able to use the PSP analog joystick to pan across the image, showing the different elements.

The PSP has come in handy a few times already. I had a meeting with a director who was interested in some graphics. I was able to show him and others a few example animations and an image that I had created that was something like what I thought he wanted. People were crowding around to see the PSP and the animations I was able to show. The meeting changed from an interview to a discussion of what they wanted me to do.

In movies, when you see a monitor or television, the graphics on the screen are not real. For example, on shows like *ER*, a computer animation, called a "playback graphic," drives the medical equipment. Recently, I was on location to supervise a GPS playback graphic that I had created. The Prop Master had a small TV monitor. We planned on connecting it to a computer and mounted it on the dash. We had tested everything, but minutes before shooting the scene, the monitor stopped working. We then went to the backup solution, which worked, but only for a few minutes at a time. Sometime during the first take, the monitor went black and displayed "Check Disk" with a red background. No one was sure if we got the shot, and the next effort did not work at all. So, I went to the producers and offered to add the screen to the monitor in post-production for free. The director was upset and came over to the producers to inform them that the shot would need to be "burned in," which would require more money. The producers told him that I would do it for free. He was surprised and left. The producer was concerned about my ability to do the new shot. The camera was handheld in the back of a car that accelerates from standing still and there would be a hand moving in front of the blank screen. I assured him that I could. I think that because I was just doing a simple animation for the movie and because I was from New Mexico, he assumed that I was only capable of simple stuff. He had heard rumors about a legendary graphics pioneer living in New Mexico who occasionally did stuff for the movies. He described this guy in detail, and I realized he was talking about me. It was kind of embarrassing when I eventually stopped him and said that's me.

I just read about a Dell 2405FPW 24" 1920 x 1200 monitor for $959.20 at www.dell.com. I read a review of it that said it was as good as or better than the Apple monitors. It has inputs for RGB, DVI-D, S-Video, Composite, and Component. It supports PIP, so you can watch TV on it in a window while you work. It has four USB ports and slots for Flash Data Cards. I want one of these. Actually, I want two of these.

***Brad Carvey** has been doing computer animations for a long time. In 1969, Brad used an analog computer, which was the size of a car, to produce his first computer animation. Brad is an electrical engineer and an Emmy award-winning member of the Video Toaster development team. He prefers to do feature film work. His credits include films like **Men in Black**, **Stuart Little**, **Black Hawk Down**, **Kate & Leopold**, and **Master of Disguise**.*

Copyright 2005 Columbia Pictures

STEALTH FX

By Ron Magid
Writer, Screenwriter
Los Angeles, California

Director Rob Cohen's need for speed inspired him to make *Stealth*.

"When I read the script, I went, 'Nothing moves as a fast on earth as hypersonic jets – that's as fast as we go here,'" says Cohen, whose previous adrenaline rushes include *The Fast and the Furious* and *XXX*. But *Stealth's* arena – a slightly futuristic Man vs. Machine tale of Top Gun pilots confronting thinking-drone aircraft, set against the all too real world scenario of taking out a terrorist network – touched Cohen's conscience. "I'm addicted to visual speed, but I also wanted to make a movie about war and technology with an ethical question."

The movie's moral conflict is epitomized when flyboy Ben Gannon (Josh Lucas), fearing that the cognizant Extreme Deep Invader (EDI) will render fighter pilots useless, refuses to send one on a dangerous mission to destroy a terrorist stronghold in densely populated downtown Rangoon. Instead, Gannon willfully pilots his hypersonic AFG Talon jet – a fictional, super-sleek stealth fighter constructed of a futuristic graphite, fiber, ceramic, and metal composite – knowing he might crash and kill thousands of civilians.

While sending EDI is clearly the better, safer choice, it's not nearly as dramatically exciting as the cinematic tour de force that follows. The terrorists have holed up in a building capped with a 14' thick steel reinforced concrete roof. The only way to penetrate the roof is to unleash the Talon's rear pulse det engines – 16 small cannons filled with explosive gas surrounding the fanjets on each side, firing in rapid sequence – then dive-bomb the structure at lightning speed, drop a truncheon bomb traveling at hypersonic velocity, and pull out just before impact.

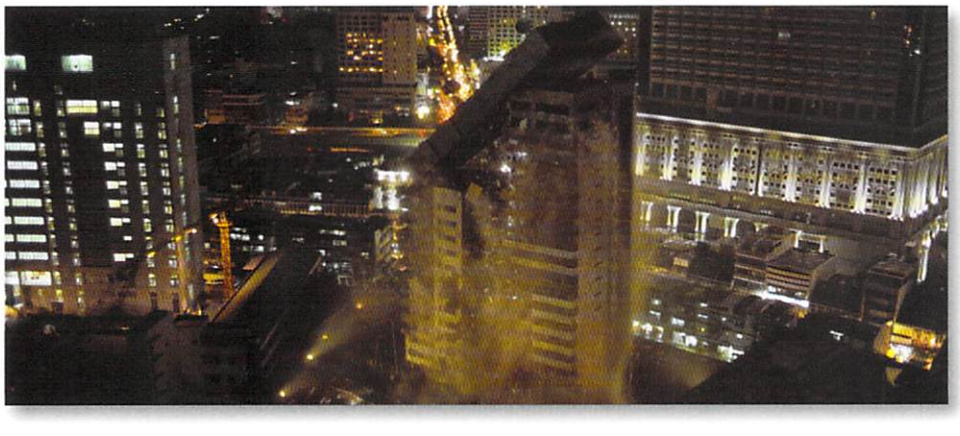
To do that, Cohen ordered Digital Domain's visual effects supervisor, Joel Hynek, a veteran of Cohen's *XXX* and a pilot himself, to stage a fantastic coup de cinema, following Gannon's Talon straight up into the heavens, then doing a whip pan to see the plane plunging headlong toward its target.

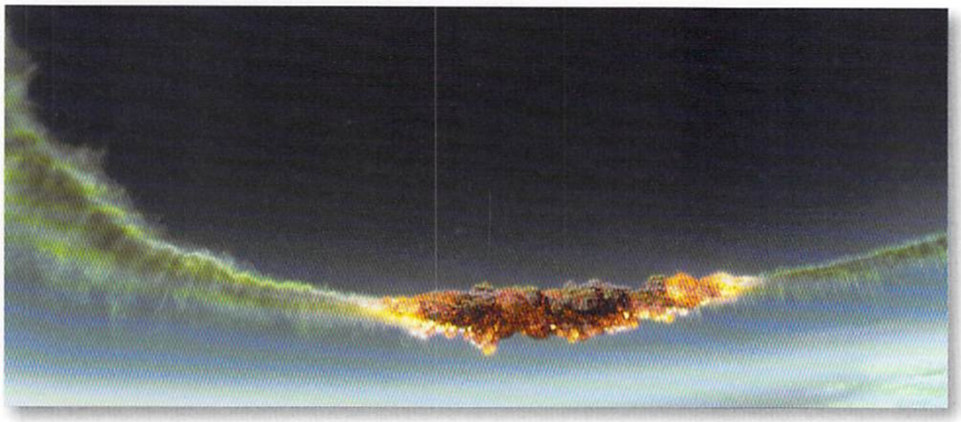
"I wanted to break down the fourth wall of the sensation of speed and the sensation of flying by changing the perspective of the viewer within the given shot," Cohen says.

These dynamic shots pushed the limits of motion control when it came to shooting the actors on stage within a mockup Talon cockpit against greenscreen in a Sydney, Australia, soundstage.

"Rob wanted to give the actors a sense of really flying," Hynek says, "so we had this cockpit set - a gigantic gimbal driven with Cooper controls. We had a Spidercam, which is a camera suspended from four cables, flying around the gimbal, which could also be driven with Cooper or handled wild."

Copyright 2005 Columbia Pictures





Copyright 2005 Columbia Pictures

Wild turned out to be the mode of choice when the complex motion control camera and cockpit actions threatened to slow Cohen's pace. When the director chose to "wing it," hooking up the live action plates to the CG planes back at Digital Domain became quite complex.

Copyright 2005 Columbia Pictures

"Rob would do his best guess on what he needed when he shot his plate on stage," says TD Swen Gillberg, whose job it was to make those hookups work. "But often, Rob would come up with different camera moves later that we didn't necessarily shoot, so we'd use what we called our 'projection setup' – a 2 1/2D process that tracked the original cockpit plates, then projected that imagery onto 3D geometry in Nuke – to make new moves."

The trick was not to twist and bend the 2D photographic image of the actor in the cockpit when the newly created camera move went off the original axis.

"We didn't want to re-record the actor from a different perspective than he was shot from," Gillberg explains. "Otherwise, he'd start stretching and looking a bit odd."

Gillberg's solution: map the 2D cockpit onto 3D geometry, then jam the 2D actor element back into that semi-dimensional cockpit.

Copyright 2005 Columbia Pictures

"We'd roto out the actor, project him onto a 2D card with his matte, then force the actor's 2D card to remain perpendicular to our new animation camera position," says Gillberg, noting that this technique allowed DD to go "all the way off axis. Anytime the camera deviated seriously from the cockpit, any occlusions that weren't covered by the original plate would be replaced by CG material, but this way, we kept as much of the photographic actor element as possible, and avoided doing a full CG hero actor up close."

On the other hand, *Stealth*'s mythical AFG (Angry Finger of God) Talons – their forward-swept wings folding, linking with the smaller canard wings near the cockpit to form a diamond shaped silhouette as its pulse-det engines fire, creating a visible shockwave bending around the supersonic jet – were made airborne using Digital Domain's CG technology to the tune of 650 shots. DD's artists modeled and animated the Talons in Maya, then textured them using real world textures pulled off a full-sized static mockup of the jet built for scenes of actors climbing in and out as it sat on an aircraft carrier deck, as well as texture paintings created in PhotoShop, and finally rendered them in RenderMan.

In the sequence Cohen calls "The Big Suck" – so named because the truncheon bomb creates a huge implosion that literally sucks the terrorists' hangout to the ground – the Talon headed straight up, then straight down at its target. Creating the airborne environment fell to Lead Compositor Brian Begun, using plates shot by Hynek from a helicopter high above Bangkok, standing in for Rangoon.

"But he couldn't get the helicopter high enough, so he shot tiles from different altitudes," Begun says. "Our job was to lay those tiles on a plane, like a satellite photograph of the city, to create a much higher altitude view. The tricky part was keeping the same quality resolution regardless of what altitude we were at, even though we're layering different resolution tiles on top of each other like a sandwich. Then we digitally matched the tiles up as best we could in Nuke, where a matte painter seamed those different resolution tiles together."

Workflow issues were significantly reduced by the fact that most of DD's elements were not only composited in Nuke, they were often created in the proprietary pipeline.

"We not only used Nuke to create the cityscape the Talon was flying over, we also used it to create a cloud environment," Begun explains. "The clouds and the night sky were originally generated in Storm, our volumetric renderer, then we created single frame images of the clouds, shot 'still photographs' of them from various angles using a stationary virtual camera, and rebuilt the stills inside of Nuke. We could then fly our virtual camera through those clouds and create almost any camera move we wanted – with certain exceptions. If we fly the camera too far off, we're going to see that there's no perspective in the clouds. Even the stars were generated in Nuke. On top of that, we added volumetric vapor and clouds whipping by close to camera to give us a sense of speed."

Via Nuke and a combination of Hynek, Begun, and Gillberg's skills, Cohen's virtual camera follows the Talon as it goes straight up like a rocket, then reverses course, plummeting toward the 14-story terrorists headquarters, the cityscape - replete with animated cars traveling on the roads - rushing toward camera, which jumps in and out of the cockpit at will, often in counterpoint to the Talon's movements.

"It's hard to sell speed cause you're up so high and the landscape doesn't look like it's really moving, so that camera rotation really helped create the sense of speed as he's flying straight down onto this building at night," Hynek says. "The closer we could make the planes fly to camera, the faster they looked, so the camera's stuck to the plane, and we have all these clouds going by and vapor and shake, which creates the sense you're going really fast."

Beyond *Stealth*'s digital firepower, "The Big Suck" demanded one of the most sophisticated miniature effects ever attempted. The 1/12 scale model building, constructed by Digital Domain's Alan Faucher and a six-man crew over three months, was a perfect recreation of a 14-story structure located in Australia. Small particles can betray scale, so bigger is always better, ironically, when it comes to miniature destruction, and at 20' tall, the miniature was the perfect size to sell the scale. The model had to be even bigger in order to appear to implode – it needed to descend into an additional 25-foot deep understructure below ground level.

"To call that 50' high rig a miniature was an oxymoron," Cohen says.

Although they created most of the Talon's dive using Hynek's aerial tiles, Begun created the shot of the bomb actually dropping into the building using stills of the model itself.

"We took stills on set looking down at the miniature before they blew it up, then projected those shots onto some rough geometry on top of our city tiles," Begun says. "That way, when the camera moved down, the perspective changed."

The result?

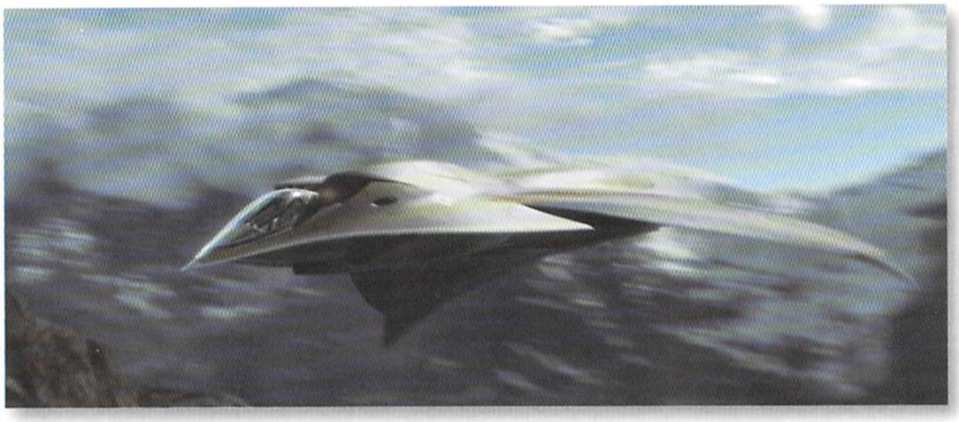
"You really feel like you're plummeting at 2200 miles an hour," Cohen says. "It's pretty breathtaking."

Back on the miniature set, a lever released cables in rapid succession, causing the building to split into three sections as the center collapsed, followed by the columns on either side.

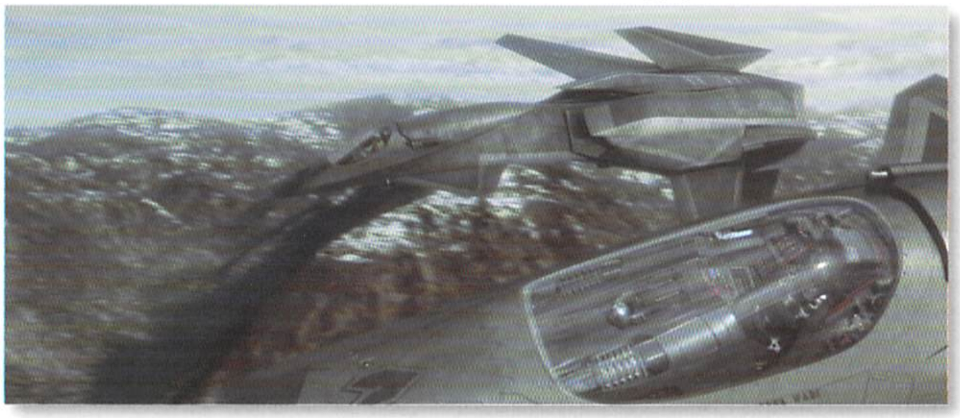
"Each column was on an elevator and would just slide down into shafts below the supposed ground level," Hynek says.

After the three columns collapsed, the thick concrete roof plunged into what had been the center of the building. Six high speed cameras running from 80 to 160 frames per second captured and slowed down the action to impart the illusion of life-sized scale, enhanced with CG debris and surrounded by plates of downtown Bangkok.

As the plane barely pulls out of its death-defying dive, it hurtles down the streets of Rangoon, again courtesy of tiles shot by Hynek – this time atop a firetruck ladder and shooting stills in all directions at 60' intervals down four


Copyright 2005 Columbia Pictures

Copyright 2005 Columbia Pictures

Copyright 2005 Columbia Pictures


Copyright 2005 Columbia Pictures

different boulevards – and assembled at Digital Domain.

"After we brought all the images in, we stabilized every image to the previous image because not every picture fit on top of the other exactly right," Gillberg says. "Brian was quality control: we had lots of people doing a really time-consuming

streets, which Begun carefully seamed together into one street in Nuke.

"After we seamed each of those different perspectives together and ran it back," Begun says, "it became moving footage."

Every frame of it was needed as the Talon blazed through the streets at what appeared to be 1000 feet per second.

Copyright 2005 Columbia Pictures

process of getting all the images in sequence. Brian also supervised color temperature correcting each image to the previous image, because the light changed, etc., while Joel was shooting on these busy streets in Bangkok."

Hynek shot 50 setups - looking north, south, east and west – on each of the four

"We went forward and backward, then looped the backward to the forward section," Gillberg says. "Because the plane's moving very quickly, we needed motion blur. Traditionally, we'd blur something like that using OptiFlow, our in-house pixel tracker that blurs in-between frames, but that didn't work because

Copyright 2005 Columbia Pictures

these were stills and over a third of the image content was lost per frame. So Doug Roble, one of our Senior Software developers, rewrote the software to enable us to handtrack individual unique pieces of the images through the frames and come up with an average OptiFlow motion blur for the sequence, which gave us a cheat on a 3D motion blur."

From those stills, Begun's team created a running footage background, then Gillberg created a new CG camera move over that, which was applied to the CG Talon and the live action footage of the actor in the cockpit set. All of it went through Nuke.

"Nuke actually had a bunch of different roles in this film," Begun says. "Besides being primarily our compositing pipeline, it was used for projections to do all kinds of hookups, and for the final assembly of our environments for every sequence. On ***I, Robot***, we started using Nuke as a shader control for our 3D artists – we call it our 'shampositing' system. For *Stealth*, Nuke is capable of supporting up to 64 discrete channels of information, and we used EXR format, which meant we actually could contain 30 or 40 or 50 different layers of imagery in one online file. So we rendered out the Talons in separate passes, then used Nuke to generate the plane before it was actually given to compositors, because it gave a lot more feedback to the 3D artists in a shorter period of time. We'd actually separate out lighting passes, texture passes, environment passes – all those things were rendered out separately, and then 3D artists had a shamposit tool that was built inside of Nuke, and they can dial that to their liking and get feedback from their supervisors, and that plane was handed off to the compositor at that point, and then we'd put that in the composite, so it ended up being touched by almost every department in some way."

Ultimately, the shots Digital Domain created for *Stealth* may rewrite the rules for creating vertigo on film.

"I just love dive-bombing things in planes and Rob is a speed nut," Hynek concludes. "We both had a good time on these shots."

Ron Magid *contributes regularly to* ***Premiere, American Cinematographer, The Hollywood Reporter, Animation*** *and many other magazines. He has recently written scripts for the* ***Making of King Kong*** *documentary for the upcoming DVD release of the 1933 classic. He has also written episodes of the Discovery Channel's* ***Movie Magic*** *and documentaries for 20th Century Fox and American Movie Classics. He is also a playwright and screenwriter.*

LightWave [8] Beginner Checklist Check your list, *Check it twice!*

Illustration by Alejandro Parrilla, Character Model by Proton

By William "Proton" Vaughan
3D Artist, Author, Teacher
Orlando, Florida

For over ten years, I have been training artists in LightWave. Although everyone comes at it with a different approach, I have seen the same basic questions arise. I've chosen the top twenty most asked questions from online and from the classroom and given them brief answers. I tell my students they should keep a mental checklist of things to look out for when working in LightWave and it has seemed to help. So here we go, the LightWave [8] Beginner Checklist.

1 *My model moves slowly in Modeler when I turn on SubPatches?*

The First thing to check is that your **Patch Divisions**, located in the **General Options** panel, isn't set too high. I suggest working at a SubPatch level of 3. I had a student that had an extreme slow down on their computer, only to find that the SubPatch level was set at 67. That means that each polygon represented 4489 polygons. That's quite a bit overboard in most cases.

2 *I worked all day on surfacing my object in Layout and saved my scene. When I opened it up the next day, all my surfaces were gone. Why did LightWave not save all my work?*

I was guilty of this when I was new to LightWave. Surfaces save to the object file, not the scene file. You must save the objects in Layout in order to save any and all surface changes created in layout.

Note: If you are working on a group project, be careful not to overwrite an object file that someone else is using in another scene. I would suggest saving a new object with a different name so that you don't accidentally change the look of someone else's scene.

3 *I applied Saslite to one of my models, but when I render I get no fur. I have checked to make sure I applied it to the correctly named surfaces – what is going on?*

Make sure that you have also applied the **Saslite Pixel Filter** to the **Add Pixel Filter** list in the **Image Processing** tab of the **Effects** Panel. This is required for rendering Saslite.

4 *After I used Joint Move on a bone, my IK stopped working. I checked all my IK settings and it all seems correct. Why doesn't it work?*

Check and make sure the **Enable IK** is on under the **Setup** tab. Using any of the bone tools turns this off. This is the first place I check when it appears IK isn't working. If **Enable IK** is selected, then the next step is to double-check all of your IK settings in the **Motion Options** panel.

5 *I've written my expressions and applied them to my object, but they don't seem to be working. Did I forget something?*

Make sure **AutoKey** is on. Most expressions require that **AutoKey** be turned on so that a keyframe is made when items are moved, updating the expressions with the information they need to carry out their tasks.

6 *Someone set up openGL sliders on a rig in my scene, but I can't seem to* *select them. How can I select the sliders to adjust them?*

You need to select the **Sliders** tool located under the **Modify** Tab. This is actually a feature that allows you to work in your scene and not accidentally adjust them.

7 *I had the perfect calculations on my dynamic cloth flag, but when I re-opened the scene it was gone and I have to recalculate it every time I open it. Isn't there a better way?*

Of course there is. Simply save the calculations from any dynamic object under the **File** Tab in the **Dynamic Properties** Panel.

Note: You must do this if you plan on rendering out your animation on a render farm.

8 *After reducing polygons on my object, I ended up with several 2-point polygons. Where did they come from?*

Often times, when you weld points on an object, the remaining points from that polygon create 2-point-polygons. For example: If you have a 4-point polygon and you weld two of the points together, you are left with two points that make a 2-point polygon.

To remove the 2-point-polygons, simply select all of them from the **Statistics** (w)window and **Delete** them.

9 *I added a cool new plug-in to Modeler that I found online, but it doesn't show up on the interface. Where did it go?*

If you load a plug-in or Lscript into LightWave and it doesn't appear under the **Additional** dropdown menu, then you need to go to the **Configure Menus** Panel and manually add it to the interface.

10 *When using Morph Mixer on my character, the morphs don't look smooth like they did in Modeler. Any idea of what it could be?*

Make sure you set your **Subdivision Order** to **Last** in the **Object Properties** Panel. Basically, that means that Layout won't freeze your mesh until all other operations finish performing. This can also help with poor deformations when using bones.

11 *I want to create some 3D text, but the Text tool is always grayed out. How do I activate the Text tool?*

Activation of the **Text** tool requires that you load at least one font. Use the **Manage Fonts Tool** button, located below the Text tool. Add as many fonts as you like.

12 *I have bones in my object and they are active, but when I move or rotate them, the object doesn't deform. Did I break LightWave?*

LightWave isn't broken; simply activate the **Enable Deform** button found under the **Setup** Tab.

13 *I decked out my machine with a killer graphics card, but my textures look like crap in openGL. Should I try a different card?*

That would be an expensive approach, but would not solve your problem. Try setting your **Texture Resolution** to a higher setting in the **Display Options** Tab.

14 *When I move about my scene, some of my objects turn into bounding boxes until I stop moving. What is going on?*

This is actually a feature to help with slower machines. If you would like to see your objects when you are moving around in your scene, raise the **Bound Box Threshold** setting found on the **Display Options** Panel.

15 *I'm used to working in 3D Studio Max and I have grown accustomed to having the toolbar on the right side of the screen. Is there a way to move the toolbar?*

Yes. This is a very common question from users that have switched from another 3D package or added LightWave to their toolkit. Under the **General Options** Tab on the **Preferences** Panel, change the **Toolbar Position** from **Left** to **Right**.

16 *I found the perfect metal surface to use on my object from the Preset panel, but when I render my scene, the metal looks nothing like the sample image. What do I need to do?*

Some of the presets, especially reflective metal presets, need either a gradient or image to reflect in order to have it match. Having something other then black as your environment is all that is needed.

17 *My Camera and Lights are way too big in my scene – can I size them down?*

Camera and Light icon size are dependent on the size of your grid. Change the **Grid Square Size** setting in the **Display Option** Tab of the **Preferences** Panel to adjust the size of the icons.

18 *I try loading sample scenes from the Content CDs, but it can't seem to find elements in the scene at load time. Are they missing?*

Be sure to change your **Content Directory** in the **General Options** Tab on the **Preferences** Panel to point at the CD. You will need to change the Content Directory to point at whatever folder you are working from in order for LightWave to know where the elements are located.

19 *I can't seem to multi-select in the Scene Editor Property panel columns. Is this feature broken?*

To multi-select, simply select the far-left side of the property cells you want to adjust. There is a small faint line in each cell, which indicates where to click.

20 *I want to render my images at 300 DPI for print – where is the DPI setting in LightWave?*

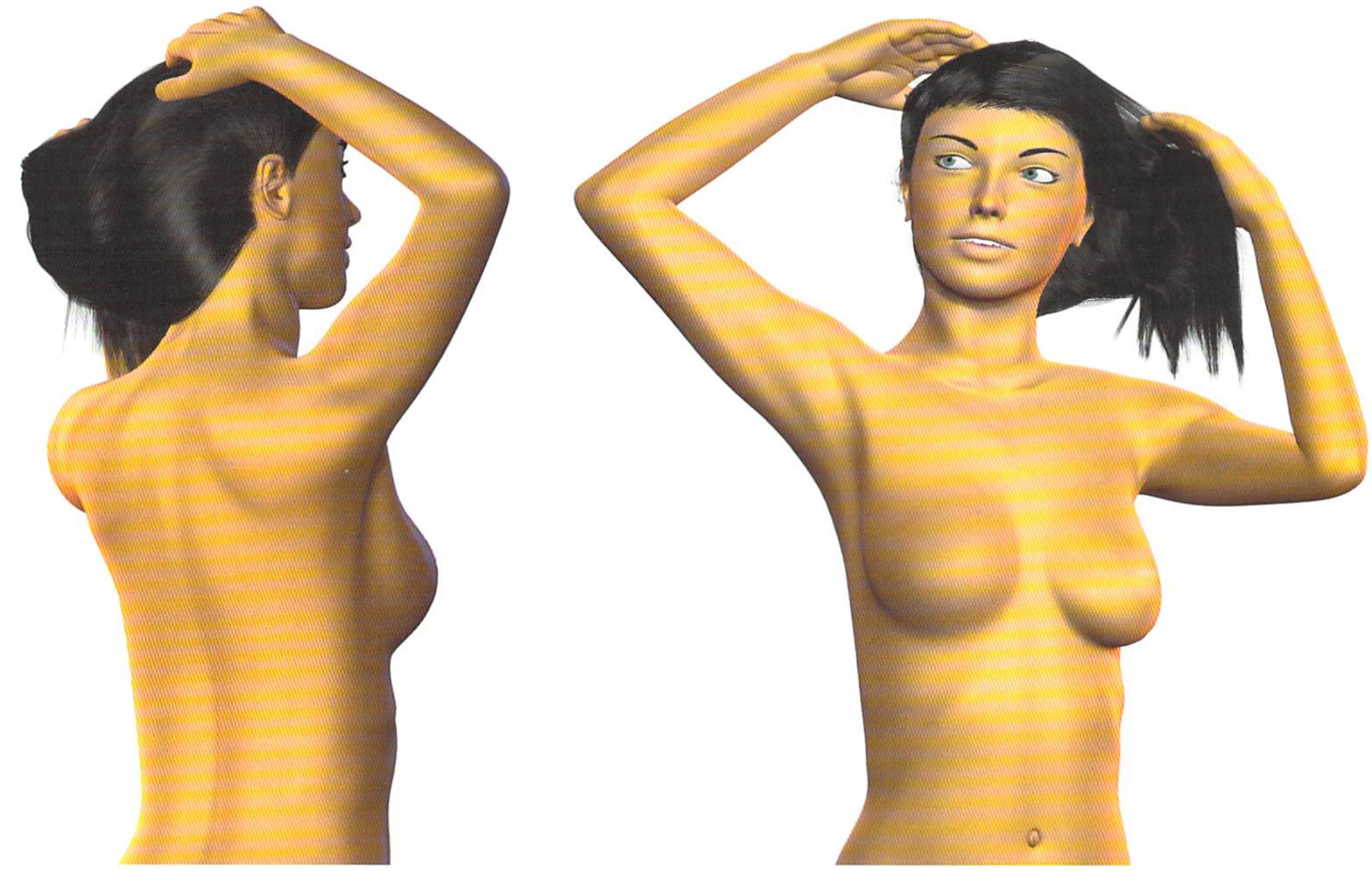
There isn't a DPI setting. LightWave deals in pixels, so it will always render at 72 DPI. It's quite easy to calculate what you should set the camera resolution at by using the **Print Assistant** found under the **Render** Tab.

"What is important is to keep learning, to enjoy challenge, and to tolerate ambiguity. In the end there are no certain answers."
-Martina Horner

William "Proton" Vaughan *is a recipient of several New Media Addy awards. William has an extensive background in creative 3D for print, web, multimedia, games and broadcast. During the last 10 years, he has delivered award-winning work for clients such as Compaq, New Line Cinema and Halliburton. William has also trained artists at several studios and schools around the world and contributed to six LightWave 3D books throughout 2003 and 2004.*

In 2002, Vaughan joined NewTek's marketing team as the LightWave 3D Evangelist, working closely with the LightWave development team, key accounts, and the growing number of end users to enhance LightWave's features set.

William is Director of Industry Relations and Instructor of The DAVE School 's upcoming Final Project. William's focus is on continuously improving the quality of education at The DAVE School, while further establishing the school's presence in the industry.



Binding the Model to the Rig

Part 4 in the series of Preparing a Human Model for Animation. **PART 4**

By Peter Ratner
Professor, Artist
Penn Laird, Virginia.

If you have been following the previous tutorials on rigging in Maya, you should now be ready to bind your model to the completed rig.

Now that you have completed the rigging process, it is time to attach the human model mesh to the skeleton. This will make it possible to deform the mesh with the skeleton. In Maya, there are two methods for binding the model to the skeleton. These are Rigid Skin and Smooth Skin.

Rigid Skinning enables joints to influence specific sets of deformable object points, while Smooth Skinning allows several joints to influence the same vertices, CVs, or lattice points.

In Smooth Skinning, the components, such as the points on a mesh, are weighted for smooth interaction. Since Rigid Skinning allows a vertex to be weighted to only one joint, it provides flexors for controlling mesh distortions.

Unlike Rigid Binding, which only allows one joint to regulate a specific set of points, Smooth Binding enables several joints to influence the same mesh vertices. Since Smooth Skinning does not use flexors to control creasing, many artists use Blend Shapes and/or Cluster Deformers to control unruly deformations at the joints.

If all of this sounds confusing and you are not sure which binding method to use, then consider this. Low polygon gaming characters traditionally use Rigid Binding, while high-resolution cinematic characters use Smooth Binding most of the time.

Since this lesson deals with high-resolution characters, I recommend that you follow the section on Smooth Binding. I'll cover Rigid Binding briefly first because you may need it for other objects, such as a collision mesh that covers the body to prevent clothes from penetrating the skin. Smooth Binding takes longer to calculate than Rigid Binding. Therefore, it is usually a good idea to limit its use to only one or two characters in a scene. Of course, all this depends on your hardware limitations.

I'll use the smooth proxy method in the binding process so that the low polygon model acts like a lattice around the high-res smooth model. When you bind the low polygon proxy model, it will, in turn, bind the smooth high-res version. When the joints deform the low polygon proxy mesh, it will work its way down the hierarchy to the high-res mesh, which will always try to stay smooth.

The eyelashes, eyebrows, eyeballs, teeth, gums, and tongue are separate objects parented to the head joint. These are not bound to the skeleton. Do not delete history on your model if you have Blend Shapes. If you delete history on a model that has Blend Shapes and is bound to a skeleton, it will delete your blend shapes and detach the skin from the rig.

By the time that you are ready to bind a model, you would be done with the modeling aspect of it. Normally, after

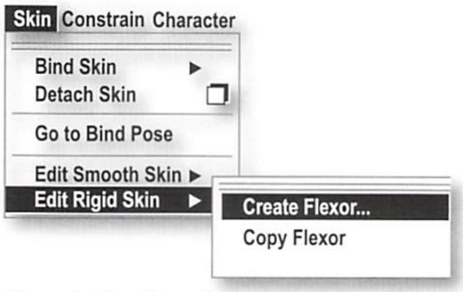


Figure 1– Creating a flexor on a selected joint.

binding a model, you do not make changes to its appearance.

RIGID BINDING A HUMAN MODEL TO THE RIG

In your perspective window, turn on wireframe view. Select the root joint located at the groin. This will select the entire skeleton. Shift-select the low-res human model. In the Animation Menu Set, go to *Skin > Bind Skin > Rigid Bind.*

Your model is currently in the "bind pose." You can experiment moving parts of the model with the various icons, cluster handles, and pole vectors, but be sure to undo everything so that the model is back in its bind pose. You can also select the root joint and choose *Skin > Go to Bind Pose.* This will put your rig back in its original pose. The high-res smooth model should also deform right along with the low poly one.

You probably noticed that the deformations by the joints had problems. We'll fix this next.

ADDING FLEXORS

STEP 1

Select the left knee joint and go to *Skin > Edit Rigid Skin > Create Flexor...* (Figure 1). For Flexor Type choose Lattice. Accept the default settings and click the Create button.

If you want to scale the flexor up or down then open the Outliner window and select both the "Lattice" and the "Base" found inside the "Lattice Group." Use the Scale tool on both of them at once. If you only select the "Lattice" without the "Base," it will affect the mesh too. Figure 2 illustrates the flexor for the left lower leg joint.

STEP 2

Test the flexor by selecting the root control arrow icon and moving it down. The legs should bend at the knee. You can now edit the flexor for more accuracy. Select the flexor and go to its Channel Editor. Next to Creasing, Rounding, Length In, and Length Out, you can change the values until the creasing at the knee joint looks the most natural. An easy way to do this is to select the name, such as Creasing in the Channel Editor, and then middle-click/drag in the perspective view to change the way the flexor affects the mesh.

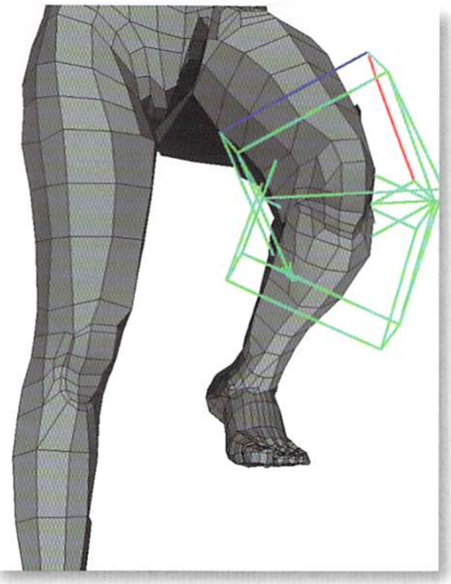


Figure 2– Adding a flexor to the lower leg joint and adjusting its values after bending the leg.

Repeat this process with the rest of the Channel Attributes. After you are done, select the root joint of the skeleton and choose *Skin > Go to Bind Pose* to put the skeleton back to its neutral position.

HIP AND SHOULDER PROBLEMS

The ball joints at the hips and shoulders behave differently than hinge joints at the knees and elbows. They have a greater degree of freedom and therefore often cause problems for animators. Lattice flexors usually work fine for hinge joints, but the ball joints often require a different solution.

STEP 1

At the hips and the armpits, create joint cluster flexors. This option appears in the *Animation > Edit Rigid Skin >Create Flexor...* Next to Flexor Type, select "jointCluster" from the dropdown menu. Select the joint cluster in the Outliner window. Move it behind the body so it will be easier to select. The icon is a "J."

STEP 2

For adjusting the hip area, position the body in a squatting pose. While the "J" (jointCluster) is selected, click the Show Manipulator tool and also look in the Channel Box under Inputs. Click on the name of the joint cluster. Its name might be something like "l_hipCluster1." Experiment with different values and observe the changes to your model. With the Show Manipulator tool you can also click and drag on the joint cluster shapes, which in turn will affect the mesh around it (Figure 3).

STEP 3

Use the same technique for adjusting the joint cluster in the armpit area. Position

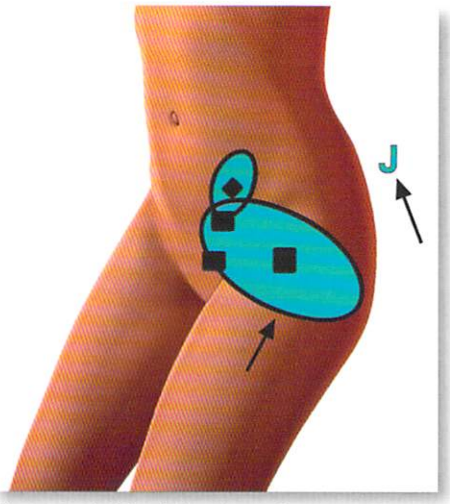


Figure 3– Adding joint cluster flexors at the hip (J icon) and modifying each with the Show Manipulator tool.

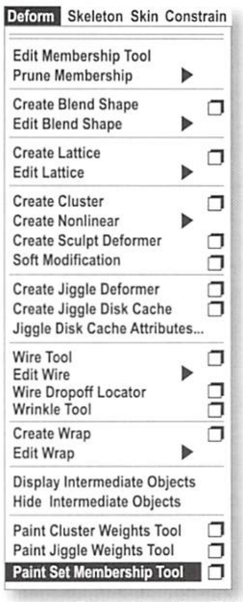


Figure 4– Selecting the Paint Set Membership tool.

the arm in various poses and then correct the settings for the joint cluster. Chances are that you will still not be totally satisfied with the deformations at the hips and shoulders. The Paint Membership tool gives you additional help to solve these problems; we'll discuss this next.

SET MEMBERSHIP

Often, when Rigid Binding a model, certain points are mistakenly assigned to the wrong bones. Reset those points to the correct corresponding bones with the Set Membership tool.

STEP 1

Hide the high-res model and select the low-res proxy mesh. Go to *Deform > Paint Set Membership Tool > Options* (Figure 4). Each joint should now exhibit a corresponding color in your perspective

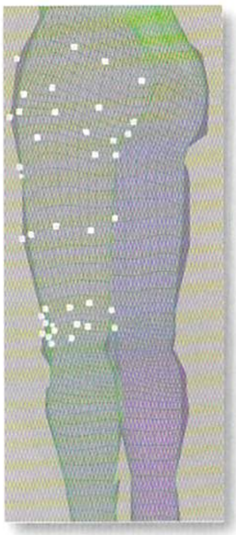


Figure 5– Using the Set Membership tool to assign vertices to specific joints.

view (Figure 5). In the Paint Set Membership Tool option box (Figure 6), under the Select Set To Modify heading, select the joint that has undesirable deformations around it. Set your Brush Size by pressing the "b" key and dragging in your view window.

In the Paint Set Membership Tool option box, under Paint Operations, use Add, Transfer, and Remove to modify which points belong to specific joints. Refer to a previous lesson on Rigid Binding or the Maya manual for a more detailed explanation of the Paint Set Membership Tool.

STEP 2

Continue creating flexors at the joints that exhibit unusual distortions. Follow the previous step to edit each flexor.

SMOOTH BINDING A HUMAN MODEL TO THE RIG

Unlike Rigid Binding, which only allows one joint to regulate a specific set of points, Smooth Binding enables several joints to influence the same mesh vertices. Since Smooth Skinning does not use flexors to control creasing at the joints, we will use Blend Shapes to manipulate these areas.

STEP 1 - SMOOTH BINDING THE MESH

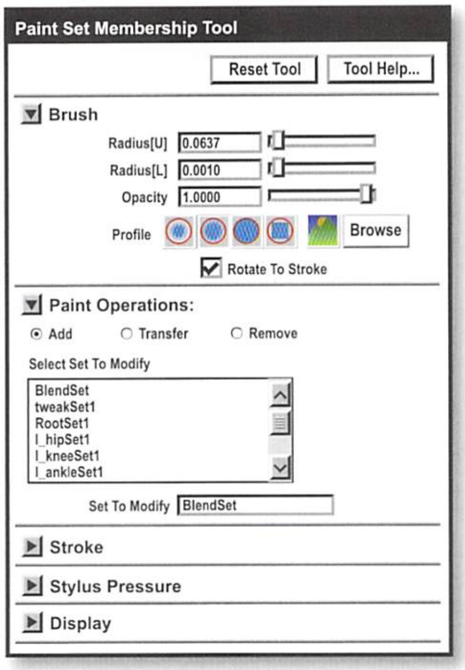
Select the root joint of the skeleton and Shift-select the mesh of your model. Go to *Animation > Skin > Bind Skin > Smooth Bind > Options*. You now have several choices for binding the skin.

BIND TO

Select the option to bind the entire skeleton. In some cases, you may decide to bind only to selected joints.

BIND METHOD

Closest Joint is the preferred method, because it bases the joint influence on the


Paint Set Membership Tool

Reset Tool | Tool Help...

Brush
Radius(U) 0.0637
Radius(L) 0.0010
Opacity 1.0000
Profile — Browse
[✓] Rotate To Stroke

Paint Operations:
[✓ Add] Transfer Remove

Select Set To Modify
BlendSet
tweakSet1
RootSet1
l_hipSet1
l_kneeSet1
l_ankleSet1

Set To Modify BlendSet

Stroke
Stylus Pressure
Display

Figure 6– The Paint Set Membership tool options box.

skeleton's hierarchy. If you decide to use Closest Distance, the software will only evaluate the vertices that are closest to the joint and ignore the skeleton's hierarchy.

MAX INFLUENCES

This allows you to limit the number of joints that can influence nearby skin points. The default setting of five is a good amount to use.

DROP-OFF RATE

This setting controls how rapidly each joint's influence on the mesh decreases with the distance from each joint. Larger numbers create a more rapid decrease in influence with distance. If you want the joints to influence points further from them, you would use smaller numerical settings. The default setting of four works fine for most characters.

The Paint Skin Weights tool works best for regulating the influence of joints when Smooth Binding.

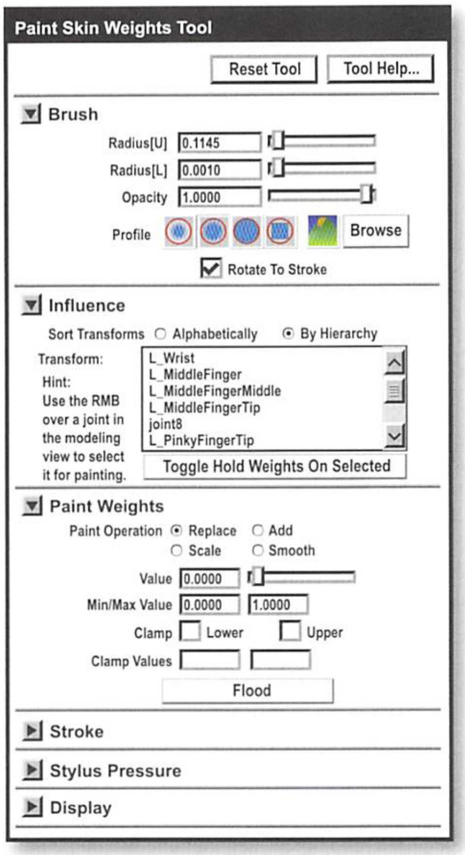
After binding skin, you can use the Paint Skin Weights Tool to edit the influence of joints in an intuitive manner.

STEP 2 USING THE PAINT SKIN WEIGHTS TOOL

The Paint Skin Weights Tool is an intuitive way of adjusting the weights of model's vertices. With this tool, you can quickly change the way joints act upon the mesh in certain areas.

STEP 1

Select the character and go to *Animate > Skin > Edit Smooth Skin > Paint Skin Weights Tool > Option*. The option box looks somewhat like the one in Figure 7. If you have


Paint Skin Weights Tool

Reset Tool | Tool Help...

Brush
Radius(U) 0.1145
Radius(L) 0.0010
Opacity 1.0000
Profile — Browse
[✓] Rotate To Stroke

Influence
Sort Transforms Alphabetically [✓ By Hierarchy]
Transform:
L_Wrist
L_MiddleFinger
L_MiddleFingerMiddle
L_MiddleFingerTip
joint8
L_PinkyFingerTip

Hint: Use the RMB over a joint in the modeling view to select it for painting.

Toggle Hold Weights On Selected

Paint Weights
Paint Operation [✓ Replace] Add Scale Smooth
Value 0.0000
Min/Max Value 0.0000 1.0000
Clamp [ ] Lower [ ] Upper
Clamp Values
Flood

Stroke
Stylus Pressure
Display

Figure 7– The Paint Skin Weights tool options box.

Smooth Shade on, you should see the model mostly black with some area displayed white and gray (Figure 8). Turn on the Wireframe on Shaded option in your view window *(Shading > Shade Options > Wireframe on Shaded).*

STEP 2

Adjust the size of your brush by pressing down on the "b" key and left mouse click-drag. In the Paint Skin Weights Tool option box, under the Influence heading, select a joint. You should see the mesh around that joint highlighted in white with gray around the edges where the influence drops off. The lighter the area, the more influence is exerted by that joint. The black areas mean that there is no influence on the mesh by that joint.

STEP 3

Switch back to your Selection tool and select one of the arm control icons. Move it up and observe the way the arms move and deform the mesh. If you see any kind of unwanted pulling on parts of the model, such as the torso, then select your model again and switch back to the Paint Skin Weights Tool (the keyboard shortcut is "y"). Under the Influence heading, select the upper arm joint. Under the Paint Weights heading, next to Paint Operation, select Replace and set the Value to 0. Under the Brush heading, set the Opacity to one. Paint the areas of the torso where you do not want any influence

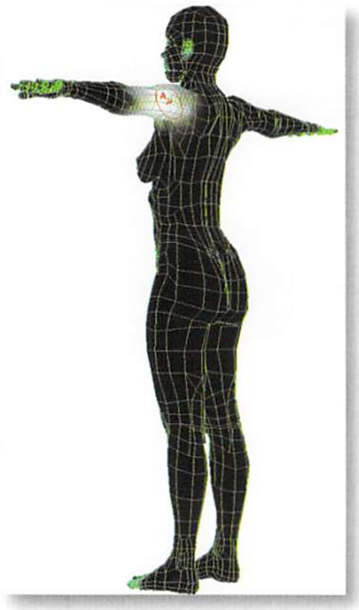
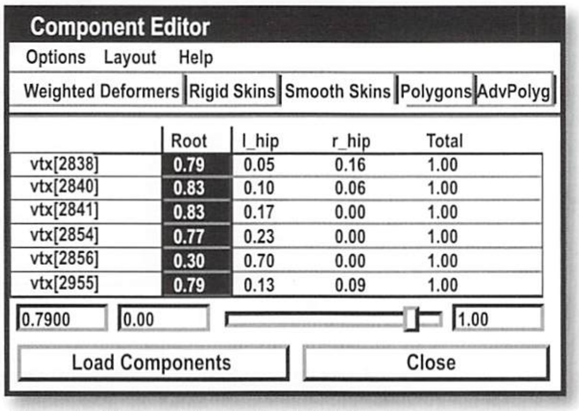


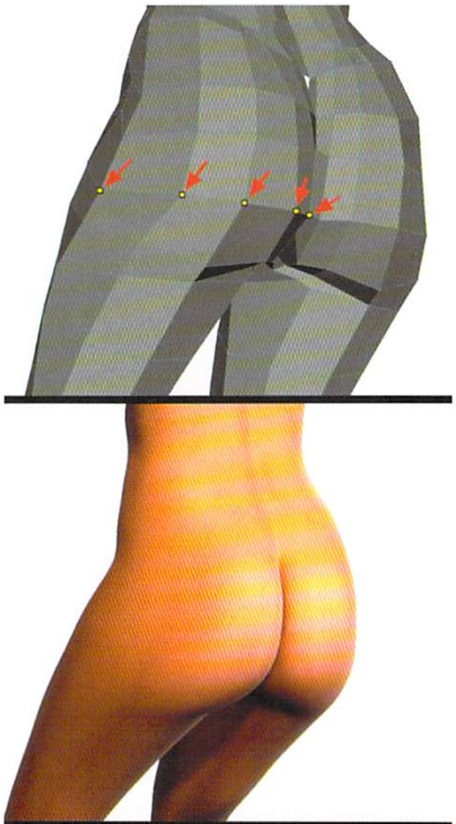
Figure 8– Adjusting the skin weights with the Paint Skin Weights tool.

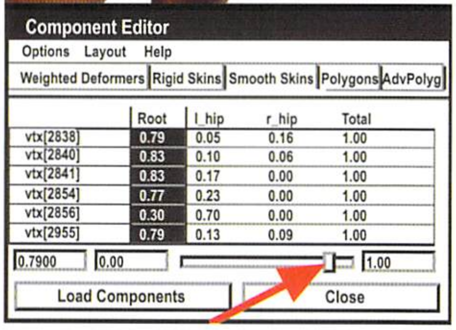

Component Editor
Options Layout Help
Weighted Deformers Rigid Skins Smooth Skins Polygons AdvPolyg
Load Components
Close

	Root	l_hip	r_hip	Total
vtx[2838]	0.79	0.05	0.16	1.00
vtx[2840]	0.83	0.10	0.06	1.00
vtx[2841]	0.83	0.17	0.00	1.00
vtx[2854]	0.77	0.23	0.00	1.00
vtx[2856]	0.30	0.70	0.00	1.00
vtx[2955]	0.79	0.13	0.09	1.00

0.7900 0.00 1.00

Figure 9– The weights for the left buttock vertices in the Component Editor.




Component Editor
Options Layout Help
Weighted Deformers Rigid Skins Smooth Skins Polygons AdvPolyg
Load Components
Close

	Root	l_hip	r_hip	Total
vtx[2838]	0.79	0.05	0.16	1.00
vtx[2840]	0.83	0.10	0.06	1.00
vtx[2841]	0.83	0.17	0.00	1.00
vtx[2854]	0.77	0.23	0.00	1.00
vtx[2856]	0.30	0.70	0.00	1.00
vtx[2955]	0.79	0.13	0.09	1.00

0.7900 0.00 1.00

Figure 10– Using the interactive slider to adjust the weights of the selected buttock vertices.

by the upper arm joint. (It should painted it black.) Turn down the Opacity setting if you want to paint grayer. You can add more white (greater weight) to the points by selecting Add next to Paint Operation and changing the value to one.

Use the same previously discussed Paint Tool keyboard shortcuts to quickly work with the Paint Skin Weights Tool. With the following shortcuts you will no longer have the Paint Skin Weights Tool option box taking up valuable interface space. These are the same options as discussed with the Paint Cluster Weights Tool.

Press the "u" key down and left click to select the paint choices.

Hold the "n" key down and left click to select the value of the Replace, Scale, or Add options.

Left-click over the area that has the joint whose weights you want to paint and select Paint Weights from the marking menu.

STEP 4

Continue adjusting the various joint influences. Switch back and forth between the Paint Skin Weights tool and your Selection tool. Move different parts of your character and correct the weights until it deforms the way you want it to. When you adjust the weights on joints in the center of the body such as the spine, neck, and head, you can turn on Reflection, found under the Stroke heading. This will duplicate the brush so that both sides of the body are painted.

When you need to have very precise control over the weights of specific vertices, then switch to your Selection tool, right-click, and select Vertex. Click on the vertices that you want to either remove influence from or add to. Select the Paint Skin Weights Tool again and paint directly on the selected vertices. The Paint tool only affects the selected points.

The following choices can also be found in the Paint Skin Weights Tool option box.

Replace– This replaces the skin weight with the weight set for the brush.
Add– You can increase the influence of nearby joints with this option.
Scale– This option diminishes the power of far away joints.
Smooth– If you want to blend any rough transitions between weights then use this option.
Clamp– Allows you to hold certain settings for specified weights so that you never go below or above a particular minimum and maximum range.
Flood– This adds day and age settings to all the skin weights for the currently selected joint. For example, if you have the Replace value set to zero and you click Flood, it will put a zero (no weights) influence on that part of the mesh. The selected joint will no longer have any effect on this part of the model.

Painting weights is a good first step for generally making most of the weight maps. It really helps to have a pressure sensitive tablet for this kind of task because it allows you much greater control than using a mouse. For example, you can press down harder to apply a greater amount of positive or negative weight.

Painting weights will only get you so far. In order to exercise pinpoint control over the weights of each vertex you need to use the Component Editor. The following steps will show you how to work with it to basically "sculpt weights."

USING THE COMPONENT EDITOR TO FIX WEIGHTS

After painting all the weight maps, it is time now to refine the weights. Many people, especially the ones without a pressure sensitive tablet, will often skip the paint weights steps and just use the Component Editor to correct all their weight maps.

STEP 1

Bend the legs of the body at Frame 6 and keyframe it there temporarily. You can move the Root Control down to do this. Make sure you keyframe the Root Control at Frame 0 first so you can put it back in its original upright position. The idea is to scrub back and forth on the Time Slider to test the weights as the body changes from its upright position to the bent pose.

STEP 2

Select a row of vertices at the specific joint such as the left buttocks. Go to *Window > General Editors > Component Editor...* Click the Smooth Skins tab. Now you can scroll until you see the various weights. The left column lists the selected vertices. (Figure 9)

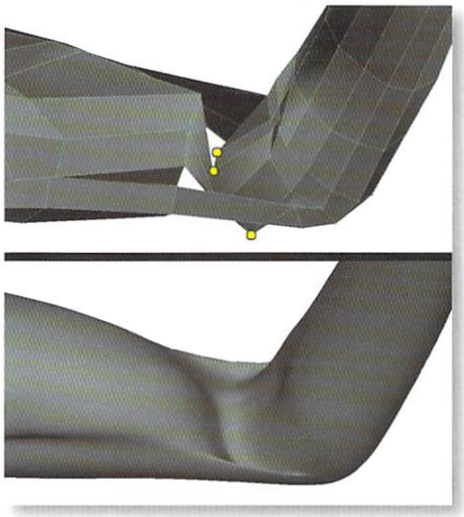
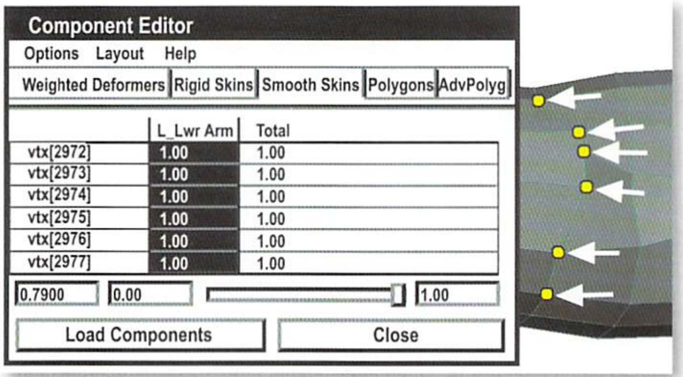


Figure 11– A creasing problem caused by a group of vertices that only have weights from the lower arm joint.



	L_Lwr Arm	Total
vtx[2972]	1.00	1.00
vtx[2973]	1.00	1.00
vtx[2974]	1.00	1.00
vtx[2975]	1.00	1.00
vtx[2976]	1.00	1.00
vtx[2977]	1.00	1.00

0.7900 0.00 1.00

Figure 12– Step 1. Opening the Component Editor after selecting the vertices that are causing the distorted creasing. This reveals the problem, which is that the points do not have the left upper arm joint, assigned to them along with the lower arm joint.

STEP 3

In the Component Editor, select a row that has weight map values such as the one under the Root heading (Figure 9). Use the interactive slider found at the bottom of the Component Editor to move the vertices until the deformation looks better.

Hide the low poly and show the smooth version so that you can see the effect it has. In fact, after selecting the vertices on the low poly model, you can hide the layer that contains it and just show the layer with the smooth proxy version. Moving the interactive slider in the Component Editor will let you see immediately how the appearance of this section on the buttock changes.

Figure 10 (previous page) shows this process. The top illustration depicts the first step, which is to select the vertices on the low poly model. The middle part of the picture illustrates the smooth proxy version after hiding the low proxy one. The bottom of Figure 10 shows how the interactive slider moves until the smooth proxy (middle section of the illustration) looks correct.

STEP 4

Sometimes you run into the problem in which a certain vertex or group of vertices are missing the desired joint's weight on them. For example, when you bend the lower arm a group of points create an undesirable crease in it (Figure 11). The following steps will show you how to add these through the Component Editor.

STEP 1 ADDING WEIGHTS

Select the group of vertices that you want to fix and open the Component Editor. Figure 12 shows that this particular group of points only have the L_Lwr Arm joint assigned to them. These will need to have the L_Upr Arm joint reassigned to them so that there are weights for both the lower and upper arm joints.

STEP 2 ADDING WEIGHTS

While the Component Editor is still open and the vertices whose weights will have to be reassigned to the upper arm joint are selected, find another vertex that has the left upper arm joint assigned to it. Shift-select this new vertex. The Component Editor will update and show a new row under the heading of L_Upr Arm (Figure 13).

STEP 3 ADDING WEIGHTS

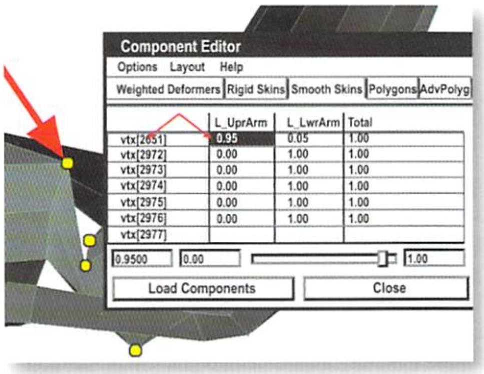
Under the L_Upr Arm column, Shift-select the zero values for the vertices that you need to fix. They are right underneath the one extra point that you selected in the previous step. Figure 14 illustrates the Component Editor with the selected points under the L_Upr Arm heading.

STEP 4 ADDING WEIGHTS

With the zero weights selected under the L_Upr Arm heading in the Component Editor (Figure 14), move the interactive slider until you see weights being added to the vertices. Observe the smooth proxy model as you move the slider until the crease inside the elbow area starts to look more normal (Figure 15, facing page).

CONCLUSION

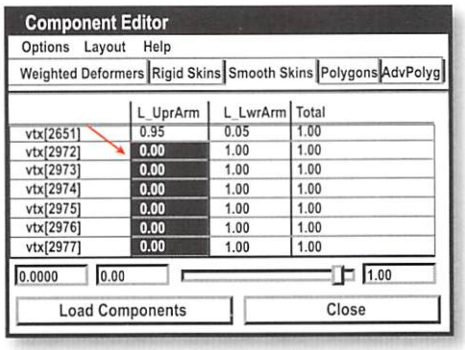
Working with weight maps can be a time consuming process. If you unbind your character or delete its history, you will lose



	L_UprArm	L_LwrArm	Total
vtx[2651]	0.95	0.05	1.00
vtx[2972]	0.00	1.00	1.00
vtx[2973]	0.00	1.00	1.00
vtx[2974]	0.00	1.00	1.00
vtx[2975]	0.00	1.00	1.00
vtx[2976]	0.00	1.00	1.00
vtx[2977]			

0.9500 0.00 1.00

Figure 13– Step 2. Shift selecting a new vertex (red arrow) that has the desired joint assigned to it. The Component Editor updates to show a new column with the L_Upr Arm joint.

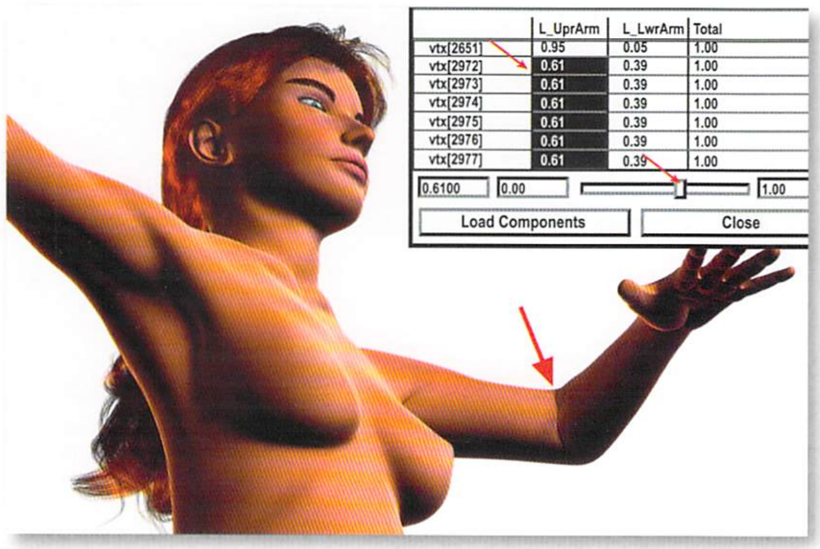


	L_UprArm	L_LwrArm	Total
vtx[2651]	0.95	0.05	1.00
vtx[2972]	0.00	1.00	1.00
vtx[2973]	0.00	1.00	1.00
vtx[2974]	0.00	1.00	1.00
vtx[2975]	0.00	1.00	1.00
vtx[2976]	0.00	1.00	1.00
vtx[2977]	0.00	1.00	1.00

0.0000 0.00 1.00

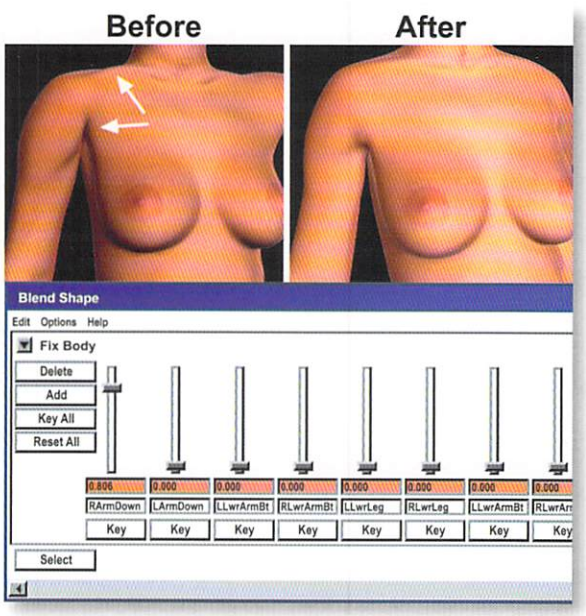
Figure 14– Step 3. Shift selecting the values of the vertices that need to be reassigned. Currently, they have zero values that will be changed with the interactive slider.

all your hard work. Therefore, it is a good idea to export the weight maps and save them as a backup. Once you are finished adjusting the weight maps select the low proxy model and go to *Animation > Skin > Edit Smooth Skin > Export Skin Weight Maps > Options*. Make the map size 2000 x 2000 and the image format JPEG.



	L_UprArm	L_LwrArm	Total
vtx[2651]	0.95	0.05	1.00
vtx[2972]	0.61	0.39	1.00
vtx[2973]	0.61	0.39	1.00
vtx[2974]	0.61	0.39	1.00
vtx[2975]	0.61	0.39	1.00
vtx[2976]	0.61	0.39	1.00
vtx[2977]	0.61	0.39	1.00

Figure 15– Step 4. Moving the interactive slider while observing the smooth proxy version of the model. The weights are reassigned so that both the upper and lower arm joints share weights on the selected vertices.



0.806	0.000	0.000	0.000	0.000	0.000	0.000	0.000
RArmDown	LArmDown	LLwrArmBt	RLwrArmBt	LLwrLeg	RLwrLeg	LLwrArmBt	RLwrAr
Key	Key	Key	Key	Key	Key	Key	Key

Figure 16– The Blend Shape "RArmDown" activates automatically with Set Driven Keys when the upper arm joint rotates down. The figure on the left does not have a Blend Shape.

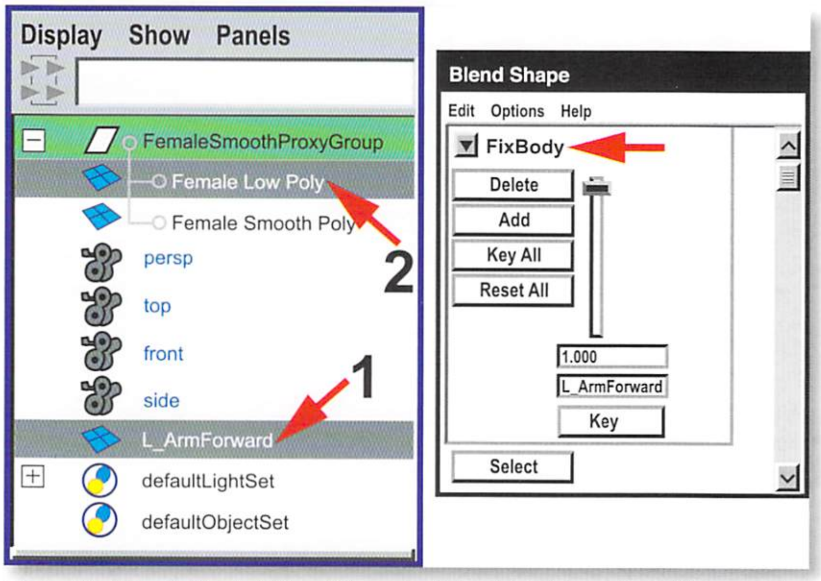
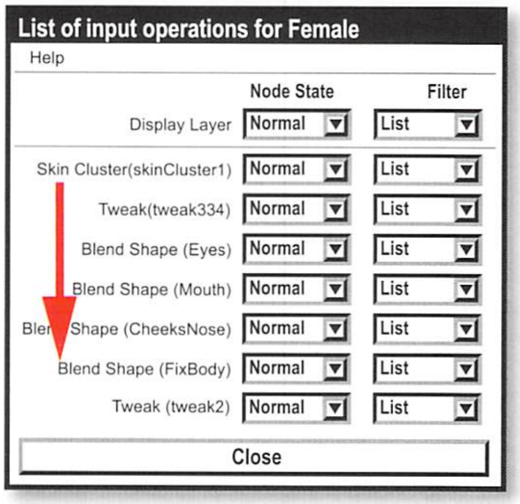


Figure 17– Step 1. The first Blend Shape for fixing joint deformations. The neutral body is imported, renamed, and selected. The low poly proxy model is then Shift selected and a new Blend Shape node named "FixBody" is created. Note, the slider is moved all the way up.



	Node State	Filter
Display Layer	Normal	List
Skin Cluster(skinCluster1)	Normal	List
Tweak(tweak334)	Normal	List
Blend Shape (Eyes)	Normal	List
Blend Shape (Mouth)	Normal	List
Blend Shape (CheeksNose)	Normal	List
Blend Shape (FixBody)	Normal	List
Tweak (tweak2)	Normal	List

Figure 18– Step 2. Right clicking on the low poly model and selecting Inputs > All Inputs... The new Blend Shape node (FixBody) is dragged with the middle mouse button below the Skin Cluster node.

USING BLEND SHAPES TO FIX DEFORMATIONS

Working with weight maps can fix the majority of aberrations on the mesh when joints rotate. Unfortunately they cannot fix all the problems that occur. This is when you would use Blend Shapes or Cluster Deformers to correct unwanted creasing and distortions. Set Driven Keys activate both Blend Shapes and Cluster deformers, repairing problems automatically when they occur.

Figure 16 illustrates the use of Blend Shapes as a remedy to a common problem, which often shows up when the extended arm that you modeled in the T-Pose rotates down. The figure on the left does not have Blend Shapes for fixing the body while the one on the right has a group of them that activate when a specific joint rotates along a certain axis. One can see the difference in the appearance of the underarm.

The following steps show how to set up a Blend Shape to control undesirable deformations at the joints. Later on you can follow the steps to set up Cluster Deformers for accomplishing the same goals. You can use either method or a combination of both.

STEP 1

Import the neutral body; name it the blend shape you want. In this case we will make it "L_ArmForward." Select this Blend Shape target first in the Outliner and then Shift-select the low poly proxy body. Go to *Animate > Deform > Create > Blend Shape > Options*. Name the new blend shape node "Fix Body." Bring up the Blend Shape box and move the new "L_ArmForward" slider all the way up (Figure 17).

STEP 2

Select the low poly body and right click to select *Inputs > All Inputs...* Middle-click/drag the Blend Shape that is on top of the stack (the latest one made) and drag it to the bottom just above the last item in the list named Tweak (Figure 18). This is important because the order of the stack should have the "SkinCluster (skin Cluster1)" at the top. SkinCluster refers to the smooth binding. Anytime you create

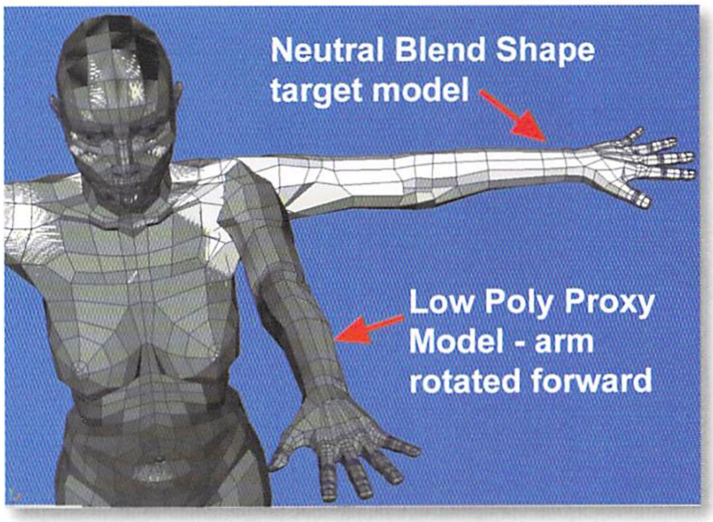

Neutral Blend Shape target model
Low Poly Proxy Model - arm rotated forward

Figure 19– Step 4. Rotating the left upper arm joint forward. The neutral pose Blend Shape target remains untouched at this point.

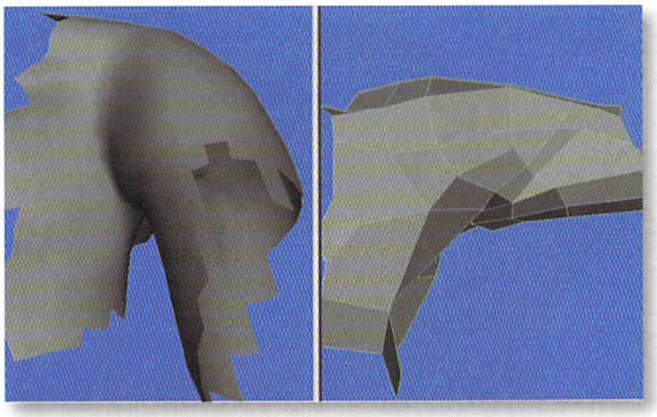
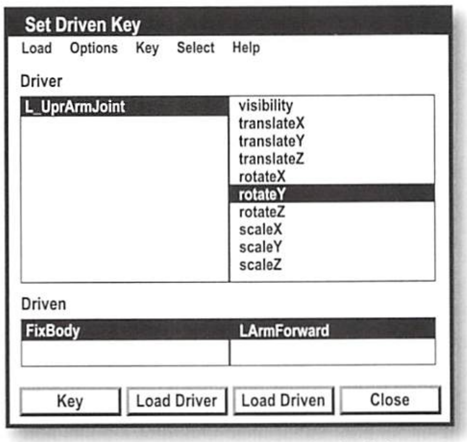


Figure 20– Step 5. Creating two separate perspectives views makes it easier to model the Blend Shape target on the right while viewing the changes in the left view that has the smooth proxy version. Both views have Isolate Select on.



Driver

L_UprArmJoint	visibility
	translateX
	translateY
	translateZ
	rotateX
	rotateY
	rotateZ
	scaleX
	scaleY
	scaleZ

Driven

FixBody	LArmForward

Figure 21– Step 7. Loading in the left upper arm joint as the driver and the left arm forward Blend Shape as the driven in Set Driven Keys.

a new Blend Shape node you should follow this step to change the order of the Input stack. If you are just adding to an existing Blend Shape node then this step is not necessary.

STEP 3

Place the neutral blend shape that you named "L_ArmForward" in its own layer. Make the low proxy and smooth proxy models reference objects in their layers so that you don't by mistake move any of their vertices. From now on you will only move the vertices of the blend shape target named "L_ArmForward." With the Shape Slider all the way up you will see a change in the low polygon proxy model each time you move vertices on the Blend Shape target mesh.

STEP 4

Go to *Modify > Evaluate Nodes > Ignore All.* This will turn off IK and let you rotate individual joints manually. Select the left upper arm joint and keyframe its neutral rotation axes at Frame 0 (Shift e). Move the Time Slider to Frame 6 and rotate the left upper arm joint forward on the Y-axis so that the arm points toward the front and keyframe it at Frame 6 (Figure 19).

STEP 5

On the Blend Shape target named "L_ArmForward," move vertices around the shoulder and armpit region to fix the deformation of the proxy model. The actual low poly model should still be in its own layer as a reference or template so that you do not by mistake move any of its vertices.

Continue finding corresponding vertices on the Blend Shape to correct the deformation on the proxy model. It is an indirect way to model because you are not directly moving vertices on the proxy model.

When you find the vertices on the Blend Shape target that you want to move, hide the layer with the Blend Shape and show either the low or smooth proxy while you move the vertices of the hidden Blend Shape target. The entire axis will be off so it takes some patience to model this way. The Blend Shape target will look odd, but the actual model will look fine.

Here is a method for making things easier. Create two perspective views side by side. In one of your view windows go to *Panels > Layouts > Two Panes Side By Side.* On the left view window pick Blend Shape. On the right view window select *Panels > Perspective > New.* You can now view your object(s) in two perspective windows and rotate the views independent of each other. You can save this layout by going to one of the view windows menu *Panel > Saved Layouts > Edit Layouts...* Click the New Layout button and type a different name such as "2 Perspectives." Close the box and go to *Panel > Saved Layouts* and you will see your saved layout listed.

Hide your Blend Shape target and show only the smooth proxy version with an Exponential Level of 1. In the left perspective panel select only the faces that you are trying to fix in the shoulder area. Go to *Show > Isolate Select > View Selected.* Now you only see the shoulder area of the smooth model in the left perspective window.

Make the layer that has the Blend Shape target visible. Go to the right perspective view window and select only the faces on the Blend Shape target that you want to move to fix the shoulder area. Continuing in the right view window, go to *Show > Isolate Select > View Selected.* Now the right view will only show the faces of the Blend Shape target.

Push and pull points in the right view only and watch the smooth proxy version in the left view update (Figure 20). Steven Stahlberg, a well-known 3D artist, has used Maya since its first release. Here is what he has to say about making these types of Blend Shapes.

"It's easier if you use the expand-selection and shrink-selection hotkeys to pick and unpick vertices that are kind of tucked away. Also create a new persp window just for focusing and zooming close to the verts you're working on, as you look at the result in another window."

Many Blend Shapes active at the same time can be confusing, but by pulling each slider one by one you can tell which Blend Shape is the most offending one, then start by editing that. Blend Shapes on the same joint shouldn't affect each other too much; it's best to make each have a separate "domain" of the movement so to speak.

STEP 6

When you are satisfied with the look of the armpit and shoulder region on the low poly and smooth proxy model, then move the Time Slider back to Frame 0 so that the left arm is back to its original T-pose (arms outstretched to the sides).

Select the Blend Shape target and

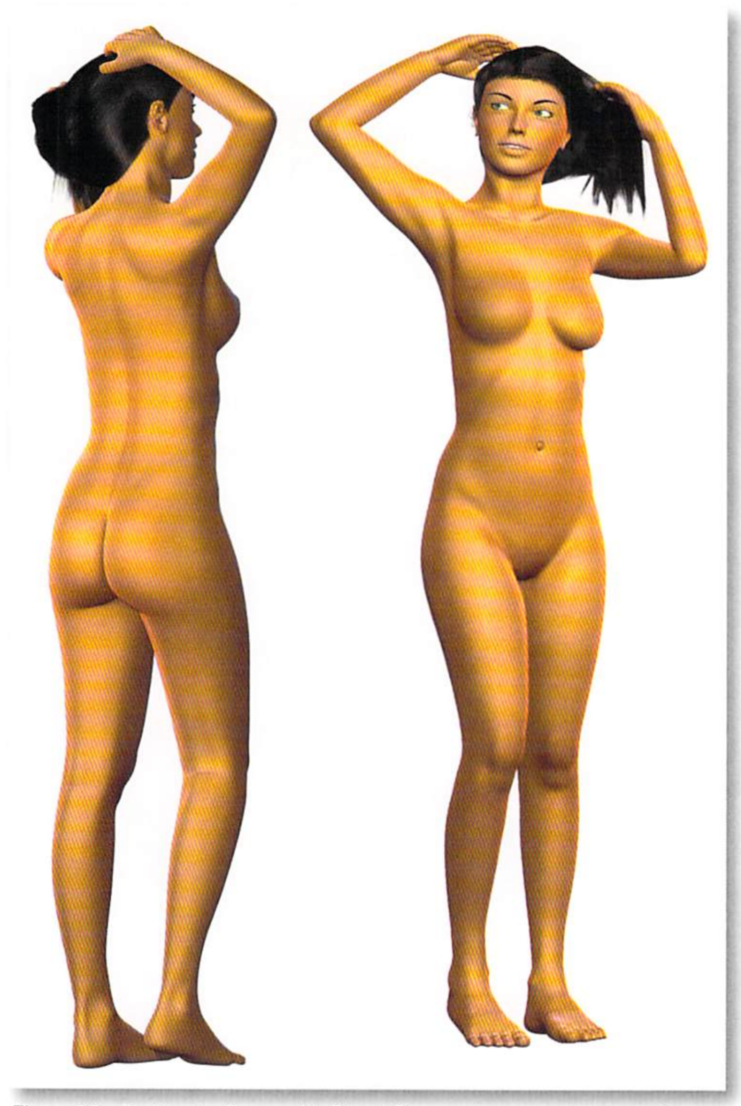


Figure 22– Posing the figure with Blend Shapes that control deformations at the joints.

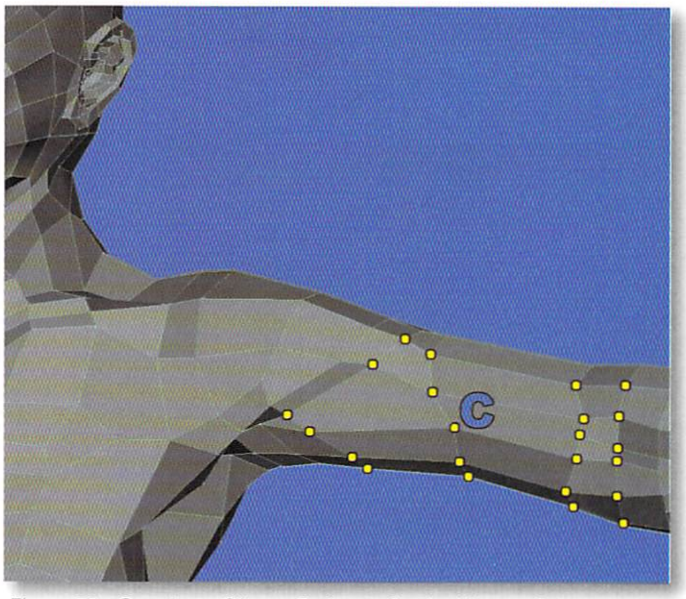


Figure 23– Creating a Cluster Deformer for the biceps of the upper arm vertices.

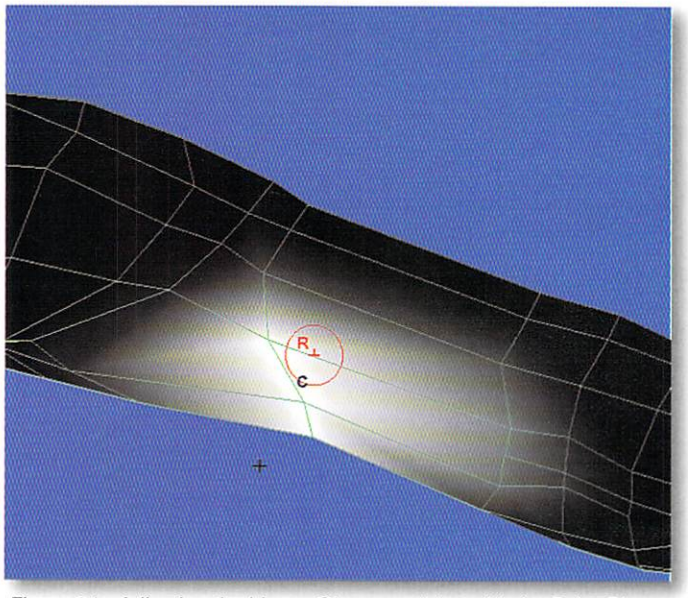


Figure 24– Adjusting the biceps Cluster weights with the Paint Cluster Weights tool.

export it as an .mb or .obj file with the name "L_ArmForward." This is a backup in case you ever need to apply it again in the future. You can now delete the "L_ArmForward" Blend Shape target.

STEP 7

Open Set Driven key box and load the lower arm bone as the driver and the "FixBody" Blend Shape node as the Driven. Select Y-axis for the Driver and "L_ArmForward" for the Driven (Figure 21). Move the Blend Shape slider that controls the "L_ArmForward" Blend Shape all the way down to zero.

STEP 8

Modify > Evaluate Nodes > Ignore All should still be in effect as well as the forward rotation of the left upper arm joint. Move the time slider to Frame 0 - so that the arm is still in its neutral T-Pose. Press the Key button in the Set Driven Key box to keyframe the Blend Shape at the zero position.

STEP 9

Move the time slider to Frame 6 where the arm is totally bent forward. Move the Blend Shape slider that controls the left upper arm forward deformation ("L_Arm Forward") all the way up to 1. Press the key button in the Set Driven Key box.

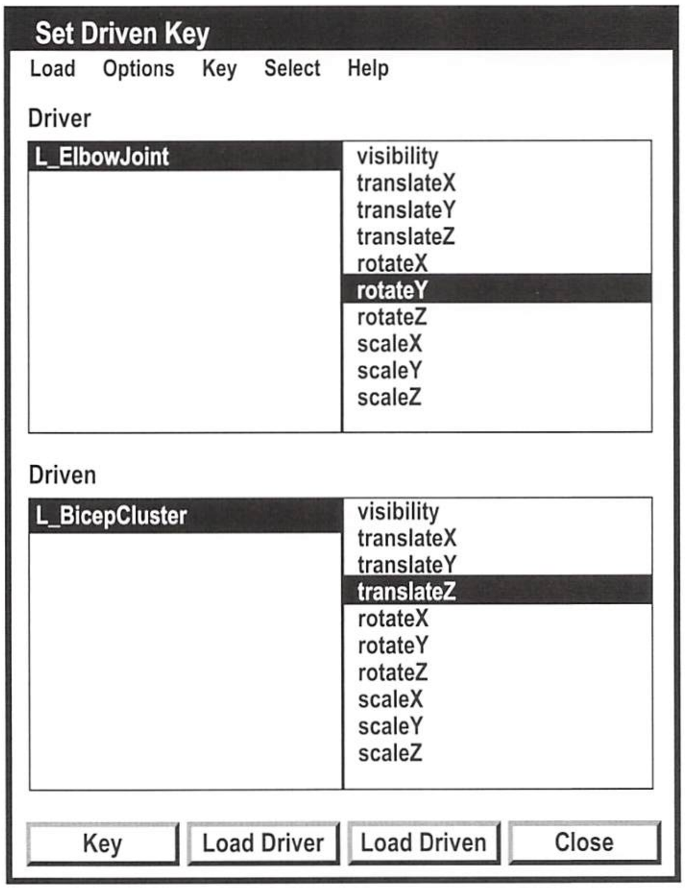
Continue making all the rest of the Blend Shapes to control the various joint deformations. Just follow the previous steps 1-9. Under Step 1, instead of going to *Animation > Deform > Create Blend Shape*, go to *Animation > Deform > Edit Blend Shape > Add > Options*. Be sure to specify the "FixBody" node. Figure 22 shows a figure posed in which a number of these Blend Shapes automatically go into effect to moderate the mesh when it is deformed.

MAKING MUSCLE DEFORMATIONS WITH SET DRIVEN KEYS DRIVING CLUSTER DEFORMERS

You can use Cluster Deformers with Set Driven Keys in a similar way to Blend Shapes. Clusters that become active when specific joints rotate control muscle deformations and creasing at the joints. If you prefer to use Blend Shapes for this type of task, there is no need for you to read any further.

STEP 1

Select the vertices on one of the upper arms that will flex the biceps (Figure 23). Go to *Animation > Deform > Create Cluster*. If you want to have an easier time selecting the Cluster, then



Set Driven Key

Load Options Key Select Help

Driver

L_ElbowJoint	visibility, translateX, translateY, translateZ, rotateX, **rotateY**, rotateZ, scaleX, scaleY, scaleZ

Driven

L_BicepCluster	visibility, translateX, translateY, **translateZ**, rotateX, rotateY, rotateZ, scaleX, scaleY, scaleZ

Key | Load Driver | Load Driven | Close

Figure 25– The elbow joint drives the upper arm muscle Cluster Deformer, making it flex.

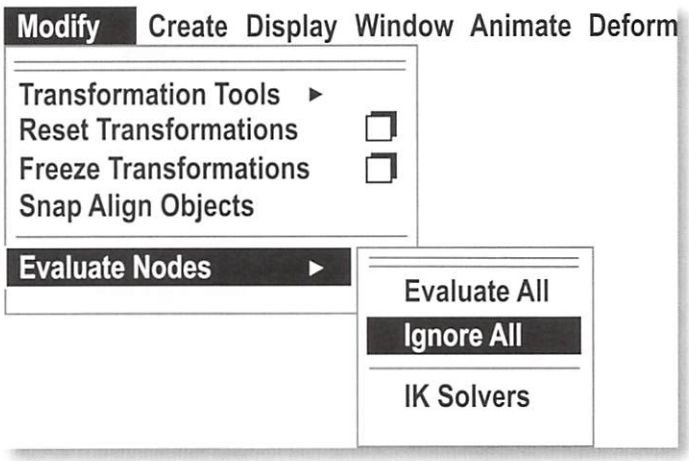


Figure 26– Turning off the evaluation of nodes allows you to move the controllers without affecting the rig.

go to *Display > Component Display > Selection Handle*. You can then move the selection handle in front of the upper arm.

STEP 2

The weights of the Cluster deformer need adjusting, so go to *Deform > Paint Cluster Weights Tool > Options*. Modify the weights so that when you pull the Cluster forward, the biceps flex the most in the middle (Figure 24, previous page).

STEP 3

Open the Set Driven Key box *(Animation > Animate > Set Driven Key > Set > Option)*. Select the elbow joint which should highlight all the lower arm joints. Click the Load Driver button. In the right hand column under Driver, select RotateY (Figure 25). Whenever the joint that controls the lower arm rotates around the Y-axis, it will activate the biceps Cluster, making the upper arm's muscle bulge.

Load the Driven by selecting the biceps Cluster handle. Click the Load Driven button in the Set Driven Key box. Under the Driven category, in the right hand column, select translateZ.

Press the Key button to key the arm muscle in its relaxed position. On your rig, select the left hand icon and move it in on the X-axis toward the body and forward on the Z-axis so that the arm bends at the elbow. You may also have to move it on the Y-axis until the lower arm is closer to the upper arm.

Select the biceps Cluster and move it forward on the Z-axis to flex the muscle. Press the Key button in the Set Driven Key box.

You can now test the arm muscle flexing by moving the hand icon around. You should see the upper arm muscle relax and then flex when the lower arm bends at the elbow and moves closer to the upper arm.

ADJUSTING THE CONTROLLERS

When you start animating your character and you decide that you want to change the location of the hand, feet, and other controller icons, you should do the following. Go to *Modify > Evaluate Nodes > Ignore All* (Figure 26). This will keep the software from changing the position of the joints when you move the controllers.

CONCLUSION

This completes the long, arduous process of rigging a biped character. It takes patience and a great amount of attention to detail to rig a character correctly. Once the job is completed, there is a big pay off – your character now deforms properly when you animate or pose it for single image renders.

The next several issues will have you take your rigged character and create Character Sets, use the Trax Editor for non-linear animation, and then work with the Graph Editor to fix and enhance your animations.

Peter Ratner *is a professor of 3D Computer Animation at James Madison University. He is the founder and head of the first Computer Animation program in Virginia. His paintings, animations, and computer graphics have been displayed in numerous national and international juried exhibitions. He is the author of 3-D Human Modeling and Animation, 1st and 2nd Editions (John Wiley and Sons) and Mastering 3D Animation, 1st and 2nd Editions (Allworth Press). He lives in Penn Laird, Virginia.*

Hair Styling Techniques in SOFTIMAGE|XSI

By Raffael Dickreuter
Animator, Instructor
Venice, California

Adding hair to characters has been a challenge for quite some time. In this tutorial, we will take a closer look at some styling techniques inside SOFTIMAGE|XSI using the integrated hair styling tools.

Before starting, we need to determine what kind of character we want to style and get the basic look done. It's important to look carefully at reference images.

We are going to create a hairstyle similar to **Figure 1**, short hair for a man.

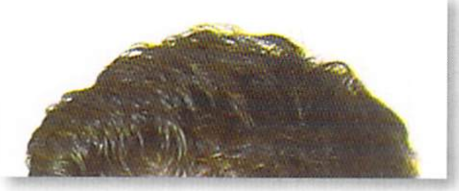


Figure 1

First, we need a character. You can import your own model, using the SOFTIMAGE|XSI Character Man (*Get > Primitive > Model Character Man*), or by following my example here and loading in a character that can be found in the local SOFTIMAGE|XSI netview. To do so, open the netview and go to SOFTIMAGE|XSI Local. Click on the Modeling tab, and then click on "Digimation." If you scroll down a bit, you will find "Man," which you can drag and drop into your viewport. This character will look like **Figure 2.** Select the hair geometry and simply delete it. We will create our own hair, so we don't need that.

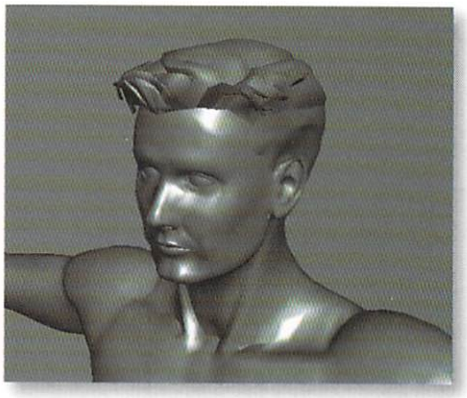


Figure 2

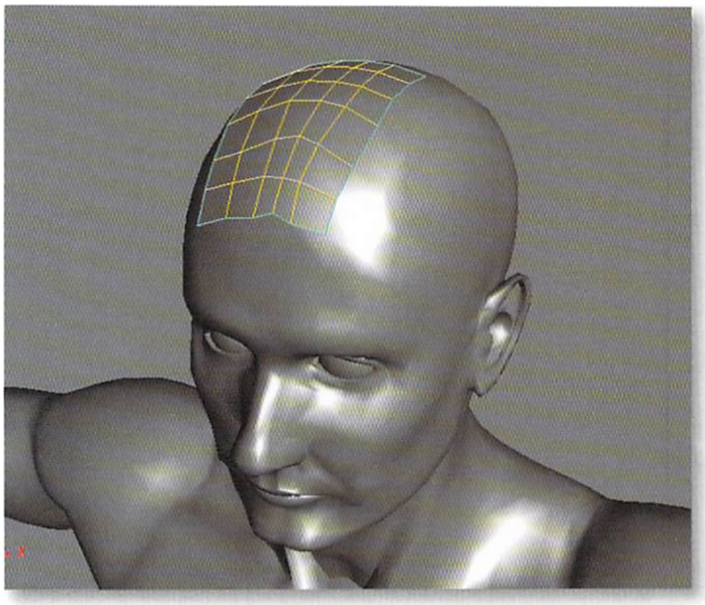


Figure 3

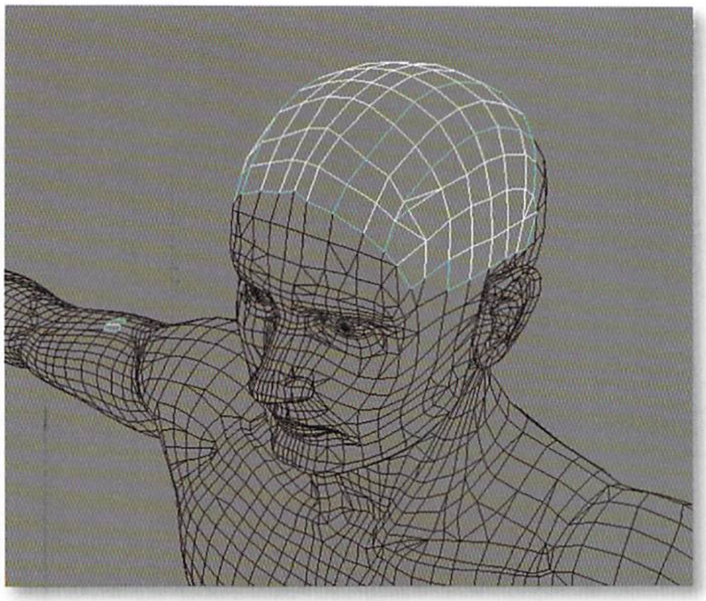


Figure 4

Before we get started, let me explain a few basic concepts that we will apply. We will generate hair from polygon geometry. We will not grow the hair from the head directly, but first extract a few polygons and then grow the hair from there. The reason for this is to give us more flexibility when we want to change the shape where the hair is growing from without changing any actual geometry of the character. Keep a flexible approach and don't get locked in later on, so we can make changes at any stage of the process.

We will also divide the hair generation into a few different pieces; this will give us more flexibility to style different parts of the head, without influencing parts we are already satisfied with.

So, let's get started. Select a few polygons on the upper part of the head where you want to grow the part. Then, go to the modeling panel and choose *Create > Property > Extract (keep).* **(See Figure 3.)**

Next, extract a few polygons on the left side of that; also extract the left and the right side near the ear. **(See Figure 4.)**

Now we have four different shapes we can start to grow hair from. Select all of them and rename them "Hair_Sources." Then select the first piece we originally extracted and start growing hair from it. To do so, go to the hair panel (CTRL + 2) and select *Create > Hair > from selection.* **(See Figure 5.)**

Click the "scale" button on the left side menu and scale them a bit. After that, click the *Comb Along Negative X-Axis.* This gives the hair a first direction to the left side. Then click *Puff > Puff Roots* and move the mouse a bit to move the hair slightly away from the head.

After these procedures, you should get a shape like **Figure 6.**

Draw a first render region using the Q key to see what we have so far. It doesn't look very stylish yet, and also doesn't seem to have a decent color. So, select the hair and click enter – this brings up the property page for the hair. To keep the rendering time as low as possible, we start off by decreasing the amount of hair from 6500 to 3500. If we feel later on that it doesn't look full enough, we can still increase the amount of hair, but for working and styling we are better off at the moment with less hair. Next, switch to the Effects tab and set the Frizz Root to 20 and the Frizz Tip to 35. Again, draw a render region to see the effect.

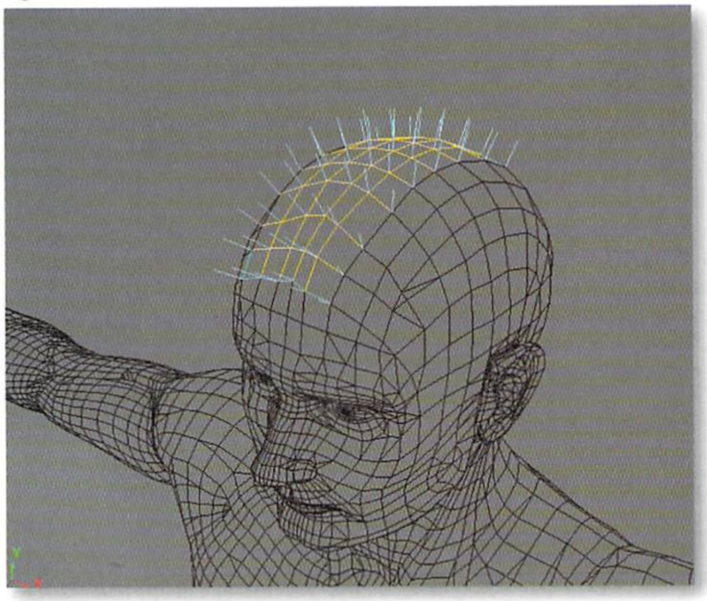
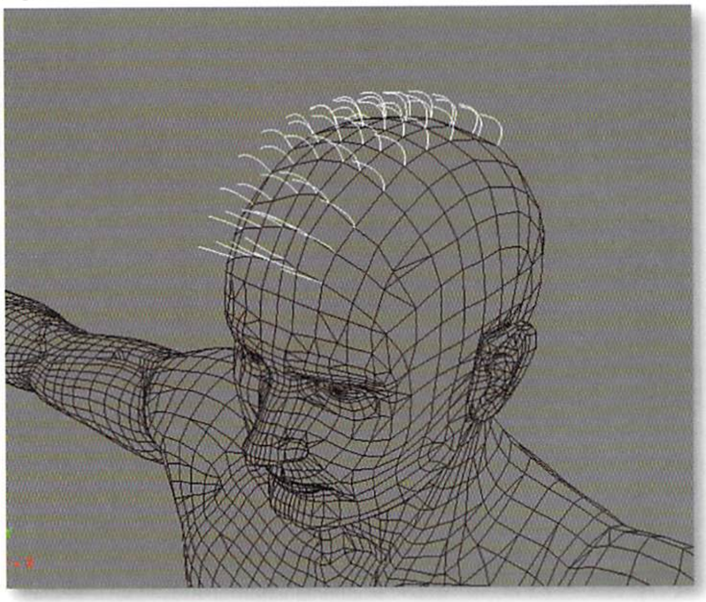
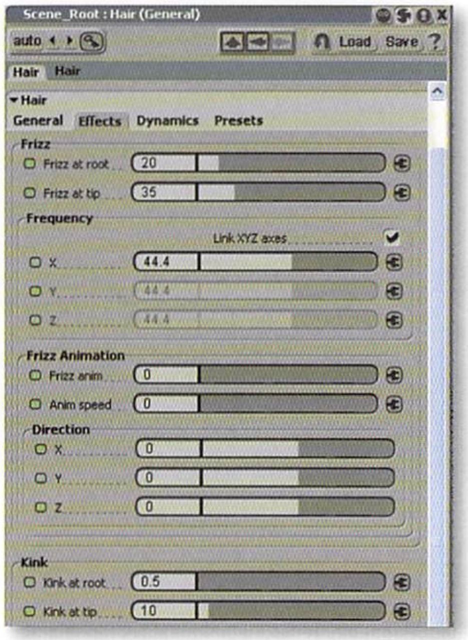
Figure 5


Figure 6




Scene_Root : Hair (General)

auto | Load | Save

Hair | Hair

Hair: General | Effects | Dynamics | Presets

Frizz
- Frizz at root: 20
- Frizz at tip: 35

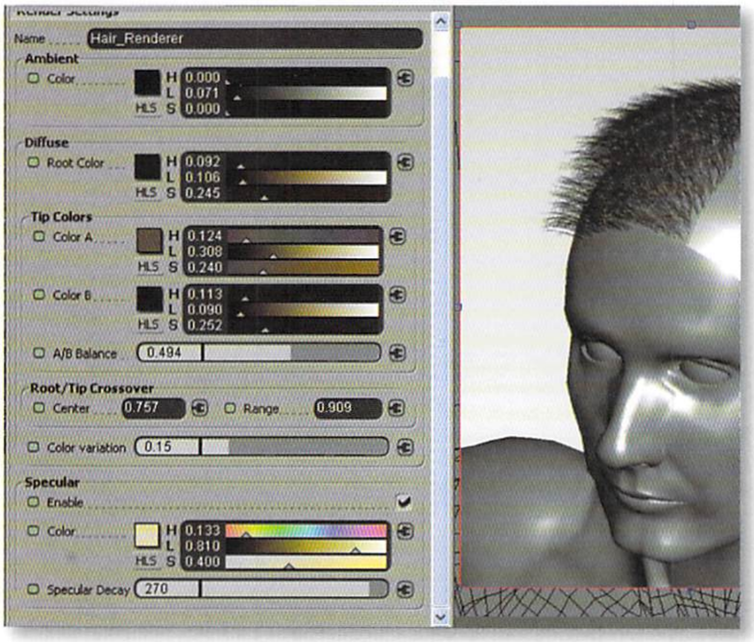
Frequency
- Link XYZ axes: ✓
- X: 44.4
- Y: 44.4
- Z: 44.4

Frizz Animation
- Frizz anim: 0
- Anim speed: 0

Direction
- X: 0
- Y: 0
- Z: 0

Kink
- Kink at root: 0.5
- Kink at tip: 10

Figure 7



Render Settings

Name: Hair_Renderer

Ambient
- Color: H 0.000, L 0.071, S 0.000

Diffuse
- Root Color: H 0.092, L 0.106, S 0.245

Tip Colors
- Color A: H 0.124, L 0.308, S 0.240
- Color B: H 0.113, L 0.090, S 0.252
- A/B Balance: 0.494

Root/Tip Crossover
- Center: 0.757
- Range: 0.909

Color variation: 0.15

Specular
- Enable: ✓
- Color: H 0.133, L 0.810, S 0.400
- Specular Decay: 270

Figure 8

Further below in the property page, set the Thickness for the root to 0.9 and the Tip to 0.7. This will make the hair a bit thinner and make it look more authentic. **(See Figure 7.)**

Now it's time to start playing around with the hair shader and change some of its colors. Select the hair and click *Modify > Shader.* Darken the Ambient and the Diffuse colors of the hair almost to black. Give the tip color a browner look. The second tip color should be somewhere between black and the brown color. Increase the Specular Decay to 270 and set the slider to a grey/white color. It's a good idea when you start shading your hair to add a very large white sphere into your scene; this makes it easier for you to see every hair strand. A three-point lighting setup can also be a good approach to see every side of your head properly. **(See Figure 8.)**

Next, hide the render region – it's time that we look at some more interactive styling techniques. If you click the "m" and you just try to modify the hair, you will notice that this doesn't work well, unless you mess up the strands. Proportional Modeling is much more effective. Click on the "Prop" button on the right side of the SOFTIMAGE|XSI interface to activate it. Then, right-click on the "Prop" button and set the Distance Limit to 1.5. Also, change the Falloff Profile to the Curve shape, as seen in **Figure 9**.

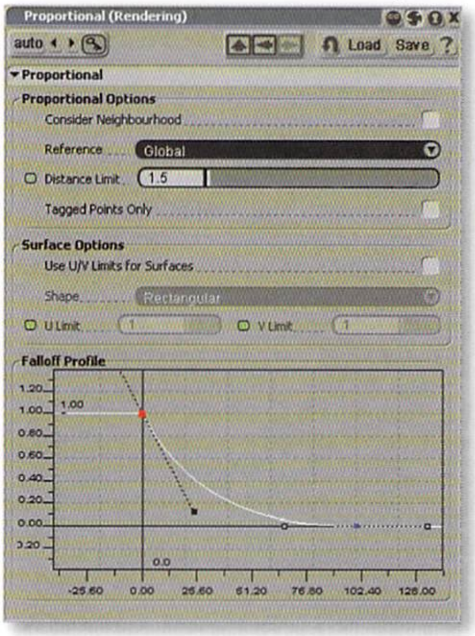
Now close that property page and click "m" to start shaping the hair a bit. Give it some more variety and style. Just start pushing and pulling points around with Proportional Modeling activated. You may also scale the hair a bit if you feel like it using the "scale" button. Play around with the Distance Limit in the proportional property page; increase it to around 10 and then shape the hair with it if it all becomes too messed up. **(See Figure 10.)**

Before we start adding more hair to the sides of our character, we do some preparation that will ensure we can work faster as soon as we have more hair.

Select your hair object and then hit (CTRL + G) to create a group. Rename it "Hair_color."

Your hair object has been added to the group. Select your hair object and click "F" while having your mouse pointer over the explorer. Expand the Hair node and drag and

Figure 9



Proportional (Rendering)

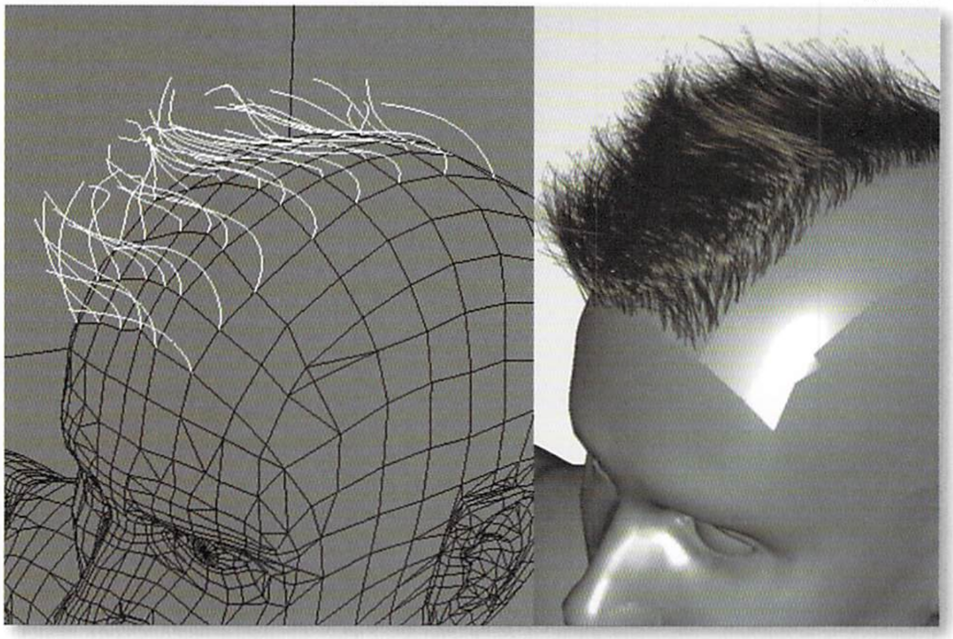
auto | Load | Save

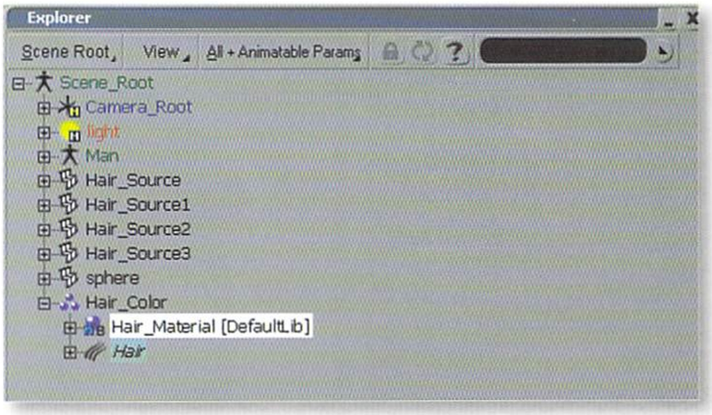
Proportional

Proportional Options
- Consider Neighbourhood: ☐
- Reference: Global
- Distance Limit: 1.5
- Tagged Points Only: ☐

Surface Options
- Use U/V Limits for Surfaces: ☐
- Shape: Rectangular
- U Limit: 1
- V Limit: 1

Figure 10





Scene_Root
Camera_Root
light
Man
Hair_Source
Hair_Source1
Hair_Source2
Hair_Source3
sphere
Hair_Color
Hair_Material [DefaultLib]
Hair

Figure 11

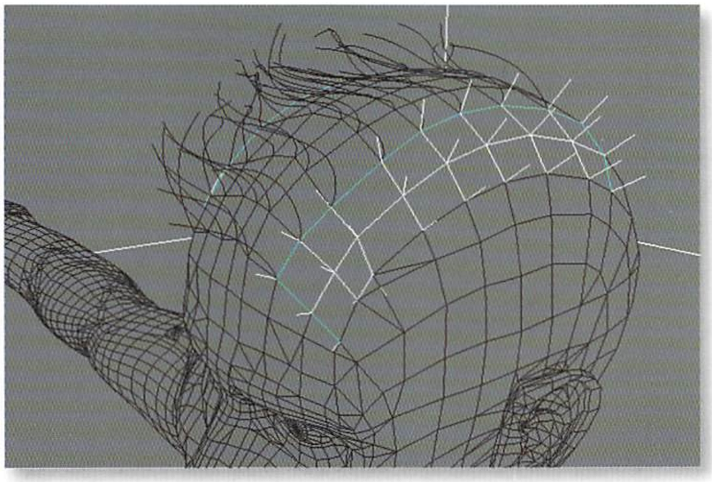


Figure 12

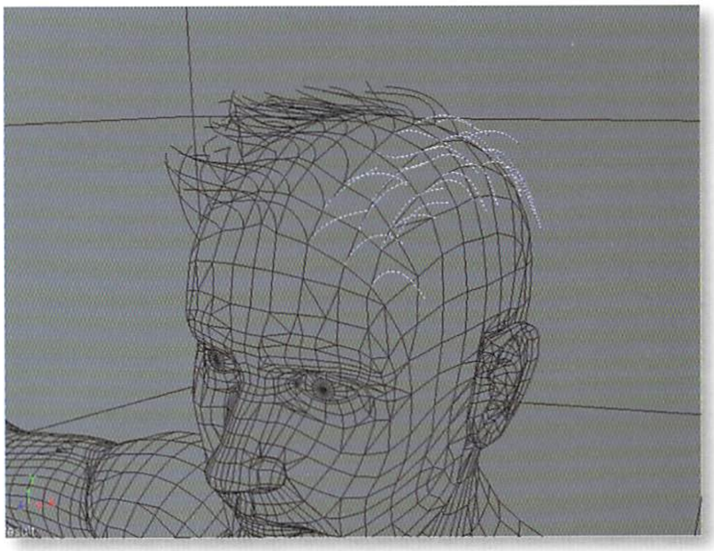
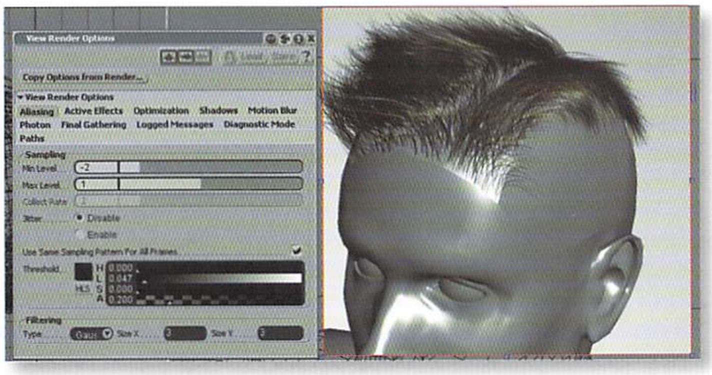


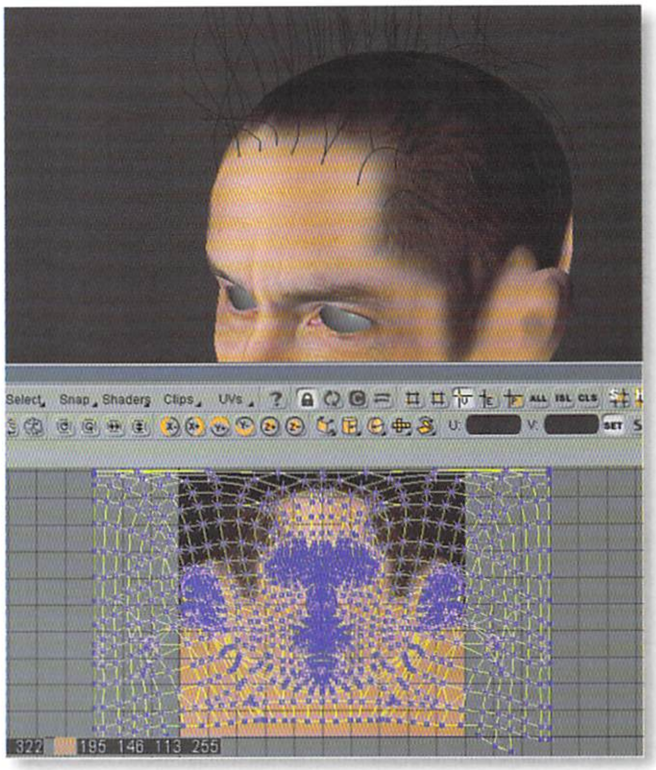
Figure 13



View Render Options
Sampling
Min Level -2
Max Level 1
Collect Rate 1
Jitter ● Disable ○ Enable
Use Same Sampling Pattern For All Frames ✓
Threshold HLS H 0.000 L 0.047 S 0.000 A 0.200
Filtering
Type Gauss Size X 3 Size Y 3

Figure 14

Figure 15



drop the Hair Material node into your newly created group. The idea is that we will have all our hair pieces in that group and that they all share the same material. **(See Figure 11.)** Now select your next Hair Source object and grow hair from it. *Create > Hair from selection.* **(See Figure 12.)**

Scale the hair a bit and then apply a comb along the X-Axis. Use the Puff Roots tool to move the hair up a bit. Then start using Proportional Modeling in combination with the "m" key to shape the hair. **(See Figure 13.)**

Now select the hair and open an explorer using the 8 key. Drag and drop the hair into your Hair_color group so now both pieces have the same color and share one material. After selecting the hair piece again, click Enter to open its property page. Set the amount of hair to 2000 and adjust the Thickness and Frizz so that they match the other hair piece.

Before drawing a render region again, go to your region options and set the Threshold to a low level and set the filtering type to Gaussian. Notice how the hair already looks more detailed and clear. **(See Figure 14.)**

Something that strikes the viewer's eye is definitely that so far the hair is too transparent; we can see the roots, which makes it look not full. The best way is definitely to add a texture to your head and paint the area where the hair is growing dark. Also, try to give it a similar color to the hair you have. Add a material to your character, add a texture, and paint some dark color where the hair is. **(See Figure 15.)**

For this, hide your Hair_source objects. After that, unhide them and start moving the points of them to adjust them so that the hair starts growing in the right place. Be careful – when adjusting the Hair_source object too much, recombing and restyling might be necessary; however, for some general tweaks, it usually works fine. Also

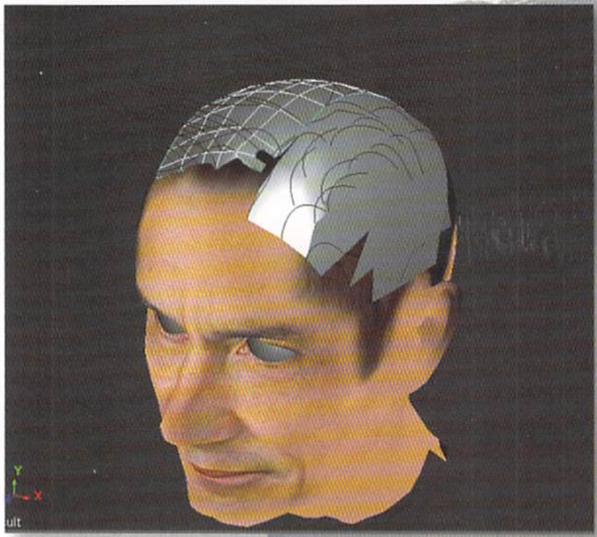


Figure 16

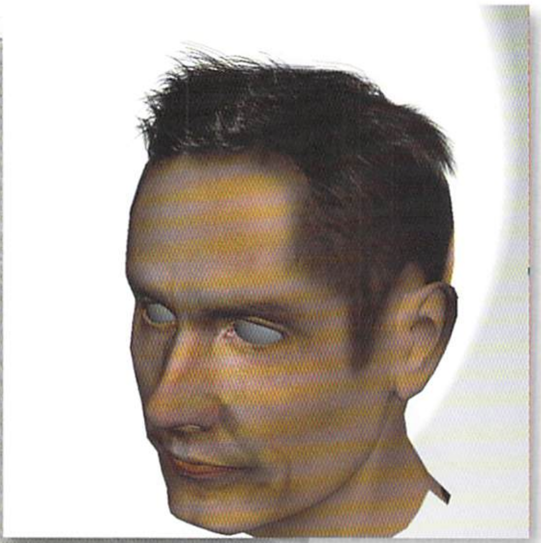


Figure 17

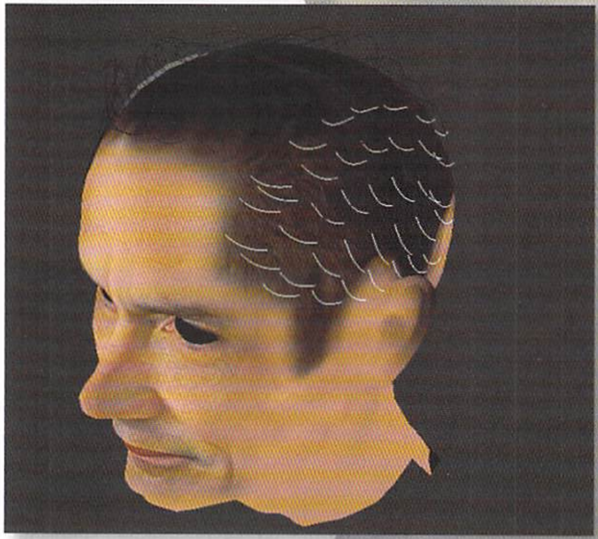


Figure 18

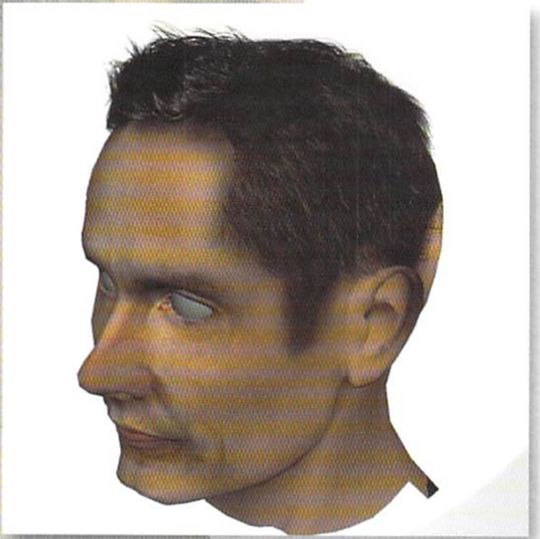


Figure 19

use Proportional Modeling for reshaping the hair source object. **(See Figure 16.)**

Now hide the Hair source objects, as we don't want them to render (their only purpose is to control where the hair grows). Draw a render region again and see what the hair looks like now with a texture underneath it. Adjust the colors of the hair in the shader property page and also start correcting your hairstyle where you feel it's not correct. **(See Figure 17.)**

Unhide the Hair source object where you want to grow hair on the sides and create some pieces of short hair. Drag and drop it into the material group so the hair colors match up. Decrease the amount of hair to 2000 and adjust the Frizz as well as the Thickness. Use the combing tools and then Proportional Modeling to give it the shape you want. **(See Figures 18 & 19.)**

Keep refining the hairstyle by playing around with the colors, Proportional Modeling the Hair source objects to get the look you want. A good idea at this point is also to give your Hair objects different wire colors; this makes it easier in wireframe or hidden line removal mode to see where the transitions are. The next challenge is to fix these areas and make a nice transition so the viewer doesn't see that the hair was broken into different pieces. **(See Figure 20.)**

Don't forget to freeze the hair often when you use Proportional Modeling or other styling tools. If you don't do it for a long time it might suddenly appear that the hair goes crazy and becomes very long and totally messed up.

You have now learned about various styling tools within SOFTIMAGE|XSI that you can apply to create various hairstyles. To learn more, explore the tools on the left menu bar that I didn't explain in this tutorial.

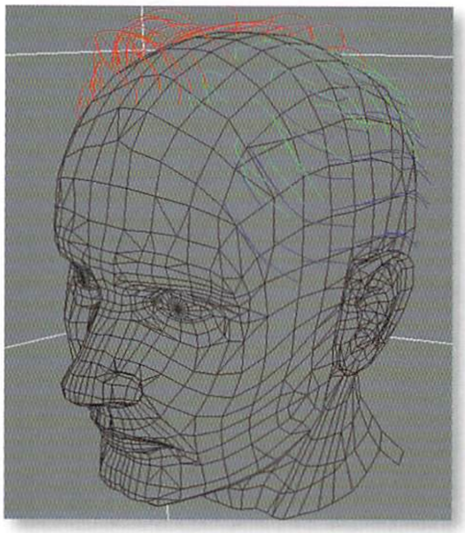
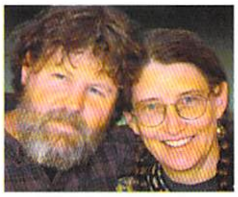


Figure 20

Raffael Dickreuter *is the founder of XSI Base.com, the largest online SOFTIMAGE|XSI community, and co-organizer of the London XSI User Group. He is a certified SOFTIMAGE|XSI Instructor and has worked in various industries including Broadcast, Web Development, Advertising, and 3D Animation.*

LightWave Essentials: *Flash Animations*



By Brad Carvey
Electrical Engineer, 3D Animator
New Mexico

Swift 3D is a plug-in that makes it easy to create Flash animations that can be show on a Web site. Swift 3D LW4.0 is a commercial LightWave plug-in that creates Flash movies from LightWave animations. The plug-in has the ability to render movies and stills with shadows, specular highlights, reflections, cartoon-style outlines, and a variety of color shadings.

DOWNLOAD AND INSTALL THE SWIFT 3D PLUG-IN

1 Download the Swift 3D LightWave plug-in from **www.erain.com**

2 Double click the downloaded plug-in and follow the install instructions.

3 Follow the instructions to add the plug-in to LightWave.

LOAD AN OBJECT OR SCENE FILE

1 Load an Object or Scene file. I choose William "Proton" Vaughn's cartoon self-portrait. I think it works well with Flash and I like the character. The model comes with LightWave 8. If the model you decide to use is a Sub-D model, you will need to freeze it first. Swift 3D only works with three and four-sided polygons.

2 Select the Swift 3D plug-in Render Options Interface. See **Figure 1**.

3 Use the dropdown menu to change the Edge Type from "Not Outlined" to "Outlined."

4 Use the dropdown menu to change the Fill Type to "Not Filled."

5 Select the Swift 3D Render Frame option. A simple line art version of William is shown in the Preview Window. See **Figure 2**.

RENDER AN ANIMATION

1 If necessary, set up an animation for your object. It does not need to be complicated.

2 Select the Swift 3D plug-in Render Options Interface. See **Figure 1**.

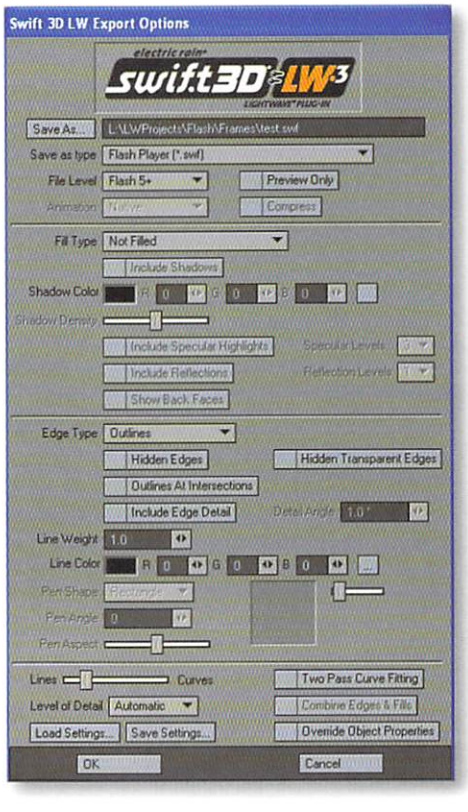
3 Use the dropdown menu to change Save as type to "Flash Player (*.swf).

4 Select the Swift 3D Render Scene plug-in.

5 The Preview window will show each frame being rendered.

6 You can use QuickTime to preview your animation.

Figure 1



Swift 3D LW Export Options

Save As... L:\LWProjects\Flash\Frames\test.swf
Save as type: Flash Player (*.swf)
File Level: Flash 5+
Fill Type: Not Filled
Shadow Color: R 0 G 0 B 0
Edge Type: Outlines
Line Weight: 1.0
Line Color: R 0 G 0 B 0
Pen Angle: 0
Detail Angle: 1.0°
Level of Detail: Automatic

Figure 2

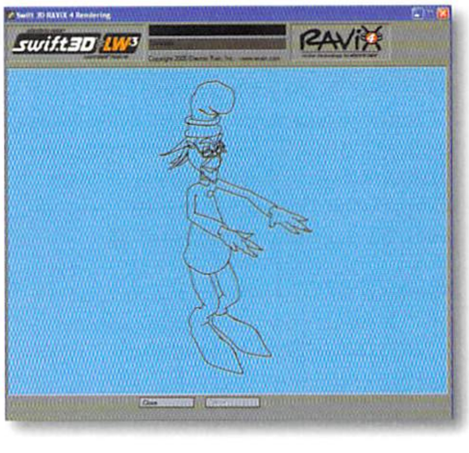
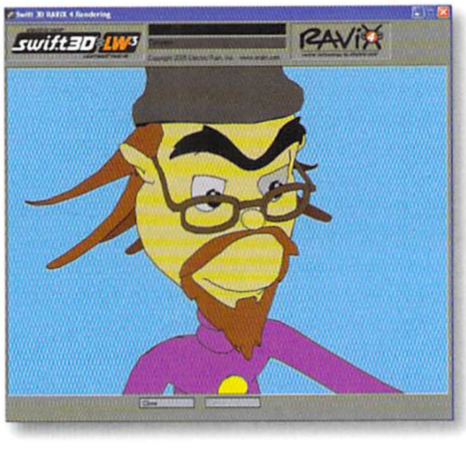


Figure 3



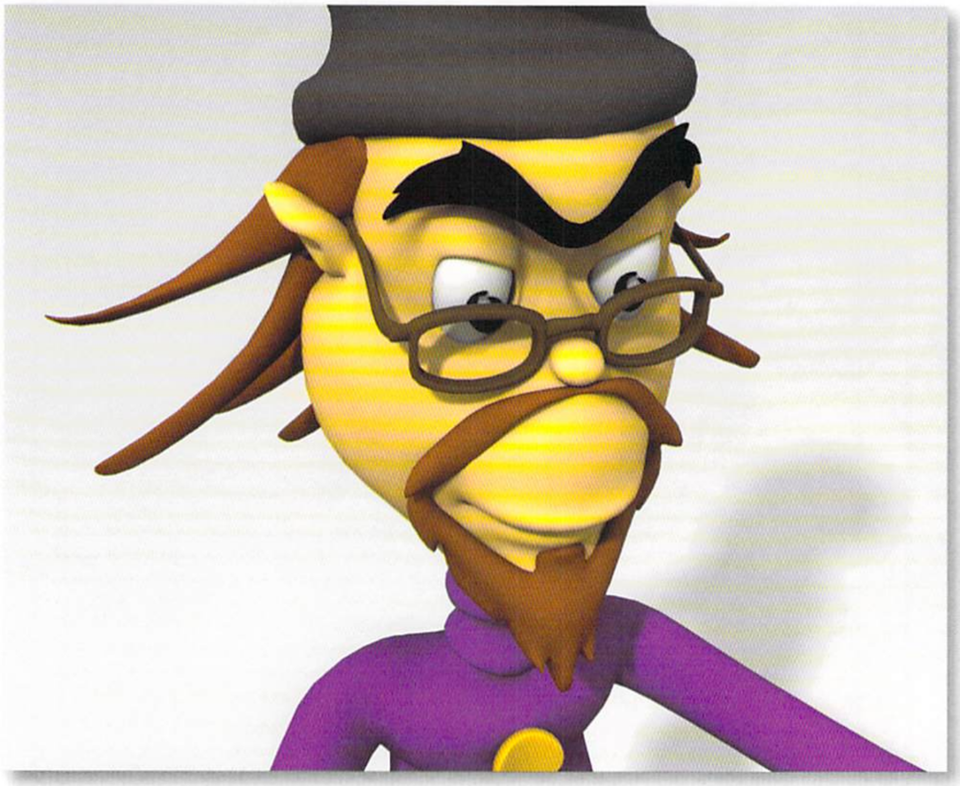


Figure 4

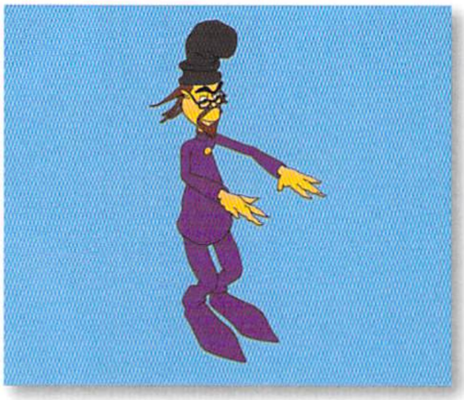


Figure 5

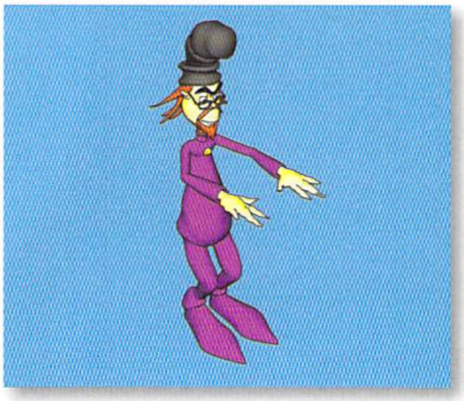


Figure 6

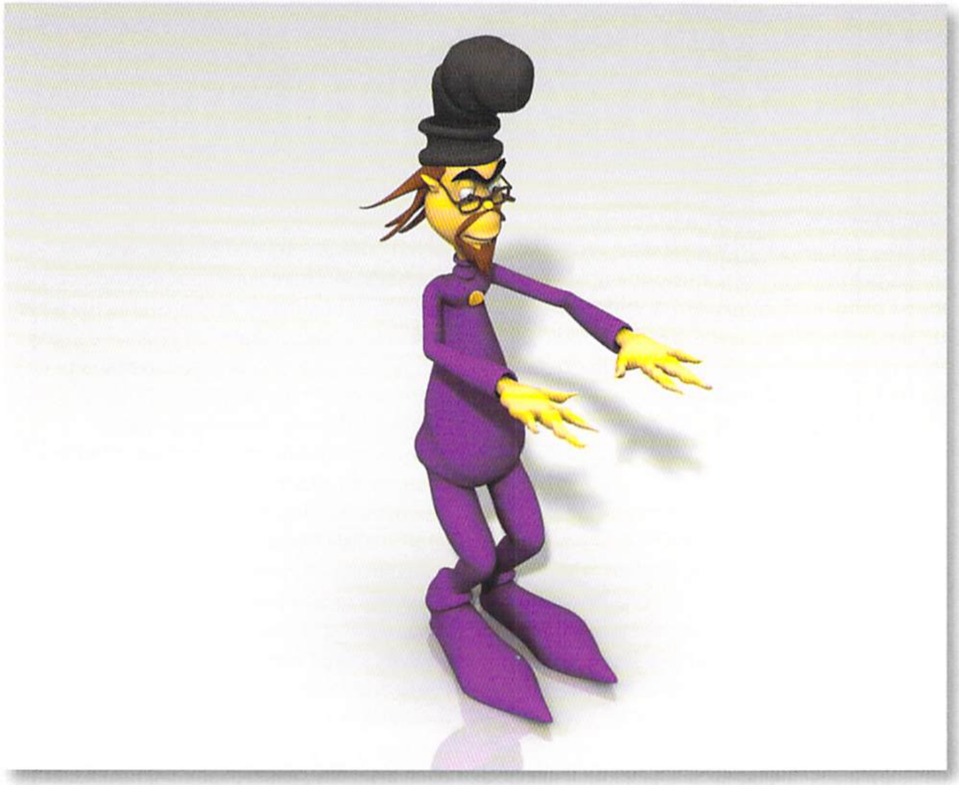


Figure 7

ADD A RENDERED FLASH MOVIE TO A WEB SITE

1 Add "<EMBED src="Your_Movie_ Name.swf" quality=high WIDTH = "500" HEIGHT="500"</EMBED>, to your websites HTML code.

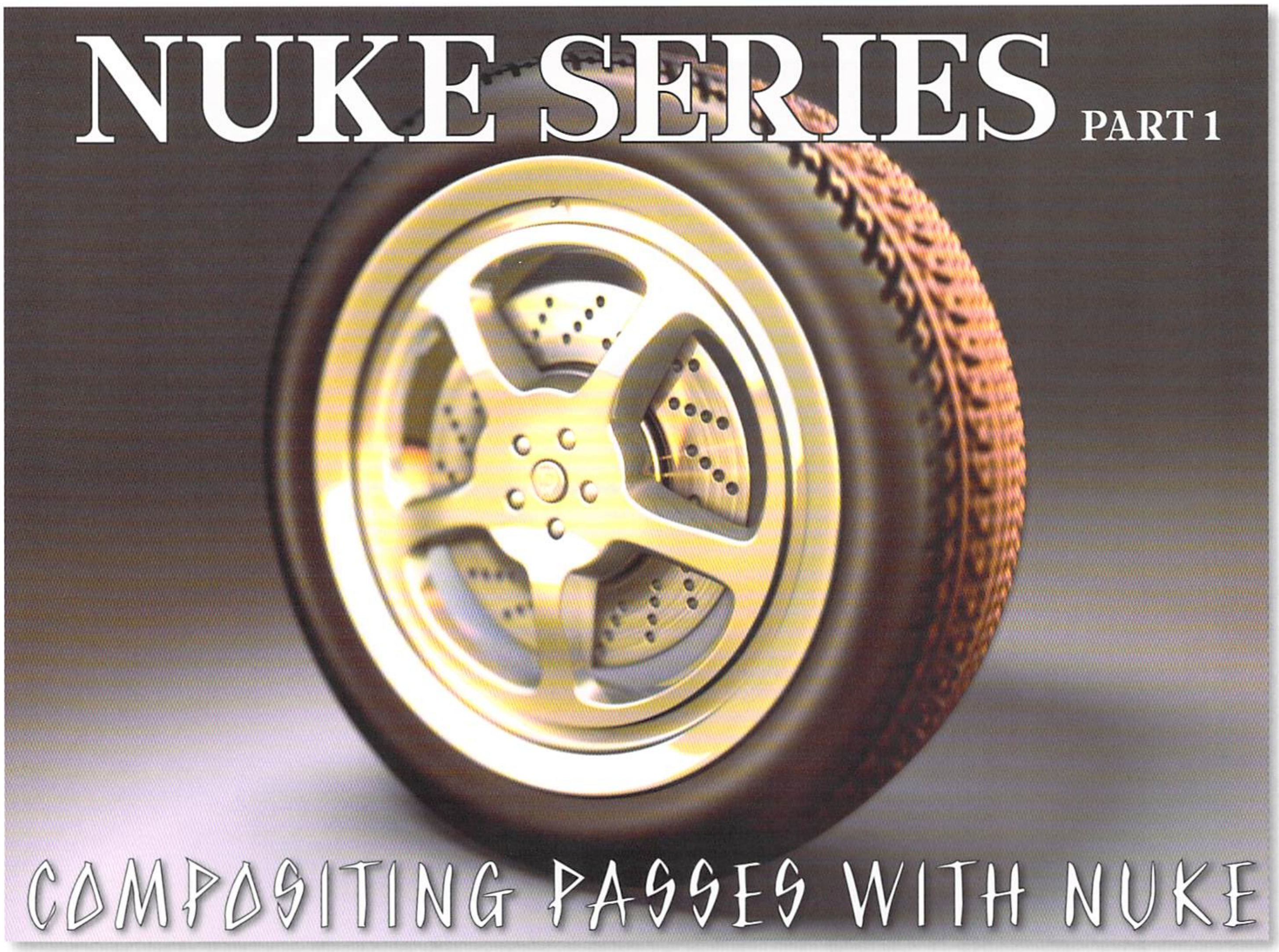
2 Upload the modified HTML and your Flash movie to your website.

You can change the Swift3D render options and create a variety of interesting animations. I have included a few different versions of William rendered with just a few of the possibilities. I also did a few high quality LightWave renders to show the difference between the look of the Flash Animations and LightWave rendered animations. **Figure 3** is a Flash still and **Figure 4** is the same image rendered with LightWave's Rendering Engine. **Figure 5** is a one color flash image with outlines. **Figure 6** is a full color mesh Flash mesh rendering with outlines. **Figure 7** is a high quality LightWave rendered image.

Brad Carvey *has been doing computer animations for a long time. In 1969, Brad used an analog computer, which was the size of a car, to produce his first computer animation. Brad is an electrical engineer and an Emmy award-winning member of the Video Toaster development team. He prefers to do feature film work. His credits include films like* ***Men in Black, Stuart Little, Black Hawk Down, Kate & Leopold,*** *and* ***Master of Disguise.***

NUKE SERIES PART 1

COMPOSITING PASSES WITH NUKE


The final rendered image.

By Boaz Livny
3D Artist, Animator, Teacher
New York City, New York

Welcome back! This tutorial follows up on the previous article, "Rendering Passes with mental ray" in issue #5. To quickly recap, in the previous article I introduced different approaches and techniques for rendering color and specialty passes using Maya and mental ray. We examined the mathematical theory behind color evaluation in order to better understand the calculation of color during the render process. Understanding color evaluation is the cornerstone for being able to foresee which passes are required for successfully recreating a render with a compositing package. We also examined the use of the mental ray Framebuffer data types, found under the Framebuffer tab in the mental ray render global tab. These data types dictate the image Bit Depth, Channels, and Color Clipping instructions.

This article is divided into two equally important tasks: the first introduces compositing with Nuke, and the second reviews compositing the passes rendered in the previous article, introducing some fairly straightforward compositing techniques using a node-based workflow. We also look at some of Nuke's unique and powerful compositing features to better understand its abilities.

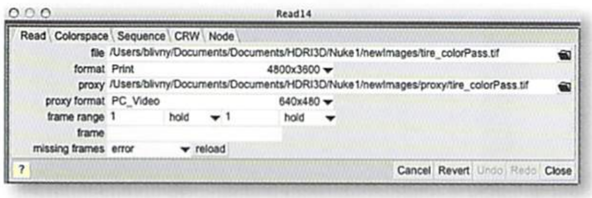
NUKE

Nuke, an award-winning compositing package developed at Digital Domain, provides solutions for many difficult compositing tasks. One of the challenges Nuke addresses particularly well is its support for compositing multi-channel images, typically ILM's OpenEXR image format. Unlike standard image files that contain up to four color channels (the RGB color channels as well as an additional Alpha channel), the OpenEXR image format expands on that ability by being able to store an unlimited amount of additional color channels. These additional channels, typically the RGBA values from several passes, represent color values from passes such as the Color passes, Specular passes, Masking passes, Light passes, and much more, all packed within an individual image file.

You can render this kind of EXR image, which contains several passes, with RenderMan. Nuke supports reading such files (multi-channels), and provides the ability to create, copy, and shuffle channels in any imaginable way, allowing you to manage up to 64 channels per script.

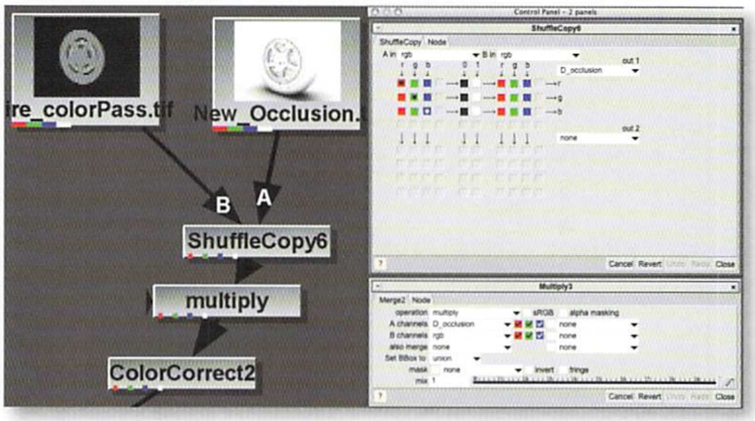
Nuke supports combining source images by appending their color channels to separate channels in the process tree downstream, downwards from the initial source node. Appending color values in the downstream eliminates the need to pipe a particular image several times throughout the composite. It is also a means for compiling an EXR image from several rendered passes. For example, although Maya supports rendering EXR images, it does not support rendering several passes into the same image as RenderMan, thus not really taking advantage of EXR capabilities. However, you can use Nuke to bridge this gap by compiling the Maya render passes into a single EXR format, image file.

Another unsurpassed ability of Nuke is its 3D integration within a compositing package, allowing you to integrate 3D elements as well as import motion cameras, apply projection mapping, and create complex composites within the OpenGL environment.



file /Users/blivny/Documents/Documents/HDRI3D/Nuke1/newImages/tire_colorPass.tif
format Print 4800x3600
proxy /Users/blivny/Documents/Documents/HDRI3D/Nuke1/newImages/proxy/tire_colorPass.tif
proxy format PC_Video 640x480
frame range 1 hold 1 hold
frame
missing frames error reload

Figure 1– Loading Images into Nuke.



A in rgb B in rgb
out 1 D_occlusion
out 2 none

operation multiply
A channels D_occlusion none
B channels rgb none
also merge none none
Set BBox to union
mask none invert fringe
mix 1

Figure 2– Nuke flowchart for compositing.

3D integration within a compositing package is nothing new; however, within Nuke, it is extremely robust and offers several useful features to help minimize production times, especially for complex shots that integrate both 3D and 2D elements.

Nuke carries out all of its processes within a script using 32 Bit floating point precision ('scripts' is a term typically used for referring to Nuke or Shake files, also 'flowcharts'). This means all images read into Nuke convert into 32 Bit floating points; all processes and effects also use this level of precision (see previous article in *HDRI 3D*, issue#5, for more on image data types). This provides a wide range of color values for use in describing gradation for each color channel, which is particularly important for film production. Grading artifacts can appear when using standard 8 Bit image processing because there is not enough grading between color values, but there is a great reduction of this sort of characteristic when each effect has a 32 Bit color scale to process color.

Another great feature is Nuke's compatibility with floating point HDR images (Tiff and EXR formats), providing an Exposure operator node to control exposure levels as you do in programs such as HDR Shop. Combining this ability with Nuke's 3D integration enables you to map an HDR image to a spherical environment within the openGL 3D environment as we do within Maya or **XSI**.

In the case of this tutorial, we have several passes that are separate image files, rather then several channels within a single OpenEXR image file. We will add these passes to separate channels in the flowchart downstream as they are required in the comp. Then, by calling their channels from the flowchart in the comp, we integrate them with the composite.

CHANNELS DEMYSTIFIED

The easiest way to comprehend the theory behind using channels within Nuke is to think of a Nuke script as a filing cabinet that can contain several folders (layers) based on a labeling system, and each folder may contain any number of files (channels), as long as the total of all files does not exceed the 64 limit per script. The limit only refers to channels (files), but you may have an unlimited amount of layers (folders), and some of those layers may be empty, thus have no compositing value. Since only channels hold actual image data, used to describe color values, the layers are primarily for organization purposes, acting as vessels that the channels live in and making for an easier compositing workflow. Ultimately, this enables you to organize the script (the cabinet) in any way you wish, copying or reusing files within different folders, or alternatively having all 64 "files" placed within one "folder." OpenEXR images work in the same way, without the 64-channel limit; in the near future, Nuke will offer unlimited channels per script, taking full advantage of OpenEXR abilities.

With respect to daily usage, we find that normally we render up to five channels, which include R, G, B, Alpha and Z-depth that are all contained within a single image file, such as the Maya Iff format. When using the OpenEXR format with Nuke, an image can contain Color, Specular, Occlusion and other "folders" in the "filing cabinet." Each of these "folders" contains "files" that represent the RGBA color values for that pass respectively. Also, an additional depth "folder" representing the Z-channel "file" may exist. When using multi-channels compositing with Nuke, you may have up to 64 "files" in any amount of "folders" within the script, which can either be read from an existing OpenEXR image or created during the compositing process, which is the process we use in this tutorial.

To translate all this into practical use, I can import several images, and append their color values into separate layers and channels in the flowchart, for example, adding the Occlusion RGB channels into a separate layer in the comp downstream, meaning that at any time during the composite I can access the Occlusion pass RGB channels through that layer, without the need to connect it an additional time into the flowchart.

On the most basic level, this process eliminates the need to pipe masking passes several times in the composite tree; hence, once it is added into its own layer and respective channels, the mask can be called upon by any operator during the composite, as long as you refer to an existing layer in the comp tree. Note that the process of appending passes to the flow chart is a bit of overkill for this tutorial; however, it helps explain the art of managing channels with Nuke.

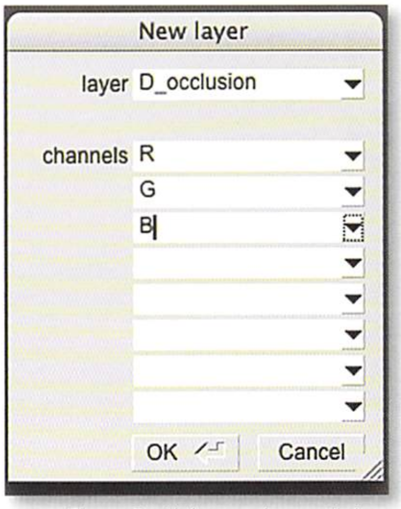
THE FIRST BASIC STEPS

Every Nuke script has a global setting; press the S key over the workspace to view it. Pressing the S key over the viewer shows you the viewer settings. The global setting for a script provides control over frame rate, time span, whether to use high-resolution images or proxy images, as well as choosing resolution for images created within Nuke, such as color bars. The Transform menu Reformat operator provides an alternative for loading proxy images. This node creates proxies on the fly by scaling the image to a pre-determined smaller size; Nuke's calculates all of its processes using that lower resolution. Both ways for using proxy images, defined either by the Read or Reformat nodes, are only used when the Proxy Mode under the global settings window is enabled; otherwise, the original high-resolution images are used.

COLOR AND REFLECTION PASSES

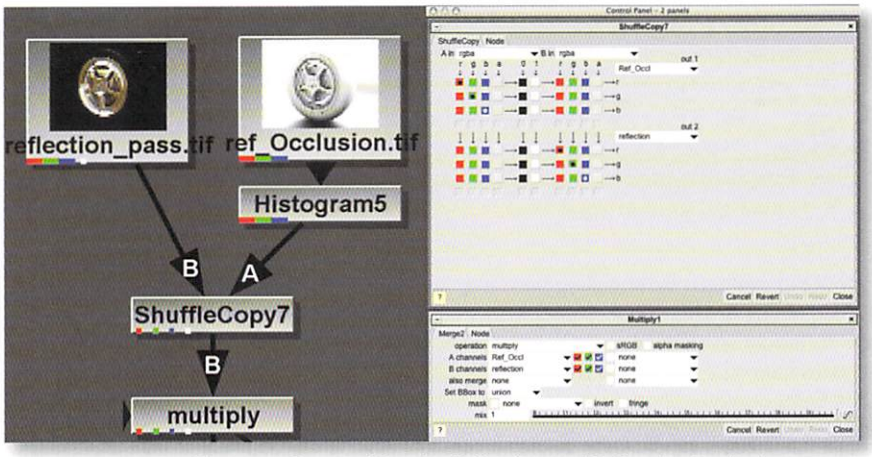
Let's first look at compositing the Color pass and Reflection pass with their respective Occlusion passes. I start by creating Read nodes, then browse to the image files from under the File option and load both high-resolution and proxy low resolution images into the script. **(Figure 1)**

Once the images are loaded, I want to multiply the Occlusion pass with the Color pass, providing a more natural feel for the Color pass color values (multiplying Occlusion passes was covered in detail in the Ambient Occlusion with mental ray article in issue#4). Before I multiply the images, I would like to start introducing working with channels by appending the Diffusive Occlusion pass RGB channels to a separate layer in the comp tree downstream, then multiply the appropriate layers and channels. There are a few different channel operators available in Nuke from under the Channel menu, providing different ways for controlling how channels copy or shuffle during the composite. **Figure 2** shows both



layer D_occlusion
channels R
G
B

Figure 3– Nuke new layer window.



ShuffleCopy7
ShuffleCopy | Node
A in rgba   B in rgba
out 1 Ref_Occl
out 2 reflection

Multiply1
Merge2 | Node
operation multiply   sRGB   alpha masking
A channels Ref_Occl   none
B channels reflection   none
also merge none   none
Set BBox to union
mask none   invert   fringe
mix 1

Figure 4– Compositing the Reflection and Reflection Occlusion passes.

Read nodes feeding into the ShuffleCopy node, which enables copying or shuffling channels from both A and B inputs. Notice that the Occlusion pass connects into the ShuffleCopy node, through the A input (foreground), and the Color pass connects using the B input (background). The B input always represents the comp tree downstream, and the A input is used to transfer selected channels from the A input to the B input downstream.

With the passes piped into the ShuffleCopy node (shuffleCopy6), I copy the RGB values under **A in** from the Occlusion pass into a new layer labeled D_occlusion displayed under **Out 1.** I accomplished this by selecting **New** from under the **Out 1** dropdown list, which opens the New Layer window, enabling you to name and define channels for a new layer **(Figure 3)**. This creates a new layer called D_occlusion, adds to the available layers and channels from the B input.

To clarify this process, the **A** and **B in** options **(Figure 2)** define which layers and channels are transferred into new layers and channels (or existing ones) defined under the **Out 1** and **2** options. This means that **Out 1** and **2** enable selecting the layer and channels to target in the comp tree downstream. **Figure 2** shows that the D_occlusion layer, RGB channels are receiving values from the **A in** RGB channels (respectively). These channels append to the B input node downstream, as the B input always represents the comp tree downstream. In this case, the Occlusion pass (RGB) channels transfer color values to new channels stored within the D_occlusion layer. The **B in** input is not used for transferring values, as all of its previous channels remain available in the comp downstream. In this case, the B input is only used for receiving channels from the A input, which become available only after (below) the ShuffleCopy node node. This process demonstrates one way you can add an image as a separate layer with appropriate channels, which you can call upon further down in the flowchart, without a need to reconnect it at that later stage.

Keep in mind that if the A input has several channels you need, in order to append them all, you might use more then one Shuffle Copy node or use the Copy Channels node. The Copy Channels node is an easier way for quickly copying channels from one node to another. Also note that with the Add channels operator allows for creation of new empty channels that you can target later in the comp downstream.

The next step uses a Multiply node found under the Merge – Merges menu that takes at least two inputs (A & B) and then multiplies the selected layers and channels. Both A & B connect to the same node (Merge – Multiply node). Normally, in compositing without multi-channel images, you would connect the A and B inputs to their respective images (separate image nodes). As you might have expected, with Nuke the B input represents the comp tree downstream. However, as we have appended the Occlusion into a separate layer, we connect both inputs to the same node and retrieve each image separately from their respective layer and channels. Under the Multiply dialog box **(Figure 2)**, we see that **A channels** shows the new D_occlusion layer selected as well as its three channels (RGB). Under **B channels** we see the RGB layer is selected, which is the "default" layer from the initial B input comp tree, and in this case represents the Color pass RGB channels. The result of multiplying the two together, as always, is stored within the B Channels input layer, the RGB layer.

After the Multiply node, a Color Correct node is applied to fine tune the result, and even though I can access each layer at any time during the composite, if I want a certain color correction to apply before the next step is calculated, I must apply it in the appropriate order. So, if I call the occlusion layer at any time and color correct it, that correction only applies then and there in the flow. Thus, in this case, I want to color correct the resulting RGB layer before adding the next step (Reflection pass); hence, I add the node beforehand.

For the Reflection Pass and Reflection Occlusion pass **(Figure 4)**, we follow a similar process, multiplying the Occlusion by the straight reflection color pass, all within a separate comp tree. Combining this comp with the Color pass comp creates a continuous comp flowchart. Notice the addition of a Histogram to the Occlusion layer before the ShuffleCopy node. This allows fine-tuning the grayscale levels for the Occlusion pass and respectively affecting the reflection values calculated further downstream with the Merge (multiply operation) node. The Histogram is added before the multiply node, enabling adjusting values, then seeing the result ripple downstream through the composite. In this case, purely for further demonstrating channel management, I applied both the Reflection Color and Reflection Occlusion to new layers and channels. **Figure 4** shows that the Reflection Occlusion is piped through input A into the ShuffleCopy, then transfers its RGB channels from its RGB layer, into a new layer labeled Ref_Occl under **Out 1**. Since the previous process showed that the B input channels are maintained further downstream from the ShuffleCopy, if you require un-tampered access to the Reflection Color pass RGB channels, you might consider copying them into a new layer, as demonstrated in the figure. Notice how **B in** takes the RGB values from the B input and appends them to a new layer labeled Reflection under **Out 2**. As before, I multiply the Occlusion pass by the Reflection pass, only in this case the merge node **B Channels** uses the Reflection layer and not the default RGB layer, thus the result of multiplying the two together is stored within the Reflection layer channels, in the B input downstream. At this point, the Reflection

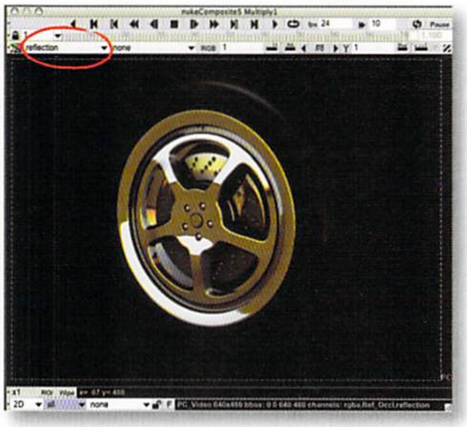


Figure 5– The Nuke viewer displaying the Reflection layer RGB channels.

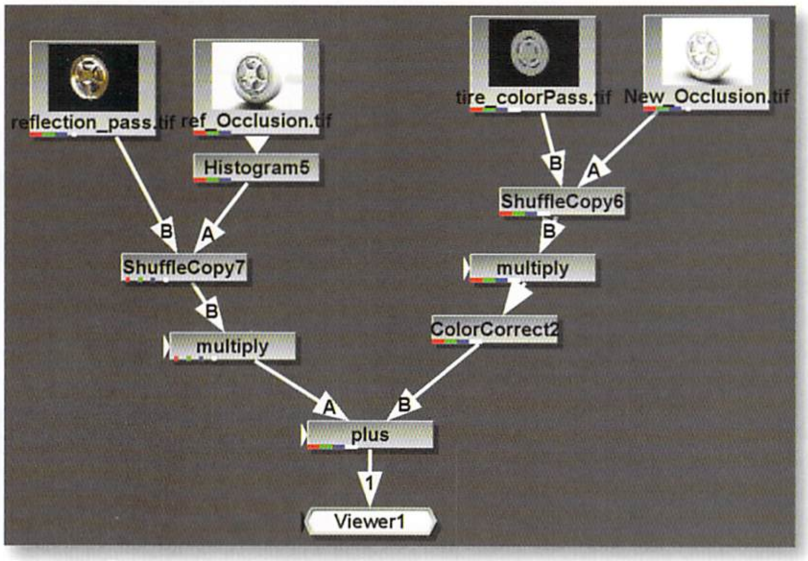


Figure 6– Compositing the Reflection and Color passes.

layer contains the multiply result and not the previous Reflection pass RGB values. Notice this result in the viewer, which enables viewing each layer and its channels.

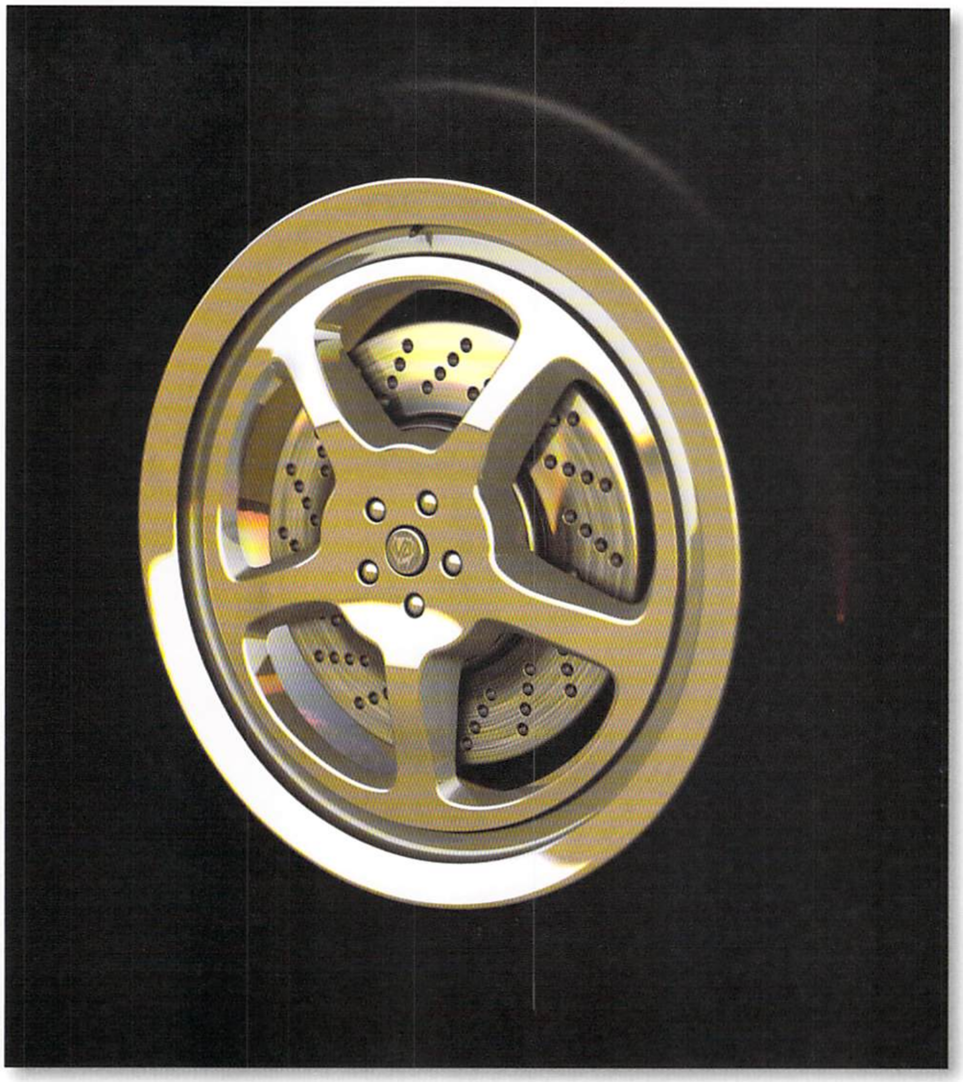
Figure 5 shows the viewer displaying the multiply result under the Reflection layer, which is marked with a circle. You can see that the Occlusion pass has scaled the Reflection pass so that it does not appear to reflect 100% uniformly across the surface, adding much needed realism to 3D renders. With respect to the viewer, if I choose to view the RGB layer at this point, the un-tampered Reflection pass would load, since we have not applied any changes to the initial RGB layer of the Reflection pass.

COMPOSITING THE COLOR PASSES

Now that we have completed the first basic steps for compositing the Color and Reflection pass, as well as reviewed managing layers and channels, let's further examine some methods for combining the first two comp trees, as well as adding the rest of the passes. Note that although some of these composite procedures represent the same process used by the 3D renderer to calculate the final image's color evaluation, this comp workflow demonstrates some of the control available when compositing passes rather then rendering the final image in one pass. The previous article in issue#5 explains the concept behind color evaluation.

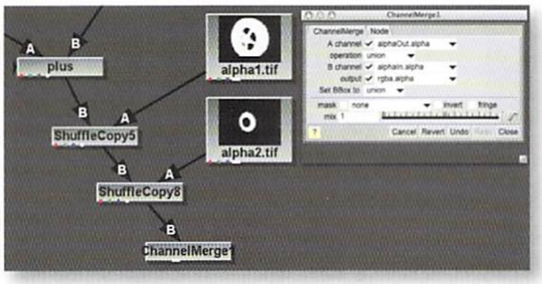
During 3D rendering, the Reflection and Specular colors are mathematically added onto the Color pass, so we must do the same in the comp. **Figure 6** shows the Plus node used to mathematically add the values from the first two processes. The Plus node is the same Merge node that is used for multiplying using a different mathematical operation for compositing channels from under the **Operation** option. **Figure 7** shows the result of adding the Reflection and Color passes together. At this point it is very important to realize that although each of these two comps have their own channels and layers, only the B input (feeding into the Merge node) maintains

Figure 7– The rendered result of adding the Reflection and Color passes.



its channels in the comp downstream, just as when using the ***ShuffleCopy*** nodes before. In this case, the Reflection and Reflection Occlusion passes cannot be called upon further downstream from this point, since they connect into the Plus node using the A input. If you would require access to any of the Reflection and Reflection Occlusion layers and channels, you might transfer those channels into the comp downstream using any of the channel operators, and perhaps before the Plus node for a better organized comp.

With respect to the previous discussion on appending Masking passes to separate channels, I will now add both Masking passes, which are renders of separate parts of the model. **Figure 8** shows the composite tree for adding these passes. This will let us use them further in the downstream to mask or block out certain effects. I add each pass to a new channel using the ShuffleCopy: AlphaOut represents the outer portion of the tire, and AlphaIn represents the inner portion. Since I need one full mask, meaning the combination of these two, I use the Channel Merge node with the Union operation as seen in **Figure 8**, this appends the result to the previously empty RGBA alpha channel. Yet again, another simple "out of the box"



ChannelMerge1
ChannelMerge | Node
A channel ✓ alphaOut.alpha
operation union
B channel ✓ alphaIn.alpha
output ✓ rgba.alpha
Set BBox to union
mask none invert fringe
mix 1
? Cancel Revert Undo Redo Close

Figure 8– Compositing the masking passes.

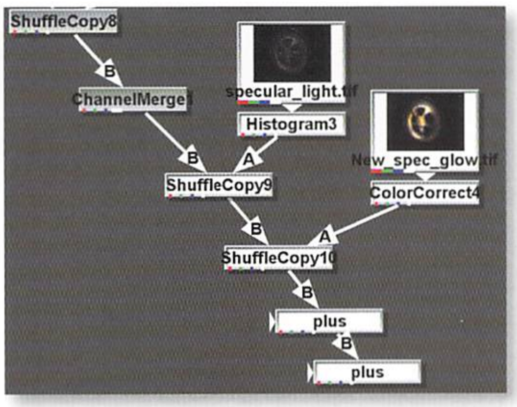
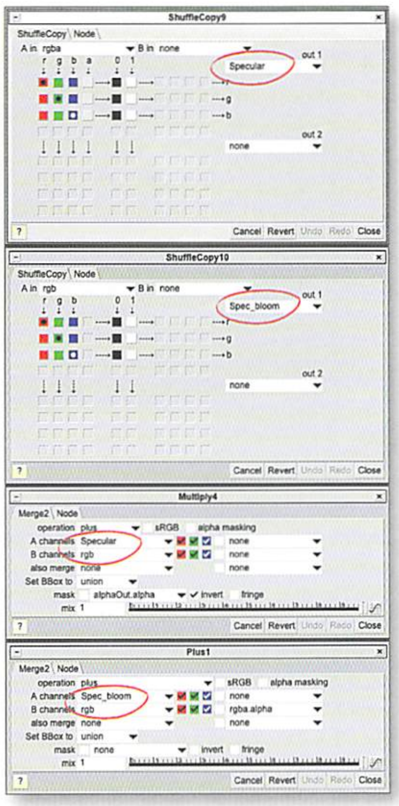


Figure 9– Compositing the Specular passes.



ShuffleCopy9
ShuffleCopy | Node
A in rgba B in none
out 1 Specular
out 2 none
? Cancel Revert Undo Redo Close

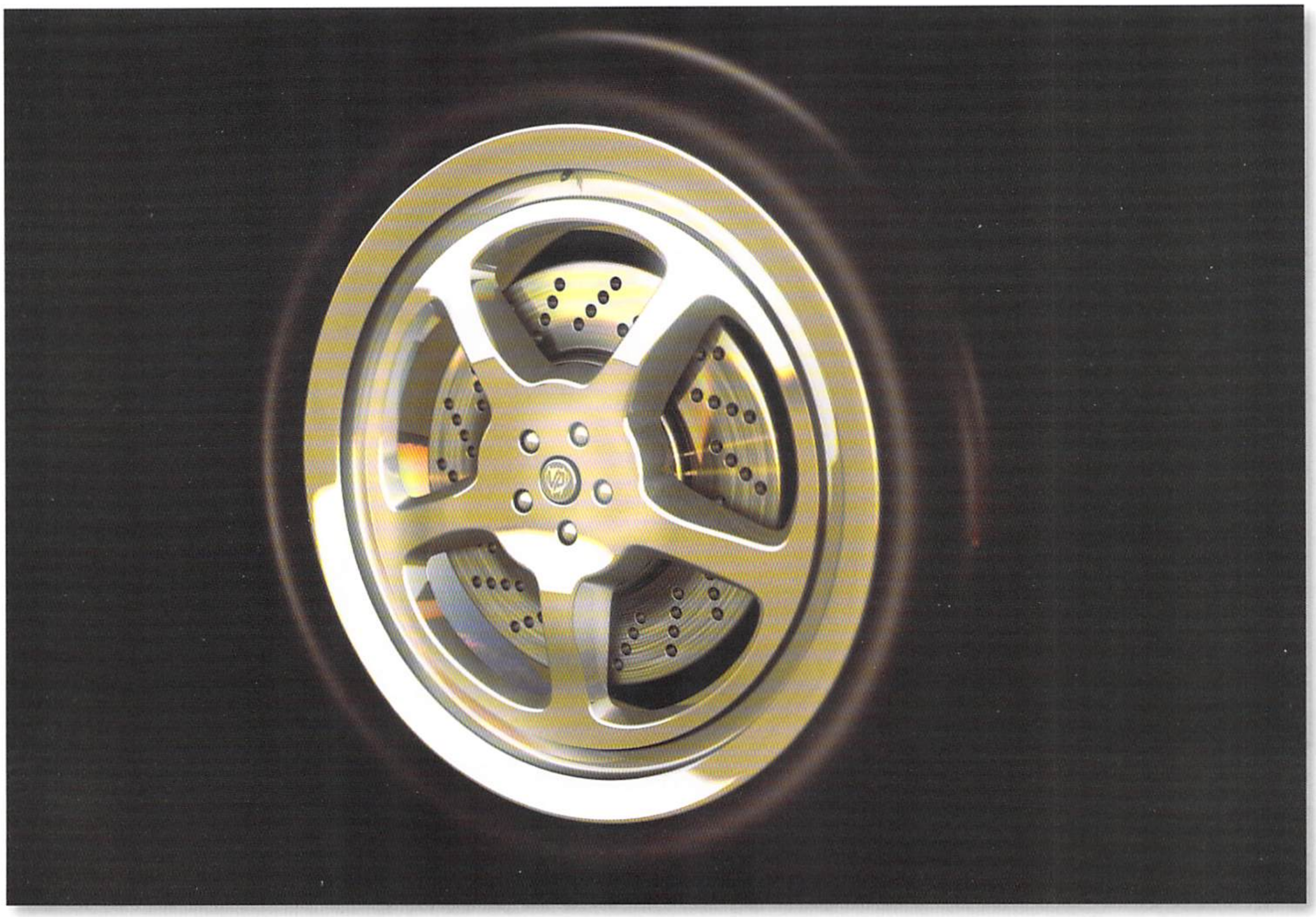
ShuffleCopy10
ShuffleCopy | Node
A in rgb B in none
out 1 Spec_bloom
out 2 none
? Cancel Revert Undo Redo Close

Multiply4
Merge2 | Node
operation plus sRGB alpha masking
A channels Specular none
B channels rgb none
also merge none none
Set BBox to union
mask alphaOut.alpha ✓ invert fringe
mix 1
? Cancel Revert Undo Redo Close

Plus1
Merge2 | Node
operation plus sRGB alpha masking
A channels Spec_bloom none
B channels rgb rgba.alpha
also merge none none
Set BBox to union
mask none invert fringe
mix 1
? Cancel Revert Undo Redo Close

Figure 10– Node attributes for compositing the Specular Passes

Figure 11– The rendered result after compositing the Specular passes.



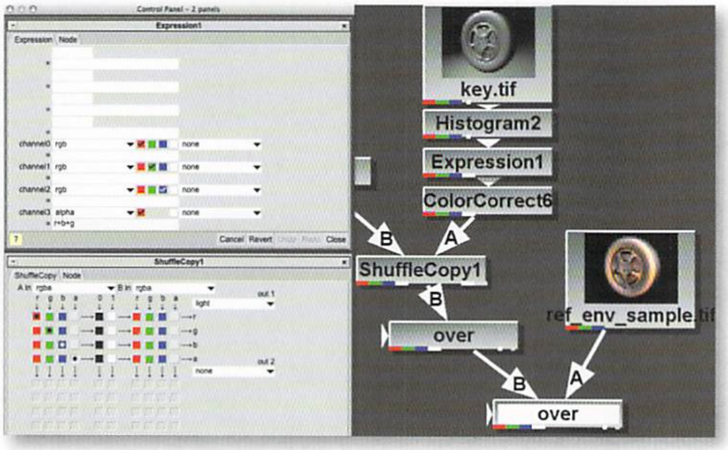


Figure 12– Compositing the Key light and Environment sampling pass.

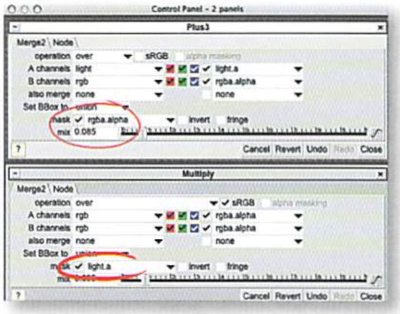


Figure 13– Node attributes for compositing the Light passes.

way to apply complex channel management processes.

There are two Specular passes, one representing the Specular bloom and the other is the standard Specular pass. I combine these two using some color control nodes to enable tweaking of their influences once they're attached to the flowchart as seen in **Figure 9**. I added these color corrective nodes at this stage, as I anticipate the need to adjust them once the flowchart is complete. Adding such nodes later in the game is not a problem; for example, if you select any node in the flowchart and apply any of the nodes available from the menus, it inserts directly between that node and the following node. We call this a non-destructive workflow, which enables applying changes at any time without the need to reassemble the entire tree.

I append the Specular color passes, as I did with the Alpha channels, onto separate layers in the comp flowchart. **Figure 10** shows the ShuffleCopy attributes for each pass, as well as the two Plus nodes attributes used when mathematically adding the Specular passes onto the RGB layer, hence onto the resulting color value from the previous Reflection and Color pass color evaluation **(Figure 11)**. Note that several nodes, such as Merge nodes, have a Mix attribute at the lower portion, enabling fine-tuning the amount of influence cast by the A input layer.

INTEGRATING THE KEY LIGHT AND BACKGROUND PLATES

The next step consists of integrating the Key Light into a separate layer, as well as passing its RGB color values into an Alpha channel, hence creating a light Alpha channel based on light intensity. **Figure 12** shows the flowchart for this stage, as well as the Expression and ShuffleCopy node attributes.

The ShuffleCopy1 node connects at the bottom of the current comp tree (under the Specular Bloom Plus node), appending the light pass comp RGBA channels to a new Light layer in the main comp downstream. The key light pass (key.tif) connects to a Histogram node, providing control over the light falloff and intensity before it pipes into the comp tree. We use the node Expression1 for transferring the light RGB values into the light Alpha channel; we will further examine this process shortly. Create the Expression node from under the Color menu, Math – Expression, and use it for manually entering custom instructions for calculating channels or linking channels. Nuke supports using the TCL (TiCkLe) scripting language for customizing scripts, typically for linking parameters; for example, linking the **Mix** attribute of one node to the **Mix** attribute on another node, creating a dependency link between two values. Another use for TCL is for applying mathematical functions that enable achieving specific color evaluations and effects using either simple math calculations, such as R+G+B, or by using complex math functions, such as cos, sin, abs, noise, and much more. Math functions are applicable in several compositing packages; use them for achieving very different results while compositing, expanding the "out of the box" compositing abilities. In this case, TCL is used to build on the possibilities available when using "built in" math functions, such as with the Merge node. There is an entire list of these mathematical functions in the Nuke User Guide. Note that you can apply expressions directly to node attributes, thus the Expression node is not a means to an end.

Some of the Expression node features enable selecting the channel to influence, adjacent to the **channel 0-3** inputs, and then setting the values for that channel by entering an expression below, adjacent to the equal sign. In this case **(Figure 12)**, I am using the Expression nodes **Channel 3** row to select the alpha channel, and then entering an expression that is used to mathematically add all three RGB channels from the Light pass together (=R+G+B), creating an Alpha channel representing the light influence region. The reason I added all three channels together is for getting a more opaque Alpha channel for properly representing the light affect. To better understand, consider you want to block out the region of influence the key light affects, for example, for adding other light passes into the composite. If you use any of the three channels (R, G, B) as a sole contributor for the hold-out mask, it will only represent the intensity values available from that channel. Perhaps that channel is not completely "white" at any given point, meaning its color value never provides a fully opaque mask for blocking-out the affect of other passes in that region. Once we add all three channels together, assuming that the resulting value exceeds a value of 1, the light region is fully opaque, guaranteeing that it will block out any other influence when used as a mask. Also, I still maintain the light falloff from the light edges, providing means for controlling light blending in regions the light influence decreases. I will use this Alpha channel as a mask for compositing the Environment Sampling pass seen in the lower portion of **Figure 13**. This process is not required for this particular comp, however I included it for demonstration purposes.

Using two Merge Over operations **(Figure 13)**, I first add the Light pass with some color correction to the RGB channels of the main comp. Next, I use the Environment Sampling pass, located at the lower portion of this figure to add some environmental color affect to the light's color values, making the light color appearance less uniform. Notice that the lower Merge node in **Figure 14** shows the Environment Sampling masked by the Light Alpha channel, which earlier was passed through the expression node. Both nodes have low Mix values so that they don't overpower the already very "bright" image. These steps show

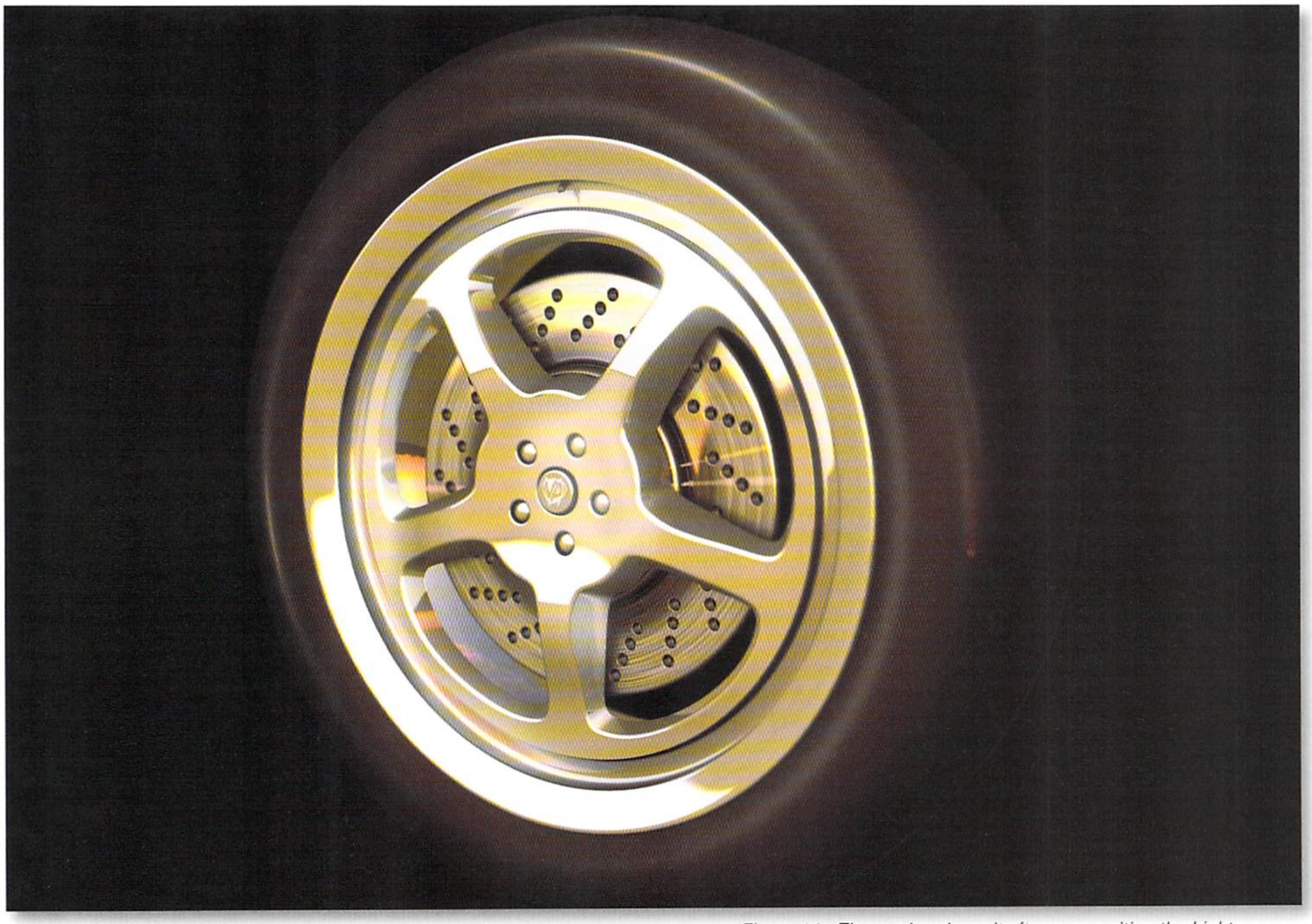


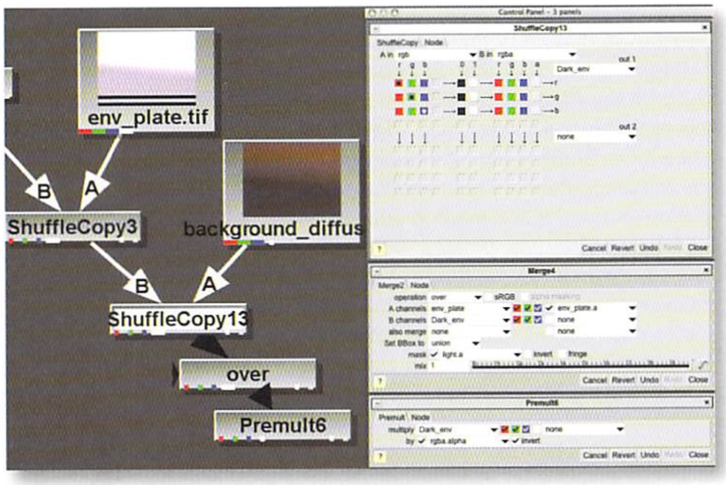
Figure 14– The rendered result after compositing the Light passes.

some of the ways you can use different passes while compositing to influence color. **Figure 14** shows the result when rendering the composite at this stage.

For the background plates, I have two identical images rendered with environmental sampling in Maya, with a difference in their light intensity. I use the Key Light Alpha channel to block the difference between the two, creating a richer light for the background, similar to the method used above for lighting the tire. **Figure 15** shows the flowchart and the relevant nodes.

The brighter image has been added to a layer labeled env_plate layer, and the darker image into a Dark_env layer. This is followed by a Merge node that adds both images together using the light Alpha as a hold-out mask, thus the brighter image color values are visible only in the light influenced regions. Finally, I use the Premultiply node for multiplying the Dark_env layer by the tire Alpha, omitting the light influence of the tire region from the background plate, as seen in **Figure 17**.

Figure 16 shows the Shadow pass added onto the background plate. If you render the Shadow pass with Maya using either the Global passes - Shadow pass, or with the Use Background shader, Maya renders a "white" shadow into the Alpha channel, as seen in the viewer portion of **Figure 16**. Convert this pass into a "black" RGB color if you need a dark shadow in the image, or use it as a mask with another Color pass. For converting this into RGB colors, first apply an Invert node to the Alpha channel, and then using the ShuffleCopy node, the Alpha channel shuffles into all three RGB color channels, making it more usable. Applying a Color Correction node adjusts the shadow color from pure black to a more suitable color value. The addition of another ShuffleCopy node appends the Shadow pass RGBA colors values into a Shadow pass layer on the main comp downstream. Finally, adding a Merge Multiply node for multiplying the Shadow pass RGB channels by the Dark_ env layer, resulting in the application of the Shadow pass to the background, as seen in **Figure 17**, and ready for compositing with the tire RGB channels.

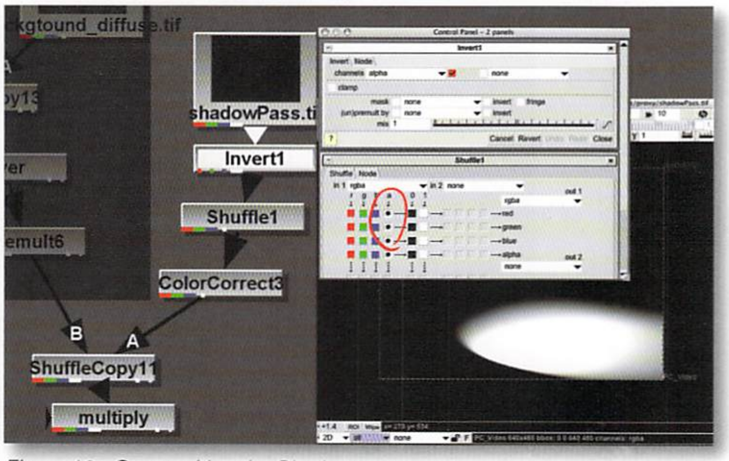


ShuffleCopy13
ShuffleCopy | Node
A in rgb ▾ B in rgba ▾ out 1 Dark_env ▾
out 2 none ▾

Merge4
Merge2 | Node
operation over ▾ sRGB
A channels env_plate ▾ ✓ env_plate.a ▾
B channels Dark_env ▾ none ▾
also merge none ▾ none ▾
Set BBox to union ▾
mask ✓ light.a ▾ invert fringe
mix 1

Premult6
Premult | Node
multiply Dark_env ▾ none ▾
by ✓ rgba.alpha ▾ ✓ invert

Figure 15– Node attributes and flowchart for compositing the background.

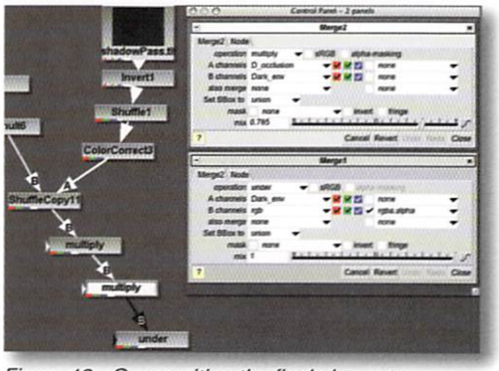


Control Panel – 2 panels

Invert1 — channels: alpha; clamp; mask: none; (un)premult by: none; mix: 1

Shuffle1 — in 1: rgba; in 2: none; out 1: rgba; out 2: none; a → red, green, blue

Figure 16– Compositing the Shadow pass.



Control Panel – 2 panels

Merge2 — operation: multiply; A channels: D_occlusion; B channels: Dark_env; also merge: none; Set BBox to: union; mask: none; mix: 0.785

Merge1 — operation: under; A channels: Dark_env; B channels: rgb, rgba.alpha; also merge: none; Set BBox to: union; mask: none; mix: 1

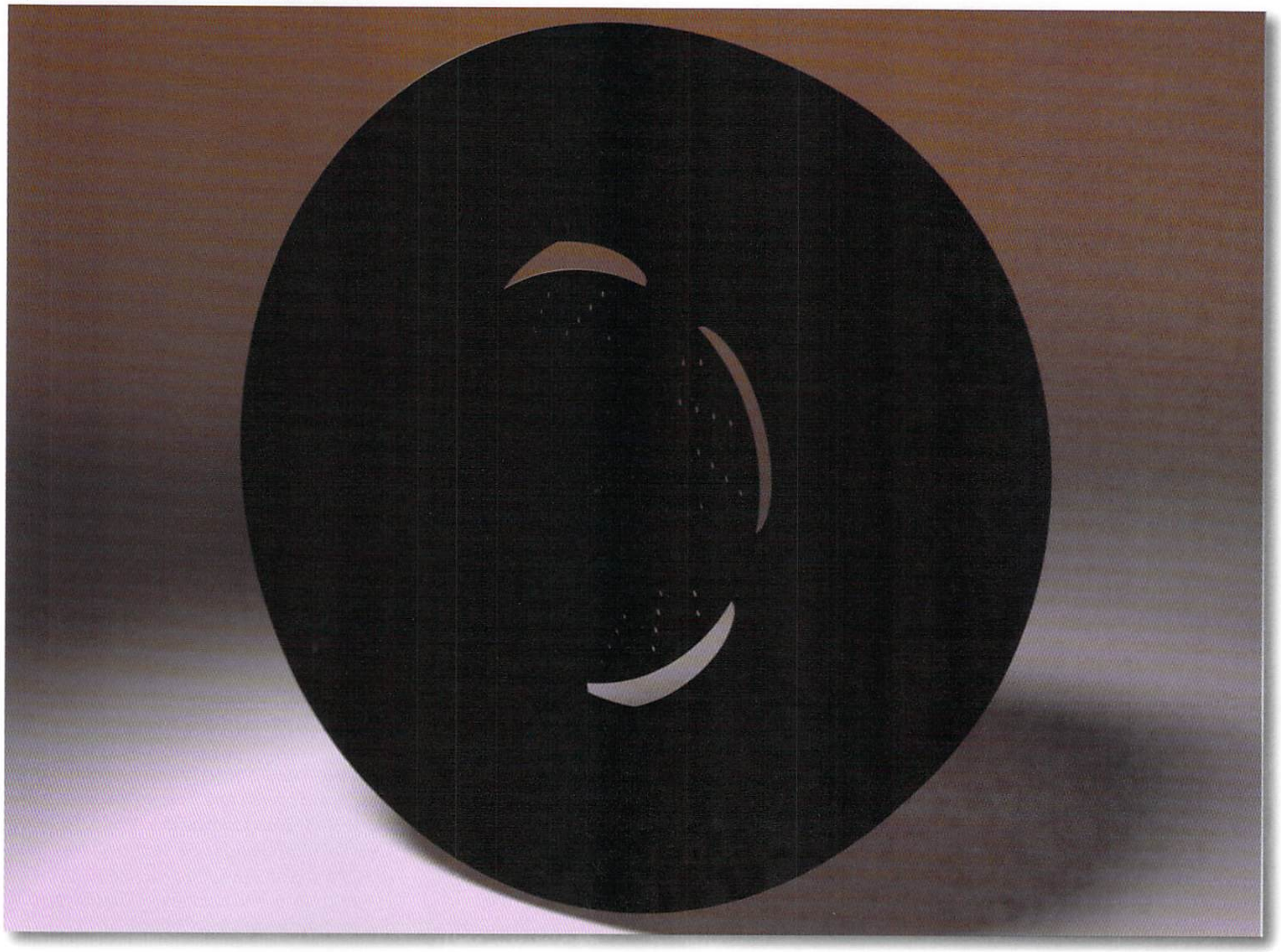
Figure 18– Compositing the final elements.

COMPOSITING THE FINAL ELEMENTS AND ADDING DEPTH OF FIELD

Before compositing the foreground with the background, I add another Merge Multiply node and use it to multiply the initial Diffusive Occlusion pass layer with the background layer, showing again the power of Nuke's architecture. Instead of reconnecting the Occlusion pass to the network, I simply call its layer under the Multiply node, as seen in **Figure 18** in the top section. Since the initial Color pass comp has been maintained as the B input throughout the entire comp, the Diffuse Occlusion layer is still live within the comp tree. The final stage for compositing the foreground with the background is fairly straightforward; another Merge node is added using the Under operation, applying the Dark_env background layer beneath the current RGB layer, which is the tire. Since the B channel outputs the result of the multiply operation to the RGB layer, I preferred using the Under operation, thus enabling the RGB layer to be entered for the B Channels, as seen in **Figure 18** in the lower section.

For creating the illusion of depth of field, I use a fake Z-depth render from Maya; this means that that the Z-depth color values are actually applied to the RGB channels under the Z-depth Read node. Using an expression,

Figure 17– The rendered background plate.



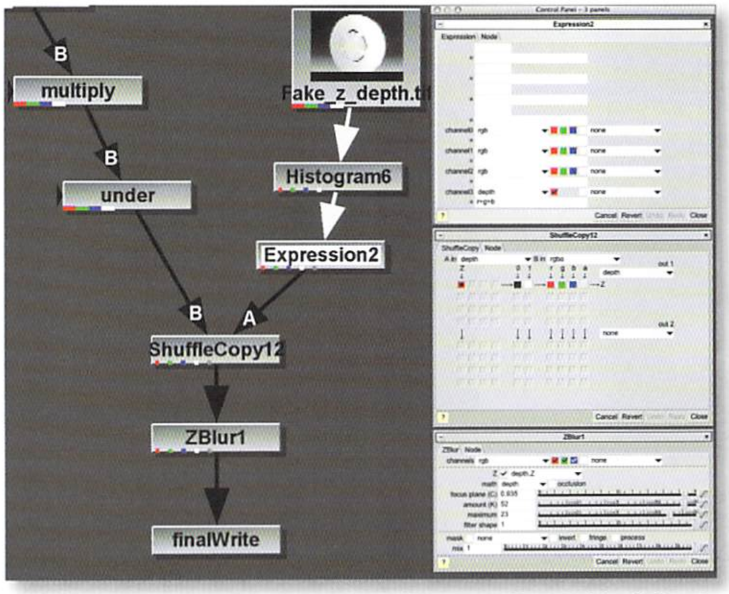


Figure 19– Compositing Z-Depth.

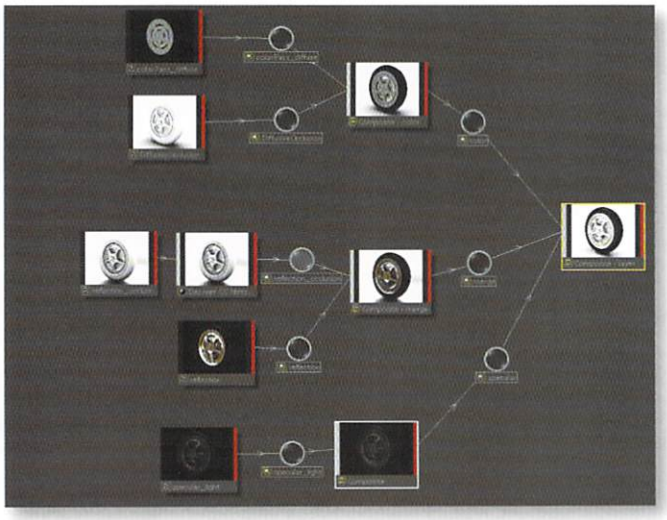


Figure 22– Compositing with nested layer.

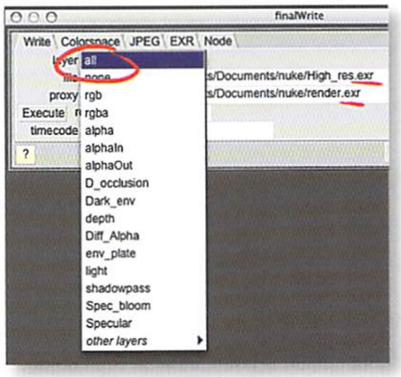


Figure 20– Executing a script.

as seen in **Figure 19**, I force the RGB values into the Z-depth layer, then using the ShuffleCopy, I append that Z-channel to the flowchart Depth layer. I apply the depth of field based on the grayscale values within the Z-channel using the Z-Blur node. The Histogram node, as seen in **Figure 19,** provides more control over the fake Z-depth grayscale ratios before its application to the flowchart, thus greatly increasing the control over the final depth of field effect for the composite. Before the final render, add a few more passes for integrating the Displacement pass for the tire, again showing some of the advantages of using a non-destructive workflow.

For rendering, Nuke provides the Write node, which adds on at the end of the flowchart as seen in **Figure 19** (FinalWrite node). Much like the Read node, the Write node has an option for defining the writing of both high-resolution and Proxy resolution images. Keep in mind that you must disable the proxy mode in the global settings for rendering high-resolution images. Note that if you choose to render using the EXR format, you can select to render **all** from under the Write node **layer** option, thus rendering all the available channels from within the comp at this stage **(Figure 20)**. This method is used for creating a single EXR image file with all the passes within their respective layers and channels, based on the channels available at the end of the comp, not including channels that have not been transferred typically from A input nodes during the comp flowchart. The final render executes using the FinalWrite node. **(See final rendered image on Page 30)**

A FEW FINAL WORDS

One of the advantages of using a node-based workflow is the ability to work without the need for nested layers; for example, a simple composite, as with the Reflection and Color passes combined, would require non-node-based compositors to apply the Color pass with the Diffusive Occlusion in one layer, then on another layer, apply the equivalent for the Reflection pass, and finally, on a third layer, the first two layers are mathematically added. This can become very cumbersome and hard to handle with complex composites, as seen in **Figure 22**. Creating a chain of nodes that pipes into the network in a non-destructive manner when using a non-node-based compositing package is difficult. Typically, the advantage of non-node-based compositors such as Combustion or After Effects is that they work conveniently with the timeline while animating motion graphics, as well as a more user friendly UI.

The node-based workflow is extremely powerful for compositing complex shots for film or commercials, particularly with 3D passes that may be superimposed with live action shots. This sort of compositing is not too commonly used as a motion graphics animation tool; however, the ability to use graphs and key animation is available as with any other compositor. Animation within such a tool is more applicable for camera motion, and within the robust OpenGL 3D environment. Bottom line: for compositing less complex graphics, tools such as After Effects and Combustion are priced appropriately and offer a convenient and somewhat familiar work environment; on the other hand, tools such as Nuke are laying the foundations for the future of compositing. The multi-channel "out of the box" capabilities that Nuke offers deal with tasks that are extremely complex in an easy and natural manner, just as it is natural to animate motion graphics with After Effects. The OpenEXR movement is proving itself in every major film production that requires complex CG shots; this proves that working in a natural way with such elements is not a bonus but a requirement for future compositing tools that wish to remain ahead of the game. For now, I believe that at that level, not only is Nuke ahead of the game, it offers an affordable commercial tool for the general market. I look forward to seeing how Nuke develops in the future, keeping a close eye on its capabilities for integrating 3D with live action effortlessly. Some of the next few tutorials will take apart an OpenEXR image, as well as show how to create a Master node that enables controlling several different settings from within the flowchart.

__Boaz Livny__ has been working with 3D for over 10 years, on film, TV, and multimedia content. He is a technically savvy artist that specializes in modeling, lighting, and rendering, with experience working the entire pipeline. Boaz's studio, www.visionanimations.com, is located in NYC and provides regular services to clients and freelance support for studios. He also teaches the Advanced Maya Course at NYU and is currently involved in establishing an advanced training center in NYC. He is also contributing to the next Sybex, Mastering Maya 7 book.

Extracting Surface Maps From Digital Camera Files

By Stephen Burns
3D Artist, Photographer, Teacher
San Diego, California

With the advent of the digital camera, as well as a reduction in their prices, the popularity of using digital images for surface mapping is increasing. What are the advantages of the new medium?

Well, one noticeable advantage is the instant gratification of previewing exposure changes, white balance changes, ISO, resolution, and format options. You can control all of these in your digital camera. In this first of a two-part article, we will explore a particular format that seems to be a mystery to artists and photographers alike: the RAW file.

To begin with, the concept of relating RAW as a format is incorrect. As 3D artists, we are familiar with a variety of formats, such as IFF, PNG, JPG, TIF, and GIF, to name only a few. In essence, these files are formatted in such a way that they can be read in a variety of graphic alteration programs like Photoshop, Paint Shop Pro, Painter, and so on. RAW, fortunately and unfortunately, can only be read by the proprietary software that came with your camera, and every company has its own.

I say fortunately because the RAW format is simply the raw ones and zeroes used when creating the image. What does this mean to you as an artist? This means that all of the shadow, mid-tone, and highlight detail are all there in the file in 16 Bit mode, provided that your camera allows you to use 16 Bits per Channel of data. Most cameras will only give you 10 or 12.

I say unfortunately because embedded in this file are all of the proprietary information and functions of that particular camera's make and model. These are commands that will not work for any other product. Therefore, other digital programs cannot read this information, and that goes for Photoshop as well. Often, Adobe places free updates online to improve the RAW interface to read the RAW files of the newer cameras.

In this article, we will make the basic color maps for the Cabrillo Monument using RAW files from the Canon Digital Rebel (SLR) with a 6.3 megapixel capability. In the new Photoshop CS 2, we will not only get an introduction to understanding the use of the new RAW interface but also how to use the new Adobe Bridge to preview and make previous RAW changes to the files before you open them into Photoshop. Next issue, you will learn how to extract the Diffusion, Specular, and bump maps in CS2. Let's have some fun!

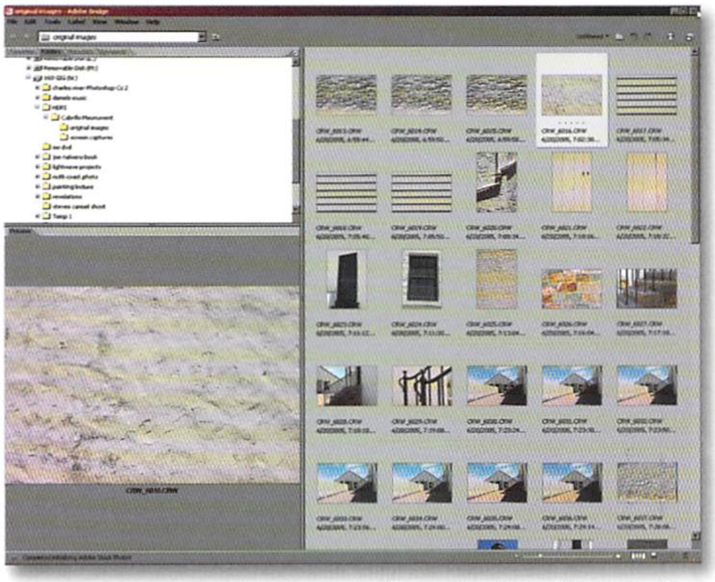


Figure 1– Adobe Bridge Interface

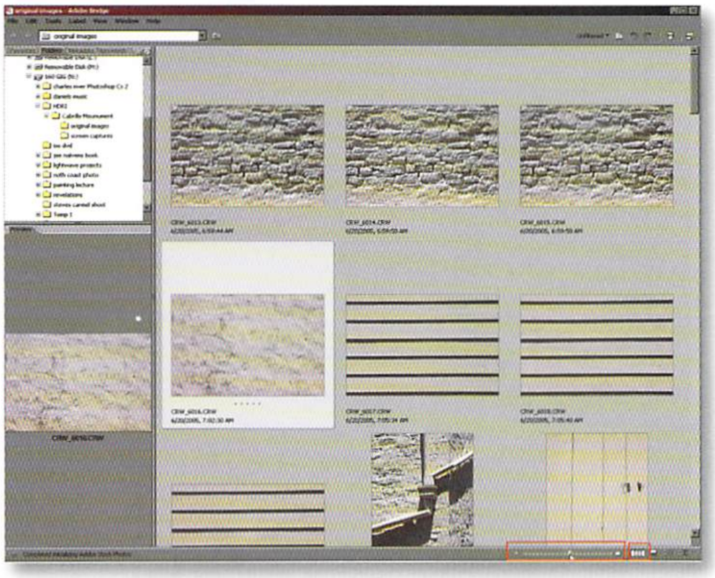


Figure 4– Resizing your thumbnails

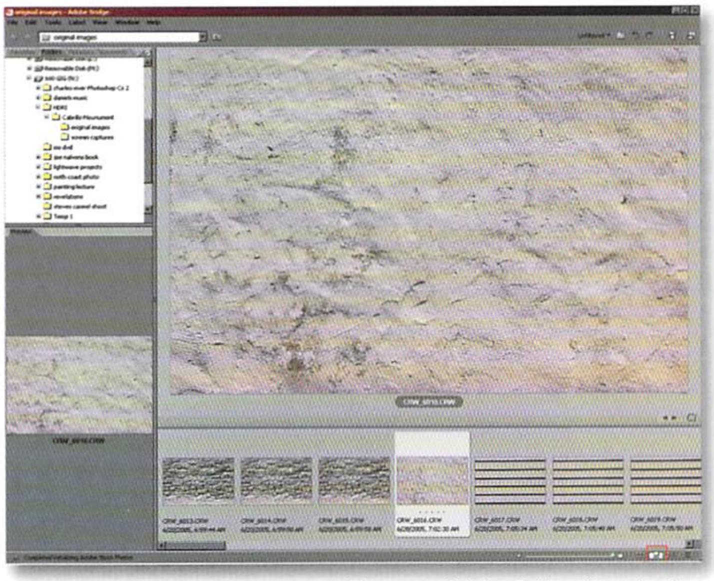
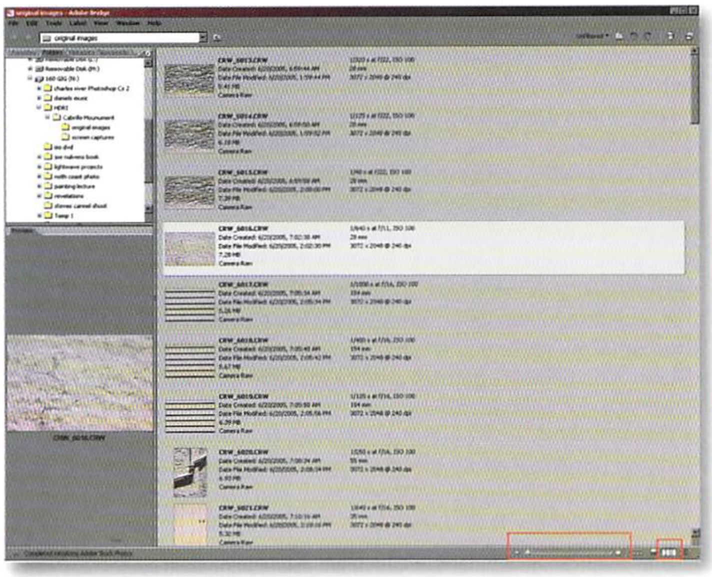
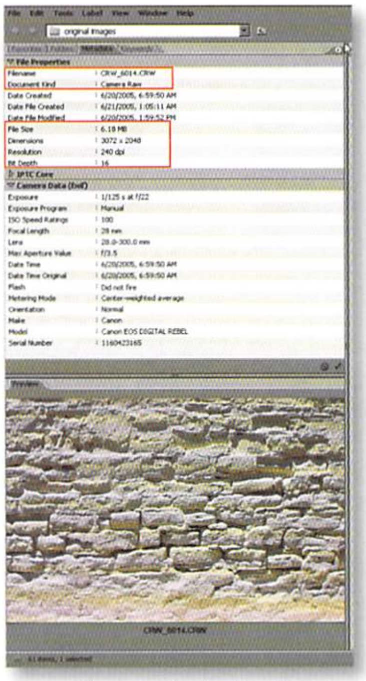


Figure 2– Adobe Bridge filmstrip view

Figure 3– Adobe Bridge thumbnails with detail





File Properties	
Filename	CRW_6014.CRW
Document Kind	Camera Raw
Date Created	6/20/2005, 6:59:50 AM
Date File Created	6/21/2005, 1:05:11 AM
Date File Modified	6/20/2005, 1:59:52 PM
File Size	6.18 MB
Dimensions	3072 x 2048
Resolution	240 dpi
Bit Depth	16
IPTC Core	
Camera Data (Exif)	
Exposure	1/125 s at f/22
Exposure Program	Manual
ISO Speed Ratings	100
Focal Length	28 mm
Lens	28.0-300.0 mm
Max Aperture Value	f/3.5
Date Time	6/20/2005, 6:59:50 AM
Date Time Original	6/20/2005, 6:59:50 AM
Flash	Did not fire
Metering Mode	Center-weighted average
Orientation	Normal
Make	Canon
Model	Canon EOS DIGITAL REBEL
Serial Number	1160423165

Figure 5– Reviewing Metadata

INTRODUCTION TO ADOBE BRIDGE

After you photograph your textures, you will preview them in Adobe Bridge, formerly called Browse. It is now a separate program that will recognize any file formats and open them in their respective programs. **Figure 1** is an example of my texture previews for this project.

Figures 2 thru 4 show some alternative views of the textures.

Also, be aware that you can resize your thumbnails using the slider bar on the lower right-hand corner of the interface.

Let's start with the basic texture of the Cabrillo Monument lighthouse. If you click on the stone texture thumbnail, a preview displays automatically. Click on the metadata folder on the top left corner and take a look at the physical properties of the file. This location will give us all of the details as to what went into creating this image. The portions we will be most interested in are the File Size, Dimension, Resolution, Bit Depth, and the File Format. **(Figure 5)**

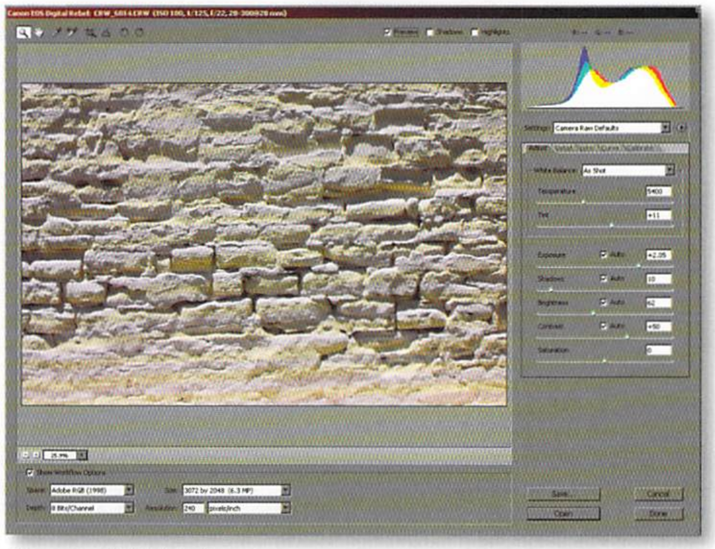


Figure 6– RAW Interface

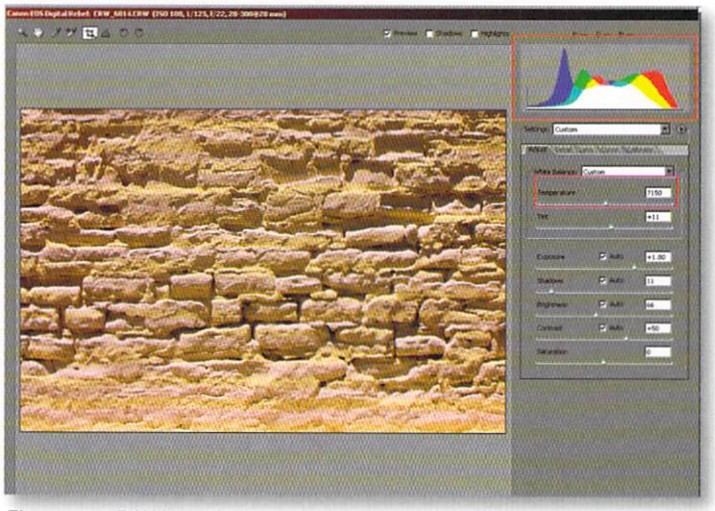


Figure 10– RAW Interface color temperature warming options

THE RAW INTERFACE

Figure 6 shows an overview of the RAW interface. You are looking at its basic preview pane to the left, your controls to the right, and workflow and resizing options on the lower left.

STEP 1

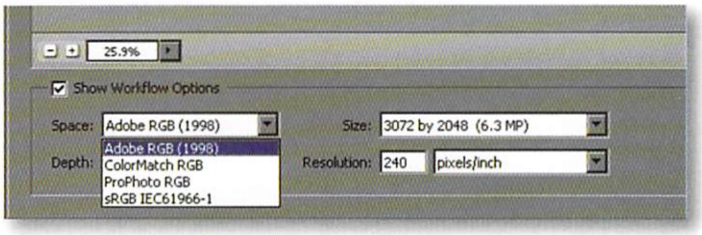
Click on each of the drop menus in the workflow area to preview your options for Color Space, Bit Depth, Sizing, and Resolution. **(Figures 7-9)**

STEP 2

Take a look at the Color Temperature slider under the Adjust tab and slide it to the right and left. Notice that as you slide to the right, your image becomes warmer (yellow), and as you drag in the opposite direction, your image becomes cooler (blue). The histogram in the top right is giving you an update as to how all of the colors are responding to any and all adjustments in the RAW interface. **(Figures 10 and 11)**

STEP 3

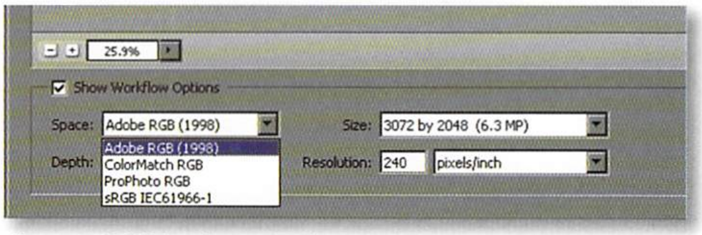
Now experiment with the Tint slider and watch how you can control magentas and greens. This is great for situations where textures are photographed near fluorescent lighting situations. **(Figures 12 and 13)**


25.9%

☑ Show Workflow Options

Space: Adobe RGB (1998)
Adobe RGB (1998)
ColorMatch RGB
ProPhoto RGB
sRGB IEC61966-1
Depth:
Size: 3072 by 2048 (6.3 MP)
Resolution: 240 pixels/inch

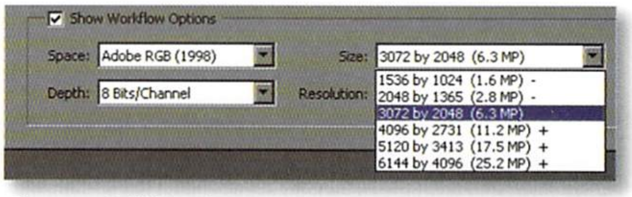
Figure 7– RAW Interface color space options


25.9%

☑ Show Workflow Options

Space: Adobe RGB (1998)
Adobe RGB (1998)
ColorMatch RGB
ProPhoto RGB
sRGB IEC61966-1
Depth:
Size: 3072 by 2048 (6.3 MP)
Resolution: 240 pixels/inch

Figure 8– RAW Interface color depth options


☑ Show Workflow Options

Space: Adobe RGB (1998)
Depth: 8 Bits/Channel
Size: 3072 by 2048 (6.3 MP)
1536 by 1024 (1.6 MP) -
2048 by 1365 (2.8 MP) -
3072 by 2048 (6.3 MP)
4096 by 2731 (11.2 MP) +
5120 by 3413 (17.5 MP) +
6144 by 4096 (25.2 MP) +
Resolution:

Figure 9– RAW Interface sizing and resolution

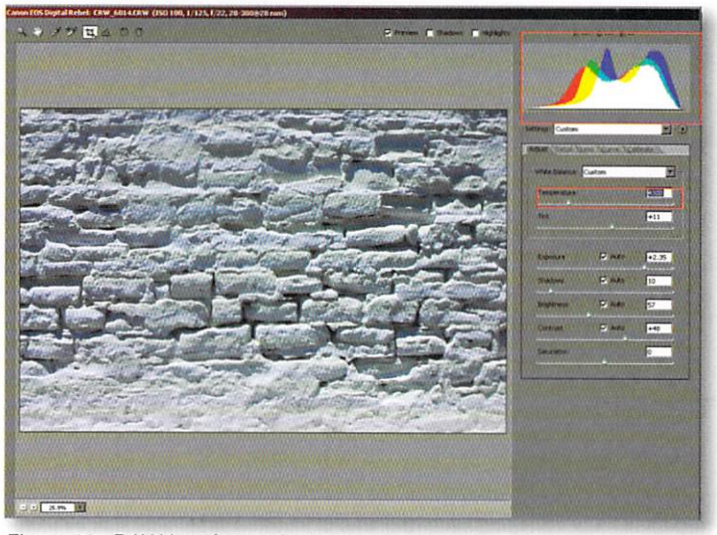
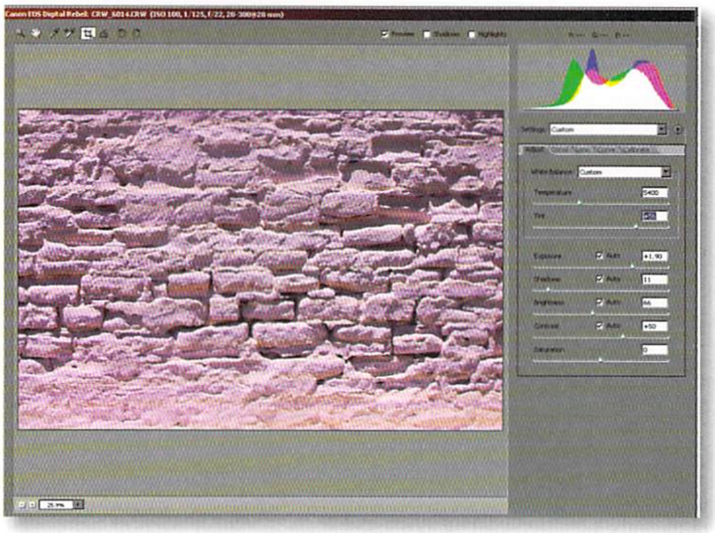


Figure 11– RAW Interface color temperature cooling options

Figure 12– RAW Interface color tinting toward magenta



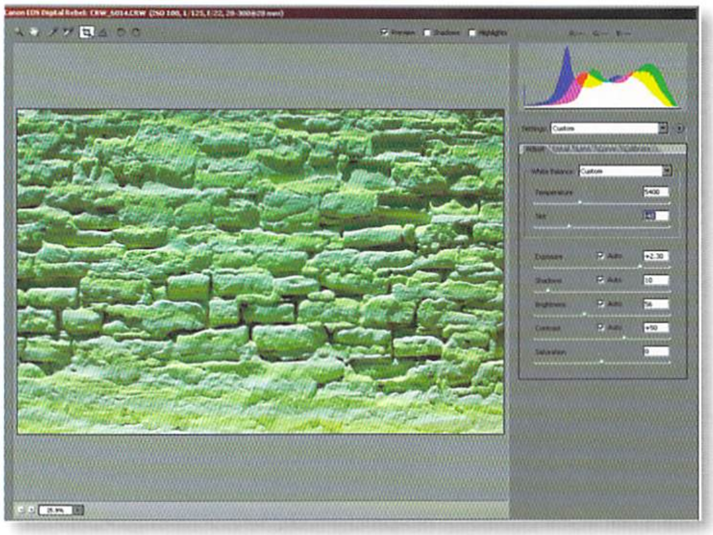


Figure 13– RAW Interface color tinting toward green

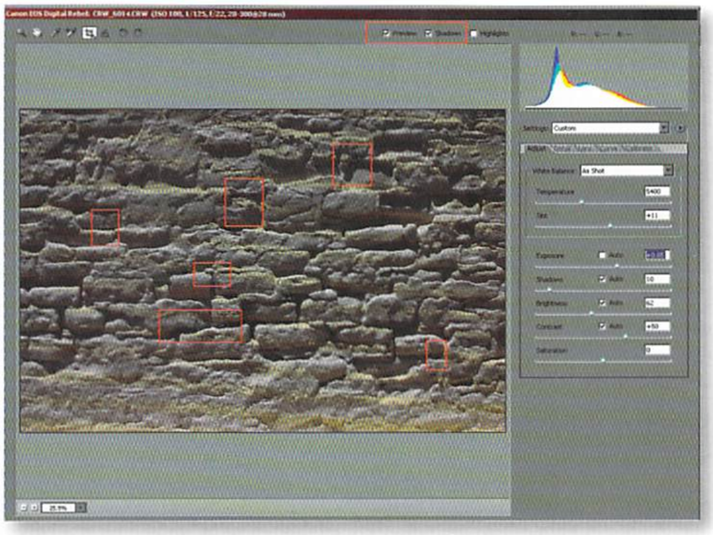


Figure 14– RAW Interface Exposure Option toward the shadows

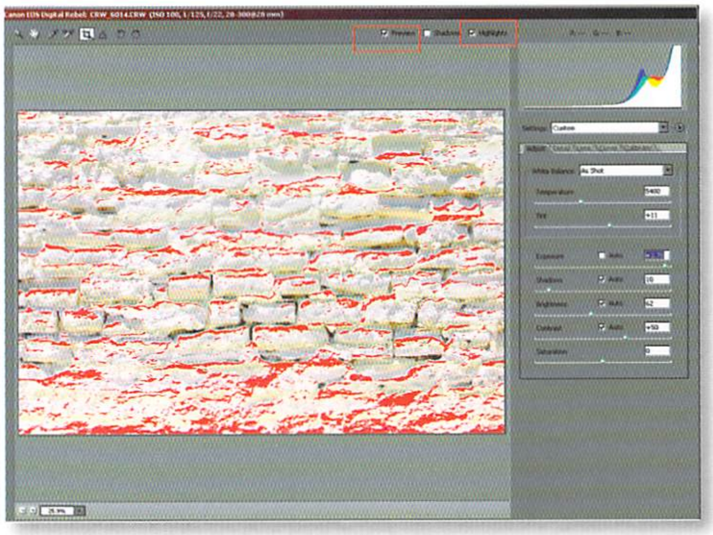


Figure 15– RAW Interface Exposure Option toward the highlights

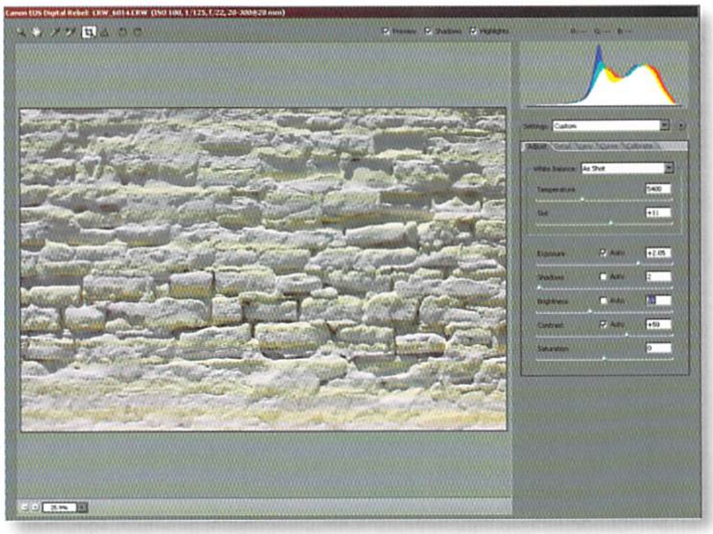
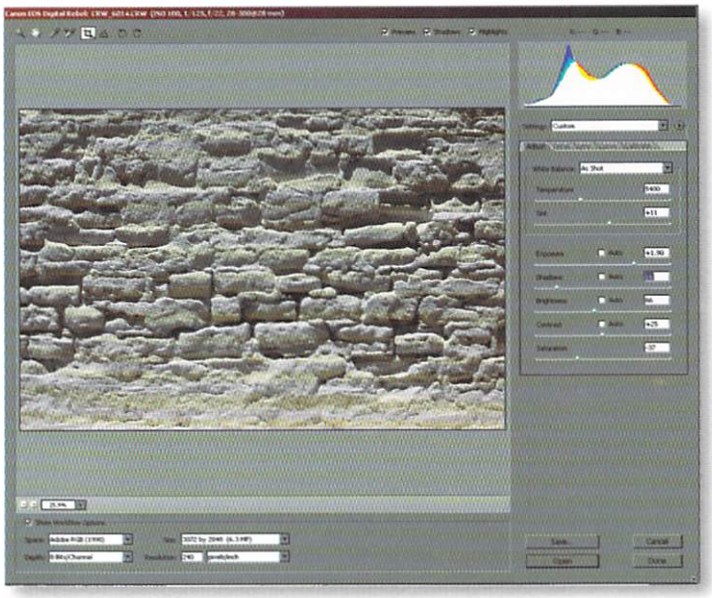


Figure 16– RAW Interface Exposure Option properly adjusted

Figure 17– Results of the Shadows slider



STEP 4

Since we are dealing with the raw data of this file, there's more information to play with than if it were formatted. The Exposure slider will help you make adjustment to any over or underexposed images. Click the Preview and Shadows tab in the top right and adjust the slider so that your image becomes darker. Look closely at the shadow region. If a blue-colored tint previews in these areas, then your shadows will have no detail. Since detail is an important matter for texturing, we should use this option regularly. **(Figure 14)**

Now do the same thing and adjust your image so that it goes almost white. Make sure the Preview and Highlights boxes are checked. Any area that is red will now have detail, so adjust it accordingly. **(Figure 15)**

Figure 16 shows the properly adjusted image with details in both shadows and highlights.

STEP 5

In this same panel, you have the ability to control your shadow and highlight detail independently using the Shadows and Brightness sliders. **(Figures 17 and 18)**

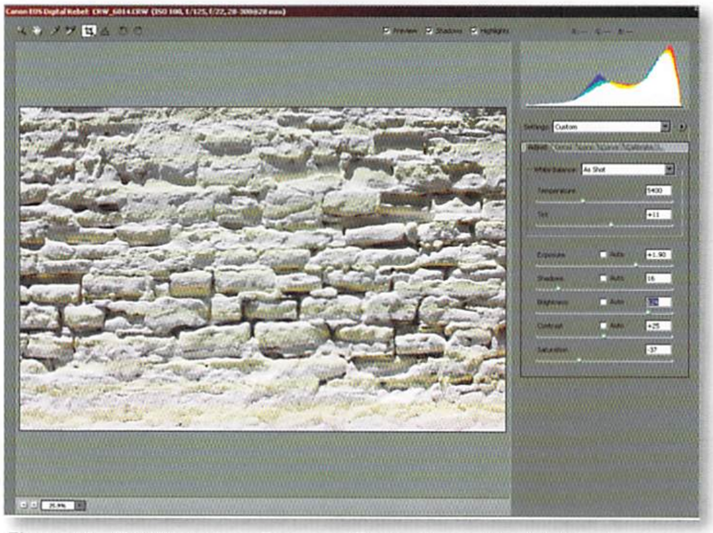


Figure 18– RAW Interface Exposure Option toward the highlights

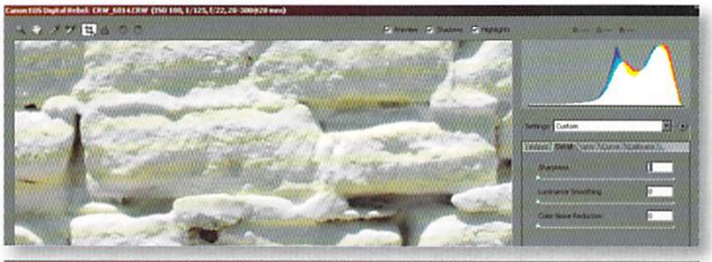
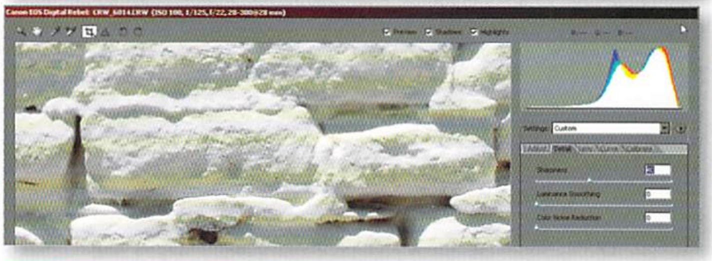
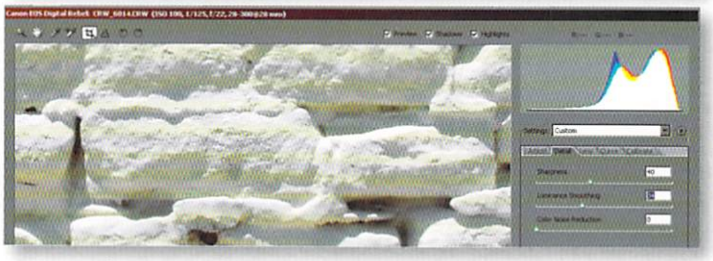




Figure 19 (Above Top)– Image without Sharpening
Figure 20 (Above Middle)– RAW image with Sharpening applied
Figure 21– (Above Bottom) RAW image with Luminance Sharpening applied

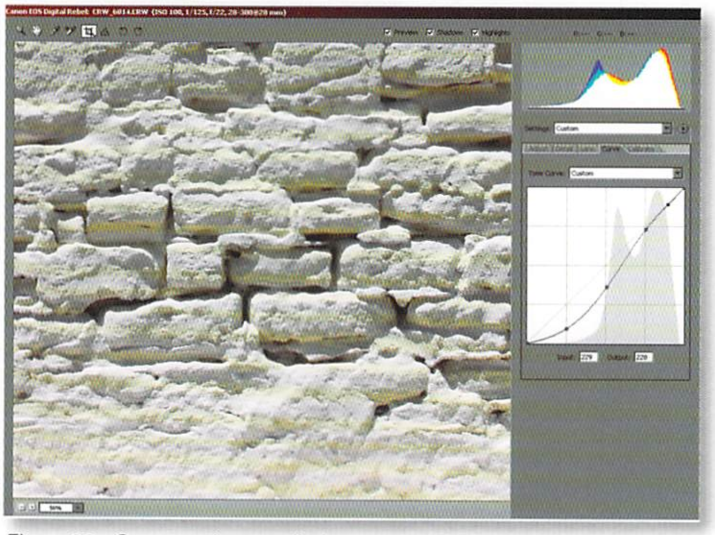


Figure 22– Curve settings applied

STEP 6

Click on the Details tab and experiment with your Sharpening options. You not only have an overall Sharpening slider, but you can sharpen the Luminance, which is your tonal gray information, as well as your Color, independently. **(Figures 19-21)**

STEP 7

If you are familiar with the use of Curves in Photoshop then click on the Curves tab and alter your tonality there. **(Figure 22)**

Next, save you settings for later application through Bridge. **(Figure 23)**

In this example, the settings are titled "texture mapping." **(Figure 24)**

STEP 8

Just to show you how easy it is to apply these RAW settings in Bridge, right-click on a recently edited texture and click "Copy Camera Raw Settings." In this example, it's the wall detail for our architecture. **(Figure 25)**

Next, click on another texture and right-click and select "Paste Camera Raw Settings." **(Figure 26)**

A dialogue box asks you what information that you would like to transfer. Select all of it. **(Figure 27)**

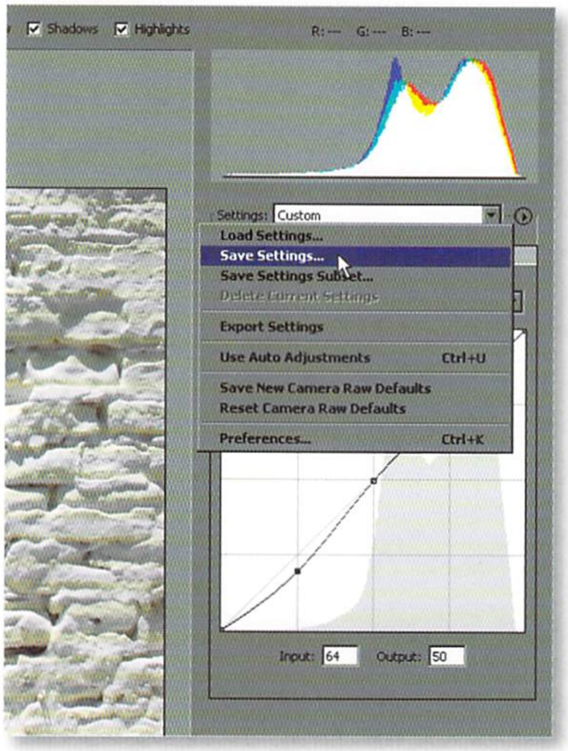
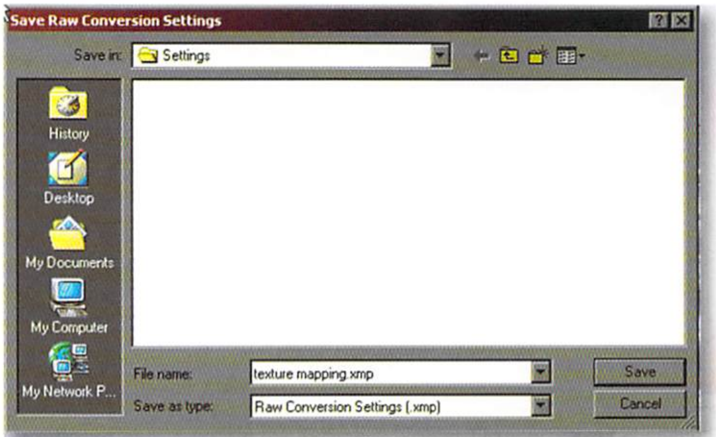


Figure 23– Saving Curve settings

The new thumbnail has been altered to reflect all of the changes made to the previous images. **(Figure 28)**

With both images opened up in Photoshop, we can see the results side-by-side. **(Figure 29)**



File name: texture mapping.xmp

Save as type: Raw Conversion Settings (.xmp)

Figure 24– RAW image with Luminance Sharpening applied

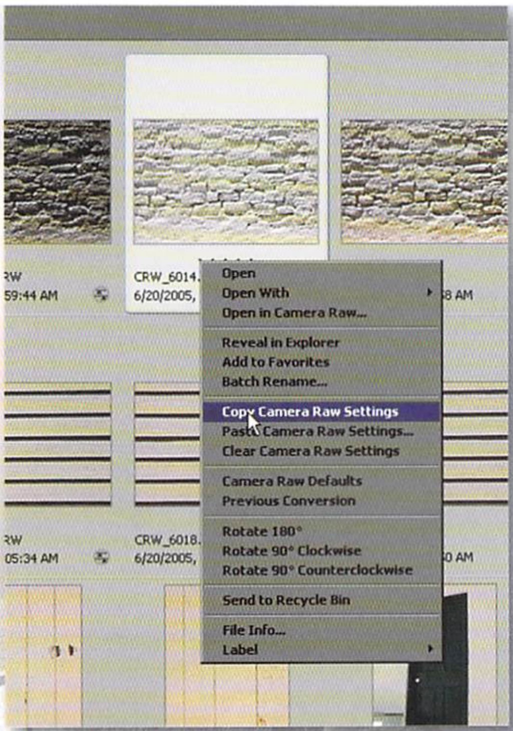


Figure 25– Copy Camera Raw settings

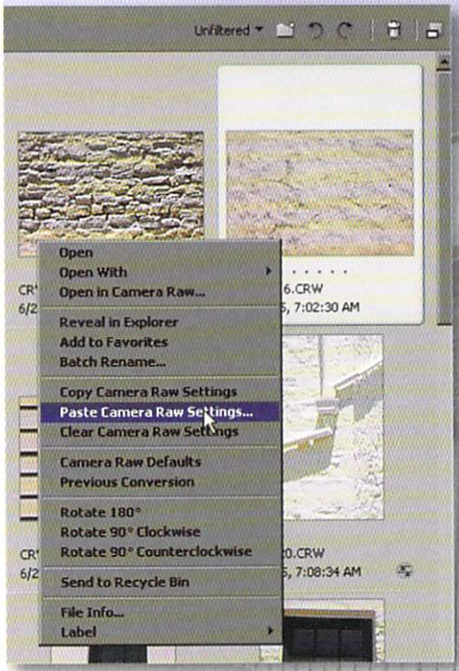
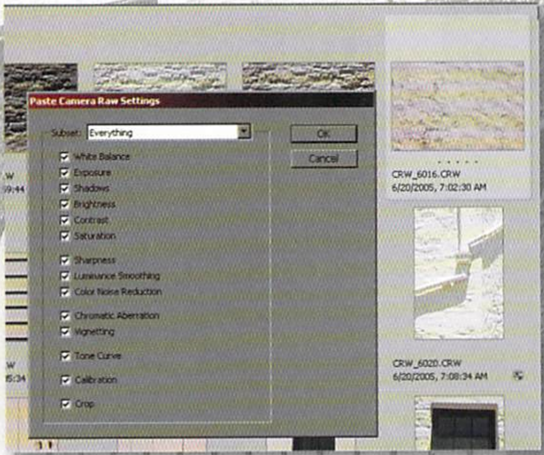


Figure 26– Paste Camera Raw setting



Subset: Everything

- [✓] White Balance
- [✓] Exposure
- [✓] Shadows
- [✓] Brightness
- [✓] Contrast
- [✓] Saturation
- [✓] Sharpness
- [✓] Luminance Smoothing
- [✓] Color Noise Reduction
- [✓] Chromatic Aberration
- [✓] Vignetting
- [✓] Tone Curve
- [✓] Calibration
- [✓] Crop

Figure 27– Pasting Dialogue settings

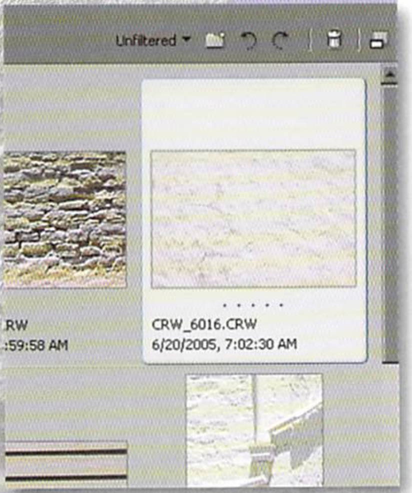
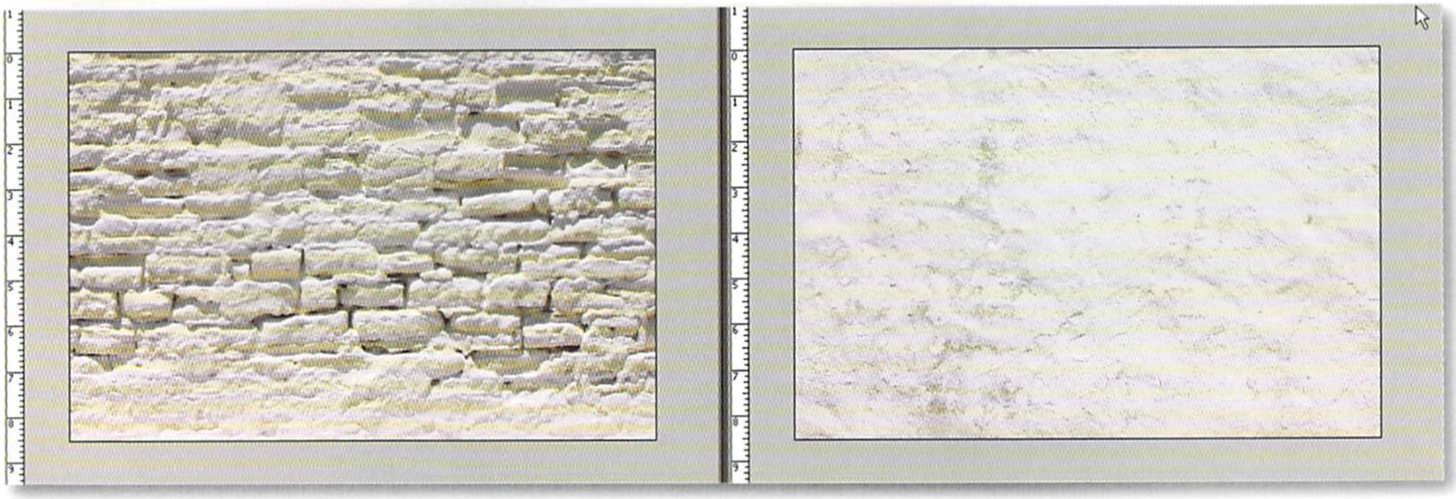


Figure 28– Results of pasting settings

Figure 29– Results of pasting settings



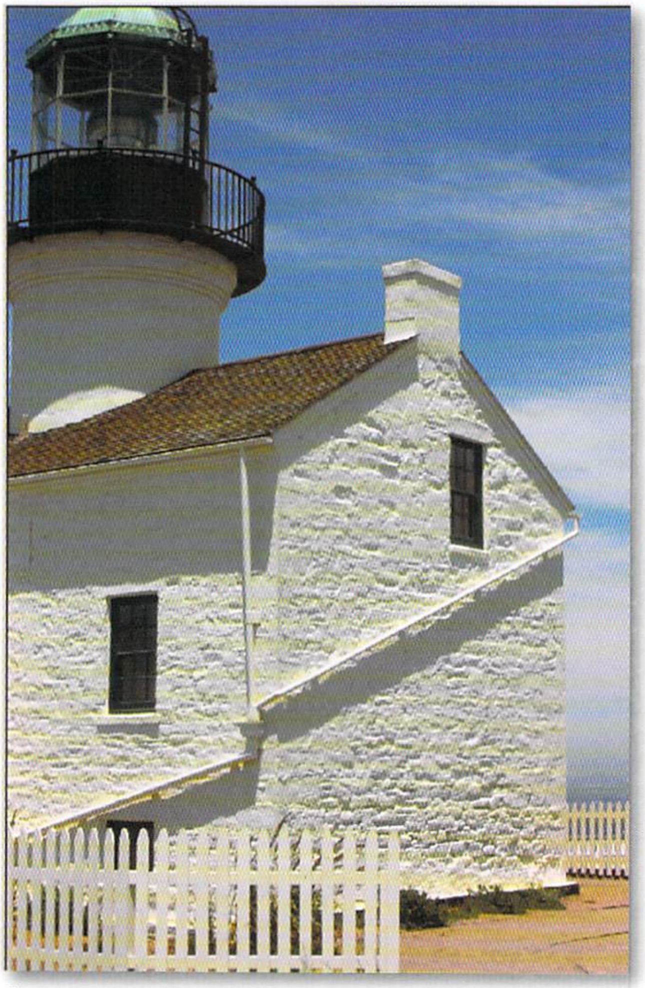


Figure 30– Lighthouse detail

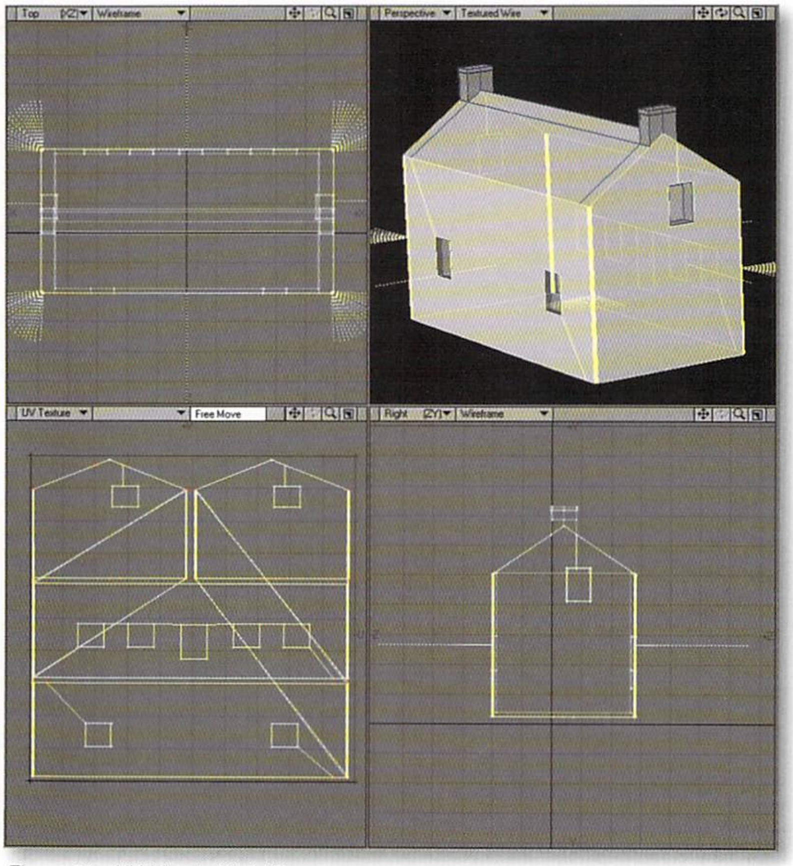


Figure 31– UV Map in LightWave

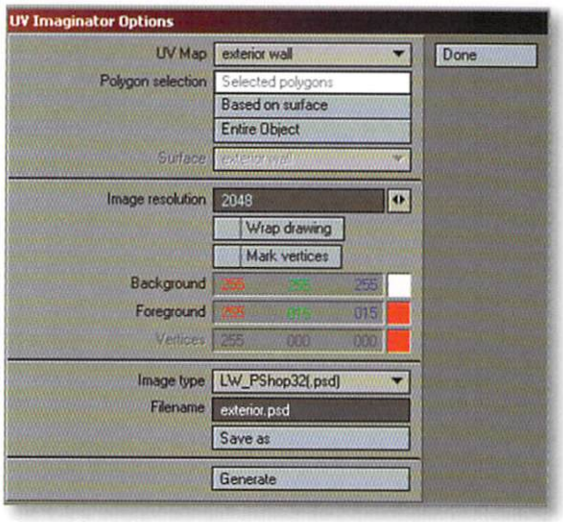
CREATING THE EXTERIOR TEXTURE

Figure 30 is an example of the original stone texture of the lighthouse.

Let's go into LightWave and manage our textures. **Figure 31** shows an example of the basic shape and UV map for the exterior of the lighthouse.

STEP 1

I used a free plug-in called UV Imaginator for creating the image templates for the exterior. I made the resolution 2048 X 2048. Make sure that the foreground color is not going to conflict with the overall color of your texture. In this example, I chose Red for the foreground and white for the background. Next, set the image type to Photoshop's native PSD format. **(Figure 32)**



UV Imaginator Options

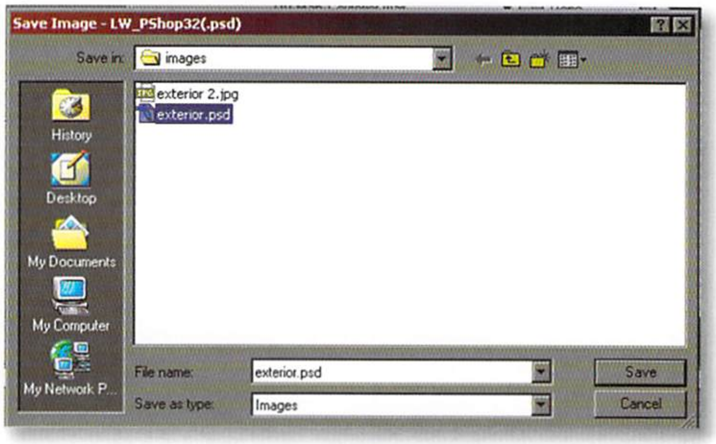
UV Map: exterior wall
Polygon selection: Selected polygons / Based on surface / Entire Object
Surface: exterior wall
Image resolution: 2048
Wrap drawing
Mark vertices
Background: 255 255 255
Foreground: 255 015 015
Vertices: 255 000 000
Image type: LW_PShop32(.psd)
Filename: exterior.psd
Save as
Generate
Done

Figure 32– UV Imaginator

STEP 2

After clicking "Save," place your file into the images folder. **(Figure 33)**

Figure 33– Saving UV Map template



Save Image - LW_PShop32(.psd)

Save in: images
exterior 2.jpg
exterior.psd
History
Desktop
My Documents
My Computer
My Network P...
File name: exterior.psd
Save as type: Images
Save
Cancel

STEP 3

In Photoshop, open the template. Access *Image>Image Size* so that you will get an idea as to the resolution to the file that you are working with. **(Figure 34)**

STEP 4

Access *Windows>Arrange>Tile Vertically* to view all three images on your desktop. **(Figure 35)**

STEP 5

Drag the stone texture into the template file. Duplicate the

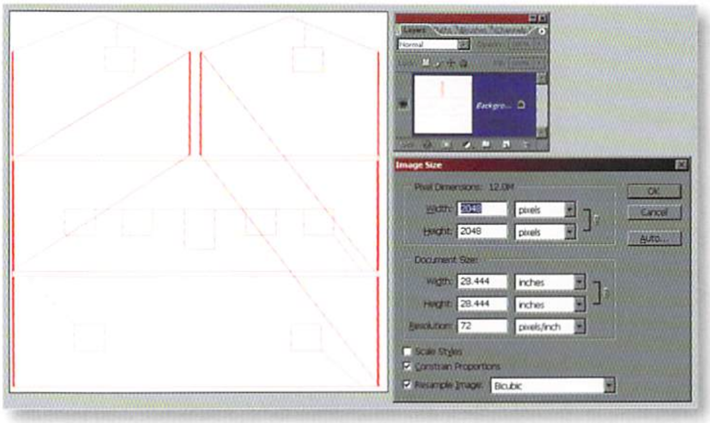


Image Size

Pixel Dimensions: 12.0M

Width: 2048 pixels

Height: 2048 pixels

Document Size:

Width: 28.444 inches

Height: 28.444 inches

Resolution: 72 pixels/inch

Scale Styles

[✓ Constrain Proportions]

[✓ Resample Image:] Bicubic

Figure 34– Image size information

template layer and place it on top of the texture. Next, change its blend mode to Multiply – this makes the whites transparent and preserves the red lines. **(Figure 36)**

STEP 6

Our main concern is that the texture is in proper proportion to the lighthouse template. Resize the stone detail until it seems to have the right proportions, as shown in the photograph of the lighthouse. **(Figure 37)**

STEP 7

Select the texture and define it as a pattern *(Edit>Define Pattern)*. **(Figure 38)**

Next, save the pattern and call it “Brick Texture.” **(Figure 39)**

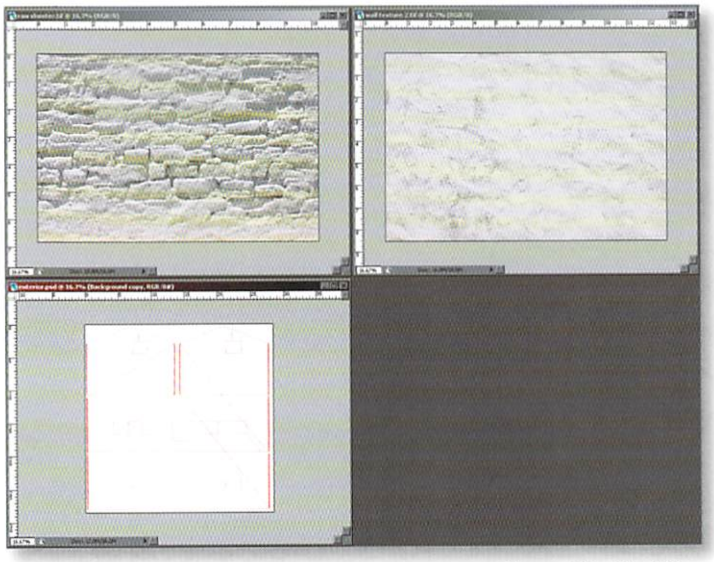


Figure 35– Tiled files

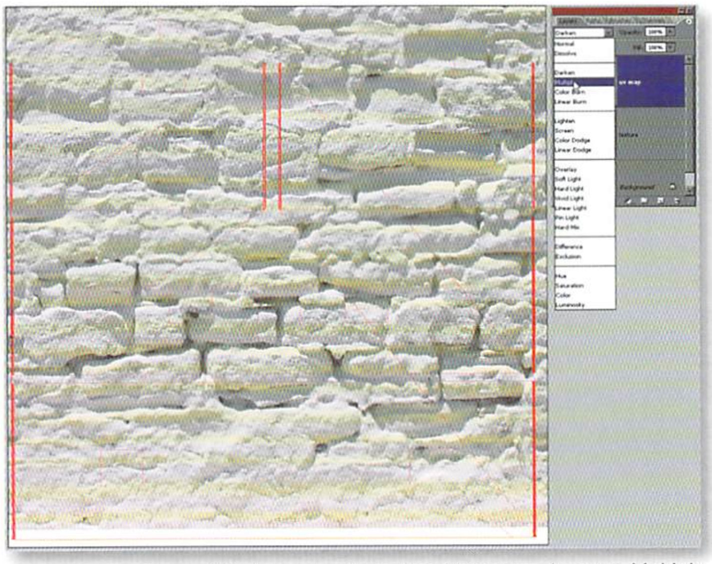


Figure 36– Layer duplicated and blend mode set to Multiply

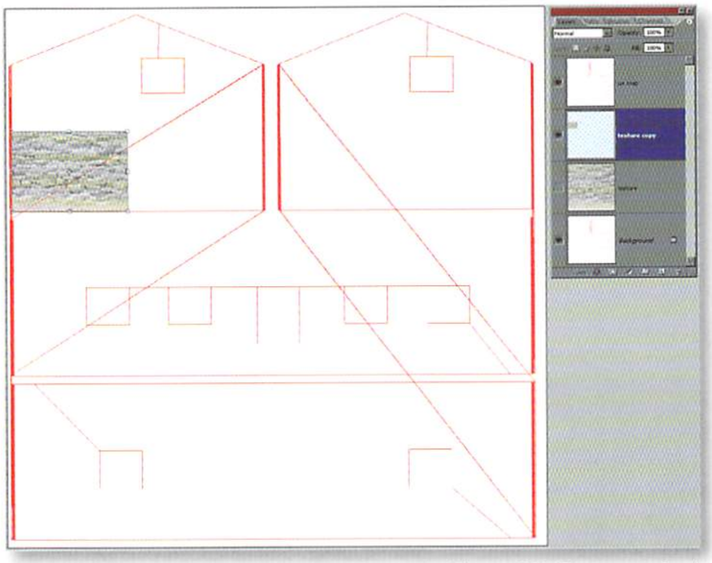


Figure 37– Texture resized

Figure 38– Pattern is defined

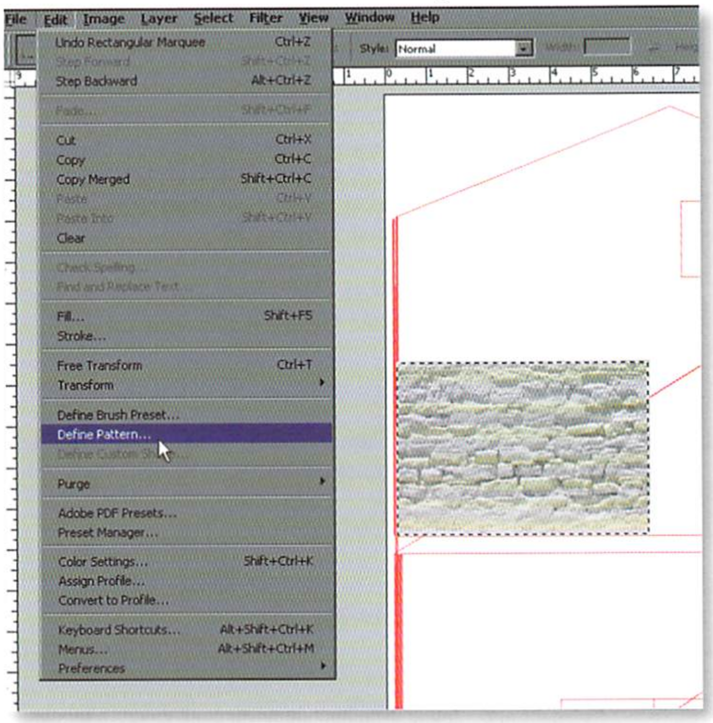
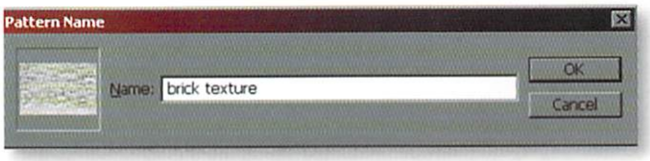


Figure 39– Pattern is saved



Pattern Name

Name: brick texture

STEP 8

Turn off the template layer so that all you can view is the texture. Select the Fill tool on the Tools Palette and access your options bar to choose your new texture. Create a new layer and fill it with the texture. **(Figure 40)**

STEP 9

We are going to make a seamless texture with our pattern. Access the Patch tool and make a selection around one of the sharp edge divisions. **(Figure 41)**

Next, drag the selection to an unblemished portion of the image and notice that a preview is automatically provided as to what section of the unblemished image is being placed. Once you release your mouse, the integrity of the original image is maintained and blended with the new one that you selected. This will give the texture a more consistent look overall. **(Figure 42)**

The results on half of the image should look like **Figure 43**.

STEP 10

Apply the same technique to the vertical lines **(Figures 44 and 45)**

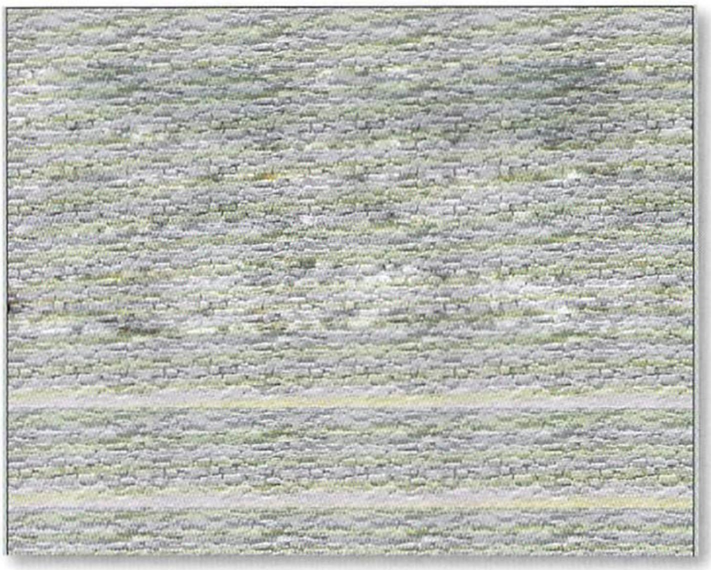


Figure 43– Patch tool applied

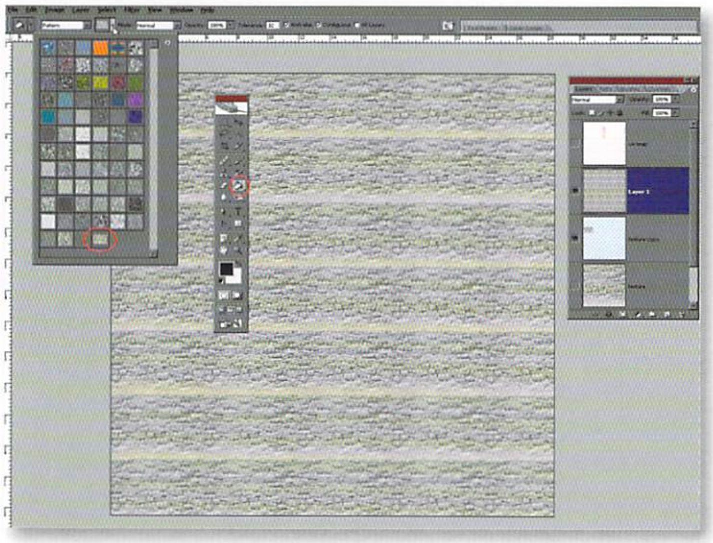


Figure 40– Layer is filled with pattern

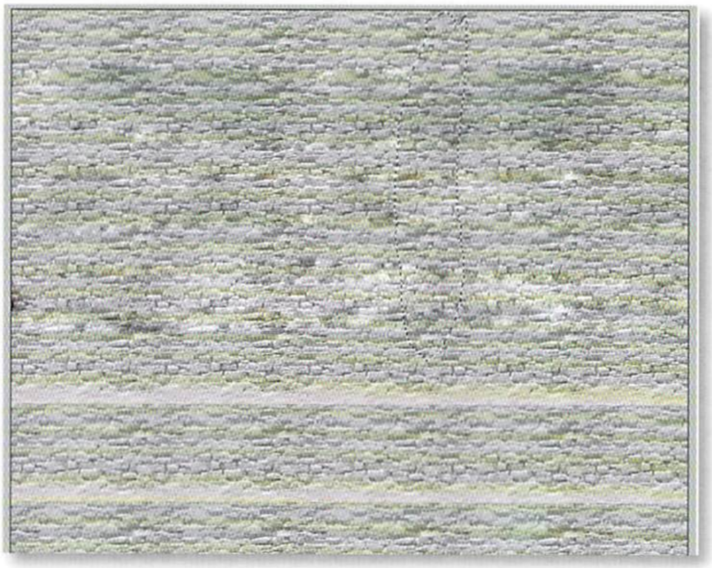


Figure 44– Patch tool applied vertically

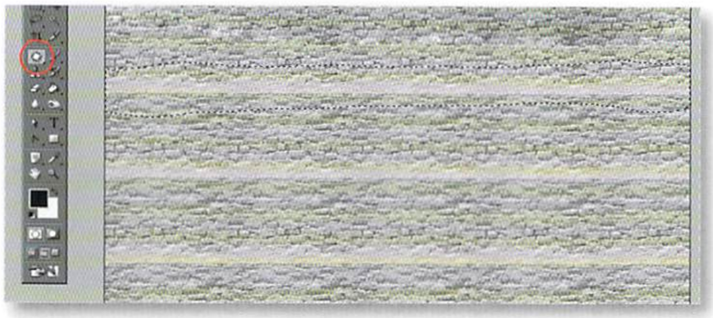


Figure 41– Layer is filled with pattern

Figure 45– End results

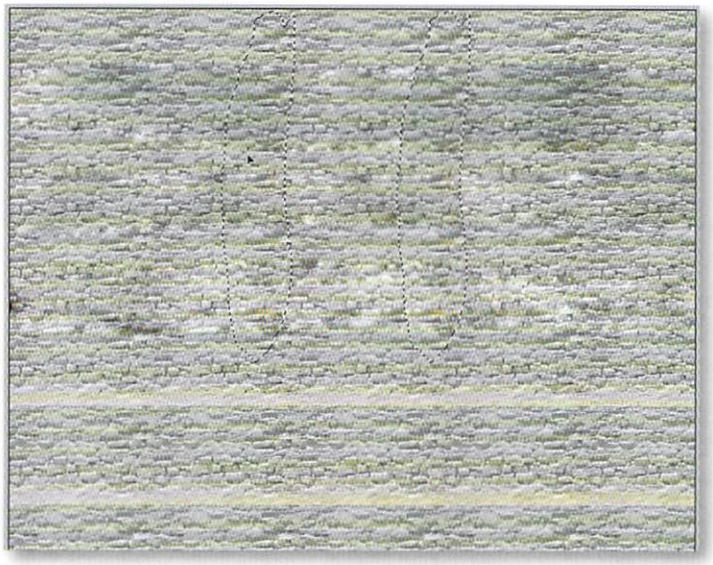
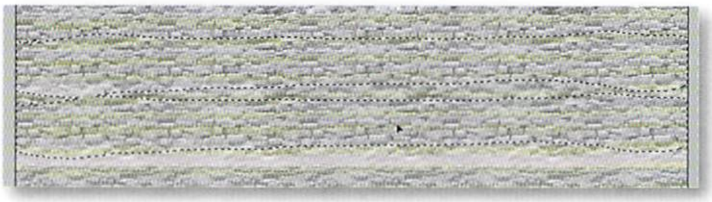


Figure 42– Patch tool applied



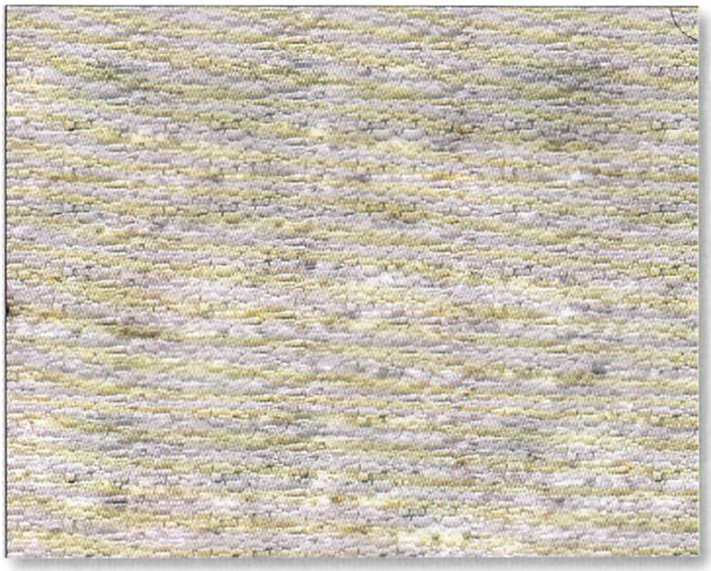


Figure 46– Patch Tool applied completely

STEP 11

Continue on with the same technique until your overall image is complete. This particular approach to creating the basic texture was chosen to assist in each view of the wall having a slightly different pattern. Having the same tiled texture everywhere is a little monotonous. This approach breaks it up a little. **(Figure 46)**

Next the other texture shown in figure 35 is placed on top of the stone. When its blend mode is set to Soft Light the tonal information of the complete texture is much more evenly distributed. This will aid greatly in making it seamless. Now merge the two layers together. **(Figure 47)**

STEP 11

To make sure that the texture is tileable, access *Filter>Other>Offset*. Divide your dimensions by half and place the results in for the width and height. **(Figure 48)**

Continue with the technique of covering up the splices until the texture is uniform. **(Figure 49)**

Turn on the template layer to view the texture in relation to the template. **(Figure 50)**

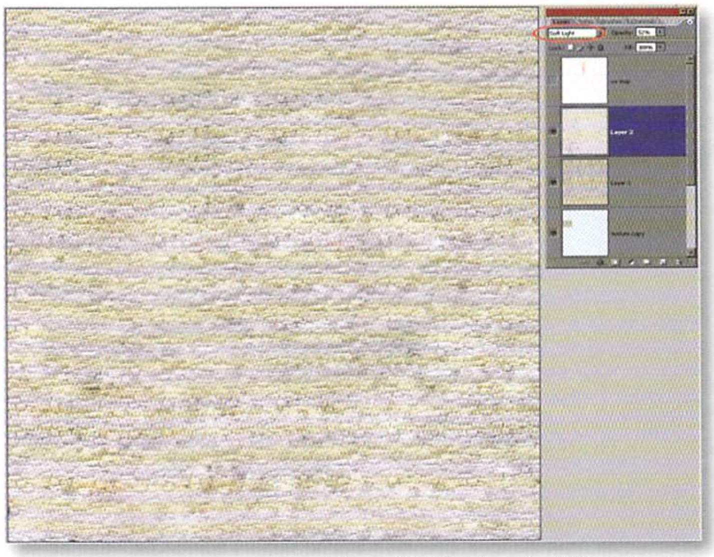

Soft Light
Opacity: 52%
Fill: 100%
uv map
Layer 2
Layer 1
texture copy

Figure 47– Overlay of additional texture.

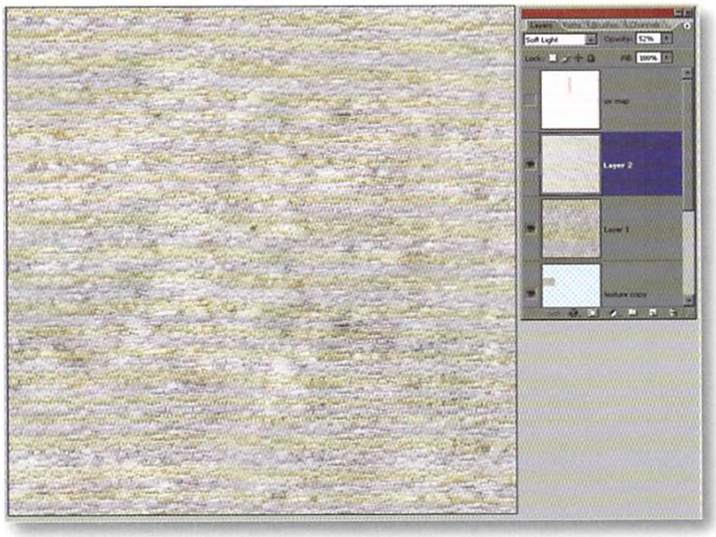

Soft Light
Opacity: 52%
Fill: 100%
uv map
Layer 2
Layer 1
texture copy

Figure 49– Editing after Offset tool is applied

Figure 48– Offset tool applied

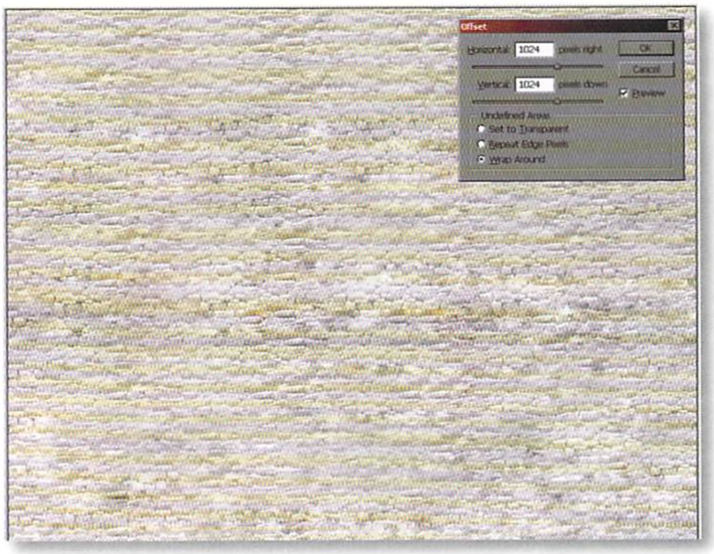
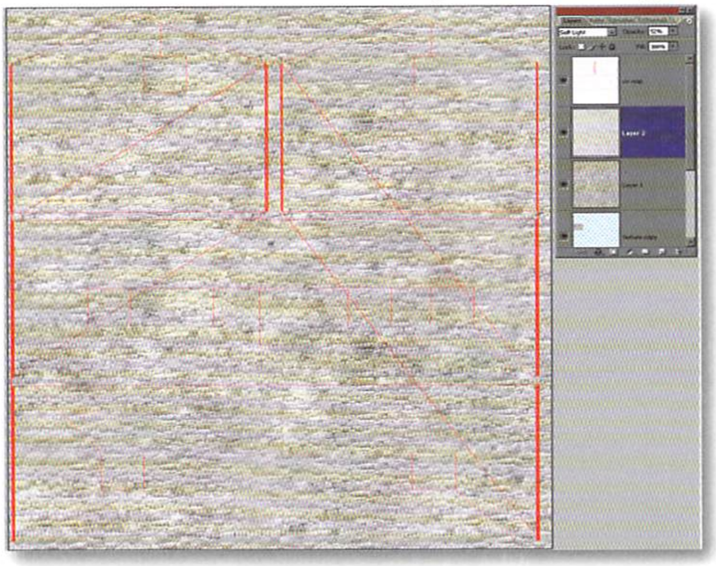

Offset
Horizontal: 1024 pixels right
Vertical: 1024 pixels down
[✓ Preview]
Undefined Areas
Set to Transparent
Repeat Edge Pixels
[✓ Wrap Around]

Figure 50– Offset tool applied


Soft Light
Opacity: 52%
Fill: 100%
uv map
Layer 2
Layer 1
texture copy

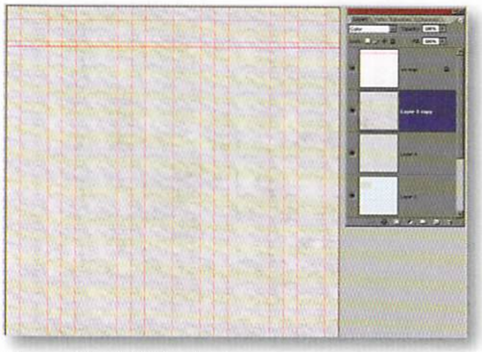


Figure 51– UV Map for tower

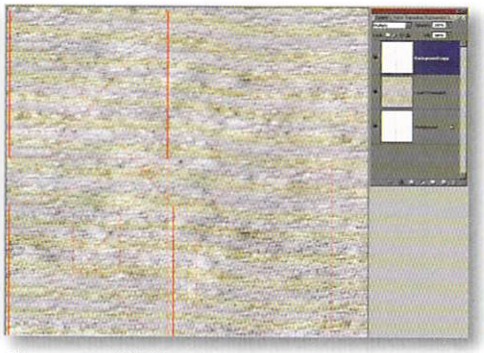


Figure 52– UV Map for secondary room

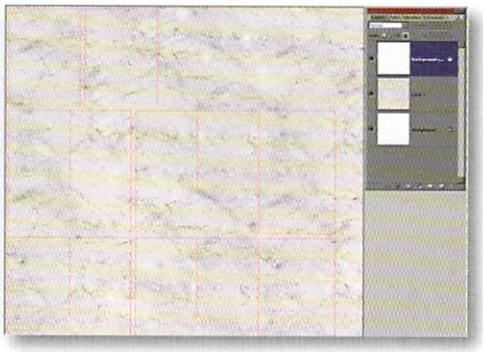


Figure 53– UV Map for chimney

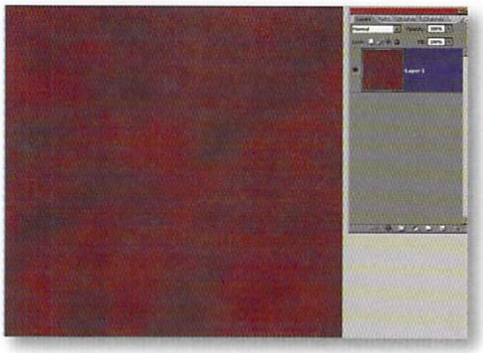


Figure 54– Layer filled with clouds

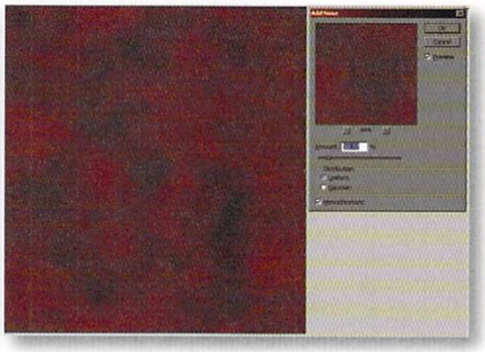


Figure 55– Noise applied

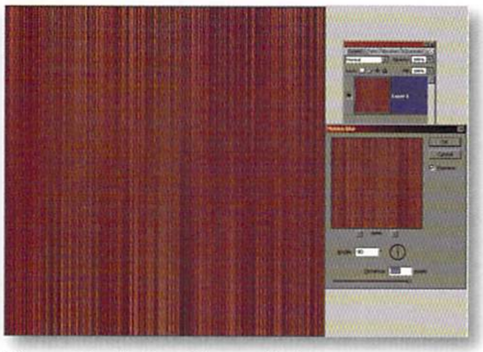


Figure 56– Motion Blur applied

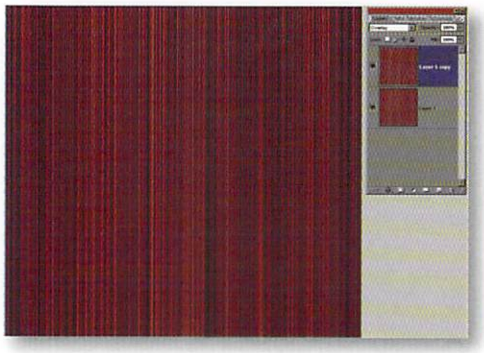


Figure 57– Motion Blur applied

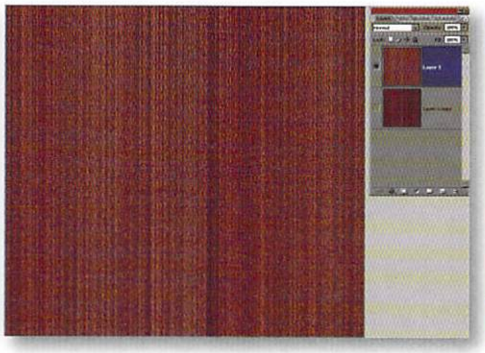


Figure 58– Noise applied a second time

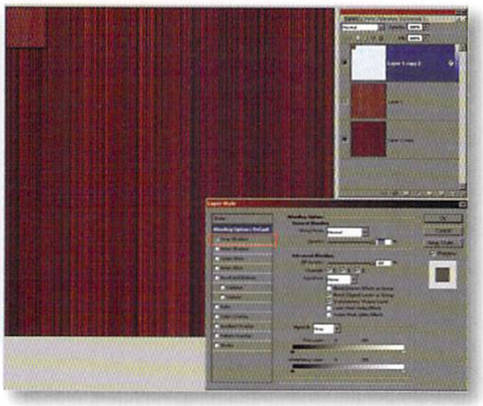


Figure 59– First tile created

See **Figures 51 thru 53** for some quick examples of the end result of some of the other textures.

MAPPING FOR THE ROOF

The tile was created using a variety of filter and blend mode combinations. One tile was created and it was duplicated to produce the entire roofing shingle effect.

STEP 1

Create a new file with the same dimensions and resolution as the stone template from the previous exercise. Using the cloud command *Filter>Render>Clouds* to fill a new layer with two shades of brown. **(Figure 54)**

STEP 2

Apply some noise to this layer *(Filter>Noise>Add Noise)*. **(Figure 55)**

STEP 3

Next, apply some motion blur *(Filter>Blur>Motion Blur)*. **(Figure 56)**

STEP 4

Duplicate this layer and set the blend mode to Overlay to accentuate the texture. **(Figure 57)**

STEP 5

Apply some noise again to give it a tactile appearance. **(Figure 58)**

STEP 6

While holding ctrl-alt-shift on the keyboard hit "N" and then "E" to merge all of the visible layers into a new layer without affecting the original ones. Afterward hit ctrl-T and resize this layer to be the basis of one of many individual tiles. **(Figure 59)**

STEP 7

While holding ctrl-alt-shift on the keyboard hit "N" and then "E" to merge all of the visible layers into a new layer. Afterward hit ctrl-t and resize this layer to be the basis of one of many individual tiles.

When done, double click on the right empty portion of the layer to access your layer effects dialogue box and apply Drop Shadow. **(Figure 60)**

STEP 8

With the Move tool activated, hold down the Alt key and duplicate the layer until you get something like **Figure 61**. Now, do the same thing, but this time, duplicate the layer set and place the

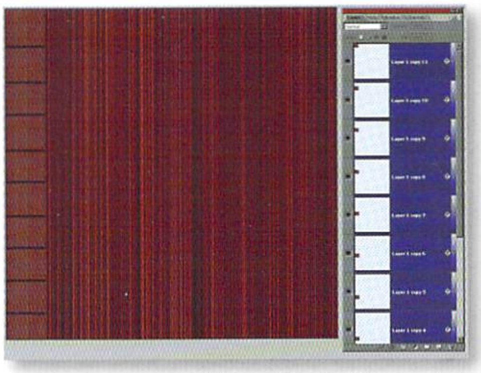


Figure 60– first tile created with drop shadow

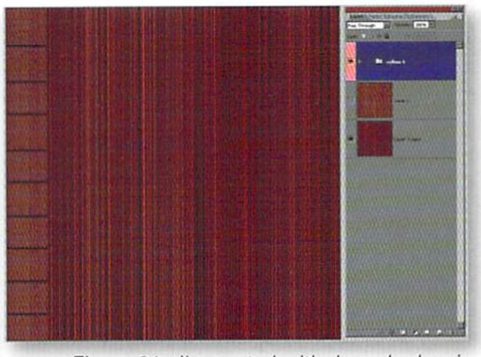


Figure 61–tile created with drop shadow is duplicated

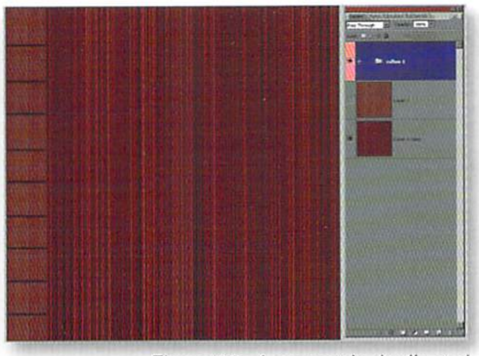


Figure 62– Layer set is duplicated

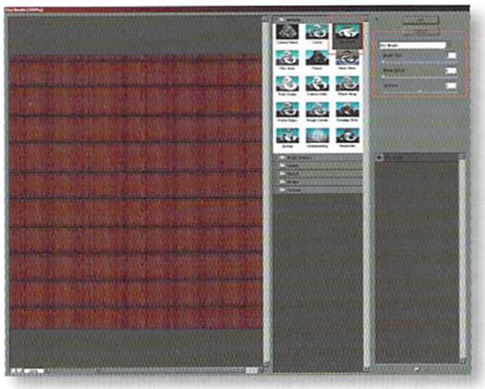


Figure 63– Filter effects applied

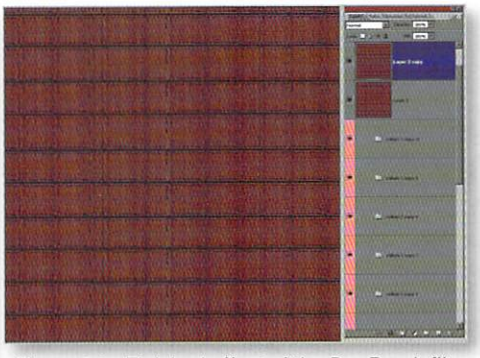


Figure 64– Finished effect of the Dry Brush filter

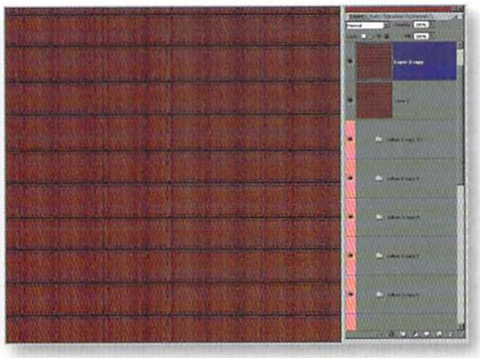


Figure 65– Fiber Filter applied

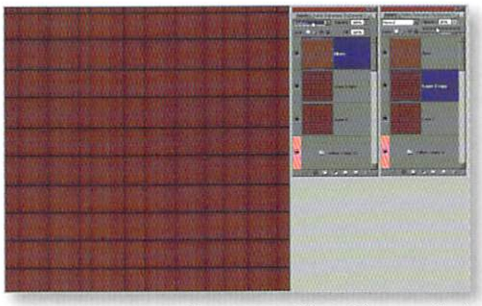


Figure 66– Fiber Filter applied

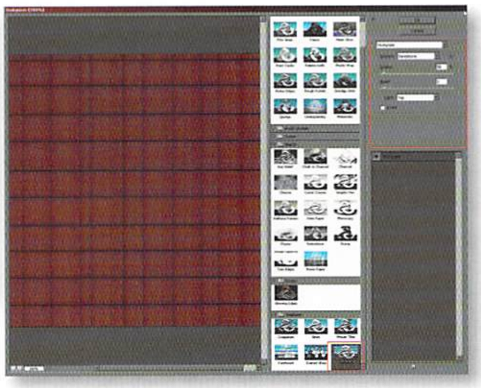


Figure 67– Texturizer Filter Dialogue

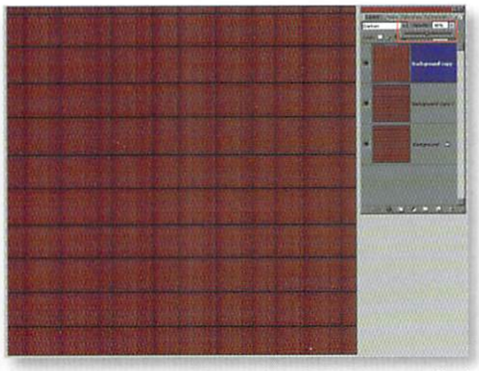


Figure 68– Texturizer applied

results horizontally, as shown in **Figure 62.**

Next highlight all of the duplicated layers by selecting the first one and shift-selecting the last one. Go to the layers submenu and choose "New Group From Layers." This creates a layer set from the selected layers. You can rename the layer set what you like, but it is named "columns" in this example.

STEP 9

Merge everything into a new layer just as you did in Step 6. Duplicate this layer to have an additional one and access the Filter Gallery (*Filter>Filter Gallery*). Apply the settings shown in **Figure 63**. This is the finished effect of the Dry Brush filter. **(Figure 64)**

STEP 10

Duplicate the layer again and apply Fibers (*Filter>Render>Fibers*). **(Figure 65)**

STEP 11

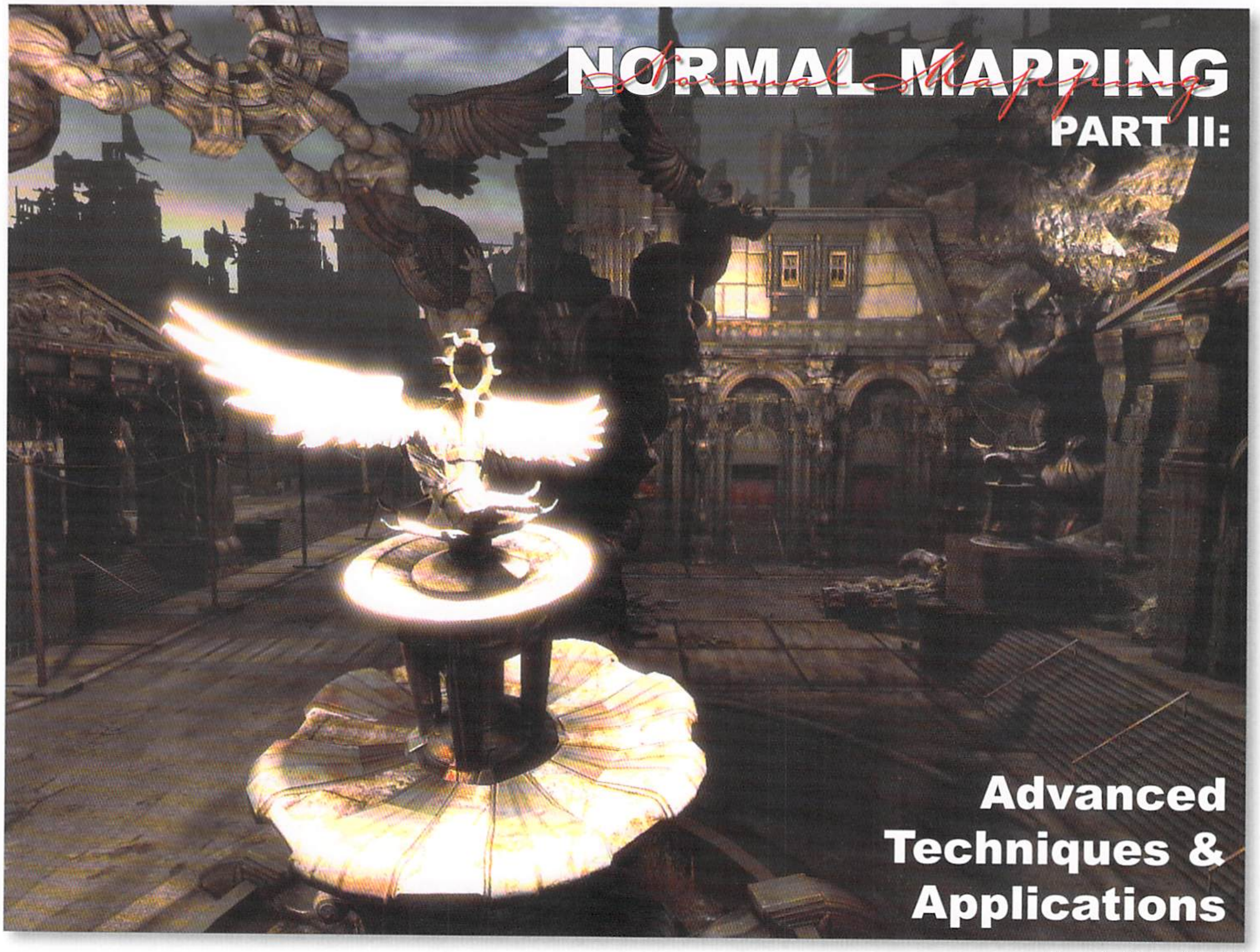
Next, change the fiber layer's blend mode to Darken and bring down the Opacity of the layer underneath is to 35%. **(Figure 66)**

STEP 12

The last thing to apply is another filter effect called Texturizer *(Filter>Filter Gallery>Texturizer)*, which gives the tiles a little more character. **(Figures 67 and 68)**

We're going to stop here for now. In the next issue, we will extract the Diffusion, Specular, and Bump maps to create the surfaces on the Cabrillo Monument Light House.

***Stephen Burns** has discovered the same passion for the digital medium as he has for photography as an art form. He began as a photographer 22 years ago and in time, progressed toward the digital medium. Stephen has been a corporate instructor and lecturer in the application of digital art and design for the past 8 1/2 years. He has been exhibiting digital fine art internationally at galleries such as Durban Art Museum in South Africa, Citizens Gallery in Yokahama, Japan, and CECUT Museum Of Mexico to name a few. His exhibiting has won him 1st place in the prestigious Seybold International digital arts contest. Owner of Burns Design, he teaches Digital Manipulation Workshops in the San Diego area and nationwide. His teaching style comes from his ability to share an understanding of Photoshop so students have the ability to intuitively apply it to his/hers creations.*



NORMAL MAPPING PART II: Advanced Techniques & Applications

Figure1– State of the Art

By Jake Carvey
Scientific Animator
West Hollywood, California

The previous installment of this article discussed the basics of normal mapping, along with methods for creating the necessary assets. In this installment, we will discuss solutions and strategies for tackling more complex challenges.

We're living in an exciting time when it comes to 3D art and craft. While special effects for live-action films continue to drive the CGI industry into more and more realistic frontiers, 3D animated movies have matured into an art form that would have the 2D and stop-motion masters flipping ecstatic cartwheels.

Two of my primary passions when I was very young were creating video games and building scale model dioramas. On one hand I was obsessed with designing 16x16 pixel x 16 color sprite animations for sprawling adventure games, which I laid out on graph paper. At the same time, I was cutting, gluing, and puttying polystyrene German soldiers until I'd converted them into space pirates, mountain men, a mermaid afloat in a resin sea, or whatever sparked my imagination. This also motivated an early interest in photography – mostly because I wanted to be able to present my dioramas in as filmic a manner as possible. The lessons I learned in both fields about optimization, organization, scale, texture, lighting, shading, lenses, and imagination have stuck with me all of these years.

With modern digital sculpting tools such as zBrush and rendering advancements such as adaptive subdivision, sub-pixel displacement, and HDR rendering, it is finally becoming possible to create digital imagery without compromising artistic vision, even in real-time game environments. Progress in the digital rendering realm gives us impressive 3D artwork, modeled, textured, and rendered so well that it is indistinguishable from photographs of scale models. (Browse the CG Channel and zBrush forums and I'm sure you'll see what I mean.)

Granted, it takes a lot of hard work and ingenuity to apply these techniques in a production environment, but the possibilities are incredible. Every day I see new digital renderings that re-inspire me the way photographs of actual miniature models used to do.

Because of the amazing advances in real-time rendering technology, CGI production for games has been drawing heavily from film production techniques, but the techniques are also now flowing in the other direction. Processes developed at the cutting edge of real-time graphics programming, such as normal mapping and photo-modeling, are becoming the new catchwords for pre-rendered effects production. In professional production environments, regardless of the specific industry, tasks must be done quickly, and finished to very high standards, while still maintaining enough flexibility, allowing for changes ranging from

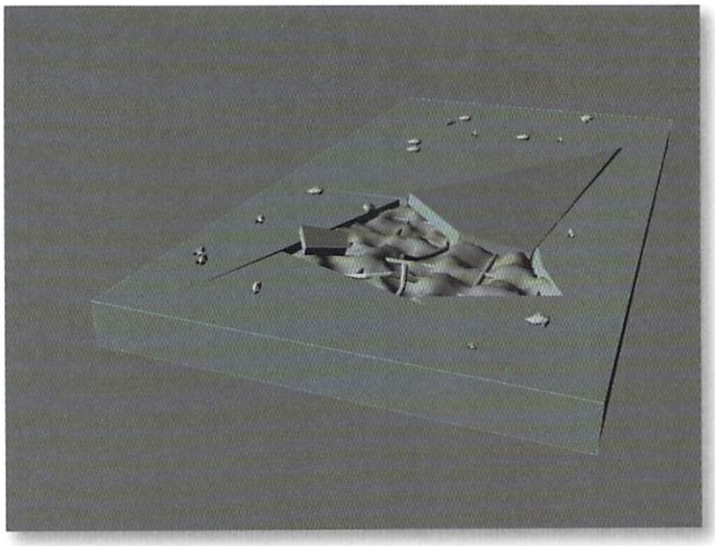


Figure 2– Basic Geometry

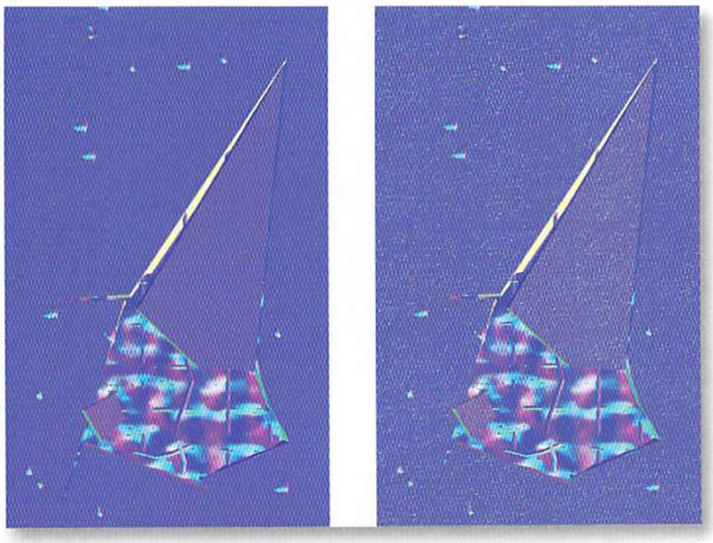


Figure 3– Normal Map without and with bumps (contrast adjusted for clarity)

minor cosmetic tweaks to major overhauls of an entire object; therefore, they quickly assimilate any technique that saves time without sacrificing aesthetics.

If you can learn to produce high quality, normal-mapped geometry quickly and predictably with a high level of aesthetic and technical quality, your services will continually be in high demand in both the gaming and special effects industries.

DIVERSITY THRU DESIGN: A CONCRETE EXAMPLE

Until recently, endlessly tiling textures were an unavoidable reality, especially in wide-open areas. About the best any artist could hope for was "detail" textures: small repeating images which could create the illusion of higher resolution imagery when viewed close up.

Today's high-resolution game engines require more precisely designed texturing strategies. Gamers are coming to expect detail in every aspect of a game – and this tendency will only continue to grow.

This means that we can't settle for bland, repetitive textures. Everything must be deep and unique, with loads of character and story. There are many great tutorials and books discussing techniques for adding grunge, grime, dents, and scars to bring "history" to every asset. Suffice it for now to say that we definitely want our textures to feel like they've been around a long time, revealing several layers of construction, destruction, repair, and disrepair.

As with any other complex project, developing convincing real-time material shaders requires careful planning. To produce high-quality results within schedule and budget constraints, we must bring both technical and artistic creativity to bear. Considering the rising costs of cutting edge game development, learning to do your job quickly and efficiently will make you a very valuable commodity indeed.

To start out, we're going to specifically outline what we are trying to achieve, using a real world example.

GOAL

Our specific goal is creating a set of deeply textured, normal-mapped concrete materials; we'll use these over and over again for RoboBlitz, a PC game in development with Naked Sky Entertainment, utilizing Epics Games' Unreal Engine 3. We need to provide a combination of subtle and drastic variations, suitable for covering large expanses without feeling repetitive. We will also eventually use additional pieces of geometry to interrupt the surfaces (piles of cables/rope/wires, piles of metal, dirt, rock).

LIMITATIONS

The textures must be applicable to very lo-res static mesh, terrain, and BSP geometry (simple geometry created in the game engine).

HUNTING & GATHERING

To figure out what elements will contribute to our final pieces, we will look no further than real life to determine the types of things which we might find in, around and underneath our concrete. A field trip in any urban or rural area can reveal a wealth of information.

Here's a quick list of possible "features" we can include: pebbles, gravel, wire mesh, conduit, metal plates, rebar, pipes, cracks, rubble, rusted wire, metal plates, water puddles, candy wrappers, pre-chewed bubble gum, used – never mind.

The point is to collect as many ideas and as mush reference as possible to provide the necessary diversity within the design constraints of your project. Once they are collected and sorted, it's time to add them into the digital KitchenAid.

Once we have our reference collected, we can decide whether specific elements need to be modeled, or can be best represented through bitmapped or procedural textures.

BASIC MODELING

Despite all of this cutting-edge mumbo-jumbo, all of the day-to-day modeling techniques (Booleans, morphs, displacement maps, and other modifiers) still have an important role to play. In this case, we will build from a very basic piece of lo-res geometry – the end result must be applicable to the simplest of surfaces, after all: a single polygon. The wireframe edges represent the simple 6-side geometry – in this case, we are only actually concerned with the single top surface. **(Figure 2)**

NURNIES: THE ICING ON THE DIGITAL CAKE

"Nurnies" is an industry term for all the little wires, boxes, dials, panels, and knobs that create the illusion of complexity on a model's surface, whether a physical miniature or digital model.

There are utilities available for all highend apps that can aid in the random (or orderly) placement of geometric details over an object's surface. You accomplish this by programmatically replicating smaller details over a mesh, or using tools for "painting" geometry over a surface interactively. Some are part of the base package, like Maya's PaintFX and modo's Paste tool; others are 3rd party solutions, like MeshPaint for LightWave (from evasion|3D).

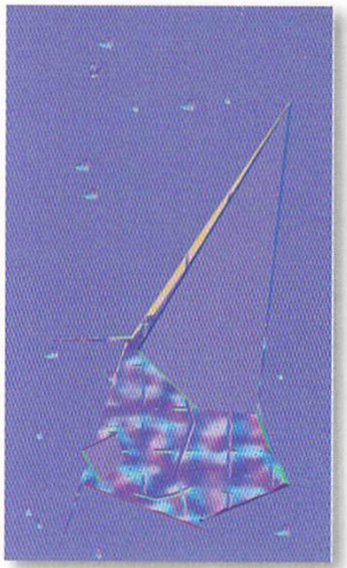


Figure 4– Normal Map Base

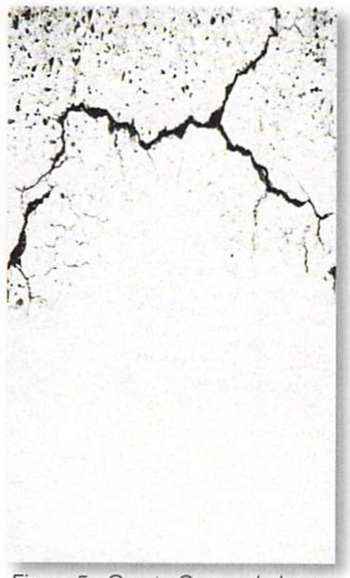


Figure 5– Create Grayscale Layer in Photoshop

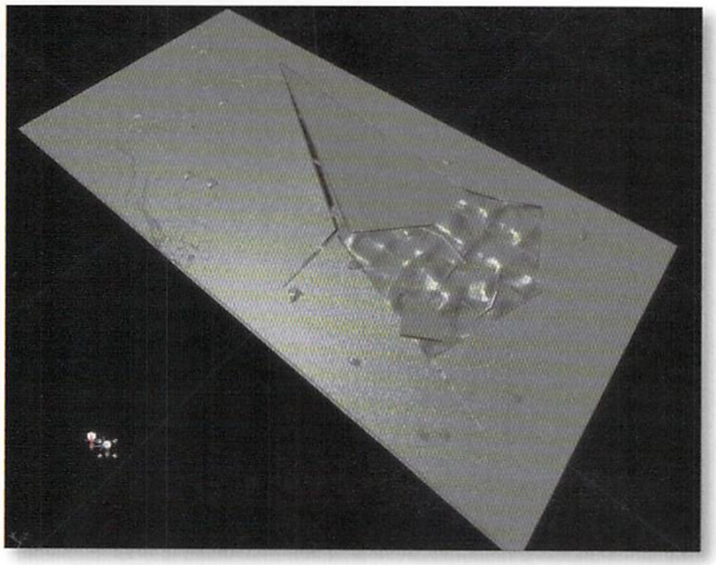


Figure 7– In-Engine Result

BUMP MAPPING

While integrated normal mapping tools often require awkward workarounds and counter-intuitive techniques for adding bump map information to a normal map, 3rd party tools such as Microwave and Kaldera can automatically calculate bumps and include them right along with the normal information derived from the geometry itself. **(Figure 3)**

We can use this capability to create richer detail, taking advantage of texturing tools available with raytrace and scanline rendering with no additional hassle.

It is still possible to create the necessary assets without 3rd party plug-ins, but in a production environment, the time saved saves money and, more importantly, allows for greater creative freedom.

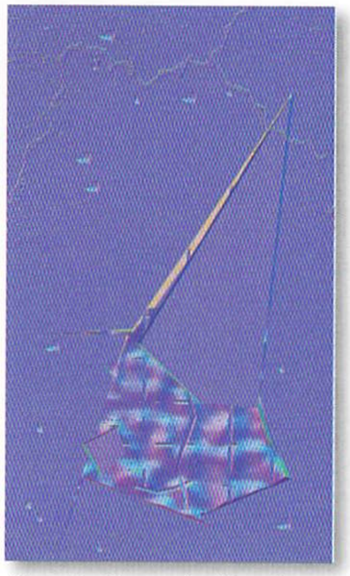


Figure 6– Composited Result After Applying Normal Map Filter

PHOTOSHOP LAYERING

It is possible to add additional layers to existing normal maps in Photoshop. Recent improvements in the nVidia NormalMapFilter plug-in make it possible to perform normal mapping on an individual layer. Previously, you could only use this filter on a flattened image.

To take advantage of the features with the most recent version of the nVidia Normal Map Filter, load a pre-existing Normal Map texture. **(Figure 4)** Next, create a grayscale layer to add further details. **(Figure 5)**

Create a duplicate of the grayscale layer for safekeeping, and turn off the original layer's visibility. Apply the Normal Map filter. The new version of the Normal Map filter does not require flattening of the image, as the old one used to do. You may have to fiddle with the options a bit, but the result should have transparent areas, layering nicely over the existing normal map information. Play with the "MinZ" and "Scale" fields to get a good mix with the underlying texture. You can also adjust the Transparency and Blending modes to achieve the best result. **(Figure 6)**

This allows for all kinds of new techniques for adding scratches, rivet heads, etc., without having to add them to a hi-res model first. It also allows for building a library of normal mapped patterns; use these over and over for adding depth and detail to normal-mapped surfaces. In fact, in this case, it could be nice to replace some of the fairly artificial looking cracks with some more natural ones, right in Photoshop.

Figure 8– Robozero UFO – Concept Art

But a word of warning: it should be obvious by now that the math involved may produce unexpected results. Adding detail to normal maps in Photoshop usually works best on flat surfaces, or more specifically, surfaces whose normals are fairly flat across the section to which you wish to add detail. Adding dents to a tightly curved headlight cowling might not work well, but adding scratches to a piece of steel armor plating could work very nicely indeed. **(Figure 7)**

NORMAL MAPPING FOR PHOTOREAL RENDERING

Normal mapping can help when it comes to speeding up rendering times for photorealistic special effects models, as well. Most high-

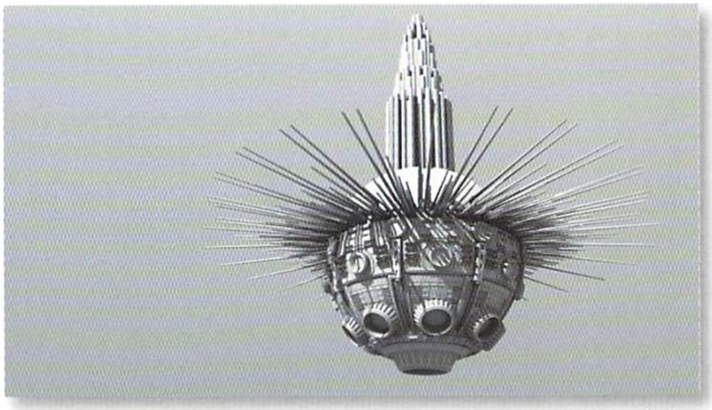


Figure 9– Robozero UFO – Hi-Res

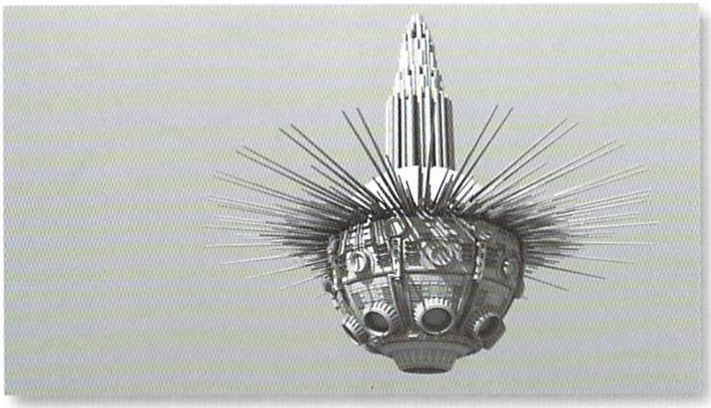


Figure 10– Robozero UFO – Lo-Res

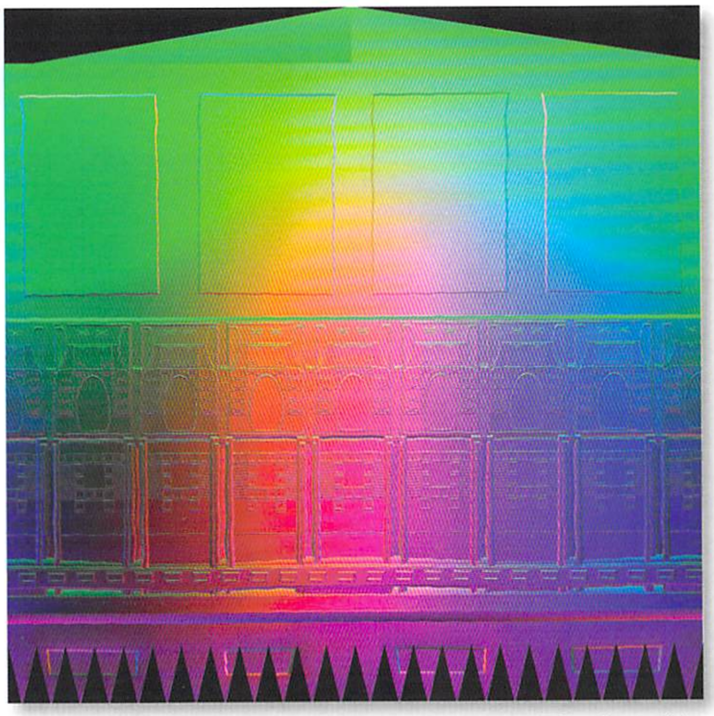


Figure 11– Robozero UFO – Normal Map

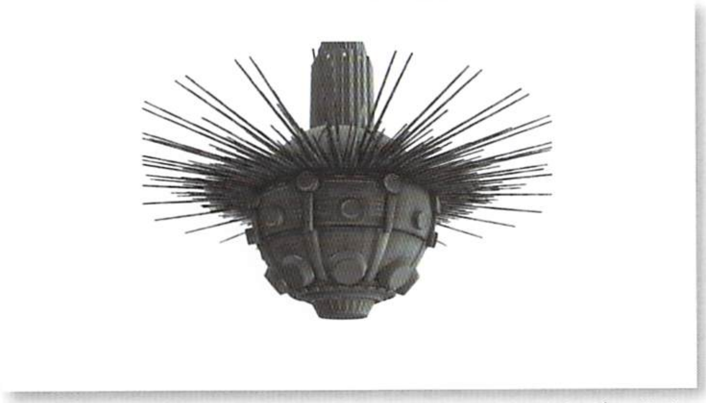


Figure 12– Robozero UFO – Rendered with Normal Map (applied to Color Channel)

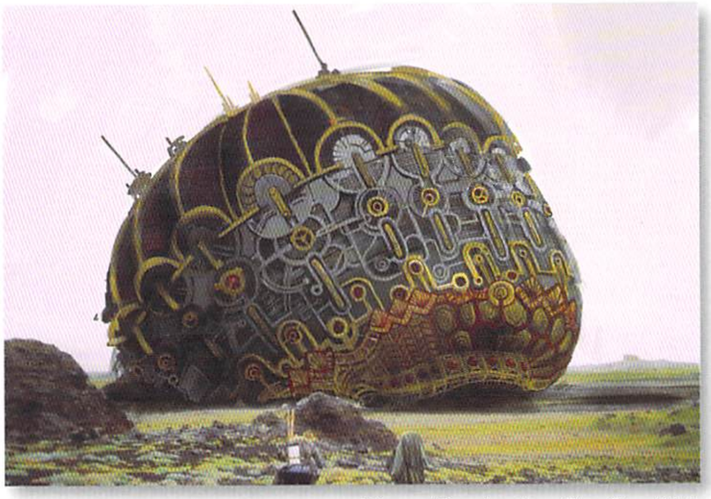


Figure 13a– Another Fine Candidate

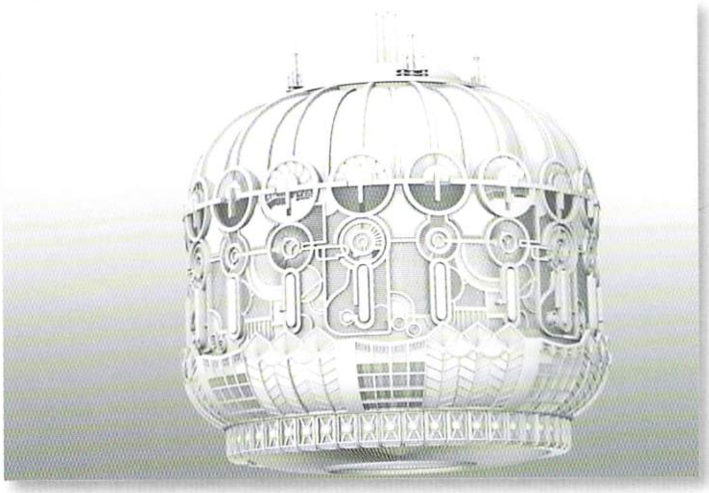


Figure 13b– Another Fine Candidate

resolution modelers tend to worry less about polygon counts and concentrate more on creating areas of deep relief with lots of bevels, modeled seams, and intricate details. They know that all of this detail goes a long way towards selling the model as real after lighting and rendering. Bump maps tend to come out looking "soft" and shallow when attempting to replace actual geometry. (They still perform an important role in breaking up the surface, but are not a valid substitute for solid geometry.) Normal maps, however, can represent much more of the necessary information, and can be used to great effect to reduce polygon counts and therefore render times for highly detailed objects.

Russia's Roswell is the working title for a History Channel show produced by Walz/O'Malley, with CGI effects provided by yours truly. Producing several minutes of animation on a limited budget, we have had to find ways to speed up rendering times as much as possible, wherever possible. Normal mapping has proven invaluable for rendering models seen at medium and long distances. We will still use the very high-resolution models for closer shots as appropriate (design by Alex Okita, models by Diego Gorlato and Peter Gend). **(Figures 8-13b)**

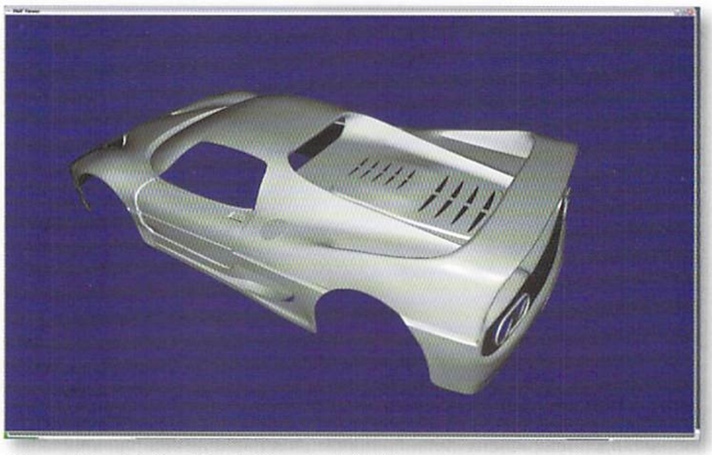


Figure 14a– Hi-Res Mesh (Shaded) (ATI)

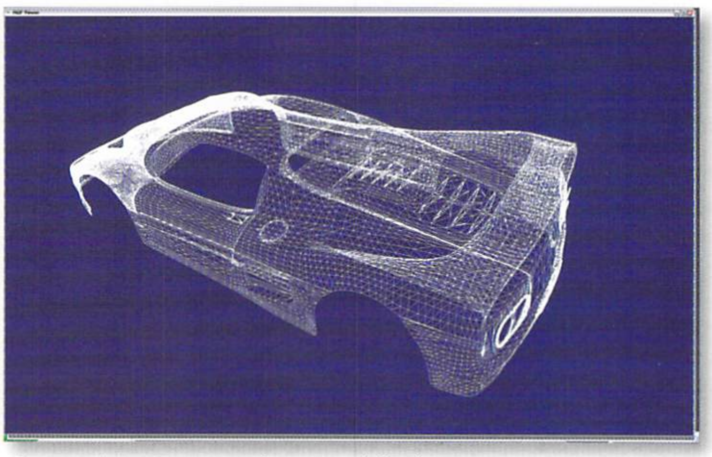


Figure 14b– Hi-Res Mesh (Wireframe) (ATI)

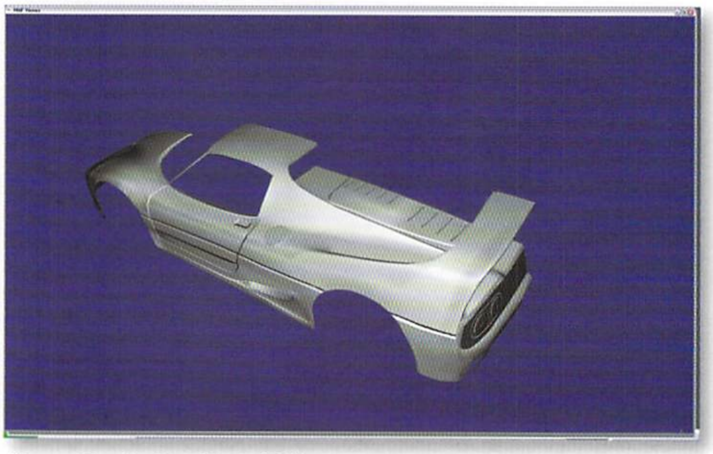


Figure 15a– Lo-Res Mesh (ATI)

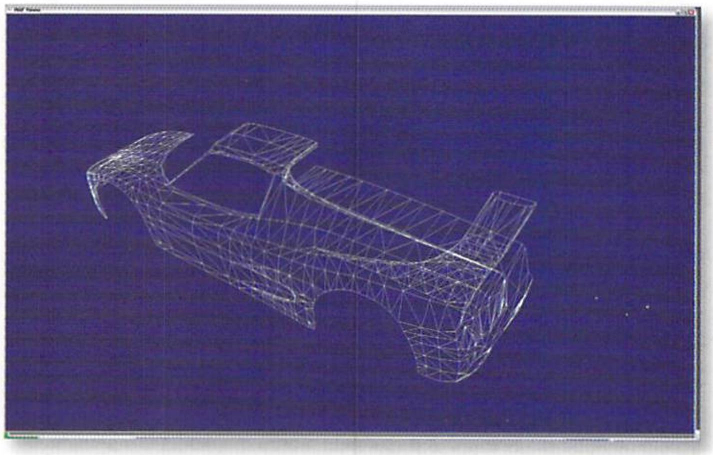


Figure 15b– Lo-Res Mesh (Wireframe) (ATI)

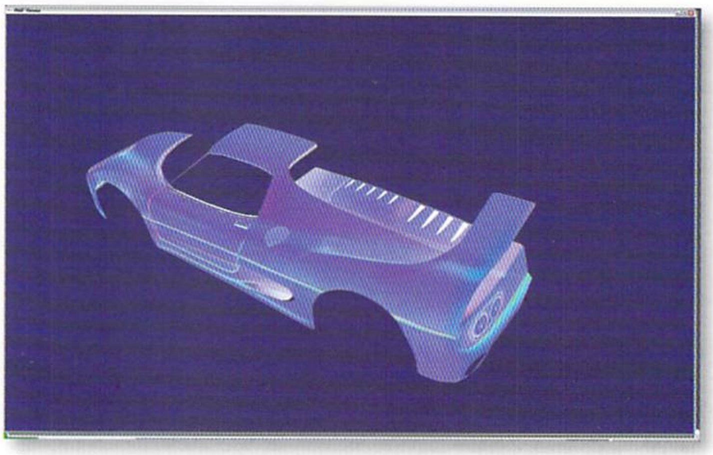


Figure 16a– Car Normals

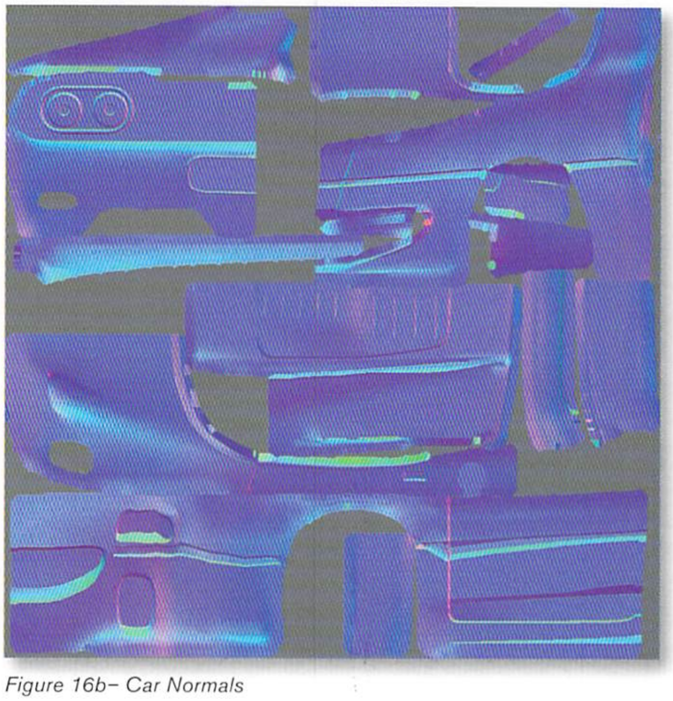


Figure 16b– Car Normals

HARD SURFACE MODELING

Normal maps aren't just for small details – use them for creating smoothly curved surfaces as well. Cars, spaceships, etc. can all benefit from ultra-smooth normal mapped curves, rather than relying on standard vertex smoothing. **(Figures 14-17b)**

I processed this model with ATI's open-source Normal Mapper program (it is included as a sample object). One nice side benefit of this particular tool is that you can generate the Ambient Occlusion map at the same time. nVidia's Melody tool also has this feature.

REALITY BYTES: TAMING COMPLEXITY

As you become familiar with the tools, it becomes obvious that you should bake certain parts separately from other parts, even if you will eventually merge them together into a single mesh for importing into the engine. Processing pieces separately can prevent problems with overlapping parts creating artifacts in normal maps and other baked

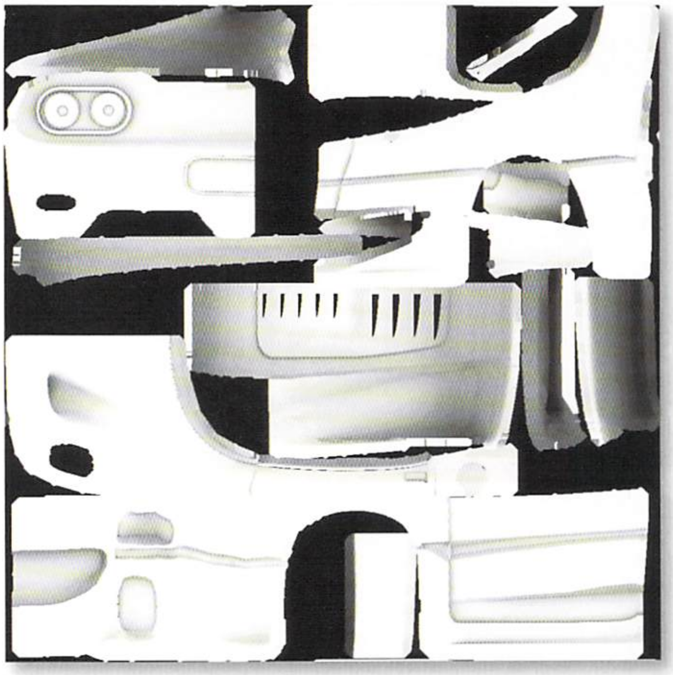


Figure 16c– Car Normals

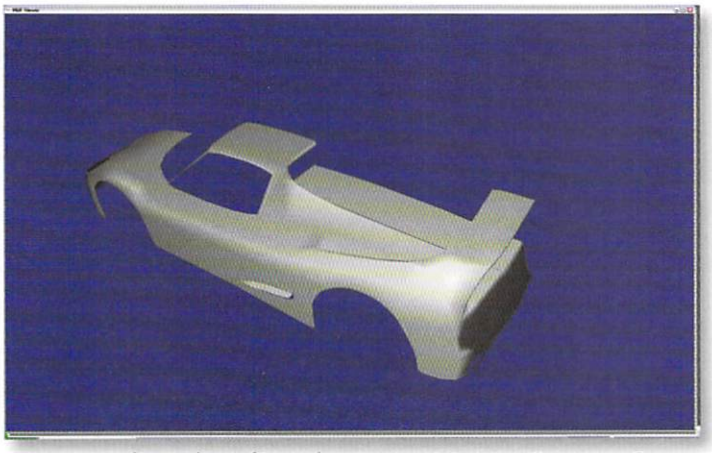
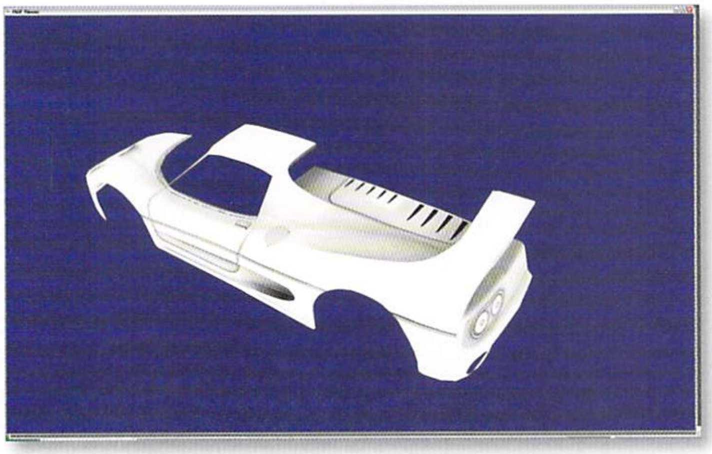


Figure 17a (Above) & b (Below) Normal and Ambient Occlusion Output; Mapped Onto Lo-Res model. (ATI)



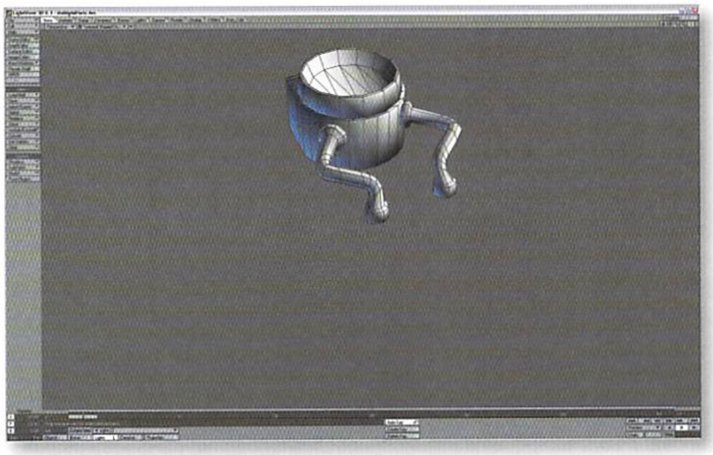


Figure 18a– Multiple parts

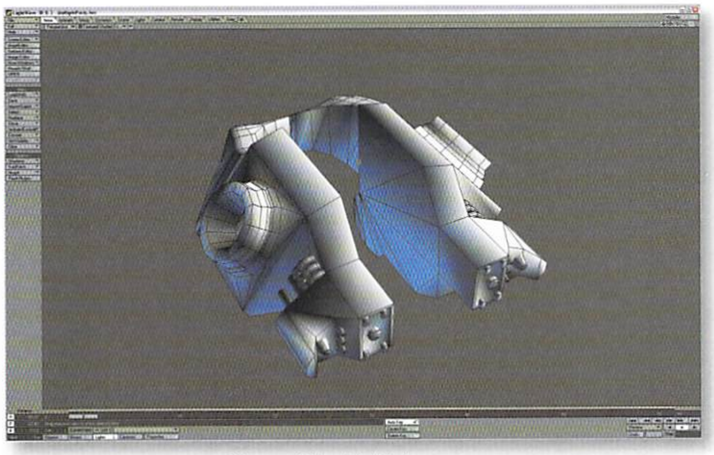


Figure 18b– Multiple parts

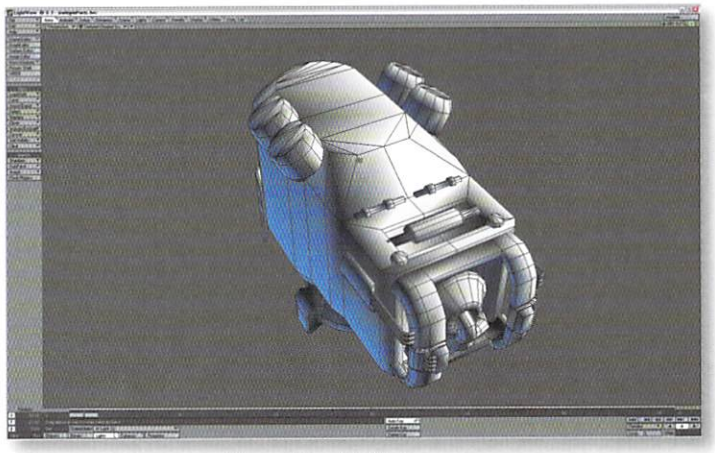
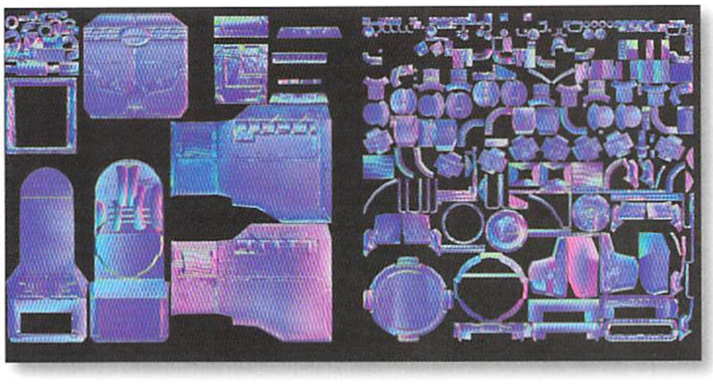


Figure 18c– Multiple parts

Figure 19–Multiple Textures



textures. **(Figures 18-21)**

Also, if certain pieces are mirrored or otherwise duplicated in regular patterns, try baking a single instance of that piece, and then duplicate it after generating the textures. It is possible in most tools to bake out multiple identical lo-res pieces at the same time, even if they use the

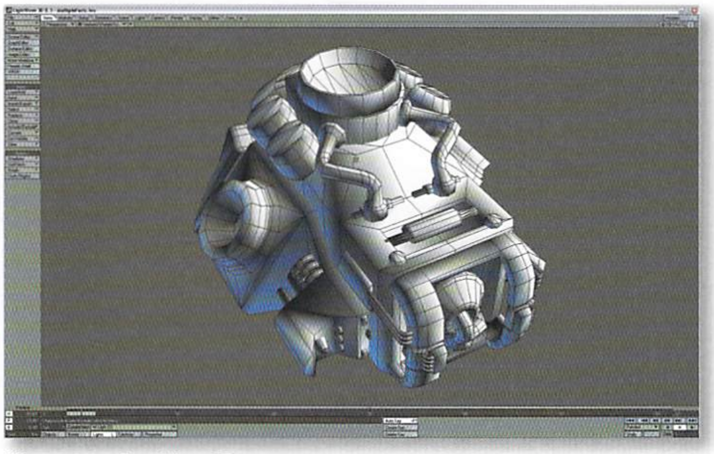


Figure 20– Combined Parts

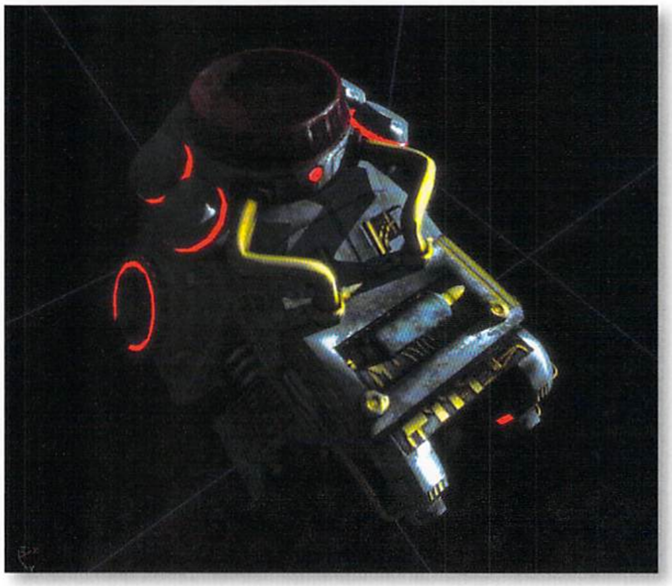


Figure 22– Final Object In-Engine

exact same UV space, but if the corresponding hi-res mesh is even slightly different, it will introduce artifacts. **(Figure 22)**

NAME THOSE ASSETS

All of this complexity requires a good naming scheme to ensure it is very clear which parts are at what stage of completion.

I use a combination of codes and dates built right in to the filename to keep track of everything. For instance: **052505_Monster_Head_LoRes_BAKER.obj** and **052405_Monster_Head_HiRes_BAKER.obj** might be the meshes used for baking out the proper textures, while **052505_Monster_Head_LoRes_COMP.obj** would be the version with any duplicated or mirrored pieces completed, ready for exporting to the engine.

Putting the date first ensures that it is easy to find the latest revisions, even if the filenames get screwy (they always do).

Always store multiple versions of your working files – if you follow the same path from hi-res source to lo-res in-game asset each time, for all of your models and textures, it will be easy to repeat if modifications are necessary. If you do make quick and dirty changes at an intermediate stage in the workflow, document the change and implement it at the base level of the pipeline as soon as possible.

PHONG FAUX PAS: SUCCESSFUL SHADER SENSIBILITIES

With all the modeling, surface prep, and baking work we've done, it's time to move on to the real-time rendering pipeline. In real-time,

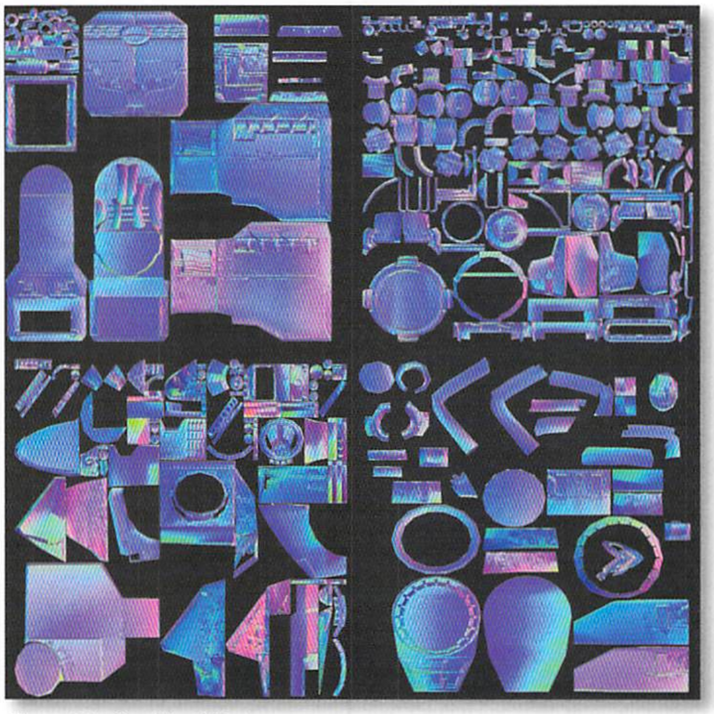


Figure 21– Combined Textures

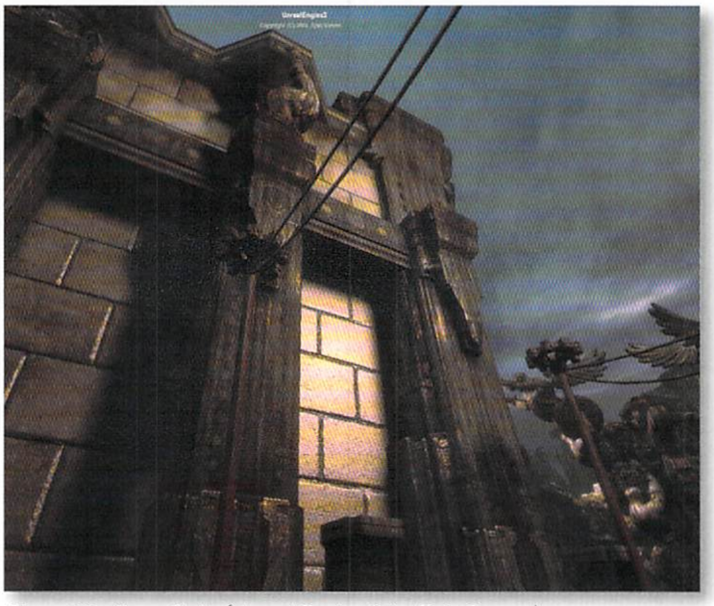
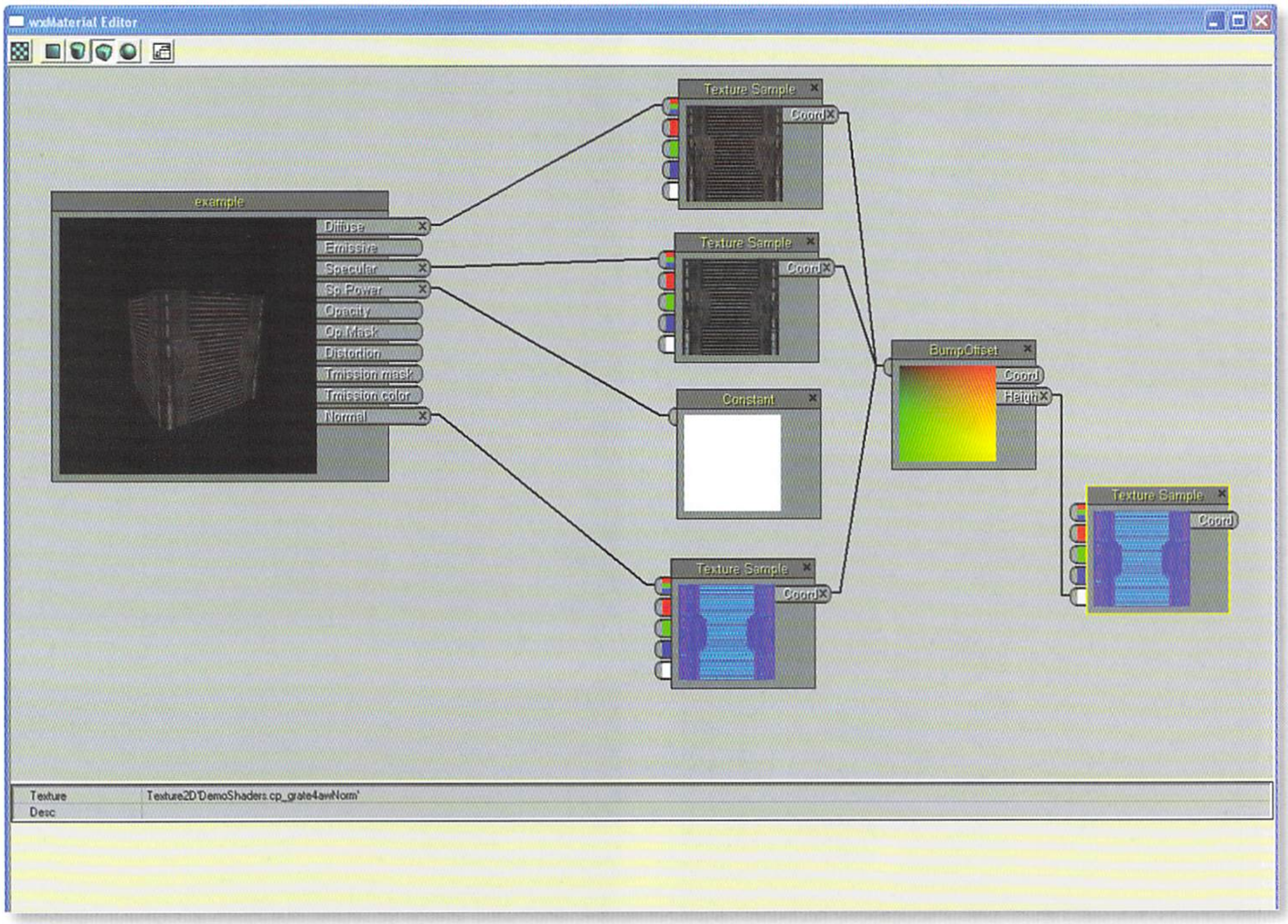


Figure 23– Spec Sells (Unreal Engine 3 by Epic Games)

there's only one rule: what you see is what you get. That's it. The best laid plans of mice and men, and all that. If all of your theories, planning, and inspiration don't stack up in the end, you're going to have to back-step. The great thing about real-time is that you can play with the available parameters and see instant results. When you see it snap into pace, you'll know you got it right – there are some techniques to help push your assets right up to the cutting edge.

OVER-CAFFEINATED

Because of the capabilities inherent in current 3D graphics acceleration hardware, most up-to-the-minute game engines utilize High Dynamic Range Rendering. This means the same benefits available in highend 3D rendering programs like mental ray, LightWave, and RenderMan are available in modern game engines. Floating point color, high contrast ranges, realistic reflections, light bloom, Ambient Occlusion – there's a way to take advantage of all



Texture	Texture2D'DemoShaders.cp_grate4awNorm'
Desc	

Figure 24– Parallax Shader Network (Unreal Engine 3 by Epic Games)

of our favorite tools in the real-time environment. There is a lot to gain by taking advantage of these new capabilities.

Of primary interest is the ability to over-crank your values – you can (and often should) set lights and surface attributes to many times "Normal Brightness." Rather than being limited to values of 0-255, the sky is the limit. A fully rounded understanding of the theories and practical applications of High Dynamic Range Imagery is essential to every digital artist. There are massive reserves of material available on the Internet that can aid in blinding enlightenment.

REFLECTIVE SPECULATIONS

Specular reflections are one of the most important aspects in bringing your normal mapped models to life. Use color liberally in your specular maps for metallic and other effects like satin and iridescence. Specular values benefit greatly when pushed well beyond normal values in the shader. Make sure that shaded, dusty, dirty, or deep areas have Specularity dulled down to proved marked contrast with shinier surfaces. **(Figure 23)**

Specular mapping is essential for selling the Normal Mapped effect. It includes the angles between the camera, the object and the light source, whereas diffuse shading only includes the angle between the geometry and the light. As the light plays across the surface, we see the reflection of the light source (the specular reflection) play across the surface. This serves to strongly reinforce the more limited visual information provided by the diffuse shading.

It's important to recognize a very important connection between CGI and traditional photography. Plain and simple, the specular component of the well-known Gourad and Phong shading models is a fake. It represents the reflection of the light source on a surface, but only goes so far in that representation. As an extreme example, when lighting actual cars in a photography studio, photographers tend to use enormous white light boxes, which reflect in the surface of the vehicle revealing the forms far more dramatically than spotlights ever could. Since the advent of HDR imagery, many artists have abandoned faked Phong specularity altogether, and insist on using raytraced or environment reflection mapping to simulate specular reflections, even on very matte surfaces.

While specular mapping still has many uses, it is important to keep the underlying concepts in mind when designing shaders.

Whenever possible, also create glossiness maps. Combined with over-cranked specular maps, glossiness maps provide dramatic changes across the surface as an object moves in relation to the lights and camera. Many of the last generation of game engines (Doom3, Far Cry, etc.) had limited support for glossiness mapping, and this led to many surfaces feeling very "plastic" since the glossiness stayed the same across surfaces, rather than changing in dusty, wet, or polished areas. In pre-rendered sequences, omitting gloss map textures can be the kiss of death for photorealism.

FREAKIN' FRESNEL

While not always practical in real-time applications, consideration for Fresnel properties of surfaces can go a very long way towards selling realism. Essentially, this means that as the surface of an object

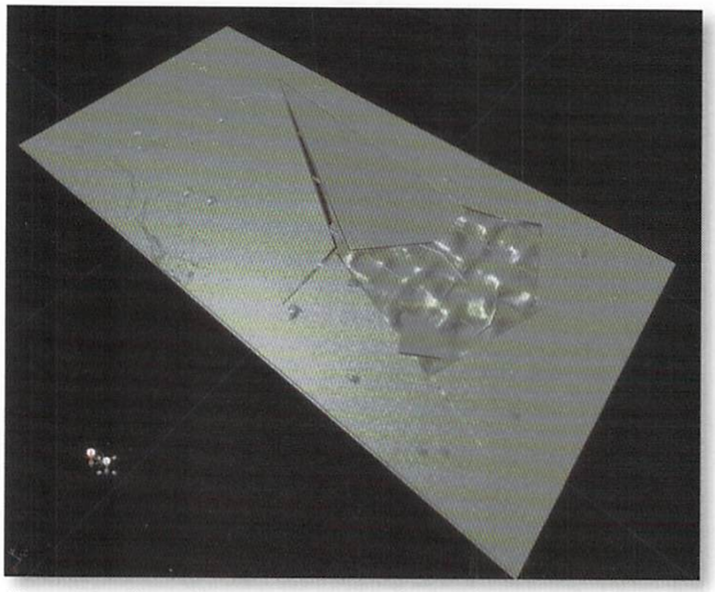


Figure 25a– Normal Normals vs. Parallax (Normal)

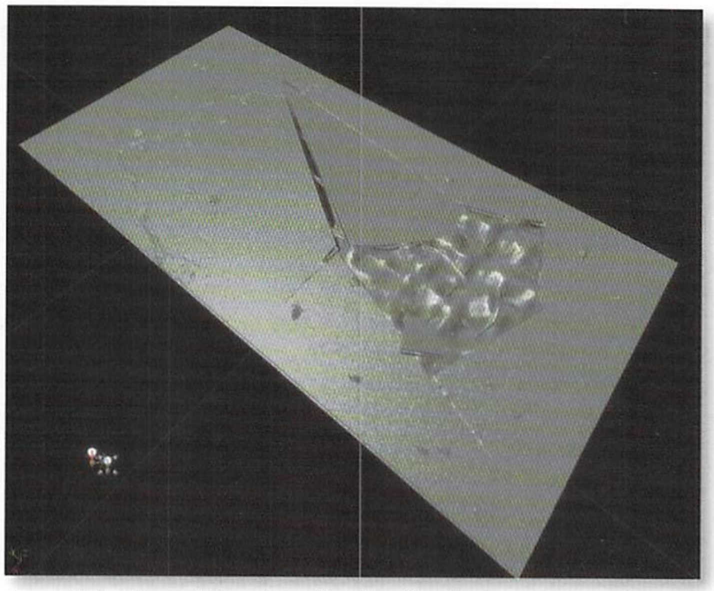


Figure 25b– Normal Normals vs. Parallax (Parallax)

becomes more perpendicular to our view, its characteristics tend to change. A very familiar example is the mirage effect of a long desert highway. The normally matte asphalt starts to reflect the sky as it recedes into the distance, creating a surreal inversion of reality. You can also observe this phenomenon when sitting in rush-hour traffic. Car paint reflects the environment far more intensely when viewed at a shallow angle.

Fresnel attributes are significant in nearly all surface properties, from transparency to reflection to iridescent color effects. Current real-time shaders tend to have limits on their overall complexity. This means you need to choose the most important components to incorporate. Reflection and transparency tend to be the most significant – so if you're creating a real-time tide pool simulation, that decision is a no-brainer. Complex car paint surfaces are another example – there are excellent examples illustrating the contribution of Fresnel effects to Specularity and Reflection, as well as Diffuse Color.

PARALLAX / BUMP OFFSET MAPPING

Most cutting-edge game engines and tools that can take advantage of normal mapping use Parallax/Bump Mapping. What it does is create the illusion of even greater depth than just a normal map alone. The technique generally requires creation of both a Normal as well as a standard grayscale Height/Displacement Map. Consult your software documentation for details on baking out Height/Displacement maps. There is a normal map to displacement map conversion program available at http//www.zarria.net, but it seems to always convert the images to 512x512. While not ideal, it is still possible to use it in a pinch, when you don't have access or time to generate a true displacement map. **(Figures 24-25b)**

The differences will obviously be more difficult to see in print – where this stuff really shines is when you can move it around and run past it and over it, and really see the light play across the surface. The Bump Offset effect seems to work best with deeper elements – the shallow cracks towards the top left edge are less convincing than the exposed underlayer. Real-time development always requires a lot of testing and tweaking to see what works best for each asset – but the beautiful part is, when it comes to shader networks and material settings, you get to see it all update immediately in the engine!

DETAIL-UP MODELING

Many game artists are understandably more familiar with low polygon modeling techniques than multi-million poly photorealistic film-style modeling. If the thought of building a nine million polygon spaceship from scratch seems a bit intimidating, it is totally reasonable to start with a lo-res mesh and add detail through subdivision, extrusion, smoothing, and other techniques. Often, all that you need for mechanical objects is the addition of tons of pipes, bolts, panels, etc., to create the necessary detail with little or no modification to the underlying lo-res mesh. Breaking the model carefully into layers of overlapping detail makes it easier to make changes and tweaks down the line.

Some simple objects like walls and floors may never even need a high resolution mesh created and can be textured using entirely 2D techniques, with good results.

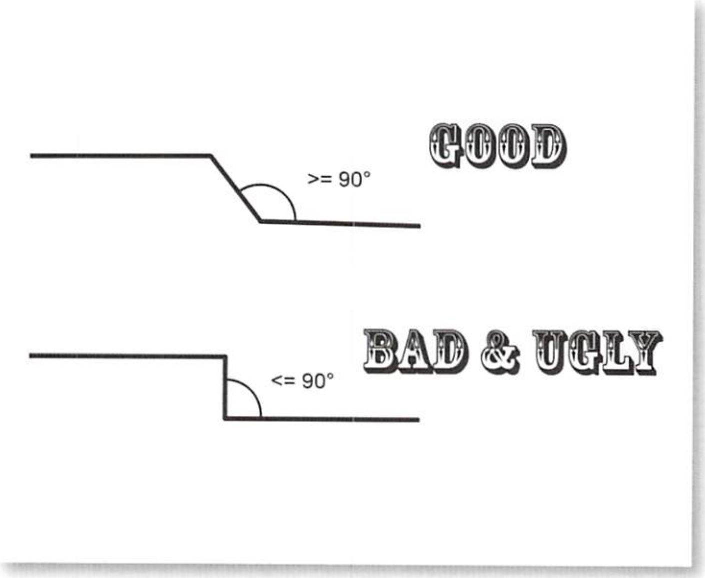
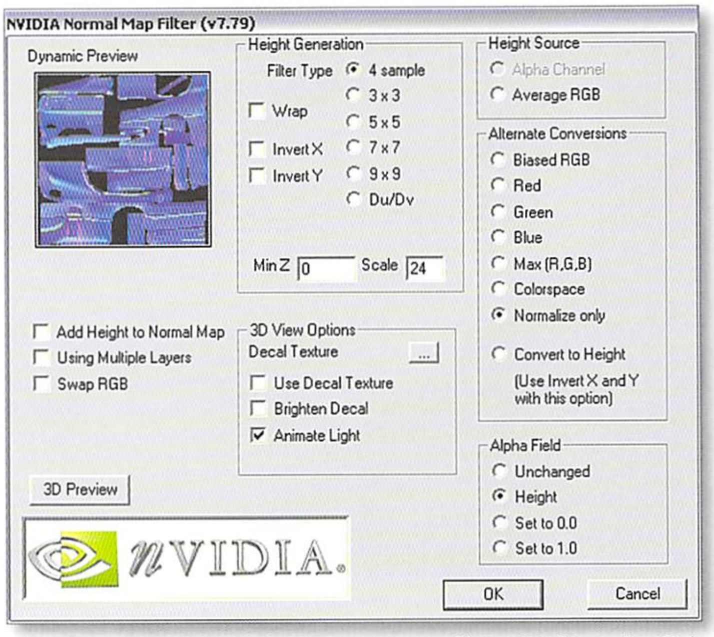


Figure 26– Sloping sides

SLIDE THE SLIPPERY SLOPE

To prevent or solve complex baking challenges, remember to make sure that the polygons of your lo-res model have sides that are sloped adequately to ensure a good "view" of the hi-res geometry. Nasty artifacts will otherwise occur, since the Lo-Res geometry will be projecting through itself. **(Figure 26)**


NVIDIA Normal Map Filter (v7.79)

Dynamic Preview

Height Generation
- Filter Type: [✓ 4 sample] / 3x3 / 5x5 / 7x7 / 9x9 / Du/Dv
- ☐ Wrap
- ☐ Invert X
- ☐ Invert Y
- Min Z: 0
- Scale: 24

Height Source
- Alpha Channel
- Average RGB

Alternate Conversions
- Biased RGB
- Red
- Green
- Blue
- Max (R,G,B)
- Colorspace
- [✓ Normalize only]
- Convert to Height (Use Invert X and Y with this option)

- ☐ Add Height to Normal Map
- ☐ Using Multiple Layers
- ☐ Swap RGB

3D View Options
- Decal Texture ...
- ☐ Use Decal Texture
- ☐ Brighten Decal
- ☑ Animate Light

Alpha Field
- Unchanged
- [✓ Height]
- Set to 0.0
- Set to 1.0

Figure 27– Normalize Only

This technique is most useful for large areas and relatively large objects (a character walking right up to the hull of a submarine for instance). The high-resolution texture maps used for characters, vehicles, etc. have plenty of resolution to hold together at close range close-up.

WHAT IS THE AIRSPEED OF A LADEN SWALLOW?

We've focused primarily on mechanical and environmental objects – but Normal Mapping procedures also work brilliantly with organic creatures and characters.

zBrush 2.0 from Pixologic has become the de facto standard for detailing creature and character models. While zBrush has been around for years, the displacement and normal mapping workflows were forged in the searing hot furnaces of recent hard-core special effects production, most notably at Weta Digital, where Bay Raitt helped to elevate zBrush to near Killer App status on the Lord Of the Rings films, using it for sculpting digital creatures such as the Cave Troll. There is support for incorporating zBrush into every major rendering workflow, from LightWave to Pixar's RenderMan. While a full discussion of zBrush is beyond the scope of this article, there are many books, training DVDs, and web-based tutorials that discuss its use in great detail.

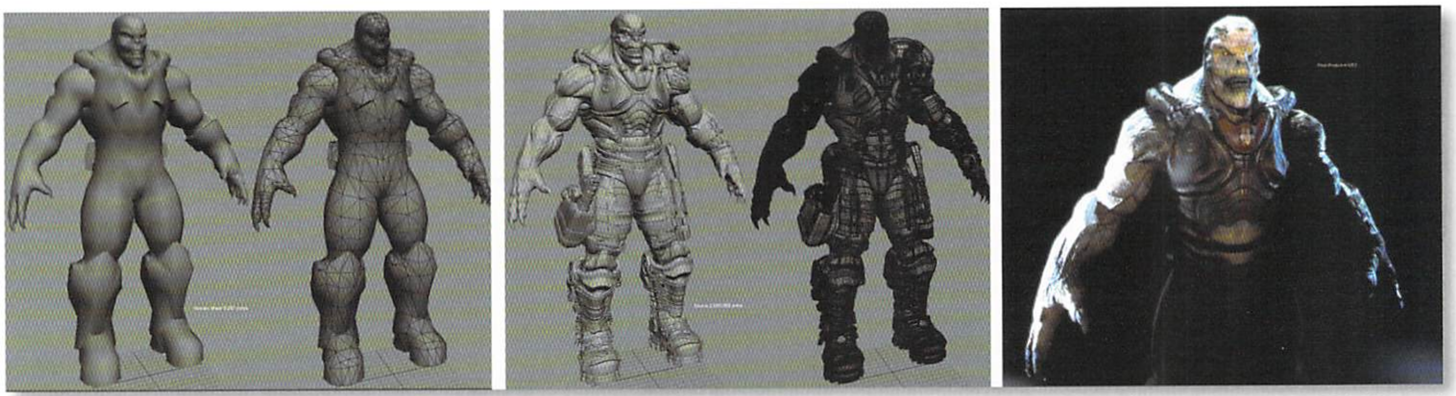


Figure 28– Character Creation Workflow (Unreal Engine 3 by Epic Games)

N.A.N. NO-NOS

Black areas in UV maps can create serious problems with certain real-time engines (specifically NAN – Not A Number – artifacts on nVidia cards). When the final texture is mip-mapped in the game, areas of black can creep into the texture and create very nasty visual bugs.

The solution is to run the image through the nVidia NormalMapFilter in Photoshop in "Normalize Only" mode. You should get in the habit of running this filter after any kind of modification to the normal map texture. **(Figure 27)**

FOR CONTROL FREAKS: MANUAL DETAIL TEXTURES

It is still possible and often desirable to use tiling normal maps to add repeating, high frequency detail to certain surfaces – a technique harkening back to the recent adolescence of real-time graphics – ripples and gentle waves on a water surface, for example. The technique can be equally as effective for fabric, asphalt, rock faces, etc. – any material that repeats many times across a surface. You can layer the detail over (or under) an object-specific normal map, providing for even greater variety.

You can accomplish this type of effect by incorporating a function into the shader/material, which uses a "Distance To Object" function. As the object gets close to the camera, the detail texture is faded in and blended with the primary texture map. It is similar in nature to mip-mapping, but whereas the engine handles mip-mapping automatically, detail texturing is a manual technique.

On the real-time front, games such as ***Doom3, Far Cry, Chronicles of Riddick,*** and ***Half Life 2*** have already incorporated normal mapping technology into their engines. ***Quake 4, KillZone2, F.E.A.R.***, and a slew of games based on Epic's Unreal Engine 3, are going to push the envelope even further, with higher polygons counts, more complex materials, and even more breathtaking lighting effects. **(Figure 28)**

You can use zBrush extensively for developing extremely detailed hi-res models, sometimes into the millions of polygons. It is also capable of generating the normal and texture maps necessary for use these real-time game engines.

RESOURCES

http://cgchannel.com/ • http://www.unrealtechnology.com
http://www.epicgames.com • http://www.ati.com/developer/
http://developer.nvidia.com/ • http://www.newtek.com
http://www.alias.com • http://www.luxology.com
http://www.softimage.com • http://www.pixologic.com
http://www.evasion3d.com • http://www.mankua.com

Jake Carvey *has been directing, producing and creating 3D animation for broadcast, games and film for over 10 years. He has worked with clients such as Sony, Disney, Universal Studios, Dunkin' Donuts, ABC, and Kid's WB!, winning numerous industry awards along the way. He loves his work because the hours are great.*

UNCLE ROBIN'S MINI GUIDE TO MEL

Volume 2: How to Make Something Actually Useful

By Robin Scher
Animator, Programmer
Los Angeles, California

BRIEF REVIEW

Greetings again, fair readers! In the first volume (see Issue #5), I detailed some of the tips and tricks of programming that would make your life easier as you started using MEL more. In this volume, I'm going to go into more specifics about MEL and Maya, and introduce you to the power and flexibility of scripts.

As a review, here is a summary of some of the key points in Volume 1. Be sure to read it for more details!

- Use shortcuts only rarely
- Make your code human-readable
- Give your variables descriptive names
- Use comments liberally
- All commands end in a semicolon
- Always start counting with zero
- Declare your variables before you use them
- Use curly braces around code blocks even if they're only one line
- Make sure your loops won't go on forever before you run them
- Keep it simple, stupid.
- Save your work before you go messing with a script!

ABOUT THIS DOCUMENT

Obviously, there are plenty of ways to learn how to script or program with MEL, and this article is only a brief introduction. It is a starting point for Maya users who are relative beginners to MEL and also has some hints for those who may have played with MEL just a bit and run into some of the weird stuff. The tips and tricks in this article should help you past those initial weird things and on to writing bigger and better scripts. Also, be sure to review Volume 1 for more details.

I have tried to use fonts and colors in order to help make things readable. Blue words signify MEL keywords or symbols, as you should type them yourself. Violet words signify commands. Keywords are actually part of the language; use them to define or control execution. Commands are the tools that give you programmatic control of the Maya tools themselves. Green is used for comments, and grey for the contents of a string variable. I display code in a `monospace font.`

You may think it's cheating to put the fancy colors into my code samples when you get nothing of the sort in the Script Editor, or the Expression Editor, or Notepad. Well, you may want to find a real code editor program. For Windows, I recommend Crimson Editor, which comes with built-in syntax highlighting for MEL. It's pretty cool, and I think it's actually free.

HELP!

By far, the most important thing about writing scripts or programs is where to get help. Believe it or not, even those of us who have been programming for decades still have to look things up almost every day.

There are two parts of the Maya documentation that you will refer to. The first is the section that actually describes the language. This part is useful for describing the actual language; you will need it less the more you write scripts and get used to how it works.

The second, however, is the invaluable MEL Command Reference. Get to know this section, and know it well. Everything that you can do is here; after so many revisions it's actually getting very well written and understandable. Read it, know it, love it, it's your biggest friend. Well, second to this wonderful article, of course.

There is also a MEL command that can give you a quick summary of help about most built-in commands and keywords. In the commandline or the Script editor, you can use the help command. Type help followed by the name of the command you want help about. For example:

```
help move;
// Result:

Synopsis: move [flags] Length Length Length [String...]
Flags:
 -a -absolute
 -dph -deletePriorHistory     on|off
 -ls -localSpace
 -os -objectSpace
 -p -parameter
 -r -relative
 -rab -reflectionAboutBBox
 -rao -reflectionAboutOrigin
 -rax -reflectionAboutX
 -ray -reflectionAboutY
 -raz -reflectionAboutZ
 -rfl -reflection
 -rft -reflectionTolerance    Float
 -rpr -rotatePivotRelative
 -spr -scalePivotRelative
 -wd -worldSpaceDistance
 -ws -worldSpace
 -x -moveX
 -xy -moveXY
 -xyz -moveXYZ
 -xz -moveXZ
 -y -moveY
 -yz -moveYZ
 -z -moveZ
```

Getting help with the help command

This outputs a quick summary of the command syntax and possible parameters. You can also type help `-doc command` to open the online HTML documentation about a specific command.

Finally, there is another great place to see how things work. Tons of scripts in the scripts directory, found under your Maya installation path, make up Maya. You should feel free to open scripts and see how the guys that wrote Maya make things work using MEL. Just be sure to make backups of them if you want to try changing things, or you may end up reinstalling Maya. Fun.

CODE BLOCKS AND PROCS AND SCRIPTS, OH MY!

MEL is the programmatic interface that you can use to control Maya. MEL creates or drives pretty much everything that you ever do in Maya at some level. In order to harness the power of Maya, you can use MEL directly and avoid the middleman.

There are several ways to get your code into Maya for execution. You can just type blocks of code into the Script Editor. You can create reusable procedures, or you can define entire systems in scripts.

One note on terminology. The programmer-speak word for executing some code is to "call" that code. I couldn't tell you why, because there are certainly neither telephones nor loud shouts involved, but this results in the wonderful term "caller," meaning the point from which the code was called. Now, the caller can be just about anything. It could be Joe Schmo user, typing the command into the Script Editor, or it could be another chunk of code that uses your code to perform part of its work. Always remember that computers are stupid and do only one thing at a time, so you don't really need to worry about who or what called your code. The computer will take care of all that for you. Just do your work, and let your caller worry about the results.

CODE BLOCKS

The most basic form of this sort of programmatic interaction is a code block. You may remember code blocks from Volume 1. To review, a code block is a logical grouping of code that acts like a black box. Once Maya starts plugging down the commands in a code block, it will execute every command in the block, in order, and the only thing that will stop it is an error in the execution of a command. A code block can be as short as a single command, or it can be a whole bunch of commands usually grouped with the curly braces {}.

If you just start typing commands into the Maya Script Editor window, you are essentially creating a little code block. Pressing the Enter key (the one on the numeric keypad, not the return key on the right of the main part of the keyboard) submits the block to Maya, which tries to execute every command you typed, in order. This is great for quick little things, but it can be kind of annoying if you want to do something more sophisticated or if you want to be able to repeat execution of your block of code on different objects or at different times.

It is possible to stash a code block on your Shelf, which makes them slightly more useful, but still, using a code block in this way is not very intuitive for users, and provides much less flexibility and control. There has to be a better way. Thankfully, MEL offers two!

PROCS

There is a special keyword in MEL, **proc**. This keyword allows you to define your own named commands, called a procedure, which work very much like any of the built-in commands. As a matter of fact, many of the built-in commands are actually just procedures, defined in various MEL files in your Maya installation directory. A proc builds on the concept of a code block by assembling one or more blocks into a reusable unit.

Essentially, you can think of procedure like a little mini computer program all unto itself. It can optionally take inputs, called parameters, and optionally return a result. Inside, it behaves like a code block, in that once you type the name of the procedure, Maya will start executing the code that makes it up, in order, not stopping unless there is some kind of error or the procedure specifically stops itself.

```
proc clearSelection()
{
  select -cl;
  print "// Nothing is selected.\n";
}
```

An example of a simple procedure definition

Notice that you don't have to put a semi-colon after the last curly bracket in your procedure definition. You can put it there, if you like (and if you type your procedures into the Script Editor yourself, one will generally get added automatically), but it is not required.

There are several advantages to using procedures over just typing code blocks in. First, once you have defined your procedure, you need only to type its name to start it executing. This is much simpler than typing a whole code block yourself, and much less error-prone than using the shelf or copying and pasting code manually. As usual, it will behoove you to give your procedures a reasonable descriptive name.

Second, you can call a procedure from anywhere that has access to the **proc** definition. You can type it in the commandline or the Script Editor, you can put it on your shelf, or you can use it in other code blocks, procedures, or scripts. You can even call your procedure from within the procedure itself. This is a powerful mechanism called recursive functions. For some types of nested data, this sort of algorithm can be extremely useful, but it comes with dangers. It is quite easy to write a recursive procedure that will call itself indefinitely, which will crash Maya.

When you define your procedure, it doesn't actually execute any of the code. What happens is that Maya reads the procedure definition, does some basic syntax checking, and then stores the procedure for later use. To actually use the procedure, just like with any other MEL command or function, you need to call it. For our example procedure above, you could type into the Script Editor:

```
clearSelection;
// Nothing is selected.
```

Accessing our custom procedure via the Script Editor

If you had anything selected before you typed the procedure name, it de-selected, and then it printed the little statement as well. It is a common practice in MEL to make sure that your output is prefaced with the comment characters, the double slashes. This allows your users to still drag and drop or copy and paste the results in the Script Editor without having to separate the commands from the output.

GLOBAL AND LOCAL PROCEDURES

Technically, MEL has two types of procedures: Global and local. A global procedure, loaded by Maya, is accessible from anywhere in the entire program. It can be used directly in the Script Editor, used in any script file, or be called from any other proc you define. You can only call a local procedure from the context that defined it. Since most procedures are defined in script files, this means that a local proc in that file can only be called by other procedures in that same file.

To make a procedure global, simply add the **global** keyword in front of the procedure definition. This adds the procedure name to the global namespace, and makes it accessible anywhere. Without the **global** keyword, the procedure will be defined locally by default.

For example, the procedure we defined above is a local procedure. It would only be accessible from the context we defined it in. If this was inside a MEL script file, then only other procedures inside that file would be able to access the proc. You could not use this proc in the Script Editor, because it would not be accessible. Here's a new version of our little procedure that is global:

```
global proc clearSelection()
{
  select -cl;
  print "// Nothing is selected.\n";
}
```

Making our custom procedure global.

Now, we can define our clearSelection procedure in a script, but access it via the Script Editor. Other procedures we may write are able to access it, as well, even if those procedures come from different script files. Using global procedures, you can build an entire system of functions that you can reuse, which is exactly what the authors of Maya have done themselves. Essentially, you become part of the development team for Maya. Now if you could only get Alias to cut you a check....

The advantage of a global procedure is obvious. You can call it from anywhere. The advantage of a local proc is less obvious. A general rule of thumb is to make all your procedures global, unless it's just a little helper routine that you specifically do not want your users to access.

PARAMETERS

To use your procedures, you generally define an "interface." The interface to a procedure consists of the parameters you pass in to your little program, and the result it returns to you. You don't have to use parameters or return values, of course, but they can be quite useful. Our little first sample procedure did not take any parameters, and did not return anything. To pass parameters in, you define the parameters that are passed inside the parentheses.

Most parameters are pretty easy to understand. For example, if you wanted to write a little script that did some sort of mathematical calculation on two numbers, you may write something like this:

```
global proc printAverage( float $value1, float $value2, float $value3 )
{
  float $average = ($value1 + $value2 + $value3) / 3;
  print( "// Average: " + $average + "\n" );
}
```

A procedure that takes three parameters

To access this procedure, you could type either of the following into the Script Editor:

```
printAverage 1 2 3;
// Average: 2
printAverage( 1, 2, 3 );
// Average: 2
```

Accessing our procedure with parameters

You could also use either of those formats inside of another proc or in a script.

What about if you want to pass an object? This is where the power of the string type comes in. You can access any objects in Maya using the name of the object, passed to your procedure as a string value.

```
global proc moveInX( string $object, float $amount )
{
print( "// moving " + $object + " " + $amount + "in the X dimension\n" );
  move $amount 0 0 $object;
}
```

A procedure that takes an object (by name) as a parameter

We can now call this procedure to move an object in the X dimension like this:

```
moveInX nurbsSphere1 5;
// moving nurbsSphere1 5in the X dimension
```

Calling a procedure that manipulates an object

You can even pass an array of objects as a parameter, and you don't even need to know how many items are in the array. Simply use the array variable syntax (see Volume 1 for more details) in the parameter definition.

```
global proc scaleObjects( string $objects[], float $size )
{
  string $each;
  for( $each in $objects )
  {
    scale $size $size $size $each;
  }
}
```

A procedure that takes an array as a parameter

You could generate your array manually, or you can use the output of a command that returns an array of strings, such as ls, as the input to this procedure:

```
scaleObjects( `ls -sl`, 2 );
```

Sending the array of selected objects to our scale procedure

RETURN VALUES

Just as it is often useful to pass inputs to your procedures, it is also quite often useful to have your procedures return a result to you. The result type is part of the "interface" to your procedure, so you need to declare what the result type of your procedure will be in the first line, where you define the interface. You also need to use the return keyword, which terminates your procedure and returns your result.

```
global proc float doCube( float $value )
{
  return $value * $value * $value;
}
```

A procedure that returns a value

Notice that the keyword **float** is after the **proc** and before the name of the procedure. This is where you put the type of result that the procedure returns. You can use any type that is valid in Maya. You can even put an array in there, and return a variable-sized array of items. (Note: Returning arrays from procedures used to cause a slow memory leak in Maya. I'm not sure if they have fixed this leak yet, so use this with caution, and always remember to save your work before you go playing around with it.)

The other new thing you see is the use of the keyword **return** in the procedure itself. This is the final command that Maya executes in your procedure, and it tells Maya what value to return to the caller. Any commands after the return statement will never execute.

When you call this procedure in the Script Editor, this is the result:

```
doCube 3;
// Result: 27 //
```

Calling a procedure with a return value

The Script Editor actually printed the result from the function, without any **print** statements in our code itself. This is just a little bonus feature of the Script Editor itself, useful for testing a procedure and seeing the result. If you called this procedure from within another procedure or a script, you would not see the result, but it would return directly to the caller for whatever purposes you desire. For example, you could use it like this:

```
float $twentyseven = doCube( 3 );
float $bignumber = `doCube $twentyseven`;
scaleObjects( `ls -sl`, `doCube 3` );
```

Using the return value from a procedure

Another thing to watch out for is that all possible flow paths end with a return statement. For example:

```
global proc float maybe( int $value )
{
  if( $value == 3 )
    return 1;
}
```

Missing return values

Here we're using an **if** statement to conditionally execute a code block that includes a **return** statement if the value we pass is equal to 3. But what happens if the value is not equal to 3?

```
maybe 0;
// Error: Procedure "maybe" is missing a return value of type float. //
```

An error! Not so good. You could use the **else** part of the **if** statement like this:

```
global proc float maybe( int $value )
{
  if( $value == 3 )
    return 1;        // return 1 if $value is 3
  else
    return 0;        // return 0 for all other values
}
```

One possible solution

An even better way is to make sure that you only have one **return** statement in your procedure. This is another great defensive programming technique, because it keeps you from getting confused about what and when you are returning from your procedure. You can use a variable to store the value, and then return the variable as the result:

```
global proc float maybe( int $value )
{
  float $result = 0;  // set a default value of 0

  if( $value == 3 )
    $result = 1;       // return 1 if $value is 3

  return $result;
}
```

A better solution

Finally, you can still use the **return** statement by itself in procedures that don't return a result. This statement will immediately stop execution of the procedure and return to the caller. This is useful for making sure that the caller has supplied a valid parameter before doing something.

```
global proc divideNumber( float $number, float $divider )
{
  // you cannot divide a number by zero!
  if( $divider == 0 )
  {
    print( "// You cannot divide a number by zero!\n" );
    return;
  }
  float $result = $number / $divider;
  print( "// Result: " + $result + "\n" );
}
```

Using return without a value

You could also put a **return** at the end of the procedure, but there is really no reason to. Not doing so is a reasonable shortcut (saves you typing those incredibly painful eight key strokes) with no dangerous side effects. If your procedure is supposed to return a value, you cannot use **return** without a value. Maya will complain if you try.

Scripts

So, what the heck is a script anyway? A script is simply a collection of one or more procedures or code blocks in a text file, with a .mel extension. Scripts allow you to create complex systems of interrelated procedures for easy sharing and modification. Maya is made up of thousands of script files that interact with the actual binary executable programs to provide most of the functionality that we all use.

One thing you get from using a script file is a simple way to have Maya find your procedure. When you type a procedure name, Maya first looks through its currently loaded store of procedures. If it cannot find a procedure with that name, it starts looking through script files, looking for a file with the same name (plus the .mel extension). If Maya finds the file, it is loaded into the system, which results in loading all of the global procedures in that file, and if one of these global procedures is the command you typed, it executes.

Now, all this automatic stuff is great, but in practical usage it is not the most reliable system. Further, once a script has been loaded, it is always accessed from memory, so if you are trying to debug a script or change its functionality, you need a way to force it to be reloaded.

This is where the source command comes in. This is a built-in Maya command that will read a .mel file and load its contents into memory. Any procedures defined in the script will override any existing definitions that were previously loaded. Again, you don't need to provide the .mel extension when using this command. The best way to ensure you are using the most up-to-date version of your procedures is to source the script before executing it.

There are some limitations to redefining procedures. You cannot change the interface to your procedure once it has been successfully loaded. If you want to change either the input parameters or the return type of your procedure, you will need to restart Maya. If Maya spits your procedure back out because of syntax problems, however, then you can change its interface to your heart's content.

VARIABLES

We talked a lot about variables in Volume 1 of this tutorial from Issue #5 of *HDRI 3D*, but there are a few more details that pertain to how we use variables in procedures. Just like the procedures, there are two types of variables in Maya, global and local. Unlike procedures, you will almost always want to use local variables, not global ones. Using global variables is inherently dangerous, because if a global variable with the same name already exists, Maya uses the existing variable. This leads to problems if the variable is a different type, or if other commands in Maya are relying on its value not changing.

To make a global variable, use the keyword **global** before the variable definition. If you define a local variable inside of a procedure with the same name as a global variable, Maya uses the local variable instead. In general, try to avoid using global variables, they're just bad news. There is almost always a better way to do what you are trying to do. You may want to look into the **optionVar** command, which can allow you to store settings globally (that also get saved in the user's preferences).

Local variables are visible only from the level of the code block in which they are declared, as well as any sub-code blocks. This is the scope of the variable. So, if you declare a local variable in your procedure, only that procedure will be able to see that variable, and other procedures with their own local variables, even if they have the exact same name, won't interfere with each other. The input parameter names can be thought of as local variables for the procedure that are pre-filled with the values supplied by the procedure's caller.

For example:

```
global proc int proc1( int $value )
{
  int $copy = $value + 1;
  return $copy;
}

global proc int proc2( int $value )
{
  int $copy = $value * 3;
  return proc1( $copy );
}

proc2 3;
// Result: 10 //
```

Extensive use of local variables

In this example, the local variable names $value and $copy are used in both proc1 and proc2, but the values never conflict with each other or confuse each other, because they are all local variables. Local variables are only visible in the scope in which they exist. Notice that even when you call one procedure from another, the variables don't conflict. In this case, proc2 passes the actual value of the variable, not the variable itself to proc1.

Any time you have curly braces defining a code block, you define a new level of scope for your variables. As long as you access the variable inside of the curly braces that contain the variable definition, you will be okay, even if you access it in nested code blocks. However, you cannot access your variable outside of the curly braces in which it was defined.

```
global proc nested( int $nest )
{
  int $a = $nest;
  if( $nest > 5 )
  {
    string $l;
    int $i;
    for( $i = 0; $i < 10; $i++ )
    {
      // here we use variables from the scope of the if
      // block ($l) and from the scope of the whole
      // proc ($a), both of which are valid.
      $l = "test" + ($i +$a);
    }
    // this is valid, because $l was declared at the same
    // scope as this call
    print $l;
  }
  else
  {
    int $j;
    for( $j = 0; $j < 10; $j++ )
    {
      // again, this is cool, all variables are at the same
      // scope or above. Also, this different declaration of
      // another local variable with the same name but a
      // different type is ok, because the other declaration
      // was in a different scope which does not overlap
      // this one.
      float $l = ($j + $a) / 2;
    }
    // this will cause an error, because we are trying to
    // access a variable outside of the scope in which it
    // was declared (inside the for loop curly braces)
    print $l;
  }
}
// Error:    print $l;
//
// Error: "$l" is an undeclared variable. //
```

Examples of local variable scope

When you try declaring this procedure in the Script Editor, it rejects it with the error shown. Always make sure that your local variables are declared in the scope in which you access them.

EXPRESS YOURSELF?

Expressions are another place where you use MEL code. Expressions are quite different than procedures or scripts, though. First and foremost, expressions are usually directly tied to an object in your scene. Expressive for using programmatic control over one or more animatable attributes of an object, and are really useful for providing some kind of incremental, rhythmic, or random animation over time.

Expressions also provide a sophisticated way to interconnect the attributes of two different objects in more complex ways than simply connecting the attributes in the Connection Editor.

Expressions can use just about any MEL code, but you cannot actually declare a procedure inside of your expression. If you want to call a procedure from within an expression, make sure that it is already loaded or available. Expressions have very specific times when they are evaluated, generally when the current time changes. Particle objects can have expressions that are called before or after other dynamic simulation calculations. Also, I would strongly advise not trying to call UI code from expressions. It just ain't gonna work like you think it will.

There are two special keywords only available for use in expressions: **time** and **frame**, which, wonder of wonders, return the current time and frame on the timeline, respectively. You can use these keywords just like they were variables, but you don't need the **$** in front of them.

In general, all of the tips that apply to programming procedures and scripts also apply to expressions. You can use the Expression Editor to create and modify your expressions. Remember to give your expressions good names. If you are trying to find the expression controlling a bouncing ball in your scene, it's a lot easier if the expressions are named "bounceBall," "dripWater," and "shakeTable," than if they're named "expression1," "expression2," and "expression3."

INTERFACING WITH THE USER

All this is well and good, but if you can't get your buddies to use your scripts, it's sort of useless. Most people don't really like interacting with programs by typing long command strings, especially when the program has a nice interface with buttons and sliders and stuff. So what is a poor programmer to do?

Well, MEL provides a whole suite of functions that allow you to create your own interface goodies. You can create windows, buttons, sliders, and everything that you see in the Maya window. As a matter of fact, MEL commands generate the entire Maya interface, and this is one of the best ways to learn how to make your own windows.

One of the first things you will probably want to do is simply tweak the existing Maya UI to add some functionality that you need at your facility for your work. The best way to do this is to copy the **.mel** file that actually creates the UI elements in question and modify that copy to include your changes. Be very careful doing this, though! If you modify the original file and don't have a backup, the only way to get it back is to reinstall Maya!

Many scripts need their own, stand-alone interface. You can use MEL to create your own windows and populate those windows with all the controls necessary for your users to interact with your script. If you do go this route, it's very important to think about how you arrange your controls. Not only does this affect the choices you make as a programmer about how you are going to create your window, but it will affect your users' ability to actually use the scripts you write. If you have a poorly thought out or hard to use interface, no one will want to use your script, no matter how useful its functionality may be.

WHEN IT ALL GOES HORRIBLY WRONG

The chances of getting your script all right on the first pass are incredibly slim. Even with decades of programming experience behind me, I still make as many typos as anyone and manage to crash a computer at least once a week. Well, not really, but feel free to believe that if it makes you feel better.

There are several steps you can go through to protect yourself against your own mistakes. First (and this is a universal law about anything to do with computers), save early, save often! I mean it. Save save save! Save your scripts often, save your Maya scenes often, save everything all the time.

Don't program your scripts with important data. While you are debugging your scripts, you want to make sure that any scene data that you use for testing is either totally unimportant stuff (like a scene where you just whip up a few poly spheres), or is saved and backed up. This is especially true of any sort of MEL code you write that manipulates things outside of the Maya environment. You can interact with the entire computer via MEL, including saving files and even deleting files. These are definitely the sorts of things where you want to use junk data to test thoroughly before you try using real production data.

Test your code very thoroughly. Push its limits, feed it invalid data, use it repeatedly, and press your buttons faster than you think you should. Basically, do all those things that you as a Maya user start doing when things seem to be going wrong, and make sure that your code can handle it, or fails gracefully. When things work perfectly, most people don't notice much, but when things don't work well, then suddenly everyone complains all the time. So put your code into those situations where it is not going to work well, and make sure it either works well enough, or fails gracefully.

On a related note, you can pretty much expect that some bozo will ignore all of your beautifully written documentation and all of the clever descriptive names you have given your input parameters, and pass your code bad data. That bozo will inevitably be you, about three days after you think you have finished your script and start using it in production. So always make sure that your code checks for bad input and fails gracefully. See the example of using a return without a value above.

About half the time, when you make syntax errors in your code, Maya will catch you and warn you when it tries to load the script. If you are typing code into the Script Editor, this happens when you press the Enter key on the numeric keypad, and when you are writing a script, this happens when you try to source the script. In these cases, Maya does not load the data, giving you the chance to fix the problems and try again. However, some errors don't show up until you try actually executing the script, and this second sort of error can lead to data loss, hanging Maya, or potentially even crashing your computer.

LOOKING BACK AND MOVING ON

Remember the task that started all of this in the first volume, where I wanted to rename a bunch of objects? Well, here's a fully implemented procedure that will do it safely:

```
global proc changeName( string $oldPart, string $newPart )
{
  // we keep a count of how many changes are made, for fun
  int $count = 0;
  string $oldName;

  // loop through every node in the entire scene, looking for our old text
  for( $oldName in `ls` )
  {
    // substitute the new text for the old text in each node name
    string $newName = `substitute $oldPart $oldName $newPart`;

    // we only call the rename command if the names are actually different!
    if( $newName != $oldName )
    {
      // display what's going on to the user
      print( "rename " + $oldName + " " + $newName + "\n" );
      // catchQuiet will keep the script from stopping if the rename fails
      catchQuiet( rename( $oldName, $newName ) );
      // increment our count
      $count++;
    }
  }

  // print a summary of what happened to the user
  print( "// Found " + $count + " node" );
  if( $count != 1 )
    print( "s" );
  print( " with the text \"" + $oldPart + "\"\n" );
}
```

A procedure to change the names of many nodes in a scene

By making this a procedure, we are able to use parameters to supply the old and new name parts. You can even set the new part to an empty string ("") to have the old text simply deleted. Because these are parameters to a procedure, you can very easily re-use this procedure to change many different names in a scene, without having to edit the code. Finally, you could save this into a file called

changeName.mel, and share it with your entire team. All you need to do is type:

```
changeName oldname newname;
```

Syntax used to invoke our new procedure

Most everything in this script should be familiar to you by now. We iterate through every element in an array or strings returned by the command **ls**. For each element, we try making a new name from the old and the requested changes. Using the **if** statement for flow control, we only try executing the **rename** command if the name has actually changed. This is a good idea, because why spend the time calling a procedure that does nothing? Finally, at the end, we print a little summary.

There is a new command I have used called **catchQuiet**. As an exercise, why did I put that in? What would happen if it were not there?

This script still has many possibilities for improvement. Looping through every single node in the scene may be excessive. Remember, this will include all the history nodes, all the shader nodes, all the utility nodes, and in a large scene, this adds up. Also, there is no progress report to the user about how far along the looping is, and no way to cancel it once it has started. All of these things are possibilities you can add to your scripts, and probably should add to scripts that you want to share with the world.

Also, you could design a graphical interface to this. Instead of having to type a command into the Script Editor, you could design a little window with fields to type the old and new parts, and buttons to start or stop the operation. These are further exercises for the reader.

CONCLUSION

Obviously, there is much more to programming MEL than can come across in two articles. People spend their lifetimes learning to program, and MEL is just as sophisticated a programming system as just about any other language. There are nuances and subtleties that you will only get the hang of after years of practice.

The best way to learn is to just dive in and start coding. Don't be afraid of lines of code; they're no more complicated than the interaction of the nodes in a shading network, or the relationships of a skeleton and IK handles. Actually, computer scientists will remind you that those systems are simply different types of programming systems, so MEL is not as foreign as you may think it is, even if you have never programmed a line of code in your life.

So go forth and program, I say. Even writing the simplest of scripts will give you a new understanding of Maya that will only help to improve your work as a modeler, animator, lighter, TD, or whatever your normal job title may be.

Finally, always remember: when something bad happens, just blame the OS. It's probably their fault, anyway.

Useful Hints to Remember

- Preface your output with the comment characters whenever possible.
- Give your procedures useful and descriptive names.
- Make procedures global unless you don't want users to use them.
- Try to give your procedures a single return point.
- Make variables local unless you absolutely need to use global variables.
- Save Early, Save Often!
- Test with dummy data.

Robin Scher *has been animating for nearly ten years, and has been programming for over twenty, and is the author of the very popular Smedge render distribution system. He also enjoys taking photographs on occasion, and has just finished traveling around the world while programming the next version of Smedge. Right now, all he wants to do is to get some sleep.*

XSI base.com

Softimage XSI Community

Forums
Jobs
Interviews
Plugins
Shaders
Contests

www.xsibase.com

Self-Service Render Farm

ResPower Super/Farm™

24x7 Access

Over 650 Render Nodes

3dStudio Max

LightWave 3D

Maya Unlimited

Vue Professional

V-Ray

Over 3 THz

Ultra-High Resolution Renders
(In Excess of 30,000x30,000!)

mental ray®

Fibre-Optic Internet

Brazil R/S

30 Tons of Air Conditioning
(That's 360,000 BTU/hr!)

Vue Infinite

www.respower.com

TeraHertz At Your Fingertips

RenderMan Artist Tools: DIVING IN

By Peter Hartwig
3D Artist
Denmark

One of the most daunting things for me, when I started working on big time feature films a few years back, was the fact that all of a sudden I was using RenderMan. There wasn't much documentation out there that would actually tell me the stuff I needed to know in order to start working. Sure, you can find books stuffed with info about the scene description language, the shader language, etc., but the hows and whys of using RenderMan with Maya were eluding me.

As it turns out, it was not all that hard to learn, once I got on the right track, which is why I am writing these articles ... to get you started. I have to tell you that I am far from a hardcore RenderMan shader writer or the like. This is why the articles will tell you what you need to know to actually work with RenderMan in production as a lighting TD.

Why another renderer? There can be numerous reasons why you would sometimes choose another renderer. Some provide better lighting controls, some are faster, and some have features available that you cannot find anywhere else. With RenderMan, the key features have to be flexibility and robustness. When you absolutely, positively have to render every last polygon in the scene, accept no substitute.

For flexibility, three main things really do it for me. First of all, there's the shading language. Basically, this is like scripting how your shaders work. You can do basically anything with it, and if you run into something you cannot do, you can add that feature via C++. Then there's the fact that RenderMan uses a rib file format. Rib files are ASCII documents (they can be binary as well, but that takes away from my flexibility rant) that describe the whole scene. After you generate your rib files, you have the ability to change them externally (for example, using a python script) before submitting the render. You could change cameras, models, lights, or whatever you need.

Last of the flexibility features is the ability to use Tcl scripting in most every parameter of your shader, once it's in Maya. Imagine that you have 100 objects that all need the same base shader, but the Diffuse and Bump maps are specific to the object. Instead of creating 100 instances of the shader and going through them by hand, you could tell RenderMan to use something like the object name or an extra attribute as the filename at render time. We will be getting into all of this later on, so don't worry if it's all gibberish at this point.

We will be using Pixar's lovely implementation of RenderMan, PRMan version 12.5, for this article. Thanks to Wendy, Ray, and Chris at Pixar for letting me borrow a license.

The parts of RenderMan that we will be using are MTOR, Slim, Alfred, and IT. MTOR, or MayaToRenderMan, is the main Maya interface for RenderMan. It handles the scene translation, etc., and makes our lives a lot easier. Slim is the RenderMan version of a Shader editor. It's node-based and used for creating shaders and materials. Then we have Alfred, which is the remote execution server ... this basically means it sends the jobs off to the renderfarm or your local machine, without you having to launch renders from commandline. Last but not least, we have IT. IT is the Framebuffer. Simply put, it is the window that pops open and displays the image you have rendered.

Let's start with MTOR. Make sure that you have the plug-ins loaded so the RenderMan menu shows up when you are in the rendering toolset mode in Maya. The things we will be concerning ourselves with to begin with in the RenderMan menu (Figure 1) are:

- **RENDER.** This is simply the button that starts the render.
- **RENDERMAN GLOBALS.** This is where we set all the parameters we need when rendering. This controls resolution, output, quality, number of machines to render on, what features to use, etc.
- **SLIM.** These are the slim tools used for visually building shaders.

Try clicking Render. Quite quickly you should have a couple of windows popping up. The first one is Alfred, which shows you that the job is rendering; if there are any errors and such, they come to you here. Then you have IT showing you a blank image. This makes sense, as there is nothing in the scene we just rendered. (Figure 2)

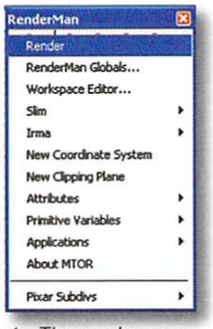
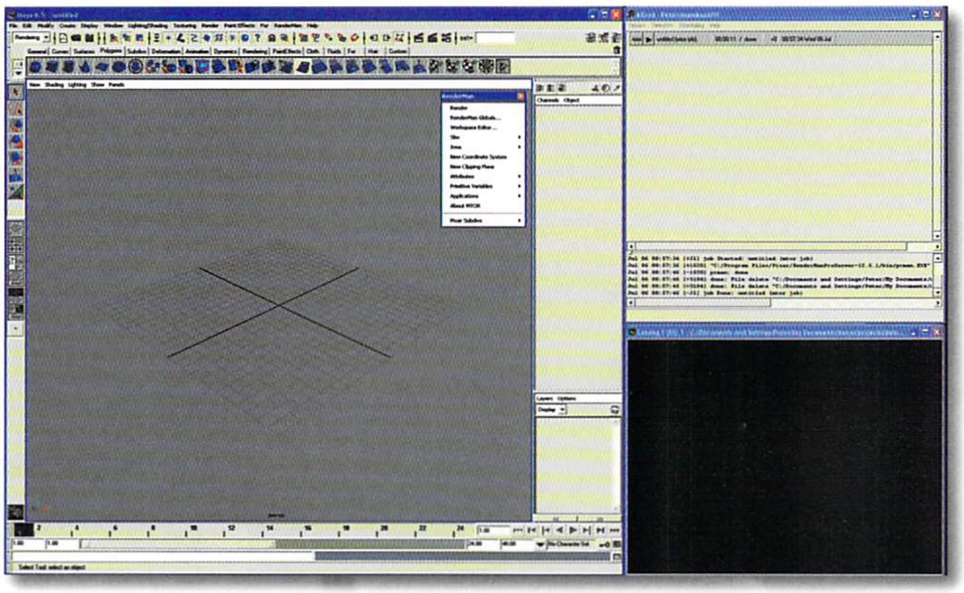
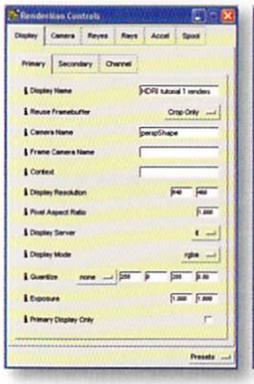


Figure 1– The render menu

Figure 2– Alfred, IT, and Maya



Screen Shots of this article can be downloaded at www.HDRI3D.com


RenderMan Controls

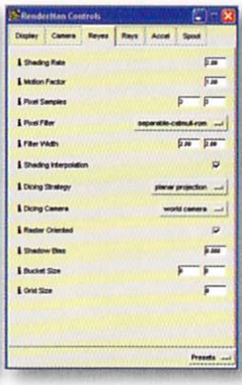
Display | Camera | Reyes | Rays | Accel | Spool

Primary | Secondary | Channel

Display Name: HDRI tutorial 1 renders
Reuse Framebuffer: Crop Only
Camera Name: perspShape
Frame Camera Name:
Context:
Display Resolution: 640 480
Pixel Aspect Ratio: 1.000
Display Server: it
Display Mode: rgba
Quantize: none
Exposure: 1.000 1.000
Primary Display Only

Presets

Figure 3– RenderMan's RenderGlobals


RenderMan Controls

Display | Camera | Reyes | Rays | Accel | Spool

Shading Rate
Motion Factor
Pixel Samples
Pixel Filter: separable-catmull-rom
Filter Width
Shading Interpolation
Dicing Strategy: planar projection
Dicing Camera: world camera
Raster Oriented
Shadow Bias
Bucket Size
Grid Size

Presets

Figure 4– The Reyes tab

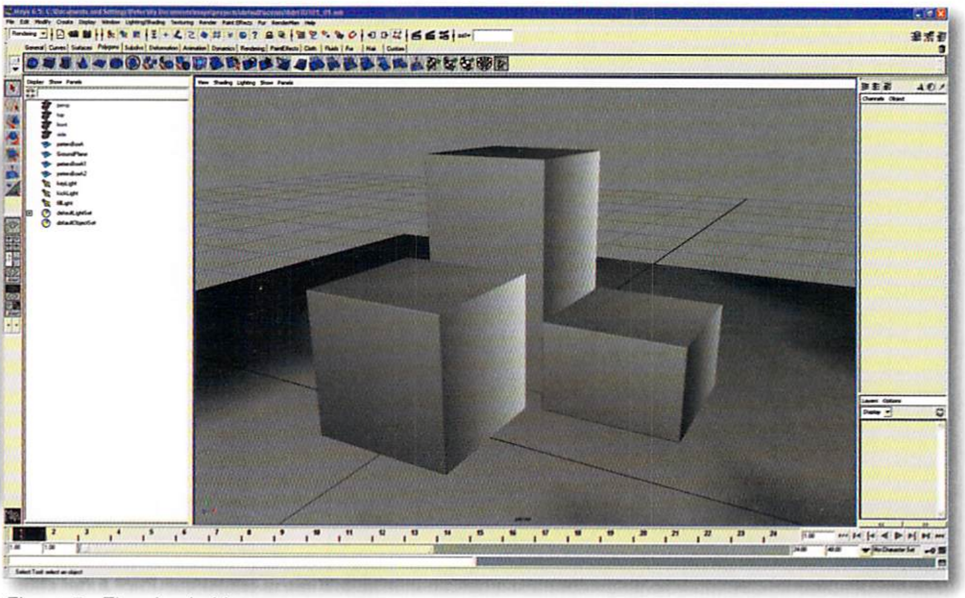


Figure 5– The simple Maya scene

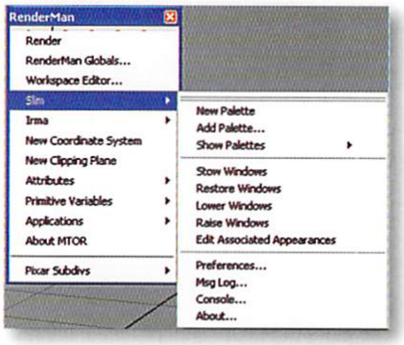


Figure 6– Finding the slim menu under the RenderMan rollout

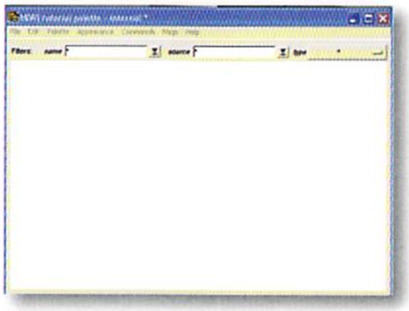


Figure 7– An empty Slim palette

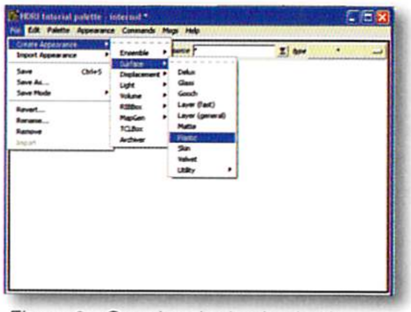


Figure 8– Creating the basic plastic material

Now go into the RenderMan Globals menu. I want to set it up for this little tutorial. The first thing we see are all the settings for the primary output. RenderMan allows you to do multiple outputs in one go, which we will look at later. Set the display name to something that makes sense, like "HDRI tutorial 1 renders."

You can see that we are using the perspShape as camera, which is fine for now. Go on down to the Resolution. I will leave it at 640 x 480, but you can scale it up and down depending on your system speed, etc. (Figure 3)

We set the display server for IT at the moment. That's fine for now, since we are rendering to the IT window, but when we need to render to files, this is the place to set that up. As you can see in the dropdown, RenderMan supports quite a few formats.

Let's quickly skip over to the Reyes tab. (Reyes is short for Render Everything You Ever Saw, by the way.) To begin with, I will just tell you about Shading Rate. This is effectively the anti-aliasing. For production renders, you will usually want this smaller than 1, but 5 is good for test renders, as it speeds up rendering a lot. Now close the RenderMan Globals window. (Figure 4)

Now, before looking at Slim, I think it's time to set up a simple scene. Three boxes on a plane should do. Not much to it, so I will let you do that on your own. Also, add three spotlights: a keylight, a fill light, and a kicklight from the back. Remember to name everything. Now when you press render, you will get an image! By the way, I need to remind you that, in this first part of the tutorial, you will be learning how to get around RenderMan, so don't expect blockbuster visuals to start with. We need to get you up to speed first. (Figure 5)

Now, in the RenderMan menu, go to Slim, and New Palette. In Slim, we have palettes to organize appearances, which are what we can assign to objects. Appearances consist of a shader and some parameters that have been set, as well as maps, more shaders, procedurals, etc., to make up the look. Palettes can be internal, local to the file, or external, where you can share them between scenes. (Figure 6)

Go to File in the palette window and click on Rename. Call the palette something like HDRI tutorial palette. You will often have loads and loads of palettes in a scene, so make an effort to keep things named in a clear fashion. (Figure 7)

Now let's create an appearance! Click *File>Create Appearance>Surface>Plastic.* As you can see, there are many different types of appearances, but we will, of course, start simple. You now have a node called plastic. If you click it once and press 'r' on your keyboard, the thumbnail will render. Double-click it to open its parameters in a window.

At the top right, you have some parameters for the thumbnail render. Shading Rate again is the anti-aliasing, controlling the overall quality. Then you have controls for the size and type of object in the thumbnail viewer, and a control to set what frame you will be looking at. This is good when using animated textures, of course.

First of all, set the name to "bluePlastic." (Figure 8) Leave the

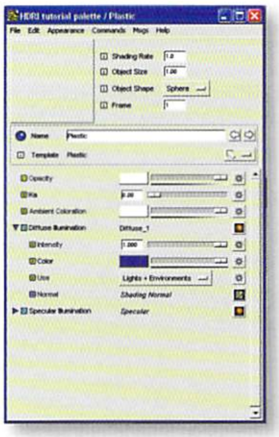


Figure 9– The blue plastic shader

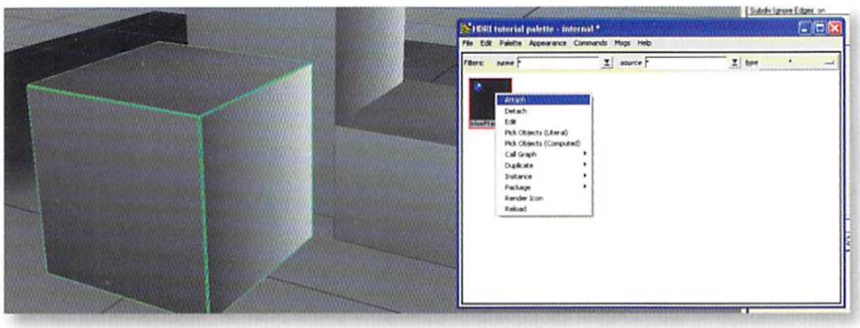


Figure 10– Image of blue box in IT

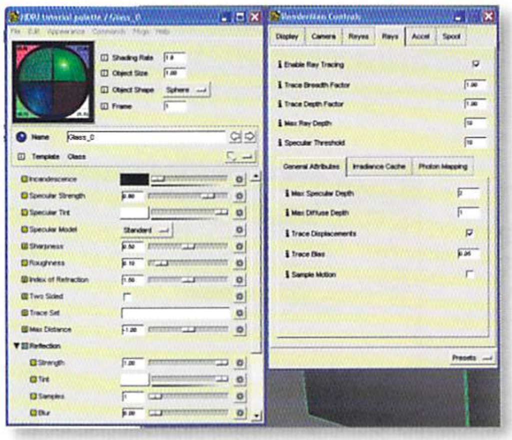


Figure 11– Left side is the glass shader and right is enabling raytracing in the RenderMan Globals

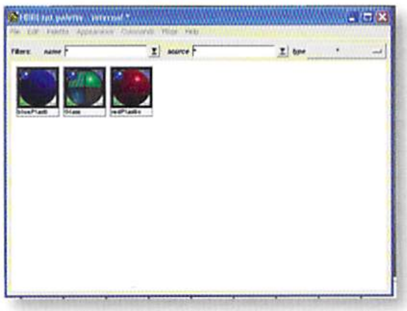


Figure 12– All the shaders in place

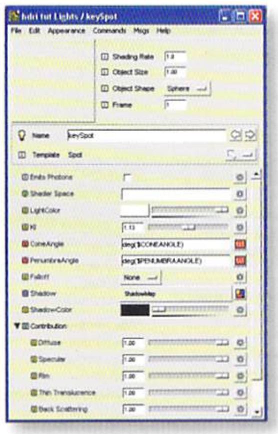


Figure 13– The keyLight shader

Opacity at white, to make sure there is no transparency. Set Ka to 0, which is the ambient light contribution to the surface. Now open the Diffuse Illumination tab, set the color to a nice blue, leave everything else as is, and close the window. (Figure 9) Now select one of the boxes in Maya, and click on the bluePlastic icon in the palette. Right-click on bluePlastic and select Attach. (Figure 10) The box now renders blue!

Go create another appearance. This time, use the one called glass. (Figure 11) Remember to rename it to something that makes sense. Now attach it to another box. When you render, the new shader is all black. Raytracing is turned off in RenderMan by default. Go back to RenderMan Globals in the RenderMan menu. Go to the Rays tab, and click on the Enable Raytracing tick. Now render again, and see the glass render as it's supposed to be.

Right-click on the bluePlastic icon and select *Duplicate>Node*. Open this new node, rename it redPlastic, make the diffuse color red, and assign it to the last box. (Figure 12)

Now it's time to work with the lights a little bit. Create a new palette and rename it "HDRI Tutorial Lights." In this palette, create a new appearance, but this time, instead of going into Surface, go into Lights, and select Spot. (Figure 13) Quickly rename it to keySpot, and select your key light in Maya and attach the light shader. Shaders and appearances can be lights as well as surfaces. In the window for the keySpot, we have the light color followed by Kl, which is the multiplier for the light color. This is where you set the intensity of the light. The appearance already has expressions set up to grab the cone angle and penumbra angle from the Maya light. Now, to add a shadow to the light, click on the icon on the far right where it says Shadow, go down to Maps, and select Shadow map. We will not go into this any more in this tutorial, but there is a lot of tweaking that can be done in the Shadow map.

Further down, you have a list of parameters under contribution. For the key light, I want you to keep them all at 1, as they are.

Now create two more light appearances. One called fillLight, that has no specular contribution, and one call kickLight, that has a very low diffuse contribution. Attach them to the two other lights.(Figure 14)

All right then, your very first RenderMan render! (Figure 15) One last thing I'd like to show you is the SubDs. Simply put, they are great.

Create another box, and in the RenderMan menu, select *Pixar Subdivs>Mesh as Subdiv*. Click the Options box for that. (Figure 16) Here you have all your regular options to play around with. One option that you might want to tick on while learning your way around RenderMan is Expose Mesh to Maya. This lets you see the mesh in the viewport. Click Apply and behold, the box is now rounded! Amazing! You will notice that the original box is now in X-ray mode. If this bothers you, go into *RenderMan>Pixar Subdivs>Toggle Object X-ray*, with the object selected. Try rendering the image now, and notice how nice and smooth it renders. (Figure 17)

The things we have gone through now should be enough to get you started. Try playing around with all the parameters you can find, and see what they do. Of course, we will go into a lot more detail later on, but now let's briefly look at one of the main features of RenderMan: the RenderMan shading language, or RSL.

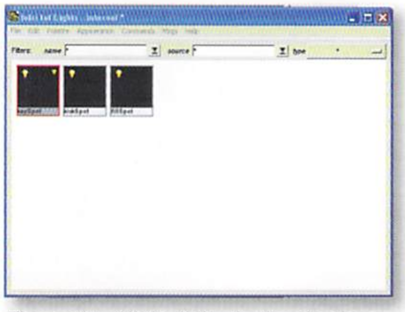


Figure 14– All the light shaders in place

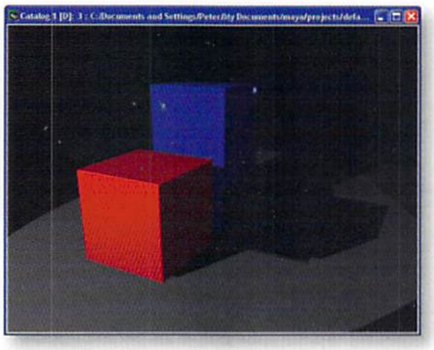


Figure 15– The rendered image

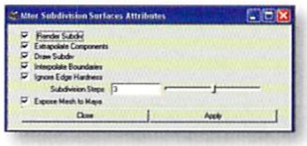

Alter Subdivision Surfaces Attributes
[✓] Render Subdiv
[✓] Extrapolate Components
[✓] Draw Subdiv
[✓] Interpolate Boundaries
[✓] Ignore Edge Hardness
Subdivision Steps 3
[✓] Expose Mesh to Maya
Close Apply

Figure 16– The Pixar Subdivs menu

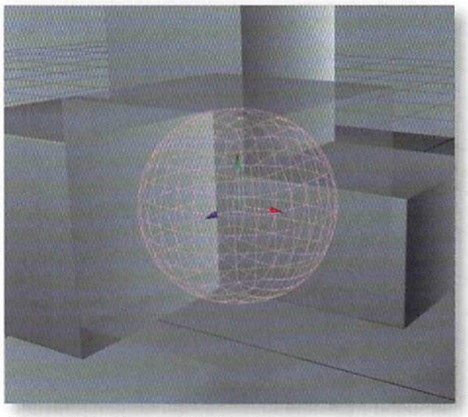


Figure 17– The subdivided box

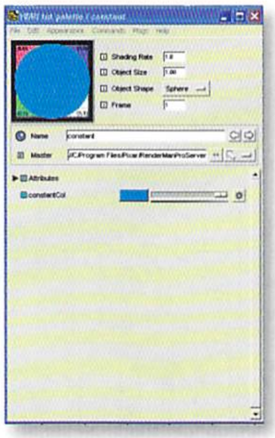

Shading Rate 1.0
Object Size 1.00
Object Shape Sphere
Frame 1
Name constant
Master //C/Program Files/Pixar/RenderManProServer
Attributes
constantCol

Figure 18– The myShader parameters

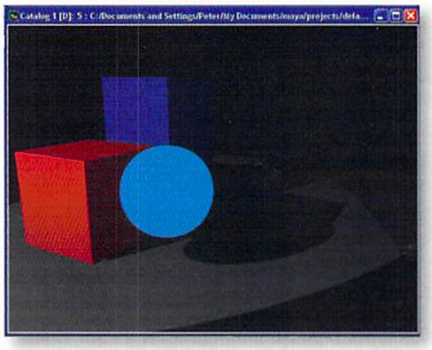


Figure 19– The rendered image with the myShader shader attached to the subdivided box

I take it anyone reading this article knows a thing or two about 3D graphics and has been working with them for a while at least. So you know all your standard shaders … you have your phong, lambert, anisotropic and what not. You can combine them in loads of ways and archive almost anything you want. *Almost* … sometimes you need to use a procedural pattern that is not included in any package, and you might need the light to react to the surface in a different way than usual. Or maybe you're replacing each null object with 10.000 strands of hair? Enter the RSL.

Writing your own shaders means big time flexibility. It will let you control every aspect of your rendering, expose the parameters you need, and render the passes you need.

I will guide you through a simple constant shader in this first part, but in the next issue, we will be taking it further…

Open a text editor and type in this:

```
  Surface myConstant (color
myConstantColor = 1)
  {
        Ci = myConstantColor;
  }
```

save the file as myConstant.sl

The first line defines that we are writing a surface shader. It is called myConstant, and it exposes one parameter, which is a color slider called myConstantColor. This is set to 1, which is white.

The curly brackets surround the definition of the shader. In this case, Ci, which is the RSL variable for output color, the color seen by the renderer, is set to the value defined by myConstantColor.

Now let's see it in action. To make life simple, I have saved my file in the RenderManProServer/bin directory, but we will set up the environment a bit better in the next issue. Navigate a shell or dos prompt to that directory and type 'shader myConstant.sl' . This should return a message saying 'myConstant: compiled', and leave you with a new file called myConstant.slo. This is the compiled shader. (Figure 18)

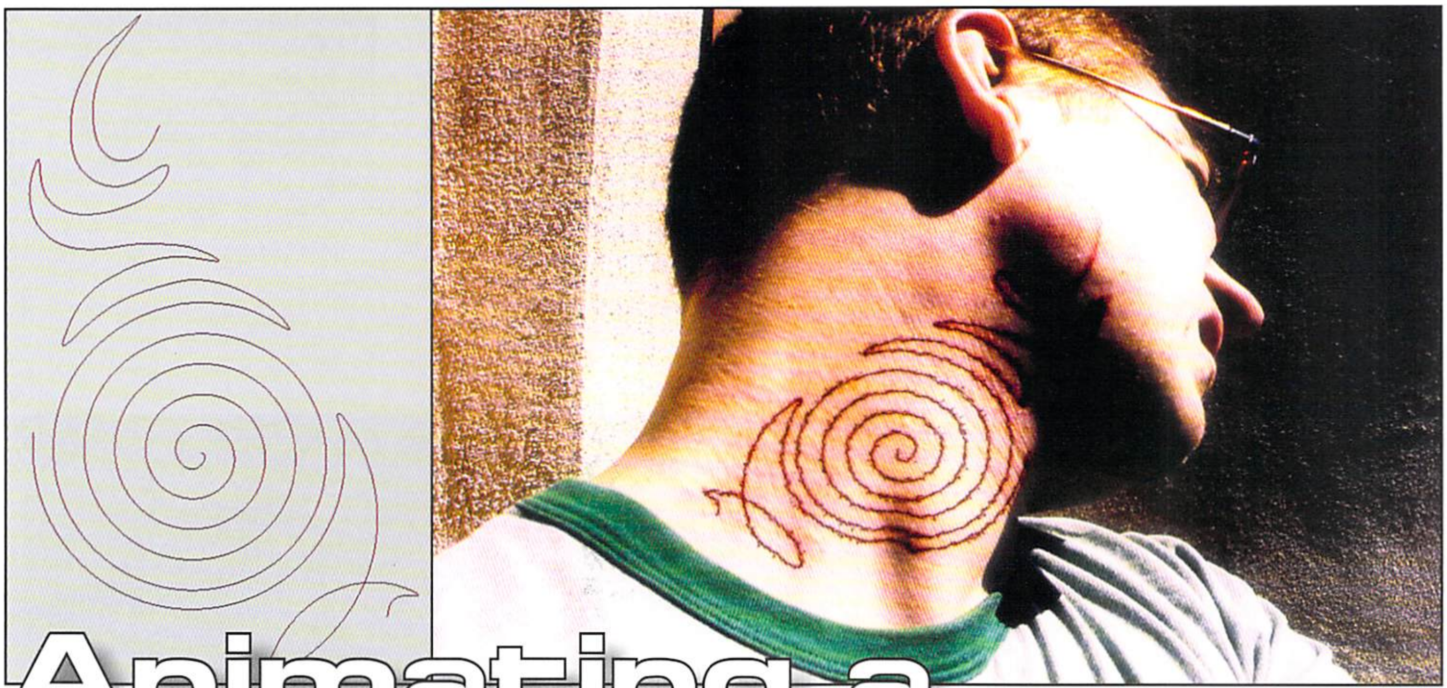
Go back into Maya, open one of your Slim palettes, and select *File>Import Appearance>Import Appearance*, navigate to the directory where you just compiled your shader, and select the myConstant.slo file. Voilà! You should have a rollout called Attributes (that we will not be worrying about right now) and a color slider called MyConstantColor. Try applying this shader to an object in the scene, set the color you want, and render. (Figure 19)

Congratulations, you have just written your first RenderMan shader!

In the next issue, we will take much deeper look at writing shaders, and in the process we will write quite a few shaders that you can use in the future. After that I will be looking at some production tips and tricks for MTOR and the RenderMan artist tools, to get you more or less up to speed on these things.

Until next time, I hope you've enjoyed this.

Peter Hartwig *grew up in Denmark, where there weren't that many choices (at the time) for learning 3D, so he taught himself and went on to the renaissance center in Nashville for a few months after he finished high school. Recently, he has been freelancing, both in the US and in Denmark, but also in London and Istanbul. Before writing this article, he finished work on Charlie and the Chocolate Factory, a remake of one of his favorite films of all time. As he says, "life is good …"*



Animating a Crawling Tattoo

By Eric Keller
Scientific Animator
West Hollywood, California

PART ONE: ANIMATING STROKES IN MAYA'S PAINT EFFECTS

If you've ever used Maya's Paint Effects module, it has probably occurred to you at some point or another that it is not the world's most intuitive animation tool. Designing a brush can take an awful lot of twiddling and test rendering before you get the results you want. Then, when it comes time to animate the brush - well, there's more twiddling. On a recent animation job, I used Paint Effects to create vines that crawled along video of a woman's arm (with some help from my friend, Nate Homan, who offered some suggestions to me that made the animation of the vines a much more pleasant experience. This tutorial is going to discuss some techniques similar to the ones I used on that job. My goal is to pass some of those tips and ideas on to you so that, the next time you decide to use animated brush strokes with Paint Effects, your life will be that much easier.

In the Issue 2 of *HDRI 3D*, I used the neon brush in Paint Effects to create some retro-futuristic computer displays. The brush strokes I used were rather simple, and animating them was just a matter of setting keyframes on the Min Clip and Max Clip sliders found in the End Bounds folder in the Strokes Shape Attribute Editor. Since the stroke was a simple beam of light with no leaves, twigs, or flowers, this technique worked very well. In this tutorial, however, the brush I'm using will have a little more detail to it, and the way in which I want to animate it is very specific; animating the End Bounds just won't do the job. Before I get into the actual tutorial, I want to do some quick comparisons to see how I can achieve the effect I want.

In this animation, I want my strokes to grow. If I animate the End Bounds using the Min and Max Clip sliders, I won't get a growing effect; I will get more of a quick broad brush stroke effect. To illustrate this, I made a simple NURBS circle with 12 sections.

1 I duplicated the circle curve and placed them side-by-side on the grid.

2 I opened up the visor *(Window>General Editors>Visor)*, and in the Plants folder under the Paint Effects tab, I selected the Leafy Twig stroke.

3 Under the Rendering menu set, I chose *Paint Effects>Template Brush Settings.*

4 I switched to the Select tool and grabbed the two circles.

5 I then chose *Paint Effects>Curve Utilities>Attach Brush to Curves.*

6 The Leafy Twig stroke is applied to both circles. In the perspective view, a simple representation of the leafy twigs appears growing off of the curve.

7 In the Attribute Editor tab for leafyTwig1 and leafyTwig2, I brought the Global Scale slider down to 1.5. You can see the results in **Figure 1.**

Now that we have some strokes, let's look at animation possibilities.

1 I opened the outliner and selected stroke1.

2 In the Attribute Editor, I clicked on the strokeShape1 tab and then opened the End Bounds folder.

3 I set the Max Clip attribute to 0.

4 I set the timeline to Frame 0.

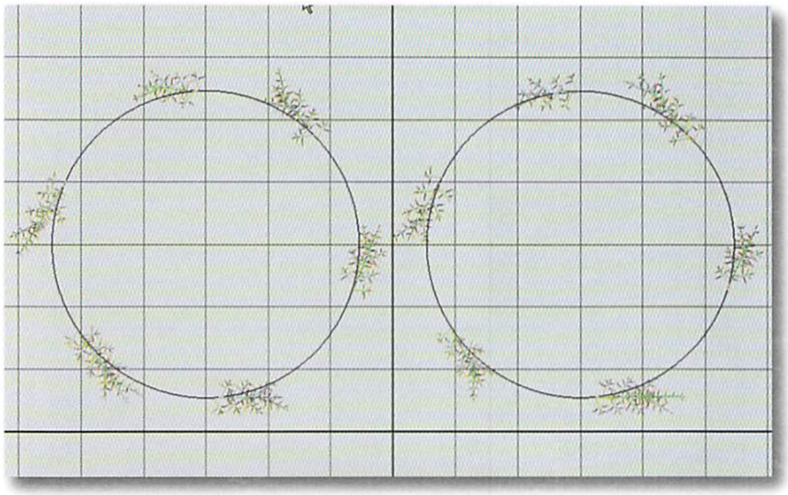


Figure 1– Two Nurbs Circles with the Leafy Twig brush applied as seen from the top view.

5 I right-clicked on the Max Clip label and chose "Set Key."

6 I moved the time slider to 60.

7 I moved the Max Clip slider to 1.

8 I set another key.

Playing back the animation, it becomes obvious that it's not really growing. The leaves are just being stamped out along the curve. The Min and Max Clip sliders are there to help you position a stroke on a curve. I don't think the intentions of the Min and Max Clip sliders are as tools for animating the growth of a stroke. It just so happens that, for simple strokes, they are a very handy couple of attributes to use for animation. I'll demonstrate why...

9 In the outliner, I selected stroke2.

10 In the Attribute Editor, I selected the tab labeled "leafyTwig2."

11 I scrolled down to the Flow Animation folder and opened it up.

12 I checked the boxes for stroke Time and Time Clip.

13 I set Flow Speed to 2.0.

With the range of the time slider set at 60 (@ 24 fps), I played the animation. If you've followed these steps, you can see the difference in the way stroke1 and stroke2 are animated. Stroke2 actually grows along the curve. Most of the detail of the growth is built into the brush, and animation is achieved by turning on Stroke Time and Time Clip and then plugging in different numbers into the Flow Speed, Start Time, and End Time.

Start Time, as you may have guessed, sets the point at which the stroke starts to animate. One might guess that End Time would refer to the time the stroke reaches the end of the curve, but one would be wrong. End Time sets the time at which the end of the stroke starts to disappear along the curve. Flow Speed is a multiplier that determines the speed of growth; fortunately, that quite is straightforward.

Here's a question, though: Why are the Start and End Time sliders calculated in seconds? Is it really that much fun to have to constantly convert frames to seconds when you want to pick a specific starting frame for the animation?

When I select both stroke1 and stroke2 and open up the Graph Editor, I notice something else. The curve for the keyframes set on stroke1's Max Clip attributes are right there for me to tweak, just like any other keyframes. However, there is nothing on the Graph Editor to represent the animation for stroke2's flow animation. Apparently, the only way to adjust this stroke's growth animation is by plugging in different values for the Flow Speed. That's not terribly flexible or intuitive. Fortunately, there's an easy way to get some keys on the Graph Editor that can be used for refining the animation:

1 I went back to the settings for Flow Animation in the Attribute Editor for leafyTwig2.

2 I right-clicked on the box labeled Time and chose "Break Connection." The box changes from yellow to white, and the value switches to zero. My brush also disappears.

3 I right-clicked on this box, again, and chose "Set Key." The box turns orange.

4 I set the Time slider to another value and chose "Set Key" again on the Time box. There are now two keyframes for the Time value; both are set to zero.

5 I set the Flow Speed of the stroke to 1.

6 In the Graph Editor, I chose *Stroke 2>strokeShape2>Brush.* There are a couple keys at zero at different times. Now I have an easy way to tweak the timing of this stroke's growth.

7 I selected the first key, and I entered 0 into both boxes in the stats box at the top of the Graph Editor. This puts the first key at a value of 0 for Frame 0.

8 I selected the second key (which is also at a value of 0), and I typed 60 and 60 into both boxes.

When I play back the animation, both strokes are animated, but the second stroke grows much more slowly. This is because I changed the flow speed from 2 to 1; thus cutting its speed in half. The easy way to fix this is to just move that key from 60 to 30 so that it reaches a value of 60 in half the time. Finally, I have to make sure that, I set *Curves>Post Infinity* to linear in the Graph Editor. I also make the post infinity curves visible by choosing *View>Infinity.*

So what exactly have I done? Simply put, I've set keyframes on the timeline for the stroke's Time value. Now the stroke understands time on a different scale than the rest of the objects in the scene. Most importantly, I have something on the Graph Editor that I can easily play with to adjust the timing of the growth. I can change the curve's interpolation, add keys, ease in and ease out, and I don't have to constantly convert frames to seconds to figure out where the growth will start. (Thanks again Nate!)

I keep the Flow Speed at 1 and the Start Time at 0 just to reduce the number of variables affecting the growth of the stroke. I handle all of the animation for this stroke by adjusting the keys on the Time value on the Graph Editor.

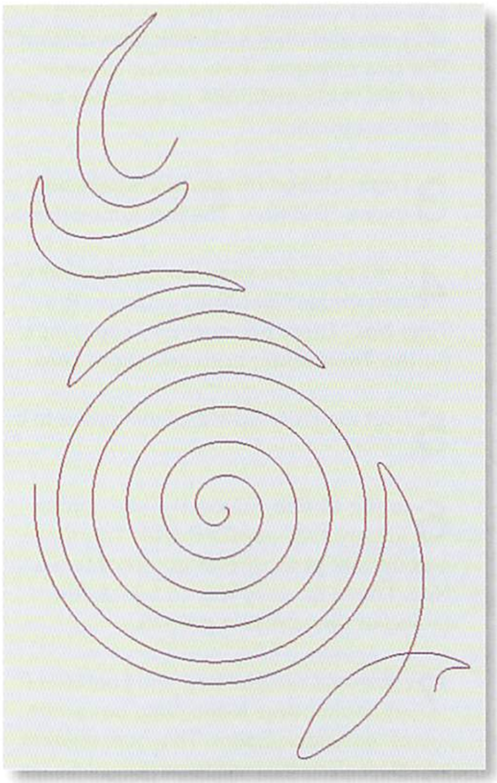
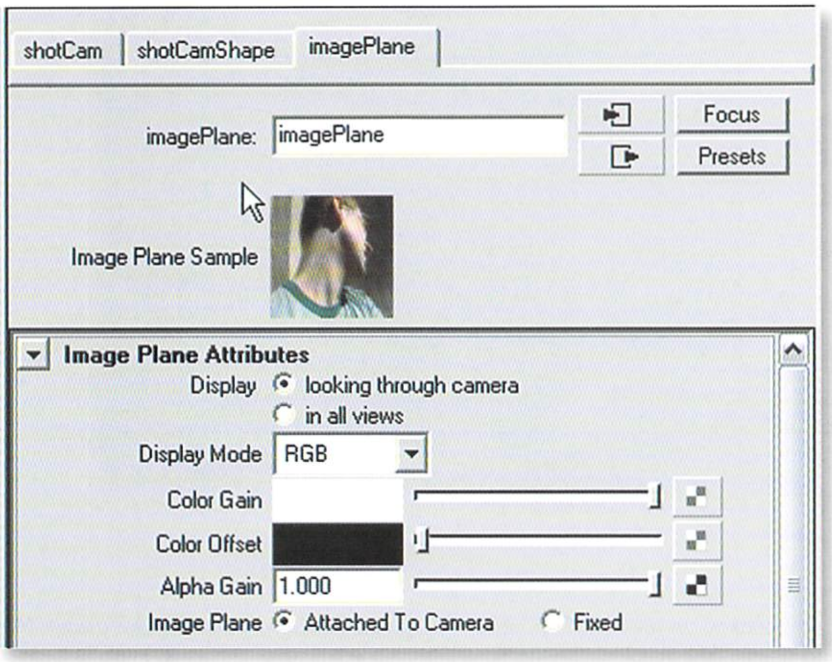


Figure 2– The design for the tattoo in NURBS curves.



imagePlane: imagePlane

Image Plane Sample

Image Plane Attributes

Display [✓ looking through camera] in all views

Display Mode RGB

Color Gain

Color Offset

Alpha Gain 1.000

Image Plane [✓ Attached To Camera] Fixed

Figure 3– Settings for the image plane attached to the shotCam.

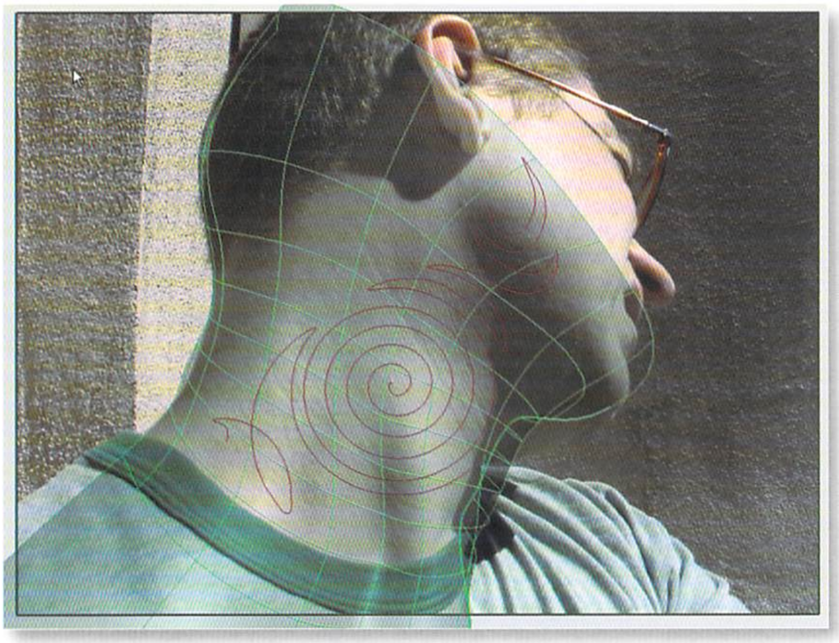


Figure 4– I used some simple geometry to match the contours of my neck in the photo. The tattoo curves are positioned in front.

PART TWO: ANIMATING A CRAWLING TATTOO

Now I'm going to demonstrate this technique in an animation. I'll also get into some ways that I customize the leafy twig brush to make it look - well, less like a leafy twig.

The inspiration for my first Paint effects tutorial (*HDRI 3D* Issue 2) came from the computer interfaces in the original *Star Wars Episode IV*. For this tutorial, my inspiration is coming from a more recent (but no less nerdy) source: *Buffy the Vampire Slayer*. Actually, it's closer to an effect that you would have seen on *Angel*, but hey, it's all the same universe.

I'll create the effect of a demonic or otherworldly tattoo burning into the skin on my neck from some invisible force using Maya's Paint Effects and some simple compositing in After Effects. Ick. (This is what I get for reading those demonic books I never returned to the Sunnydale High School Library.)

I'm using a digital photo I took of myself as a basis. These techniques are slightly overkill for a still photo, but they would work well on some video with a little movement and maybe some match moving. That's too much for this tutorial, but this should give you some ideas for more in-depth exploration.

MAKING THE TATTOO:

1 The first thing I did was design a tattoo in Maya using some NURBS curves **(see Figure 2)**. Alternatively, the tattoo could be designed in Illustrator or Photoshop and imported into Maya as an EPS file. NURBS curves are not exactly the most elegant drawing tools in the world.

2 I then created a new camera and named it shotCam.

3 I set my perspective window to shotCam and created an image plane by choosing (in the camera's view menu at the top of the perspective window) *View>Image Plane>Import Image...*

4 In the file chooser window, I located the image file and loaded it up.

5 In the Attribute Editor for the imagePlane (the easiest way to get to this is by selecting the shotCam and clicking on the imagePlane tab in the Attribute Editor), I made sure that the Display was set to Looking Through Camera, Display Mode was on RGB, and the image plane was attached to the camera **(Figure 3)**.

6 While looking through the shotCam with the image plane visible, I

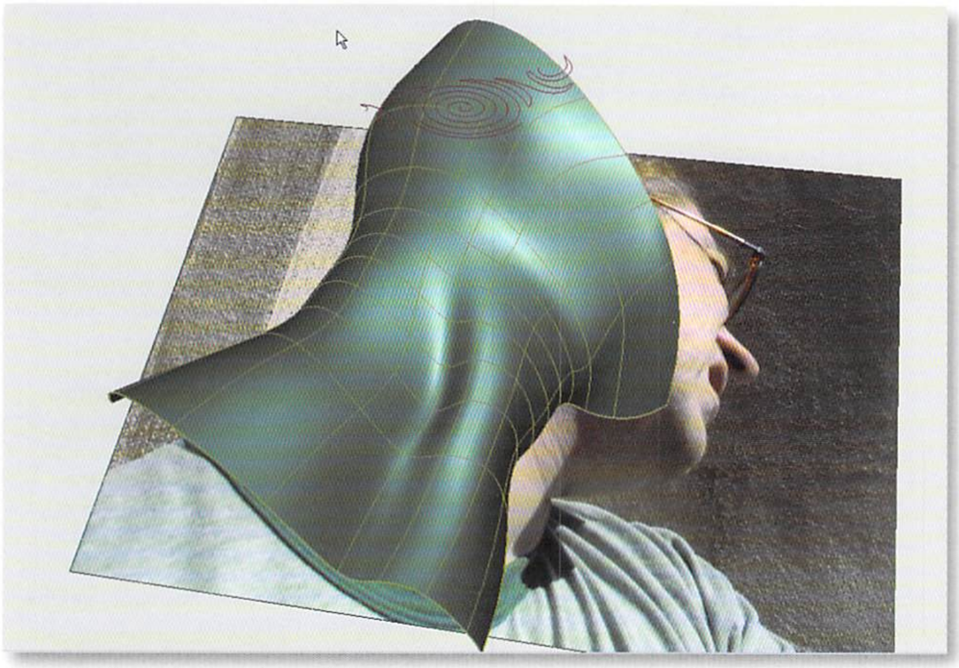


Figure 5– The contours of the neck have been exaggerated so that the 3D quality of the tattoo really comes through in the final composite.

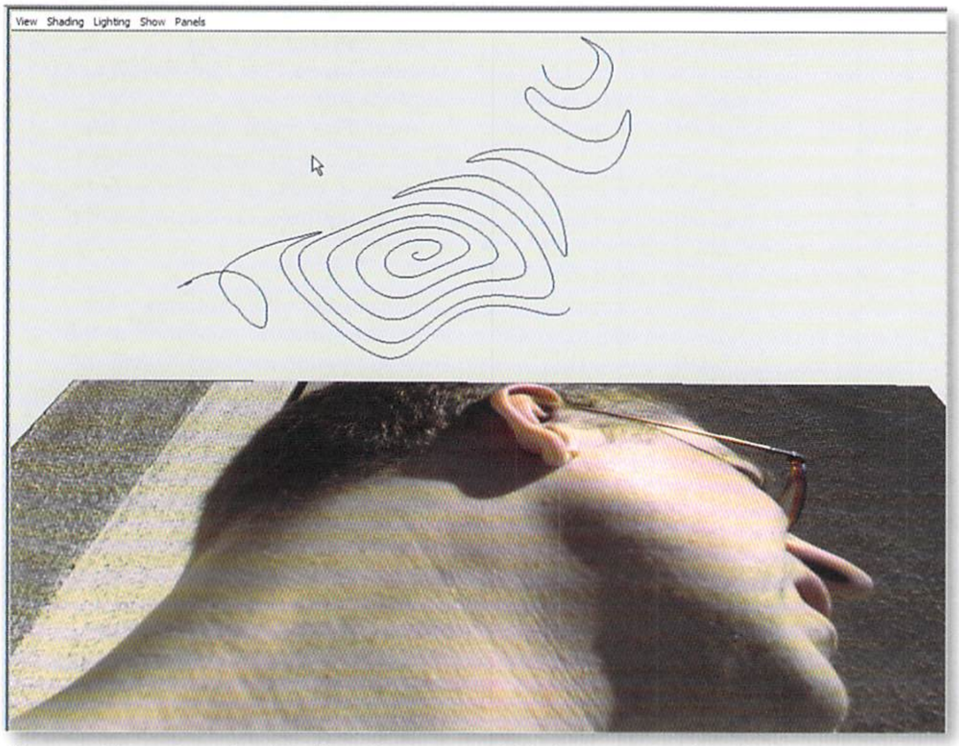


Figure 6– The projected and duplicated curves are visible in the perspective window. You can see how they conform to the neck contours.

scrolled around until the tattoo curves were positioned where I wanted the tattoo to be drawn on my skin.

7 I locked off shotCam by shift-click selecting its Rotate and Translate attributes in the Channel Editor, and then chose "Lock Selected" from the menu when I right-clicked over the highlighted channels.

8 I then modeled a simple NURBS surface from a cylinder, based on the contours of my neck, as shown in the photo **(Figure 4)**. You can see in **Figure 5** how I exaggerated the contours of my neck. This is because I wanted to make sure that the curves looked as though they were moving over a three-dimensional surface in the final composite. Sometimes this requires some overcompensation in the model.

9 Once sure the model looked the way I wanted it to and the tattoo was positioned correctly, I selected the tattoo curves and then the surface, switched to the shotCam, and chose (from the modeling menu set) *Edit NURBS>Project Curves on Surface*. In the options, I chose Active View, Tolerance 0.0100, and the complete Curve Range.

10 Once the curves are projected, there may be some additional repositioning and scaling to do to get the curves placed exactly right. The best way to do this is to translate and scale the original tattoo curves that were used for the projection. The curves on the surface will update because of the history connection they maintain with the original curves.

At this point, you could apply a brush stroke and animate it on the projected curves on surface. However, I prefer to duplicate these surface curves and rebuild them. Sometimes this allows more flexibility down the road. If you don't delete the history, the new duplicated curves will still update if you've animated the original NURBS surface.

HERE'S WHAT I DO NEXT:

11 I select the curves that have been projected onto the surface.

12 I choose *Edit Curves>Duplicate Surface Curves.*

13 When I hide the original NURBS Surface, I now see my tattoo curves floating in space **(Figure 6).** By looking through the shotCam, the curves are seen conforming to the topography of my neck.

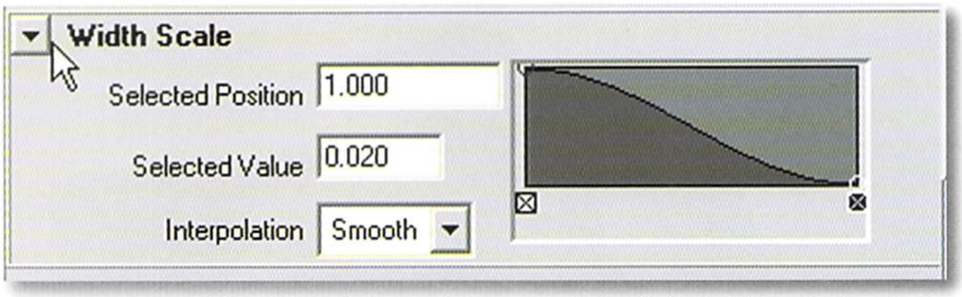
14 When I select these new Curves and take a look at the CVs, it is apparent that they have many more CVs than the curves I used to originally project onto the surface. In fact, in the Attribute Editor for the main spiral curve, the Min Max Value is from 0 to 305.421 and the spans are at 219. Leaving the curve like this will affect how the brush stroke will animate along the curve, so it's a good idea to rebuild the curve. Generally, the fewer spans you have, the faster the stroke will grow along the curve.

15 I select each of the curves and choose *Edit Curve>Rebuild Curve.* In the Options box, I choose Uniform for Rebuild Type and set the Parameter Range to 0 to #Spans. For the Number of Spans, I put 100, but this can be changed after applying the rebuild.

16 After applying this, I can open up the Attribute Editor for the

rebuildCurve1 node and use the slider to adjust how many spans will be in the rebuilt curve. The curve can be seen in the perspective window, updating as I move the slider or as I type in numbers. I found that, in this particular case, 150 is ideal for staying true to the original shape of the original tattoo curve.

Once I have my curves the way I like, it's time to actually get some Paint Effects strokes in the scene.



Width Scale

Selected Position 1.000

Selected Value 0.020

Interpolation Smooth

Figure 7– The width scale box can help custom design the look of the ends of each tube.

17 I apply the leafy twig brush to both my tattoo curves. I spend a long time tweaking the settings until I get what I want. Below is a description of the final stroke, and I've made a note of which settings are important to adjust and why I set them the way I did.

Under the stroke's shape node, the only setting I sometimes change is the Sample Density. Sometimes, increasing this will make the stroke look smoother, but it can affect the timing of the flow animation. Most of the rest of these settings I left alone in this particular animation. The leafyTwig1 tab contains most of the settings that will affect the look of the brush stroke. I arrived at these settings through experimentation. I've offered descriptions for a few that I felt were important. In addition, I've listed these in the order in which they appear going down the Attribute Editor, but that's not the order I use when creating a brush. I usually bounce back and forth between settings, adjusting here and there until I achieve the look. I wish I could recreate exactly how I did this, but that would be nearly impossible. After some experience and experimentation, you'll learn which settings you want to go to first and which you want to adjust enough for a finishing touch.

- Brush Type: Paint
- Global Scale: 20 – This looked right for this scene. Sometimes I set keys on this slider to achieve an alternative growth affect for my brushes. Actually, this is an animated scale rather than growth, but it can be effective sometimes, depending on what you are trying to do.
- Brush Width: .05 – This spreads out the tubes that make up this brush. Leaving it at 0 would make a thick line; moving it up causes the brush to be spread out along the length of the curve.
- Stamp Density: 12 – This can affect the resolution of the final brush stroke; a lower density will cause the brush stroke to look more like tiny dots.
- I set all of the shading attributes to black since I want the final composite to look like a tattoo. I'll be doing the color adjustments in After Effects.
- I turned on the check box for map opacity and set the texture type to fractal. This can break up the stroke a little.
- I turned Illumination off.
- I turned Shadows off.
- I turned Tubes on, but turned off tubes completion.
- Tubes Per Step is set at 1.0.
- Segments is at 6.
- Length Min is at 10 and Max is at 10

These are good settings to play with when designing a brush. I can only recommend that you fool with them. It's pretty obvious that they set a range for the length of the tubes that make up the brush.

- Tube Width1 is at .03; tube Width2 is at .01 – Width1 sets the width of the start of the tubes, and Width2 sets the width for the end (or tip). You can adjust them further by playing with the bias sliders.
- The width scale window can help to create a shape for the profile of the curve. The right side is the width at the tip of the tube. For this brush, I created a smooth curve that goes down to a point **(Figure 7).**
- I set the tube direction to flow along the normal.
- Under growth, I added flowers so that I could create tiny spikes growing like barbs from the tattoo. I could have done this with leaves or twigs, but after some experimentation, I found that flowers worked the best.
- Under the Flowers tab, I set petals in flower to 1.
- Num Flowers at 1.
- Petal Dropout at .661.
- Petal Length at .545.
- Petal Base Width at .091.
- Petal Tip Width at 0.
- Petal Width Scale ended up looking similar to the Tube Width Scale so that I could achieve that spiky look.
- Flower Angle1 and 2 were set at 90 so that the flowers would grow straight out, perpendicular to the tubes that make up the stroke. I could have had them spread out as they grew by setting these to different values. These are some good sliders to play with when you design a stroke.

That's it for the flowers; most of the other settings I left alone. The last group of settings I twiddled with is below:

- Under *Behavior>Displacement* I set the Displacement Delay to 0.
- I set the Noise Value to .12.
- Noise Frequency to .827.
- The noise settings give some turbulence to the way the tubes are drawn. Wiggle and Curl are also fun to play with, although in this case I left them at 0.
- Under Forces, I set Path Follow to 1.0 and Path Attract to 0.
- Length Flex is also set to 1 – this is actually one of the first settings I adjusted in order to get the leafy twigs to move along the curve as opposed to off a tangent.

Most of the other settings were left alone, with the exception of the Flow Animation settings. Using the techniques I described in the opening of this tutorial, I animated the stroke growing along the curve. Once I got some keys on the Graph Editor for the Time attribute, I was able to add some keyframes and tweak it so that the tattoo grew unevenly over time.

PART THREE: COMPOSITING

Once I was happy with the way the animation looked and moved, I turned the visibility of the image plane off and rendered out a sequence of 240 frames at video resolution. I made sure that the image sequence had an Alpha channel included. The Color channel of the images is going to look rather dark, as the image is of a black stroke against a black background. Looking at the Alpha channel reveals the shape of the stroke.

The rest of the animation was completed in After Effects with some very simple techniques:

1 I started a new composition in After Effects and imported the still image and the tattoo stroke sequence.

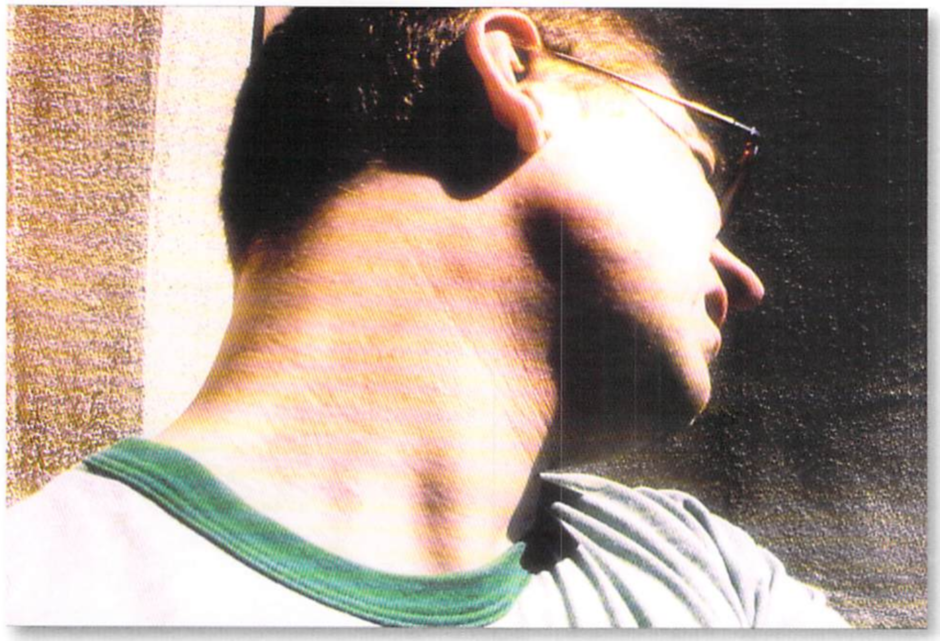


Figure 8– The image with a little processing in After Effects.

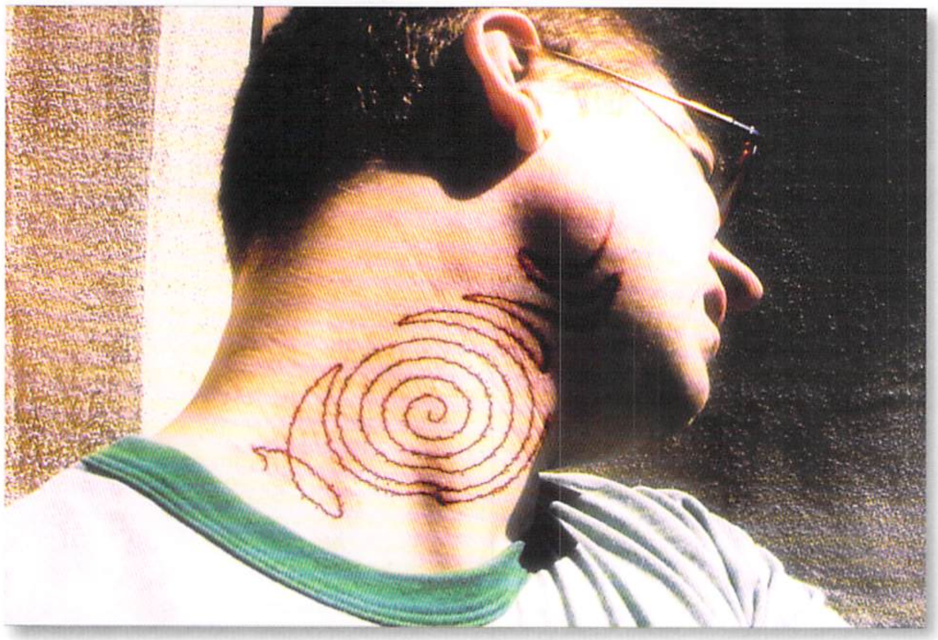


Figure 9– The final image with the tattoo burned into the skin.

2 My composition is 720 x 486 and 30 frames per second. The timeline is set to 240 frames.

3 I dragged the still photograph onto the timeline and did a little image processing to achieve a more dramatic look. I was going for something like the look they used in *Angel* for Cordelia's visions.

4 To do this, I applied the curves affect to the image and adjusted until there was a fair amount of contrast. Then I added the Channel Mixer effect. I tweaked the settings under Red-Red and Red-Green until the look was warm and kind of blown out.

5 I duplicated this layer and set the new layer (the one on top) to screen mode.

6 I added a fast blur effect to this top layer to help blow it out a little more and give it a dream-like quality.

7 I brought the opacity of the top layer down to 53 percent. At this point, I was happy with how the image looked. See **Figure 8**.

8 I dragged the tattoo image sequence onto the timeline. I wanted to get a burned-in look, so I set the mode to classic color burn.

9 I added a fast blur to this layer to fatten it up a little. The blur was set to 7.

10 I duplicated this layer with all of its effects just to fatten it some more.

11 I duplicated a third time and then pushed the frame sequence of this third version so that it started about 10 frames later than the original layers. I set its mode to color burn and brought is blurriness down to 3. I set the opacity of this layer to 35.

12 Finally I duplicated this layer one last time and set the mode of this fourth image sequence layer to multiply. I removed the fast blur effect from this layer. Opacity was still at 35. I also delayed it by moving the sequence so that it started at Frame 25.

13 The resulting effect of all this duplicating and adjusting was that the tattoo looked like it was being burned into the skin. You can see this in **Figure 9**.

The advantage of using paint effects for this type of effect is a little more apparent when things are moving around, but of course that adds a lot more complexity to the animation. You can think about adding bones or a deformer to the original NURBS neck geometry for some additional animated effects.

In closing, I hope this tutorial has given you some ideas on how you can create some interesting shots using Maya's Paint Effects. In addition, I hope that you've picked up some tips on how Paint Effects can be a little easier to work with when you employ some clever workarounds. There is a rendered version of the animation available for download on the *HDRI 3D* website located in the Resources section at **www.hdri3d.com/resources**.

Eric S. Keller *is a digital artist and animator living and working in West Hollywood, CA. He enjoys learning, studying, teaching, creating, and spending time with his wife and their two dogs. His background is in art, music, and science.*

DEATH, TAXES, AND *Type Write-Ons*

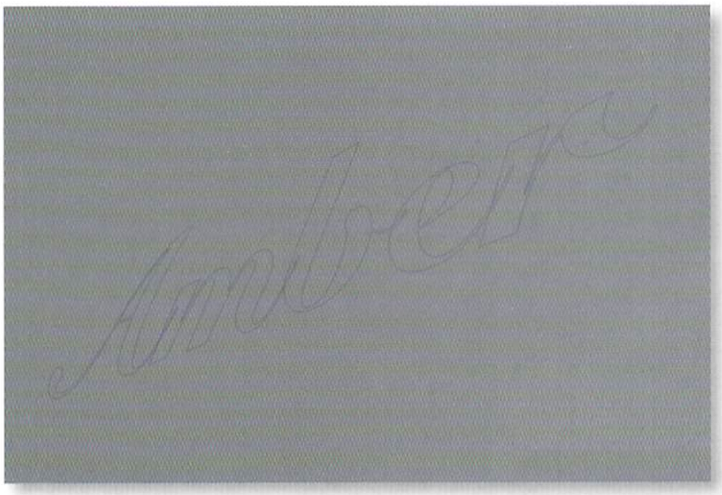


Figure 1– The outline file. Layed out inside Adobe Illustrator, then imported into LightWave Modeler.

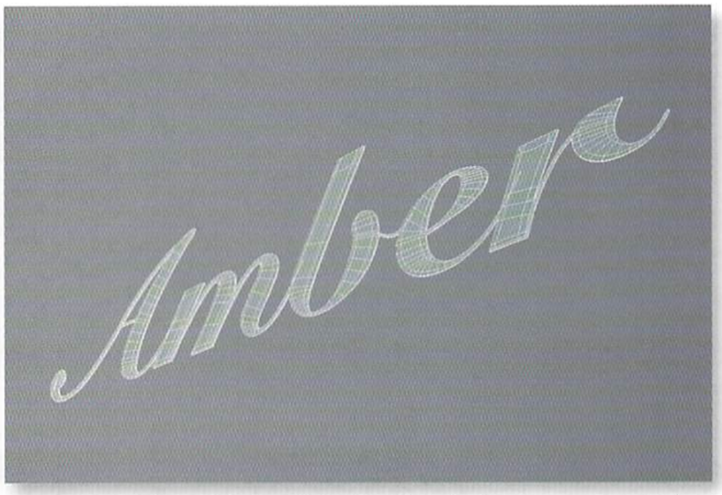


Figure 2– The modeled text in SubPatch mode. It's up to you how intricate your model gets. Just remember to spell whatever it is you're spelling correctly!

By Aaron Kent
3D Animator
Brooklyn, New York

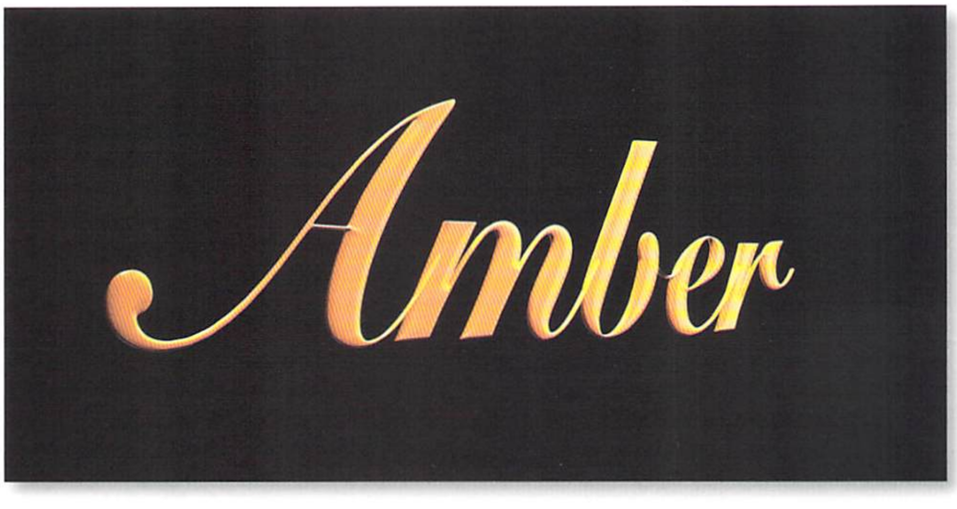
There are three things in life that are inevitable if you're a broadcast designer and animator: death, taxes, and write-ons. Actually, I'd venture a guess that it's easier to avoid death than write-ons. We'll create our cursive text in LightWave, then we'll create the write-on in After Effects using masks and the vector paint plug-in. 2D write-ons look great, but we'll go one step farther: we'll render out an Alpha pass to import into LightWave to be used as a clip map.

STEP 1 Modeling our text. I usually layout my text or logos in Adobe Illustrator and then import the outlines into LightWave to model. The LightWave Text tool is sufficient for most jobs, but when I want more fine-tuning, I find Illustrator a bit quicker to adjust text. I'm going to write on the name "Amber" for this tutorial. (Amber who? Amber Goddard, our beloved Managing Editor of HDRI *3D*, who else?)

Figure 1 shows the imported outlines in LightWave, ready for modeling. Figure 2 shows the modeled text. Since handwriting is organic in nature, I like to subpatch the text geometry. Subpatching the geometry adds tessellation at render time and smoothes the shape of the characters. A word of warning though: in LightWave, subpatching works correctly on objects made of three and four sided polygons. Any more or any less, and you get rendering errors. So I try to spend a bit of time on smoothing most 3D text if I can help it. Although it's quicker to just bevel text, this results in some fairly amateur looking graphics. If the geometry is going to play any prominent role in your graphics, you ought to spend a little time on creating it pristine from the beginning. So for this model, I took a little time to cap the faces of each character with polygons so I could subpatch them. Note: This can take a little time, so before you embark on the modeling and animating stages, make sure you're spelling the words right! Figure 3 shows a test render of the Amber text.

STEP 2 Now we'll go into After Effects to create the 2D write-on. We'll jump back into the LightWave Layout for a second and get a screen grab of the geometry. It's good to use the geometry as a reference instead of the Illustrator file that created it because small variations occur as you model and bevel

Figure 3– A render of the text.



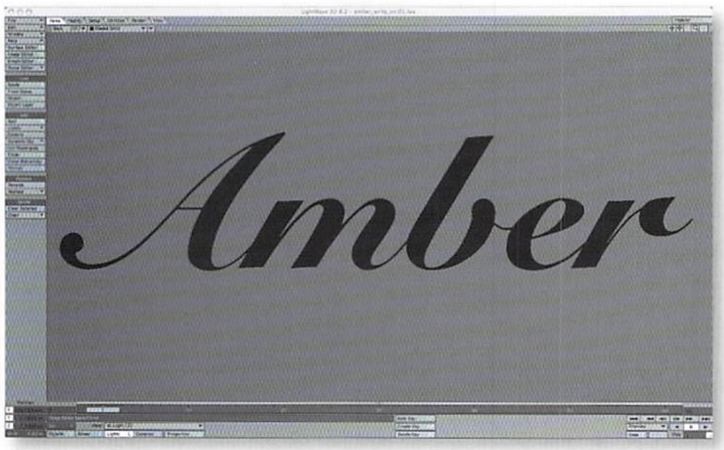


Figure 4– Grab a reference image from one of the orthographic viewports inside LightWave.

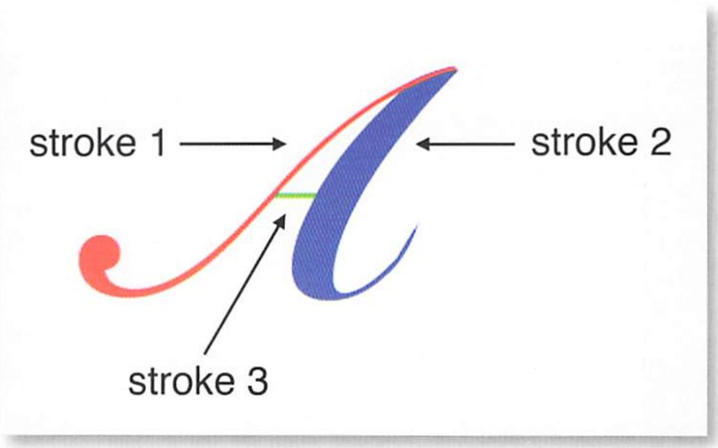


Figure 5– The Anatomy of a Character. Take a second to inspect whatever it is you're writing on and figure out how many strokes you'll need to reveal each character.

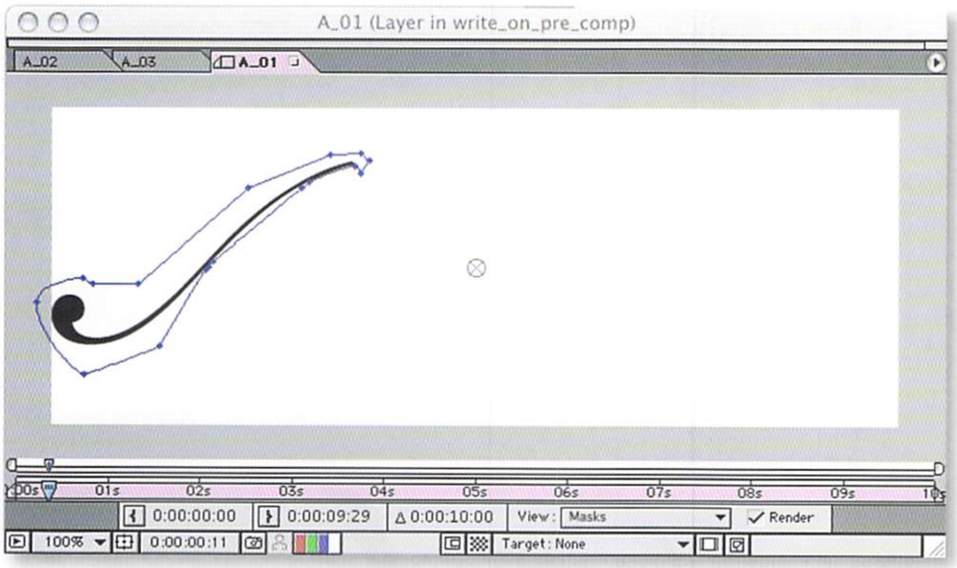


Figure 6– The first stroke in the character A. I'll duplicate the layer two more times and isolate each of the characters' other strokes.

the characters. Then we'll ensure that our clipping map will line up nicely when we apply it later.

In Layout, choose the back viewport and use whatever your screen grabbing software solution is to grab an image of the geometry. ***(Note: Pressing "Shift + Print Screen" copies the screen to the clipboard; paste the clipboard into an image and save with a program like Photoshop.)*** Some Orthographic rendering capabilities in LightWave would be nice in this situation. I think there might be a plug-in available. A quick search of www.flay.com turns up a number of useful plug-ins and/or lscripts for orthographic rendering. Isomtericity from http://projectoxygen.sourceforge.net/lw.html and Make Ortho Camera from http://jeremy.lwidof.net/lscript/ seem like they could do the job. Still, the ability to quickly render a high-resolution frame from any LightWave viewport would be a valuable addition to the software.

Figure 4 shows the reference image we'll use in After Effects. I've turned the text black so I can get a cleaner outline. Also, I've cropped the image so that it's exactly at the edges of the text. In Photoshop, I use the Replace Color Adjustment to switch the gray background to white (see figure 5). I could have also doctored the LightWave preference file to color the interface differently, but the Photoshop route is faster. Again, an orthographic rendering setting would be a great addition to LightWave.

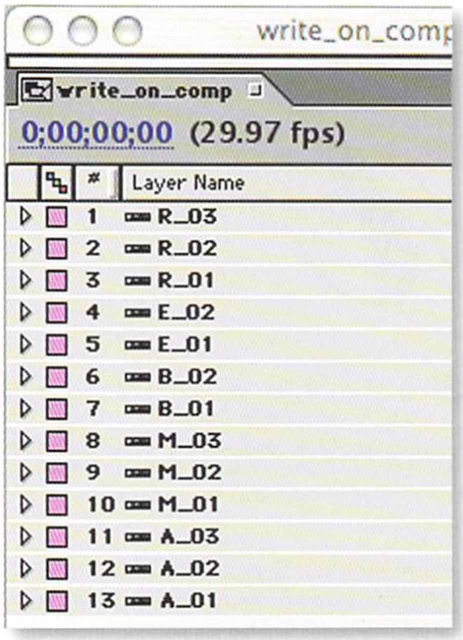
OK, in After Effects, we're going to break the text into a bunch of different pieces using masks. Then, we'll use the Vector Paint plug-in to reveal each piece. The benefit of using masks to break the sections of each character apart is that your animation conforms more to what a write-on actually mimics; a hand, writing on the text. Take a look at your hand while you spell your own name. In my case, I draw the "A" in three strokes. Take a look at figure 5. The first two strokes connect to create the inverted "V" Then, I pick the pencil up off the page and create the third, horizontal stroke. If you're animating a mask to reveal the character, this can be painful to animate (if not impossible and time consuming). But, if I mask the "A" into logical sections, as in figure 6, and then reveal each with the Paint tool, it not only becomes faster, but looks more natural. Also, by staggering the layers, it becomes a lot more intuitive when it's time to animate the speed and dynamic of the characters appearing.

Suppose your client says, "I love it, make it faster." With masks dealing with a lot of keyframes, it might not re-time like you expect. With the Vector Paint plug-in, all you have to do is increase the playback speed; and that only involves dragging one slider. You get the idea. Maybe the first two strokes of the "A" happen almost imperceptibly slower then the third stroke. The third stroke happens a little faster.

It's these subtle differences that really make your animation stand out. Go ahead; write the capital letter "A" a couple times on a piece of paper. Just try to write it at a steady pace and you'll see it just doesn't naturally occur that way. That's not to say that you couldn't write everything out at a computer perfect pace to create a pretty cool "techie" write-on. The sky's the limit. The mask and Vector Paint technique just opens up a lot of animation options while speeding the process up, no matter what the write-on style.

OK. That said, I'll start masking the characters.

Create a composition with a white background that is the size of the text file we just created and start masking off different sections of each character. Use your intuition, or, just write the word on a

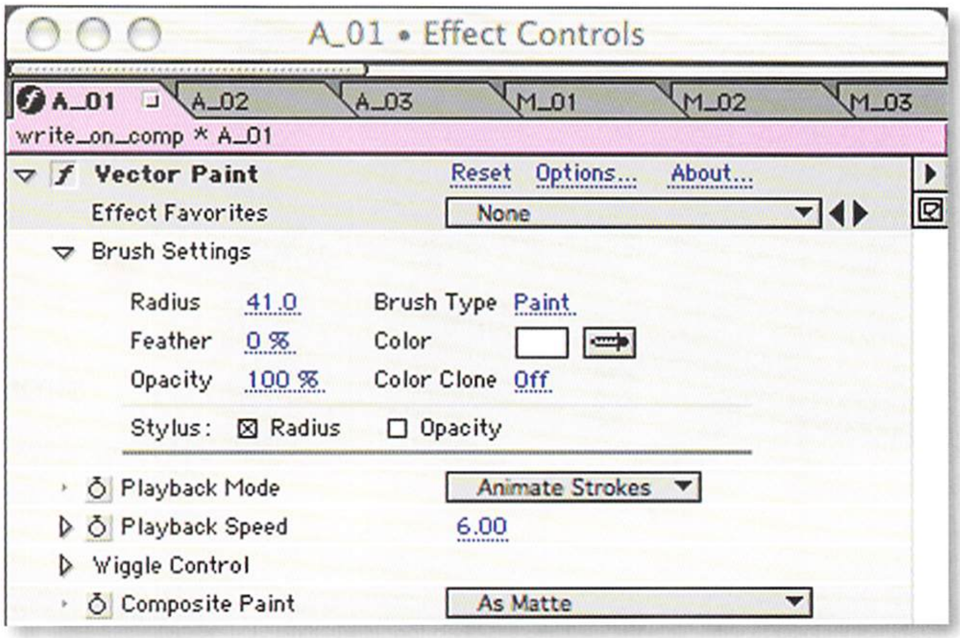

write_on_comp

write_on_comp

0;00;00;00 (29.97 fps)

#	Layer Name
1	R_03
2	R_02
3	R_01
4	E_02
5	E_01
6	B_02
7	B_01
8	M_03
9	M_02
10	M_01
11	A_03
12	A_02
13	A_01

Figure 7– Once I'm done isolating each stroke I have 13 layers to reveal using the Vector Paint plug-in.


A_01 • Effect Controls

A_01 | A_02 | A_03 | M_01 | M_02 | M_03

write_on_comp * A_01

Vector Paint — Reset — Options... — About...

Effect Favorites: None

Brush Settings

Radius: 41.0 — Brush Type: Paint

Feather: 0 % — Color

Opacity: 100 % — Color Clone: Off

Stylus: ☒ Radius ☐ Opacity

Playback Mode: Animate Strokes

Playback Speed: 6.00

Wiggle Control

Composite Paint: As Matte

Figure 8– The Vector Paint Effects window. This plug-in might as well be called the Swiss Army knife of broadcast design.

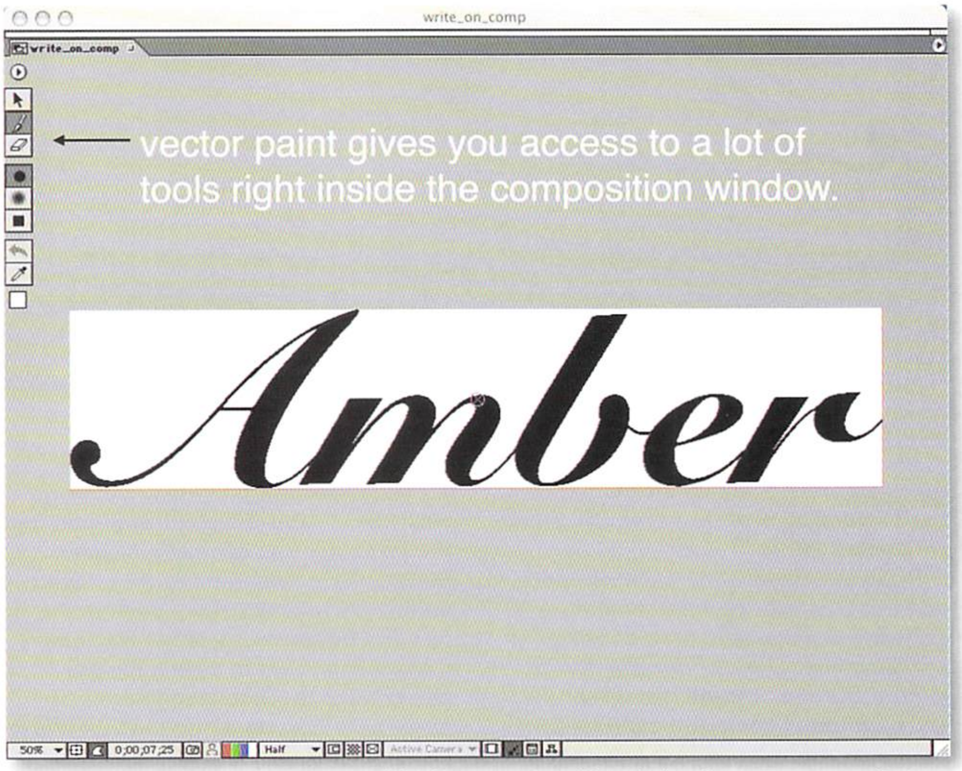

write_on_comp
vector paint gives you access to a lot of tools right inside the composition window.
Amber
50%
0;00;07;25
Half
Active Camera

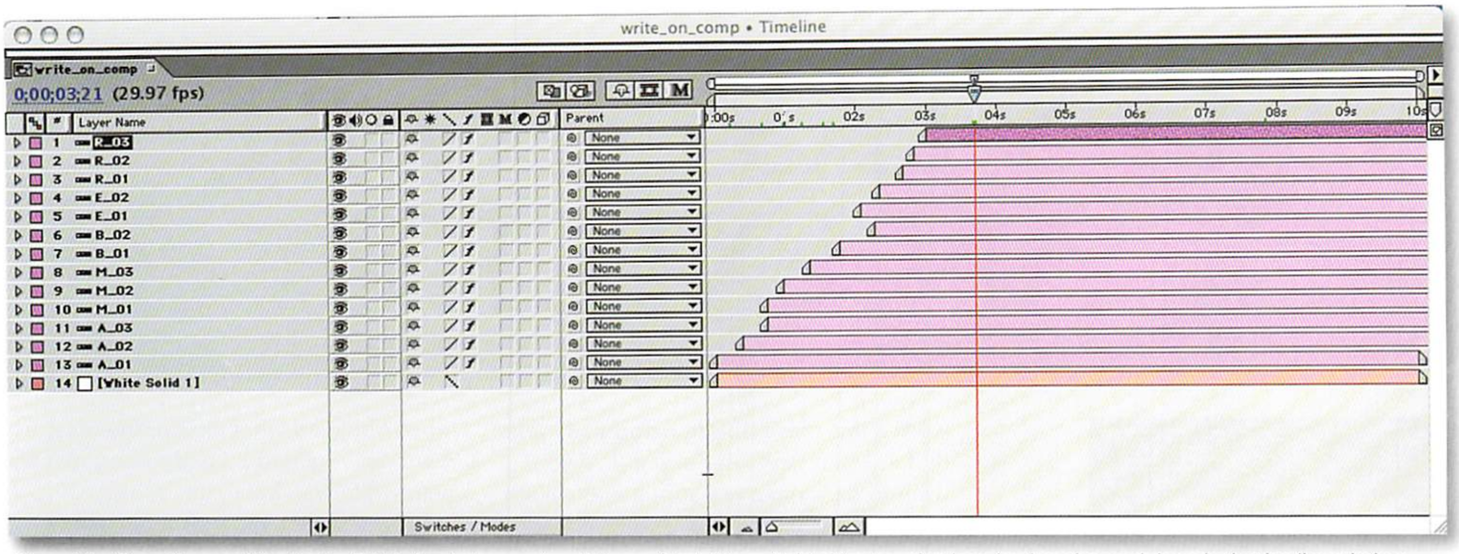
Figure 9– Vector Paint also opens a tool bar in your composition window for easy access to features while you're working with the plug-in.

piece of paper and see what your hand does naturally. If you look at figure 7, I ended up with thirteen layers to spell out the word Amber.

The next step is to use the Vector Paint plug-in to reveal each of the layers. Select layer A_01, or whatever you choose to spell. For me, it's the first leg of the "A." Apply the plug-in *(Effects>Paint>Vector Paint)* you'll see the plug-in window come up. First off, let's set your brush size. Figure 8 shows the plug-in window. I've set the Brush Radius size to 41. Adjust the brush depending on the size of the text you're revealing. You can see the radius of your pointer change in the composition window as you do so. I've found it's good to be slightly wider than the object or text you're revealing, but not too much bigger. You should have to concentrate when you draw your paint stroke over the layer. It makes the reveal look more realistic. Too big a paintbrush and it ends up looking like a clumsy wipe more often than not.

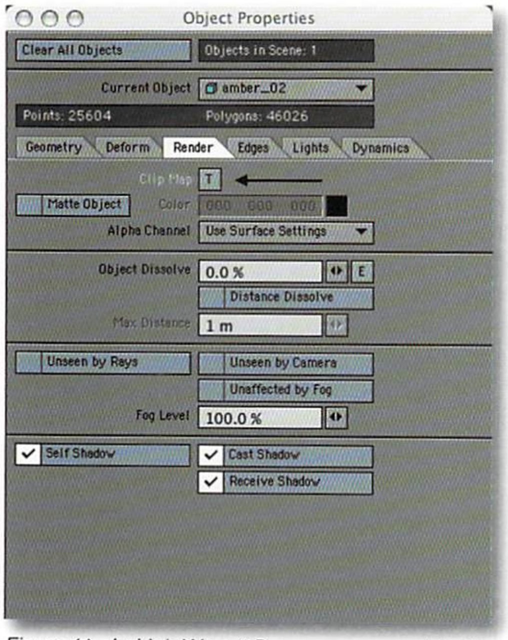
Next, I set the playback mode to animate strokes in the effects window. This way, After Effects will record the motion of the brushed stroke once we start painting. The trick is to get each section with one, smooth stroke. If you mess up and miss an edge, click the little arrow bullet in the top left corner of the composition window next to the Vector Paint palette, select the bad stroke, and hit the delete key. You can see the Vector Paint Composition tools in figure 9.

OK, go ahead and paint the stroke over the first leg of the "A." Once I'm happy with my stroke, I jump back to the effects window and switch the Composite Paint option to "As Matte" and then set my Playback Speed to about 5 or 6. Ram preview your comp now and you'll see the first leg of the "A" writing on nicely. I'll continue and apply the Vector Paint plug-in to the rest of the layers. Once I've revealed each layer, we'll stagger the layers in the timeline window so that each stroke follows the previous one at a comfortable pace. Look at figure 10 to see the layers once they're staggered in the comp window. Remember, you can always take artistic liberty. Chances are, the amount of time you have to write text on in broadcast graphics is usually a lot less than it would occur physically. You might have to reveal the strokes fast and overlap the reveal of the characters. You might have to reveal each character at the same time. Again, it's whatever works and looks best per the situation.



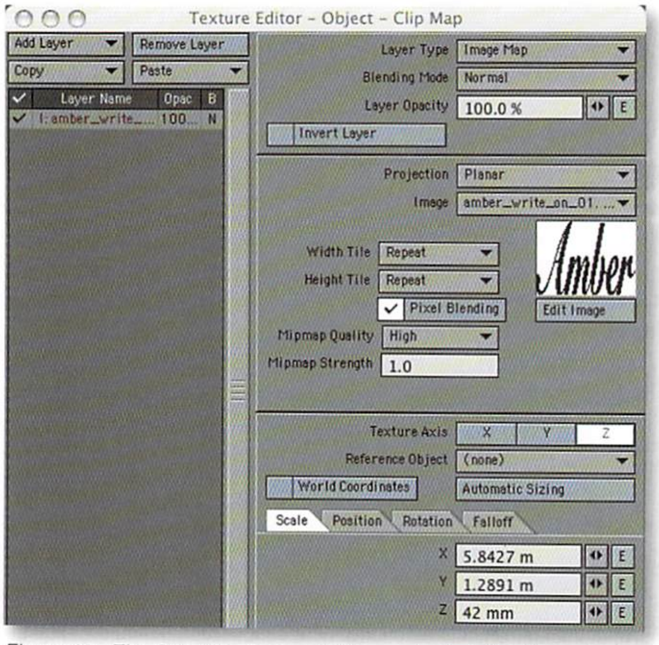
#	Layer Name	Parent
1	R_03	None
2	R_02	None
3	R_01	None
4	E_02	None
5	E_01	None
6	B_02	None
7	B_01	None
8	M_03	None
9	M_02	None
10	M_01	None
11	A_03	None
12	A_02	None
13	A_01	None
14	[White Solid 1]	None

Figure 10– Once I've used the Vector Paint plug-in to reveal each layer of text, it's really intuitive to order them by dragging each layer in the timeline window.



Objects in Scene: 1
Current Object: amber_02
Points: 25604
Polygons: 46026
Clip Map: T
Matte Object
Color: 000 000 000
Alpha Channel: Use Surface Settings
Object Dissolve: 0.0 %
Distance Dissolve
Max Distance: 1 m
Unseen by Rays
Unseen by Camera
Unaffected by Fog
Fog Level: 100.0 %
✓ Self Shadow
✓ Cast Shadow
✓ Receive Shadow

Figure 11– In LightWave 3D, we access an object's clipping feature on the Render tab of the Object Properties panel. See the arrow above if yer confused.



Layer Name	Opac	B
✓ 1: amber_write...	100...	N

Layer Type: Image Map
Blending Mode: Normal
Layer Opacity: 100.0 %
Invert Layer
Projection: Planar
Image: amber_write_on_01. ...
Width Tile: Repeat
Height Tile: Repeat
✓ Pixel Blending
Mipmap Quality: High
Mipmap Strength: 1.0
Texture Axis: X Y Z
Reference Object: (none)
World Coordinates
Automatic Sizing
X: 5.8427 m
Y: 1.2891 m
Z: 42 mm

Figure 12– The Clip Map Texture Editor works just like the surfacing interface in LightWave. We'll leave our Projection in Planar mode and on the Z-axis. Hitting the Automatic Sizing button should line up our clip map nicely.

After completing our 2D write-on in After Effects, we'll render it out as a QuickTime movie and use it as a clip map in LightWave.

STEP 3 In LightWave, let's load our text object. With the object selected, hit the "p" key to open the Object Properties panel. On the Properties panel, click on the Render tab, and then click again on the clip map button towards the top. Figure 11 shows the render tab on the Properties panel. Clicking on the Clip Map button opens the Texture Editor window. We'll apply the movie we rendered here. We'll set the Projection Axis to Z and hit the Automatic Sizing button as seen in Figure 12. The map should line up nicely. (That's why we cropped the reference image right to the edges of the text.)

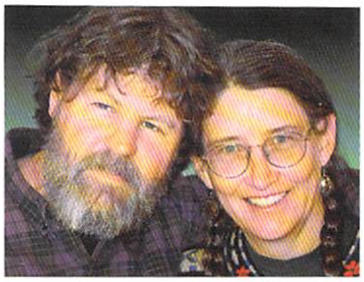
Note: The fit isn't perfect. The object has a little problem right on its X-axis seam. I've found the fast fix for this problem is to also render a version of my text without the clip mate and then "chase" the Write-on pass with the Solid Text pass in After Effects employing a really simple wipe. If you wanted, you could also apply the clip map using a UV texture map and then edit the map to clean up the edges. For cursive text, you sometimes end up with some pretty thin sections to edit in the UV map, so it takes a little time, but I'd only be that big a stickler about it if the text was rotating radically enough to make the problem apparent.

We'll that about sums it up. Hopefully, you'll never be afraid of a 2D write-on again, or a 3D one for that matter. Now if you get off yer butt and pay your taxes you'd have most of the bases covered.

Aaron Kent *lives in Brooklyn. During the day he works for a broadcast design company. You can see their work at www.vizualsolutions.com. At night he procrastinates, but still plans to make short 3D films. If he ever gets around to finishing it, you could see a bunch of random things on his Web site: www.shiftlessdreams.com (but don't count on it.).*

FINAL RENDER:

IT'S EASY TO BE AN EXPERT



By Brad Carvey
Electrical Engineer, 3D Animator
New Mexico

People tend to expect a lot from "Experts." The problem with most people's thinking is that they hold experts to very high standards. Mr. Belvedere was a film character in the early 50s. He was an expert on everything. You could ask him any question on any topic and he would know everything. I think that people tend to expect experts to be geniuses or people that are extremely dedicated to learning all they can about a particular subject. It's possible for anyone to become an expert on something. The key to becoming an expert is to narrow your focus. With a complex program like LightWave, after learning the basics, you can concentrate on something very specific and become an expert on that subject.

Learning something as complex as LightWave can be overwhelming for the beginner. If the novice concentrates on learning just a little bit at a time, it won't be long before they can create something interesting. I think of learning a foreign language as similar to learning how to use a computer language or program. It's not very difficult to learn a single word in a foreign language. For example, it you needed to learn the word for water in Spanish, it would not take long to find the correct word and to memorize it. After memorizing the word agua (water), you would need to use the word a few times before being confident that it would work. For example, if you went to a Mexican restaurant and asked for agua and they brought you a glass of water, you would feel confident after successfully getting water a few times. The same thing happens if you are learning Java or Photoshop. You try something, and if you repeatedly get the same result, you start to develop some confidence that you know what you are doing.

To learn a language, you will need to learn and use lots of words. Eventually, you will learn enough words to form sentences. The sentences may not be perfect, but you will be able to communicate. As you learn more "words" in LightWave, you will be able to create models and animate them. Even with only a few words, some people will be able to creatively use their limited vocabulary to convey interesting ideas. The same concepts apply to LightWave. Some people can learn only a few things in LightWave and still be able to create interesting animations, while others concentrate on learning lots of commands but don't use their extensive vocabulary to create cool stuff.

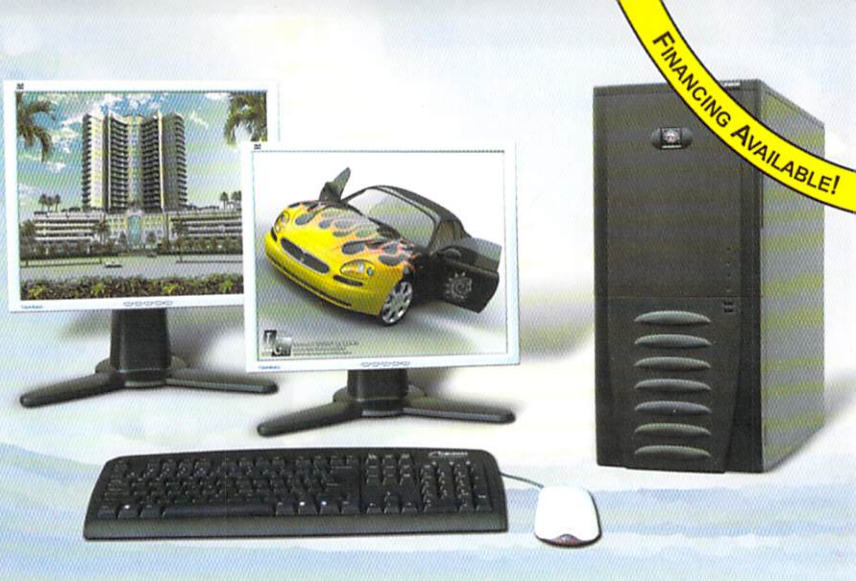
To become an expert on everything, or even a small subset of everything, is impossible. Becoming an expert on LightWave is very difficult. Most LightWave experts do not know every command and plug-in, but they do know how to find what they need to accomplish whatever they need to do. I typically don't have time to learn something new in LightWave unless I need to know it to complete a job. I became an expert on character lip-syncing while doing a Virtual Kris Kristofferson for a Tom Clancy mini-series called *Netforce*. I had played around with lip-syncing and Magpie, a lip-syncing program, for a long time, but when I applied what I knew, the results were not very good. I had to very quickly learn what was not in the books.

I suggest that you pick a command and become an expert on that command. Digital Confusion is an interesting plug-in that it would be worthwhile becoming an expert on. Before trying to use Digital Confusion or any other command in any other program, do some research. Collect any information that you can find on your new subject. I prefer to print out any information I find and put it in a folder. I like to take notes as I am learning, and those notes go in the same folder. Try adjusting all of the settings and take notes on how the rendered image changes. Experiment with using values that may generate unusual results, like manually entering negative numbers. Print out some of the images and write notes on them. Save all images and scene files to a CDROM and put that in the folder as well. Then, the next time you or someone you know needs to use Digital Confusion, you can say with confidence that you are a Digital Confusion Expert.

__Brad and Andrea Carvey__ have been doing computer animations for a long time. In 1969, Brad used an analog computer, which was the size of a car, to produce his first computer animation. Andrea, an archeologist, prefers to do scientific animations; her credits include programs like Discovery Channel's __Understanding Cars.__ Brad is an electrical engineer and an Emmy award-winning member of the Video Toaster development team. He prefers to do feature film work. His credits include films like __Men in Black__, __Stuart Little__, __Black Hawk Down__, __Kate & Leopold__, and __Master of Disguise__.

TSUNAMI

EDIT. ANIMATE. FAST.



TSUNAMI Animation Station

Safe Harbor's TSUNAMI Animation Station is a custom-built workstation designed around YOUR needs as an animator. Choose from a long list of components to maximize your time and creativity. Available in Pentium™ 4 and Single or Dual Xeon® configurations with blazing fast graphics boards from PNY and plenty of storage options for those extra large projects. Add SoftImage|XSI, LightWave 3D, or Cinema 4D animation software, and you'll have a professional-level powerhouse which is sure to "wow" your clients! Price yours today at www.sharbor.com/build-yours.

LightWave 3D [8] v8.5

This Emmy award-winning software now has many new updates, including a new OpenGL hardware shader, easier integration of 3D objects from other applications, improved particle dynamics, and over 100 new configurable keyboard shortcuts ..... CALL

LWCAD 1.5

New version adds five new tools to enhance your LightWave modeling experience, and existing tools have three new snapping modes, new ortho modes, adjustable dimension helpers, and more. Now includes Cap Hole Studio 2. Available for both Mac and PC 199.99

Dosch 3D: Baseball

Play ball, 3D style! Fifty-six fully textured models depicting America's favorite pastime. Includes everything baseball such as bats, gloves, a stadium, equipment, scoreboards, and more. In multiple file formats for little league through major league needs! 112.00

SoftImage|XSI Foundation

Get everything you need to take your work to the next level of realism: Two-CPU mental ray® v.3.3 rendering, advanced subdivision surface modeling, full character rigs, Cloth & Particles, and more! Training DVDs included 585.99

Competitive Upgrade .409.99

Animation:Master 2005

This powerful animation software includes features such as Spline/Patch modeling, keyframe and timeline animation, IK, lip-sync motion capabilities, texture creation, multiplane compositing, audio syncing, unlimited resolution rendering, and so much more! 249.99

ZBrush 2 with Tutorial

A blend of innovative 2D and 3D tools enables you to paint, sculpt and texture, all in real-time. You don't have to be an engineer to realize the power of ZBrush; instead, you'll find that creating stunning art is quick, effortless, and FUN! Includes Kurv Studios tutorial ... 449.99

ButterflyNetRender 3.0

Manage your network rendering with ease and sophistication! Now supports LightWave 3D [8], 3DSMax 6+, and Maya 6. New features include: HTML Viewer/Internet support, 'FrameCycler' support, post commands, scene linking, AVI building, Maya 6 MR support and more 579.99

3D Fluff Tutorial Bundle

Designed for the intermediate to advanced Cinema 4D animator. Learn about modeling, using Photoshop, Sketch & Toon, and more! This bundle includes Design and Visualization, Radiosity Interiors, and Non-Organic Modeling 225.00

Each Volume 75.99

Terms: POs accepted from schools and government agencies • All checks require 5-7 days to clear • Defective products replaced promptly • RMA number required for all returns • Returns accepted within 15 days, in original packaging, postage prepaid, undamaged • Opened software is not returnable • Shipping charges are not refundable • Returns are subject to a 15% restocking fee • Not responsible for typos • Prices are subject to change

Screenshot Credits:
Grandview Plaza West - Miami, FL: © Paradigm Productions LLC. Charles T. Gaushell and Jason Lee, artists

Maserati: © Elvis Blazencic

Value Added Reseller
Professional Video

Information
262-548-8120

Tech Support/RMAs
262-548-8157
Mon-Fri 8:30am-5:00pm CST

Fax: 262-548-8130

Safe Harbor Computers
W226 N900 Eastmound Dr.
Waukesha, WI 53186

www.sharbor.com

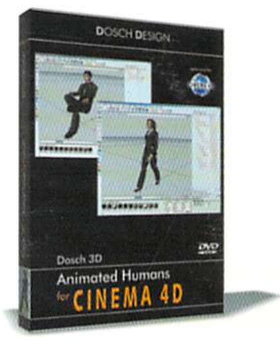
YOUR SOURCE FOR ALL OF YOUR ANIMATION NEEDS!

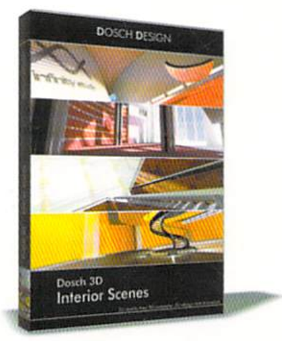
www.sharbor.com

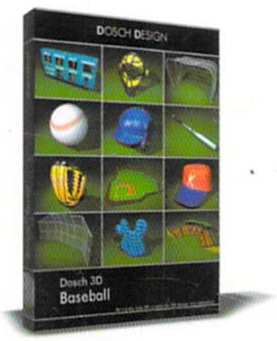
800-544-6599

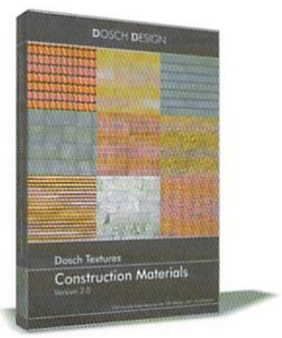
SAFE HARBOR COMPUTERS

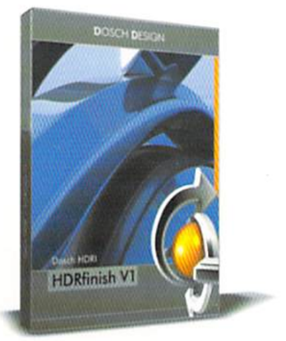
DOSCH DESIGN


Dosch 3D: Animated Humans for Cinema4D

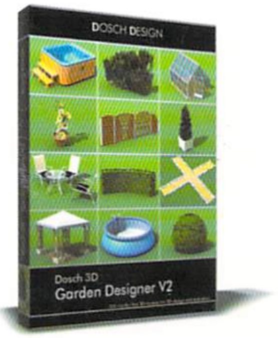

Dosch 3D: Interior Scenes


Dosch 3D: Baseball


Dosch Textures: Construction Materials V2

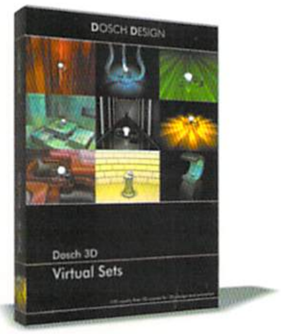

Dosch HDRI: HDRfinish V1

Dosch HDRI: Skies

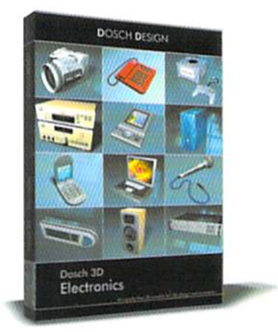

Dosch 3D: Garden Designer V2

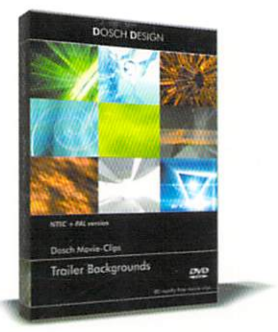
Dosch HDRI: Radiant Skies

Dosch 3D: Cars 2005

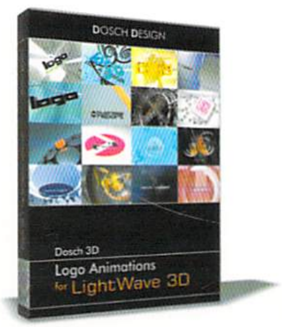

Dosch 3D: Virtual Sets

***Innovative* and *easy to use* computer graphics products:**

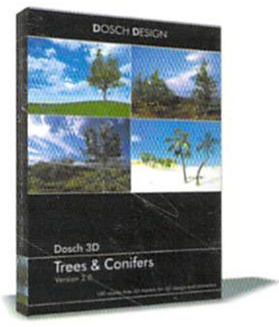

Dosch 3D: Electronics


Dosch Movie-Clips: Trailer Backgrounds

Dosch 3D: Surrounding Skies V2


Dosch 3D: Logo Animations for LightWave3D

Dosch 3D: Job Poses


Dosch 3D: Trees & Conifers V2

Dosch 3D: Shop Design

3D-Design, Animation, Visualization

Besides complete 3D-models and scenes ▸**Dosch 3D**, Dosch Design products contain surface materials ▸**Dosch Textures**, High Dynamic Range Images ▸**Dosch HDRI**, as well as 2-dimensional objects for architectural visualizations ▸**Dosch VizImages**.

Animated movie sequences ▸**Dosch Movie-Clips**, plus quality music and sound effects ▸**Dosch Audio** complement this product segment.

Graphic Design, Desktop-Publishing, Webdesign

▸**Dosch LayerFX** products unlock a new realm of creative freedom - and therefore substantial time and cost savings.

www.doschdesign.com

DOSCH DESIGN US-Shipping Center
603 Ruskin Drive - Altoona, PA 16602 - USA
Phone/Fax: ++1 (814) 943-2807
1 (866) 3D-DOSCH (US/Canada Toll-free)

DOSCH DESIGN Europe
Gotthard-Schuell-Str. 14 - 97828 Marktheidenfeld - Germany
Phone: +49(0)9391-915853 - Fax +49(0)9391-915854
info@doschdesign.com