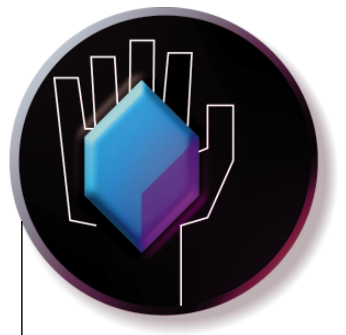


Rays of light

POVRay is a lot of raytracing software downloadable in a 4Mb website file, and despite its heavy maths and programming bias, Benjamin Woolley got to grips with it.

As someone once said, “There is no such thing as a free lunch”. But there *is* such a thing as a free raytracing 3D graphics package. POVRay (Persistence Of Vision Raytracer) is among the most sophisticated around and, as I write this column, the Beta test phase of version 3 is drawing to a close. It is available in all flavours including DOS, Mac, Amiga, Unix, Linux and, as tested here, Windows. By the time you read this, the final version should be available, and I urge you to download it from the official POVRay website at www.povray.org or CompuServe’s GRAPHDEV forum.

The self-installing executable you get is over 4Mb, but this includes documentation and a generous helping of sample files and some fairly substantial binaries. It is quite remarkable that you can get such a lot of software in a file of this size, given that a commercial 3D package would come on a CD-ROM and colonise half your hard disk.

Installation is no more than a double-click on the downloaded file and a few simple answers to a few simple questions. By the time the disk-thrashing is over, you should find POVRay for Windows seamlessly settled into your Windows 3.11, 95 or NT system (it is a full 32-bit application) and ready to run.

I have to admit that the first time I used it, my feeling was one of disappointment. With POVRay, you re-enter a world that many of us had hoped to leave behind: the world of programming, command-line interpreters, declarations, variables and, (ugh!) maths. What follows is a sample taken from a tutorial in the help file for creating a colour gradient:

If you want to start one of the colours at a specific angle, you’ll first have to convert the angle to a colour map index. This is done by using the formula

```
color_map_index = (1 - cos(angle))
/ 2
```

where the angle is measured against the negated earth’s surface normal.

I have not personally encountered a cosine since fifth form, and can’t remember what one is, except that it has something to do with angles.

However, the mathematically timid should persist because, as I soon began to discover, the wonder of POVRay is that even without maths or a fondness for programming languages you can achieve a great deal.

Unlike commercial 3D graphics packages, POVRay is just a renderer. It does not include a modeller. This means that it takes scene descriptions, essentially 3D graphics programs, and turns them into rendered images. To use it, you have to write these scene descriptions yourself or use a program that will generate them for you.

I began by trying to write a few scenes for myself. There is a series of tutorials in the help file which helps you start coding, and you will find that you are soon able to create simple objects. The best way to proceed is to copy the lines of code supplied in the tutorial and paste them into a text editor. You can save the text as a file with the .pov extension, start up the POVRay renderer, and watch the scene emerge before your very eyes. If you have made a syntax error, POVRay reports which line caused the problem. By adjusting a few parameters here and there and re-rendering the scene, you can begin to get a feel for their effect.

An example of just about the simplest scene description file you can get is given in Fig 1. The “include” statement at the beginning merges the commands contained in the “colors.inc” file into the scene description. “Include” files can contain any legitimate POVRay statement, but are typically used to contain data, such as the definitions of colours, shapes and textures. They define Cyan as having the red/green/blue values 0, 1, 1 which means no red, full green and full blue. The rest of the program does as indicated in the comments prefixed with //.

Fig 1

```
#include “colors.inc”

// First create a background colour
  background { color Cyan }

//add a camera at position 0 units along the x
//axis, 2 units along the y axis, and -3 units
//along the z axis.  It points towards another
//point at co-ordinates 0,1,2, the position of the
//sphere
  camera {
    location <0, 2, -3>
    look_at  <0, 1,  2>
  }

//create a sphere, two units in diameter and
//colour it yellow
  sphere {
    <0, 1, 2>, 2
    texture {
      pigment { color Yellow }  //
    }
  }

//add a white light
  light_source { <2, 4, -3> color White}
```

Listing for a simple POVRay scene description

POVRay has an incredibly powerful scene description language that allows you to create 3D fractals, superquadric ellipsoids (which are objects with soft edges), halo effects, layered fog, and dust clouds. It creates more than those packages costing thousands of pounds. In its atmospheric capabilities, for example, it is ahead of 3D Studio Release 4. To exploit such features to the full, you should probably spend some time poring over the documentation, probably buy a book or two (for example, *Ray Tracing Worlds with POVRay* by Alexander Enzmann, Lutz Kretzschmar and Chris Young), and get the CD-ROM.

For the lazy ones among us, there is an easier way. You get an existing file and mess around with it. This is what I did to produce the image in Fig 2, which was rendered using POVRay version 3. I created it by adapting a 3D Studio file, which meant I could use 3D Studio’s modeller to work on the geometry. There are POVRay modellers available as shareware, the best-known of which is Moray. It is not the most wonderful piece of software and if anyone knows of anything better that is free or cheap, drop me a line. Meanwhile, I shall continue to look around for myself.

I converted the 3DS files generated by 3D Studio using a lovely freeware program called 3DS2POV, by Steve Anger and Jeff Bowermaster, downloaded from the CompuServe GRAPHDEV forum. Using a text editor, I adapted the resulting .pov file by borrowing little bits of extra code from the tutorials, to create the clouds in the background. I spent nearly all my time with POVRay using this jackdaw strategy, taking existing bits of code, playing around with them and rendering up the result to see what sort of mess I had made.

Since POVRay is freeware, widely distributed and designed to run on just about every type of computer you can think of, short of Babbage’s Analytical Engine, there are endless samples you can use and abuse in this manner. All samples are generously donated by their authors and widely posted across the internet and on various online services, mostly on CompuServe.

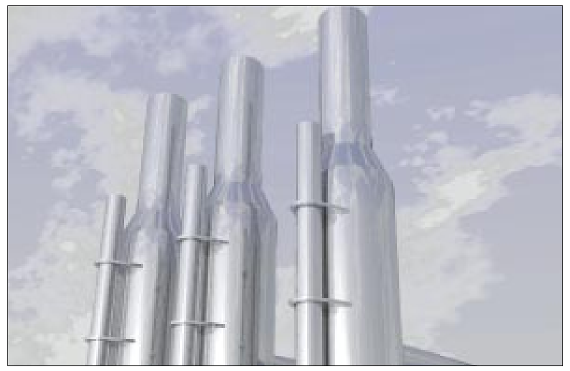


Fig 2 Scene rendered using POVRay. The reflections show some of the advantages of raytracing over faster but cruder scanline renderers

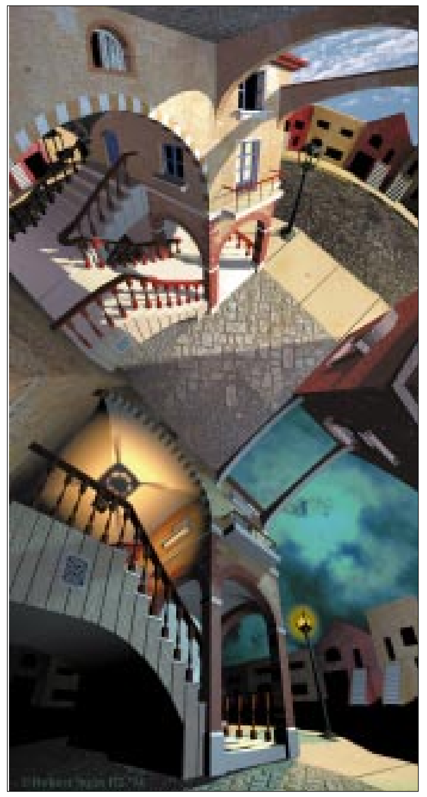
Night and Day

The image pictured right first appeared in *3D Artist*, the American magazine for 3D graphics users. I found it on the magazine’s web site, which is well worth a visit: www.3dartist.com.

The picture may be familiar to some of you. It is by Maurits Escher, the Dutch artist who for many expresses the weirdness and beauty of the information age with his mind-boggling images of infinite loops and distorted perspectives. People interested in the origins of his association with computing should look at Douglas Hofstadter’s wonderful book, *Godel, Escher and Bach*.

Escher’s best-known picture is probably “Ascending and Descending” (1960), which shows little elfin figures trooping up and down a staircase, the bottom of which impossibly connects to the top. Less well known is “High and Low”. Well, you see a version of it here, renamed “Night and Day”. It was created by Richard Stein III, a 3D artist who worked on the 7th Guest and 11th Hour games for Trilobyte, and who kindly gave me permission to reprint the picture.

Stein generated the image using 3D Studio. He points out, “An X-Y-Z-based program doesn’t have enough perspective points for us to build this type of image accurately. Morph software won’t do it. Stretching the camera lens beyond fish-eye won’t do it. But bending the objects in a specific way just might fake it.” And this is exactly what he has done, bending the objects away from the centre so that the illusion of Escher’s original picture is reproduced. The result is a fake in the sense that the illusion would quickly be lost if you tried to create an animation that moved through the scene. Because it was generated from a 3D model, Stein could render the scene as a “stereo pair”: two pictures showing the same scene from slightly different perspectives, thus giving the image real depth. I spent ages staring at the pair on my monitor, and succeeded in getting flashes of the stereoscopic effect. Escher would have loved it.



At the time of writing, Robert had put a range of his stereo pairs on the web. They are splendid, and you may still find them at www.tbyte.com/people/stein/stereo.htm. I have also reprinted the picture because, to me, it provides an object lesson in the effective use of materials: look at the floor in the centre of the image; the sheen of the stone is perfect. The walls have a rich, almost tactile texture to them. It just goes to show how wrong people are in thinking that computer-generated art is plastic-looking.

• PCW Contacts

Benjamin Woolley, writer and broadcaster, can be contacted at **woolley@illumin.co.uk**. His home page is **www.illumin.co.uk/woolley/**