Integrated Lighting System
Design And Visualisation.

"The simulation and presentation of lighting effects."

RADIANCE USERS MANUAL

VOLUME TWO

SIMON CRONE

ARCHITECTURAL DISSERTATION

NOVEMBER 1992

Architectural Dissertation 599 November 1992

INTEGRATED LIGHTING SYSTEM

DESIGN AND VISUALISATION.

The simulation and presentation of lighting effects.

Volume Two

RADIANCE USERS MANUAL

Simon Michael Dalrymple Crone.

Tutor Neville D'Cruz

Co Tutor Terry McMinn

Preface

This users manual is intended to provide a concise condensed version of the documentation that is available with the RADIANCE program. It contains information gathered from the RADIANCE Users Manual (Draft), the RADIANCE reference manual, the RADIANCE UNIX manual pages, correspondence with the author, (Greg Ward at LBL) and much personal experience.

Contents

1.0 Introduction to the LBL's RADIANCE Package.

RADIANCE is a computer software package developed by the Lighting Systems Research group at Lawrence Berkeley Laboratory under the direction of Greg Ward. It is a research tool for accurately calculating and predicting the visible radiation in a space. The program uses three dimensional (3D) geometric models as input, to generate spectral radiance values in the form of photo realistic images. The package though is more than just a photo-realistic renderer.

By using accurate input into the program, such as manufacturers photometric data for specific lighting fixtures, designers are able to confidently evaluate their designs without the risk of being led astray by visually appealing yet totally inaccurate images. The RADIANCE software package is of most use when dealing with innovative, experimental lighting designs. The program can account for both specular and diffuse interreflections thus allowing both the designer and client a genuine view of a finished space.

There are three steps to producing such an image.

1) The first involves creating or converting a three dimensional description of a physical environment or scene (ie an office interior; rooms, furniture lights etc) into simple geometric elements that can be interpreted by the RADIANCE package. Such elements include polygons, spheres, cylinders and cones.

2) These must then be assigned a specific material or property, for example metal, glass, wood, marble etc. This second step also includes the setting up of specific light sources, their strength, type and distribution if necessary.

3) The final step is to render the scene to produce an image. This image may then be "cleaned", "analysed", and "filtered" in a variety of ways depending on the required application. This process is of course an iterative one. The designer can easily go back and change the geometry or material specifications until the required design has been reached.

2.0 Scene description Input requirements.

Scene description (3D geometry and material properties) is passed to the RADIANCE program in the form of any number of text files. These files specify the size, position, shape and material type. These files can be created by hand or produced by another program (a CAD package and converter etc).

2.1. General file specification.

All scene description files have the same format, that is a combination of individual *primitives* or building blocks. For example a material *primitive* may be defined (say, as a red material), then an object *primitive* may be defined (say, as a polygon) that uses the previously defined material (ie producing a red polygon *primitive*).

All scene *primitives* have the following format:

An optional comment

modifier **type** **identifier**

n A number (n) of string arguments.
Ø A number (Ø) of integer arguments (not used at present)
n A number (n) of real (decimal) arguments

The **modifier** must be either the word **void** or a name (ie. an **identifier**) of a previously defined *primitive*. The word **void** is used when the *primitive* does not need to be modified by any other *primitive*.

The **type** must be one of RADIANCE's *primitive* types. They can be either material types (eg plastic, glass, metal etc), object types (eg polygons, spheres, cones etc) or one of the special types (eg pattern, material or mixture).

The **identifier** is simply a unique name with which to label the *primitive*. This name can then be used as a **modifier** in the subsequent definition of any *primitive*.

One special kind of *primitive* **type** is **alias**. This type allows any number of **identifier**s to be defined to the one **modifier**.

The format is simply,

modifier	**alias**	**identifier**
	reference	

where the *reference* is a previously defined **identifier** (ie object or material *primitive*). The **alias** type is basically used to copy a *primitive*s definition to a new name.

The number (n) and type of string and real arguments (ie words or numbers separated by spaces following the initial **n**) depends upon the primitives **type**. The string arguments are usually file names and transformation information. The integer arguments are not currently used by the RADIANCE program and as such is always Ø. The real arguments are always numbers.

There are a few simple rules that must be followed in the description of a scene.

An object *primitive* must have at least one material *primitive*. (ie an object must be made from a material).

A **modifier** must be defined before it can be used.

Only the most recent definition of a **modifier** will be used. (ie if the same name (**identifier**) has been used to label two different *primitives* only the second definition will apply to any following *primitives*. Thus it is possible to redefine an **identifier** once it has been used).

A comment line must begin with a hash sign # and end with a return.

Any line that begins with an exclamation mark **!** will be treated as a command and executed. The executed program's output will then be taken as input into the RADIANCE program.

An example of a Basic RADIANCE Primitive description:

(Explanations in brackets are not part of the scene description).

```
#       A red material definition           <-( Comment )

( modifier ) ( type )       ( identifier )

void         plastic        red_material
Ø                                           <-( No string arguments )
Ø
5       1    Ø    Ø    Ø    Ø                 <-( Five numeric arguments
                                              specifying the colour red
                                             (1 Ø Ø for RGB ), reflectance
                                            (Ø) and roughness (Ø))


#       A copy of the red material          <-( Comment )

( modifier ) ( type )       ( identifier )

void         alias          red_material_copy
                            red_material    <-( reference )


#       A red sphere called ball            <-( Comment )

( modifier )        ( type )       ( identifier )

red_material        sphere         ball
Ø                                           <-( No string arguments )
Ø
4       Ø    Ø    Ø    1                     <-( four arguments specifying
                                              position (Ø,Ø,Ø) & radius (1))

#       A red cylinder called pipe          <-( Comment )

red_material_copy   cylinder       pipe
Ø                                           <-( No string arguments )
Ø
```

```
7    Ø    Ø    Ø                    <-( Seven arguments
     Ø    Ø    2                    specifying start point (Ø,Ø,Ø),
     1                              end point (Ø,Ø,2) & diameter
                                 (1))
```

2.2. 3D Geometry

The easiest way to create RADIANCE scene geometry is by using a 3D CAD system and importing the geometry through a conversion program. If this is not possible RADIANCE provides a number of object creation programs which can be used to create simple scenes. The description of all of these programs is beyond the scope of this manual. The other alternative, albeit a slow one, is to enter the geometry straight into a text editor.

RADIANCE uses a right handed coordinate system. That is the z vector or axis points up, the x vector or axis points east with the y vector or axis pointing north. The choice of units is totally up to the designer so long as the values are kept within a reasonable range (about $1Ø^{-5}$ and $1Ø^{8}$ in size).

RADIANCE requires the user to be aware of the direction of each objects surface normal. The surface normal specifies the front of the object ie the side that it will be viewed from. An easy way to tell the surface normal direction is to use a right hand rule. By following the sequential direction of points around an object (ie clockwise or anti-clockwise) with the index and middle finger, the thumb then is pointing in the direction of the surface normal.

A scene is made up from a combination of simple geometry types. RADIANCE uses the following object primitive types.

2.2.1. Polygon

Polygons are specified by a list of three dimensional vertices. These vertices proceed in a counter-clockwise direction when viewed from the front (ie into the surface normal). The last vertex

is connected to the first automatically. Holes may be included in a polygon by using internal vertices connected to the outer perimeter by coincident edges or seams.

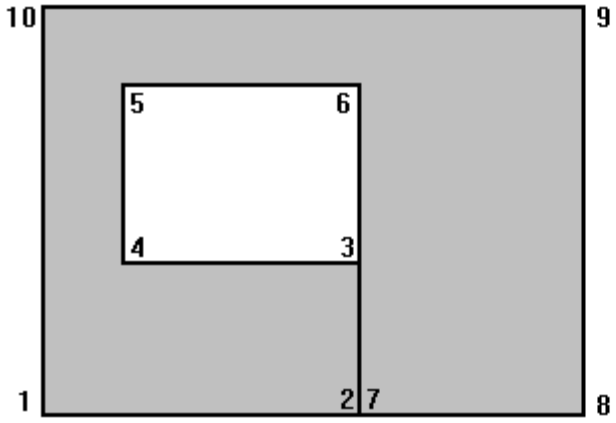

Figure 1. Example of coincident edges or seam.

The polygon primitive:

modifier	**polygon**			**identifier**
Ø				
Ø				
n	x1	y1	z1	<-(vertex one)
	x2	y2	z2	<-(vertex two)
	.....			
	xn	yn	zn	<-(vertex n)

Example:

red_material	polygon		ground_plate
Ø			
Ø			
12			
	Ø	Ø	Ø
	1Ø	Ø	Ø
	1Ø	1Ø	Ø
	Ø	1Ø	Ø

There is no limit to the number of vertices. Self intersecting polygons, (such as bow ties) should be avoided.

2.2.2. Sphere

A sphere is simply defined as a centre point and a radius. The surface normal of a sphere points away from the centre.

The sphere primitive:

```
modifier      sphere      identifier
Ø
Ø
4     x       y     x           <-( centre point )
      r                         <-( radius )
```

Example:

```
red_material  sphere      ball
Ø
Ø
4     Ø       1Ø    Ø
      1
```

2.2.3. Bubble

A bubble is a sphere whose surface normal points towards its centre. It can be thought of as a hollow sphere.

2.2.4. Cone

A cone is a cylinder with differing end diameters. One of the ends may be a point. It is specified by two endpoints of its central axis and a starting and ending radii.

The cone primitive:

```
      modifier      cone  identifier
Ø
Ø
8     xØ    yØ      zØ          <-( starting point )
      x1    y1      z1          <-( ending point )
      rØ    r1                  <-( starting radius
                                    & ending radius )
```

Example:

```
red_material  cone   megaphone
Ø
```

```
Ø
8       Ø       Ø       Ø
        5       Ø       Ø
        1       3
```

2.2.5. Cup

A cup is simply a cone whose surface normal points inward. ie a hollow cone.

2.2.6. Cylinder

A cylinder is similar to a cone, but its starting and ending radius are equal.

The cylinder primitive:

```
        modifier        cylinder        identifier
Ø
Ø
7       xØ      yØ      zØ              <-( starting point )
        x1      y1      z1              <-( ending point )
        r                               <-(  radius )
```

Example:

```
        red_material    cylinder        wand
Ø
Ø
7       Ø       Ø       Ø
        5       Ø       Ø
        1
```

2.2.7. Tube

A tube is a cylinder whose surface normal points inward.

2.2.8. Ring.

A ring is a circular disk defined by a centre point, a surface normal direction vector and an inner and outer radius. The sequence of the two radii does not matter and one of the radii may be zero.

The ring primitive:

modifier		**ring**	**identifier**	
Ø				
Ø				
8	xØ	yØ	zØ	<-(starting point)
	x1	y1	z1	<-(normal vector)
	rØ	r1		<-(inner radius & outer radius)

Example:

	red_material	ring	dinner_plate
Ø			
Ø			
8	Ø	Ø	Ø
	Ø	Ø	1
	1	Ø	

2.2.9. Source.

A source is a special type of geometric primitive type. It is not really a surface but more of a direction in the form of a disk. It is used to represent objects (usually lights) that are very distant. A source is described by a direction to its centre and by the number of degrees subtended by its disk. It can be thought of as the sky hemisphere.

The source primitive:

modifier		**source**		**identifier**	
Ø					
Ø					(direction vector to
4	x	y	z		<- object centre)
	angle				<-(angle subtended by disk)

Example:

red_material		source		sunsetsky
Ø				
Ø				
4	Ø	Ø	1	
	18Ø			

2.2.1Ø. Instance.

The instance primitive type is used for making multiple copies of previously defined primitives. It is different from the alias type as it makes copies of geometric objects, not just material definitions. An instance uses a previously converted scene description (called an octree see section 3.1) and transformation information (see section 2.4.1 *xform*). Instancing can quickly create a large complicated or repetitive scene from the one simple object (eg. Instancing one seat to create a theatre auditorium) without using the same amount of memory.

The instance primitive:

modifier	**instance**	**identifier**	
n + Ø Ø	octree	transformation	<-(octree name & any no of trans)

Example:

void 2 Ø Ø	instance tree	-s .5	small_tree _copy t 1Ø Ø Ø	<-(Scales and shifts the new copy of a tree)

2.3. Material Assignments.

In order to create a realistic image all scene geometry must be assigned a material. This material, which can be a combination of a number of other materials, determines how light will interact with the geometric surface. RADIANCE offers four classes of materials:

2.3.1. Normal materials

Normal materials can be classed as either *plastic, metal, trans* or *mirror.* They are defined as having a diffuse and specular component, a colour and a roughness factor. A purely specular material would have a roughness factor of Ø. A totally diffuse

material is treated as a Lambertian surface. The *mirror* type is a special case.

2.3.1.1. Plastic

Plastic is a material with uncoloured highlights. It is defined by a red green and blue reflectance value, a specularity value and by a roughness value. A positive roughness value will display highlights (uncoloured by the materials modifier) but not show any reflections from other objects.

The plastic primitive:

modifier		**plastic**		**identifier**
Ø				
Ø				
5	R	G	B	<-(Colour)
	spec	rough		<-(specularity & roughness)

Example:

void	plastic		gloss_white_paint
Ø			
Ø			
5	1	1	1
	.Ø3	.Ø1	

Acceptable values:

colour	[Ø:1],[Ø:1],[Ø:1]	Min Ø, Max 1
	black - white	
specularity	[Ø:1]	Min Ø, Sug. max Ø.Ø7
	matte - satin	
roughness	[Ø:1]	Min Ø, Sug. max Ø.Ø2
	polished - low gloss	

2.3.1.2. Metal

The metal material is similar to plastic except that its highlights are modified by the material colour.

The metal primitive:

modifier	**metal**			**identifier**
Ø				
Ø				
5	R	G	B	<-(Colour)
	spec	rough		<-(specularity & roughness)

Example:

void	metal		brass
Ø			
Ø			
5	.68	.27	.ØØ2
	.95	Ø	

Acceptable values:

colour	[Ø:1],[Ø:1],[Ø:1]	Min Ø, Max 1
	Ø-1ØØ% reflectance	
specularity	[Ø:1]	Sug Min Ø.5, Max 1
	dirty - clean	
roughness	[Ø:1]	Min Ø, Sug Max Ø.5
	polished - roughened	

2.3.1.3. Trans

The trans material is basically a translucent plastic. It takes the same parameters as plastic as well as transmission factor and a transmitted specularity value. The transmission factor is the fraction of penetrating light that travels through the material. The fraction of transmitted light that is not

diffusely scattered is the specular transmitted value. This material is infinitely thin and will modify the colour of the scattered light.

The trans primitive:

modifier	**trans**			**identifier**
Ø				
Ø				
5	R	G	B	<-(Colour)
	spec	rough		<-(specularity & roughness)
	trans	tspec		<-(transmission & transmitted specularity)

Example:

void	trans		lamp_shade
Ø			
Ø			
5	.7	.3	.2
	Ø	Ø.Ø5	
	.5	.5	

Acceptable values:

colour	[Ø:1],[Ø:1],[Ø:1] black - white	Min Ø, Max 1
specularity	[Ø:1] matte - satin	Min Ø, Sug. max Ø.Ø7
roughness	[Ø:1] polished - low gloss	Min Ø, Sug. max Ø.Ø2
transmission	[Ø:1] opaque - transparent	Min Ø, Max 1
transmitted specularity	[Ø:1]	Min Ø, Max 1

diffuse - clear

2.3.1.4. Mirror

The mirror material is used to produce secondary source reflections. It can only be used on planar surfaces (eg rings and polygons) and is defined by red, green and blue reflectance values. An optional string argument may be included in the primitive to specify a different material to be used for shading non-source rays.

The mirror primitive:

modifier	**mirror**			**identifier**
Ø +	modifier			<-(Optional)
Ø				
3	R	G	B	<-(Colour)

Example:

void	mirror		silver_mirror
Ø			
Ø			
3	1	1	1

Acceptable values:

colour	[Ø:1],[Ø:1],[Ø:1] black - silver	Min Ø, Max 1

2.3.2. Lights

Lights are materials that are self-luminous or emissive surfaces. They may be polygons, spheres, disks, sources or cylinders (provided they are long enough). The variations of the light type material are *spotlight*, *illum* and *glow*.

All the light types are defined by a red green and blue radiance value. Ways of accurately obtaining these values are discussed in section 3.Ø.

2.3.2.1. Light

The light primitive type is the basic material for light emitting surfaces. Cones are currently not supported as light sources. Modifiers (especially patterns) may be used to specify a lights output distribution.

The light primitive:

modifier		**light**		**identifier**	
Ø					
Ø					
3	R	G	B		<-(radiance value)

Example:

void	light		light_bulb	
	Ø			
	Ø			
	3	128	128	128

Acceptable values:

colour (Ø:inf),(Ø:inf),(Ø:inf)		Min Ø, Max infinite
	output brightness	

2.3.2.2. Spotlight

The spotlight primitive type is used for self-luminous surfaces that require a directed output. It is defined with red, green and blue radiance values as well as an orientation (output direction) vector and a full cone angle (in degrees). The orientation vector determines the distance

of effective focus behind the source centre (ie the focal length).

The spotlight primitive:

modifier	**spotlight**			**identifier**
Ø				
Ø				
3	R	G	B	<-(radiance value)
	angle			<-(cone angle)
	x	y	z	<-(direction vector)

Example:

void	spotlight			spot_light
Ø				
Ø				
3	128	128	128	

Acceptable values:

colour	(Ø:inf),(Ø:inf),(Ø:inf)	Min Ø, Max infinite
	output brightness	
angle	[Ø:36Ø]	Min Ø, Max 36Ø
	no shadows - always shadows	
direction	(-inf:inf),(-inf:inf),(-inf:inf)	Min & Max infinite
	any aimed orientation	

2.3.2.3. Illum

The illum primitive is used for secondary light sources with broad distributions. The secondary light source is treated like any other light primitive except when it is viewed directly. It then takes on the characteristics of a different

material, or becomes invisible. They are of the most use when dealing with brightly illuminated surfaces or windows.

The illum primitive:

modifier	**illum**			**identifier**
1	**modifier**			<-(new material)
Ø				
3	R	G	B	<-(radiance value)

Example:

void	illum		window
1	glass		
Ø			
3	12	12	12

Acceptable values:

colour (Ø:inf),(Ø:inf),(Ø:inf)	Min Ø,Max infinite
output brightness	

2.3.2.4. Glow

The glow primitive is used for surfaces that are self-luminous, but limited in their effect. The material is defined with red, green and blue radiance values and also a maximum radius for shadow testing.(ie any object that is outside the radius will not cast a shadow from this source).

The glow primitive:

modifier	**glow**			**identifier**
Ø				
Ø				
4	R	G	B	<-(radiance value)
maxrad				<-(maximum radius)

Example:

void		glow		aquarium
Ø				
Ø				
4	12	12	12	15ØØ

Acceptable values:

colour	(Ø:inf),(Ø:inf),(Ø:inf)	Min Ø, Max infinite
	output brightness	
maximum radius	[Ø:inf)	Min Ø, Max Infinite
	no shadows - always shadows	

2.3.3. Dielectric materials

A dielectric material is a transparent material that refracts and reflects light, such as water or crystal. The material thus has an index of refraction and a specific spectral absorbance RADIANCE has a number of dielectric primitive types such as the interface type and glass type.

2.3.3.1. Dielectric

The dielectric primitive is as described above. It is defined by the red, green and blue transmission in each wave length and by its index of refraction. An optional parameter, the Hartmann constant, (which is usually zero) describes how the index of refraction changes as a function of wavelength. A pattern will only modify the refracted value.

The dielectric primitive:

modifier	**dielectric**			**identifier**
Ø				
Ø				
5	R	G	B	<-(transmission

					value)
	n	Hc			<-(refraction index & Hartmann constant)

Example:

void		dielectric			crystal
Ø					
Ø					
5	.5	.5	.5	1.5	Ø

Acceptable values:

transmission	[Ø:1],[Ø:1],[Ø:1]	Min Ø, Max Ø
	black - transparent	
refractive index	(1:2>	Min 1, Sug max 2
	vacuum - diamond	
Hartmann's constant	<-2Ø:3Ø>	Min -2Ø, Max 3Ø
	negative dispersion - positive dispersion	

2.3.3.2. Interface

The interface primitive type is a boundary between two dielectrics (ie water and crystal). Ordinary dielectics are surrounded by a vacuum. The interface is defined by two sets of transmission and refractive indexes, the first being the inside, the second the outside.

The interface primitive:

modifier		**interface**		**identifier**
Ø				
Ø				
8	R1	G2	B3	<-(transmission

					value 1)
	n1				<-(refraction index 1)
	R2	G2	B2		<-(transmission value 2)
	n2				<-(refraction index 2)

Example:

void		dielectric			surface
Ø					
Ø					
8	.5	.5	.5	1.5	
	.7	.7	.7	1.9	

Acceptable values:

interior transmission	[Ø:1],[Ø:1],[Ø:1]	Min Ø, Max 1
black - transparent		
interior refractive index	(1:2>	Min 1, Max 2
vacuum - diamond		
exterior transmission	[Ø:1],[Ø:1],[Ø:1]	Min Ø, Max 1
black - transparent		
exterior refractive index	(1:2>	Min 1, Max 2
vacuum - diamond		

2.3.3.3. Glass

The glass type primitive is a specially modified dielectric. The material has been optimised to only produce one reflected ray and one transmitted ray through a single thin surface. In this way internal reflections are avoided. The glass type has a standard refractive index of 1.52 and all that is needed to be defined is the transmission at normal incidence.

The glass primitive:

modifier		**glass**		**identifier**
Ø				
Ø				
3	R1	G1	B1	<-(transmission value 1)

Example:

void		glass		glass_window
Ø				
Ø				
3	.96	.96	.96	

Acceptable values:

transmission	[Ø:1],[Ø:1],[Ø:1]	Min Ø, Max 1
	black - transparent	

2.3.4. BRDF materials

BRDF materials are primitive types with bidirectional reflectance distribution functions (thus BRDF's). They are specific plastic like materials that get accurate specular distributions from either procedurally defined functions or from data files. As such they are beyond the scope of this manual. Information on the BRDF materials may be found in the RADIANCE Reference manual.

2.4. Pattern modifiers.

A pattern is defined as a perturbation (shift) in a materials colour. It effects the reflectance or transmittance properties of an object. There are two ways of specifying a pattern. They are either through a procedural function (ie a mathematical calculation based upon RADIANCE information or random functions) or through a coordinate mapping of data

from files (ie the shift in colour is dependent upon the data in one or a number of files).

2.4.1. Material Transformations

Patterns as well as textures often need transformations to scale, move and rotate defined materials onto an objects surface. The transformations available are the ones provided by XFORM. The ones most commonly used to transform materials are:

-t x y z translate the material along the vector x y z.

-rx (ry, rz) degrees Rotate the material degrees about an axis.

-s factor Scale the material by a factor.

2.4.2. Procedural Patterns

A procedural pattern as previously mentioned is a pattern that is dependent upon a particular calculation. This calculation usually takes values from the RADIANCE package, such as surface normals or ray intersection points etc., and combines them into a value that is then used to change the material's colour. The two types of procedural pattern types are *colofunc* and *brightfunc.*

2.4.2.1. Colorfunc

The *colorfunc* type primitive will change the colour (the red green and blue value) of a material. It is defined in terms of a function file (where the calculation occurs) and numerous arguments required by that function. The overall changed colour values may also be scaled and transformed.

The colorfunc primitive:

modifier	**colorfunc**	**identifier**
4 + red green blue		funcfile
transformations		

```
Ø
n       A1      A2      A3      ...     An
```

Example:

```
void    colorfunc       hundreds_and_ thousands
6       red     green   blue    speckle.cal     -s 1Ø
Ø
1       .Ø1
```

speckle.cal

-Start of function file ----------

{ Hundreds and thousands colour function

A1 = degree of spottyness. }

```
red = noise3a( Px/A1, Py/A1, Pz/A1);

green = noise3b( Px/A1, Py/A1, Pz/A1);

blue = noise3c( Px/A1, Py/A1, Pz/A1);
```

- EOF ----------

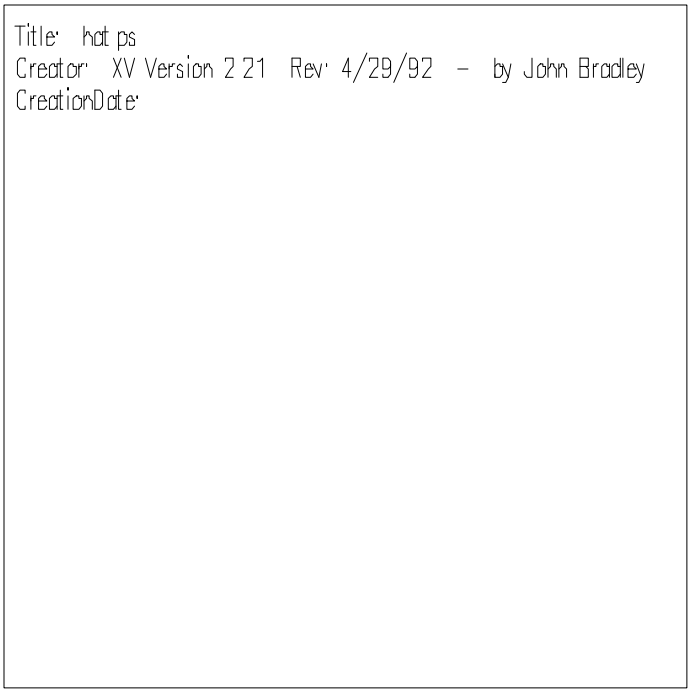

Figure 2. Hundreds and Thousands

Explanation:

A hundreds and thousands colour pattern is being described through a procedural pattern. The string arguments red, green and blue are returned from the calculation *speckle.cal* and used to change the materials colour. The whole pattern is then scaled by a factor of ten. The real argument, A1 is also used in the calculation specifying a smoothness (in this case, Ø.Ø1).

The Px, Py and Pz values found in the function file are variable that are predefined as the intersection point of a surface and a ray. The noise3a in the function file is a predefined standard noise function. The file *rayinit.cal* contains all the standard functions and defined variables. This procedural pattern primitive will basically randomly associate a colour to any surface that is defined with this material.

2.4.2.2. Brightfunc

The brightfunc primitive type is the same as colorfunc except that it only changes the brightness of the material colour not the colour itself.

The brightfunc primitive:

modifier	**brightfunc**	**identifier**		
2 + Ø	reflectance	funcfile		transformations
n	A1	A2	A3	...	An

Example:

void	brightfunc	stripes	
4 Ø	refl	stripes.cal	-s 1Ø
3	1	Ø.5	Ø.2

stripes.cal

-Start of function file ----------

{ Stripes function

A1 = Brightness of stripe (Ø to 1)

A2 = Brightness of material (Ø to 1)

A3 = width of strip as fraction of unit length }

refl = if(A3 - frac(Px), A1, A2);

- EOF ----------

Figure 3. Stripes.

Explanation:

This function file simply returns a reflectance value brightness. If the x ray intersection point is inside a strip, the function returns A1 (1), else it returns a A2 (.5). The function works on a unit scale so that as the width specified in A3 is Ø.2, then the strip width would be one fifth of the unit length. The pattern is then transformed by scaling (-s 1Ø) so that in the end the pattern will have a strip 2 units wide.

2.4.3. Data mapping patterns.

The primitive types *brightdata* and *colordata* are similar to procedural functions in that they modify a materials reflectance or transmittance. However instead of being defined procedurally their patterns are defined in a data file. *Colorpict* is a special type of *colordata* that takes a RADIANCE picture file as the input rather than three separate data files.

2.4.3.1. Colordata

The primitive colordata uses three separate data files, one for each colour, to modify a materials colour. This interpolated data map is m-dimensional. The way that the data is looked up and optionally filtered must be defined in another separate file. This function file has the original red green and blue colour values passed to it as parameters.

The colordata primitive:

modifier	**colordata**		**identifier**	
7+m+	rfunc	gfunc	bfunc	
	rdatafile		gdatafile	bdatafile
	funcfile		x1 x2 ... xm	transformations
Ø				
n	A1	A2	A3 ...	An

See section 2.5.2 (texdata) for an example of using data files.

2.4.3.2. Brightdata

The primitive type brightdata is similar to colordata except that it is monochromatic. ie It only changes the brightness of the material. As such only one data file is required.

The brightdata primitive.

modifier	**brightdata**		**identifier**	
3 + m +	rfunc		datafile	
	funcfile		x1 x2 ... xm	transformations
Ø				
n	A1	A2	A3 ...	An

2.4.3.3. Colorpict

The primitive *colorpict* as already mentioned uses a two-dimensional image stored in RADIANCE picture format to

produce a coloured pattern (or picture). The dimensions of the image are normalised so that the smaller dimension is always one unit in length with the other dimension being the ratio between the larger and the smaller. ie An image of 5ØØ x 388 would have the box coordinate size of (Ø, Ø) to (1.48, 1). The *colorpict* type normally uses a predefined function file called picture.cal. This function file always maps the new picture pattern in the xy plane with its origin at (Ø, Ø, Ø). Thus more often than not the picture must be rotated, transformed and scaled. This file provides a number of options for tiling and mapping the picture onto flat surfaces depending upon the arguments specified on the string argument line.

The options are:

pic_u & pic_v	Straight forward picture mapping.
tile_u & tile_v	Tiling of the picture
match_u & match_v picture	Tiles, mirrors and matches edges

RADIANCE also supplies a number of functions for mapping a picture onto spheres, cylinders and other non flat surfaces.

The colorpict primitive:

modifier	**colorpict**			**identifier**
7 +	rfunc	gfunc	bfunc	pictfile
	funcfile		x1 x2 ... xm	transformations
Ø				
n	A1	A2	A3 ...	An

Example.

void	colorpict	carpet_tiles	
11	red green blue		carpettile.pic

```
        picture.cal     tile_u tile_v   -rz 9Ø -s 1Ø
Ø
1       1.48
```

Figure 4. Carpet tiles ?

Explanation.

This primitive produces a tiled carpet pattern from a RADIANCE picture called carpettile.pic. Since the tile_u and tile _v string arguments were supplied the picture will repeatedly tile the picture image. The real argument (1.48) specifies the height to width ratio of the tiles. The picture pattern is then rotated 9Ø degrees around the z axis and scaled by a factor of ten.

2.4.4. Text Patterns

Text patterns are primitive types that produce text. The text font is defined by an auxiliary font file and the text may be defined as part of the primitive or in an external file. There are two types of text patterns.

2.4.4.1. Colortext

The colortext primitive is dichromayic writing in a polygonal font. The size, orientation, aspect ratio and slant of the characters are defined by right and down motion vectors. The foreground colour , the background colour and upper left origin for the text block must also be given to define the material.

The colortext primitive:

modifier	**colortext**			**identifier**
2	fontfile		textfile	
Ø				
15	Ox	Oy	Oz	<-(Origin for text)
	Rx	Ry	Rz	<-(Direction of text)
	Dx	Dy	Dz	<-(Slope of text)
	rfore	gfore	bfore	<-(Foreground colour)
	rback	gback	bback	<-(Background colour)

Example.

void	colortext		page
2	helvet.fnt		text.txt
Ø			
15	Ø	1Ø	Ø
	1	Ø	Ø
	-.2	-1	Ø
	Ø.2	Ø.3	Ø.8
	1	1	1

Figure 5. Page.

Explanation.

The text is read in from the text.txt file and displayed in the helvet.fnt font. The upper left corner of the text is defined at (Ø 1Ø Ø). The text is orientated horizontally across the page (1 Ø Ø) and has an aspect ratio of 1 as the R vector is one unit across (1 Ø Ø) and D vector is one unit down (-.2 -1 Ø). The characters are of one unit in size and slant slightly forward as the bottom of the characters are .2 of a unit behind the top (1 Ø Ø) vs (-Ø.2 -1 Ø). The font colour is blue on a white background

2.4.4.2. Brighttext

The brighttext primitive is similar to the colortext primitive except it is monochromatic.

The brighttext primitive:

modifier	**brighttext**	**identifier**
2	fontfile	textfile

```
Ø
11      Ox      Oy      Oz      <-( Origin for text )
        Rx      Ry      Rz      <-( Direction of text )
        Dx      Dy      Dz      <-( Slope of text )
        foreground      background      <-( Brightness )
```

Example.

```
void    brighttext      page2
2       helvet.fnt      text2.txt
Ø
11      Ø       Ø       Ø               <-( Origin )
        Ø       1Ø      Ø               <-( Rotation )
        5       -.2     Ø               <-( Slant )
        .2      1                       <-( Brightness )
```

Explanation.

The text is read in from the text2.txt file and displayed in the helvet.fnt font. The upper left corner of the text is defined at (Ø Ø Ø). The text is orientated vertically up the page (Ø 1Ø Ø) and has an aspect ratio of .5 as the R vector is ten units (Ø 1Ø Ø) and D vector is one five units (5 -.2 Ø). The characters are of ten units in size and slant slightly forward as the bottom of the characters are .2 of a unit behind the top (Ø 1Ø Ø) vs (5 -Ø.2 Ø). The font has a brightness of .2 and is on a white background.

2.5. Texture Modifiers.

Where pattern modifiers alter (perturb) a material's colour, texture modifiers perturb a material's surface normal. This perturbation may be defined as a function or specified by data. A texture unlike a pattern takes into account the direction of light that is illuminating the surface.

2.5.1. Texfunc

The texfunc primitive uses a function file to specify a procedural texture. The function file, like all other function files, uses

predefined RADIANCE variables in calculations to shift a surface normal.

The texfunc primitive:

modifier	**texfunc**			**identifier**
4 +	xpert	ypert	zpert	
	funcfile		trans	
Ø				
n	A1	A2	...	An

Example.

void	texfunc		groove	
4	xpert	ypert	zpert	groove.cal
Ø				
1	.1			

Groove.cal

```
{-Groove function file ----------------------------------------}
{       Groove function for a surface in the xy plane
        origin at ( Ø Ø Ø )

A1 = Width of horizontal ( x axis ) groove as fraction of
        unit length.
}

xpert = Ø;                      { don't change the x normal }
zpert = Ø;                      { don't change the y normal }
ypert = if( A1/2 - frac( Py ),  { if ray in bottom grove }
                        Ø.5,            { return +ve pert }
                        if( A1 - frac( Py ),    { if ray in top groove
                        }
                                -Ø.5,   { return -ve pert }
                                Ø       { else no pert }
                        )
        );
```

{- EOF ----------}

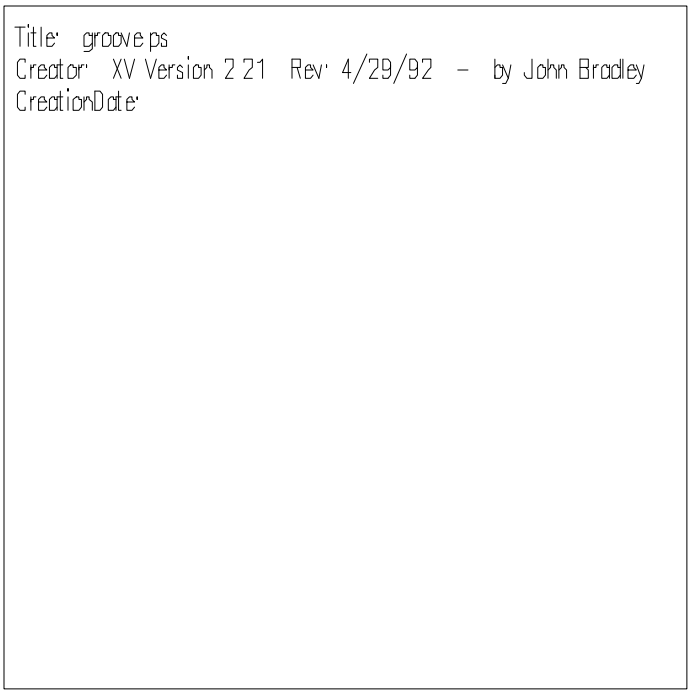

Figure 6. Groovy Texture.

Explanation.

The values returned from the function file groove.cal are added to the surface normal for each specific point of the surface. The calculation checks if the y coordinate of the intersection point lies within a grove. If it is then the y value of the normal vector is changed. The function uses an if statement (if(a, b, c) - if a is +ve, return b, else return c), which is defined in the *rayinit.cal* file.

2.5.2. Texdata

The texdata primitive type uses three data files to get the surface normal perturbations rather than a function file.

The texdata primitive:

modifier	**texdata**		**identifier**	
8 +	xfunc	yfunc	zfunc	<-(function args)

```
        xdataf ydataf zdata          <-( data file names )
        fname xØ x1 ... xf           <-( function file name )
Ø
n       A1     A2     ...    An     <-( function file args )
```

The data file format.

```
N
beg1  end1  m1
beg2  end2  m2
....
begN  endN  mN

DATA
```

Where N - The number of dimensions in array
begx - The beginning coordinate value
endx - The ending coordinate value
mx - The size of the dimension array

All data is separated by white space and no comments are allowed.

Example.

```
void   texdata      tile
9      pass_dx      pass_dy      pass_dz
       xpert.dat    ypert.dat    zpert.dat
       tex.cal      frac( Px)    frac( Py)
Ø
1      .5
```

Tex.cal

```
{-Basic texdata function file ----------------------------------}
pass_dx( dx, dy, dz) = dx * A1;          { get x normal }
pass_dy( dx, dy, dz) = dy * A1;          { get y normal }
pass_dz( dx, dy, dz) = dz * A1;          { get z normal }
```

```
        no_pert( dx, dy, dz) = Ø;              { don't change normal }
        {- EOF --------------------------------------------------------------}

Xpert.dat      ( A simple text file )
        2                       <-( Number of dimensions of array )
        Ø       1       4       <-( first dimension array size of 4 )
        Ø       1       4       <- ( second dimension array size of 4 )
        -1      -1      1       -1      <-( Data for array [1][] )
        Ø       Ø       Ø       Ø       <- (Data for array [2][] )
        Ø       Ø       Ø       Ø       <- ( .... )
        1       1       1       1       <- ( Data for array [4][] )
        EOF

Ypert.dat      ( A simple text file )
        2
        Ø       1       4
        Ø       1       4

        -1      Ø       Ø       1
        -1      Ø       Ø       1
        -1      Ø       Ø       1
        -1      Ø       Ø       1
        EOF

Zpert.dat      ( A simple text file )
        2
        Ø       1       4
        Ø       1       4

        Ø       Ø       Ø       Ø
        Ø       Ø       Ø       Ø
        Ø       Ø       Ø       Ø
        Ø       Ø       Ø       Ø

        EOF
```

Figure 7. A Quilt texture

Explanation.

This texture primitive produces a bumpy quilt texture. The surface normal data is read in from the three data files, one normal from each file. The function file is used to manipulate and then return the surface normal. The real argument in this example (A1) determines the smoothness (or height) of the texture surface.

2.6. Miscellaneous primitive types.

RADIANCE also provides other primitive types that do not fall under any of the above groups. They include the material types *antimatter, prism1 and prism2* and the types *mixfunc mixdata* and *mixtext* for blending one or more textures and patterns.

2.6.1. Antimatter

The primitive type *anitmatter* is a material that is able to "remove" volumes from other volumes. A ray that passes into an *anitmatter*

object becomes blind to all the specified modifiers. The first string argument (a modifier) will be used to shade the area between the regular volume and the *anitmatter* volume. If this modifier is *void*, then the *anitmatter* will appear completely invisible.

The antimatter primitive.

modifier		**antimatter**		**identifier**
N	mod1	mod2	...	modN
Ø				
Ø				

2.6.2. Prism1 and Prism2

The primitive type prisms are materials for general light redirection from prismatic glazing, generating secondary light sources. They can only be used to modify planar surfaces and should not result in light focusing or scattering. The string arguments specify the coefficient for the redirected light and its direction. The *prism2* type is identical to *prism1* except that it provides for two ray redirections rather than one.

The prism1 primitive.

modifier		**prism1**				
5 +	ceof	dx	dy	dz	funcfile	trans
Ø						
n	A1	A2	...	An		

The prism2 primitive.

	modifier		**prism2**		**identifier**	
5 +	ceof1	dx1	dy1	dz1		
	coef2	dx2	dy2	dz2	funcfile	trans
Ø						
n	A1	A2	...	An		

2.6.3. Mixfunc

The mixfunc primitive mixes two modifiers procedurally. The foreground and background arguments must be modifiers that have been previously uniquely defined. The vname argument is the coefficient defined in the funcfile that defines the influence of the foreground (The background coefficient is 1-vname).

The mixfunc type.

modifier	**mixfunc**		**identifier**	
4 +	foreground	background		
	vname	funcfile	trans	
Ø				
n	A1	A2	...	An

2.6.4. Mixdata

The *mixdata* primitive is similar to *mixfunc* except that it combines two or more modifiers using a data file instead of a calculation.

The mixdata primitive.

modifier	**mixdata**			**identifier**	
5 + n +	foreground			background	
	func	datafile		funcfile	
	x1	x2	...	xn	trans
Ø					
m	A1	A2	...	Am	

2.6.5. Mixtext

The mixtext uses one modifier for the text foreground and one modifier for the background.

The mixtext primitive.

modifier	**mixtext**	**identifier**
4	foreground	background
	fontfile	textfile
Ø		

9	Ox	OY	Oz
	Rx	Ry	Rz
	Dx	Dy	Dz

3. RADIANCE Light Specifications

The correct definition of light sources is critical to produce accurate images in RADIANCE. RADIANCE is able to use manufactures data in terms of specific distributions and lamp colour in its calculations.

RADIANCE, like most colour software only deals with a single band of red, green and blue. Although the human eye is able to construct almost any colour from combinations of these three colours, it is not the same as continuously sampling the entire spectrum. It is thus possible for inaccuracies to enter the rendered image. This RGB model is however, the easiest to emulate on current computer hardware and thus the most widely accepted and utilised.

When the human eye views a scene that is lit by non heavily weighted colour lights it automatically colour balances the scene as to appear white and natural. Thus even if a number of particular frequencies are absent, such as some of the higher (blue and violet) frequencies in an incandescent light source, the overall scene still appears white. If no colour balancing occurs the scene no longer looks natural. In RADIANCE there are two ways to colour balance an image. The first is to use all white light sources and the second is to use the *pfilt* program to filter the rendered image.

The RADIANCE package also provides a file called *lamp.tab* that contains useful lamp information. This file is used by the *lampcolor* program and by the *pfilt* program to obtain the RGB value of different lamps. The file defines lamp types, chromaticity coordinates and depreciation lists.

3.1. Calculating radiance values

All RADIANCE light sources require three radiance values. One for red, green and blue. To calculate the radiance value, the total initial lumen value of the light and the total surface area of the light must be known. This will produce one radiance value that is used for the red, green and blue radiance values resulting in a white balanced light fixture.

The calculation is a four part one:

1. Convert the total lumen value of the lamp into watts by dividing the lumens by 179. (watts)

2. Divide this value by Pi. (watts/steradian)

3 Divide the watts by the total emitting surface area of the lamp in square meters. (watts/steradian/m^2)

4. Compensate for fixtures and lumen depreciation by a factor of 5 to 2Ø %. (watts/steradians/m^2)

Example.

A 75 watt GLS (general lighting service incandescent lamp) has a radius of 3Ø mm a total initial lumen value of 96Ø.

Thus the radiance value for this white light source is..

power = 96Ø lumen / 179

= 56.471 watts

area = (.Ø3Ø)2 * Pi *4

= Ø.Ø1131 m^2

radiance value = power / area / Pi

= 56.471 / Pi / Ø.Ø1131

= 15Ø.944 watts/steradian/m^2

depreciation factor = 5%

Final radiance value = .95 * 15Ø.994

= 143.397 watts/steradian/m^2

The RADIANCE package provides a program called *lampcolor* that does this calculation for you. It asks for the total lumens, the geometry type of

the source, the type of lamp (from the *lamp.tab* file) etc. and produces three radiance values (red, green and blue).

3.2. Using IES distribution data

As mentioned in the introduction, RADIANCE has the ability to use manufactures' photometric data to provide accurate distributions of light for their sources. The RADIANCE package comes with the standard IES (Illuminating Engineering Society) data files for around 5Ø different lighting fixtures. These data files can be converted straight into RADIANCE scene descriptions by a program called *ies2rad*.

3.2.1 The ies2rad program

The *ies2rad* program produces a RADIANCE file, which contains the light source geometry and a data file, which contains the light distribution data. The light source geometry is always centred at the origin, aimed in the negative z direction and orientated so that the Ø degree plane is along the x axis, and as such must be transformed to its final position.

Ies2rad usage:

ies2rad [options] inputfiles...

common options:

-l libdir	Default path to look for ies data files.
-p predir	Output subdirctory name.
-o outname	Output file name (no extension)
-duntis	Output dimension units.
-i rad	Specify illum sphere for geometry (radius)
-t lamp	Specify lamp type (use default for white balanced lamp)
-m factor	Multiplication factor

Example.

```
ies2rad -dm/1ØØØ -t default -o spot iesØ6
```

This converts the IES luminaire type iesØ6 (R-4Ø flood with specular anodized reflector skirt: 45 degree cutoff) into a RADIANCE geometry file *spot.rad* and a distribution data file *spot.dat* with units in millimetres and using a colour balanced light.

The geometry file *spot.rad*:

```
# ies2rad -dm/1ØØØ -t default
# Dimensions in millimetres
#<IES #6, R-4Ø FLOOD WITH SPECULAR REFLECTOR
SKIRT; 45 DEG CUTOFF
#<LAMP=R-4Ø FLOOD
# Ø watt luminaire, lamp*ballast factor = 1

void brightdata iesØ6_dist
4 flatcorr spot.dat source.cal src_theta
Ø
1 1

iesØ6_dist light iesØ6_light
Ø
Ø
3 19.1358 19.1358 19.1358

iesØ6_light polygon iesØ6.d
Ø
Ø
12
        -114.3 -114.3 -Ø.25
        -114.3 114.3  -Ø.25
        114.3  114.3  -Ø.25
        114.3  -114.3 -Ø.25
```

```
iesØ6_light polygon iesØ6.u
Ø
Ø
12
        -114.3 -114.3 Ø.25
        114.3  -114.3 Ø.25
        114.3  114.3  Ø.25
        -114.3 114.3  Ø.25
```

The distribution file *spot.dat*:

```
1
Ø Ø 21
        Ø      5      15     25
        35     45     55     65
        75     85     9Ø     95
        1Ø5    115    125    135
        145    155    165    175
        18Ø

        7.3743         7.3743         5.64246        3.26257
        1.31844        Ø.1229Ø5       Ø      Ø
        Ø      Ø      Ø      Ø
        Ø      Ø      Ø      Ø
        Ø      Ø      Ø      Ø
        Ø
```

3.2.1. Customisation of IES files

Because of the way that the IES specifies its basic light source geometries, the physical representation of the specified lamp, often is unsatisfactory. For instance the iesØ1 lamp type will produce six polygons in the shape of a box to represent a spherical globe.

One way to overcome this is to specify a new geometry for the light source and use the distribution data file produced by ies2rad. This

is a relatively straight forward process provided specific lamp colours are not required. (A second technique is described later if non white balanced lamp colours are desired.) The first step is to define the new lamp geometry and replace the geometry created by the ies2rad program. The next step is to calculate the total surface area of this new light source. The final step is edit the radiance values of the light primitive. The new values (red, green and blue) are simply the reciprocal of the total surface area (in meters) of the light source.

Example:

This example uses the same light distribution as above but instead of a 228mm x 288mm light source it utilises a 14Ø mm diameter disk (as a recessed down spot might look).

The modified geometry file *myspot.rad*:

```
# ies2rad -dm/1ØØØ -t default
        # Dimensions in millimetres
        #<IES #6, R-4Ø FLOOD WITH SPECULAR REFLECTOR
        SKIRT; 45 DEG CUTOFF
        #<LAMP=R-4Ø FLOOD
        # Ø watt luminaire, lamp*ballast factor = 1

        void brightdata iesØ6_dist
        4 flatcorr spot.dat source.cal src_theta
        Ø
        1 1

        iesØ6_dist light iesØ6_light
        Ø
        Ø
        3       64.96  64.96  64.96  <-( 1/area of 14Ø dia disk )

        iesØ6_light ring iesØ6.myspot
        Ø
```

```
Ø
8
        Ø       Ø       Ø
        Ø       Ø       -1
        7Ø      Ø
```

If a spherical light geometry is required then the brightfunc primitive must be slightly altered. In the above examples the brightfunc modifier is dealing with flat sources and thus uses the *flatcorr* string argument in its calculations. This must be changed to *corr* when using a sphere as the light geometry.

When specific lamp geometry is required utilising different lamp colours then the following technique can be used: Firstly the three individual radiance values are obtained by using the lampcolor program as described previously. Each of these values is then multiplied by PI and divided by the maximun value found in the second half of the IES distribution data file (in the *spot.dat* example that value would be 7.3734). The result is then used as the final radiance values for the source.

3.3. Daylighting

The RADIANCE package supplies a program called *gensky* that creates a scene description for the CIE standard sky distribution. This description can be for any time of the year, any where in the world using either a sunny sky, with or without sun, or a cloudy sky. The material and surface used for the sky are left up to the user.

The output sky distribution is given as a brightness function called *skyfunc*. The x axis points east, the y axis points north and the z axis corresponds to the zenith.

```
Usage:      gensky      month day hour      [ options ]
            gensky      -ang altitude azimuth [ options ]
            gensky      -defaults
```

The gensky options are:

-s	Standard CIE clear sky
+s	Clear sky with sun.
-c	Standard CIE overcast sky
+c	Uniform cloudy sky
-g frl	Average ground reflectance
-b brt	Zenith brightness
-t trb	Turbidity factor
-a lat	Latitude in degrees north (-ve for south)
-o lon	Longitude in degrees west
-m mer	Standard meridian in degrees west of Greenwich

Example.

```
#       Hemispherical Blue Sky
#       Sunny with sun for Perth W.A
#       on 16th March, 1Ø:ØØ am

!gensky 3 16 1Ø +s -a -32 -o 115.6 -m 12Ø

skyfunc         glow    skyglow
Ø
Ø
4       .9      .9      1       Ø

skyglow         source          sky
Ø
Ø
4       Ø       Ø       1       18Ø
```

4. Image Rendering

Once a scene has been fully defined in terms of its geometry and materials it can be rendered into a two dimensional image. All that needs to be chosen is the particular viewing point. RADIANCE uses the simulation technique of image-oriented raytracing. This involves tracing a ray of light backwards from the viewers eye position, to one or more sources, taking into effect specular reflections, transmissions and all geometries. The reason for doing this in reverse as opposed to the real world model, is that of all of the rays that are reflected and refracted from a light source, only a very small number actually enter the eye.

4.1. Oconv

To reduce the time taken to generate images, RADIANCE uses octrees to sort the geometry in a scene. An octree recursively subdivides spaces into nested octants or cubes which contain no more than a set number of objects. When a ray is traced, intersection calculations are only performed on those objects which lie in the cubes of the intercepting ray, not across the whole scene and thus reducing the time required to render a scene.

The *oconv* program is used to create an octree file from scene description files. This octree file is then used as input for the rendering programs. An octree may be frozen by using the -f option in the oconv program to have its information stored in a binary format at the end of the octree file. This enables the octree to be faster loading, machine independent and not depend upon the original scene description files.

4.2. Rview

Rview is a ray-tracing rendering program for interactively viewing a scene in perspective. It is used not as a final image renderer but as a device for debugging scenes, for evaluating lighting and for setting viewing parameters. It displays a rough image of the scene on the screen and slowly increases its resolution. The users can interrupt this refining and enter a command into a dialogue box to change a number of settings such as the viewing type, size and magnification, the exposure or refinement

frame. Most of rview's command line options can be specified interactively with the dialogue box *set* command, while running the program.

Rview command line usage:

rview [options] octree <-(Renders interactive view)

rview -defaults <-(Lists default values)

Common rview options. (* denotes interactive (dialogue box) command only)

Viewing Parameters.

*last file Load viewing parameters from file.

*aim [*mag* [x y z]] Zoom in by mag on specified point. If no point is specified then the cursor is used to select the view centre.

*view [file] Saves current viewing parameters to file. If file is left out then rview prompts for the following view settings.

-vtt View type. t can be either 'v' for perspective view, 'l' for parallel view, 'a' or 'h' for fish eye views.

-vp x y z Viewing point. (or centre point for parallel view)

-vd xd yd zd View direction vector.

-vu xd yd zd View up direction

-vh val Horizontal field of view in degrees.

-vv val Vertical field of view in degrees.

-vf file Get viewing parameters from a file.

Common program variables.

*set [var [val]] Changes or checks specified program variables. If val is absent the current value of var is displayed. If var is absent a list of all available variables are displayed. The following variables can be specified on the command line.

-ab N Sets the number of ambient bounces to N which determines the number of diffuse bounces calculated through an indirect calculation.

-av red green blue Set the ambient light value to a radiance of red, green and blue to be used in place of an indirect light calculation. Examples of quick ways of calculating this are shown below.

Miscellaneous parameters.

*frame [xmin ymin xmax ymax] or [all]
Sets the frame for refinement. If the bounding box coordinates are not given, the cursor is used to pick the box boundaries. The *frame all* command will reset the box to the total image size.

*exposure [spec] Adjusts the exposure. The spec value can begin with either a '+' or '-' (specifying the a number of fstops), or an '=' (specifying an absolute value). If the '=' option is given without the spec value or if the spec value is omitted altogether then the cursor is used to pick a point for normalisation.

*new Restarts the rendering of the image.

*quit Quit the program.

*^R Redraw the screen.

*write.[file] Write the current image to the file (at current resolution).

Calculating the ambient light value.

Ambient light levels are specified when indirect calculations are not required (ie when -ab is Ø see rpict). There are two ways for quickly acquiring a rough estimate of ambient light values. The first is to select a point using the interactive trace command that is half way between full shadow and light shadow. The trace command returns the object selected, its location, material and most importantly, its luminance value. This value can then be used as the ambient value.

The second technique involves setting the -ab command line option to 1 at runtime. After a rough image has appeared set the ab value to Ø and set the av values until the new sections of the image match the colour of the original image. Use these values for the final rendering using *rpict*.

4.3. Rpict

Rpict is the program that produces a high resolution picture of a scene from a given perspective. The image may take a few minutes or many hours to generate depending on the resolution of the final picture and the desired picture accuracy. The rpict program output is controlled by the specification of a number of command line variables. These fall into several categories such as views, resolution, direct and indirect calculations.

Rpict usage;

rpict [options] octree > imagefile <-(Create image file)

rpict [-defaults] <-(List variable defaults)

Common rpict options.

Viewing parameters.

Rpict takes the same viewing variables as *rview* (see above). Usually these viewing parameters are fine tuned in rview and saved to a file. The rpict program then simply reads this file using the -vf option.

Resolution parameters.

The horizontal and vertical resolution values determine the amount of detail in the final image and as such heavily influence the pictures rendering time. For large images it is possible to reduce this time by using image plane sampling (-ps).

-x xres Specify x resolution.

-y yres Specify y resolution.

-ps size Sets the sample pixel spacing for adaptive subdivision on the image plane.

-pj frac Sets the sample jitter to frac. This value, between Ø and 1 is used when anti-aliasing by randomly sampling over pixels.

Direct calculation parameters.

The direct calculation does not use any interreflected component and as such should be provided with an ambient light level (as described with rview). By the programs defaults, sources are treated as if they emanate from a point. By jittering a ray to a source by an amount proportional to the sources size, a more accurate image results. This, however, requires that the image plane sampling be set to zero thus resulting in longer rendering times.

-av red green blue Sets the ambient values (as discussed above)

-dj frac Sets direct jittering to frac. (between Ø and 1)

Indirect calculation parameters.

The indirect calculation uses an interreflected component for ambient light. Each calculation produces a number of rays which

are stored and used for interpolation on nearby values. These indirect illuminance values may be stored in a file which can be shared for slightly faster renderings.

-ab N Sets the number of ambient bounces to N which determines the number of diffuse bounces.

-af file Sets the ambient file to file.

Miscellaneous parameters.

-lr N Limit number of reflections to N.

-t sec Sets the time between progress reports.

-e file Sends progress reports (from -t) and error messages to file instead of standard error.

The following table shows typical rpict values in relation to rendering speed and accuracy.

Param	Description	Min	Fast	Accur	Max	Default
-ps	pixel sampling	16	8	4	1	4
-pj	anti-aliasing jitter	Ø	Ø.6	Ø.9	1	Ø.67
-dj	source jitter	Ø	Ø	Ø.7	1	Ø

4.4. Pfilt

The pfilt program performs anit-aliasing and scaling on an image file produced by rpict. Other options include setting the exposure of the image, colour balancing and creating star highlights around bright areas of a picture.

Pfilt usage;

pfilt [options] [file]

Common pfilt options.

-x res Set the x resolution to xres. If a slash followed by a real number is given for xres, then the new x resolution will be set to the original resolution divided by the real number.

-y yres Set the y resolution to yres (same as -x option).

-e exp Adjusts the exposure. The exp value can begin with either a '+' or '-' (specifying the a number of fstops), otherwise it is interpreted as a straight multiplier.

-t lamp Colour balance the image as if the lamp fixture type was used.

-r rad Use Gaussian filtering with a radius of rad to provide highest quality images.

-n N Set number of star points for star pattern to N.

-h lvl Set the intensity for which areas will start to draw star patterns.

Reducing the image resolution by two or three and using a Gaussian filter radius of around .6 produces the highest quality anti-aliased image.

5. References

Greg Ward. *"The RADIANCE 2.Ø Synthetic Imaging System Reference Manual."*

Lawrence Berkeley Laboratory.

Berkeley, California.

Cindy Larson *"RADIANCE - Users Manual (Draft)"*

Lawrence Berkeley Laboratory.

Berkeley, California. Nov 1991.

Greg Ward RADIANCE manual pages.

(Principle Author) Lawrence Berkeley Laboratory.

Berkeley, California.

John E. Kaufman "*IES Lighting handbook, Reference volume.*"

Illuminating Engineering Society

New York. 1981

Appendix A

Location of files.

All execulables may be found in
/usr/local/bin

All library files, eg function files, picture files etc may be found
/usr/local/lib/ray

RADIANCE PROGRAM LIST.

aedimage	- RADIANCE driver for AED 512 color graphics terminal
arch2rad	- convert Architrion text file to RADIANCE description
calc	- calculator
cnt	- index counter
dayfact	- compute illuminance and daylight factor on workplane
ev	- evaluate expressions
falsecolor	- make a false color RADIANCE picture
findglare	- locate glare sources in a RADIANCE scene
genbox	- generate a RADIANCE description of a box
genprism	- generate a RADIANCE description of a prism
genrev	- generate a RADIANCE description of surface of revolution
gensky	- generate a RADIANCE description of the sky
gensurf	- generate a RADIANCE description of a functional surface
genworm	- generate a RADIANCE description of a functional worm
getbbox	- compute bounding box for RADIANCE scene
getinfo	- get header information from a RADIANCE file
glare	- perform glare and visual comfort calculations
glarendx	- calculate glare index
ies2rad	- convert IES luminaire data to RADIANCE description
lam	- laminate lines of multiple files
lampcolor	- compute spectral radiance for diffuse emitter
lookamb	- examine ambient file values

mkillum - compute illum sources for a RADIANCE scene
neat - neaten up output columns
normpat - normalize RADIANCE pictures for use as patterns.
oconv - create an octree from a RADIANCE scene description
pcomb - combine RADIANCE pictures.
pcompos - composite RADIANCE pictures.
pfilt - filter a RADIANCE picture
pflip - flip a RADIANCE picture.
pinterp - interpolate/extrapolate view from pictures
protate - rotate a RADIANCE picture.
psign - produce a RADIANCE picture from text.
pvalue - convert RADIANCE picture to/from alternate formats
ra_bn - convert RADIANCE picture to/from Barneyscan image
ra_pixar - convert RADIANCE picture to/from PIXAR picture
ra_ppm - convert RADIANCE picture to/from a Poskanzer Portable Pixmap
ra_pr - convert RADIANCE picture to/from pixrect rasterfile
ra_pr24 - convert RADIANCE picture to/from 24-bit rasterfile
ra_rgbe - change run-length encoding of a RADIANCE picture
ra_t16 - convert RADIANCE picture to/from Targa 16 or 24-bit image file
ra_t8 - convert RADIANCE picture to/from Targa 8-bit image file
ra_tiff - convert RADIANCE picture to/from a TIFF color or greyscale image
rcalc - record calculator
replmarks - replace triangular markers in a RADIANCE scene description
rpict - generate a RADIANCE picture
rtrace - trace rays in RADIANCE scene
rview - generate RADIANCE images interactively
thf2rad - convert GDS things file to RADIANCE description
total - sum up columns
ttyimage - RADIANCE driver for X window system
xform - transform a RADIANCE scene description
xglaresrc - display glare sources under X11
ximage - RADIANCE driver for X window system
xshowtrace - interactively show rays traced on RADIANCE image under X11