CMU RayTracer
Version 6.0: Brief Instruction Manual

Designer: Robert Thibadeau
Programmers: Tim Chow, Steve Handerson, and David Tin-Nyo

CMU-RI-TR-88-18

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

October 1988

Abstract

This report provides a brief manual for version 6.0, the first limited release of the **CMU RayTracer**. The CMU RayTracer is a complete implementation of **Adaptively Supersampled Distributed Ray Tracing with Octree Encoding for Texture Mapped Polygonal Models**. This graphics ray tracing program, like its cousins at Cornell, Bell Laboratories, and elsewhere, was intended to provide a testbed and research base which correctly incorporates the most recent "received views" in ray tracing for highly realistic graphics. It is the only ray tracing system of which we are aware that correctly incorporates the entire state of the art circa 1987. The novel elements of this program include (a) the only test combining "adaptively supersampled distributed ray tracing" with "octree and spatial enumeration encoding" (cf., [6]); (b) the only program which by program switches compares "octree," "spatial enumeration" and "z-buffer" performance assuming bounding boxes; (c) the only program which texture maps surface normals for light bending effects as well as color effects; (d) a fast normal interpolation scheme for polygons of any number of edges; (e) all standard illumination models; (f) supported compatibility across a large number of machine types; (g) a scheme for ray-culling called "inclusion cones"; and (h) sphere, cone cylinder and plane normals inference.

PART ONE: GETTING STARTED

1.1 Introduction

This package contains version 6.0 of the **CMU raytracer**. It is a complete implementation of **Adaptively Supersampled Distributed Ray-Tracing with Octree Encoding for Texture Mapped Polygonal Models.** The raytracer is written in **C** and version 6.0 has been ported to the following machines: **APOLLO BSD4.2, MSDOS IBM AT, CRAY UNICOS XMP, MACH VAX,** and **SUN OS.** Currently only the **APOLLO, APOLLO** with **AT&T TARGA** graphics board, and **IBM AT** with **AT&T Truevision**^{tm} **Image Capture Board (ICB)** are supported for display.

1.2 The basics

The home directory consists of two sub-directories: the code directory and the scenes directory. The code directory is often named after the release number, such as 6.0.

1.2.1 The code directory

This directory contains the following files: **.c** files (source code), **.bin, .o,** or **.obj** files, makefiles, and one or more raytracer executables. The executable "**rtn**" runs on all supported machines, and just creates a **.gro** file. To look at the result, you need to find one of the machines supported for display, and use the appropriate "show" program. (Just type "show name" where name is the name of the .gro file without the ".gro" suffix.)

1.2.2 The scenes directory

The scenes directory contains **.scn** files, optical surface files (e.g., **square**), two files **rndtable** and **rndtable2** (random number tables) and a file called **Attributes.** The **.scn** files are used as input to the file **rt**; each contains a description of a picture, including which optical surfaces appear in the picture. Each optical surface file contains a polygonal description of an object. The **Attributes** file contains a list of different kinds of surfaces. For more information, see section 2.3 below.

To draw a picture, first change directories to get into the scenes directory. Then type

../6.0/rt twocubes *(assuming the code directory is called "6.0")*

to raytrace the scene described in the file twocubes. Some initialization information will be printed, and then a window will be opened in the top left-hand corner of the screen. First, the outline (called a wire-frame) of the scene will be drawn, and then the picture will be raytraced one line at a time, starting from the bottom. Depending on the complexity of the scene, the running time will range between thirty minutes to several hours.

When the picture is completed, the program waits for you to type a carriage return, upon which the picture disappears from the screen. The picture is not lost, however; it has been written to a **.gro** file ("**twocubes.gro**" in the above example) in the scenes directory.

1.3 Some raytracing theory

The process of seeing with the eye can be modelled as a large number of light rays entering the eye. The idea of raytracing is to trace these rays back to determine where they came from and thereby determine their color and intensity. The CMU raytracer does exactly that: rays are fired at the scene; when they hit a surface, reflected and refracted rays are generated. The algorithm proceeds recursively, creating a tree of rays; a branch terminates when it hits a light, leaves the scene or reaches the prespecified maximum depth. At each intersection point, the light rays are modified according to the optical properties of the surface.

The chief drawback of raytracing is its computational cost. The most time-consuming process in the algorithm is finding intersections. The CMU raytracer allow the user to select one of two options: one is to test each ray against every polygon in the scene, and the other is to use octrees. The octree method divides the world up into eight octants, each of which may be subdivided recursively. Only the polygons in the same oct as the ray are tested. For even more speed, the ray is traced using an integer **DDDA** algorithm similar to the DDA algorithm used in drawing lines in 2D graphics. (See also section 4.0.)

The other key feature of a raytracer is its illumination model, which determines what happens to each ray when it hits a surface. The first problem one encounters is that light is arriving at the point of intersection from all different directions, but obviously it would be prohibitively costly to take all of them into account. The CMU raytracer handles this problem using a technique called distributed ray tracing: a random selection of directions are chosen and averaged. (See the paper by Cook, Porter and Carpenter, SIGGRAPH '84 Conference proceedings.) Since this technique creates a lot of noise, several samples are taken for each pixel and averaged ("super-sampling"). In practice, this technique produces very low contrast pictures. Thus, at each intersection point, a "shadow feeler" is projected towards the light sources to determine whether or not the point is in shadow with respect to that light source; this heightens contrast. The second problem is deciding how to modify the color and intensity of a light ray upon reflection or refraction. Several techniques have been proposed; the CMU raytracer allow the user to select the model he prefers. (See section 2.1.)

PART TWO: OPTIONS

2.1 Command line switches

The CMU raytracer allows the user to select different options, some of which have already been mentioned. Most of these options are controlled by switches that the user supplies as arguments in the command line. For example, to raytrace "twocubes.scn" without distributing the rays, type

../6.0/rt twocubes -Dd *(again, assuming "6.0" is the code directory)*

while in the scenes directory. Except for the .scn file, which is mandatory, you can have any number of arguments, and in any order. If no switches are specified, rt chooses certain defaults (see the Appendix).

For a full description of the switches, see the Appendix. You can also get a brief summary of the available switches by running rt without any arguments. Please pay special attention to the following switches:

-S *There can be only one -S switch per command line.*

-Oi *This makes rt behave like an early version of the CMU raytracer, before certain bugs had been ironed out. It should be used alone, without other switches.*

-V -M -Dp -Df *These eight switches tell rt to expect different file formats.*

-Of *See sections 2.2 and 2.3 below for more details.*

-R -Po -Pt See Appendix.

Other options can be selected by specifying the parameters in the various input files, which will now be described.

2.2 The .scn files

Following is a sample **.scn** file:

```
center lghtlck sptlncde octr graphmode normallength maxdepth maxdist
$ 0 2 1 4 3 .5 2 1
eyept & lookpt
$ 10.0 30.0 -75.0  0.0 0.0 0.0
screendist
$ 25.0
view angles x,y
$ 32.0 25.0
sample resolution x,y
$ 1
viewarea in pixels x,y (integers)
$ 512 400
output matrix boundaries (integers)
$ 0 0 511 399
name of file with list of attributes
$ Attributes
entities
$ 3
Entity 1: Cube
$ 1 optical surface
$ 1 12 0 cube      3  2 0 2  0 2 0 0
Entity 2: Checkerboard
$ 1 optical surface
$ 2 0 0 checker8    4  0 -3 0  0 0 0
Entity 3: White Square Light
$ 1 optical surface
$ 3 4 0 square     10  3 20 5  20 0 -10
$ 1
$ 0 lenspts.dat prescrip.dat 0.25 -5 -8 0 0 0 0 8 34
```

All lines are ignored except those whose **first non-blank character is a dollar sign**.

The first number controls **auto-centering.** Specifying 1 here asks the raytracer to change the look-point to the center of all the objects. You can then look at the output to determine where this is, to insert into the scene file (or modify slightly).

The second number is set to zero if there is to be no **shadow-feeling**. Setting it to 1 means that the vertices of the emissive polygons are treated like point light sources for shadow-feeling purposes. Setting it to 2 means that the centers of the polygons are the point light sources.

The third and fourth numbers in the first line should both be set to zero if you want to use the method of testing every ray against every polygon to **find intersections**. To use **octrees**, set the first number to one and the second number to the depth of subdivision of the octree. Three to seven are all good depths. To use **SEADS**, set the first number to the number of voxels on a side, and the second number to zero.

The fifth number is currently unused. It used to specify the output device.

The sixth number gives the length of the little green **normals** that appear in the wireframe pictures.

The seventh number in the first line gives the **maximum depth** of the ray tree. The smaller the number, the faster and coarser the image synthesis.

Eyept and **lookpt** refer to the location of the eye and the center of view respectively. The choice of view is not complete, since there's no way to specify rotation about the view axis (or which direction is up). As a rule of thumb, "up" is the positive y direction, and most scene files look along the z axis (to mimic the internals), which increases into the screen.

Screendist gives the distance from the eye to the screen.

The **view angles** give the angle subtended by the scene at the eyepoint.

The **sample resolution** controls the degree of supersampling. Square this number to get the maximum number of samples per pixel.

Output matrix boundaries gives the position of the window to be raytraced. The four numbers are the left, top, right and bottom of the window respectively. In this example, 0 0 512 399 means that the whole picture will be drawn.

One of the older versions of **rt** expected the contents of what is now **the Attributes file** in place of the line "$ Attributes" that we see here. Using the **-Df** switch in the command line causes rt to revert to this old format.

At this point it is possible to insert a line "$@name" where **@name** is the name of a file containing shape descriptions. See section 2.4 below. If this line is present, the **-M** switch *must* be invoked when **rt** is called. Conversely, the **-M** switch must NOT be used if the line is not present. See also section 2.5 below.

The number under "entities" (3 in this case) gives **the number of entities in the scene**. Entity 1 is a cube. The "$ 1 optical surface" means that the entity consists of only one optical surface (i.e., only one optical surface file is to be read). In the next line, the first number is just an index number and is ignored; the second number gives the number of the attribute in the attribute list (see section 2.3 below), and the third number refers to the filetype of the optical surface file (see section 2.5 below). Next comes the name of the optical surface file.

rt also accepts .scn files with a fourth number, just before the name of the optical surface file. This fourth number can be either zero or one. If it is zero, it is ignored; if it is one, it tells rt that this optical surface is to be **smooth-shaded**. If you wish to insert this fourth number, YOU MUST USE THE **-Po** OR THE **-Pt** SWITCH IN THE COMMAND LINE. Unpredictable errors will result otherwise. The fourth

number must be present for each optical surface if it is present at all. Furthermore, using **-Po** or **-Pt** when the fourth number is not present will cause unpredictable errors. Finally, you cannot use both **-Po** and **-Pt** in the same command line; you must use either one or the other.

Following the optical surface filename is a number indicating the **scale.** Then come three coordinates, indicating the position of the optical surface. Finally, there are three numbers indicating rotation about the x,y and z axes respectively (in degrees).

The last two lines in the file should be present if and only if the **-V (texture mapping)** switch is invoked in the command line. The "$ 1" in the above example indicates that there is 1 texture mapped optical surface. The next line contains a texture-mapped optical surface description. The first number is the attribute number; following that is the polygon filename, the texture map filename, the scale, the translation vector, the rotation angles (in degrees), and the number of rows and columns of polygons in the texture-mapped optical surface. Note the similarity to the normal optical surfaces. The main difference is that the texture-mapped optical surfaces cannot be Phong-shaded; even if the **-Po** or the **-Pt** switch is invoked, you should not add an extra field anywhere in these lines. Finally, note that the format of the polygon file and the texture map file is extremely rigid and specific. The polygons must form a grid of near-rectangles and the texture map must consist of cylindrical elements. See the files "**lenspts.dat**" and "**prescrip.dat**" for the details.

2.3 The Attributes file

Following is the start of the Attributes file:

```
attributes
$ 13
color(r,g,b) lighting model(pow,tpow,e,a,d,s,t,ref)  sdist tdist   description
$ 0.0 0.2 0.98  2.0 0.0 0.0 0.0 0.05 0.9 0.1 0.0  1.05  4.0 1.0 0.0 0.0  0
$ 0.8 0.2 0.0  2.0 0.0 0.0 0.0 0.05 0.8 0.4 0.0  1.05  4.0 1.0 0.0 0.0  1
$ 0.8 0.03 0.02 2.0 2.0 0.0 0.0 0.05 0.3 0.6 0.99 1.5  4.0 1.0 0.0 0.0  2
```

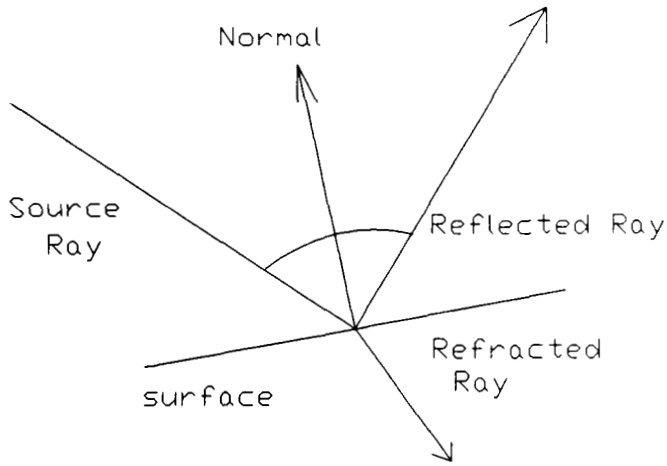
(The descriptive comments at the end of each line have been omitted.)

Again, lines not starting with a dollar sign are ignored. "**$13**" means that there are thirteen attributes in the list. Each subsequent line is an attribute description.

The first three numbers are the **red, green and blue intensities** of the surface and should be between 0.0 and 1.0 inclusive.

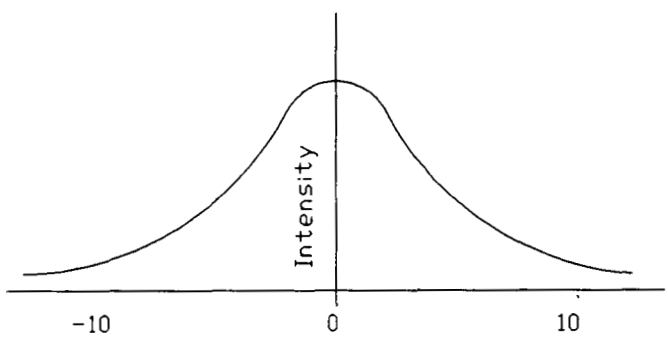
Specular reflection and specular refraction are controlled by a cosine to the power n law. The fourth number on the line gives this **exponent n for specular reflection** and the fifth number is the **exponent for specular refraction.** In general, 2.0 is a good default; higher numbers give sharper highlights.

The next five numbers are weighting factors that determine **how much light is emitted, ambient, diffusely reflected, specularly reflected and transmitted respectively**. These five numbers should add up to about 1.0.



The next number is the index of refraction. It is also possible to have three (RGB) indices of refraction here; this should be done if and only if the **-R** switch is used when rt is invoked. Note that -R means that ALL of the attributes must have three indices of refraction.

The next four numbers are distributed ray tracing parameters. A specularly reflected (or refracted) ray is distributed in a cone with an angle theta and according to a cosine to the power n law. These four numbers are **theta, n for reflection** and theta, **n for refraction** respectively. If you want to add your own attributes, we recommend making these numbers "60 4.0 60 4.0."



Finally, there is an index number. This is the number that should appear in the .scn file as the attribute number.

To append attributes of your own, simply add a new line to the file, making sure you adhere to the correct format. Remember to increment the first number in the file. The last field, the index number, should be the next whole number after the index number of the previous attribute.

NOTE: using the **-Dp** switch in the command line tells rt that the fourth and fifth numbers ("2.0 0.0" in the above example) are NOT present. Again, one must be consistent: if the two numbers are present, they must be present for EVERY attribute, and if they are absent, they must be absent from EVERY attribute. If the **-Dp** switch is used incorrectly, unpredictable errors will result. If you use **-Dp**, the program acts as if these two numbers were both 2.0.

NOTE ALSO that **-Of** is the equivalent of **-Dp** and **-Df** together.

2.4 The shape file

Since many real-life objects are spheres, cylinders and cones, and these objects cannot be represented accurately using polygons, it seems desirable to be able to simulate them in the raytracer. The technique adopted is to have a list of such shapes in memory, with the polygons that approximate its surface pointing to them. When such a polygon is intersected, we use the normal as calculated from the shape's parameters (radius etc.) instead of the normal to the polygon.

Obviously, to implement this, the list of shapes must be read in from a file. Thus, a shape file is (optionally) present in the scenes directory. (The **-M** switch is used if and only if this file is present; see also 2.2 above and 2.5 below.) As with the .scn and Attributes files, lines not starting with a $ are ignored. The first line contains the number of shapes in the file. Each line after the first contains seven floating point numbers.

If the shape is a sphere, the first three numbers are the coordinates of the center of the sphere, and the fourth number is the radius of the sphere. The last three numbers are ignored (but must nevertheless be present).

If the shape is a cylinder, the first three numbers are the coordinates of a point on the axis, the fourth number is the radius, and the last three numbers are the coordinates of another point on the axis.

If the shape is a cone, the first three numbers are the coordinates of the apex of the cone, the fourth number is the half-angle at the apex, and the last three numbers are the coordinates of a point on the axis other than the apex.

All coordinates are with respect to the same axes as the polygons.

Note that there is nothing in the shape file to indicate which of the three shapes is intended. This information is provided in the optical surface file (see 2.5 below).

2.5 The optical surface file

Three different optical surface file formats are accepted by rt. The format is specified by a number in the .scn file (see 2.2 above).

Below is a sample optical surface file (filetype zero).

```
4
0 -1 0 -1
1  1 0 -1
2  1 0  1
3 -1 0  1
4
0  0 1
1  1 2
2  2 3
3  3 0
1
```

The first number is the number of vertices in the optical surface. Following that are the vertex descriptions: an index number, followed by the three coordinates. Next comes the number of edges in the optical surface. Following that are the edge descriptions: an index number, followed by the index numbers of the vertices that are its endpoints. Next comes the number of polygons in the optical surface. Following that are the polygon descriptions: an index number, followed by the number of edges, followed by the index numbers of its edges, followed by the direction of the outward-pointing normal to the polygon, followed by an attribute offset. (The attribute offset is the number that you add to the attribute number of the optical surface [given in the .scn file; see 2.2 above] to get the attribute number of the polygon. This enables different polygons in the same optical surface to have different colors, for instance.)

If (and only if) the **-M** switch is used, every polygon description must have two more numbers after the attribute offset: an integer and a long integer. The integer is a mark: 0 means that the polygon is just an ordinary polygon, 1 means that the polygon interpolates normals, 2 means that the polygon is part of a polygonal approximation of a sphere, 3 means that the polygon is part of a polygonal approximation of a cylinder, and 4 means that the polygon is part of a polygonal approximation of a cone. If the mark is 2, 3 or 4, the long integer following it is the index number of the appropriate shape. See also 2.2 and 2.4 above.

Filetype 1 means that the optical surface file is a **Phigs+** file. Note that polygons in Phigs files cannot be smooth-shaded or marked.

Filetype 2 means that the optical surface file is one produced by the polygonizer of Fritz Prinz's Group at CMU. Following is an example:

```
{solid
                {face {loop 1 7 3}}
                {node {position  1.0 1.0 1.0}}
                {node {position  -1.0 1.0 1.0}}
                {face {loop 0 1 2}}
                {node {position  1.0 -1.0 1.0}}
                {face {loop 0 2 6}}
                {face {loop 2 5 6}}
                {face {loop 5 7 6}}
                {node {position  -1.0 -1.0 1.0}}
                {node {position  -1.0 -1.0 -1.0}}
                {face {loop 3 7 4}}
                {face {loop 4 7 5}}
                {node {position  1.0 -1.0 -1.0}}
                {face {loop 1 3 2}}
                {face {loop 1 0 6}}
                {node {position  1.0 1.0 -1.0}}
                {face {loop 1 6 7}}
                {face {loop 2 3 4}}
                {face {loop 2 4 5}}
                {node {position  -1.0 1.0 -1.0}}
}
```

{node {position 1.0 1.0 1.0}} is a vertex description; the three floating point numbers are the coordinates of the vertex. **{face {loop 1 7 3}}** is a polygon description; the three numbers are the index numbers of the vertices of the polygon in counterclockwise order. (The order is important since it determines which direction is "outside.") The index number is the number of vertex descriptions that appear before the vertex in question. For example, to find vertex 3, go down the file until the fourth "node."

IMPORTANT: *the polygons in a file of filetype 2 MUST be triangles.*

There is at present no way of marking polygons when the filetype is 2. Note also that the format of Chen's files is expected to change sometime in the future, to something more like filetype zero.

PART THREE: THE INTERSECTION PROBLEM

3.1 Octrees, spatial encoding, and profiling

An eventual goal of the project is to develop hardware to do fast ray-polygon intersection. Spatial encoding in the CMU raytracer is to verify the technique, as well as being somewhat useful as is.

All methods subdivide the world-box into smaller boxes (**voxels**), each of which contains a list of polygons. The basic idea is to use a differential analyzer (**ddda**), similar to that used in raster line-drawing, to tell us which voxels we have to look in. The hope is that, for fine enough subdivisions, the boxes that have anything in them have a high probability of containing an actual intersecting polygon.

One possible problem with the method is that of near-hits. This is where we check a voxel and get a polygon intersection, but the intersection point isn't in that voxel. In this case, there's a possibility that a subsequent voxel contains an intersection with a different polygon. This is an arbitrarily bad problem if the ray is moving alongside a large polygon. By default, such near hits are ignored. You can choose to accept near hits by specifying the **-on** switch, or limit the number that can occur with the **-tv** switch. For low octree depths (5-7), this probably isn't worth bothering with.

PART FOUR: BUGS

4.1 Switches

The -I and -T switches should probably be on by default.

APPENDIX: COMMAND LINE SWITCHES

Running rt without any arguments gives you a list of switches. This appendix explains them in greater depth.

-n *No output file*

This switch disables the production of a .gro file. This is probably useful only in testing or very short runs.

-w *Wireframe only, no raytracing*

This is probably used only in front-end user interfaces, since you can just as easily interrupt the program.

-m2.0 *Multiplies screen display values by 2.*

If the picture is too dark or too bright, you can use this switch to compensate. This is the only current option for mapping internal intensities to good (i.e., visible) display values. You may want to use this in conjuction with the -A option in order to get a good spread of values. Note that RGB values cannot exceed 255, so if this multiplication causes an intensity to exceed 255, the RGB values are scaled down proportionally to stay within this limit (that is, provided the -c switch is not used).

-tv *Terminate rays after N fruitless voxel hits*

Mainly useful to the hardware designers, and for large octree spatial subdivisions in any case.

-c *Toggle intensity clipping. Currently on by default.*

Specifying -c turns off clipping: if an intensity exceeds the 255 limit, it goes ahead and displays it anyway. Can give very strange colors sometimes.

-od *Specify octree depth (overrides scene file)*

May be of use to programs/people studying the effects of different octree depths.

-oo *Draw (output) octree boxes on the wireframe*

To see what the octree building machinery is doing.

-ol *Specify the length of the octlist, as a power of 2*

Useful to either

1) Specify a larger octree storage area (only happens for large octree depths); or

2) Specify a small octree storage area, for machines that allocate expensively (like the Cray).

-on *Don't verify near hits (just accept them)*

Probably a bad idea except for large octree depths, and maybe not even then. The default is to ignore them.

-of *Don't fudge (ddda is precise enough)*

This should be calculated automatically, but isn't. See part three for details.

-op *Use new fillplane routines.*

The old octree building routines were order n^3, this is n^2 (n is octree levels). It's still not completely trusted, and it's not necessary for small octree depths.

-ss *Specify the seads size as a power of 2*

The scene file specifies this as the actual number, and this probably should too.

-p *Profile ddda operation.*

Prints out various internal metrics at the end of the run. Useful with the -r option.

-r *Fire random rays.*

Doesn't produce a .gro file, so this is only useful with the -p option or in debugging.

-S *(mutually exclusive) specular reflection models:*

-S must be followed by another letter; by itself it is meaningless.

-Sp *Phong*

-Sg *Phong with geometric attenuation factor*

-Stg1.0 *Torrance-Sparrow with Gaussian dist.; the # is m*

-Stb1.0 *Torrance-Sparrow with Beckmann dist.; the # is m*

The different -S switches give different specular highlights. You cannot use more than one of these switches in a single command line. The default is Phong. The differences between these models are fairly subtle; see references [1] to [5] for more details. "m" is the root mean square slope of the microfacets.

-R *Each attribute in .scn file has 3 (RGB) indices of refraction*

See section 2.3 in the manual.

-Wr *Warn lights: Rogers*

Only applicable when there is shadow feeling. The intensity of the shadow feeler is attenuated by pow(cos(x),N) where x is the angle between the shadow feeler and the normal to the emissive polygon and N is the fourth number in the attribute description (see section 2.3 and reference [1]).

-Wc 0.7 *Warn lights: conical; # is angle in radians: must be 2.*

Same as -Wr except if x is greater than the specified angle, no light is obtained.

-Wt1.0 *Warn lights: Thibadeau; # is dist. from poly*

The point source associated with the emissive polygons is displaced by the specified distance away from the polygon in the direction of the negative of the normal to the polygon. During shadow feeling, the point of intersection is connected to this displaced point; if this line intersects the polygon, we behave like -Wr, and otherwise we get no light.

-A40.0,1.0 *Attenuation effect: intensity divided by dist/40.0 + 1.0*

The intensity of a light ray is attenuated with distance according to the formula. Notice that this is NOT the inverse-square law. The inverse square law usually produces pictures with excessive contrast. By default, attenuation is turned on, with parameters 200.0 and 1.0.

-T *Specular refraction*

This should probably be a default. Currently, the program only considers shadow feelers that are on the same side of the polygon as the outward- pointing normal. -T makes the program consider shadow feelers on the other side if the polygon is not opaque.

-C *Color of specular highlights:*

Only one of the -C switches can appear on each command line.-C must be followed by another letter; by itself it is meaningless.

-Co *Object color attenuates light color*

Specular highlights are filtered through the color of the polygon.

-Cl *Light color only*

Specular highlights are the color of the incident light.

-Cf *Fresnel function applied to RGB components separately*

Specular highlights vary according to the wavelength of the incident light. See reference [1].

-D *disable (not mutually exclusive):*

-D by itself is meaningless; it must be followed by another letter.

-Dx *disable adaptive supersampling*

See the -X switches. This forces the program to take the maximum number of samples per pixel (the number given in the .scn file squared; see section 2.2).

-De *disable exit ray culling*

Exit ray culling is a method of reducing the number of rays that leave the world without hitting anything. The problem is that some preprocessing needs to be done. This option might come in handy if you have so many polygons that the setup time is prohibitive.

-Dd *disable Distributing*

See section 1.3. Specifying -Dd means that only one transmitted and one reflected ray is generated at each ray-polygon intersection. Use this option for quick (but unrealistic) pictures.

-Dt *prevent shadow feelers from penetrating Transparent polys*

By default, if a shadow feeler encounters a polygon on its way to the light, it continues through the polygon unless the polygon is opaque. Specifying -Dt forces the shadow feeler to stop dead at the first polygon it hits (regardless of whether the polygon is transparent or not).

-Da *disable Attenuation with distance (excluding LightLock)*

Specifying -Da means that the light rays (except for the shadow feelers) do not attenuate with distance.

-Dl *disable attenuation with distance of Lights*

Specifying -Dl means that shadow feelers do not attenuate with distance.

-Ds *disable Shadow feeler*

Eliminate shadow feeling altogether. This is the most physically correct option, but in practice it will take days to produce a satisfactory image.

-Dr *disable Reflection*

Don't generate reflected rays, even at a polygon that is reflective.

-Df *read attributes from .scn and not separate File -Dp Power fields not present in attribute description*

See section 2.3.

-Db *not Both sides of polygons are emissive*

Specifying -Db means that only the "outside" surface of emissive polygons emits light.

-P *Phong shading field present in .scn file:*

See section 2.2. You cannot use both -Po and -Pt in the same command line. -P without an "o" or a "t" immediately after it is meaningless.

-Po *Phong shaded objects are opaque*

Forces these objects to be opaque, overriding the attribute description. This is the preferred option.

-Pt *Phong shaded objects are transparent*

Do not force opaqueness. (Doesn't force transparency, either.)

-Ot *Old intersection test*

Smooth-shading can cause problems with intersection testing. The default is to correct these problems. -Ot disables this correction.

-Oi *Old illumination model*

Pretty much useless. -Oi makes the program simulate the earliest version of the raytracer, which had several bugs in it.

-Of *Old format*

Same as -Df -Dp.

-K *scale down ka, ks, etc. so that intensity need not be clipped (looks at -Dr and -Ds switches)*

Not the default. Produces correct but dark pictures.

-I *apply illum. model to distrib. rays as well as shadow feelers*

Should probably be the default. Currently, the distributed rays are not given the full illumination model treatment. -I ensures that they are calculated correctly.

-L250 *draws only from line 250 up*

-E1.5 *Sharp breaks in Phong-shaded objects if angle 1.5 radians*

Not fully tested. This is the smooth-shading/intersection problem again.

-F0.3,2.0 *Fuzzy shadow feeler: 0.3 is distangle, 2.0 is distfac*

Not fully tested. It distributes the shadow feeler in an attempt to produce fuzzy shadows.

-X *Chi-square adaptive sampling:*

The idea of adaptive sampling is to keep sampling until the variance of the samples is sufficiently low. (Note: -X on its own is meaningless.)

-Xp0.1,0.05 *Stop sampling pixels when prob that variance0.05 is .1*

-Xr0.1,0.05 *Ditto, except with distributed rays and not pixels*

The defaults for the maximum variance and the probability are all 0.2.

-M op surf files mark polys; *Shape file name in .scn file (doesn't work with Phigs or Chen's files)*

See section 2.4, and also sections 2.2 and 2.5.

-V *texture mapping*

See section 2.2.

-d *debug switches:*

-d by itself is meaningless.

-dc *print column numbers*

Prints the column number of the pixel on each sample. This is on by default on the PC, since it's interruptible only when doing i/o.

-ds *(strike) debug ddda by comparing actual results (slow)*

Only useful for debugging octree or SEADS ray-plane intersection. Does it the slow way, testing each polygon, and compares the results.

-dsp *(strike print) print hits different only in poly, but not oct*

Useful in conjuction with -ds. For errorful ray-plane intersections, does them again and traces the ddda operation.

-dso *(strike oct) print octs looked in*

Unused.

-di *print intersections*

Only prints intersections done the old way (no spatial encoding).

-doo *Output the octree to octree.lst*

Prints a text file with polygon id's. Not used much.

-doi *Debug octree interactively*

Doesn't work, and causes other things to break.

-dof *Debug octree fillplane*

Used to debug the new fillplane routines; currently does nothing.

-df (file) *debug file parsing*

Used in debugging object file parsing.

-dd *debug color info*

On the apollos, prints the color map and the pixel values as they're output.

-dx *Debug chi-square stuff*

Prints the number of samples at each pixel and ray-plane intersection.

-de *Debug exit ray culling*

Checks culled rays to see that they really don't intersect anything. Also prints statistics at end. Useful in conjunction with **-p.**

References:

[1] *Procedural Elements for Computer Graphics* by David F. Rogers. (McGraw-Hill 1985)

[2] "A Reflectance Model for Computer Graphics," *ACM Transactions on Graphics,* 1, 1 (Jan. 1982), 7-24.

[3] "Illumination for Computer Generated Pictures," *Communications of the ACM*, 18, 6 (June 1975), 311-317.

[4] "Models of Light Reflection for Computer Synthesized Pictures," *Proceedings of SIGGRAPH '77* (San Jose, Calif., July 20-22, 1977).

[5] *A Reflection Model for Realistic Image Synthesis,* Master's Thesis, Cornell University, Ithaca, N.Y., 1981.

[6] Fujimoto, Akira, Takayuki Tanaka and Kansei Iwata, "ARTS: Accelerated Ray-Tracing System", *IEEE Computer Graphics and Applications*, 6(4), April 1986, 16-26.