# 6

# Texturing

**TEXTURING TRANSFORMS THE** dull-gray denizens of the modeling world into lifelike objects. Although texturing is nothing without good lighting, good lighting is nothing without good texturing. In fact, good texturing can disguise limitations within the model itself. This is most true with displacement mapping. In any case, Maya does not limit itself to the texturing of NURBS surfaces and polygons. You can texture Particles and Paint Effects brushes with a wide range of techniques. Whichever element you choose to texture in Maya, certain basic steps will help you achieve great results. These steps include the creation of a diverse, high-resolution texture library and the ability to slap generic bitmaps together in the Hypershade.

**This chapter's topics are organized into the following techniques:**

- Building a Texture Library
- Fractals and Noise in Math, Nature, and Maya
- Combining Generic Bitmaps in Maya
- Coloring Particles Individually
- Texturing Paint Effects Tubes
- Industry Tip: Rendering ZBrush Displacement Maps in Maya

## Building a Texture Library

An important task for successful texturing is the creation of a texture library. This holds true for a professional as well as an independent or student animator. A thoroughly stocked texture library can save you a tremendous amount of time and supply a great deal of diversity for any texturing project.

A texture library is generally stocked with three types of bitmaps: photos, scans, and hand-painted maps. With current technology, high-resolution digital cameras and flatbed scanners are relatively cheap and should be a part of any animator's arsenal. Hand-painted bitmaps, due to their labor intensiveness, represent the pinnacle of texturing. Feature, visual effects, and game studios regularly employ full-time texture painters. Regardless, the best hand-painted textures usually start with photos and scans.

As an animator, you can either create your own bitmaps, purchase them, or download them for free. You can buy texture-library CDs, such as Total Textures (`www.3dtotal.com`), that are stocked with hundreds of images at a reasonable price. Due to the low resolutions and a lack of useful diversity, however, such library bitmaps are generally inferior to those you might make yourself. Online texture archives, such those available at 3D Cafe (`www.3dcafe.com`), suffer from the same issues, although you can often find a limited number of high-quality maps for free. If you do choose to use a premade library or download bitmaps, be aware of any copyright issues. In today's litigious society, it's best to play it safe.

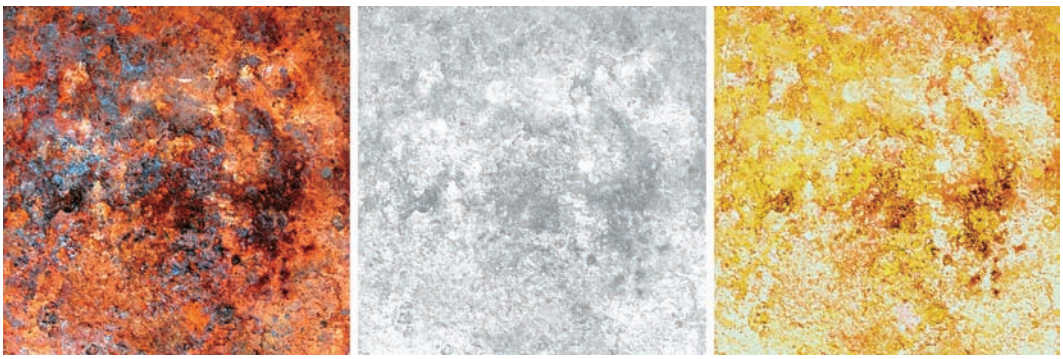
If you choose to photograph your own textures, it's best to avoid strong lighting situations. That is, you want to avoid heavy shadows or intense light colors that betray the time of day. In other words, a bright, noonday sun is not necessarily the best light source for photographing a chunk of rusty metal or brick wall. Rather, a bright but overcast day, where the light and shadows are diffuse, gives you the best results. Ideally, you want to use the textures in your library for a wide range of 3D lighting situations. Thus, their integral lighting should be neutral. Along those lines, avoid wide-angle lenses, such as 24mm or 28mm; they will insert a great deal of perspective distortion into the photo and make the resulting texture difficult to apply in Maya

In terms of texture resolutions, bigger is better. Although it's easy enough to scale down an image, a scaled-up image is fraught with quality-destroying artifacts. If possible, keep all your bitmaps somewhere between 512 and 2048 (2K) lines of resolution. The resolutions should be large enough to stand up to a full-screen render. If the texture is applied to a model that fills the screen, the texture quality should not break down. Thus, if you're rendering for video, a 1024 texture resolution, which is twice the resolution of the 480 lines dedicated to standard television, is a safe bet. In comparison, textures created for 35mm feature film are sometimes as large as 4096 (4K) lines of resolution. That said, if a texture is destined for a small item that will never get close to the screen, such a button, pencil, or sticker, you can reduce the resolution to twice the number of vertical lines the texture might take up. For example, the button on a character's shirt may never take up more the 10 vertical lines of the render, thus a 20×20 pixel size will suffice.

## Choosing Texture Bitmaps

Some texture bitmaps are more useful than others. In fact, once you create a texture library, you may discover your own favorites that have a wide range of applications. The following are suggestions for stocking a library:

**Rusty metal** Rusty metal bitmaps are extremely flexible. Although their origin is obviously metallic, they are excellent for creating any material that is noisy, depredated, or dilapidated. Even if the existing colors are not perfect, rusty metal can make aesthetic bump maps. If the bitmap color is tweaked in a digital paint program, the flexibility becomes even greater.



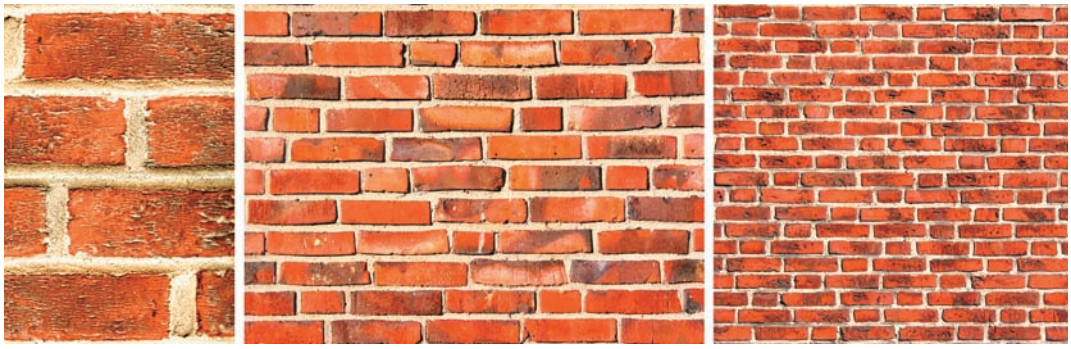
(Left) Rusty metal bitmap. (Middle) Bitmap converted to grayscale, which is ideal for bump mapping. (Right) Bitmap color adjusted in Photoshop to create a brand-new material.

**Ground** Various ground textures are important to any modeled set piece. This includes dirt, concrete, asphalt, and tile.

(Left to right) Asphalt, parched earth, and dirt bitmaps

**Wall** Wall textures go hand in hand with ground textures. Photos of plaster and painted drywall are needed for interiors. Stucco, metal siding, cinderblock, and bricks are needed for exteriors.

The use of bricks brings up the issue of tiling. Many 3D textures made available through texture library CDs and online archives feature a small section of brick that is intended to be repeated over and over. In reality, photos of brick walls at different scales provide the greatest flexibility. For example, if you need to texture the exterior of a 3D building, applying a bitmap that features a 10×10-foot section of brick wall is far more convincing than a 1×1-foot section that has been tiled repeatedly. If the model features a smaller section of brick, a bitmap that features a 5×5- or 1×1-foot section may be fine.



Three brick bitmaps with different scales

Brick is not the only benefactor of multiple scales. Any texture, whether it be metal, ground, wall, fabric, or wood, is best represented in the library by the equivalent of close-up, medium, and wide shots. That said, the two wall textures that are best suited for tiling are wallpaper and tile. Assuming that the wallpaper or tiles are in good condition, only the key pattern need be captured by the bitmap.

Wallpaper and ceramic tile bitmaps ready to tile in 3D

**Fabric** Although you can apply the Cloth texture in Maya in a convincing manner, it cannot duplicate specific woven or printed patterns. In addition, the Cloth texture does not address any kind of fabric wear and tear. Scans of various cloths, such a denim, silk, and lace, prove

to be handy. Scans of patterned or embroidered clothing are equally useful. For example, when replicating a pair of jeans, scans of various jean parts can be combined into a texture bitmap. Thus, detail such as stitching, buttons, and frayed holes are captured easily.

A texture constructed from multiple scans of a pair of blue jeans

**Wood** Wood is difficult to replicate with procedural textures alone. Photos of actual wood provide the best solution. Photos featuring uncut wood panels, such a plywood or veneer, give you the most flexibility when positioning the texture on model furniture or other man-made objects. Photos featuring interlocking wood slats are ideal for floors or the walls of rustic buildings. Close-ups of various tree barks are also useful for re-creating outdoor foliage.

(Left to right) Particle board, wood siding, and tree bark bitmaps

**Grayscale** Grayscale textures are ideally suited for many Maya attributes, such as **Diffuse**, **Transparency**, and **Bump Mapping**. Maintaining a unique set of grayscale bitmaps gives you more flexibility when texturing. For example, a unique map that represents smudges or condensation is useful for anything made out of glass. On the other hand, it is beneficial to keep grayscale variations of color bitmaps. For example, you can adjust a grayscale version of a skin texture and map it to the **Specular Roll Off** attribute.

(Left) A color bitmap for a character's face. (Right) The same bitmap converted to grayscale and adapted for the **Specular Roll Off** attribute. *Original bitmap painted by Andrey Kravchenko.*

Although you can apply color bitmaps to single-channel attributes, Maya must discreetly convert their color values to scalar values. In this situation, contrast is usually lost and the resulting texture is not as ideal.

**Project photos and scans** Specific projects usually require specific textures. As such, photos and scans must be taken. This texture category would include printed materials (magazine covers, book pages, written notes, labels, posters, and so on) and architectural details (wall plates, molding, warning signs, and so on). Once a project has ended, it pays to keep the project-specific textures in the texture library. Although it's difficult to predict what the

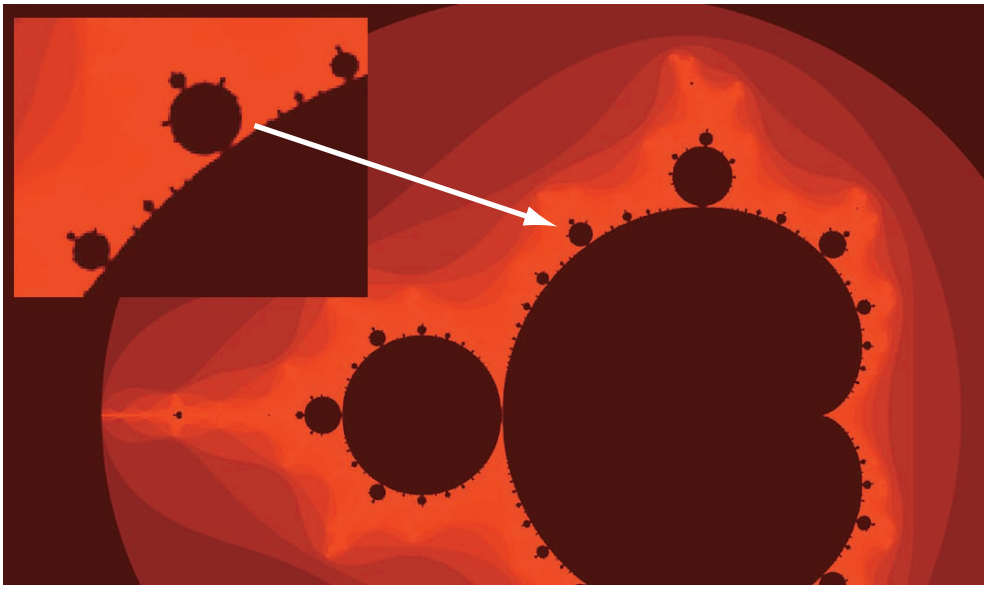
texture requirements of future projects will be, a large texture library can ultimately save texturing time. Even if a particular texture proves to be unusable as is, you may be able to quickly adapt it in a paint program.

### Exporting Maya Shaders

A texture library need not be limited to bitmaps. You can export Maya shading networks at any point. Exporting the shading network allows you to maintain all the attribute settings of the material plus maintain connections to various maps and utilities. To export a shading network, highlight the material icon in the Hypershade window and choose **File → Export Selected Network**. Exported shading networks are saved, by themselves, in the `.mb` or `.ma` format. To import a saved shading network, choose **File → Import**.

## Fractals and Noise in Math, Nature, and Maya

Mathematically, fractals are geometric objects that feature self-similar patterns at every scale. That is, when a fractal is viewed graphically, the patterns seen at every level of magnification are similar (although not strictly identical). Although fractals appear chaotic at a distance, they are each based on a simple, reiterative mathematical equation. The equation is solved over and over with each output serving as an input for the next step. The most famous fractal, the Mandlebrot set, is based on the equation $z(n + 1) = zn^2 + c$.



A Mandlebrot set. The circular "buds" appear on all edges at all levels of magnification.

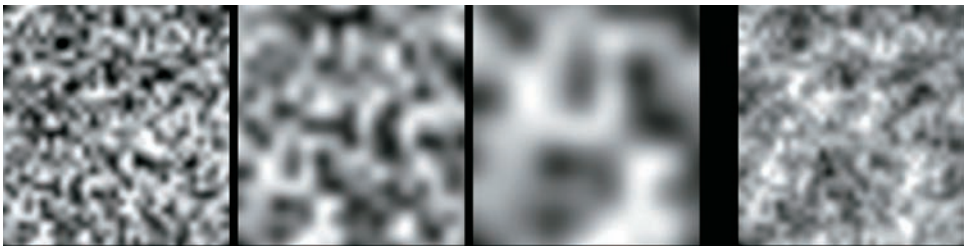
Fractals are common in nature. Fractal patterns are found in mountain ranges, cloud formations, ice crystals, lightning bolts, seed pods, plant leaves, and seashells. For a mathematician studying fractal math, the cracks and crevices of a rock are similar to those of an entire mountain range.

Naturally occurring fractal patterns

Fractals in the natural world differ from mathematical fractals in two major ways:

- Self-similarity encompasses a limited range of scale. That is, the recursion is finite. In other words, a tree branch does not split indefinitely into smaller and smaller branches.
- The self-similarity is less exact.

In contrast, Maya provides two fractal textures: Fractal and Solid Fractal. However, these textures do not replicate any specific fractal in nature or popular fractal studied in fractal mathematics. In fact, they are based on the Perlin noise. Perlin noise was developed by Ken Perlin in the early 1980s as a means to develop procedural textures. Perlin noise is able to create patterns by averaging multiple noise functions generated from a series of random numbers. In other words, multiple noise patterns, each at a different scale or frequency, are combined for complex results.



(Left) Three Perlin noise functions, each at a different scale. (Right) The averaged result.

Since the various scales are similar in appearance (interlocking blobby tubes), Perlin noise qualifies as a unique style of fractal. The purest Perlin noise in Maya is produced by the Fractal texture with its **Level Max** attribute set to 1. The Noise texture, when its **Noise Type** attribute is set to Perlin Noise, is almost identical. In fact, the **Level Max** attribute of the Fractal texture is similar to the **Depth Max** attribute of the Noise texture. Both attributes allow for additional scales of noise to be added.

All Maya procedural textures utilize variations of classic Perlin noise. For example, the Mountain texture generates Perlin noise but skips a smoothing function, which leaves hard-edged grains. The Marble and Wood textures apply Perlin noise to separate trigonomic functions, which in turn create looping or snaking veins.

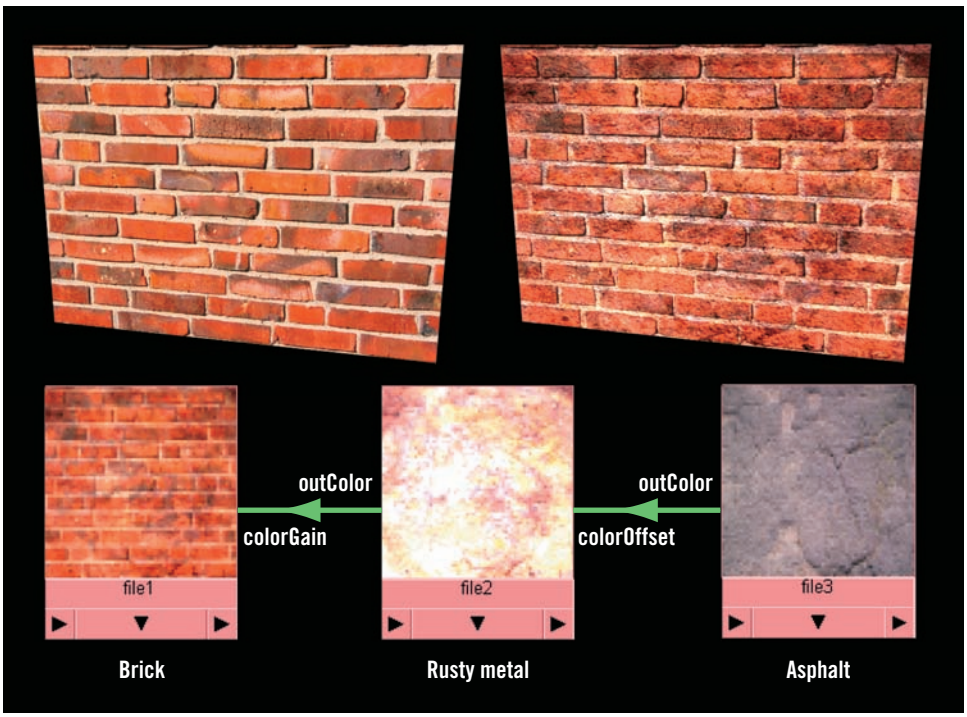
The one tool in Maya that mimics natural fractals is Paint Effects. Any of the plant or tree brushes follow simple rules to determine where branches, leaves, and flowers are produced. For example, when comparing the treeBare brush to a real tree, the way in which branches split is surprisingly similar. You can change the rules governing Paint Effects brushes by adjusting various attributes within the Tubes section of the Paint Effects Attribute Editor tab.

(Left) treeBare Paint Effects brush. (Right) Real tree.

Ultimately, it pays to think of Maya fractal textures as a means to "dirty up" or "randomize" a surface. Even the cleanest, most finely manufactured surfaces in the world are not immune to patterns. For example, plastic products often carry an extremely fine grain, which you can replicate by applying a Noise texture as a low-intensity bump map. A water glass picks up smudges, which you can replicate by applying a low-contrast Fractal texture as a transparency map. Of course, texturing is not limited to man-made items. If you'd like to convert a smooth hill into a 3D mountain range, apply a Noise texture as a displacement map. If you'd like to convert a primitive sphere into a slightly lumpy ball of clay, apply a Volume Noise texture as a bump map.

## Combining Generic Bitmaps in Maya

Tight production deadlines often necessitate quick texturing jobs. One trick is to combine generic textures within Maya. The **Color Gain** and **Color Offset** attributes, common to every texture, provides an efficient means of achieving this. For example, when a bitmap featuring a clean brick wall is combined with rusty metal and asphalt bitmaps, the end result is more complex and interesting.




(Top left) Clean wall bitmap. (Top right) Same bitmap with the addition of rusty metal and asphalt bitmaps connected through its **Color Gain** attribute. (Bottom) The corresponding shading network.

To achieve a similar result, open the Attribute Editor tab of the file texture in which the wall bitmap is loaded. In the Color Balance section, click the Map button beside the **Color Gain** attribute. In the Create Render Node window, click the File button. Load the rusty metal bitmap into the new file node. In the new file node's Color Balance section, click the Map button beside the **Color Offset** attribute. In the Create Render Node window, click the File button. Load the asphalt bitmap into the new file node. Continue the process of nesting file textures within file textures by mapping the Color Gain or Color Offset attributes. Theoretically, no limit exists for the number of textures that are strung together in a single shading network.

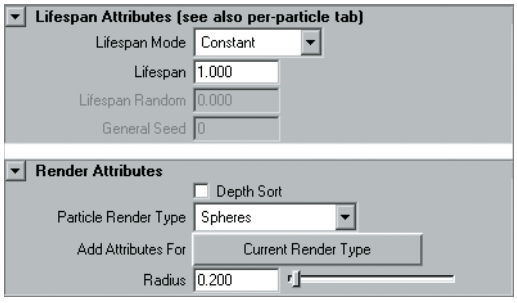
Technically speaking, **Color Gain** works as a multiplier. Whatever is mapped to **Color Gain** is *multiplied* by the original map. In contrast, **Color Offset** works as an offset factor. Whatever is mapped to **Color Offset** is *added* to the original map. Hence, **Color Gain** maps tend to make the end result darker and **Color Offset** maps tend to make the end result whiter and more washed out.

## Coloring Particles Individually

You can divide particle texturing into two main categories: hardware rendered and software rendered.

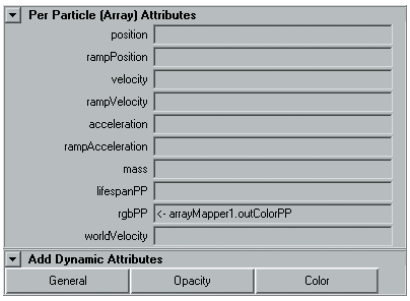
You can assign hardware-rendered particles, which include MuliPoint, MultiStreak, Points, Spheres, and Streak, to any material. However, only the color information is used. In addition, you must render the particles with the Hardware Render Buffer window or the Hardware renderer, or else the particles disappear. Although texturing capability of hardware-rendered particles is limited, you are free impart unique colors to individual particles. To do so, create a per-particle attribute with the following steps:

1. Create a new scene. Create a particle emitter by switching to the Dynamics menu set and choosing **Particles → Create Emitter**. Open the Hypergraph Scene Hierarchy window, select the particle1 node, and open its Attribute Editor tab. Switch to the particleShape1 tab. In the Lifespan Attributes section, switch **Lifespan Mode** to Constant. Change **Lifespan** to the number of seconds you would like the particles to live. In the Render Attributes section, switch the **Particle Render Type** attribute to the hardware-rendered particle type of your choice.




(Top) The Lifespan Attributes section. (Bottom) The Render Attributes section.

2. In the Add Dynamic Attributes section, click the Color button. The Particle Color window opens. Check the **Add Per Particle Attribute** check box and click the Add Attribute button. The window closes and the **rgbPP** attribute is added to the Per Particle (Array) Attributes section.



The Per Particle (Array) Attributes and Add Dynamic Attributes sections

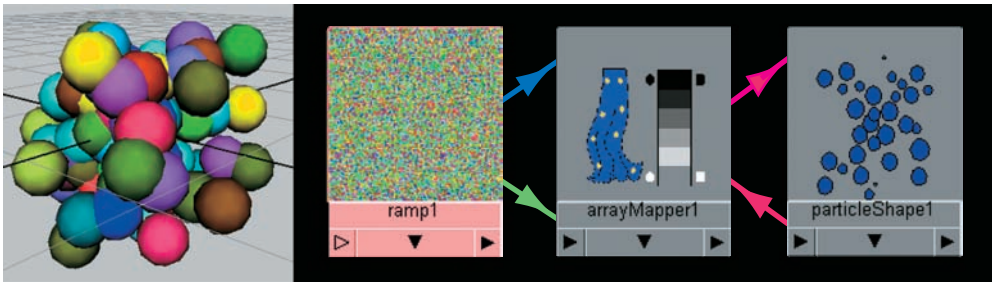
3. RMB click the field to the right of **rgbPP** and choose Create Ramp from the shortcut menu. A Ramp texture and an Array Mapper utility are added to the particleShape1 shading network. This is represented by the following text, which appears in the field to the right of **rgbPP**:

```
<- arrayMapper1.outColorPP
```

4. Open the Hypershade window. Switch to the Textures tab. MMB drag the ramp1 texture icon into the work area. Click the Input And Output Connections button (the icon has two arrows surrounded by a box). The arrayMapper1 and particleShape1 nodes are revealed. The Array Mapper utility is designed to relate the age of each particle to a position on the ramp texture. A young particle receives its color from the bottom of the ramp. An old particle receives its color from the top of the ramp. At this stage, the particles simply change from red to green to blue.

5. To make the particle colors more interesting, open the ramp1's Attribute Editor tab. Insert additional color handles into the ramp field. The more color handles you create, the greater the particle diversity. Change the **Noise** attribute to 1 and the **Noise Freq** attribute to 5. The ramp is thereby scrambled.

6. In a workspace view, check on Smooth Shade All. Play the Timeline. The particles take on unique colors based on the ramp1 texture.



Hardware-rendered particles colored with a Ramp texture and Array Mapper utility

An example scene, which uses Sphere hardware-rendered particles, is included as `hard_color.mb` in the Chapter 6 scene folder on the CD. With this example, the particles move through various colors. To assign a specific color to a particle and have the particle

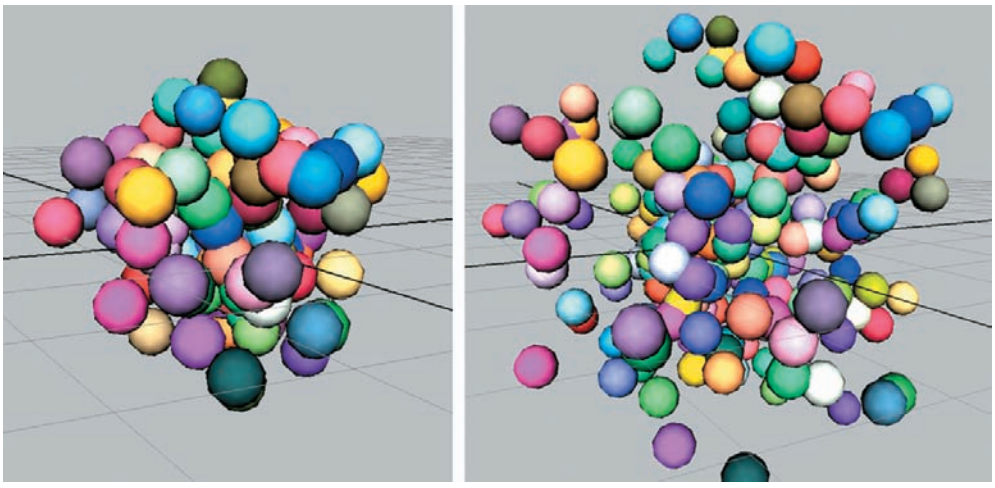
keep that color throughout its life, you must apply an expression. To adapt the particle emitter that was created at the start of this section, continue with these additional steps:

7. In the Per Particle (Array) Attributes section, RMB click the field to the right of **rgbPP**. Highlight the `<-arrayMapper1.outColorPP` menu item, which reveals a secondary shortcut menu. Choose Break Connection. The connections to the arrayMapper1 node are destroyed.

8. RMB click the field to the right of **rgbPP** and choose Creation Expression from the shortcut menu. The Expression Editor opens. Type the following code into the work area and click the Create button:

```
float $r = rand(1) + .25;

float $g = rand(1) + .25;

float $b = rand(1) + .25;

vector $pColor = <<$r, $g, $b>>;

particleShape1.rgbPP = $pColor;
```

9. Play the Timeline. Each particle receives a randomly chosen **Red**, **Green**, and **Blue** channel value, which is stored by the `$r`, `$g`, and `$b` variables. The `rand(1)` function generates random values from 0 to 1. An additional 0.25 is added to each channel, thereby biasing brighter colors. The three channels are stored in the vector `$pColor` so that they may be passed to the **rgbPP** attribute. A sample scene is included as `hard_color_2.mb` in the Chapter 6 scene folder on the CD.



Hardware-rendered particles assigned random, life-long colors

If you're curious as to which colors are assigned to an individual particle, RMB click over the particle in a workspace view, select Particle from the marking menu, and select the particle. Choose **Window → General Editors → Component Editor** and switch to the Particles tab. **rgbPP R**, **rgbPP G**, and **rgbPP B** values are listed in their own columns.

## Coloring Software-Rendered Particles

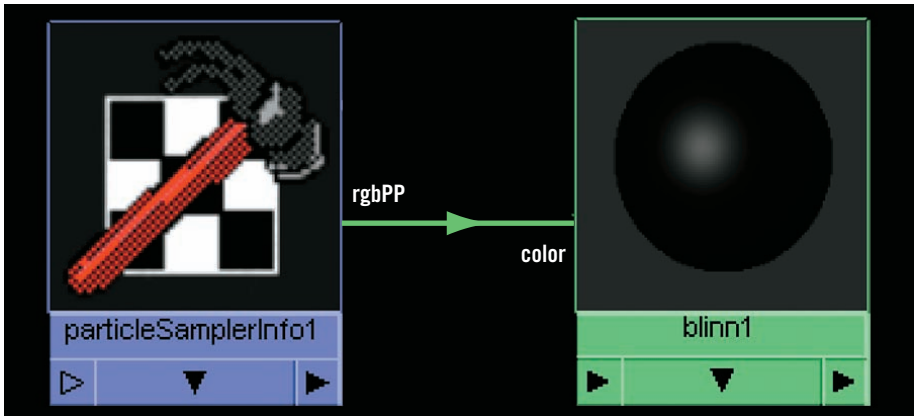
Cloud and Tube particles must be assigned to a Particle Cloud material. You can base the Cloud and Tube particles' color, transparency, and incandescence on their lifespan by mapping a Ramp texture to the **Life Color**, **Life Transparency**, and **Life Incandescence** attributes of the assigned Particle Cloud material. Maya automatically provides an Array Mapper utility and all the appropriate connections.

In contrast, you can assign Blobby Surface particles to Lambert, Blinn, Phong, Phong E, and Anisotropic materials. Blobby Surface particles have the advantage of supporting all the standard attributes of these materials, including specularity, transparency, and the application of color maps. However, to color Blobby Surface particles individually, you must connect a Particle Sampler utility to the particle network. To do this, follow these steps:

1. The initial steps are identical to those used for hardware-rendered particles. In fact, you can begin by following steps 1 through 4 at the start of the preceding section. The end result should be a particleShape1 node connected to arrayMapper1 and ramp1 nodes via the added **rgbPP** attribute.

2. Open the Attribute Editor tab for the particleShape1 node. Switch **Particle Render Type** to Blobby Surface. In the Hypershade, create a new Blinn material and MMB drag it into the work area. Assign the blinn1 node to the particleShape1 node. Play the Timeline. At this point, the particles remain the same color as the blinn1 node's **Color** attribute.

3. MMB drag a Particle Sampler utility (found in the Particle Utilities section of the Create Maya Nodes menu) into the work area and drop it on top of the blinn1 node. The Connect Input Of menu appears. Choose Other. The Connection Editor opens. In the left column, highlight **rgbPP**. In the right column, highlight **Color**. Close the Connection Editor.




A Particle Sampler utility connected to a Blinn material. No connection line exists between the Particle Sampler and the Array Mapper, which is connected to the particle node.

4. Play the Timeline and randomly choose frames to test-render. The appropriate ramp colors appear. (The particles must be rendered; the workspace view is not accurate in this situation.)

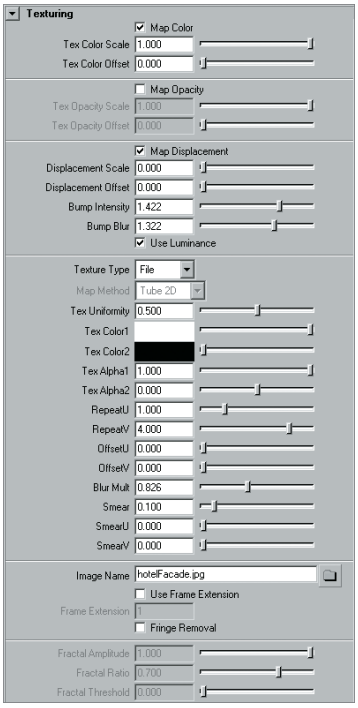
The Particle Sampler utility serves as a go-between, retrieving information from the Array Mapper utility and passing in on to the assigned material. For the Particle Sampler utility to function, it must be connected to a material that is assigned to a particle node that is connected directly to an Array Mapper utility. Since **Particle Type** is set to Blobby Surface, the particles have a desire to stick together. As such, the colors of the particles bleed onto neighboring particles. This effect is even more extreme if the **Threshold** attribute is raised above 0. A sample scene is included as `soft_color.mb` in the Chapter 6 scene folder on the CD.

Although the Particle Sampler utility is needed to individually color software-rendered particles, you are free to adjust the ramp texture or apply expressions as described in steps 5 through 9 of the preceding section.

## Texturing Paint Effects Tubes

Many Paint Effects brushes utilize Maya IFF bitmaps as textures. You can replace these with your own bitmaps for a customized result. The way in which the bitmaps wrap around the Paint Effects tubes, however, does not match standard geometry. Regardless, you can replace the texture following these steps:

1. Switch to the Rendering menu set. Choose **Paint Effects → Get Brush**. The Visor opens. Pick a brush by highlighting a brush icon. Brushes such as birchLimb (in the trees folder) or hotel (in the cityMesh folder) make obvious use of texture bitmaps. Paint a test stroke.
2. Select the resulting stroke and open its Attribute Editor tab. Switch to the Paint Effects tab to the immediate right. Expand the Shading section. In the Texturing subsection, change the **Image Name** attribute by browsing for the bitmap of your choice. Paint Effects brushes support any of the standard Maya image formats. Render a test. Your bitmap appears on the tubes generated by the stroke.



The Texturing subsection of a Paint Effects Attribute Editor tab

### Controlling the Paint Effects Texture

To control the way in which your bitmap repeats across each tube and how it is rendered, you an adjust the various attributes found above **Image Name** in the Texturing subsection.

Descriptions of these attributes follow:

**Texture Type** Determines what type of texture is applied to the tube. Checker, U Ramp, V Ramp, and Fractal options provide built-in procedural textures. The File option allows for a bitmap to be named by the **Image Name** attribute.

**Map Method** Determines the method by which the chosen texture is applied to the tube. The Full View option projects the texture from the camera view. The Brush Start option also projects the texture from the camera view, but the scale of the texture is determined by the distance the stroke is from the camera. The Tube 2D option maps the texture in world space and always centers the texture on the tube so that it faces the camera. The Tube 3D option maps the texture directly onto the tube so that it wraps around the circumference. If the camera is animated and the view shifts, Tube 3D is the best option. All the **Map Method** options are grayed out and inaccessible to any of the "mesh" brushes (for example, those in the plantsMesh, objectsMesh, and flowersMesh folders). Generally, mesh brushes map the texture in a fashion similar to 3D Tube.

**RepeatU, RepeatV, OffsetU, OffsetV, and Tex Uniformity** **RepeatU** and **RepeatV** determine how many times the texture is repeated in the U and V directions. On a tube, U runs the length and V runs around the circumference. **OffsetU** and **OffsetV** offset the texture in the U and V directions. Keep in mind that the **RepeatU** and **RepeatV** values do not always correspond directly to the render. For example, If both **RepeatU** and **RepeatV** are set to 1, a texture may appear multiple times along the tube of a long stroke. To control this, you can adjust the **Tex Uniformity** attribute. Low **Tex Uniformity** values force the renders to correspond more closely to the actual **RepeatU** and **RepeatV** values. High **Tex Uniformity** values increase the number of times the texture is tiled. **Tex Uniformity** works in a similar fashion for both the Tube 2D and Tube 3D mapping methods.

**Map Color, Tex Color Scale, and Tex Color Offset** **Map Color** must be checked for the tube to carry a texture bitmap. If **Map Color** is unchecked, the tube color is derived solely from the **Color1** attribute, which is found in the Shading section.

**Tex Color Scale** serves as a scaling factor that multiplies the texture color by the **Color1** color. If **Map Color** is checked and **Tex Color Scale** is set to 1, the texture is tinted the **Color1** color. If **Tex Color Scale** is less than 1, the texture color and **Color1** color are averaged. If **Tex Color Scale** is set to 0, only the texture color is used and **Color1** is ignored. **Tex Color Offset** serves as an offset for the **Tex Color Scale** calculation. The higher the **Tex Color Offset** value, the more white is introduced into the color.

**Tex Color1 and Tex Color2** **Tex Color1** is the first color used by the **Texture Type** attribute. **Tex Color2** is the second color. If **Texture Type** is set to File, **Tex Color1** tints the highest values in the bitmap and **Tex Color2** tints the lowest values. If **Texture Type** is set to Checker, U Ramp, V Ramp, or Fractal, **Tex Color1** and **Tex Color2** provide the colors for the resulting pattern.

**Map Opacity, Tex Opacity Scale, and Tex Opacity Offset** If checked, **Map Opacity** supports alpha information provided by the texture determined by **Texture Type**. If **Texture Type** is set to Checker, U Ramp, V Ramp, or Fractal, the RGB values of the texture are interpreted as alpha (by default, black becomes transparent). If **Map Opacity** is unchecked, opacity is determined by the **Transparency1** attribute, which is found in the Shading section. **Tex Opacity Scale** is a scaling factor that multiplies the texture's alpha values by the **Transparency1** value. **Tex Opacity Offset** serves as an offset for the **Tex Opacity Scale** calculation.

**Tex Alpha1 and Tex Alpha2** **Tex Alpha1** defines the opacity values of pixels in the texture that possess the highest RGB values. **Tex Alpha2** defines the opacity values of pixels in the texture that possess the lowest RGB values. To reverse the transparency of a tube, set **Tex Alpha1** to 0 and **Tex Alpha2** to 1.

**Map Displacement, Displacement Scale, and Displacement Offset** If checked, **Map Displacement** displaces the tube geometry using the texture defined by **Texture Type**. **Displacement Scale** sets the amount of displacement relative to the tube width. Higher values create greater displacement. If the RGB values within the texture are less than 0.5, the tube is displaced inward. You can reverse this effect by using a negative **Displacement Scale** value. **Displacement Offset** determines the amount of additional displacement that is independent of the tube width. Displacement is supported only by "mesh"-style brushes.

**Bump Intensity and Bump Blur** If **Map Displacement** is checked, a bump mapping effect is created with the texture defined by **Texture Type**. The higher the **Bump Intensity** value, the more intense the bump mapping. **Bump Blur** softens the bump map. To see the bump map by itself, set **Displacement Scale** and **Displacement Offset** to 0. For the bump map to appear, you must also check **Per Pixel Lighting**, which is found in Mesh section. **Per Pixel Lighting** forces the tube lighting calculation to be carried out on a per-pixel basis rather than a per-vertex basis.

**SmearU, SmearV, and Smear** **SmearU** and **SmearV** distort the texture in the U and V directions by applying a noise defined by the **Smear** attribute. The **Smear** value determines the frequency of the distorting noise. The higher the value, the higher the noise frequency, the finer the distortion detail, and the more obscured the original texture is. If **SmearU** and **SmearV** are set to 0, no distortion occurs.

**Blur Mult** Defines the amount of anti-aliasing applied to the texture. A value of 1 produces fair results. Higher values create blurrier textures. A value of 0 turns the anti-aliasing off.

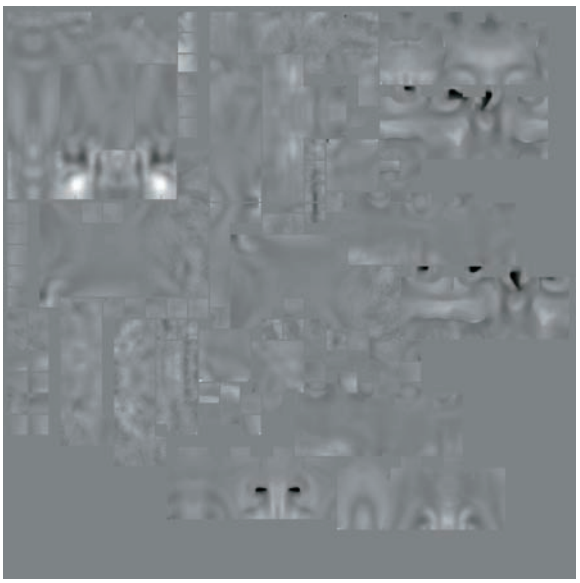
As a final note, **Color2** and **Transparency2** attributes, found in the Tube Shading subsection, are not available to all Paint Effects brushes. If they are available, they determine the color and opacity of the tube tips. In addition, their values are subject to calculations involving **Tex Color Scale**, **Tex Color Offset**, **Tex Opacity Scale**, and **Tex Opacity Offset**.

## Industry Tip: Rendering ZBrush Displacement Maps in Maya

Pixologic ZBrush is fast becoming an important modeling tool in the 3D visual effects and gaming world. Since the program is fairly new, integration with Maya remains somewhat unintuitive. In particular, importing displacement maps and rendering them properly is fraught with peril. Rocky Bright, Jr., a media arts and animation instructor at the Art Institute of Las Vegas, offers the following steps for avoiding displacement headaches. Many of the individual tips in this section were originally put forward by such animators as Sunit Parekh, Scott Spencer, and others posting to the ZBrush Central forum (`www.zbrushcentral.com`).

The first two steps prepare the displacement map within ZBrush and Photoshop:

1. Create a displacement map within ZBrush. In short, this involves the creation of a low-resolution object, the tessellation of the object through the Divide tool, the painting of detail, and the exportation of the displacement bitmap. If you're new to ZBrush, you can find tutorials within the ZBrush help topics. Aside from standard ZBrush procedures, three special steps are recommended:
   - While the displaced object remains low resolution, export it out as an OBJ file through the **Tool → Export** menu. The file will be loaded into Maya in step 3.
   - Before the bitmap is exported, open the **Alpha** menu and note the value beside the **Alpha Depth Factor** attribute; this number is required in step 5.
   - Save the displacement bitmap in the TIFF file format.



An exported displacement map. ZBrush saves the image in 16-bit grayscale.

2. Open Adobe Photoshop or a similar digital paint program. Double-check the color management profile. Ideally, you *do not* want the program to apply color management to any opened displacement bitmap. If the program does, the bitmap's grayscale may shift in value and the displacement may be thrown off. To turn off the color management in Photoshop CS2, for example, choose **Edit → Color Settings**. In the Color Settings window, set the Working Spaces **RGB** drop-down menu to sRGB IEC61966-2.1. Set the Color Management Policies **RGB** and **Gray** drop-down menus to Off.

   Open the displacement TIFF. Popular paint programs, such as Photoshop CS2, correctly recognize the bitmap as a 16-bit grayscale image. Convert the bitmap to 16-bit RGB. The displacement must be converted to RGB for Maya to interpret it correctly. In CS2, choose **Image → Mode → RGB Color**.

   If you did not flip the image vertically in ZBrush, do so now. If the image is not flipped, it will appear upside down on the model.

   Save the image. The bitmap must be saved as 16-bit. If the bitmap is inadvertently converted to 8-bit, banding and stair-stepping will occur over the entire model.

   The remaining steps occur within Maya:

3. Create a new scene. Import the low-resolution OBJ file created in ZBrush. If the object imported into Maya is faceted, switch to the Polygons menu set and choose **Normals → Soften Edge**. Delete the history by choosing **Edit → Delete By Type → History**.

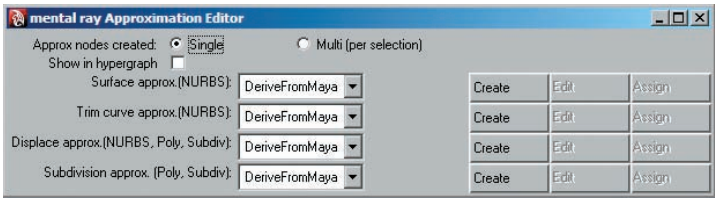
4. Open the Hypershade and assign a Blinn, Phong, or Phong E material to the object. Using a material with specular quality makes the displacement detail easier to adjust. Open the material's Attribute Editor tab. Switch to the shading group node tab by clicking the Go To Output Connection button (beside the Presets button). Click the Map button beside the **Displacement Mat**. attribute. In the Create Render Node window, click the File button. Open the new file node in the Attribute Editor, click the file browse button beside **Image Name**, and load the displacement TIFF.

5 In the Color Balance section of the file node's Attribute Editor tab, change the **Alpha Gain** to the same value that was listed for **Alpha Depth Factor** in ZBrush. To determine a correct **Alpha Offset** value, use this formula:

```
Alpha Offset = (Alpha Gain / 2) * -1
```

   The purpose of this formula is to remap 50 percent gray values to 0. Thus, what represents zero displacement in ZBrush equals zero displacement in Maya. By default, the **Alpha Gain** and **Alpha Offset** attributes control the intensity of all displacement maps within Maya. **Alpha Gain** is a multiplier and **Alpha Offset** is an offset factor, much like **Color Gain** and **Color Offset**.

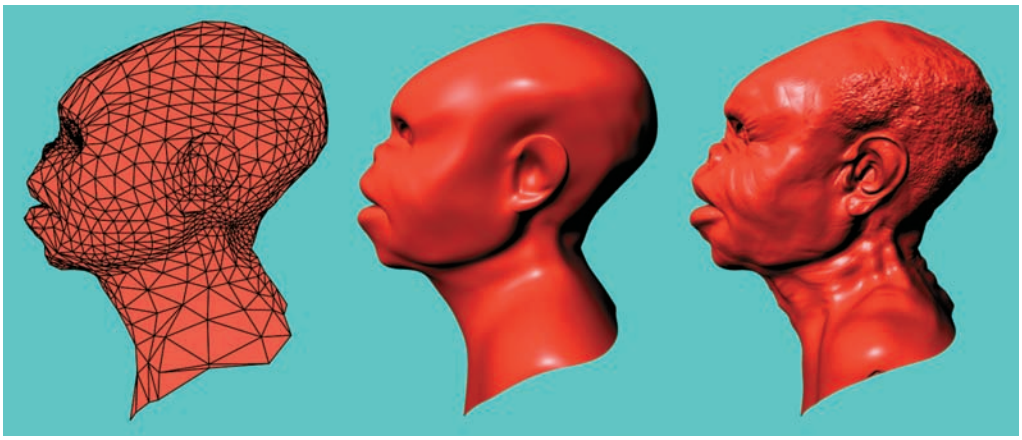
6. With the polygon object selected, choose **Window → Rendering Editors → mental ray → Approximation Editor**. With the mental ray renderer, approximation is the process by which NURBS are converted to polygons and polygons are triangulated for displacements. (Approximation is roughly equivalent to the tessellation of NURBS undertaken by the Maya Software renderer).




The mental ray Approximation Editor

With the polygon object selected and the mental ray Approximation Editor window open, check the **Show In Hypergraph** check box. Click the Create button beside the **Subdivision Approx.** attribute. The Hypergraph window automatically opens and displays a new mentalraySubdivApprox1 node connected to a mentalrayItemsList node. Click the Edit button that appears to the immediate right of the Create button in the mental ray Approximation Editor window. The Attribute Editor tab for the mentalraySubdivApprox1 node is loaded. Change the **Approx Method** attribute to Spatial.

7. Open the Render Settings window. Switch **Render Using** to mental ray. Render a test.



(Left to right) Triangulated low-resolution head; same head with the Soften Edge tool applied; final render of displaced head. You can download this model, along with the matching displacement map, at `www.pixologic.com`.

If the displacement is too subtle, increase the **Alpha Gain** value and change the **Alpha Offset** accordingly. If the displacement detail is not fine enough, you can increase the mentalraySubdivApprox1 node's **Min Subdivisions** and **Max Subdivisions** attributes.

Occasionally, imported displacement bitmaps cause triangular artifacts or unusual creases to appear on the model. Should this happen, you can apply the Smooth tool. Select the object and choose **Mesh → Smooth** with the default options. Since the polygons are subdivided, this can significantly slow the render.

If rendering artifacts persist or the model appears puffy and balloonlike, double-check the displacement TIFF to ensure that the colors have not been shifted by incorrect color profile management or bit depth. Slight variations in RGB values can lead to major changes in the displacement.

If the camera is destined for animation, open the mentalraySubdivApprox1 Attribute Editor tab and uncheck **View Dependent**. If **View Dependent** is left checked, render "pops" may occur on the displacement. Also, if the rendering is bogging down, one way to save a time is to uncheck **Ray Tracing** in the mental ray tab of the Render Settings window.

It's possible to render ZBrush displacement maps with the Maya Software renderer. However, the results are generally inferior. If you do choose this route, you can refine the displacement by increasing the **Initial Sample Rate** and **Extra Sample Rate** values (found in the Displacement Map section of the surface's Attribute Editor tab).

## ROCKY BRIGHT, JR.

Rocky is a lead instructor at the Art Institute of Las Vegas, where he specializes in texturing, lighting, and visual effects classes. Before joining the institute, he worked as a modeler for Tri-Dimensional Studios in Tampa, Florida, and Trevi Manufacturing, Inc. in Las Vegas, Nevada. To learn more about Rocky's work, visit `www.thingamatoon.com`.