Raster3D

SYNOPSIS

The Raster3D molecular graphics package consists of a core program ***render*** and a number of ancillary programs (***balls, normal3d, rastep, rods, ribbon***) that read atomic coordinates from Protein Data Bank (PDB) files to produce scene descriptions for input to ***render***. Raster3D can also render images composed using other programs such as ***Molscript***, ***ortex***, and ***XtalView***. Two shell scripts, ***stereo3d*** and ***label3d***, automate the process of producing stereo pairs and of including labels in the rendered image.

Raster3D uses a fast Z-buffer algorithm to produce high quality pixel images featuring one shadowing light source, additional non-shadowing light sources, specular highlighting, transparency, and Phong shaded surfaces. Output is in the form of a pixel image with 24 bits of color information per pixel.

Raster3D does not depend on any graphics hardware. Four common image output formats are supported: AVS, JPEG, TIFF, and SGI libimage. To actually view or manipulate the images produced, you must also have installed an image viewing package (e.g. John Cristy's ImageMagick or the SGI libimage utilities). The Raster3D rendering program can be integrated with ImageMagick to expand the flexibility of output formats and interactive use.

Ancillary programs are provided for the generation of object descriptions based on atomic coordinates stored in PDB format. Although the rendering program is not specific to molecular graphics, Raster3D is not intended as a general purpose ray-tracing program. Some of the algorithms used have been chosen for speed rather than generality; that is, they happen to work well for the types of images the program is intended for, but may produce odd results if used for very different types of image.

Raster3D is freely available. If you use the package to prepare figures for publication, *please* give proper credit to the authors: Merritt and Bacon, 1997, *Methods in Enzymology* **277**:505-524.

HISTORY

Version 2.5: The ***render*** program now supports Z-clipping, indirection of header records, interpolated coloring of cylinders, JPEG output, and piped output to ImageMagick for image format conversion. Support for the inclusion of PostScript labels in rendered images has been updated by incorporation of the ***r3dtops*** code into ***render*** under control of the command line option *-labels*. ***render*** now also accepts command line options that override information in the input header records. In particular *-size HHHxVVV* will force the output image to be exactly HHH by VVV pixels, and the various anti-aliasing and matting options may be selected using the options *-aa*, *-alpha*, and *-draft*.

Version 2.4: The ***render*** program now supports generalized quadric surfaces. There is a new utility program ***rastep*** (Raster3D thermal ellipsoid program) which generates "thermal ellipsoid" representations of atoms based on the B_{iso} or U_{ij} entries (ANISOU cards) in a PDB file. The option of colouring based on B values has been added here and elsewhere. The Raster3D distribution kit now includes a subdirectory containing sample material definitions illustrating the use of transparency, 2-sided materials, and other possibilities. Triangles now may be colored by interpolation of colors assigned to the three vertices. Input lines beginning with '#' are now treated as pure comments. A new object type allows specification of global rendering options such as depth-cueing. Filenames used for indirect input (input lines beginning with @) are searched for first in the current directory, then in a library directory specified by the environmental variable R3D_LIB.

Version 2.3: The ***render*** program now supports file indirection, additional anti-aliasing options, a separate alpha blend (matte) channel in the rendered image, and some additional properties for special materials. There is now an option to specify an explicit bond radius to ***rods***. There is a new option *-stereo* for the ***normal3d*** utility, and a new shell script ***stereo3d*** is provided to automatically generate a side-by-side stereo pair from a Raster3D input file.

Version 2.2: The major new feature in Raster3D Version 2.2 is support for rendering transparent objects. Transparent molecular surfaces may be described in terms of triangles with explicit surface normals. This permits Raster3D to render images which include both transparent molecular surfaces and the previously supported modes of representing protein structure. In keeping with the philosophy of the package, the intention is not to duplicate the power of more general programs for composing or describing molecular surfaces. The idea rather is to provide a tool for creating images with greater resolution and photorealism than is generally possible using a graphics workstation screen. Also new in this version is the ***normal3d*** utility, a tool to facilitate merging Raster3D input files created by different programs.

Version 2.1: Raster3D Version 2.1 supports rendering of five object types: spheres, triangles, planes, smooth-ended cylinders, and round-ended cylinders. New to Version 2.1 is support for two "object types" which are really modifiers for characteristic properties of existing objects. One of these allows specification of explicit surface normals at the vertices of triangles; the other allows specification of the surface reflectance properties of individual objects. Starting with version 2.1 the utility program for composing space-filling models, formerly called ***setup***, is renamed to ***balls***. This change is to avoid conflict with system utilities also called "setup". If desired, backwards compatibility with existing scripts may be retained by creating an alias or symbolic link for the command ***setup***.

EXAMPLES

Using only programs included in the Raster3D distribution one can create and render space-filling models, ball-and-stick models, ribbon models, and figures composed of any combination of these. The following set of commands would produce a composite figure of an Fe-containing metalloprotein with a smoothly shaded ribbon representation of the protein and spheres drawn for the Fe atoms:

```
#
# Draw smooth ribbon with default color scheme 2,
# save description (with header records) in ribbon.r3d
#
cat protein.pdb | ribbon -d2 > ribbon.r3d
#
# Extract Fe atoms only, and draw as spheres.
# Color info is taken from colorfile.
# Save description (with no header records) in irons.r3d
#
grep "FE" protein.pdb | cat colorfile - | balls -h > irons.r3d
#
# combine the two descriptions and render as AVS image file
#
cat ribbon.r3d irons.r3d | render > picture.x
```

One can alternatively use Molscript to produce a Raster3D input file by using the -r switch. Integrated use of Molscript/Raster3D/ImageMagick allows one to describe, render, and view 3D representations of existing Molscript figures.

```
molscript -r infile.dat | render | display avs:-
```

A similar example using ***xv*** as an image viewer, and assuming that TIFF support has been built into the render program:

```
molscript -r infile.dat | render -tiff image.tif
xv image.tif
```

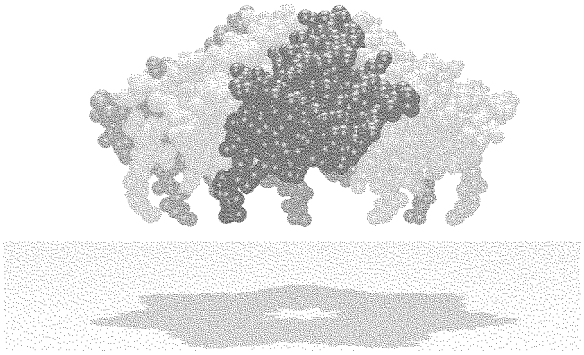
The Raster3D distribution also includes a filter utility which will convert the 24-bit color output stream from ***render*** into a dithered black & white PostScript image:

```
render < description.r3d | avs2ps > picture.ps
```

SUGGESTIONS FOR BETTER PICTURES

Tailoring your image to the output device: The quality of pictures generated by Raster3D is ultimately limited by the output device. Although you will probably compose and preview your figures on a workstation screen, you will probably want to re-render the final version with a larger number of pixels before sending it to a film recorder or high performance color printer. For example, a typical film recorder can produce slides with a resolution of roughly 4000x3000 pixels (much larger than can be displayed on a workstation screen). The number of pixels in your rendered image is controlled by the parameters (NTX,NTY) and (NPX,NPY) in the 2nd and 3rd header records input to the ***render*** program, or by the *-size* command line option. You should also be aware that color balance and particularly the appropriate "gamma correction" varies from one device to another. Raster3D itself applies no gamma correction; if you need one you will have to apply it to the generated image files afterwards. This is a standard image processing procedure, and may be a selectable print option for your output device. If you will be using a particular output device regularly, it is worth an initial round of experimentation to determine the best gamma value for future runs. The appropriate gamma correction can then be applied to each rendered picture before sending it for printing.

Shadowing: The primary purpose of shadowing is to convey a feeling of depth in the rendered image. This can be particularly effective when there are a relatively small number of elements in the scene, or when there is a grouping of foreground objects which cast shadows onto a separate grouping of background objects. However if there are a large number of elements in the scene, e.g. a ribbon and arrow representation of a large molecular assembly, the use of shadows may actually complicate the visual impact and make the image less satisfactory. In this case you may wish to try rendering the image both with and without shadows, or you might experiment with the location of the primary light source (the SOURCE parameter in the 12th header record input to ***render***). You can de-emphasize the shadows by increasing the secondary (straight-on) light source contribution parameter STRAIT. A separate use of shadowing can be to convey information which otherwise would not be apparent in the figure. The example below uses shadowing to indicate a central pore, avoiding the need for a second orthogonal view.



"No substance can be comprehended without light and shade"
-The Notebooks of Leonardo da Vinci

Making your pictures modular: So long as you stick to a consistent coordinate system, you can build up a complex Raster3D scene from bits and pieces created by various different tools and programs. For example, you could create a scene that is defined by a viewpoint selected interactively in Xfit, and that contains a protein molecule drawn by Molscript, a molecular surface drawn by GRASP, and a "floor" or bounding box from the Raster3D library of pre-described objects. This scene can conveniently be described to the render program by using file indirection in the input stream, which might look something like this:

```
#
# Header records written from inside Xfit
```

```
@viewpoint.r3d
#
# Molscript V2.0 output file
@secondary-structure.r3d
#
# Make the molecular surface (from GRASP via ungrasp)
# transparent by using a material definition
# from the Raster3D library $R3D_LIB
@transparent.r3d
@surface.r3d
@end_material.r3d
#
# Add a few extra goodies
@red.r3d
@floor.r3d
```

This makes it very easy to experiment with your composition without having to edit huge input files. You could, for example, change the color of the floor by substituting the library file *blue.r3d* for the file *red.r3d*. Or you could render the same scene from a different viewpoint by changing the view matrix in *viewpoint.r3d*. Even better, you could select from a number of pre-defined views described by header records in files *view1.r3d*, *view2.r3d*, *view2.r3d*, and so on just by making *viewpoint.r3d* be a symbolic link to the particular view you want to render. That way you needn't edit any input files at all to shift the scene. This approach is particularly useful for producing animated sequences (see the HTML guide to using Raster3D for animation).

Side-by-side figures and stereo pairs: The EYEPOS parameter input to the ***render*** program specifies a viewing distance for the resulting image. You may think of this as equivalent to the distance between a camera and the object being photographed. EYEPOS = 4 means that the distance from the camera to the center of the object is four times the width of the field of view. Generally the sense of depth conveyed by the rendered image is slightly increased by positioning the virtual camera reasonably close to the object. However, if you are composing a figure containing two or more similar objects which are next to each other, e.g. a comparison of two variants of the same protein structure, then the resulting parallax may be more of a hindrance than a help. Since the virtual camera is centered, it will "see" the right hand object slightly from the left, and the left hand object slightly from the right. This results in different effective viewpoints for paired objects which would otherwise be identical. To overcome this effect you may wish to set EYEPOS to 0.0, which disables all perspective and parallax. The same considerations apply for the production of stereo pairs.

Stereo pairs and Molscript: All Raster3D objects emitted by a Molscript run are placed into a single scene description. That is, the pairs of "plot" and "end plot" statements in a Molscript input file have no effect in Raster3D mode. Therefore a Molscript file that describes a stereo pair as two separate plots will not work correctly when fed through to ***render***. You should instead use Molscript to produce a single [mono] scene description for Raster3D, and run it through ***render*** twice to produce the two images making up a stereo pair. There is a shell script in the Raster3D distribution which automates this procedure. The script, ***stereo3d,*** will automatically render an input file to ***render*** as a side-by-side stereo pair rather than as a single image. The recommended procedure for producing stereo pairs using Molscript/Raster3D is simply to produce a single (mono) image descriptor file, and then render that file using the ***stereo3d*** script. Here is an example:

```
#
# Use molscript to generate a Raster3D input file
#
molscript -r < image.mol > image.r3d
#
# Render image once to check that it's what we want
#
render -tiff mono.tiff < image.r3d
display mono.tiff
#
# Render it again, this time as a stereo pair
# Note: the output file is always called stereo.tiff
#
stereo3d image.r3d
```

```
display stereo.tiff
```

Creating your own colour descriptions: The Raster3D distribution contains a number of pre-defined colouring schemes in the directory /materials (environmental variable R3D_LIB). If you choose to modify these, or create your own, then you should be aware that ***render*** works internally with the square root of the specified colour values (yes it's a very strange concept!). To convert a normal RGB colour triple into a Raster3D color specification it is therefore necessary to square each component. For example, to specify a half-intensity yellow you would square the components of the pure RGB triplet (0.5, 0.5, 0.0) to yield (0.25, 0.25, 0.0) in the COLOUR record input to ***render***. The Raster3D output option to Molscript performs this conversion automatically, so you should not have to alter the colour specifications to switch between PostScript and Raster3D modes. The HTML file *r3d_colorpicker.html* provides an interactive colour previewing tool.

Composing figures in other programs: Suppose you are already working in some interactive graphics program, FRODO for instance, and wish to reproduce the current viewpoint/orientation for a Raster3D picture. If the program will dump the current view matrix then you will probably be able to use it as a view matrix for Raster3D also. However many programs (including FRODO and Molscript) dump a matrix which is the transpose of the matrix used by Raster3D. The next section will describe how to convert this into a TMAT matrix for lines 13-16 of a Raster3D input file header.

Users of Alwyn Jones' program O should obtain a copy of the program ***o2mol*** from the O ftp site. Once you have composed your view in O you can convert to a Molscript/Raster3D viewpoint description by dumping the O datablock named .GS_REAL

```
O> write_form .gs_real omatrix.dat
Heap> Format: <cr>
```

followed by a shell command

```
o2mol < omatrix.dat > o2mol.out
```

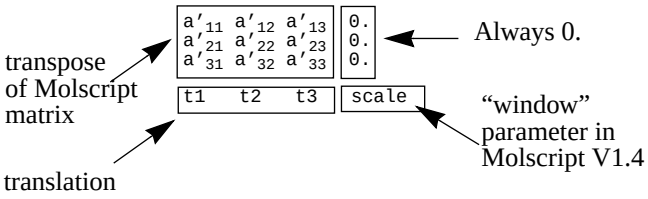
The file o2mol.out will now contain instructions for reproducing the O viewpoint in Molscript. Also, the 3x3 matrix which ***o2mol*** gives as "for coordinate transforms" is in the correct form to use directly as a Raster3D matrix if you aren't processing with Molscript first.

Duncan McRee's ***XtalView*** program (***Xfit***) can create Raster3D input files or images directly from the current screen view, including atoms, bonds, view objects, electron density, etc. Many Raster3D rendering options can be varied using control widgets in the ***Xfit*** pop-up plotting menus. ***Xfit*** can also be used simply to create a set of Raster3D header records describing the current view.

Another interactive tool you might be interested in is the ***VMD*** program from the Theoretical Biophysics group at the University of Illinois. ***VMD*** provides a wide variety of methods for interactively rendering and coloring a molecule, and can generate a Raster3D input file which will very nearly duplicate the view composed on your workstation screen.

Duplicating the view matrix used in a Molscript figure: When Molscript runs it normalizes the coordinates of objects in the figure so that they are described by an identity transformation matrix. The 3x3 matrix printed out by Molscript to the terminal is the transpose of that needed by Raster3D programs. Swap the entries about the diagonal from upper left to lower right to create a new matrix before copying it into the TMAT header records for ***render*** or ***normal3d***. The translation components reported by Molscript may be used as reported. If you are using Molscript V1.4, take the scale parameter in TMAT from the value of the "window" parameter in your Molscript run.

To normalize other Raster3D input files describing objects still in the original PDB coordinate space (so that they can be merged with your Molscript output), replace the four TMAT records in the header with a new matrix built as below and then feed the resulting file through ***normal3d***.



Converting Molscript labels to Raster3D: The most frequent question I get about Raster3D is "How can I get the labels from Molscript into my Raster3D picture?". There is now a very easy answer - *"Install Molscript Version 2!"* The newer Molscript version (available via the web from *http://www.avatar.se/molscript*) draws labels in such a way that they will simply appear in your Raster3D image along with everything else.

The rest of this section is therefore now obsolete, at least with regard to Molscript. It may, however, be useful as an example of how to mix Raster3D and PostScript processing. Generally this requires more effort than the result is worth. I recommend simply labelling your Raster3D output image using an interactive annotation tool on a workstation screen (e.g. the annotate mode of ImageMagick's ***display*** program). However, for anyone bound and determined to try, I will describe below how it can be done.

- The first thing to do is to establish the size of your eventual image in pixels. Let us say that you want a 3-inch by 2-inch print to be produced on a 300 dpi (dots per inch) dye sublimation printer. 300 dpi times the desired size gives 900x600 pixels for the image size. Construct a header file for the ***render*** program which describes this image size; e.g.

```
My new picture with no anti-aliasing
45 30    NTX,NTY number of tiles in x, y
20 20    NPX,NPY number of pixels per tile
1        anti-aliasing scheme 1 (none) or 4 (smoothed)
0 0 0    black background
(etc...)
```

  Pay particular attention to the fact (yes, it's confusing) that if you use the old anti-aliasing schemes 2 or 3 then the eventual number of pixels is not simply the product NTX*NPX,NTY*NPY. This is explained in the render manual.

- Now make sure that your Molscript input file is actually drawing in a box of the shape you have just described. To use the procedure I describe below, you must use the "area" command of Molscript to force the lower left corner to (0,0) and the upper right corner to, in this case, (900,600). Furthermore there must be no bordering frame drawn around the image. So your Molscript input file will begin:

```
plot
noframe
area 0. 0. 900. 600. ;
background black ;
```

- Run Molscript with the -raster3d option to produce an input file to render which uses the header you constructed in step 1.

```
molscript -r < picture.mol | render | convert avs:- picture.ppm
```

- Edit the molscript file to remove or comment out all drawing commands except for the labels. Run Molscript again, this time in PostScript mode, to produce a PostScript file containing only the labels.

```
molscript < picture_labels.mol > labels.ps
```

- Use ghostscript to convert the PostScript file into a raster image exactly the same size as your Raster3D image.

```
gs -sDEVICE=ppm -sOutputFile=labels.ppm -g900x600 labels.ps
```

At this point you have two images:

```
picture.ppm (900x600 Raster3D image from original file)
```

```
labels.ppm  (900x600 image containing only the labels)
```

- Use the ***combine*** program from ImageMagick to draw the labels on top of the Raster3D image.

```
combine -compose atop labels.ppm picture.ppm pic+labels.ppm
```

I used the ppm raster file format throughout this example just because it works in both ImageMagick and ghostscript. (*Warning: This may be installation dependent*.) Now I'll convert it back to a TIFF file.

```
convert pic+labels.ppm beautiful.tiff
```

You can see this is a rather lengthy procedure. Furthermore, the label positioning will be slightly off compared to the pure PostScript version because Raster3D and Molscript treat perspective differently. As I say, I prefer to just type in the labels interactively so that I can see where they are going.

Black & White figures: It is possible to use a general image processing program (e.g. ImageMagick) to convert a color figure to a monochrome figure. In general a straight conversion will produce an image which is much too dark. In order to improve the result you can try breaking the conversion up into several steps: first convert from full 24-bit color to a smaller number of colors (say 256), next apply a substantial gamma correction (e.g. gamma 2.0) to lighten the image, and finally convert the color image to monochrome being sure to select a dithering option if it is available. Some experimentation with this process can produce acceptable, although not ideal, monochrome images suitable for printing on a standard laser printer. You may have to repeat the color reduction and gamma correction steps before finally converting to monochrome. Better results can be obtained by using the auxiliary program ***avs2ps*** as a filter. This utility will convert any AVS-format image, including the default output stream from ***render***, directly into a dithered black & white PostScript image. The ***avs2ps*** program is included with the Raster3D distribution.

NOTES ON TRANSPARENT SURFACES

MATERIAL descriptors (object type 8) include a parameter controlling transparency. Material descriptions apply to subsequent objects in the render input file until an termination record (object type 9) is encountered. This means that an entire surface representation, for instance one generated by the programs ***GRASP*** (Nicholls et al., 1991) or ***MSMS*** (Sanner et al, 1995), can be rendered as transparent by inserting one record in front of it and another at the end. This is illustrated in example 6 in the Raster3D distribution kit.

Nothing is free, of course, and this includes transparency. I tried very hard to leave normal bread-and-butter operation of the rendering program unaffected by support for transparency. If you are actually rendering an image which contains transparent objects, however, the program may slow down by as much as a factor of 3. If your machine seems to do much worse than this, you can try the fix described at the end of this section.

Here is an example of a MATERIAL descriptor which specifies a transparent surface:

```
8
17.0    0.6       -1.0 -1.0 -1.0   0.9     0 0 0 0
 ↑      ↑          ↑                ↑      ↑
MPHONG MSPEC       SR  SG  SB          CLRITY   OPTS
```

The first record specifies a MATERIAL descriptor (object type 8). The following record contains a set of parameters which apply to this material.

MPHONG	overrides the global Phong parameter in the header; a lower value (<20.) smears out specular highlights over a larger area, which is probably a good idea for transparent surfaces.
MSPEC	overrides the global SPECLR parameter in the header; controls what fraction of the total shading is due to specular reflection. This probably should be > 0.5 for a transparent object.
SR,SG,SB	red, green, and blue components of the specular highlights. Remember that each object has an associated RGB triple that determines its colour. But the "colour" of a transparent object is somewhat problematic, so you have the option of passing the

object's basic colour through to appear also in the specular highlighting. This is indicated by setting SR, SG, SB negative as in the example above.

CLRITY This is a value between 0.0 and 1.0 indicating the degree of clarity of the material. 0 indicates a completely opaque surface, while 1 indicates a completely transparent surface. Values in the range of 0.8 to 1.0 are probably appropriate.

OPTS These four parameters are reserved for future expansion of the MATERIAL specification. The first one (OPTS[1]) affects the handling of transparency.

OPTS[1] = 0

All transparent objects will be rendered, subject to the limit that at a depth of three overlapping transparent objects the third is effectively opaque

OPTS[1] = 1

Only the topmost surface of any given transparent material will be rendered at all. This will, for example, cause internal cavities to disappear from a molecular surface covering an entire protein. It also allows rendering only the outside surface of a transparent CPK model.

OPTS[1] = 2

Render rear-facing portions of transparent spheres and cylinders (normally only the front surface is rendered.

Here is an example of a portion of a simple input file to render which draws three spheres, one of which is transparent.

```
[ many records, including header, omitted, ...]

# The following object (a sphere) will get whatever properties were
# defined in the header records.
2
 -0.3  0.0 0.0    0.2    0.7 0.3 0.1
# Now we specify material properties which override the header values
# and draw another sphere. This one will be transparent, with yellow
# highlights.
8
 17.0  0.5              1.0 1.0 0.0   0.9   0 0 0 0
2
  0.3  0.0 0.0    0.2    0.7 0.3 0.1
9 This record is NOT just a comment; it terminates the special material
9 Now draw one more sphere, which again has no special properties
2
  0.5  0.0 0.0    0.2    0.7 0.3 0.1
```

Here is another version of the same input fragment, using file indirection and a pre-defined transparency description from the library of materials (environmental variable R3D_LIB).

```
[ many records, including header, omitted, ...]

# The following object (a sphere) will get default properties
2
 -0.3  0.0 0.0    0.2    0.7 0.3 0.1
# Now we specify a transparent material and another sphere
@transparent.r3d
2
  0.3  0.0 0.0    0.2    0.7 0.3 0.1
# Terminate transparency and draw a third sphere
@end_material.r3d
2
  0.5  0.0 0.0    0.2    0.7 0.3 0.1
```

What to do if transparent objects seem to turn your machine to molasses: The extra time involved in rendering transparent objects is mostly due to bookkeeping overhead rather than to the actual calculation of surface properties, at least on the machines I have clocked it on. The algorithm used, however, does include at least one cosine calculation for each pixel and I can easily imagine that some machines (or some FORTRAN libraries) would execute this calculation very slowly indeed. Therefore I have put a commented-out alternative algorithm in the render source code, along with instructions for re-compiling. The alternative algorithm produces a slightly less realistic result (in my purely empirical opinion) but avoids any cosine functions.

RASTER3D DISTRIBUTION SITE

anonymous ftp site:

ftp.bmsc.washington.edu

via WWW:

http://www.bmsc.washington.edu/raster3d/raster3d.html

contact:

Ethan A Merritt
Dept of Biological Structure, Box 357742
University of Washington
Seattle WA 98195
merritt@u.washington.edu

RELATED PROGRAMS

In order to build in support for the direct output of TIFF image files, you must separately obtain and install a copy of the TIFF library (libtiff.a). Several implementations are available, including one by Sam Leffler which may be obtained via anonymous ftp from *ftp://sgi.com/graphics*.

In order to build in support for direct output of JPEG images, you must have a copy of the JPEG library. If your system does not already have this library, you can obtain it directly from the Indepent JPEG Group development project via anonymous ftp from *ftp://ftp.uu.net/graphics/jpeg/*.

In order to view the rendered images on a workstation screen you must also obtain and install an image viewer. Two commonly available viewers are John Christy's ***ImageMagick*** (*http://www.wizards.duport.com*, also available via anonymous ftp from *ftp.x.org* among other places) and John Bradley's ***xv***.

The ***avs2ps*** utility for direct production of black & white PostScript images is now included in the Raster3D distribution. It replaces earlier machine-dependent programs in the ***viewtools*** kit.

Several auxiliary programs are available from the ftp site above which may be useful, though they have not been kept up to date with the current Raster3D version. These include two attempts to aid in the composition of Raster3D images (***atoms***, ***prerast3d***) and tools for conversion to half-toned monochrome images and annotation and display of the images (***viewtools***). Most of the code was written specifically for SGI workstations, and may not transport well to other platforms.

Some other programs with options for output to Raster3D:

Conscript

A program for generating electron density isosurfaces (an alternative to the usual chicken-wire). Source and executables available from the Conscript web page: *http://www.bioresi.com.au/conscript*.

Molscript

There is now a Molscript web page at *http://www.avatar.se/molscript*

ORTEX

ORTEX V7 (an interactive descendent of Carroll Johnson's ORTEP program) now supports Raster3D as an output mode. ORTEX runs under DOS/Windows. *http://www.ucg.ie/cryst/software.htm*

Xfit/XtalView

General crystallographic model building, map fitting, and analysis program by Duncan McRee, available in both academic and commercial versions, *http://www.scripps.edu/pub/dem-web/index.html*

VMD

Biomolecular visualization tool from the Theoretical Biophysics group at the University of Illinois, *http://www.ks.uiuc.edu/Research/vmd/*

Raster3D AUTHORS

Originally written by David J. Bacon and Wayne F. Anderson. Ancillary programs by Mark Israel, Stephen Samuel, Michael Murphy, Albert Berghuis, and Ethan A Merritt. Extensions, revisions, and modifications by Ethan A Merritt.

CITING THE PROGRAMS

If you use the Raster3D package to prepare figures for publication, *please* give proper credit to the authors; the proper citation for the most recent version of the package is Merritt & Bacon (1997) as given below.

REFERENCES

Bacon, D.J., & Anderson, W.F. (1988). "A Fast Algorithm for Rendering Space-Filling Molecule Pictures". (abstract of paper presented at the Seventh Annual Meeting of the Molecular Graphics Society). *J. Molec. Graphics* **6**, 219-220.

Kraulis, P. (1991) "MOLSCRIPT: a program to produce both detailed and schematic plots of proteins". *J. Appl. Cryst.* **24**, 946-950.

Lawrence, M.C. & Bourke, P. (2000). "CONSCRIPT: a program for generating electron density isosurfaces for presentation in protein crystallography." *J Appl Cryst* **33**, 990-991.

Merritt, E.A. & Murphy, M.E.P. (1994) "Raster3D Version 2.0 - A Program for Photorealistic Molecular Graphics". *Acta Cryst.* **D50**, 869-873.

Merritt, E.A. & Bacon, D.J. (1997). "Raster3D: Photorealistic Molecular Graphics". *Methods in Enzymology* **277**, 505-524.

Nicholls, A., Sharp, K. & Honig, B. (1991). *PROTEINS, Structure, Function and Genetics* **11**, 281-96.

Sanner, M.F., Olson, A.J., & Spehner, J-C. (1995). "Fast and robust computation of molecular surfaces". *Proc. aath ACM Symp. Comp. Geom.* C6-C7.

The figure of the *M. leprae* groES heptamer used to illustrate shadowing is courtesy of Shekhar Mande.

AVS2PS

SYNOPSIS

avs2ps converts an AVS image input on *stdin* to monochrome PostScript on *stdout*

```
avs2ps [-b] [-dpi xxx] < infile.avs > outfile.ps
```

avs2ps converts a 24-bit color image file in AVS format into a dithered monochrome PostScript image with the same number of pixels as the input file. ***avs2ps*** may be used as a filter for the output of the ***render*** program to produce a PostScript file directly. The code is machine independent, and does not impose restrictions on the tile size in the original image. It supersedes less general programs in the viewtools package.

avs2ps converts the input stream to a greyscale image and then applies an empirical algorithm for contrast enhancement, dithering, and error diffusion to produce a monochrome output image. The basic approach is a variant of Floyd-Steinberg error diffusion.

EXAMPLES

Produce an unbordered black & white image suitable for printing on a 300 dpi PostScript printer:

```
render < description.r3d | avs2ps > image.ps
```

Add a border, and prepare image for a 400 dpi printer:

```
render < description.r3d | avs2ps -b -dpi 400 > image.ps
```

OPTIONS

-b

Draw a border around the figure. By default ***avs2ps*** will produce a borderless image 0.5 inch in from the bottom left of the page.

-dpi xxx

By default ***avs2ps*** writes header records into the PostScript output file which are correct for a 300 dpi printer (e.g. an HP Laserjet IIIsi). If there is a mis-match between the header records and the actual resolution of the printer the image quality is substantially degraded. This option allows one to specify a different printer resolution (e.g. *-dpi 400* for a Next printer, or *-dpi 95* for Display PostScript on a 19" 1280x1024 workstation screen).

AUTHORS

Ethan A Merritt. Dithering algorithm derived from code by Randy Read and Albert Berghuis.

BALLS

SYNOPSIS

balls is a preprocessor which prepares a description of a space-filling model for the Raster3D molecular graphics package

```
balls [-h]
```

balls reads a file describing atom colours and/or a PDB coordinate file and produces a stream of Raster3D descriptor records on *stdout*, one sphere for each atom in the input file. The file produced by ***balls*** may be fed directly to ***render*** or it may be combined with descriptor files produced by other Raster3D utilities.

EXAMPLES

To describe a simple space-filling model coloured by residue type:

```
cat mycolours.pdb protein.pdb | balls | render > mypicture.avs
```

To include a pre-selected view matrix with the same model:

```
cp view1.matrix setup.matrix
cat mycolours.pdb protein.pdb | balls | render > mypicture.avs
```

To prepend header records describing a pre-selected scale and view:

```
cat mycolours.pdb protein.pdb | balls -h > balls.r3d
cat header.r3d balls.r3d | render > mypicture.avs
```

OPTIONS

-h

Suppress header records in output. By default ***balls*** will produce an output file which starts with header records containing a default set of scaling and processing options. The *-h* flag will suppress these header records so that the output file contains only sphere descriptors. This option is useful for producing files which describe only part of a scene, and which are to be later combined with descriptor files produced by other programs.

DESCRIPTION

The input to ***balls*** consists of a single text file containing colour information and atomic coordinates in PDB data bank format. Coordinates are output as Raster3D descriptor records with colours and sphere radii assigned according to the COLO records described below. By default, atoms are assigned CPK colours.

By default the output file contains a set of header records as required by the render program. The header is constructed to include a TMAT matrix corresponding to the transformation matrix contained in file setup.matrix (if it exists), or to the Eulerian angles contained in file setup.angles (if it exists).

Colours are assigned to atoms using a matching process, using COLOUR records prepended to the input PDB file. Raster3D uses a pseudo-PDB record type with COLO in the first 4 columns:

Columns	*Contents*
1 - 4	COLO
7 - 30	Mask (described below)
31 - 38	Red component
39 - 46	Green component
47 - 54	Blue component
55 - 60	van der Waals radius in Angstroms
61 - 80	Comments

Note that the Red, Green, and Blue components are in the same positions as the X, Y, and Z components of a PDB ATOM or HETA record, and the van der Waals radius goes in place of the Occupancy. The Red, Green, and Blue components must all be in the range 0 to 1.

The Mask field is used in the matching process as follows. First the program reads in and stores all the ATOM, HETA, and COLO records in input order. Then it goes through each stored ATOM/HETA record in turn, and searches for a COLO record that matches the ATOM/HETA record in all of columns 7 through 30. The first such COLO record to be found determines the colour and radius of the atom.

In order that one COLO record can provide colour and radius specifications for more than one atom (e.g., based on residue or atom type, or any other criterion for which labels can be given somewhere in columns 7 through 30), the # symbol is treated as a wildcard. I.e., a # in a COLO record matches any character found in the corresponding column in an ATOM or HETA record. All other characters must match exactly. Note that the very last COLO record in the input should have # symbols in all of columns 7 through 30 in order to provide a colour for any atom whose ATOM/HETA record fails to match any previous COLO record. This idea of matching masks for colour specifications is due to Colin Broughton.

Several files of COLO records, including one based on Bob Fletterick's "Shapely Models" and another mimicking CPK model parts, are provided as samples.

ENVIRONMENT

The files setup.matrix and setup.angles, if they exist, affect the header records produced by ***balls***.

AUTHORS

Originally written by David J. Bacon and Wayne F. Anderson. Extensions and revisions by Ethan A Merritt.

LABEL3D

SYNOPSIS

label3d processes a Raster3D input file containing label descriptors (object types 10, 11, 12) into a TIFF image label3d.tiff containing both the molecular graphics image and the associated labels. ***label3d*** can accept input either from a file or from *stdin*.

```
label3d infile.r3d; display label3d.tiff
```

or

```
cat list-of-files | label3d -
```

DESCRIPTION

Unfortunately, the Raster3D ***render*** program cannot fully process labels. To overcome this lack, earlier Raster3D distributions included an auxilliary program ***r3dtops***, which complements ***render*** in that it can process the labels but not the graphics objects. The ***r3dtops*** code is now part of the ***render*** program itself, and is invoked if the *-labels* option is present.

The ***label3d*** script automates the process of running a Raster3D input file through ***render***, running the resulting PostScript output through ***ghostscript***, and then recombing the two component images into a single TIFF image containing both the molecular graphics objects and the associated labels. Several intermediate files are produced by the script, and one output file is created. Any existing files having these names will be destroyed.

label3d.tmp	- temporary copy of input stream
label3d.ps	- PostScript description of labels created by ***r3dtops***
label3d.ppm	- PBM+ format raster image produced by ***ghostscript*** from label3d.ps
render.tiff	- TIFF image produced by ***render*** containing molecular graphics but no labels
label3d.tiff	- TIFF output image containing both molecular graphics and labels

ENVIRONMENT

By default the script assumes a fontscale appropriate for a 300 dpi printer, invoking ***render*** with the option *-fontscale 3.0*. You can change this by setting the environmental variable FONTSCALE.

This script requires TIFF support in ***render***, the unix utility *sed*, and installed versions of ***ghostscript*** and ImageMagick.

LISTING

```
#!/bin/csh -f
#
#        Raster3D labeling script label3d V2.5
#
# Notes:
#        - Assumes label support in render (in previous version
#          label support was from separate program r3dtops).
#        - This version uses sed to find image size
#          (previous versions used awk/nawk).
#        - Requires ghostscript and ImageMagick.
#        - Depending on what size images you typically produce,
#          you might want to change the FONTSCALE definition below
#
if ${?FONTSCALE} == '0' setenv FONTSCALE "3.0"
#
if ($1 == ''-'') then
   cat > label3d.tmp
else
```

```
   cp $1 label3d.tmp
endif
#
rm -f render.tiff label3d.ps label3d.ppm
render -labels -fontscale $FONTSCALE -tiff render.tiff < label3d.tmp
if (! -e render.tiff) exit(-1)
if (! -e label3d.ps) then
   mv render.tiff label3d.tiff
   exit(0)
endif
setenv IMAGESIZE `identify render.tiff | sed -e 's/.* \([0-9]*x[0-9]*\)
.*/\1/'`
alias gs3d gs -sDEVICE=ppm -dNOPAUSE -q -sOutputFile=label3d.ppm
gs3d -g$IMAGESIZE label3d.ps -c quit || echo "GhostScript error" && goto
cleanup
combine -compose over render.tiff label3d.ppm label3d.tiff
#
cleanup:
rm -f label3d.ps
rm -f label3d.tmp label3d.ppm render.tiff
#
```

BUGS

The script should pass through options to render. Labels always appear on top of the molecular graphics image, even labels which should be occluded by an object in the foreground.

AUTHOR

Ethan A Merritt

NORMAL3D

SYNOPSIS

normal3d reads a Raster3D input file from *stdin*, applies any coordinate manipulations specified in the header, and writes the modified file to *stdout.*

```
normal3d [-h] [-expand] [-stereo] < infile.r3d > normalized.r3d
```

The output file from ***normal3d*** describes exactly the same image as the original input file. All header records are left unchanged except for the transformation matrix (which becomes the identity matrix) and the format specifiers (which are set to *). The *-h* flag will suppress all header records in the output file. All objects in the input file are also in the output file, but their coordinate descriptions have been normalized (i.e. the original transformation matrix has been applied).

The *-expand* flag causes the program to in-line and normalize all instances of file indirection in the input stream. This results in a single render input file containing no file indirection. The default is to simply copy file indirection lines (those begining with @) to the new input file without opening them or normalizing their contents.

normal3d also reports the total number of objects in the input file by object type, and gives the array sizes which would be required for the render program to process this file. It may therefore be used to determine how large the array sizes in ***render*** should be set, should one of your image descriptions exceed the values compiled into ***render***.

EXAMPLES

Feed a large file through ***normal3d*** to judge array size requirements for rendering:

```
normal3d < largeinputfile.r3d > /dev/null

 tmat (v' = v * tmat):
 -0.7543700     0.2779890     -0.6327600    0.0000000E+00
  8.3310001E-02 0.9392580      0.3367110    0.0000000E+00
  0.6530600     0.2012850     -0.6973090    0.0000000E+00
  -54.14897     -48.99439      -13.01923     25.00000
 ------------------------------
 spheres   =    67
 cylinders =  5128
 triangles = 23402
 ------------------------------
 Compare these to the array dimensions in render:
 special materials =      1 (check against MAXMAT)
 total objects     =  47402 (check against MAXOBJ)
 details           = 506943 (check against MAXDET)
 shadow details    = 364323 (check against MAXSDT)
 ------------------------------
 True center of input coordinates (not used):
 -0.7435400 3.0629992E-02 -0.4582600
```

OPTIONS

-h

Suppress header records in output. This option is useful for producing files which describe only part of a scene, and which are to be later combined with descriptor files produced by other programs.

-expand

in-line and normalize all instances of file indirection (lines beginning with @) in the input file.

-stereo

The *-stereo* flag causes the program to generate two new files, *left.r3d* and *right.r3d*, containing header records suitable for rendering the normalized object description file as a side-by-side stereo pair (see ***stereo3d***). Header records in the primary output file are always suppressed if the *-stereo* option is selected.

AUTHORS

Ethan A Merritt

R3DTOPS (Raster3D to PostScript label conversion)

SYNOPSIS

The conversion of Raster3D label records (object types 10, 11, 12) into PostScript is performed by the ***render*** program under control of the *-labels* command line option. In earlier versions of Raster3D this function was performed by a separate program ***r3dtops***, which is now obsolete. The ***label3d*** shell script automates the process of rendering a Raster3D input file containing labels. This process requires installation of ***ghostscript***.

OPTIONS (to the render program)

-fontscale xx

It is problematic to interpret the font sizes requested in Raster3D font descriptor records so that they produce the desired size font on the eventual output device or printer. By default, the eventual output device is assumed to be approximately 100dpi, which is typical of computer monitors. However, printers generally range from 300 to 1200 dpi. The parameter *xx* is a scale factor by which the number of pixels per PostScript unit should be increased. The ***label3d*** script sets this by default to 3.0, which is appropriate for eventual printing on a 300 dpi printer, otherwise to the current value of the environmental variable FONTSCALE.

-labels [filename]

Process labels (object types 10,11,12) and create a PostScript output file in addition to the rendered image. If no filename is given, the output file is named *label3d.ps*. This default is used by the ***label3d*** script.

OBJECT DESCRIPTORS RECOGNIZED

Object type 10

"font name" size "alignment"

The default values are "Times-Bold" 10 "Left". Legal fonts specifiers are subject to the details of label processing implementation. In the context of ***label3d/r3dtops/ghostscript*** processing,

fontname	is any PostScript font name recognized by ***ghostscript***,
size	is the fontsize in points (modified by the *-fontscale* option)
alignment	is "Right", "Left", "Center", or "Offset".

Object type 11

[XYZ] [RGB]

Label text

Object type 12

reserved (single line of text)

SPECIAL CHARACTERS

A small number of escaped characters are allowed in the label text string. These may be useful for tweaking the position of your labels:

\n new line
\b backspace
\v vertical tab (move up 1/2 line)
\A Angstrom symbol

TEX-LIKE SYNTAX

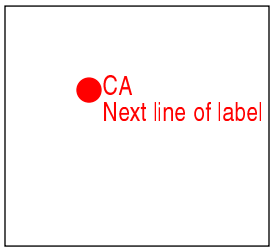
Certain conventions from the text-setting program ***TEX*** are supported by the Raster3D label processing code. In particular the character '_' introduces a subscript, while the character '^' introduces a superscript. If more than one character is to be sub- or super-scripted, the entire string can be enclosed in curly brackets {}. Greek characters and other symbols can be referred to by name if they are prefixed with a backslash.

EXAMPLES

Models of label processing and the use of the ***label3d*** script are given in example7 of the Raster3D distribution.

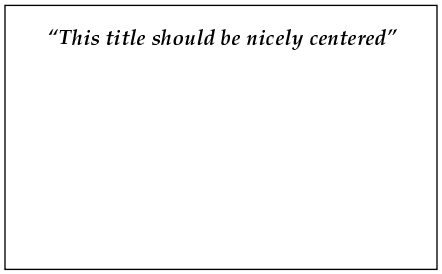
Here is a file fragment that places a two-line label immediately to the left of a particular atom (sphere):

```
# Here is a red sphere
2
11.31 43.45 -4.91   0.60    1.0 0.0 0.0
#
# And here is a label, also in red
10
"Helvetica-Narrow-Bold" 12.0 "Left"
11
11.31 43.45 -4.91           1.0 0.0 0.0
  CA\nNext line of label
```



Here is a file fragment that places a title across the top of an image independent of rotation/translation/scaling:

```
# Begin absolute coordinates
19
# Choose a nice big font for the title
10
"Palatino-BoldItalic" 30. "Center"
# Place it at top middle of figure
11
0.0 0.45 0.90              0.0 0.0 0.0
"This title should be nicely centered"
# End absolute coordinates
@end_material.r3d
```



Here are some examples of **TEX**-like label syntax:

label text	***appears in rendered image as***
N^{\epsilon2}-O^{\delta1} = 2.7Å	$N^{\varepsilon 2}$ - $O^{\delta 1}$ = 2.7Å
3\sigma [F_o - F_c]	3σ [F_o - F_c]
data \infty - 2.0\A	data ∞ - 2.0Å

The Angstrom symbol may either be entered directly as 'Å' [character 197 (octal 305)] or as '\A'.

BUGS

If a label contains **TEX**-like escape sequences it is always processed as "Left-align", since the program doesn't know enough about the eventual string width to center it properly.

Labels always appear on top of the molecular graphics image, even labels which should be occluded by a foreground object.

AUTHOR

Ethan A Merritt

RASTEP (Raster3D Thermal Ellipsoid Program)

SYNOPSIS

```
rastep  < infile.pdb > ellipsoids.r3d

rastep -tabulate [tabfile] [-by_atomtype] [-com [comtabfile]] \
       < infile.pdb > statistics.text
```

rastep reads a PDB coordinate file. This file must contain ANISOU records describing atoms refined with Anisotropic Displacement Parameters U_{ij}. ***rastep*** can either create an input file for the Raster3D ***render*** program or perform a statistical analysis of the atomic anisotropy for various classes of input atoms. By default the program creates an ellipsoid+stick scene description in which each atom is represented by an ellipsoid enclosing an isosurface of the probability density function. These are commonly known as thermal ellipsoids.

The program can be run in an alternate mode, controlled by the *-tabulate* option, in which the primary output to *stdout* is a list of the Eigenvalues of the U_{ij} matrix, followed by the corresponding atomic anisotropy and isotropic U_{eq}, for each atom in the input file with both an ATOM record and a matching ANISOU record.

EXAMPLES

Describe thermal ellipsoids at the 50% probability level, with default CPK colors, and send it for immediate rendering into a TIFF image file. The angle of view is automatically adjusted to spread atoms out as much as possible in the XY plane of the image:

```
rastep -auto < infile.pdb | render -tiff picture.tiff
```

Describe the same ellipsoids colored by B_{iso}, and create an input module with no header records for inclusion in a composite image:

```
rastep -h -Bcolor 10. 30. < infile.pdb > ellipsoids.r3d
cat header.r3d ellipsoids.r3d otherstuff.r3d | render -tiff picture.tiff
```

List anisotropy of individual atoms to *stdout* and summarize the distribution of anisotropy to a separate file:

```
rastep -tab summary.out < infile.pdb > anisotropy.out
```

OPTIONS

-auto

Auto-selection of viewing angle, chosen to minimize the spread of the atoms along the view direction.

-Bcolor Bmin Bmax

Assign colors based on B values rather than matching ATOM records against input or default COLOUR records. Atoms with B <= Bmin will be colored dark blue; atoms with B >= Bmax will be colored light red; atoms with Bmin < B < Bmax will be assigned colors shading smoothly through the spectrum from blue to red.

-fancy[0-3]

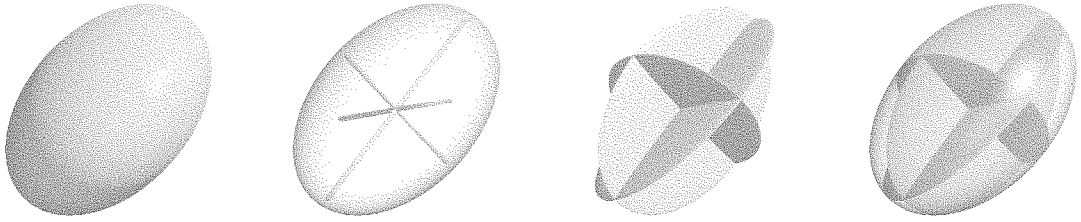
The *-fancy* option selects increasingly complex representations of the rendered ellipsoids (see figure)

-fancy0 [default] solid surface

-fancy1 draw principal axes of ellipsoid with a transparent bounding surface

-fancy2 draw colored equatorial planes of the ellipsoid

-fancy3 draw colored equatorial planes with a transparent bounding surface



-fancy0 -fancy1 -fancy2 -fancy3

-h

Suppress header records in output. By default ***rastep*** will produce an output file which starts with header records containing a default set of scaling and processing options. The *-h* flag will suppress these header records. This option is useful for producing files which describe only part of a scene, and which are to be later combined with descriptor files produced by other programs.

-iso

Force isotropic probability surfaces (spheres). By default ***rastep*** will look for ANISOU records in the PDB file and use these to generate ellipsoids. If no ANISOU record is present for a given atom, the B value given in the ATOM record will be used to generate a sphere instead. Selecting the *-iso* option will force the program to use the B value in the ATOM record even if an ANISOU record is also present.

-mini

Auto-orientation (as in *-auto*) and small size plot (176x208) described in header records.

-nohydrogens

Do not plot hydrogens, even if present in PDB file.

-prob Plevel

By default, isosurfaces are drawn to enclose the 50% probability level in the probability density function described by the U_{ij} values in the ANISOU record. The *-prob* option allows you to select a different probability level instead. If 0<*Plevel*<1 this value is interpreted as a fraction; if *Plevel* > 1 this value is interpreted as a percent.

-radius R

By default, ***rastep*** draws bonds with radius 0.10Å between neighboring atoms using the same algorithm as ***rods***. This option allows you to change the radius of the bonds. If the radius is set to 0 no bonds will be drawn.

OPTIONS FOR STATISTICAL ANALYSIS

-tabulate [tabfile]

The *-tabulate* option requests that the program accumulate and print statistics on the distribution of anisotropy among input atoms rather than producing an input file for ***render***. The principle axes and anisotropy of each atom are written to *stdout*. An overall statistical summary is written to *tabfile* if specified, otherwise to *stdout*.

-by_atomtype

The *-by_atomtype* option is a modifier to *-tabulate*. It causes a further subdivision of atoms by atom type (columns 77:78 in the PDB ATOM records) during the preparation of statistical summaries.

-com [comtabfile]

Tabulate distribution of anisotropy in shells by distance from center-of-mass. Output to *comtabfile* if specified, otherwise to *stdout*.

NOTES

There is little, if any, consistency in format among the various programs which write out anisotropic displacement parameters. This program interprets the U^{ij} values in the order specified for ANISOU records in PDB format. That is, columns 29-70 of the PDB record are interpreted as integers representing 10^4 Å^2 x U^{ij}, in the order U^{11}, U^{22}, U^{33}, U^{12}, U^{13}, U^{23}. Note in particular that the order of cross-terms is *not* the same as that used by ORTEP or SHELX, which do not use PDB format. However, the program ***shelxpro*** will produce correctly formatted PDB records from a SHELX coordinate file.

AUTHOR

Ethan A Merritt

REFERENCES

M.N. Burnett & C.K. Johnson (1996). “ORTEP-III: Oak Ridge thermal ellipsoid plot program for crystal structure illustrations”. ORNL-6895, Oak Ridge National Laboratory. Table 6.1

K.N. Trueblood et al (1996). “Atomic Displacement Parameter Nomenclature, report of a subcommittee on atomic displacement parameter nomenclature”. *Acta Cryst.* **A52**, 770-781.

E.A. Merritt (1999). “Expanding the Model: Anisotropic Displacement Parameters in Protein Structure Refinement". *Acta Cryst.* **D55**, 1109-1117.

RENDER

SYNOPSIS

render is the central image-rendering program of the Raster3D molecular graphics package.

Input is taken from stdin; output mode is controlled from the command line:

```
render                          AVS image to stdout
render [-quality NN] -jpeg      JPEG image to stdout
render -sgi [outfile]           output to SGI libimage file
render -tiff [outfile]          output to TIFF file
render -out outfile.xxx         pipe output to ImageMagick for conversion to image type xxx
```

Render reads an ascii file consisting of several header lines followed by individual object descriptors. The format of the header records and the object descriptors is given below. Objects are rendered using a fast Z-buffer algorithm to produce a high quality pixel image which contains one shadowing light source, additional non-shadowing light sources, specular highlighting, transparency, and Phong shaded surfaces. Output is to *stdout* [or optional file] in the form of a pixel image with 24 bits of color information per pixel. The default output file format is an AVS image sent to *stdout*. Support for JPEG, TIFF or SGI libimage output may be optionally included during installation of the Raster3D package. Additional output formats are possible by using the *-out* option if the ImageMagick package is installed also.

There are several utility programs that will create all or part of an acceptable input file by reading atomic coordinates from a PDB file. These include ***balls***, ***rastep, rods***, and ***ribbon*** from the Raster3D distribution kit, and Molscript [Per Kraulis (1991) *J. Appl. Cryst.* **24**, 946-950].

EXAMPLES

Render a simple space-filling model coloured by residue type:

```
cat mycolors.pdb protein.pdb | balls | render > mypicture.avs
```

Display the same picture on an X-windows display directly using ImageMagick:

```
cat mycolors.pdb protein.pdb | balls | render | display avs:-
```

Render and display the same picture using the SGI libimage format:

```
cat mycolors.pdb protein.pdb | balls | render -sgi picture.rgb
ipaste picture.rgb
```

Render a stick-figure model of a cofactor jointly with a ribbon model of a protein. Note that the header records are generated by the ***ribbon*** command in this case; the *-h* flag suppresses generation of a second set of header records in the ***rods*** command. The ImageMagick ***convert*** command is used to save the resulting image as a TIFF file.

```
ribbon -d4 protein.pdb > ribbon.r3d
cat colors.pdb cofactor.pdb | rods -bs -h > cofactor.r3d
cat ribbon.r3d cofactor.r3d | render | convert avs:- picture.tiff
```

If TIFF support has been built directly into Raster3D, then the previous command can be modified to produce a TIFF file without conversion:

```
cat ribbon.r3d cofactor.r3d | render -tiff picture.tiff
```

OPTIONS

-aa

Force anti-aliasing (SCHEME 4) regardless of the scheme specified in the file header

-alpha

Force output of transparency information (SCHEME 0) regardless of the scheme specified in the file header. This only works if the output format supports an alpha channel (AVS, TIFF, but not JPEG). Note that this turns off anti-aliasing. This option can be used to produce an image with a transparent background.

-draft

Turn off anti-aliasing (force SCHEME 1) to increase rendering speed.

-fontscale XX

Only meaningful in conjunction with the *-labels* option. Modifies the interpretation of font sizes during label processing. *-fontscale 3.0* will generate PostScript labels that are approximately the nominal font size when printed at 300 dpi. Specifying the *-fontscale* option on the command line overrides the environmental variable FONTSCALE; if neither is specified, the value defaults to 3.0

-help

Print summary of options.

-invert

Invert the image top-to-bottom. This may be necessary if you are using some odd viewing program.

-jpeg

Only if compiled with -DJPEG_SUPPORT. By default ***render*** will produce an AVS-compatible image on stdout. The *-jpeg* flag will cause it to output a JPEG image stream to stdout instead.

-labels [outfile.ps]

Process labels (object types 10,11,12) and create a PostScript output file in addition to the rendered image file. If no filename is given, the output file is named *label3d.ps*. This option is used by the ***label3d*** script. More detail is given under the ***r3dtops*** heading.

-out filename.xxx

Only if compiled with -DIMAGEPIPE. The *-out* flag will cause ***render*** to pipe output to the ImageMagick command ***convert***, with instructions to convert it to an image file whose type is determined by the filename extension *xxx*.

-quality NN

Only meaningful in conjunction with the *-jpeg* flag; sets the output image quality parameter. Allowable values range from 1-100 (default = 90).

-size HHHxVVV

Override the image size parameters (NTX,NTY,NPX,NPY) in the file header and produce an output image that is exactly HHH pixels in the horizontal and VVV pixels in the vertical.

-sgi [filename.rgb]

Only if compiled with -DLIBIMAGE_SUPPORT. The *-sgi* flag will cause ***render*** to output an SGI libimage style *.rgb file instead of writing to stdout. In this case default output filename is render.rgb, but specifying a file on the command line will override this default.

-tiff filename.tiff

Only if compiled with -DTIFF_SUPPORT. The *-tiff* flag will cause ***render*** to output a TIFF image to the specified file instead of writing to stdout.

-transparent

(Same as *-alpha*) Force output of transparency information (SCHEME 0).

HEADER RECORDS

The required header records of a render input file are described below. Except where noted, each set of parameters listed is read from a single line in Fortran free format. The names are of variables in the program source code.

TITLE

Anything you like, up to 80 characters.

NTX,NTY

Number of "tiles" in each direction. (The output display is considered to be divided up into an array of identical rectangular tiles.) The maximum is 192 unless you have increased MAXNTX and MAXNTY in render.f.

NPX,NPY

Number of computing pixels per tile in each direction. Maximum = 36 (MAXNPX, MAXNPY in render.f) The program runs faster when NPX, NPY are small numbers; i.e. if you have a choice, it is better to increase the size of your image by increasing the number of tiles (NTX,NTY) rather than the size of the tiles (NPX,NPY).

SCHEME

Pixel averaging (anti-aliasing) scheme. Anti-aliasing reduces the jaggedness of edges at the cost of additional computation. If you are going to matte your images against an externally generated background, use scheme 0 (matting and anti-aliasing do not mix well).

0 means no anti-aliasing, include alpha blend (matte) channel in output image

1 means no anti-aliasing, no alpha channel

2 means anti-alias by averaging 2x2 computing pixels for each output pixel

3 means anti-alias by averaging 3x3 computing pixels for each 2x2 output pixels (obsolete - use scheme 4 instead)

4 anti-alias as in scheme 3, but header specifies *final* raster size, rather than *computing* raster size.

I.e. schemes 0. 1, and 4 produce a NTX*NPX by NTY*NPY pixel image; scheme 3 produces a (2/3)NTX*NPX by (2/3)NTY*NPY image. Scheme 3 requires that NPX and NPY be divisible by 3; schemes 2 and 4 require that NPX and NPY be divisible by 2.

Images can be previewed using scheme 1 for greater speed, and re-rendered with anti-aliasing scheme 4 with no change in the output image size. No changes to NTX,NTY,NPX,NPY are required in this case.

BKGND

Background colour (red, green, and blue components, each in the range 0 to 1).

SHADOW

T to calculate shadowing within the scene, F to omit shadows

IPHONG

Phong power (e.g., 25) for specular highlights. A smaller value results in a larger spot.

IPHONG = 0 disables specular highlighting and all processing of ribbon triangles

STRAIT

Straight-on (secondary) light source contribution (e.g., 0.15). The primary light source contribution (see also SOURCE below) is given by PRIMAR = 1 - STRAIT.

AMBIEN

Ambient illumination contribution (e.g., 0.05). Increasing the ambient light will reduce the contrast between shadowed and non-shadowed regions.

SPECLR

Specular reflection contribution (e.g., 0.25).

The diffuse reflection quantity is given by DIFFUS = 1 - (AMBIEN+SPECLR). Ambient and diffuse reflections are chromatic, taking on the specified colour of each object, whereas specular reflections are white.

EYEPOS

You can think of the image produced by ***render*** as corresponding to a photograph taken by a camera placed a certain distance away from the objects making up the scene. This distance is controlled by the EYEPOS parameter. EYEPOS = 4 describes a perspective corresponding to a viewing distance 4 times the narrow dimension of the described scene. EYEPOS = 0 disables perspective and parallax altogether.

SOURCE

Primary light source position (e.g., 1 1 1). This is a white light point source at infinite distance in the direction of the vector given (see note on co-ordinate convention below). The secondary light source is always head-on. Only the primary light source casts shadows.

TMAT

Homogeneous global transformation for input objects, given as a 4x4 matrix on 4 lines just as you would write it if you intended it to be a postfix (suffix) operator. The upper left 3x3 submatrix expresses a pure rotation, the lower left 1x3 submatrix gives a translation, the upper right 3x1 submatrix should be zero (otherwise extra perspective is introduced), and the lower right scalar (h) produces global scaling. Note that the scale factor h ends up being applied as an inverse; i.e. a larger value of h will result in shrinking the objects in the picture. Input coordinate vectors [x y z] are extended with a 1 to make them homogeneous, and then post-multiplied by the entire matrix; i.e. [x´ y´ z´ h´] = [x y z 1][TMAT], then the ultimate co-ordinates are [x˝ y˝ z˝] = (1/h´)[x´ y´ z´].

INMODE

Object input mode (1, 2, or 3), where mode 1 means that all objects are triangles, mode 2 means that all objects are spheres, and mode 3 means that each object will be preceded by a record containing a single number indicating its type. The Raster3D programs use only mode 3.

INFMT or INFMTS

Object input format specifier(s). For object input modes 1 and 2, there is just one format specifier INFMT for the corresponding object type, while for mode 3, there are three format specifiers INFMTS on three lines. The first describes the format for a triangle, the second for a sphere, and the third for a cylinder. Each format specifier is either a Fortran format enclosed in parentheses, or a single asterisk to indicate free-format input.

SAMPLE HEADER

```
My picture. (describes a 1280 x 1024 pixel anti-aliased image)
80 64          tiles in x,y
24 24          pixels (x,y) per tile
3              anti-aliasing level 3; 3x3->2x2
0 0 0          black background
F              no shadows cast
25             Phong power
0.25           secondary light contribution
0.05           ambient light contribution
0.25           specular reflection component
4.0            eye position
1 1 1          main light source position (from over right shoulder)
1 0 0 0        view matrix describing input coordinate transformation
0 1 0 0
0 0 1 0
0 0 0 0.6      no translation; enlarge objects in picture by 66% (1/.6)
```

```
3             mixed objects
*             (free format triangle and plane descriptors)
*             (free format sphere descriptors)
*             (free format cylinder descriptors)
```

OBJECT TYPES

Object type 1 triangle

x1, y1, z1, x2, y2, z2, x3, y3, z3, red, green, blue

Object type 2 sphere

x, y, z, radius, red, green, blue

Object type 3 round-ended cylinder

x1, y1, z1, R1, x2, y2, z2, R2, red, green, blue
(R1 is the cylinder radius, R2 is currently ignored).

Object type 4 not used

Object type 5 flat-ended cylinder

x1, y1, z1, R1, x2, y2, z2, R2, red, green, blue
(R1 is the cylinder radius, R2 is currently ignored).

Object type 6 plane

x1, y1, z1, x2, y2, z2, x3, y3, z3, red, green, blue

Object type 7 explicit surface normals at vertices of preceding triangle

u1, v1, w1, u2, v2, w2, u3, v3, w3
Explicit vertex normals for preceding triangle object. This object must directly follow the triangle object (see also object type 17). The intended use of this object type is to allow description of arbitrary molecular surfaces in terms of a tesselation by triangles.

Object type 8 material properties

These values override the specification of lighting and specular highlighting in the header records, allowing some objects to have different surface properties from the rest. The specified values will apply to all subsequent objects until an object of type 9 is encountered. The parameters are read in free format from the following line of the input stream:

MPHONG	(floating point) override global Phong parameter for specular highlighting
MSPEC	(floating point) override global specular scattering contribution
SR,SG,SB	RGB triple specifying color of reflected light (by default all reflections are white), a negative value for any component will default to the base colour component of the object being rendered
CLRITY	(floating point value between 0.0 and 1.0) The degree of transparency for this material; 0.0 indicates an opaque surface and 1.0 indicates a purely transparent one.
OPTS(4)	four additional fields are reserved for future expansion of the material properties list; these must be present. The first field controls alternative algorithms for rendering transparent objects:

OPT(1) = 0 render self-occluding portions of a transparent material separately

OPT(1) = 1 render only the “top” surface of transparent material

OPT(1) = 2 render the rear surface of transparent spheres and cylinders

The last field is non-zero to signal that additional modifier records will follow immediately. Each modifier record must constitute a single line, and OPT(4) states how many of these there will be. For example

OPT(4) = 2 means that the next two lines contain additional material modifiers

Object type 8 modifiers

Additional modifiers to MATERIAL descriptors (type 8) are now supported. Each modifier record must constitute a single line of input, and the total number of such modifier lines must be specified in the final parameter of the MATERIAL record. Modifiers currently supported include SOLID and BACKFACE records, as shown in the following example of a 2-sided material that is solid red on one side and solid blue on the other side:

```
# Here is an example of specifying a double sided material
# The MATERIAL record itself specifies an opaque material with PHONG
# value to 25 and specular parameters taken from the header records,
# and says that there are two following modifer lines.
# The SOLID modifier gives an RGB color triple that over-rides colors on
# subsequent objects.
# The BACKFACE modifier specifies a separate RGB color for the other side
# of this material, and sets the Phong control parameter and specular
# component of this surface to zero (i.e. it’s a matte blue surface).
# The net result of specifying both SOLID and BACKFACE in this case is to
# describe a material that is shiny red on one side and matte blue on the
# other side.
8
20. -1.   1. 1. 1.   0.0    0 0 0 2
SOLID     1.0 0.0 0.0
BACKFACE  0.3 0.3 1.0   0 0
```

Here is the complete list of modifiers

```
SOLID       red green blue
BACKFACE    red green blue mphong mspec
FRONTCLIP   zfront
```

Object type 9 terminate special properties

Terminates previous set of special material properties (object type 8) or isolation from TMAT transformation (object type 15).

Label descriptors (object types 10-12)

Version 2.3 introduced preliminary support for labels in a render input file. This has been substantial upgraded in version 2.5. Object types 10, 11, and 12 are used to specify labels. The current implementation of label handling is only partially done in ***render***. The executable shell script ***label3d*** will automatically feed an input file containing labels through ***render***, ***ghostscript*** and ***ImageMagick*** to yield a final image containing both the renderered molecular graphics objects and the specified labels. Here is an example:

```
# Force the label coordinates to be interpreted as screen coords
# (obviously you wouldn’t do this if you want the label to get the
# same translation/rotation/scaling as other objects do)
15
# Choose a font
10
”Times-Italic” 12. “Center”
# specifiy a label
11
0.0 0.45 0.0       1.0 0.0 0.0
```

```
I am a red title centered at the top of the image
# Terminate isolation from coordinate transformation
9
```

For more information on label processing, please see the documentation for ***r3dtops*** and ***label3d***.

(If you are developing a higher-level application that will pass labels to Raster3D for rendering, please contact me so that we can make sure the protocols are compatible -*EAM*).

Object type 10

"font name" size "alignment"

The default values are "Times-Bold" 10 "Left".

Legal fonts specifiers are subject to the details of label processing implementation. In the context of ***label3d/r3dtops/ghostscript*** processing, fontname is any PostScript font name recogized by ***ghostscript***, size is the fontsize in points (assuming eventual printing at 300 dpi), and alignment is either "Right", "Left", "Center", or "Offset".

Object type 11

[XYZ] [RGB]

Label text

Object type 12

reserved (single line of text)

Object type 13 Glow light source.

This is a colored, non-shadowing, light source with finite [x y z] coordinates and a limited range of illumination. Control parameters are read in free format from a single line of input following the line specifying the object type.

GLOWSRC(3)	[x y z] coordinates of light source
GLOWRAD	limiting radius of light source (see GOPT)
GLOW	fractional contribution (0.0 - 1.0) of glow light to total lighting model
GOPT	(integer 0/1/2/3/...) - controls functional form in which limiting radius is applied [under development]
GPHONG	Phong parameter controlling specular highlights from glow light source
GLOWCOL(3)	RGB triple specifying color of glow light source

Object type 14 Quadric surface

X, Y, Z, limiting_radius, red, green, blue, A, B, C, D, E, F, G, H, I, J

Quadric surfaces include spheres, cones, ellipsoids, paraboloids, and hyperboloids. The motivation for this code was to allow rendering thermal ellipsoids for atoms, so the other shapes have not been extensively tested. A quadric surface is described by 10 parameters (A ... J). The surface is the set of points for which Q(x,y,z) = 0,

where

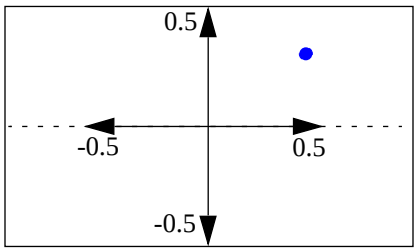
$$Q(x,y,z) = Ax^2 + By^2 + Cz^2 + 2Dxy + 2Eyz + 2Fzx + 2Gx + 2Hy + 2Iz + J$$

Although these parameters are sufficient to describe a quadric surface located anywhere, for efficiency during rendering it is also useful to know the center and a bounding sphere. So by convention we require that (A...J) describe a surface centered at the origin, and add additional parameters X, Y, Z that specify a translation component. Therefore a quadric surface descriptor to render has the 17 parameters listed above. Points further from the origin (prior to translation!) than the limiting radius are not rendered.

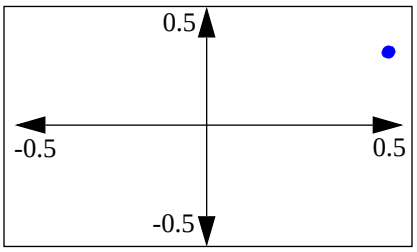
Object type 15 begin isolation from TMAT transformation

Objects following this descriptor are interpreted as being in an absolute coordinate system centered at the origin and having unit extent in X, Y, and Z. If the image is square, the coordinates run from roughly -0.5 to +0.5 on both X and Y. If the image is rectangular, coordinates run from -0.5 to +0.5 on the shorter dimension. The rotation and translation described by the TMAT matrix is not applied. This isolation from TMAT applies to subsequent objects until an object of type 9 is encountered.

The coordinate system used following object type 19 is similar except that the horizontal and vertical scales are treated separately, so that the point [-0.5,+0.5] is always at the top left corner. In both figures below the blue dot is drawn at coordinates [+0.5,+0.3].



Coordinate system used following object 15. Coordinates run from -0.5 to +0.5 along the smallest dimension of image. Coordinate units along the other dimension are the same, which means that the extremes are greater than +/-0.5



Coordinate system used following object 19. Coordinates run from -0.5 to +0.5 along both dimensions of image, regardless of the aspect ratio.

Object type 16 Global properties

Object type 16 is always followed by a single record that affects the rendering of the entire scene. The global properties currently supported are listed below. FOG allows depth-cueing using either a linear or an exponential model of fog density. A record simply consisting of the keyword FOG will produce reasonable depth-cueing (linear decrease in intensity front closest object to farthest object). If you need to control the depth-cueing indepently from the specific rendered objects in a scene (i.e. to keep the depth-cueing consistent between frames of an animation) then you should specify explicit values for fogfront, fogback and fogden.

FOG **fogtype fogfront fogback fogden**

fogtype 0 = linear multiplier = fogden * (fogfront - Z) / (fogfront - fogback)
1 = exponential multiplier = 1. - exp(-fogden * (fogfront - Z) / (fogfront - fogback))

fogfront 0 = Z coord of front object, else fraction of front clipping plane

fogback 0 = Z coord of back object, else fraction of rear clipping plane

fogden (see equations for linear and exponential models controlled by fogtype); fogden=0.0 will default to some reasonable value

BACKCLIP **zback**

zback Z coord of rear clipping plane for all objects in original units of original coordinate system, but relative to the center of the transformed coordinate system. That means if you are working in PDB coordinates, then everything more than zback Ångstroms behind the center is clipped.

FRONTCLIP **zfront**

zfront Z coord of front clipping plane for all objects in original units of original coordinate system, but relative to the center of the transformed coordinate system. That means if you are working in PDB coordinates, then everything more than zfront Ångstroms in front of the center is clipped.

ROTATION Matrix

Matrix 3x3 rotation matrix applied by post-multiplication after the coordinate transformation in TMAT. Because it is post-multiplication, this is the transpose of the equivalent rotation if it were specified in TMAT. This rotation applies to all following coordinates, but not to objects that have already been processed.

TRANSLATION X Y Z

X Y Z 3 element translation applied after the coordinate transformation in TMAT and the post-rotation matrix, if any. This translation applies to all following coordinates, but not to objects that have already been processed.

Object type 17 explicit coloring for vertices of preceding triangle or cylinder

Red1 Green1 Blue1 Red2 Green2 Blue2 Red3 Green3 Blue3

Normally a triangle is colored uniformly in the color specified by the RGB triple in the TRIANGLE record itself. Using this additional object, however, you can specify a separate color for each vertex of the triangle. Points in the interior of the triangle are colored by interpolation using the vertex colors. This is commonly used in conjunction with explicit vertex normals (object type 7) to describe a smooth surface in terms of a triangular mesh. In this case the component triangles of the surface are described by three successive objects of types 1, 7, and 17. (The ordering 1, 17, 7 is also acceptable).

This mechanism may also be used to generate interpolated coloring along the length of a cylinder. In this case the third RGB triple is ignored.

Object type 18 explicit transparency at vertices of preceding triangle or cylinder

Trans1 Trans2 Trans3

Normally an object is transparent only if it belongs to a transparent material. Using this additional object, however, you can specify a separate degree of transparency at each vertex of an object. The transparency at points in the interior of a triangle or cylinder is determined by interpolation. All three values (Trans1 Trans2 Trans3) are required for a triangle; the third value is ignored for a cylinder.

Object type 19 begin isolation from TMAT transformation

See object 15 for details.

Object type 0 force end of input stream (optional)

Further records in input file will not be read.

Comments

At any point in the input stream to render where an object descriptor would be legal, it is also legal to insert a line beginning with the '#' character. In this case the line is ignored, and may be used as a comment. Earlier versions of the program recommended using object type 9 as a comment delimiter, but this has the potential disadvantage of prematurely terminating the scope of a special material.

FILE INDIRECTION

At any point in the input stream to render where an object descriptor would be legal, it is also legal to insert a line beginning with '@'. In this case the remainder of the line is interpreted as the name of a file from which further input is taken. This mechanism makes it possible to re-use standard objects in multiple rendered scenes, e.g. a set of bounding planes or standard definitions of material properties. When input from this level of file indirection is terminated by encountering an object descriptor of type 0 (or an end of file), control returns to the previous input stream. Multiple levels of file indirection are possible. The entire set of header records can also be read via file

indirection if the first line of the render input stream begins with '@', but indirection within the header is not supported.

The requested file is first searched for relative to the current directory. If this search fails, the suffix '.r3d' is added to the file name and the search is repeated. If this also fails, the file is searched for relative to the directory specified by the environmental variable R3D_LIB. Typically this would be set via a command such as

```
csh/tcsh:   setenv R3D_LIB /usr/local/src/raster3d/materials
sh/ksh:     R3D_LIB=/usr/local/src/raster3d/materials; export R3D_LIB
```

If the filename ends in .Z or .gz, the program will attempt to uncompress it into a scratch file using the ***gunzip*** command.

SAMPLE INPUT DATA FOLLOWING HEADER

The following records describe a single red triangle and a single blue sphere. The blue sphere is affected by whatever material properties are defined in the file material1.r3d

```
@red
1
-.1 0. 0.   .1 0. 0.   0. -.2 0.5     1.0 0.0 0.0
@material1.r3d
2
0.3 0.3 0.0     0.1    0.0 0.0 1.0
@end_material
0
```

ERROR MESSAGES

Possible shadowing error NSXMAX = xxx

This is usually caused by an object which extends far out of the field of view, for example a plane surface. In most cases the shadowing "error" refers to a shadow which lies outside of the image entirely. However, if your image does in fact contain missing or truncated shadows you can overcome this problem by re-compiling the render program with larger values of NSX and NSY as indicated by the error message.

INCREASING THE ARRAY SIZES IN RENDER

If you are creating images of very large proteins, or if you are rendering surfaces with many facets, then you may fill the storage arrays in render. If so you will get an error message something like

```
STOP 1234 **** too many objects - increase MAXOBJ and recompile
```

The parameters which you may need to increase are contained in the following lines of the program source file *render.f*:

```
*
* Maximum number of objects (was 7500)
PARAMETER (MAXOBJ = 25000)
*
* Array elements available for object details
PARAMETER (MAXDET = 250 000, MAXSDT = 250 000)
*
* Array elements available for sorted lists ("short" lists)
PARAMETER (MAXSHR = 150 000, MAXSSL = 150 000)
```

You can use the ***normal3d*** utility to calculate how many objects and object details are needed for your picture. The sorted list sizes are harder to calculate in advance, but the render program itself should indicate how much space would be required to complete a picture which it cannot currently handle.

Increase the appropriate PARAMETER values by editing the program source file *render.f*, and recompile the render program by typing "make render".

AUTHORS

Originally written by David J Bacon. Extensions, revisions, and modifications by Ethan A Merritt.

RIBBON

SYNOPSIS

```
ribbon [-h] [-d[0123456]] pdbfile
```

or (to take PDB records from stdin)

```
ribbon [-h] -d[0123456] -
```

Ribbon reads a PDB coordinate file and produces a file on *stdout* containing Raster3D descriptor records for a ribbon representation constructed from a triangular mesh. The file produced by ribbon may be fed directly to ***render*** or it may be combined with descriptor files produced by other Raster3D utilities.

EXAMPLES

To describe a the entire protein chain as a single ribbon colored smoothly from blue at the N-terminus to red at the C-terminus:

```
ribbon -d2 protein.pdb | render > chain_picture.avs
```

To color a multi-chain protein with specified colors for each chain:

```
cat chaincolors.pdb protein.pdb | ribbon -d5 - > chains.r3d
```

OPTIONS

-h

Suppress header records in output. By default ***ribbon*** will produce an output file which starts with header records containing a default set of scaling and processing options. The *-h* flag will suppress these header records so that the output file contains only triangle descriptors. This option is useful for producing files which describe only part of a scene, and which are to be later combined with descriptor files produced by other programs.

-d[0123456]

By default ***ribbon*** requires interactive input to select ribbon parameters and coloring information. Five default coloring schemes are implemented, however, and these may be selected as a command line option to bypass any interactive input.

-d or *-d0*	same as *-d2* below
-d1	solid color ribbon (defaults to blue)
-d2	shade from blue at N-terminus to red at C-terminus
-d3	one surface of ribbon is blue, other surface is grey
-d4	shade front surface from blue to red, back surface is grey
-d5	color separate chains using successive color cards from input stream. Note that pattern matching on the color records is *not* done; colors are simply taken sequentially as new chains are encountered.
-d6	use color templates from COLOUR records in the input file to take color from nearest CA

DESCRIPTION

The input to ***ribbon*** consists of a single text file containing colour information [optional] and atomic coordinates in PDB data bank format. Only CA and carbonyl O atom records are required; all other input atoms are ignored. Ribbon parameters and colouring specified interactively when the program is run. Keyboard interaction may be bypassed by selecting one of the default colouring schemes using the *-d* flag. A triangular mesh ribbon is output as Raster3D descriptor records. By default the output file contains a set of header records as required by the ***render*** program. The header is constructed to include a TMAT matrix corresponding to the transformation matrix contained in file setup.matrix (if it exists), or to the Eulerian angles contained in file setup.angles (if it exists).

Ribbon produces a continuous smooth trace of the protein backbone. For more complicated representations of protein secondary structure it is better to use a different program, e.g. Molscript, rather than ***ribbon***.

ENVIRONMENT

The files setup.matrix and setup.angles, if they exist, affect the header records produced by ***ribbon***.

AUTHORS

Original ribbon code written by Phil Evans for the CCP4 version of FRODO. Modification to describe solid ribbons as triangular mesh for Phong shading in Raster3D package by Ethan A Merritt.

RINGS3D

SYNOPSIS

rings3d searches through a PDB file looking for residues containing 5- or 6-membered rings, then produces a Raster3D output file of ring-filling triangles.

```
rings3d [-bases] [-protein] [-sugars]  < infile.pdb > outfile.r3d
```

rings3d matches residue types from an internal list of residue and atom names. It will fail to find residues not in its list, and fail to recognize atoms with non-standard names.

EXAMPLES

To render a DNA molecule with bases filled in:

```
cat $R3D_LIB/dna.colours dna.pdb | rods -radius 0.05 > temp.1
rings3d -bases < dna.pdb > temp.2
cat temp.1 temp.2 | render -tiff dna.tiff
```

OPTIONS

-bases

Fills in purine and pyrimidine rings from A C G T U residues.

-protein

Fills in sidechain rings of HIS PHE TRP and TYR residues.

-sugars

[*This is the default*] Fills in pyranose rings of GAL GLC NAG NGA MAN SIA residues.

BUGS

Limited residue type - the database of residue types should be kept externally, so you don't have to rebuild the program to add a new type.

Crinkled planes - There should be an option to do a least-squares best plane through supposedly flat rings. Then again, seeing a crease in a "flat" ring may force people to consider whether their planarity restraints are tight enough.

AUTHORS

Ethan A Merritt

RODS

SYNOPSIS

rods is a preprocessor for ball-and-stick figures in the Raster3D molecular graphics package

```
rods [-h] [-b] [-radius R] [-Bcolor Bmin Bmax]
```

rods reads a PDB coordinate file including COLOUR records as described below, and an output stream of Raster3D descriptor records. The file produced by ***rods*** may be fed directly to ***render*** or it may be combined with descriptor files produced by other Raster3D utilities.

EXAMPLES

To describe a simple bonds-only model coloured by residue type:

```
cat mycolours.pdb protein.pdb | rods | render > mypicture.avs
```

To render the same molecule as ball-and-stick:

```
cat mycolours.pdb protein.pdb | rods -b | render > mypicture.avs
```

OPTIONS

-h

Suppress header records in output. By default ***rods*** will produce an output file which starts with header records containing a default set of scaling and processing options. The *-h* flag will suppress these header records so that the output file contains only object descriptors. This option is useful for producing files which describe only part of a scene, and which are to be later combined with descriptor files produced by other programs.

-b

By default ***rods*** will describe bonds only; the -b flag will cause it to include spheres at the atom positions also, yielding a ball-and-stick representation.

-radius R

By default ***rods*** will use 0.2Å radius cylinders for bonds; the *-radius* flag allows you to change the cylinder radius to some other value. R is a floating point number.

-Bcolor Bmin Bmax

Assign colors based on B values rather than from atom or residue types. Atoms with B <= Bmin will be colored dark blue; atoms with B >= Bmax will be colored light red; atoms with Bmin < B < Bmax will be assigned colors shading smoothly through the spectrum from blue to red.

DESCRIPTION

The input to ***rods*** consists of a single text file containing colour information and atomic coordinates in PDB data bank format. Coordinates are output as Raster3D descriptor records with colours and sphere radii assigned according to the COLO records described below. Ball-and-stick figures have atoms drawn at 0.2 * Van der Waals radius, connected by rods with a default 0.2 Å cylindrical radius. Atoms are connected if they lie closer to each other than 0.6 * the sum of their Van der Waals radii.

By default the output file contains a set of header records as required by the ***render*** program. The header is constructed to include a TMAT matrix corresponding to the transformation matrix contained in file setup.matrix (if it exists), or to the Eulerian angles contained in file setup.angles (if it exists). Header records may be suppressed using the *-h* option.

Colours are assigned to atoms using a matching process, using COLOUR records prepended to the input PDB file. If If no COLOUR records are present in the input file, atoms will receive default CPK colors (C=grey,

O=red, N=blue, S=yellow, P=green, other=magenta).Raster3D uses a pseudo-PDB record type with COLO in the first 4 columns:

Columns	*Contents*
1 - 4	COLO
7 - 30	Mask (described below)
31 - 38	Red component
39 - 46	Green component
47 - 54	Blue component
55 - 60	van der Waals radius in Angstroms
61 - 80	Comments

Note that the Red, Green, and Blue components are in the same positions as the X, Y, and Z components of an ATOM or HETA record, and the van der Waals radius goes in place of the Occupancy. The Red, Green, and Blue components must all be in the range 0 to 1.

The Mask field is used in the matching process as follows. First the program reads in and stores all the ATOM, HETA, and COLO records in input order. Then it goes through each stored ATOM/HETA record in turn, and searches for a COLO record that matches the ATOM/HETA record in all of columns 7 through 30. The first such COLO record to be found determines the colour and radius of the atom.

In order that one COLO record can provide colour and radius specifications for more than one atom (e.g., based on residue or atom type, or any other criterion for which labels can be given somewhere in columns 7 through 30), the # symbol is treated as a wildcard. I.e., a # in a COLO record matches any character found in the corresponding column in an ATOM or HETA record. All other characters must match exactly. Note that the very last COLO record in the input should have # symbols in all of columns 7 through 30 in order to provide a colour for any atom whose ATOM/HETA record fails to match any previous COLO record. This idea of matching masks for colour specifications is due to Colin Broughton.

ENVIRONMENT

The files setup.matrix and setup.angles, if they exist, affect the header records produced by ***rods***.

AUTHORS

Ethan A Merritt

STEREO3D

SYNOPSIS

stereo3d is a shell script that renders a single Raster3D input file as a side-by-side stereo pair, leaving the resulting TIFF image in a file stereo3d.tiff.

```
stereo3d [-border] render_input_file.r3d
```

DESCRIPTION

stereo3d uses the Raster3D utilities ***label3d***, ***normal3d*** and ***render***, and ImageMagick utilities ***identify***, ***mogrify***, and ***montage***. Three intermediate files are produced by ***normal3d***:

stereo3d.tmp - normalized version of the input file

left.r3d - header records for left eye view

right.r3d - header records for right eye view

Additional intermediate files may be created by ***label3d***. The two views are separately rendered to yield images left.tiff and right.tiff (deleted upon completion), optionally given black borders, and merged to form a single side-by-side stereo pair stereo.tiff.

ENVIRONMENT

This script requires TIFF support in ***render***, the unix utility *sed*, and ImageMagick. The three intermediate files listed above are created each time the script is run. Any existing files named left.tiff, right.tiff or stereo.tiff will be destroyed.

The ***label3d*** script does not pass any parameters through to ***label3d*** or ***render***, so to change the font scaling you must set the environmental variable FONTSCALE.

LISTING

```
#!/bin/csh -f
#
if ($1 == ''-'') then
   normal3d -stereo -expand > stereo3d.tmp
else
   normal3d -stereo -expand < $1 > stereo3d.tmp
endif
#
echo "@stereo3d.tmp" >> left.r3d
echo "@stereo3d.tmp" >> right.r3d
render -tiff left.tiff  < left.r3d
render -tiff right.tiff < right.r3d
setenv SIZE `identify left.tiff \
    | sed -e 's/.* \([0-9]*x[0-9]*\) .*/\1/'`
printenv SIZE
montage +frame +label -background black -geometry $SIZE+0+0! \
    left.tiff right.tiff stereo.tiff
rm -f left.tiff right.tiff
```

(The actual script is somewhat more complicated, in that it first attempts to identify and process labels using the label3d script, and does some additional image processing to add a border if the *-border* option is chosen.)

AUTHORS

Ethan A Merritt