PovZine

Summer 1996

- Moray Tutorial
- GIF Animation
- Animation w/Perl
- Review of Trispective
- Tips & Tricks

Editorial

We're Back

Well, I'm back! I've been on a short sabbatical that turned into a year long project. During that time I've had no free time to devote to PovZine (and other assorted duties that my spouse keeps reminding me about).

At any rate a lot has happened since the last PovZine issue. I've become a published author (my book "3D File Formats: A Programmers Reference" will be in book stores in September), POV-Ray V3.0 betas have had a public release, and most importantly the O.J. trial is over.

A Public Apology

I'd like to publicly apologize to several folks that have submitted articles. This issue has taken far longer to produce than I would have ever imagined. In this issue I've simply tried to get some of my backlog published. There are several articles that didn't make to this issue (because I wanted to get something out quickly). I expect to publish another issue, soon, that will contain the remainder of my article backlog. Plus some new material.

Distribution Format

At this point it looks like Adobe has won the portable documentation wars. From now on I will be publishing PovZine as an Adobe Acrobat document (and an HTML version as long a Gita continues to convert my source documents – Thanks Gita). I am discontinuing distributing a Common Ground version starting with this issue. I'm somewhat reluctant about this decision, but am forced into it because the current free reader for Common Ground doesn't appear to work reliably on Windows NT systems.

Cool Software

One great things that has happened in the last year is the appearance of Trispective from 3DEye. Trispective is the first mainstream commercial software package (that I'm aware of) that directly supports POV-Ray. Egghead Software, in their summer catalog, has Trispective priced at $149.00. Trispective can export models as POV-Ray V2.2 files. Granted, the conversion process isn't totally painless, but considering the other commercial options this is a "Good Thing."

Final Ramblings

I've heard that Waite Group Press has been bought out by Macmillian and that they've pulled Ray Tracing Creations out-of-print. This doesn’t bode well for POV-Ray users who have had the advantage of being able to buy Chris Young's great book. I was hoping for a POV-Ray V3.0 upgrade to RTC, but that doesn't look likely very now.

Keith Rule
http://www.europa.com/~keithr
keithr@europa.com

PovZine

Summer 1996

Mission Statement

PovZine exists to serve the POVRay users community by providing useful POVRay related information in a way that is friendly and [hopefully] entertaining.

This magazine is intended to be freely available to all POVRay users.

Information

PovZine is published when enough time and material avaliable to create an issue. It is available at the URL `http://www.povray.org/povzine` and at `http://www.europa.com/~keithr`.

Submissions

PovZine is looking for unsolicited articles, art, questions, etc. Please email information about your article, art, or your questions, to keithr@europa.com. ■

Features

The Next Issue - I expect the next issue to be out in the mid-fall (probably around Nov. 1, 1996). If you have suggests, art work, or articles to contribute please send me email at *keithr@europa.com.*

#Declare Beginner

by Andy Moorer and Shannon Moon
Email: flyingshan@aol.com or
smoore@garnet.acns.fsu.edu.

Q: How do I run it?

Errr, don't expect to execute POV and suddenly have a program with handy menus and a graphical interface. It doesn't work that way. How people use it to make graphics usually goes something like this.

They use a plain text editor or word processor to write a text-file. This file usually has the extension .pov (instead of name.txt or whatever) in order to set it apart as a POV file to the author. The file contains a list of commands which describe the scene the author (or artist) wishes to create. The artist then runs that file through POV which interprets the file and creates a 24-bit Targa (tga) file at whatever resolution the author specifies. They do it using commands like this:

```
povray +Iinput.pov +Ooutput.tga +W640
+H480 +Linclude
```

...or some variant of this. The +I +O +W etc. are 'switches' which tell POV specifics such as the name of the input file, the output file, the width and height of the image and where to find other files which are referenced in the artists .pov file (these are usually things called include files and they have the .inc extension - this comes up later and is important.)

This sounds like a lot of work (and it is) but it is also the power of POV. Artists have a degree of control over their image which isn't otherwise possible. Also, it makes it possible to write an incredible number of utilities to generate objects, animations, special effects and so on. There are many "modeler" programs out there which let you create your images using wireframe graphics which are then automatically translated into the special .pov text files. Think of POV as a 3-d graphics language and povray.exe as its interpreter.

Q: I made a POV text file and ran it through povray.exe. Why is it complaining that it can't find a file?

This is the single most common question of new users. Did the text file make any references to other files life GIFs or .inc files? Look for these statements at the top of your file:

```
#include "colors.inc"
#include "textures.inc"
#include "shapes.inc"
```

These statements instruct pov to find the file in quotes and read them as well. These files (called include files) are text files which usually define a bunch of terms to be used later. For example, the colors.inc file defines colors by a name that you can reference (such as White) rather than a lengthy definition of a white color (which would be color red 1.00 blue 1.00 green 1.00). These three above are the most common, and they are in almost every POV file because they are so useful. It's a good idea to print out and read these so you know what's in them.

Anyway, if the .pov file you're trying to render has any references to outside files, povray will look in the same directory it's in, and then every directory you define using the +L switch. The includes are usually kept in a directory names "include," so almost every render uses the switch "+Linclude." Get it? You can have a bunch of directories defined by using +L several times if you wish:

`povray +imyfile.pov +omyfile.tga +W640 +H480 +linclude +lproject +lstuff` ...and so on.

If the file still isn't found, or you never used the +L switch, you get the error.

Q: Ok, I made a file and rendered it, but it's all distorted. What's going on?

First, blocky looking files. If you used the +D switch when rendering POV will draw on the screen a poor rendition of what your file looks like. This is only that, a poor rendition. POV targa files contain much more information than you see on the screen this way, and are full, beautiful 24-bit color files. That's why they're so big. When you convert them to gifs, jpegs etc. you lose a lot of information. You will also lose information when displaying them on a cruddy monitor, or using a cruddy display program, or printing them out at low resolutions (such as with a 300dpi printer.) But the information is there in the file, and will come out if you do something like display it on a high end system or send it off for printing on an imagesetter or whatever.

Also remember that the image has only as many pixels as you define using the +W and +H switches. If you take a 120 by 80 image and make it fit to the edges of your 21 inch monitor you're going see a nice, blocky image. If you want higher quality images you've got to render at higher resolutions. Also, make sure that POV has its quality switch set at +q9 for top quality (see the manual for more info on switches to adjust specific aspects of image quality.)

It's up to you to make sure your file has something in it to render, and lights to see it by. If you want to render the inky blackness of empty space you're not going to see much.

Let's see, distortion... if your image is coming out stretched, make sure that your "aspect ratio" is the same in the rendered file and your screen resolution. For instance a 200x200 pixel image has a 1 to 1 aspect ration, but a 640x480 screen does not. If you display this pic on a 640x480 screen it's going to look stretched. It's not, but it will display that way. Try re-rendering the image using +W and +H to match your screens aspect ratio. There is a LOT more to resolution and aspect ratios, and already we're getting beyond the scope of this text. POV itself allows you to define attributes of it's camera just as if you were taking a picture in real life, and these also effect the image and can cause distortions etc. If this info above doesn't help, then ask people in the POV community. They're usually eager to assist, particularly after you've made a real effort to figure it out yourself! :)

Q: Can I make animations using POV?

Yes, but I suggest getting familiar with POV before attempting animation with it. Most animations in POV use batch files to create and render each frame of the animation. You end up with a bunch of .tga files, each being a frame of the animation, which are usually then spliced together using a program called DTA (Dave's Targa Animator) to create animations in the .fli and .flic formats.

POV includes a special variable called "clock." The value of clock is defined by the +K switch when rendering. You can make a batch file which changes the value after the +K switch (+k1 +k10 +k20 etc.) each time a frame is rendered, and objects can then be translated, rotated, scaled etc. by that clock amount... a lot is possible but it takes some clever thinking and usually some good utilities to do anything of complexity.

Happily there are ample utilities out there for animation. POV 3.0 is said to include a great deal of animation support, opening up the possibility of creating animations with greater ease than ever . Other upcoming possibilities are realworld physics animation, collision detection, keyframe animation, particle systems and more. This is possible but NOT simple using version 2.x. Most POV users are very eagerly awaiting the possibilities version 3.0 will open up. It will add a tremendous amount of power to an already powerful program, and is a major upgrade.

Q: This is too much like work. Why should I use this when I can pay for a package which makes raytracing easier?

There are a lot of answers to this.

First, those packages are expensive (think $200 for the most cheesy to $2500 or more) and unless you're going to get the higher end packages their output often isn't as good. POV is free and require more effort, but delivers images comparable to the output of packages that cost big-time.

Second, the commercial packages aren't exactly easy to use themselves. There's no simple way into quality 3-d. If you're looking for quick-and-simple 3d, there are indeed packages (such as simply 3d etc.) which are cheap and easy. If you really want to get into it, POV isn't much harder than any of the other professional level packages.

Third, POV gives you complete control of your scene and an understanding of your image that usually results in your learning many more subtle tricks of the trade. By getting into the guts of the image you develop a way of thinking which is more powerful: it's the same sort of difference that exists between a c programmer and a user of a multimedia development package. The results may be similar at first, but the c-programmer can go places the other can't.

Fourth, once you master POV you have an understanding of the concepts and methods used in all 3d packages, and you'll find it's much easier to get started. You'll also find yourself doing more with the software than you would have without the knowledge you gained using POV. You'll also have the respect of the artists who know POV. Look in an issue of the magazines "3d Artist" or "Computer Graphics World" and you'll often see references to POV and it's cousins (Rayce, polyray etc..)

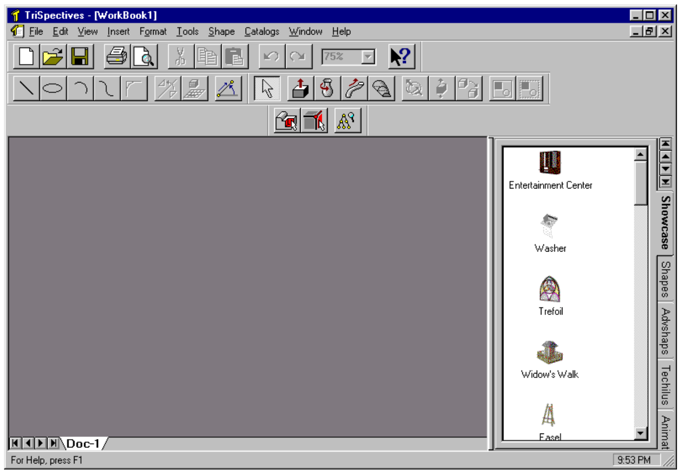
Finally, who says you can't use both? The commercial packages have a lot to offer if you have the money for them. Many professional artists out there use several packages, including pov, in conjunction with each other. ❑

A Review of Trispective

Keith Rule

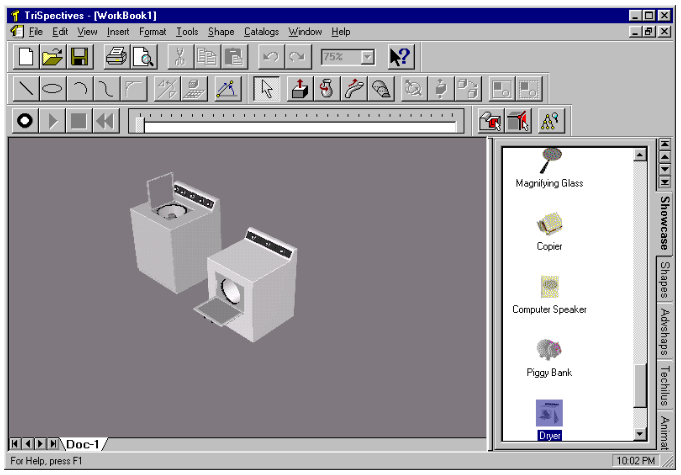
Trispective is a commercial 3D program by 3D/Eye Inc. It is a 3D design program for Win95 and Windows NT. For a POVRay user this is more that just yet another 3D application. This 3D program writes POVRay V2.2 files!

A Simple Animation Example

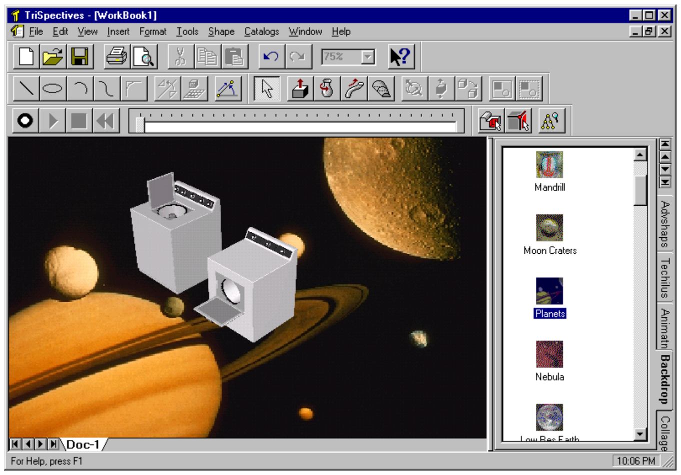


When you start up Trispective you are presented with a tool palette on the top of the window and a catalog of drag-and-drop items along the right side. Each of the tabs of the catalog displays a palette of items that can be dropped on to a scene or an object in a scene. The tabs include *Shapes*, *Showcase*, *Collage*, *Colors*, *Surfaces*, *Decals*, *Animation*, *Dimension*, and *Text*.

To design a simple scene we could drop a washer and dryer onto our worksheet.



To add a background to the scene we can simply drag a new background onto the document from the backdrops catalog.

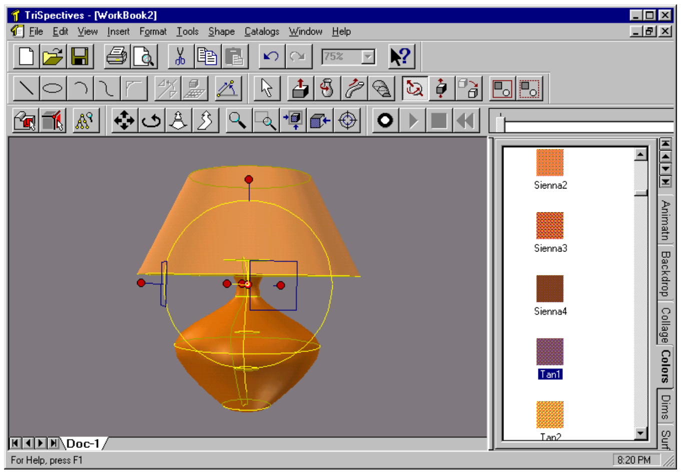


To animate the scene all you have to do is drop an animation object onto each of the objects in the scene. For this scene I dropped the *Height Spin* and *Length Move* objects from the animate catalog on both the washer and the dryer. All I need to do play the animation, which shows a washer and dryer spinning in space from the center to the bottom left of the screen, is press the animate and then the play buttons on the toolbar.

In just a few seconds I was able to build a debatably interesting animation scene. To save the animation as a Windows AVI file is choose File|Export Animation and fill out the dialogs. ❑

Exporting to POVRay

Let's use a model of a lamp that I created with Trispective.


TriSpectives - [WorkBook2]
File Edit View Insert Format Tools Shape Catalogs Window Help
Sienna2
Sienna3
Sienna4
Tan1
Tan2
Animatn Backdrop Collage Colors Dims Surf
Doc-1
For Help, press F1
8:20 PM

I created this lamp model by using the "Spin Shape" tool to create that base of the lamp and the lamp shade. I then added surface colors to the components and then grouped them together.

To export this as a POVRay model you select File|Export Model… and choose POVRay as the file type. By default, Trispective wants to emit the model with a suffix of .pov. In reality, Trispective emits an object rather than a scene (meaning there are no lights or cameras). Since I prefer POVRay objects to be placed in .inc files, I named the POVRay output file as 'lamp.inc' and then choose Save.

The resulting file looks like the following:

```
// world bounding box:
// low <182.3,-176.785,140.731>
// high <451.731,93.1506,428.706>

#declare __Mtl1 = texture
{
    pigment {
        color red 1 green 0.501961 blue 0
    }
    finish {
        diffuse 0.9
        phong 0.6    phong_size 65.25    }
}

#declare __Mtl11 = texture
{
    pigment {
        color red 1 green 0.647059 blue
0.309804
    }
    finish {
        diffuse 1
        phong 0.5    phong_size 0    }
}

// object bounding box:
//  low <228.953,-136.4,145.886>
//  high <408.927,43.4063,222.95>

#declare _lamp01 = union
{
    smooth_triangle{
      <329.381,-82.9699,152.845>,
      <0.177929,-0.636824,-0.750198>,
      <330.687,-87.9868,157.474>,
      <0.180445,-0.646485,-0.74128>,
      <321.662,-89.3899,157.38>,
      <0.039427,-0.66841,-0.742747>
      }

      .
      .
      .

#declare lamp = union
{
  object { _lamp01 texture { __Mtl1 } }
  object { _lamp02 texture { __Mtl1 } }
  object { _lamp03 texture { __Mtl1 } }
  object { _lamp04 texture { __Mtl1 } }
  object { _lamp05 texture { __Mtl1 } }
  object { _lamp06 texture { __Mtl1 } }
  object { _lamp07 texture { __Mtl1 } }
  object { _lamp08 texture { __Mtl1 } }
  object { _lamp09 texture { __Mtl1 } }
  object { _lamp10 texture { __Mtl1 } }
  object { _lamp11 texture { __Mtl11 } }
  object { _lamp12 texture { __Mtl11 } }
  object { _lamp13 texture { __Mtl11 } }
}

// realize instance of defined object:
_lamp
```

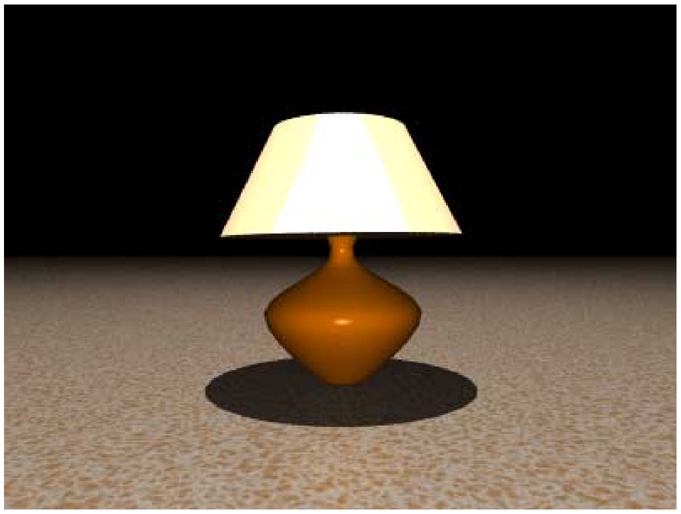
All this POVRay code looks pretty good except for the last couple of lines. The last line is a syntax error and should be removed. I checked with the folks at Trispective and they agree this is erroneous code. They will fix this problem in a future release. However, this is a very minor problem and doesn't really affect the usefulness of the POVRay code that been emitted.

Notice that the beginning comment is the bounding box for the world. This information is very useful when creating a POVRay scene that includes this lamp object.

To create a scene that will render this object

```
#include "colors.inc"
#include "lamp.inc"

camera {
      location  <0, 0, -2>
      direction <0, 0,  1>
      up        <0, 1,  0>
      right   <4/3, 0,  0>
      look_at   <0, 0, 0>
}

light_source {
      <0, 0, -2>
      color red 0.75 green 0.75
      blue 0.75
}
light_source {<0, 5, 0> color White}


// lamp bounding box:
//  low <182.3,-176.785,140.731>
//  high <451.731,93.1506,428.706>
// The bounding box info is from lamp.inc
object {
  lamp

  // Center the lamp at <0, 0, 0>
  // by performing this generic
  // translation: translate <
  //        (low.x - high.x)/2 - low.x,
  //        (low.y - high.y)/2 - low.y,
  //        (low.z - high.z)/2 - low.z>
  translate <-317.0155, 41.8172,
                -284.7185>

  // Scale the lamp so that it's
  // height is 1.
  // This is done by performing
  // the following generic
  // scale operation:
  // scale <1/abs(low.z-high.z), ...>
  scale <1.0/287.975,
      1.0/287.975, 1.0/287.975>

  // Z is up in Trispective, this
  // rotation makes Y up.
  rotate <-90, 0, 0>
}

// Put down a speckled floor
plane {
  y, -0.5
  texture {
    pigment {
       granite
         color_map {
            [0.0 1.01 color
            White color Gold]
         }
         scale <0.1, 1.0, 0.1>
    }
    finish {
      ambient 0.075
      diffuse 0.5
      reflection 0.05
      }
    }
  }
```

This POVRay code defines a camera, some light sources and instantiates a lamp object. The object is centered at <0, 0,0> and scaled so that the height of the lamp is 1.



The resulting image is not bad considering the original model took about 2 minutes to create with Trispective. The conversion to a POVRay model only takes a few seconds.

Incorporating the model into the scene is very straightforward. Hopefully, the next version of Trispective will export complete POVRay scenes.

For a die-hard POVRay user like myself, having a reasonably low-cost commercial modeler that can write POVRay models is very attractive. Trispective can be found in many discount mail order software catalogs and at many software stores such as Egghead Software.

If you'd like to try a free (well it's really about $10.00) 30 day trial version of Trispective, you can order that directly for 3D-Eye. The street price of Trispective is between $150 and $200 in the US. The Pro version is a bit more. To contact 3D-Eye either check out there webpage (http://www.eye.com/) or call them using their toll-free number 1-800-469-6514. ❑

Tutorial #2 - Modeling with MORAY 2.0's Bezier Editor

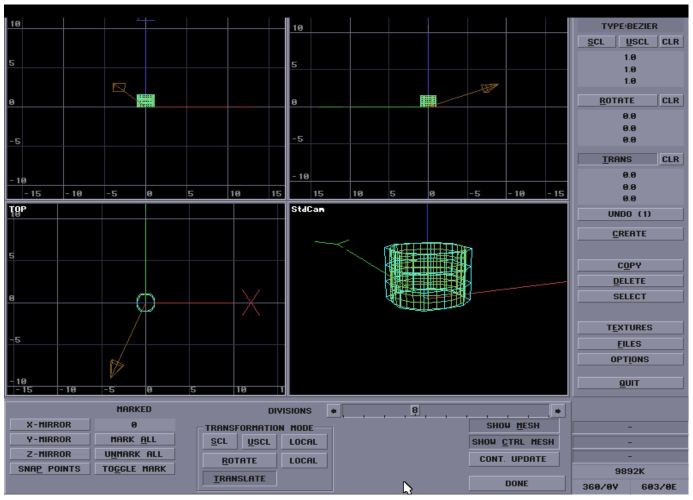
By Robert A. Mickelsen

Many people have commented on the objects I have created using the bezier editor in MORAY, and it has always been a bit of a mystery to me why. To me, the bezier editor is highly intuitive and relatively easy to use. The use of control points to model is much easier than trying to drag around triangle vertices. It just takes a bit of practice.

At the request of several friends I have written the following tutorial to demonstrate the basic method of modeling with the bezier editor. This tutorial assumes a basic knowledge of POV-Ray 2.2 and MORAY 2.0. Both programs are required to follow along.

We are going to model a fish complete with detailed fins. It will be composed mostly of bezier patches although there will be some parts made of primitives. To begin, start MORAY and go with the default camera and light source.

1. Click on CREATE, BEZIER PATCH, and name it 'Body1'. Select 'Cylinder (4 - patch) and click on 'OK, Create'. Click on 'Extended Edit'.



This is the bezier editor. It appears much the same as the main screen editor but will only contain the bezier object and a camera. No other objects or light sources will be present. The STD CAM view is the same as in the main edit screen. The TOP, FRONT, and SIDE views also work the same but independently of the main edit screen. You can pan around the scene in any window by dragging the mouse with the CTRL key depressed and zoom in and out in any window by dragging the mouse with the ALT key depressed, just as in the main edit screen. Below the four views are groups of buttons. On the right side is the Divisions slide bar. This will effect the complexity of the display but will have no effect on the final bezier object. Accept the default value of eight for now. Directly beneath that are three buttons. The first one toggles 'Show Mesh'. The patch mesh is green and represents the actual bezier object you are editing. The second button toggles 'Show Ctrl Mesh'. The control mesh is light blue and represents the control mesh for the patch object. There will be times when you may want to see just the patch mesh or just the control mesh, so you turn the other one off.

On the left side are the mirror buttons. These simply flip the object around whatever axis is indicated. Next to them, under the word 'Marked', are three buttons. 'Mark All' selects all control points in the bezier object. 'Unmark All' deselects all selected points in the bezier object. 'Toggle Mark' turns off and on the last group of selected control points. Control points can also be marked (selected) by clicking on them with the shift key depressed or by dragging a box around them with the shift key depressed. The real power of this modeler comes from the ability to select specific points and perform transformations on them.

The final group of buttons is the transformation mode box. These transform selected points in a manner consistent with the main editor. By clicking on 'local' in either rotate or scale, you transform the marked points around a local origin rather than around the world origin. (This is a feature that I wish POV had).

2. Begin by going to each of the view windows and Alt-dragging to zoom in so that the patch object is larger and easier to see. Click on the 'Mark All' button and then on 'Local Rotate'. Use the mouse in the Front view window to rotate the cylinder x*90.

3. Click on SCL (scale) and drag the mouse in the front view so that the cylinder is about 3 units long (x) and 2 units high (z). Then drag the mouse in the side window so that the cylinder is about 1 unit wide (y). Note that there is no way to enter numerical values for these transformations as in the main edit screen, nor is there any undo button. The transformation and undo buttons to the right of the screen have no effect in the bezier editor. (Write to Lutz if you don't like this).

4. Now, in the front window hold down the shift key and drag a box around all the control points except the ones at the extreme left. This will unmark them. In the side window, with the LOCAL SCL button still depressed, drag the remaining marked points until they make a small ring. Alt-drag to zoom closer and scale them smaller still. Be careful not to allow the points to cross.

   Then Alt-drag to zoom *out* until the entire patch can be seen in that window again. (Hint: If the grag-scaling seems too sensitive and the points seem tojump from one extreme to another, try zooming in a little closer. That willsometimes clear up the problem.)

5. Click on 'Mark All' and drag in the Front view until the object is about 3 units high and 4 units long. Shift-drag a box around the points on the extreme right and extreme left to unmark them, and then drag-scale the remaining points in the Front window to give your fish body a tapered bullet-like shape. Turn off the control mesh to examine your shape more closely. Note that you can still perform the scaling with the control mesh turned off as long as points are still selected. Finally, 'Mark All' and translate the entire object so that the broad end rests precisely on x=0 and is perfectly centered on the y and x axis. Click on 'Done' and again on 'Done' to return to the main edit screen.

6. In the main edit screen, copy the bezier object (Body1) without any transformations. Click on 'OK', then 'Extended edit'. Back in the bezier editor, click on 'x-mirror', then on 'Unmark All'. Zoom into the side window until you can once again see the tiny ring of points you made in step four.

   Shift-drag a box around them to mark them. Now scale them back up until you can see them clearly and separately in the Front window (you are working in the side window). This might be tricky. The drag-scaling can be quite sensitive and it might take you a few tries to achieve this. Just be sure you don't get the control points crossed. You are about to model the fish's mouth so approximate how large an opening you need.

7. In the front window, shift-drag a box around the highest and lowest of the marked points. Note that in the side view, this unmarks both highest and lowest points. With the SCL button still depressed scale the remaining four points in the side view so that their value in the z axis is very close but their value in the y axis remains the same like this:

   Click on translate and, in the Front view,drag the marked points back (-x) so that there is a deep 'V' shape for the fish's mouth. Click on 'Done'. Return to the main edit screen and admire your work. Save the file (F2). Name it FISH1.MDL.

8. Looking at the fish it appears that the tail end needs to be more extended to accomodate the caudal fin. You want to be careful not to change the control points at the wide end of the patch because that is where it joins the front end of the fish. Right now, the points are identical so the two halves will unite seamlessly. You only want to edit the points at the far left end of the patch object. Select the rear half of the fish (Body1) and return to the extended edit screen. Click on 'Unmark All'.

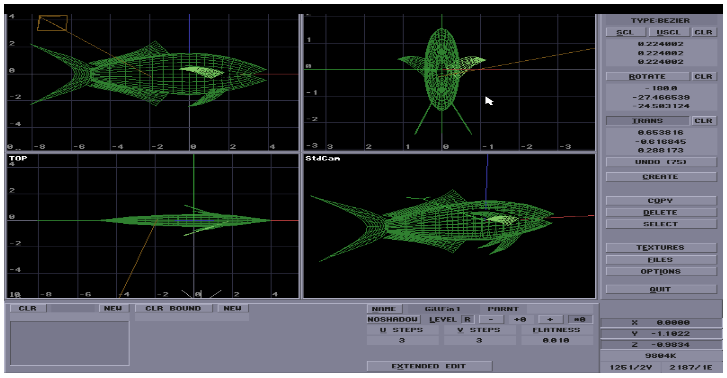
In the front view, shift-drag a box around the four points that are scaled to a point (step 4). Zoom into the side view until you can clearly see the individual points and click on 'local' scale. Drag the mouse in the side view to scale the points apart until you can see it in the front view. Scale it so that the z value increases but leave the y value small. You want the tail to be fairly flat to accomodate the flat caudal fin. Click on translate and move the marked points to x*-5. Click on 'Done' and return to the main edit screen. At this point you may have to adjust your camera and look_at to see all of the fish centered in the STD CAM screen. Save your work (F2).

9. Click on 'Create", 'Bezier Patch', and Name it 'Caudal_Fin'. Select 'Sheet', and 'OK, Create'. Go to 'Extended Edit' and click on 'Mark All'. Rotate the patch in the side view so that it appears on its side in the Std Cam view. Then, local scale it to about 3x3 units. Shift-drag a box in the front view to unmark all points except those on the extreme right side. Scale them down to about 1 unit wide. 'Unmark All'.

Shift-drag a box in the front view around all of the points on the extreme left end of the patch and scale them up to about 5 units wide. Unmark all. In the side view, shift-drag a box around the center of the patch. All the center points should be selected. Now, in the front view, shift-drag a box around any marked points on the far right of the patch to unmark them. Now, only the remaining six center points should be marked. Drag-scale them until they are very close to each other, but not crossed or touching. Click on translate and move them +x right up to but not beyond the right end of the patch. The patch should look like a fish fin now. Turn off the control mesh for a clearer look.

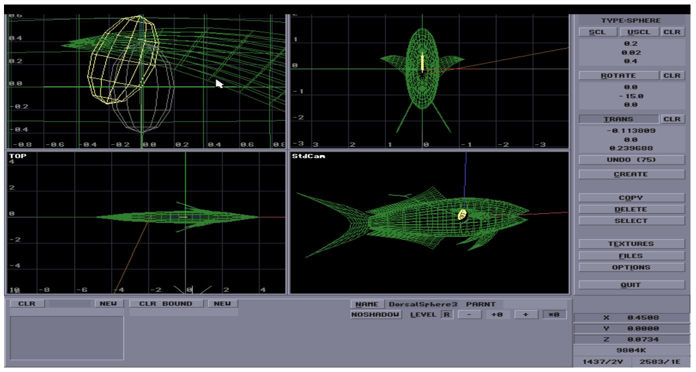
Click on 'Done' and return to the main edit screen. You will have to scale and translate the fin to get it to fit, but this should not be difficult to do. Once in place, save your work. Your fish is starting to take shape.

10. In a similar manner, create sheet beziers for all of

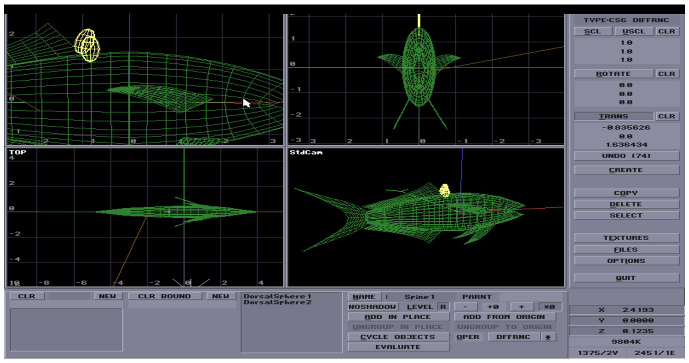


the other fins and place them around the fish. Fish fins come in pairs so make one, copy it, and them mirror it to make the other. Fins to create: dorsal and anal, right and left pectoral, right and left gill fins.

Scale and rotate them to place them properly. You might want to refer to a picture of a fish for proper placement of the fins. When you are done, your fish should now look something like this:

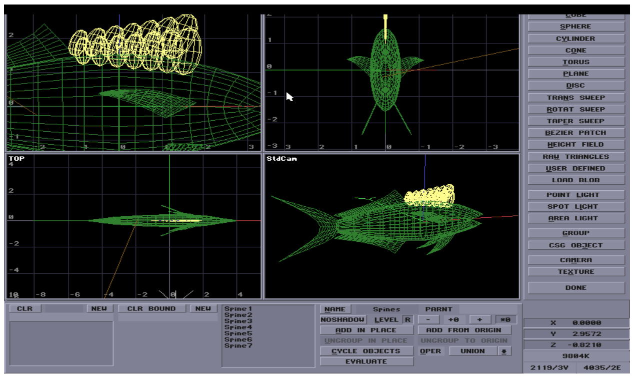


11. The trickiest fin is the spiny dorsal fin. Since you already have 9 patch objects, two of which are composed of 4 patches each for a total of 15 patches, I do not recommend attempting to build the spiny dorsal fin out of beziers. Instead we will make it out of spheres. First, create a sphere and call it 'DorsalSphere1'. Click on 'Done'. Scale it x*.2, y*.01, and z*.4. Zoom into the front view so you can see it clearly. Copy this sphere and rotate it y*-15. Click on 'OK' and zoom into the front view so you can see what you are doing. Change the scaling on this second sphere to y*.02. Translate it so that it obscures the upper left portion of the first sphere like this:



12. Now click on 'Create', 'CSG Object', and name it 'Spine1'. Click on 'Add in Place', and from the select screen click first on 'DorsalSphere1' and then on 'DorsalSphere2'. Right mouse click to return to the main edit screen. Click on the 'OPER' down arrow and select 'DFFRNC' (difference). Click on 'Done'.

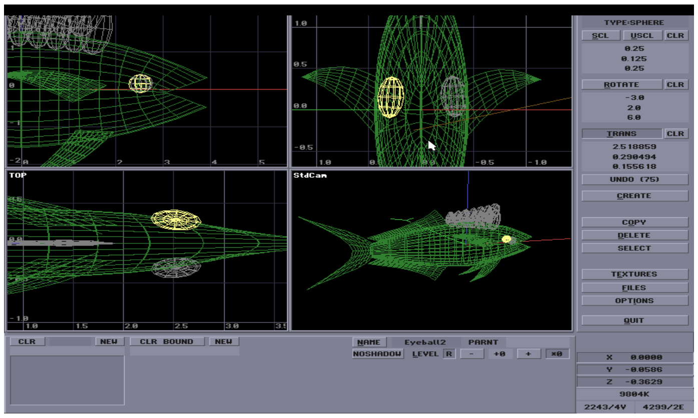
    Zoom back out of the front view until you can see the fish's back. Translate the Spine1 into place just in front of the dorsal fin. Remember that only the lower sphere will show up because this is a CSG difference. Now comes the tricky part. Save your work!



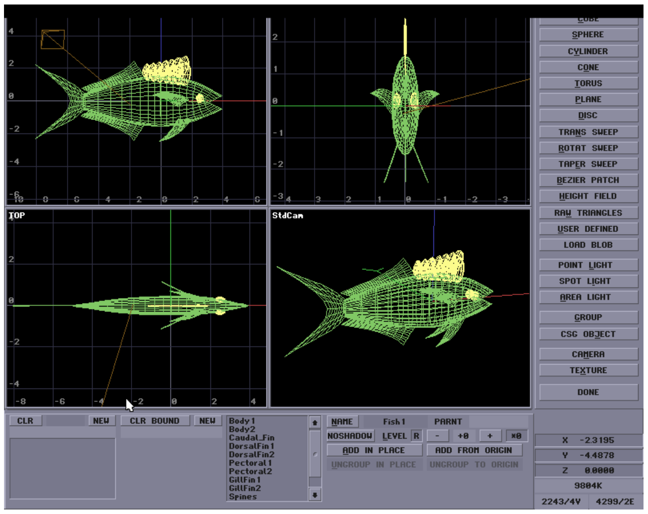
13. The spiny dorsal fin consists of a series of spines like the one we just created, so you must copy this one to make the rest. But the spines gradually increase in size as you go so you must introduce a scale value into the copy window to insure that the increase is consistent. Furthermore, the spines are about the same distance apart so a translate value must be added also.

    Finally, they rotate slightly to follow the curvature in the fish's back so a rotate value must be added as well. With the Spine1 selected, click on 'Copy', and enter these values in the copy window: SCL FCT <1.1, 1.1, 1.1>; ROT OFF y*-1; XLAT OFF x*.4; copies 6; reference 'yes'. Click on 'OK'. Six additional spines are created, gradually increasing in size, translating up the fish's back consistently, and rotating slightly to conform to the curvature in the fish's back.

    The final step is to union the spines. Click on 'Create', 'CSG Object', and name it 'Spines'. Click on 'Add in Place' and, from the select screen, click on each spine followed by a right click and it will be added to the Union. When you are done, right click to return to the main edit screen and save your work. (F2)

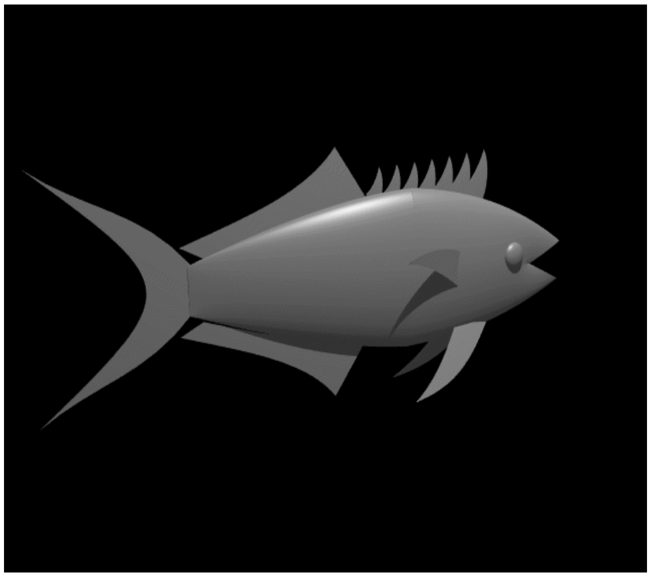


14. Our fish needs eyes. This is a comparatively easy thing to model. Make a sphere, name it 'Eyeball1', and translate it into position on the fish's head. Fish's eyes are not round, they are sort of like a flattened oblong shape, so first scale the eyeball USCL y*.2, then SCL the y value to .125. Rotate it so that it rests against the fish's head. Copy it and reverse the x and z rotation values and translate it in the top view to the other side of the fish's head.



15. Almost done. Click on 'Create', 'Group', and name it 'Fish1'. Click on 'Add in Place'. From the select screen, click first on 'Body1', then right click, then 'Body2' (right click), and in succession all the fins, spines and eyes. All of the parts should have been added to the object 'Fish1'. Right click to return to the main edit screen.

16. Let's view this guy to see how we did. But first, he needs a texture. We will give him a dummy texture for now, just so we can check out our modeling. Click on 'Textures', 'Create', and name it 'DummyTex1'. Pigment Type: Solid Color. Drag the red, green, and blue sliders to the right all the way to turn the sample box white. Accept all of the other default values. Click on 'Done'. In the main screen with the Fish1 object still selected, click on DummyTex1 in the texture box at the lower left of the edit screen. F2 to save. Click on 'Done', then 'Options'. Select 320x240 resolution. Click on 'Done'. Are you ready? Press F9.



17. Hey, that's not half bad! It will probably need some tweaking. It should be fairly easy to reposition any slightly errant elements. Look closely at the bezier patches. If they appear to be faceted or show gaps, simply select that patch object and change u steps and v steps from 3 to 5 and flatness from .01 to 0. (Keep in mind that this will greatly increase memory requirements, parsing and tracing time.) If by any chance you need to make any changes to the spines, changing Spine1 will change them all since they are linked. When you have your fish just the way you want it you are ready to begin texturing it. We are not going to go into texturing in this article. That is something I think I will save for the next issue. But until then feel free to use MORAY's powerful new texture editor to create textures for the various parts of your fish. I will show you some cool image-mapping tricks in my next article. Save your fish! You will need it for the next tutorial! <G> ❑

Tips & Tricks

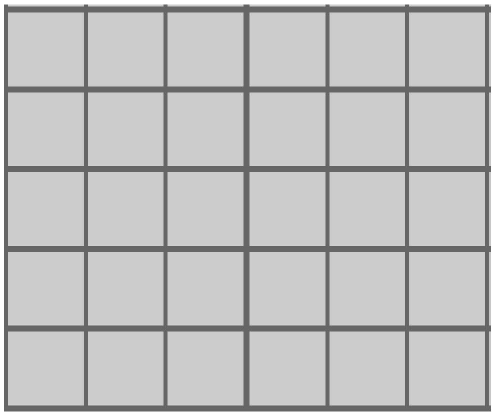
Q: Does anyone know how I can create bathroom tiles in POV-ray?! What I mean is a small ,slightly reflective square surrounded by a thin white line. Then repeated over a plane or box to give the effect of a tiled wall. I'm sure I've seen it in a scene, but can't work it out! Any ideas appreciated.

A: In POV-Ray V2.2 one of the easiest ways to make a convincing tile surface it to create a heightfield using POV-Ray of a tiles surface. The heightfield is then used in a POV-Ray scene.

The first step is to create a tga file that contains a depth representation of a tile surface. This is done by overlaying a vertical and horizontal line gradients over a white surface. The following POV-Ray source code creates a tile heightfield depth image.

```
#include "colors.inc"

#declare TileTex =
texture {
  pigment {color White}
   finish {
     phong 0.2
     phong_size 50
     ambient .8
     crand .05
   }
}
texture {
   pigment {
    gradient x
    color_map {
      [0.025 color White]
      [0.05, 0.5 color Gray30 filter 1
            color Gray30 filter 1]
      [0.525, 0.975 color Gray90 filter 1
            color Gray90 filter 1]
      [0.975 1.0 color White color White]
    }
   }
  finish {
    phong 0.0
    phong_size 50
    ambient .4
    crand .05
  }
}
texture {
   pigment {
    gradient z
    color_map {
      [0.025 color White]
      [0.05, 0.5 color Gray30 filter 1
            color Gray30 filter 1]
      [0.525, 0.975 color Gray90 filter 1
            color Gray90 filter 1]
      [0.975 1.0 color White color White]
    }
   }
  finish {
    phong 0.0
    phong_size 50
    ambient .4
    crand .05
  }
}

camera {
  location <0, 15, 0.000001>
  right <1.333, 0, 0>
  look_at <0, 0, 0>
}

plane {
        y, 0
        texture {TileTex}
}
```

When this POV-Ray scene is rendered, the following images is produced.

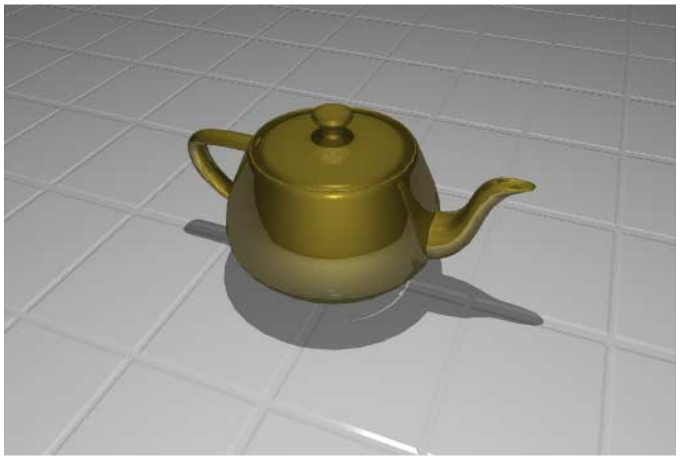


This image is used in a scene as a heightfield. The heightfield command in POV-Ray converts the color indexes in an image into a 3D object. This following POV-Ray scene uses the tile heightfield as part of the scene.

```
#include "colors.inc"
#include "textures.inc"
```

```
#include "teapot.inc"

camera {
      location <1, 2.5, -2.5>
      right <640/480, 0, 0>
      look_at <0, 0.5, 0>
}

light_source {
      <1, 2.5, -2.5>
      color rgb <0.75, 0.75, 0.75>
}
light_source {
      <0, 5, 0>
      color rgb <0.5, 0.5, 0.5>
}
height_field  {
      tga "tile.tga"
      water_level 0.01
      texture {pigment{White}
         finish{Shiny}}
      scale <15, .01, 15>
      translate <-7.5, 0, -7.5>
      //rotate <-10, 0, 0>
}

object{Teapot translate <0, 0.6, 0>}
```

The resulting image shows a teapot on a tiles surface. This may seem like a lot of steps, but it is simpler than modeling the individual tile element and then repeating that element across a plane.



❑

POV Animation Using Perl

by Gavin Smyth

Introduction

The “standard” way of generating animations using POV is to produce a number of POV source files, one for each frame; render those, and then combine them into a large AVI or FLI file. The generation of the POV source files is done by hand, which is very tedious, or by writing a program to do it, for example in Basic or C. Somewhere in between these two extremes are useful tools such as ANIMDAT (which can be found on `ftp.povray.org`). These have the notion of a template file containing references to variables which are automatically updated per frame: however, I found that the calculations ANIMDAT is capable of performing were too limited.

In an animation I was working on recently, one of the parameters was derived from a messy equation which had no analytic solution. I had to use numerical approximation (Newton-Raphson root finding if you really want to know) and it was not possible to do this within ANIMDAT. Being rather lazy, I did not fancy the idea of writing a Basic or C program and coincidentally, round about the same time, I came across something called perl on a UNIX machine. I’ll explain a bit about perl in the next section, and then show how I used it for generating POV scripts.

The good news is that perl is also available to run under DOS (and in fact, I’ve done more of my perl/POV stuff under DOS than UNIX). I’ll list a few locations later.

OK: what is perl?

Perl is yet another one of those very useful languages that UNIX hackers (in the nicest possible sense of the word) periodically come up with. It combines the best features of awk, sed and shell scripts and, although it initially looks cryptic, it is actually very easy to use. There are two versions commonly in use: version 4 is very stable and has been around for a while, mainly because the author (Larry Wall) was working on a slicker and more sophisticated version 5. There are some incompatibilities between the two versions, but the very small subset of the language I’ve used here causes no problems.

If you’ve used UNIX and its common utilities, perl will look very familiar. The only significant and not immediately obvious features are:

- all “simple” variables always have their names prefixed with $, (other variable types, such as arrays use different prefix characters);
- variables in strings inside double quotes are evaluated whereas those in single quoted strings are not;
- in “if” statements, etc., curly braces even when there is only a single statement in the “then” or “else” part;
- to put dollar signs, quotation marks or backslashes inside strings, one has to “escape” them by putting a backslash first; and
- comments begin with a #.

The animation script

The idea behind my perl script is a loop which calculates values for some variables, then printing out a string containing references to those variables to a POV file per frame. As well at this, it writes lines to a DOS batch file to call POV on those files. (I could have made perl call POV directly, but doing it this way leaves more DOS memory for POV, and gives me a chance to examine the files before running POV.) Anyway, here’s the script—the numbers are not part of the script but reference the notes following the code:

```
      # POV animation generator script.
      #
      # created 27.07.95                                          SGS
      #
      # Usage: rename to <template file stem>.pl, fill in the (POV) template
      # string and calculation, and then call as -
      #
      #          perl <template file stem>.pl <number of scenes>
      #
      # Note that the template file stem must be short enough to form output
      # files with that and an underscore and scene number appended, along
      # with the suffix ".pov". The program also produces a .bat file which
      # can be used to run POV on all the files.

1     $#ARGV == 0 || die( "Usage: perl $0 <num scenes>, stopped" );

2     $namePrefix = substr( $0, $[, index( $0, "." ) );
      print( "Building with stem $namePrefix\n" );

3     open( BAT, ">$namePrefix.bat" );

4     # Some useful constants
      $pi = 3.1415926;
      $deg2rad = $pi / 180;
      $numScenes = $ARGV[ 0 ];

      for( $scene = 0; $scene < $numScenes; $scene++ )
      {
        print( "Producing scene $scene\n" );
5       $fraction = $scene / $numScenes;

6     HERE'S WHERE THE CALCULATION GOES...

7       $fileRoot = sprintf( "%.4s_%03d", $namePrefix, $scene );
        open( SCRIPT, ">$fileRoot.pov" );
        print( SCRIPT "
8     HERE'S WHERE THE TEMPLATE GOES...
      " );
        close( SCRIPT );
9       print( BAT "c:\\pov2\\povray +Lc:\\pov2\\include" );
        print( BAT " -w160 -h100 +ft -d -i$fileRoot.pov -o$fileRoot.tga\n" );
      };

10    print( BAT "c:\\makefli\\makefli $namePrefix\n" );
      print( BAT "del *.tga\n" );
      print( BAT "del *.pov\n" );
      close( BAT );
```

As you can see, very little code is needed to produce an animation generator framework. All the hard work has been done by the perl authors!

Notes:

1 The script is used by filling in the calculation and template and then: `perl script numFrames`. This first line simply traps the wrong number of arguments passed in and prints out an error message.

2 The POV files are called `name_XXX.pov`, where `XXX` is a three digit number, and `name` is taken from the name of the script itself. Perl scripts are conventionally named `something.pl`, so this line strips off the suffix. Note that, because of the DOS file name limitations, the prefix should be four or fewer characters.

3 The batch file containing the POV commands is called `name.bat`.

4 I place a few useful constants here. You can put in anything you need, of course.

5 `$fraction` ranges from 0 for the first frame to (almost) 1 for the last.

6 This bit should be filled in with the details of the calculation.

7 Here the POV filename is created as described above. The `%.4s` limits the names step to the first four characters, and the `%03d` ensures that the number is always printed as three digits.

8 The double quoted string contains the POV template, with `$` variables corresponding to those set up in the calculation. Note that quotation marks, etc. in this string must be escaped. The string can span multiple lines, and perl will retain the ends of lines.

9 These two lines print out the command to run POV. Note that the print command does not introduce an extra new line by itself, so I was able to split a long line into two shorter ones. The file locations are specific to my system: you'll have to change them for wherever you keep the POV files.

10 These are a few cleanup lines: the first combines the TGA files into a FLI animation (you could use DTA, TGAFLI or whatever for this) and the others delete the working files.

When the script is run, you'll have a bundle of POV files and a batch file to run POV on them. Incidentally, I decided not to make the perl script read separate template and calculation files. There's so little "animation independent" perl required that I just copy it around with the animation specific details.

A simple example

This is a very simple example to give you an idea of how the thing works. A more complex and interesting script would take up too much space! OK, the script `ball.pl` contains the script above with the following as the calculation section:

```
  $t = $fraction;
  $x = $t * 2;
  if( $t < 0.5 )
  {
    $y = 1 - 4 * $t * $t;
  }
  else
  {
    $y = 0.75*(1-4*(1-$t)*(1-$t));
  }
```

This is calculating the equation of a ball dropping under "gravitational" influence, hitting a surface (at time 0.5) and bouncing back up.

`ball.pl` also contains the following template definition:

```
  print( SCRIPT "
#include \"colors.inc\"

sphere { <$x, $y, 0>, 0.2
      texture {
        pigment { color Blue } } }

camera { location <0.5, 0, -3>
         look_at <0.5, 0, 0> }

light_source { <0, 0, -60>
               color White }
" );
```

Things to note are the escaping of the quotes for the include line, the use of the `x` and `y` variables, and the final newline in the string to ensure the batch file ends with a newline.

I could run this with, for example:

```
      perl ball.pl 10
```

to generate 10 POV files—here are the first three, `ball_000.pov`, `ball_001.pov` and `ball_002.pov:`

```
#include "colors.inc"

sphere { <0, 0, 0>, 0.2
      texture {
        pigment { color Blue } } }

camera { location <0.5, 0, -3>
         look_at <0.5, 0, 0> }

light_source { <0, 0, -60>
               color White }

#include "colors.inc"

sphere { <0.2, 0.96, 0>, 0.2
      texture {
        pigment { color Blue } } }

camera { location <0.5, 0, -3>
```

```
         look_at <0.5, 0, 0> }

light_source { <0, 0, -60>
               color White }

#include "colors.inc"

sphere { <0.4, 0.84, 0>, 0.2
      texture {
        pigment { color Blue } } }

camera { location <0.5, 0, -3>
         look_at <0.5, 0, 0> }

light_source { <0, 0, -60>
               color White }
```

You can see how the variables are replaced appropriately.

As well as these, I also have a `ball.bat` file which contains the lines to run POV on my system. I won't bother showing it here since it should be obvious!

A couple of tips

In a more realistic example, I would have most of the POV work done in a separate include file and have a small template which sets up some `#declare` lines and then includes the other file—for example, something like:

```
  print( SCRIPT "
#declare ankleAngle = $angle
#declare footPos = < $x, $y, $z >
#include \"leg.inc\"
" );
```

This keeps down the size of the perl script!
When I started on the project which led to this work, I made a lot of use of the POV transformations to move objects around. One of the things I had to calculate was the location of a point on the surface of one of the objects. Unfortunately, I could not get the information directly from POV, so I had to repeat the same transformations within the perl script. Since I was repeating the same calculations in two places, I was a bit worried that the accuracy of the values would differ and I'd be left with gaps or overlaps, but none were noticeable. However, I think in future, I'll use POV's transformations for object distortion and placing static objects, and move the rest around in perl.

Where to find perl

As promised, here are some pointers to FTP sites for perl.

The (unix) sources can be found with GNU stuff, such as at `prep.ai.mit.edu` in `/pub/gnu`, but we're more interested in MS-DOS variants:

Within Simtel (`oak.oakland.edu` or one of its many mirrors), in `msdos/perl`, you'll find `bperl4x.zip`—this is a fairly stable version (4.036), and the one I have used most often. There are beta versions of perl 5 in a few locations, such as: `ftp.einet.net` in `/pub/perl5` or `ftp.ee.umanitoba.ca` in `/pub/msdos/perl/perl5`. I have tried neither of these, but have confirmed that they exist (I think they are the same).

There are variants for lots of other platforms too, and the best place to look is the perl "frequently asked questions:" see the usenet archives or the file `pub/perl/doc/FAQ` on `ftp.cis.ufl.edu`.

You'll find some perl reference documentation in the same places, or have a look at the usenet group `comp.lang.perl`. There are also a few perl books, though the current crop deals mainly with perl 4. A couple of examples are: Programming Perl, by Larry Wall and Randal Schwartz; or Learning Perl, by Randal Schwartz. ❑

Creating Animated GIF Images - Part 1

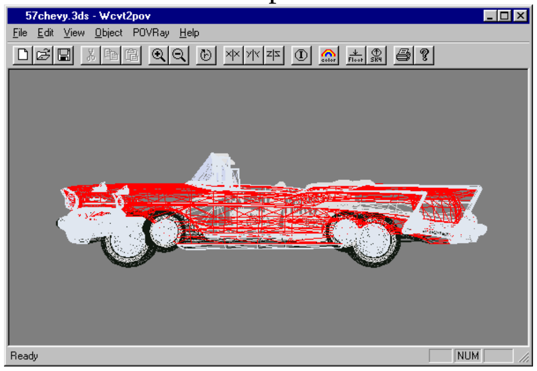
by Keith Rule

Getting Started

I'm sure you've run across those nifty animated GIF images on web pages on the Internet. Now days, it's difficult to find a web page that doesn't have at least one of these images on it.

It turns out that with the help of POVRay and a shareware tool called the "GIF Construction Set" you can turn out these images with only a modest amount of effort.

The first step is to figure out something cool to animate. For this article I will rotate a 57Chevy model. This is one of my favorite models from the Avalon web site (you can find Avalon at http://www.viewpoint.com). To create the initial POVRay scene I simply read the 57chevy.3ds file I downloaded from Avalon into wcvt2pov. I then add the colors I want for each part of the car.



Wireframe view of Chevy.3ds

After the coloring is added I save the scene as a POVRay file. The resulting file is very large (3.84 Megs). To rotate the car round the Y-Axis (the up direction) I need to add the rotate instructions to the end of the car model. The last few lines of the car model something look like this:

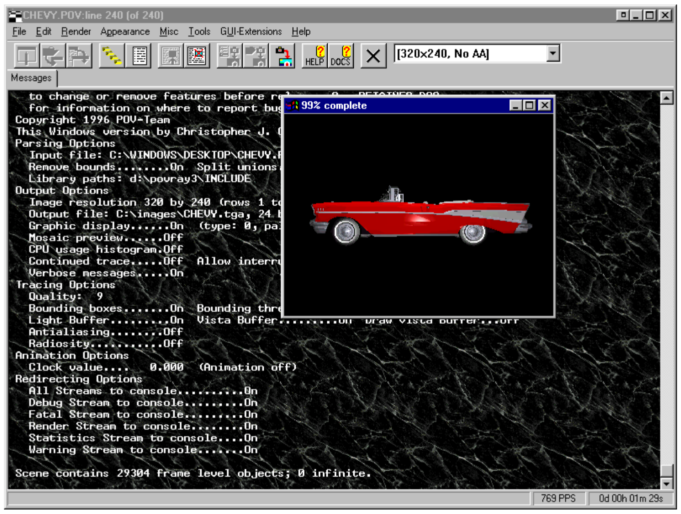
```
          .
          .
          .
      triangle {
        <-5.395, -1.284, -2.849>,
        <-5.39, -1.283, -2.838>,
        <-5.39, -1.286, -2.838>
      }
      texture{_rfrim}
    }
  }
}
```

Add the bolded line to the model.

```
          .
          .
          .
      triangle {
        <-5.395, -1.284, -2.849>,
        <-5.39, -1.283, -2.838>,
        <-5.39, -1.286, -2.838>
      }
      texture{_rfrim}
    }
  }
  rotate <0, 360*clock, 0>
}
```

This line will cause the car to rotate around the Y-Axis (which is the up direction for Wcvt2pov emitted scenes).

This image, when rendered by POV-Ray, looks like this:


CHEVY.POV:line 240 (of 240)
File Edit Render Appearance Misc Tools GUI-Extensions Help
HELP
DOCS
[320x240, No AA]
Messages
99% complete
769 PPS
0d 00h 01m 29s

```
  to change or remove features before re
  for information on where to report bug
Copyright 1996 POV-Team
This Windows version by Christopher J. C
Parsing Options
  Input file: C:\WINDOWS\DESKTOP\CHEVY.P
  Remove bounds........On  Split unions
  Library paths: d:\povray3\INCLUDE
Output Options
  Image resolution 320 by 240 (rows 1 to
  Output file: C:\images\CHEVY.tga, 24 b
  Graphic display......On  (type: 0, pal
  Mosaic preview......Off
  CPU usage histogram.Off
  Continued trace.....Off  Allow interru
  Verbose messages.....On
Tracing Options
  Quality:  9
  Bounding boxes.......On  Bounding thre
  Light Buffer.........On  Vista Buffer.........On  Draw Vista Buffer...Off
  Antialiasing........Off
  Radiosity...........Off
Animation Options
  Clock value....   0.000  (Animation off)
Redirecting Options
  All Streams to console..........On
  Debug Stream to console.........On
  Fatal Stream to console.........On
  Render Stream to console........On
  Statistics Stream to console....On
  Warning Stream to console.......On

Scene contains 29304 frame level objects; 0 infinite.
```

Chevy.pov rendered in POV-Ray V3.0 for Windows

Creating The Animation Frames

Now that we have a viable scene, it's time to insert the action.

In the past we need to construct a program, or batch file to create the POV-Ray command line for each frame of the animation. This was a tedious operation (though much less tedious than invoking POVRay by hand for each frame). However, in the Windows version of the POV-Ray V3.0 beta there is a much easier way to generate each image in an animated scene. This can be done by using the +KFI*n* and +KFF*n* command line flags. These flags set the initial frame number and the last frame number in an animation sequence. The +KFI*n* flag sets the initial frame. If the initial frame was frame zero the value of this flag would be +KFI0. The +KFF*n* flag sets the frame number of the final frame in the sequence. If the final frame was forty, the flag would be +KFI40.

To specify this animation sequence to POV-Ray you select the menu selection "Render|Edit Settings/Render…" option. This brings up the Edit Settings dialog. Set the command line options to: "+KFI0 +KFF40"

When you start rendering the scene, it will start with frame 0 and finish at frame 40. The clock variable will be set to (current frame number)/40 or the values 0, 1/40, 2/40, …, 40/40. Each of the frames will be rendered and placed in a separate file. Since the file read in and rendered was name "chevy.pov" in this example, each of the rendered files will be named "chevy*n*.pov" where *n* is the frame number. The files available when rendering has completed will be "chevy00.tga", "chevy01.tga", …, "chevy40.tga".

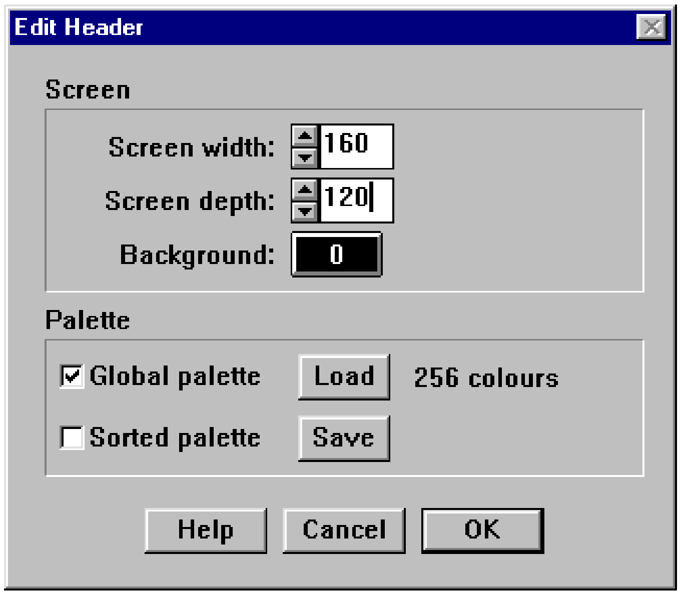
At this point we have all the frames necessary to create our animation. The only step left is to create the GIF animation.

Creating The GIF Animation File

Creating a GIF animation file is straightforward using the "GIF Construction Set". The "GIF Construction Set" (which will be referred to as GIFCS from here on) is a shareware program by Alchemy Mindworks. It is absolutely indispensable for creating GIF animation files.

Step 1: Starting The Project

Execute GIFCS and select "File|New". By default, GIFCS will create a 640x480 image header. Select that entry, and double click. This will bring up an edit dialog. Change the screen height and width to the size of your animation files. In this case the height and width is 160 x 120.

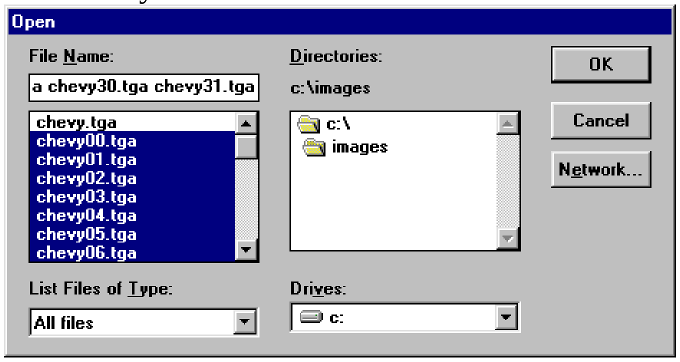


Screen width: 160
Screen depth: 120
Background: 0
[✓] Global palette
256 colours
[ ] Sorted palette

The GIF header edit dialog

Step 2: Adding The Animation Files

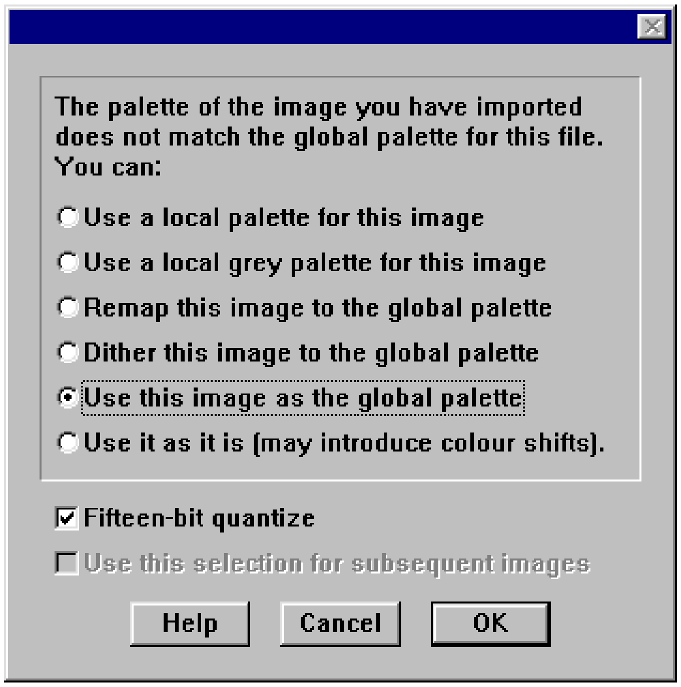
Select the Insert button from the toolbar. This will bring up a selection palette. The first item in this palette is image. Select that item. This will bring up a file dialog box. Select all the files you wish to insert into your animation.



File Name: a chevy30.tga chevy31.tga
chevy.tga
chevy00.tga
chevy01.tga
chevy02.tga
chevy03.tga
chevy04.tga
chevy05.tga
chevy06.tga
Directories: c:\images
c:\
images
List Files of Type: All files
Drives: c:

Step 3: Creating the Global Palette

After you've selected OK on the file dialog box, a dialog will appear that will give you several options for dealing with the palette for the first frame in the animation. Select "use this image as the global palette" and OK.



The palette of the image you have imported does not match the global palette for this file. You can:

- Use a local palette for this image
- Use a local grey palette for this image
- Remap this image to the global palette
- Dither this image to the global palette
- [✓ Use this image as the global palette]
- Use it as it is (may introduce colour shifts).

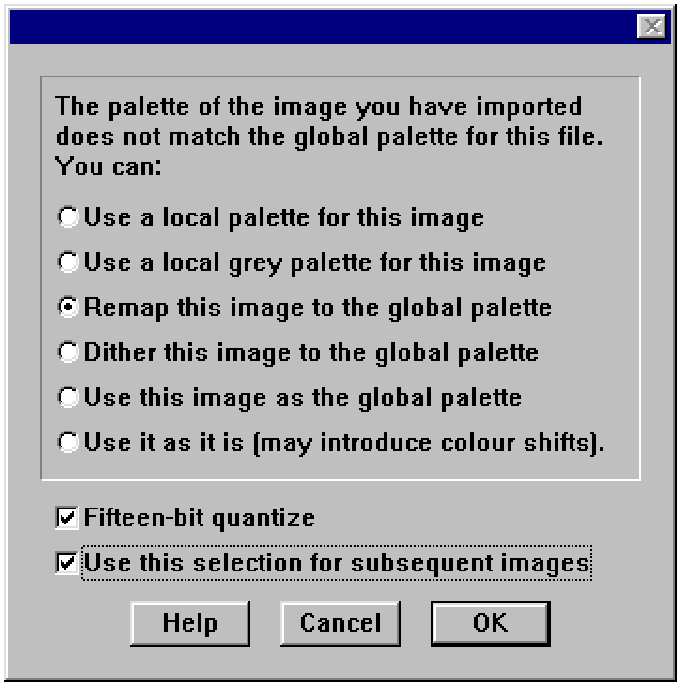
[✓ Fifteen-bit quantize]

Use this selection for subsequent images

The dialog used to create the global color palette from the first animation frame.

Step 4: Converting the rest of the frames

After the palette has been converted for the first frame and that frame is loaded into the animation file, it is time to load the rest of the images.



The palette of the image you have imported does not match the global palette for this file. You can:

- Use a local palette for this image
- Use a local grey palette for this image
- [✓ Remap this image to the global palette]
- Dither this image to the global palette
- Use this image as the global palette
- Use it as it is (may introduce colour shifts).

[✓ Fifteen-bit quantize]

[✓ Use this selection for subsequent images]

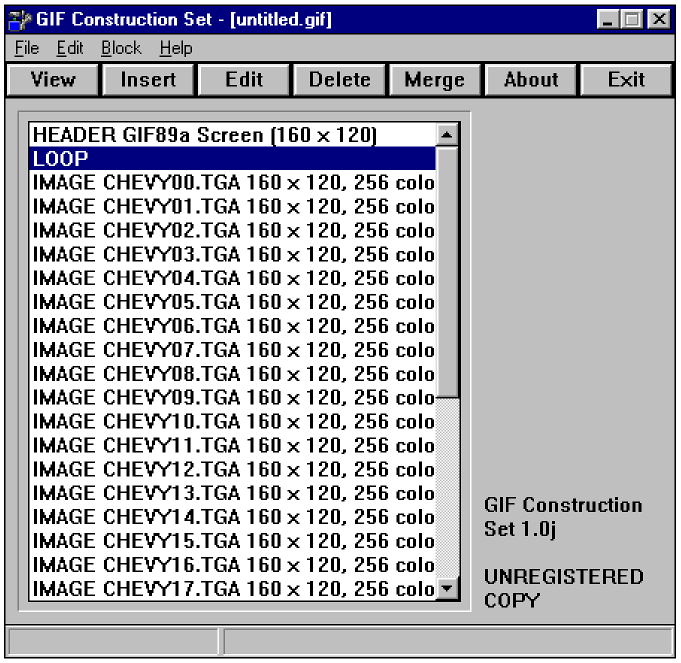
The dialog used to create the color palette from the rest of the animation frames.

Since all of the subsequent files have already been selected, all that is necessary to remap these images to the global palette and insert these files into the animation is to select the "Remap this image to the global palette" item in the conversion dialog.
After this selection has been made, an OK has been selected, all the remaining selected files will be converted. Please note that this conversion may take along time. Be patient.

Step 5: Adding Looping and Timing

All that's left in creating the GIF animation is adding a loop at the top of the image and the time interval between each frame.

To add a loop, select the header (the first item in the list), then select the Insert item in the toolbar. Choose Loop from the object palette.



GIF Construction Set - [untitled.gif]
File Edit Block Help
View Insert Edit Delete Merge About Exit

```
HEADER GIF89a Screen (160 x 120)
LOOP
IMAGE CHEVY00.TGA 160 x 120, 256 colo
IMAGE CHEVY01.TGA 160 x 120, 256 colo
IMAGE CHEVY02.TGA 160 x 120, 256 colo
IMAGE CHEVY03.TGA 160 x 120, 256 colo
IMAGE CHEVY04.TGA 160 x 120, 256 colo
IMAGE CHEVY05.TGA 160 x 120, 256 colo
IMAGE CHEVY06.TGA 160 x 120, 256 colo
IMAGE CHEVY07.TGA 160 x 120, 256 colo
IMAGE CHEVY08.TGA 160 x 120, 256 colo
IMAGE CHEVY09.TGA 160 x 120, 256 colo
IMAGE CHEVY10.TGA 160 x 120, 256 colo
IMAGE CHEVY11.TGA 160 x 120, 256 colo
IMAGE CHEVY12.TGA 160 x 120, 256 colo
IMAGE CHEVY13.TGA 160 x 120, 256 colo
IMAGE CHEVY14.TGA 160 x 120, 256 colo
IMAGE CHEVY15.TGA 160 x 120, 256 colo
IMAGE CHEVY16.TGA 160 x 120, 256 colo
IMAGE CHEVY17.TGA 160 x 120, 256 colo
```

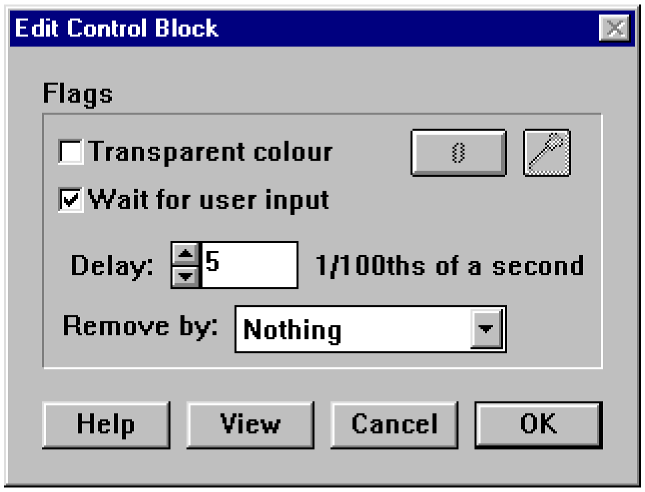
GIF Construction Set 1.0j

UNREGISTERED COPY

A loop control added to the GIF file.

Next, select each image and insert a control. To edit the control either double click the control or select edit from the toolbar.

I usually select a time delay of 5 for animations with 20 or more images. Experiment with the timing to see what works for your animation.


Edit Control Block

Flags

☐ Transparent colour

☑ Wait for user input

Delay: 5 1/100ths of a second

Remove by: Nothing

The Edit Control Dialog

Step 6: Viewing the Result.

To view the result, select the View item form the toolbar. This will play the animation (use the ESC key to return to the GIFCS program). When the animation looks good to you, save the GIF file and you have created a animated GIF file suitable for your web page.

GIF Animation Tips

- *Keep the images small*. The smaller the images the smaller the finished animation file. Even this example at 160 x 120 is probably a little large for a GIF animation.
- *Fewer frames are better*. At forty frames the example in this article is probably on the upper limit of what you should consider reasonable for a GIF animation.
- *Take advantage of symmetry*. One way to reduce the number of frames is to take advantage of the symmetry of rotating objects. For example if you are rotating a cube, you probably only need to generate the frames for 90 degrees of the rotation. Just repeating that will give the appearance of the cube rotating the full 360 degrees.

Conclusion

Though I'd never say this process is trivial, it is straightforward and the results speak for themselves.

In the next issue I will show how to animate a slightly more complicated image to produce a GIF animation that is immediately useful on your web page. ❑