The RADIANCE 2.2
Synthetic Imaging System

Greg Ward
Lawrence Berkeley Laboratory
1 Cyclotron Rd.
Berkeley, CA 94720
(510) 486-4757

1. Introduction

RADIANCE was developed as a research tool for predicting the distribution of visible radiation in illuminated spaces. It takes as input a three-dimensional geometric model of the physical environment, and produces a map of spectral radiance values in a color image. The technique of ray-tracing follows light backwards from the image plane to the source(s). Because it can produce realistic images from a simple description, RADIANCE has a wide range of applications in graphic arts, lighting design, computer-aided engineering and architecture.

Figure 1.

The diagram in Figure 1 shows the flow between programs (boxes) and data (ovals). The central program is *rpict,* which produces a picture from a scene description. *Rview* is a variation of *rpict* that computes and displays images interactively.

A scene description file lists the surfaces and materials that make up a specific environment. The current surface types are spheres, polygons, cones, and cylinders. They can be made from materials such as plastic, metal, and glass. Light sources can be distant disks as well as local spheres, discs and polygons.

From a three-dimensional scene description and a specified view, *rpict* produces a two-dimensional image. A picture file is a compressed binary representation of the pixels in the image. This picture can be scaled in size and brightness, anti-aliased, and sent to a graphics output device.

A header in each picture file lists the program(s) and parameters that produced it. This is useful for identifying a picture without having to display it. The information can be read by the program *getinfo.*

2. Scene Description

A scene description file represents a three-dimensional physical environment in Cartesian (rectilinear) world coordinates. It is stored as ascii text, with the following basic format:

```
# comment

modifier type identifier
n S1 S2 S3 .. Sn
0
m R1 R2 R3 .. Rm

modifier alias identifier reference

! command

 ...
```

A comment line begins with a pound sign, ‘#’.

The scene description *primitives* all have the same general format, and can be either surfaces or modifiers. A primitive has a modifier, a type, and an identifier. A modifier is either the identifier of a *previously defined* primitive, or "void"†. An identifier can be any string (ie. sequence of non-blank characters). The *arguments* associated with a primitive can be strings or real numbers. The first integer following the identifier is the number of string arguments, and it is followed by the arguments themselves (separated by white space). The next integer is the number of integer arguments, and is followed by the integer arguments. (There are currently no primitives that use them, however.) The next integer is the real argument count, and it is followed by the real arguments.

An alias gets its type and arguments from a previously defined primitive. This is useful when the same material is used with a different modifier, or as a convenient naming mechanism. Surfaces cannot be aliased.

A line beginning with an exclamation point, ‘!’, is interpreted as a command. It is executed by the shell, and its output is read as input to the program. The command must not try to read from its standard input, or confusion will result. A command may be continued over multiple lines using a backslash, ‘\’, to escape the newline.

Blank space is generally ignored, except as a separator. The exception is the newline character after a command or comment. Commands, comments and primitives may appear in any combination, so long as they are not intermingled.

2.1. Primitive Types

Primitives can be surfaces, materials, textures or patterns. Modifiers can be materials, textures or patterns. Simple surfaces must have one material in their modifier list.

†The most recent definition of a modifier is the one used, and later definitions do not cause relinking of loaded primitives. Thus, the same identifier may be used repeatedly, and each new definition will apply to the primitives following it.

2.1.1. Surfaces

A scene description will consist mostly of surfaces. The basic types are given below.

Source

A source is not really a surface, but a solid angle. It is used for specifying light sources that are very distant. The direction to the center of the source and the number of degrees subtended by its disk are given as follows:

```
mod source id
0
0
4 xdir ydir zdir angle
```

Sphere

A sphere is given by its center and radius:

```
mod sphere id
0
0
4 xcent ycent zcent radius
```

Bubble

A bubble is simply a sphere whose surface normal points inward.

Polygon

A polygon is given by a list of three-dimensional vertices, which are ordered counter-clockwise as viewed from the front side (into the surface normal). The last vertex is automatically connected to the first. Holes are represented in polygons as interior vertices connected to the outer perimeter by coincident edges (seams).

```
mod polygon id
0
0
3n
        x1    y1    z1
        x2    y2    z2
        ...
        xn    yn    zn
```

Cone

A cone is a megaphone-shaped object. It is truncated by two planes perpendicular to its axis, and one of its ends may come to a point. It is given as two axis endpoints, and the starting and ending radii:

```
mod cone id
0
0
8
        x0    y0    z0
        x1    y1    z1
        r0    r1
```

Cup

A cup is an inverted cone (ie. has an inward surface normal).

Cylinder

A cylinder is like a cone, but its starting and ending radii are equal.

```
mod cylinder id
0
0
7
        x0      y0      z0
        x1      y1      z1
        rad
```

Tube

A tube is an inverted cylinder.

Ring

A ring is a circular disk given by its center, surface normal, and inner and outer radii:

```
mod ring id
0
0
8
        xcent ycent zcent
        xdir  ydir  zdir
        r0    r1
```

Instance

An instance is a compound surface, given by the contents of an octree file (created by oconv).

```
mod instance id
1+ octree transform
0
0
```

If the modifier is "void", then surfaces will use the modifiers given in the original description. Otherwise, the modifier specified is used in their place. The transform moves the octree to the desired location in the scene. Multiple instances using the same octree take little extra memory, hence very complex descriptions can be rendered using this primitive.

There are a number of important limitations to be aware of when using instances. First, the scene description used to generate the octree must stand on its own, without referring to modifiers in the parent description. This is necessary for oconv to create the octree. Second, light sources in the octree will not be incorporated correctly in the calculation, and they are not recommended. Finally, there is no advantage (other than convenience) to using a single instance of an octree, or an octree containing only a few surfaces. An xform command on the subordinate description is prefered in such cases.

2.1.2. Materials

A material defines the way light interacts with a surface. The basic types are given below.

Light

Light is the basic material for self-luminous surfaces (ie. light sources). In addition to the source surface type, spheres, discs (rings with zero inner radius), cylinders (provided they are long enough), and polygons can act as light sources. Polygons work best when they are rectangular. Cones cannot be used at this time. A pattern may be used to specify a light output distribution. Light is defined simply as a RGB radiance value (watts/rad2/m2):

```
mod light id
0
0
3 red green blue
```

Illum

Illum is used for secondary light sources with broad distributions. A secondary light source is treated like any other light source, except when viewed directly. It then acts like it is made of a different material, or becomes invisible. Secondary sources are useful when modeling windows or brightly illuminated surfaces.

```
mod illum id
1 material
0
3 red green blue
```

Glow

Glow is used for surfaces that are self-luminous, but limited in their effect. In addition to the radiance value, a maximum radius for shadow testing is given:

```
mod glow id
0
0
4 red green blue maxrad
```

If maxrad is zero, then the surface will never be tested for shadow, although it may participate in an interreflection calculation. If maxrad is negative, then the surface will never contribute to scene illumination.

Spotlight

Spotlight is used for self-luminous surfaces having directed output. As well as radiance, the full cone angle (in degrees) and orientation (output direction) vector are given. The length of the orientation vector is the distance of the effective focus behind the source center (ie. the focal length).

```
mod spotlight id
0
0
7 red green blue angle xdir ydir zdir
```

Mirror

Mirror is used for planar surfaces that produce secondary source reflections. This material should be used sparingly, as it may cause the light source calculation to blow up if it is applied to many small surfaces. This material is only supported for flat surfaces such as polygons and rings. The arguments are simply the RGB reflectance values, which should be between 0 and 1. An optional string argument may be used like the illum type to specify a different material to be used for shading non-source rays.

```
mod mirror id
1 material
0
3 red green blue
```

Prism1

The prism1 material is for general light redirection from prismatic glazings, generating secondary light sources. It can only be used to modify a planar surface (ie. a polygon or disk) and should not result in either light concentration or scattering. The new direction of the ray can be on either side of the material, and the definitions must have the correct bidirectional properties to work properly with secondary light sources. The arguments give the coefficient for the redirected light and its direction.

```
mod prism1 id
5+ coef dx dy dz funcfile transform
0
n A1 A2 .. An
```

The new direction variables *dx, dy* and *dz* need not produce a normalized vector. For convenience, the variables *DxA*, *DyA* and *DzA* are defined as the normalized direction to the target light source. See section 2.2.1 on function files for further information.

Prism2

The material prism2 is identical to prism1 except that it provides for two ray redirections rather than one.

```
mod direct1 id
9+ coef1 dx1 dy1 dz1 coef2 dx2 dy2 dz2 funcfile transform
0
n A1 A2 .. An
```

Plastic

Plastic is a material with uncolored highlights. It is given by its RGB reflectance, its fraction of specularity, and its roughness value. Roughness is specified as the rms slope of surface facets. A value of 0 corresponds to a perfectly smooth surface, and a value of 1 would be a very rough surface. Specularity fractions greater than 0.1 and roughness values greater than 0.2 are not very realistic. (A pattern modifying plastic will affect the material color.)

```
mod plastic id
0
0
5 red green blue spec rough
```

Metal

Metal is similar to plastic, but specular highlights are modified by the material color. Specularity of metals is usually .9 or greater. As for plastic, roughness values above .2 are uncommon.

Trans

Trans is a translucent material, similar to plastic. The transmissivity is the fraction of penetrating light that travels all the way through the material. The transmitted specular component is the fraction of transmitted light that is not diffusely scattered. Diffusely scattered light is modified by the material color. Translucent objects are infinitely thin.

```
mod trans id
0
0
7 red green blue spec rough trans tspec
```

Plastic2

Plastic2 is similar to plastic, but with anisotropic roughness. This means that highlights in the surface will appear elliptical rather than round. The orientation of the anisotropy is determined by the unnormalized direction vector *ux uy uz*. These three expressions (separated by white space) are evaluated in the context of the function file *funcfile*. If no function file is required (ie. no special variables or functions are required), a period ('.') may be given in its place. (See the discussion of Function Files in the Auxiliary Files section). The *urough* value defines the roughness along the **u** vector given projected onto the surface. The *vrough* value defines the roughness perpendicular to this vector. Note that the highlight will be narrower in the direction of the smaller roughness value. Roughness values of zero are not allowed for efficiency reasons since the behavior would be the same as regular plastic in that case.

```
mod plastic2 id
4+ ux uy uz funcfile transform
0
6 red green blue spec urough vrough
```

Metal2

Metal2 is the same as plastic2, except that the highlights are modified by the material color.

Trans2

Trans2 is the anisotropic version of trans. The string arguments are the same as for plastic2, and the real arguments are the same as for trans but with an additional roughness value.

```
mod trans2 id
4+ ux uy uz funcfile transform
0
8 red green blue spec urough vrough trans tspec
```

Dielectric

A dielectric material is transparent, and it refracts light as well as reflecting it. Its behavior is determined by the index of refraction and transmissivity in each wavelength band per unit length. Common glass has a index of refraction (n) around 1.5, and a transmissivity of roughly 0.92 over an inch. An additional number, the Hartmann constant, describes how the index of refraction changes as a function of wavelength. It is usually zero. (A pattern modifies only the refracted value.)

```
mod dielectric id
0
0
5 rtn gtn btn n hc
```

Interface

An interface is a boundary between two dielectrics. The first transmissivities and refractive index are for the inside; the second ones are for the outside. Ordinary dielectrics are surrounded by a vacuum (1 1 1 1).

```
mod interface id
0
0
8 rtn1 gtn1 btn1 n1 rtn2 gtn2 btn2 n2
```

Glass

Glass is similar to dielectric, but it is optimized for thin glass surfaces (n = 1.52). One transmitted ray and one reflected ray is produced. By using a single surface is in place of two, internal reflections are avoided. The surface orientation is irrelevant, as it is for plastic, metal, and trans. The only specification required is the transmissivity at normal incidence. To compute transmissivity (tn) from transmittance (Tn) use:

```
tn = (sqrt(.8402528435+.0072522239*Tn*Tn)-.9166530661)/.0036261119/Tn
```

Standard 88% transmittance glass has a transmissivity of 0.96. (A pattern modifying glass will affect the transmissivity.) If a fourth real argument is given, it is interpreted as the index of refraction to use instead of 1.52.

```
mod glass id
0
0
3 rtn gtn btn
```

Plasfunc

Plasfunc in used for the procedural definition of plastic-like materials with arbitrary bidirectional reflectance distribution functions (BRDF's). The arguments to this material include the color and specularity, as well as the function defining the specular distribution and the auxiliary file where it may be found.

```
mod plasfunc id
2+ refl funcfile transform
0
4+ red green blue spec A5 ..
```

The function *refl* must take three arguments, the x, y and z direction towards the incident light, and should integrate to 1 over the projected hemisphere. At least four real arguments must be given, and these are made available along with any additional values to the reflectance function. Currently, only the contribution from direct light sources is considered in the specular calculation.

Metfunc

Metfunc is identical to plasfunc and takes the same arguments, but the specular component is multiplied also by the material color.

Transfunc

Transfunc is similar to plasfunc but with an arbitrary bidirectional transmittance distribution as well as a reflectance distribution. Both reflectance and transmittance are specified with the same function.

```
mod transfunc id
2+ refl funcfile transform
0
4+ red green blue rspec trans tspec A7 ..
```

Where *trans* is the total light transmitted and *tspec* is the non-Lambertian fraction of transmitted light. The function *refl* should integrate to 1 over each projected hemisphere.

BRTDfunc

The material BRTDfunc gives the maximum flexibility over surface reflectance and transmittance, providing for spectrally-dependent specular rays and reflectance and transmittance distribution functions.

```
mod BRTDfunc id
10+  rrefl  grefl  brefl
     rtrns  gtrns  btrns
     rbrtd  gbrtd  bbrtd
     funcfile  transform
0
6+ red green blue rspec trans tspec A7 ..
```

The variables *rrefl, grefl* and *brefl* specify the color coefficients for the ideal specular (mirror) reflection of the surface. These values are not modified by *rspec,* thus the diffuse reflectances *red, green* and *blue* implicitly exclude the mirrored specular component and should be set accordingly. Similarly, the variables *rtrns, gtrns* and *btrns* specify the color coefficients for the transmitted direction. These values are modified by the total transmittance, *trans,* but not by *tspec.* The functions *rbrtd, gbrtd* and *bbrtd* take the direction to the incident light and compute the color coefficient for the non-Lambertian part of reflection and transmission. These functions are modified by *rspec* and *tspec* appropriately, and they are expected to integrate to 1 over each projected hemisphere.

Plasdata

Plasdata is used for arbitrary BRDF's that are most conveniently given as interpolated data. The arguments to this material are the data file and coordinate index functions, as well as a function to optionally modify the data values.

```
mod plasdata id
3+n+
        func datafile
        funcfile x1 x2 .. xn transform
0
4+ red green blue spec A5 ..
```

The coordinate indices *(x1, x2,* etc.) are themselves functions of the x, y and z direction to the incident light. The data function *(func)* takes a single variable, which is the interpolated value from the n-dimensional data file.

Metdata

As metfunc is to plasfunc, metdata is to plasdata. Metdata takes the same arguments as plasdata, but the specular component is modified by the given material color.

Transdata

Transdata is like plasdata but the specification includes transmittance as well as reflectance. The parameters are as follows.

```
mod transdata id
3+n+
        func datafile
        funcfile x1 x2 .. xn transform
0
4+ red green blue rspec trans tspec A7 ..
```

Antimatter

Antimatter is a material that can "subtract" volumes from other volumes. A ray passing into an antimatter object becomes blind to all the specified modifiers:

```
mod antimatter id
N mod1 mod2 .. modN
0
0
```

The first modifier will be used to shade the area leaving the antimatter volume and entering the regular volume. If mod1 is void, the antimatter volume is completely invisible. Antimatter does not work properly with the material type "trans", and multiple antimatter surfaces should be disjoint. The viewpoint must be outside all volumes concerned for a correct rendering.

2.1.3. Textures

A texture is a perturbation of the surface normal, and is given by either a function or data.

Texfunc

A texfunc uses an auxiliary function file to specify a procedural texture:

```
mod texfunc id
4+ xpert ypert zpert funcfile transform
0
n A1 A2 .. An
```

Texdata

A texdata texture uses three data files to get the surface normal perturbations. The variables *xfunc, yfunc* and *zfunc* take three arguments each from the interpolated values in *xdfname, ydfname* and *zdfname.*

```
mod texdata id
8+ xfunc yfunc zfunc xdfname ydfname zdfname vfname x0 x1 .. xf
0
n A1 A2 .. An
```

2.1.4. Patterns

Patterns are used to modify the reflectance of materials. The basic types are given below.

Colorfunc

A colorfunc is a procedurally defined color pattern. It is specified as follows:

```
mod colorfunc id
4+ red green blue funcfile transform
0
n A1 A2 .. An
```

Brightfunc

A brightfunc is the same as a colorfunc, except it is monochromatic.

```
mod brightfunc id
2+ refl funcfile transform
0
n A1 A2 .. An
```

Colordata

Colordata uses an interpolated data map to modify a material's color. The map is n-dimensional, and is stored in three auxiliary files, one for each color. The coordinates used to look up and interpolate the data are defined in another auxiliary file. The interpolated data values are modified by functions of one or three variables. If the functions are of one variable, then they are passed the corresponding color component (red or green or blue). If the functions are of three variables, then they are passed the original red, green, and blue values as parameters.

```
mod colordata id
7+n+
        rfunc gfunc bfunc rdatafile gdatafile bdatafile
        funcfile x1 x2 .. xn transform
0
m A1 A2 .. Am
```

Brightdata

Brightdata is like colordata, except monochromatic.

```
mod brightdata id
3+n+
        func datafile
        funcfile x1 x2 .. xn transform
0
m A1 A2 .. Am
```

Colorpict

Colorpict is a special case of colordata, where the pattern is a two-dimensional image stored in the RADIANCE picture format. The dimensions of the image data are determined by the picture such that the smaller dimension is always 1, and the other is the ratio between the larger and the smaller. For example, a 500x338 picture would have coordinates (u,v) in the rectangle between (0,0) and (1.48,1).

```
mod colorpict id
7+
        rfunc gfunc bfunc pictfile
        funcfile u v transform
0
m A1 A2 .. Am
```

Colortext

Colortext is dichromatic writing in a polygonal font. The font is defined in an auxiliary file, such as *helvet.fnt.* The text itself is also specified in a separate file, or can be part of the material arguments. The character size, orientation, aspect ratio and slant is determined by right and down motion vectors. The upper left origin for the text block as well as the foreground and background colors must also be given.

```
mod colortext id
2 fontfile textfile
0
15+
        Ox Oy Oz
        Rx Ry Rz
        Dx Dy Dz
        rfore gfore bfore
        rback gback bback
        [spacing]
```

or:

```
mod colortext id
2+N fontfile . This is a line with N words ...
0
15+
        Ox Oy Oz
        Rx Ry Rz
        Dx Dy Dz
        rfore gfore bfore
        rback gback bback
        [spacing]
```

Brighttext

Brighttext is like colortext, but the writing is monochromatic.

```
mod brighttext id
2 fontfile textfile
0
11+
        Ox Oy Oz
        Rx Ry Rz
        Dx Dy Dz
        foreground background
        [spacing]
```

or:

```
mod brighttext id
2+N fontfile . This is a line with N words ...
0
11+
        Ox Oy Oz
        Rx Ry Rz
        Dx Dy Dz
        foreground background
        [spacing]
```

By default, a uniform spacing algorithm is used that guarantees every character will appear in a precisely determined position. Unfortunately, such a scheme results in rather unattractive and difficult to read text with most fonts. The optional *spacing* value defines the distance between characters for proportional spacing. A positive value selects a spacing algorithm that preserves right margins and indentation, but does not provide the ultimate in proportionally spaced text. A negative value insures that characters are properly spaced, but the placement of words then varies unpredictably. The choice depends on the relative importance of spacing versus formatting. When presenting a section of formatted text, a positive spacing value is usually preferred. A single line of text will often be accompanied by a negative spacing value. A section of text meant to depict a picture, perhaps using a special purpose font such as hexbit4x1.fnt, calls for uniform spacing. Reasonable magnitudes for proportional spacing are between 0.1 (for tightly spaced characters) and 0.3 (for wide spacing).

2.1.5. Mixtures

A mixture is a blend of one or more textures and patterns. The basic types are given below.

Mixfunc

A mixfunc mixes two modifiers procedurally. It is specified as follows:

```
mod mixfunc id
4+ foreground background vname funcfile transform
0
n A1 A2 .. An
```

Foreground and background are modifier names that must be uniquely defined in the scene description. Vname is the coefficient defined in funcfile that determines the influence of foreground. The background coefficient is always (1-vname). Since the references are not resolved until runtime, the last definitions of the modifier id's will be used. This can result in modifier loops, which are detected by the renderer.

Mixdata

Mixdata combines two modifiers using an auxiliary data file:

```
mod mixdata id
5+n+
        foreground background func datafile
        funcfile x1 x2 .. xn transform
0
m A1 A2 .. Am
```

Mixtext

Mixtext uses one modifier for the text foreground, and one for the background:

```
mod mixtext id
4 foreground background fontfile textfile
0
9+
        Ox Oy Oz
        Rx Ry Rz
        Dx Dy Dz
        [spacing]
```

or:

```
mod mixtext id
4+N
        foreground background fontfile .
        This is a line with N words ...
0
9+
        Ox Oy Oz
        Rx Ry Rz
        Dx Dy Dz
        [spacing]
```

2.2. Auxiliary Files

Auxiliary files used in textures and patterns are accessed by the programs during image generation. These files may be located in the working directory, or in a library directory. The environment variable *RAYPATH* can be assigned an alternate set of search directories. Following is a brief description of some common file types.

2.2.1. Function Files

A function file contains the definitions of variables, functions and constants used by a primitive. The transformation that accompanies the file name contains the necessary rotations, translations and scalings to bring the coordinates of the function file into agreement with the world coordinates. The transformation specification is the same as for the *xform* command. An example function file is given below:

```
{
        This is a comment, enclosed in curly braces.
        {Comments can be nested.}
}
                        { standard expressions use +,-,*,/,^,(,) }
vname = Ny * func(A1) ;
                        { constants are defined with a colon }
const : sqrt(PI/2) ;
                        { user-defined functions add to library }
func(x) = 5 + A1*sin(x/3) ;
                        { functions may be passed and recursive }
rfunc(f,x) = if(x,f(x),f(-x)*rfunc(f,x+1)) ;
                        { constant functions may also be defined }
cfunc(x) : 10*x / sqrt(x) ;
```

Many variables and functions are already defined by the program, and they are listed in the file *rayinit.cal.* The following variables are particularly important:

Dx, Dy, Dz	- incident ray direction
Px, Py, Pz	- intersection point
Nx, Ny, Nz	- surface normal at intersection point
Rdot	- cosine between ray and normal
arg(0)	- number of real arguments
arg(i)	- i'th real argument

For BRDF types, the following variables are defined as well:

NxP, NyP, NzP	- perturbed surface normal
RdotP	- perturbed dot product
CrP, CgP, CbP	- perturbed material color

A unique context is set up for each file so that the same variable may appear in different function files without conflict. The variables listed above and any others defined in rayinit.cal are available globally. If no file is needed by a given primitive because all the required variables are global, a period ('.') can be given in place of the file name. It is also possible to give an expression instead of a straight variable name in a scene file, although such expressions should be kept simple as they cannot contain any white space. Also, functions (requiring parameters) must be given as names and not as expressions.

Constant expressions are used as an optimization in function files. They are replaced wherever they occur in an expression by their value. Constant expressions are evaluated only once, so they must not contain any variables or values that can change, such as the ray variables Px and Ny or the primitive argument function arg(). All the math library functions such as sqrt() and cos() have the constant attribute, so they will be replaced by immediate values whenever they are given constant arguments. Thus, the subexpression cos(PI*sqrt(2)) is immediately replaced by its value, -.266255342, and does not cause any additional overhead in the calculation.

It is generally a good idea to define constants and variables before they are referred to in a function file. Although evaluation does not take place until later, the interpreter does variable scoping and constant subexpression evaluation based on what it has compiled already. For example, a variable that is defined globally in rayinit.cal then referenced in the local context of a function file cannot subsequently be redefined in the same file because the compiler has already determined the scope of the referenced variable as global. To avoid such conflicts, one can state the scope of a variable explicitly by preceding the variable name with a context mark (a back-quote) for a local variable, or following the name with a context mark for a global variable.

2.2.2. Data Files

Data files contain n-dimensional arrays of real numbers used for interpolation. Typically, definitions in a function file determine how to index and use interpolated data values. The basic data file format is as follows:

```
N
beg1 end1 m1
0 0 m2 x2.1 x2.2 x2.3 x2.4 .. x2.m2
 ...
begN endN mN
DATA, later dimensions changing faster.
```

N is the number of dimensions. For each dimension, the beginning and ending coordinate values and the dimension size is given. Alternatively, individual coordinate values can be given when the points are not evenly spaced. These values must either be increasing or decreasing monotonically. The data is m1*m2*...*mN real numbers in ascii form. Comments are not allowed in data files.

2.2.3. Font Files

A font file lists the polygons which make up a character set. There are no comments, and all numbers are decimal integers:

```
code n
        x0 y0
        x1 y1
         ...
        xn yn
 ...
```

The ascii codes can appear in any order. N is the number of vertices, and the last is automatically connected to the first. Separate polygonal sections are joined by coincident sides. The character coordinate system is a square with lower left corner at (0,0), lower right at (255,0) and upper right at (255,255).

2.3. Generators

A generator is any program that produces a scene description as its output. They usually appear as commands in a scene description file. An example of a simple generator is *genbox*. *Genbox* takes the arguments of width, height and depth to produce a parallelepiped description. *Genrev* is a more sophisticated generator that produces an object of rotation from parametric functions for radius and axis position. *Gensurf* tessellates a surface defined by the parametric functions x(s,t), y(s,t), and z(s,t). *Genworm* links cylinders and spheres along a curve. *Gensky* produces a sun and sky distribution corresponding to a given time and date.

Xform is a program that transforms a scene description from one coordinate space to another. *Xform* does rotation, translation, scaling, and mirroring.

3. Image Generation

Once the scene has been described in three-dimensions, it is possible to generate a two-dimensional image from a given perspective.

The image generating programs use an *octree* to efficiently trace rays through the scene. An octree subdivides space into nested octants which contain sets of surfaces. In RADIANCE, an octree is created from a scene description by *oconv*. The details of this process are not important, but the octree will serve as input to the ray-tracing programs and directs the use of a scene description.

Rview is ray-tracing program for viewing a scene interactively. When the user specifies a new perspective, *rview* quickly displays a rough image on the terminal, then progressively increases the resolution as the user looks on. He can select a particular section of the image to improve, or move to a different view and start over. This mode of interaction is useful for debugging scenes as well as determining the best view for a final image.

Rpict produces a high-resolution picture of a scene from a particular perspective. This program features adaptive sampling, crash recovery and progress reporting, all of which are important for time-consuming images.

A number of filters are available for manipulating picture files. *Pfilt* sets the exposure and performs anti-aliasing. *Pcompos* composites (cuts and pastes) pictures. *Pvalue* converts a picture to and from alternate forms.

Currently only a few graphics output devices are supported. *Tttyimage* produces a crude character representation of an image on a dumb terminal. *Aedimage* produces output on an AED 512 graphics terminal, and *ximage* produces an image on an X-window server. Output is also available on certain dot-matrix printers and the Dicomed film recorder. The list of supported output devices is expected to grow as the system is made more widely available.

4. Acknowledgements

This work was supported by the Assistant Secretary of Conservation and Renewable Energy, Office of Building Energy Research and Development, Buildings Equipment Division of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Additional work was sponsored by the Swiss federal government under the Swiss LUMEN Project and was carried out in the Laboratoire d'Energie Solaire (LESO Group) at the Ecole Polytechnique Federale de Lausanne (EPFL University) in Lausanne, Switzerland.

5. References

Ward, G., ''Measuring and Modeling Anisotropic Reflection,'' *Computer Graphics*, Chicago, July 1992.

Ward, G., P. Heckbert, ''Irradiance Gradients,'' *Third Annual Eurographics Workshop on Rendering*, to be published by Springer-Verlag, held in Bristol, UK, May 1992.

Ward, G., ''Adaptive Shadow Testing for Ray Tracing,'' *Second Annual Eurographics Workshop on Rendering*, to be published by Springer-Verlag, held in Barcelona, SPAIN, May 1991.

Ward, G., ‘‘Visualization,’’ *Lighting Design and Application*, Vol. 20, No. 6, June 1990.

Ward, G., F. Rubinstein, R. Clear, ‘‘A Ray Tracing Solution for Diffuse Interreflection,’’ *Computer Graphics*, Vol. 22, No. 4, August 1988.

Ward, G., F. Rubinstein, ‘‘A New Technique for Computer Simulation of Illuminated Spaces,’’ *Journal of the Illuminating Engineering Society*, Vol. 17, No. 1, Winter 1988.