## NAME

arch2rad - convert Architrion text file to RADIANCE description

## SYNOPSIS

**arch2rad** [ **−n** ][ **−m mapfile** ] [ **input** ]

## DESCRIPTION

*Arch2rad* converts an Architrion text file to a RADIANCE scene description. The material names for the surfaces will assigned based on the default mapping or the mapping rules file given in the *−m* option. A mapping file contains a list of materials followed by the conditions a surface must satisfy in order to have that material.

For example, if we wanted all surfaces for blocks with RefId "thingy" and Color 152 to use the material "wood", and all other surfaces to use the material "default", we would create the following mapping file:

```
default ;
wood (RefId "thingy") (Color 152) ;
```

All surfaces would satisfy the first set of conditions (which is empty), but only the surfaces in blocks with RefId "thingy" and Color 152 would satisfy the second set of conditions.

Each rule can have up to one condition per qualifier, and different translators use different qualifiers. In *arch2rad,* the valid qualifiers are *Layer, Color, Face* and *RefId*. A condition is either a single value for a specific attribute, or an integer range of values. (Integer ranges are specified in brackets and separated by a colon, eg. [−15:27], and are always inclusive.) A semicolon is used to indicate the end of a rule, which can extend over several lines if necessary.

The semantics of the rule are such that "and" is the implied conjunction between conditions. Thus, it makes no sense to have more than one condition in a rule for a given qualifier. If the user wants the same material to be used for surfaces that satisfy different conditions, they simply add more rules. For example, if the user also wanted surfaces in blocks with RefId "yohey" with Colors between 50 and 100 to use "wood", they would add the following rule to the end of the example above:

```
wood (Color [50:100]) (RefId "yohey") ;
```

Note that the order of conditions in a rule is irrelevant. However, the order of rules is very important, since the last rule satisfied determines which material a surface is assigned.

By convention, the identifier "void" is used to delete unwanted surfaces. A surfaces is also deleted if it fails to match any rule. Void is used in a rule as any other material, but it has the effect of excluding all matching surfaces from the translator output. For example, the following mapping would delete all surfaces in the Layer 2 except those with the color "beige", to which it would assign the material "beige_cloth", and all other surfaces would be "tacky":

```
tacky ;
void (Layer 2) ;
beige_cloth (Layer 2) (Color "beige") ;
```

If neither the *−m* nor the *−n* options are not used, *arch2rad* uses the default mapping file "/usr/local/lib/ray/lib/arch.map". This file simply assigns materials based on color, using the identifiers "c0" through "c255". Appropriate materials for these identifiers are contained in "/usr/local/lib/ray/lib/arch.mat".

The *−n* option may be used to produce a list of qualifiers from which to construct a mapping for the given Architrion file. If the *−m* option is used also, only those blocks matched in the mapping file will be added to the qualifier list.

## DETAILS

Architrion blocks are divided into about 6 polygons. The reference, opposite and end faces must all be quadrilaterals (ie. four-sided polygons), though one or more faces may disappear in certain degenerate cases. The bottom face will usually be a quadrilateral, though it may be written out as two triangles if the face is non-planar or one triangle if there is a degenerate side. The top face is treated the same as the bottom face.

Openings are currently handled using the antimatter material type. An antimatter material called "opening" is defined that "clips" all faces for the current block, and patches the edges of the hole with the material defined for the face "sill". If no rule is given specifically for the sill face, then the most specific material (ie. the material in the latest rule) for this block is used. An antimatter opening will not function properly if there is another surface intersecting it, or rendering is attempted from within the opening. Overlapping openings or openings with shared boundaries will also fail. There is currently no support of Architrion "frame" libraries.

Naming of the output faces is based on layer number, reference id and output block number (sequential from 1 to the total number of output blocks). If there is no reference id name, the layer name is used (if available) instead of the layer number. If there is a reference id number but no name, that is added as well. Names are truncated to the first 12 characters, so the ends of long names may be lost. Also, spaces in names are replaced by underscores ('_'). Finally, the face id is added to the end of the block name for each output polygon. An example identifier for a polygon is:

```
l3.window_overh.3155.ref
```

This would be the reference face of output block number 3155, reference id name "window overhangs" in layer number 3.

## EXAMPLE

To create a qualifier list for building.txt:

```
arch2rad -n building.txt > building.qual
```

To translate building.txt into a RADIANCE file using the mapping building.map:

```
arch2rad -m building.map building.txt > building.rad
```

To create an octree directly from an Architrion file using the default mapping and materials:

```
oconv source.rad /usr/local/lib/ray/lib/arch.mat '\!arch2rad building.txt' > building.oct
```

## FILES

/usr/local/lib/ray/lib/arch.map /usr/local/lib/ray/lib/arch.mat

## AUTHOR

Greg Ward

## SEE ALSO

ies2rad(1), oconv(1), thf2rad(1), xform(1)

## NAME

bgraph - do a set of batch graphs to a metafile

## SYNOPSIS

**bgraph** [ **−type ..** ][ **+variable value ..** ][ **file ..** ]

## DESCRIPTION

*Bgraph* reads each graph *file* in sequence and converts it to a plot suitable for use by a metafile driver program. If no files are given, the standard input is read.

The graph type can be defined with a *−type* option. Types are simply include files which set default values for certain variables. The actual include file name is the type concatanated with ".plt". Typical types are "scatter", "line", and "curve". A scatter graph shows only points. A line graph shows only lines connecting points. A curve graph shows both points and connecting lines.

Variables can be set explicitly with *+variable value* options. The following standard graph variables are supported:

**fthick** The frame thickness, valued from 0 to 4. A value of 0 turns the frame off.

**grid** The grid: 1 is on, 0 is off.

**include** The include file name. Graph input is taken from the file. If the file is not found in the current directory, it is searched for in a set of standard locations.

**legend** The legend title.

**othick** The origin axis thickness, valued from 0 to 4. A value of 0 turns the origin off.

**period** The period for a polar plot. For a plot in degrees, use 360. For radians, use 6.283. A value of 0 (the default) indicates a Cartesian plot.

**subtitle** The graph subtitle.

**symfile** The point symbol metafile.

**tstyle** The frame tick mark style. The default value is 1, which is outward tick marks. A value of 2 is inward ticks, 3 is cross ticks. A value of 0 disables frame tick marks.

**title** The graph title.

**xlabel** The x axis label.

**xmap** The x axis mapping function. An x axis mapping xmap(x)=log(x) produces a log x axis.

**xmax** The x axis maximum.

**xmin** The x axis minimum.

**xstep** The x axis step.

**ylabel** The y axis label.

**ymap** The y axis mapping function. An y axis mapping ymap(y)=log(y) produces a log y axis.

**ymax** The y axis maximum.

**ymin** The y axis minimum.

**ystep** The y axis step.

In addition to the standard graph variables, each curve has a set of variables. The variables for curve 'A' all begin with the letter 'A'; the variables for curve 'B' all begin with the letter 'B', and so on. Up to 8 curves are supported on a single graph, 'A' through 'H'. The variables for curve 'A' are:

**A** The function for curve 'A'. If *Adata* is undefined, *xmin, xmax* and *Anpoints* are used to determine which x values to plot. If *Adata* is defined and *A* is a function of a single variable (ie. A(x)), data values are interpreted as x values to be plotted. If *Adata* is defined and *A* is a function of two variables (ie. A(x,y)), data values are interpreted as (x,y) pairs and *A* becomes a mapping function for the data.

**Acolor** The color for curve A. The values 1-4 map to black, red, green, and blue respectively. A value of 0 turns curve A off.

**Adata** The point data for curve 'A'. If *Adata* is set to the name of a file, data is read and interpreted from that file. If *Adata* is set to a command (beginning with an exclamation, '!'), the output of the command is read as data. Otherwise, data is read from the current file. Data values are separated by white space and/or commas. A semicolon or end of file indicates the end of data.

**Alabel** The label for curve 'A'. The curve label is printed in the legend when a curve is defined.

**Alintype** The line type for curve 'A', valued from 0 to 4. A value of 0 turns line drawing off. A value of 1 is solid, 2 is dashed, 3 is dotted, and 4 is dot-dashed.

**Anpoints** The number of symbol points for curve 'A'. If *Adata* is defined, all points will be connected with the selected curve line, but only *Anpoints* points will be indicated with a symbol. This prevents messy graphs when large number of points are defined. If *A* is defined and *Adata* is not, *Anpoints* is used along with *xmin* and *xmax* to determine which x values to plot.

**Asymsize** The symbol radius for curve 'A'. The default size is 100. A value of 0 turns symbols off.

**Asymtype** This is the name of a segment in *symfile* which defines the graphics symbol for curve 'A'.

**Athick** The line thickness for curve 'A', valued from 0 to 4. A thickness of 0 turns line drawing off.

## GRAPH FILE FORMAT

A graph file contains definitions for graph and curve variables. These definitions fall one per line in the following formats:

```
vreal       = expression       # real variable
vfunction(x) = expression(x)    # function
vstring      = "string"         # string variable
vdata        = filename         # data file
vdata        = "!command"       # data generator
variable     = continued \
      line                      # newline escaped
vdata        =                  # data
            v1 v2 v3 v4 ... ;
```

Comments are preceded by a '#', and continue to the end of the line. Except for comments, the newline can be escaped with a backslash. Note that in the special case where data is contained in the graph file, a definition will continue over more than one line. Data values can be separated by commas or white space, and reading continues until a semicolon is reached. No comments are allowed in the data section of a file.

An expression is an algebraic formula containing numbers, variables, functions, and the standard operators {+,-,*,/,^,(,)} (see icalc(1)). Besides the variables described in the last section, definitions of intermediate real variables and functions are allowed for convenience. They may be used in expressions of graph and curve variables.

## EXAMPLE

A file to graph the sine function is:

```
title = "Sine Function from 0 to Pi"
PI = 3.141592653589793
A(x) = sin(x)
xmin = 0
xmax = PI
Anpoints = 100
```

Or, to graph selected points:

```
title = "Sine Function at 0, .2, .6, and .8"
A(x) = sin(x)
Adata =
        0 , .2
        .6 , .8
;
```

The commands to plot these files might be:

```
bgraph −line sine1.plt | psmeta | lpr

bgraph −curve +ymin −1 +ymax 1 sine2.plt | x11meta
```

## FILES

/usr/local/lib/meta/*.mta /usr/local/lib/meta/*.plt *.plt

## AUTHOR

Greg Ward

## BUGS

There is no mechanism provided for undefining a variable. An axis mapping function which is not invertible (monotonically increasing or decreasing) confuses the program terribly.

## SEE ALSO

dgraph(1), gcomp(1), icalc(1), igraph(1), metafile(5), x11meta(1)

## NAME

bsdf2klems - generate XML Klems matrix description of a BSDF

## SYNOPSIS

**bsdf2klems** [ **−n spp** ][ **−h|−q** ][ **−l maxlobes** ] [ **bsdf.sir ..** ]
or
**bsdf2klems** [ **−n spp** ][ **−h|−q** ] **bsdf_in.xml**
or
**bsdf2klems** [ **−n spp** ][ **−h|−q** ][ **{+|-}forward** ][ **{+|-}backward** ][ **−e expr** ][ **−f file** ] **bsdf_func**

## DESCRIPTION

*Bsdf2klems* produces a Klems matrix representation of a bidirectional scattering distribution function (BSDF) based on an intermediate representation (in the first form), an input XML representation (in the second form), or a functional description (in the third form). A complete XML description is written to the standard output, which is normally redirected to a file.

The Klems matrix representation divides the input and output hemisphere into a default 145 patches. The *−h* option may be used to reduce this number to 73 patches per hemisphere. The *−q* option may be used to reduce this number to 41 patches. Neither option is recommended unless the distribution is known to be approximately diffuse. The *−l* option may be used to specify the maximum number of lobes in any interpolated radial basis function. The default value is 15000, which generally keeps the interpolation tractable. Setting the value to 0 turns off this limit.

Normally, multiple samples are taken from random points on each input and output patch to improve accuracy. The number of samples to take for each input-output patch pair may be controlled using the *−n* option, which defaults to 256.

The first invocation form takes a intermediate scattering representation as produced by *pabopto2bsdf(1)* or similar, and produces a Klems representation with as many components as there are independent input distributions. Each intermediate scattering file contains one of the four components, and if the first component is isotropic, all components must be isotropic. A similar rule holds for anisotropic inputs. Only the center of each incident patches is sampled, due to the time required to interpolate incident positions.

In the second invocation form, an input XML representation is resampled to produce the desired Klems matrix representation. This is primarily used to convert a tensor tree representation into a matrix for annual daylighting simulations. Any components in the input are reproduced on output, and inline geometric descriptions are passed unchanged.

In the third invocation form, *bsdf2klems* takes a functional specification of a BSDF. The named function should accept 6 parameters corresponding to the normalized incident and exiting vectors, respectively. By convention, these vectors point away from the surface, and a positive Z-component corresponds to the front side. The Y-component corresponds to the "up" orientation of the surface, as specified in the eventual scene description that references the XML output. If the function only takes 3 parameters, then the variables "Dx", "Dy", and "Dz" will be assigned to the reverse of the outgoing direction at each evaluation. (I.e., the vector will point into the surface and Dz will be negative on the front side.) This simplifies conversion of functional BSDF specifications using the legacy material primitives "plasfunc", "metfunc", and "transfunc".

The function is defined by one or more *−e* and *−f* options, and should obey both Helmholtz reciprocity and integrate to less than 1 over each projected incident hemisphere for energy conservation.

Similar to the *genBSDF(1)* command, the *+backward* option (default) specifies that rays arriving from the front side of the surface will be tested for reflection and transmission. If both forward and backward (front and back) distributions are needed, the *+forward* option may be given. To turn off the backward components, use the *−backward* option. Computing both incident hemispheres takes about twice as long as one, but is recommended when rays will be impinging from either side.

## EXAMPLE

To take two components of an intermediate BSDF representation and create a full Klems matrix representation:

```
bsdf2klems transmitted.sir reflected.sir > combined.xml
```

To reduce a tensor tree representation into a half-Klems matrix representation:

```
bsdf2klems -h ttree.xml > klems_half.xml
```

To create a low-res BSDF corresponding to a one-sided, isotropic Phong distribution:

```
bsdf2klems -e 'phong(ix,iy,iz,ox,oy,oz) = if(iz, .1+((iz+oz)/sqrt((ix+ox)^2+(iy+oy)^2+(iz+oz)^2))^50, 0)' phong > phong.xml
```

## AUTHOR

Greg Ward

## SEE ALSO

bsdf2ttree(1), dctimestep(1), icalc(1), gendaymtx(1), genklemsamp(1), genskyvec(1), mkillum(1), genBSDF(1), pkgBSDF(1), rcontrib(1), rtrace(1)

## NAME

bsdf2ttree - generate XML tensor tree description of a BSDF

## SYNOPSIS

**bsdf2ttree** [ **−g Nlog2** ][ **−t pctcull** ][ **−l maxlobes** ] [ **bsdf.sir ..** ]

or

**bsdf2ttree −t{3|4}** [ **−g Nlog2** ][ **−t pctcull** ][ **{+|-}forward** ][ **{+|-}backward** ][ **−e expr** ][ **−f file** ] **bsdf_func**

## DESCRIPTION

*Bsdf2ttree* produces a tensor tree representation of a bidirectional scattering distribution function (BSDF) based on an intermediate representation (in the first form) or a functional description (in the second form). A complete XML description is written to the standard output, which is normally redirected to a file.

The maximum resolution of the tensor tree may be controlled by the *−g* option, which defaults to a value of 6. This corresponds to a peak resolution of 2^6 (64) in each dimension. Due to memory and time constraints, it is not recommended to set *−g* higher than 7, which corresponds to a 128x128x128x128 initial sampling, or 268 million values.

The initial sampling is pared down by the percentage specified with the *−t* option, which defaults to 90. Setting this parameter to -1 turns culling off entirely, which may be useful for comparisons.

The first invocation form takes a intermediate scattering representation as produced by *pabopto2bsdf(1)* or similar, and produces a tensor tree representation with as many components as there are independent input distributions. Each intermediate scattering file contains one of the four components, and if the first component is isotropic, all components must be isotropic. A similar rule holds for anisotropic inputs. The *−l* option may be used to specify the maximum number of lobes in any interpolated radial basis function. The default value is 15000, which generally keeps the interpolation tractable. Setting the value to 0 turns off this limit.

In the second invocation form, *bsdf2ttree* takes a functional specification of a BSDF. The named function should accept 6 parameters corresponding to the normalized incident and exiting vectors, respectively. By convention, these vectors point away from the surface, and a positive Z-component corresponds to the front side. The Y-component corresponds to the "up" orientation of the surface, as specified in the eventual scene description that references the XML output. If the function only takes 3 parameters, then the variables "Dx", "Dy", and "Dz" will be assigned to the reverse of the outgoing direction at each evaluation. (I.e., the vector will point into the surface and Dz will be negative on the front side.) This simplifies conversion of functional BSDF specifications using the legacy material primitives "plasfunc", "metfunc", and "transfunc".

The function is defined by one or more *−e* and *−f* options, and should obey both Helmholtz reciprocity and integrate to less than 1 over each projected incident hemisphere for energy conservation. If the *−t3* option is specified, the defined function is assumed to be isotropic. If the *−t4* option is given, the function is assumed to be anisotropic.

Similar to the *genBSDF(1)* command, the *+backward* option (default) specifies that rays arriving from the front side of the surface will be tested for reflection and transmission. If both forward and backward (front and back) distributions are needed, the *+forward* option may be given. To turn off the backward components, use the *−backward* option. Computing both incident hemispheres takes about twice as long as one, but is recommended when rays will be impinging from either side.

## EXAMPLE

To take two components of an intermediate BSDF representation and create a high-resolution tensor tree with 85% culling:

```
bsdf2ttree -g 7 -t 85 transmitted.sir reflected.sir > combined.xml
```

To create a low-res BSDF corresponding to a one-sided, isotropic Phong distribution:

```
bsdf2ttree -g 5 -t3 -e 'phong(ix,iy,iz,ox,oy,oz) = if(iz,
.1+((iz+oz)/sqrt((ix+ox)^2+(iy+oy)^2+(iz+oz)^2))^50, 0)' phong > phong.xml
```

## AUTHOR

Greg Ward

## SEE ALSO

bsdf2klems(1), icalc(1), genBSDF(1), pkgBSDF(1)

## NAME

cnt - index counter

## SYNOPSIS

**cnt** N ..

## DESCRIPTION

*Cnt* counts from 0 to N-1, producing N lines of output. If multiple arguments are given, *cnt* produces a nested array of values where the final counter rotates fastest through its range. *Cnt* is most useful in conjunction with *rcalc(1)* to produce array values.

## EXAMPLE

To create a 3 by 5 array:

```
cnt 3 5
```

## AUTHOR

Greg Ward

## SEE ALSO

neaten(1), rcalc(1), rlam(1), total(1)

## NAME

compamb - compute good ambient value for a rad input file

## SYNOPSIS

**compamb** [ **−c** ][ **−e** ] **rad_input_file**

## DESCRIPTION

*Compamb* computes a good ambient value for the specified *rad(1)* variable file and appends it to the file as a "render= −av" option. If the *−c* option is specified, then *compamb* includes color information in the computed ambient value, rather than estimating a grey value to avoid rendering color shifts. If the *−e* option is specified, then *compamb* also computes a good exposure value for this scene, and appends it to the *rad* file as well.

*Compamb* is a shell script that makes calls to other RADIANCE programs and utilities to do the actual work. A substantial amount of time may be required to complete this script, since *compamb* calls *rpict(1)* to render low resolution frames for each view in the *rad* file, setting "QUALITY=High" to compute interreflections. The resulting ambient file is thrown away, since it would disagree with the new −av setting used for the final renderings. This method is preferable to setting the *−aw* option of *rpict,* which frequently results in splotchy artifacts.

## AUTHOR

Greg Ward Larson

## SEE ALSO

lookamb(1), rad(1), rpict(1)

## NAME

cv - convert between metafile formats

## SYNOPSIS

**cv** [ −r ][ file .. ]

## DESCRIPTION

*Cv* reads each human readable metafile *file* in sequence and converts it to a binary form. If the option −*r* is specified, the reverse conversion is performed.

If no input files are specified, the standard input is read.

## EXAMPLE

To convert the binary file meta.bin to its human-readable equivalent, and put the result in meta.human

```
cv −r meta.bin > meta.human
```

## AUTHOR

Greg Ward

## SEE ALSO

meta(3), metafile(5), pexpand(1), psort(1)

## NAME

dayfact - compute illuminance and daylight factor on workplane

## SYNOPSIS

**dayfact** [ falsecolor options ]

## DESCRIPTION

*Dayfact* is an interactive script for computing workplane illuminance, and daylight factors and potential daylight savings using *rtrace(1)*. The script *falsecolor(1)* is then used to draw contour lines on the resulting Radiance picture.

## AUTHOR

Greg Ward

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland.

## SEE ALSO

falsecolor(1), glare(1), rtrace(1), ximage(1)

## NAME

dctimestep - compute annual simulation time-step(s) via matrix multiplication

## SYNOPSIS

**dctimestep** [ **−n nsteps** ][ **−o ospec** ][ **−i{fld}** ][ **−o{fld}** ] **DCspec** [ **skyf** ]
**dctimestep** [ **−n nsteps** ][ **−o ospec** ][ **−i{fld}** ][ **−o{fld}** ] **Vspec Tbsdf.xml Dmat.dat** [ **skyf** ]

## DESCRIPTION

*Dctimestep* has two invocation forms. In the first form, *dctimestep* is given a daylight coefficient specification and an optional sky vector or matrix, which may be read from the standard input if unspecified. The daylight coefficients are multiplied against these sky values and the results are written to the standard output. This may be a list of color values or a combined Radiance image, as explained below.

In the second form, *dctimestep* takes four input files, forming a matrix expression. The first argument is the View matrix file that specifies how window output directions are related to some set of measured values, such as an array of illuminance points or images. This matrix is usually computed by *rcontrib(1)* for a particular set of windows or skylight openings. The second argument is the window transmission matrix, or BSDF, given as a standard XML description. The third argument is the Daylight matrix file that defines how sky patches relate to input directions on the same opening. This is usually computed using *genklemsamp(1)* with *rcontrib* in a separate run for each window or skylight orientation. The last file is the sky contribution vector or matrix, typically computed by *genskyvec(1)* or *gendaymtx(1),* and may be passed on the standard input. This data is assumed by default to be in ASCII format, whereas the formats of the View and Daylight matrices are detected automatically if given as binary data. The *−if* or *−id* option may be used to specify that sky data is in float or double format, respectively, which is more efficient for large matrices. (Note that binary double data may not be read from stdin.)

Sent to the standard output of *dctimestep* is either a color vector with as many RGB triplets as there are rows in the View matrix, or a combined *Radiance* picture. Which output is produced depends on the first argument. A regular file name will be loaded and interpreted as a matrix to generate a color results vector. A file specification containing a '%d' format string will be interpreted as a list of *Radiance* component pictures, which will be summed according to the computed vector.

The *−n* option may be used to compute multiple time steps in a single invocation. The sky input file must contain the number of columns specified in each sky patch row, whether it is read from the standard input or from a file. The columns do not need to be given on the same line, so long as the number of values totals 3*Nsteps*Npatches. Input starts from the first patch at the first time step, then the first patch at the second time step, and so on.

The *−o* option may be used to specify a file or a set of output files to use rather than the standard output. If the given specification contains a '%d' format string, this will be replaced by the time step index, starting from 1. In this way, multiple output pictures may be produced, or separate result vectors (one per time step). If the standard output is used or the *−o* option specifies a single output file, then an information header will precede the output. This can be removed if desired using the *getinfo* command with a single hyphen ('-') argument.

The *−of* or *−od* option may be used to specify IEEE float or double binary output data, respectively. This enables *dctimestep* to be used as a pure matrix multiplier, as the output file with header specifying the format is suitable for subsequent calls.

## EXAMPLES

To compute workplane illuminances at 3:30pm on Feb 10th:

```
gensky 2 10 15:30 | genskyvec | dctimestep workplaneDC.dmx > Ill_02-10-1530.dat
```

To compute an image at 10am on the equinox from a set of component images:

```
gensky 3 21 10 | genskyvec | dctimestep dcomp%03d.hdr > view_03-21-10.hdr
```

To compute a set of illuminance contributions for Window 1 on the Winter solstice at 2pm:

```
gensky 12 21 14 | genskyvec | dctimestep IllPts.vmx Blinds20.xml Window1.dmx > Ill_12-21-14.dat
```

To compute Window2's contribution to an interior view at 12 noon on the Summer solstice:

```
gensky 6 21 12 | genskyvec | dctimestep view%03d.hdr Blinds30.xml Window2.dmx > view_6-21-12.hdr
```

To generate an hourly matrix of sensor value contributions from Skylight3 using a 3-phase calculation, where output columns are time steps:

```
gendaymtx -of Tampa.wea | dctimestep -if -n 8760 WPpts.vmx shade3.xml Skylight3.dmx > wp_win3.dat
```

Generate a series of pictures corresponding to timesteps in an annual simulation:

```
gendaymtx NYCity.wea | dctimestep -n 8760 -o tstep%04d.hdr dcomp%03d.hdr
```

To multiply two color matrices (second matrix is IEEE-float with 145 RGB columns) into a IEEE-double result (also 145 RGB columns):

```
getinfo - < Inp2.fmx | dctimestep -n 145 -if -od Inp1.fmx > Inp1xInp2.dmx
```

## AUTHOR

Greg Ward

## SEE ALSO

gendaymtx(1), genklemsamp(1), genskyvec(1), getinfo(1), mkillum(1), rcollate(1), rcontrib(1), rtrace(1), vwrays(1)

## NAME

dgraph - do a set of graphs to a dumb terminal

## SYNOPSIS

**dgraph** [ **-w width** ][ **-l length** ][ **+variable value ..** ][ **file ..** ]

## DESCRIPTION

*Dgraph* reads each graph *file* in sequence and converts it to a character plot displayable on any ascii device. If no files are given, the standard input is read.

Across the top of the plot, the extrema are printed. This is the only indication of the axis size. Curves are represented with their respective letter ('A' for curve A, etc.) at each point. Where two or more curves cross, a number is shown instead.

The size of the output array can be specified as a certain *width* and *length*. The default size is 79 by 22.

Variables can be set explicitly with +*variable value* options. See bgraph(1) for details.

## EXAMPLE

To get a quick glimpse of the sine function from 0 to 4.

```
dgraph
A(x)=sin(x)
Anpoints=100
xmin=0
xmax=4
^D
```

## AUTHOR

Greg Ward

## BUGS

There is no mechanism provided for undefining a variable.

## SEE ALSO

bgraph(1), gcomp(1), icalc(1), igraph(1)

## NAME

epw2wea - weather file converter

## SYNOPSIS

**epw2wea file_name.epw file_name.wea**

## DESCRIPTION

*epw2wea* transforms an EnergyPlus weather data (.epw) file into the DAYSIM weather file format, for use with the RADIANCE gendaymtx program.

## AUTHOR

Christoph Reinhart, Jason Turner

## SEE ALSO

gendaymtx(1)

## NAME

ev - evaluate expressions

## SYNOPSIS

**ev 'expr' ..**

## DESCRIPTION

*Ev* evaluates expressions given on the command line, and sends the results to the standard output, one per line. An expression contains real numbers, function calls, and the following operators:

```
+ - * / ^
```

Operators are evaluated left to right, except '^', which is right associative. Powers have the highest precedence; multiplication and division are evaluated before addition and subtraction. Expressions can be grouped with parentheses. All values are double precision real.

The following library of functions is available:

**if(cond, then, else)**
: if cond is greater than zero, then is evaluated, otherwise else is evaluated.

**select(N, a1, a2, ..)**
: return aN (N is rounded to the nearest integer). If *N* is zero, the number of available arguments is returned.

**rand(x)** compute a random number between 0 and 1 based on x.

**floor(x)** return largest integer not greater than x.

**ceil(x)** return smallest integer not less than x.

**sqrt(x)** return square root of x.

**exp(x)** compute e to the power of x (e approx = 2.718281828).

**log(x)** compute the logarithm of x to the base e.

**log10(x)** compute the logarithm of x to the base 10.

**sin(x), cos(x), tan(x)**
: trigonometric functions.

**asin(x), acos(x), atan(x)**
: inverse trigonometric functions.

**atan2(y, x)** inverse tangent of y/x (range –pi to pi).

## EXAMPLE

To pass the square root of two and the sine of .5 to a program:

```
program `ev 'sqrt(2)' 'sin(.5)'`
```

## AUTHOR

Greg Ward

## SEE ALSO

icalc(1), rcalc(1)

## NAME

falsecolor – make a false color RADIANCE picture

## SYNOPSIS

**falsecolor** [ **–i input** ][ **–p picture** ][ **–cb | –cl | –cp** ][ **–e** ][ **–s scale** ][ **–l label** ][ **–n ndivs** ][ **–lw lwidth** ][ **–lh lheight** ][ **–log decades** ][ **–m mult** ][ **–pal palette** ][ **–r redv** ][ **–g grnv** ][ **–b bluv** ]

**falsecolor –palettes**

## DESCRIPTION

*Falsecolor* produces a false color picture for lighting analysis. Input is a rendered Radiance picture.

By default, luminance is displayed on a linear scale from 0 to 1000 cd/m2, where dark areas are purple and brighter areas move through blue, green, red to yellow. A different scale can be given with the *–s* option. If the argument given to *–s* begins with an "a" for "auto," then the maximum is used for scaling the result. The default multiplier is 179, which converts from radiance or irradiance to luminance or illuminance, respectively. A different multiplier can be given with *–m* to get daylight factors or whatever. For a logarithmic rather than a linear mapping, the *–log* option can be used, where *decades* is the number of decades below the maximum scale desired.

A legend is produced for the new image with a label given by the *–l* option. The default label is "cd/m2", which is appropriate for standard Radiance images. If the *–i* option of *rpict(1)* was used to produce the image, then the appropriate label would be "Lux".

If contour lines are desired rather than just false color, the *–cl* option can be used. These lines can be placed over another Radiance picture using the *–p* option. If the input picture is given with *–ip* instead of *–i,* then it will be used both as the source of values and as the picture to overlay with contours. The *–cb* option produces contour bands instead of lines, where the thickness of the bands is related to the rate of change in the image. The *–cp* option creates a posterization effect where colours are banded without the background image showing through. The *–n* option can be used to change the number of contours (and corresponding legend entries) from the default value of 8. The *–lw* and *–lh* options may be used to change the legend dimensions from the default width and height of 100x200. A value of zero in either eliminates the legend in the output.

The *–e* option causes extrema points to be printed on the brightest and darkest pixels of the input picture.

The *–pal* option provides different color palettes for *falsecolor.* The current choices are *spec* for the old spectral mapping, *hot* for a thermal scale, *eco* for a blue-red-yellow scale, and *pm3d* for a variation of the default mapping, *def.* A Radiance HDR image of all available palettes can be created with the *–palettes* option. The remaining options, *–r, –g,* and *–b* are for changing the mapping of values to colors. These are expressions of the variable *v,* where *v* varies from 0 to 1. These options are not recommended for the casual user.

If no *–i* or *–ip* option is used, input is taken from the standard input. The output image is always written to standard output, which should be redirected.

## EXAMPLES

To create a false color image directly from *rpict(1):*

```
rpict –vf default.vp scene.oct | falsecolor > scene.hdr
```

To show the available color palettes:

```
falsecolor -palettes | ximage
```

To create a logarithmic contour plot of illuminance values on a Radiance image:

```
rpict –i –vf default.vp scene.oct > irrad.hdr
rpict –vf default.vp scene.oct > rad.hdr
falsecolor –i irrad.hdr –p rad.hdr –cl –log 2 –l Lux > lux.hdr
```

## AUTHOR

Greg Ward
Axel Jacobs (Perl translation and –pal options)

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland. The 'eco' palette was sponsored by Foster + Partners in London.

## SEE ALSO

getinfo(1), pcomb(1), pcompos(1), pextrem(1), pfilt(1), pflip(1), protate(1), psign(1), rpict(1), ximage(1)

## NAME

fieldcomb - combine two or more field frames for video animation

## SYNOPSIS

**fieldcomb** [ **-el-o** ][ **−r** ] [ **-f combined.hdr** ] **field0.hdr field1.hdr ..**

## DESCRIPTION

*Fieldcomb* combines two or more field frames into half as many final video frames for field-based animations. Normally, the even scanlines in the output are given over to the first, third, fifth, etc. pictures given on the input, and the odd scanlines are given over to the second, fourth, sixth, etc. pictures on the input. This behavior corresponds to the *−e* option, which is set by default. To reverse this ordering, the *−o* option must be specified.

Normally, *fieldcomb* will name output frames by dividing the numbers on the input field file names by 2, inserting a 'C' right before the number, adding leading zeroes to make it always four digits. For example, the command:

fieldcomb anim12.hdr anim13.hdr anim14.hdr anim15.hdr

will produce two output pictures:

animC0006.hdr & animC0007.hdr

If an odd number of input files is given on the command line, the last file is held over and used as the first file in the next invocation of *fieldcomb* in the same directory. In this way, the program works well with the "TRANSFER" facility of *ranimate(1),* which does not guarantee an even number of frames will be passed every time.

If only two input files are given, the *−f* option may be used to specify a different output name of the user's choosing.

If the *−r* option is specified, then the input files will be removed once they have been combined.

## AUTHOR

Greg Ward

## ACKNOWLEDGEMENT

Work on this program was sponsored by Iebele Abel.

## SEE ALSO

getinfo(1), pcomb(1), pfilt(1), phisto(1), ran2tiff(1), ranimate(1), rpict(1)

## NAME

findglare - locate glare sources in a RADIANCE scene

## SYNOPSIS

**findglare** [ **−v** ][ **−ga angles** ][ **−t threshold** ][ **−r resolution** ][ **−c** ][ **−p picture** ][ view options ] [[ rtrace options ] **octree** ]

## DESCRIPTION

*Findglare* locates sources of glare in a specific set of horizontal directions by computing luminance samples from a RADIANCE picture and/or octree. *Findglare* is intended primarily as a preprocessor for glare calculation programs such as *glarendx(1),* and is usually accessed through the executive script *glare(1).*

If only an octree is given, *findglare* calls rtrace to compute the samples it needs. If both an octree and a picture are specified, *findglare* calls rtrace only for samples that are outside the frame of the picture. If *findglare* does not have an octree and the picture does not completely cover the area of interest, a warning will be issued and everything outside the picture will be treated as if it were black. It is preferable to use a picture with a fisheye view and a horizontal and vertical size of at least 180 degrees (more horizontally if the *−ga* option is used -- see below). Note that the picture file must contain correct view specifications, as maintained by *rpict(1), rvu(1), pfilt(1)* and *pinterp(1).* Specifically, *findglare* will not work on pictures processed by *pcompos(1)* or *pcomb(1).* It is also essential to give the proper rtrace options when an octree is used so that the calculated luminance values are correct.

The output of *findglare* is a list of glare source directions, solid angles and average luminances, plus a list of indirect vertical illuminance values as a function of angle. Angles are measured in degrees from the view center, with positive angles to the left and negative angles to the right.

By default, *findglare* only computes glare sources and indirect vertical illuminance for the given view (taken from the picture if none is specified). If the view direction is not horizontal to begin with (ie. perpendicular to the view up vector), *findglare* will substitute the closest horizontal direction as its view center. The *−ga* option can be used to specify a set of directions to consider about the center of view. This specification is given by a starting angle, ending angle, and step angle like so:

start-end:step

All angles must be whole degrees within the range 1 to 180. Multiple angle ranges may be separated by commas, and individual angles may be given without the ending and step angles. Note that *findglare* will complain if the same angle is given twice either explicitly or implicitly by two ranges.

*Findglare* normally identifies glare sources as directions that are brighter than 7 times the average luminance level. It is possible to override this determination by giving an explicit luminance threshold with the *−t* option. It usually works best to use the 'l' command within *ximage(1)* to decide what this value should be. Alternatively, one can use the 't' command within *rvu(1).* The idea is to pick a threshold that is well above the average level but smaller than the source areas.

If the sources in the scene are small, it may be necessary to increase the default sample resolution of *findglare(1)* using the *−r* option. The default resolution is 150 vertical samples and a proportional number of horizontal samples. If besides being small, the sources are not much brighter than the threshold, the *−c* flag should be used to override *findglare's* default action of absorbing small sources it deems to be insignificant.

The *−v* flag switches on verbose mode, where *findglare* reports its progress during the calculation.

## EXAMPLE

To calculate the glare sources in the image "scene.hdr":

```
findglare −p scene.hdr > scene.glr
```

To compute the Guth visual comfort probability from this result:

```
glarendx −t guth_vcp scene.glr
```

To compute the glare for a set of angles around the view "good.vp" from the octree "scene.oct" using an ambient level of .1:

```
findglare −vf good.vp −ga 10-60:10 −av .1 .1 .1 scene.oct > scene.glr
```

## AUTHOR

Greg Ward

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland.

## SEE ALSO

getinfo(1), glare(1), glarendx(1), pfilt(1), rpict(1), rtrace(1), rvu(1), xglaresrc(1), ximage(1)

## NAME

gcomp - do computations on a graph file.

## SYNOPSIS

**gcomp** [ **-amilh** ][ **+variable value ..** ][ **file ..** ]

## DESCRIPTION

*Gcomp* reads each graph *file* in sequence and computes the specified calculations. The type options are as follows:

**−n** Print the name of each curve.

**−a** Print average and standard deviation of each curve.

**−m** Print minimum and maximum for each curve.

**−i** Print Romberg's approximation to the integral of each curve.

**−l** Print the slope, intercept, and correlation coefficient using the least squares method of linear regression.

**−h** Do not print a header in the output.

The calculations will be displayed as columns in the order they are specified on the command line. If no files are given, the standard input is read.

Variables can be set explicitly with *+variable value* options. The only truely useful variables for this program are xmin and xmax. They determine boundaries for the calculations.

## EXAMPLE

To compute the approximate integral of sin(x)/log(x) from 2 to 4:

```
gcomp −i
A(x)=sin(x)/log(x);
Anpoints=100;
xmin=2;
xmax=4;
^D
```

## AUTHOR

Greg Ward

## BUGS

Only the y values can be used for computation.

## SEE ALSO

bgraph(1), dgraph(1), icalc(1), igraph(1)

## NAME

genBSDF - generate BSDF description from Radiance or MGF input

## SYNOPSIS

**genBSDF** [ **−c Nsamp** ][ **−n Nproc** ][ **−r 'rcontrib opts...'** ][ **−t{3|4} Nlog2** ][ **{+|-}forward** ][ **{+|-}backward** ][ **{+|-}mgf** ][ **{+|-}geom unit** ][ **−dim Xmin Xmax Ymin Ymax Zmin Zmax** ] [ **geom ..** ]

## DESCRIPTION

*GenBSDF* computes a bidirectional scattering distribution function from a Radiance or MGF scene description given on the input. The program assumes the input is in Radiance format unless the *+mgf* option is specified. The output conforms to the LBNL Window 6 XML standard for BSDF data, and will include an MGF representation of the input geometry if the *+geom* option is given, followed by one of "meter," "foot," "inch," "centimeter," or "millimeter," depending on the scene units. The default is to include the provided geometry, which is assumed to be in meters. Geometry output can be supressed with the *−geom* option, which must also be followed by one of the above length units.

Normally, *genBSDF* computes components needed by a backwards ray-tracing process, *+backward*. If both forward and backward (front and back) distributions are needed, the *+forward* option may be given. To turn off backward components, use the *−backward* option. Computing both components takes about twice as long as one component, but is recommended when rays will be impinging from either side.

The geometry must fit a rectangular profile, whose width is along the X-axis, height is in the Y-axis, and depth is in the Z-axis. The positive Z-axis points into the room, and the input geometry should not extend into the room. (I.e., it should not contain any positive Z values, since the putative emitting surface is assumed to lie at Z=0.) The entire window system should be modeled, including sills and edge geometry anticipated in the final installation, otherwise accuracy will be impaired. Similarly, materials in the description should be carefully measured.

Normally, the input geometry will be positioned according to its actual bounding box, but this may be overridden with the *−dim* option. Use this in cases where the fenestration system is designed to fit a smaller (or larger) opening or is offset somehow.

The variance in the results may be reduced by increasing the number of samples per incident direction using the *−c* option. This value defaults to 2000 samples distributed over the incoming plane for each of the 145 Klems hemisphere directions.

On multi-core machines, processing time may be reduced by the *−n* option, which specifies the number of simultaneous processes to run in *rcontrib(1)*. The *−r* option may be used to specify a set of quoted arguments to be included on the *rcontrib* command line.

The *−t4* mode computes a non-uniform BSDF represented as a rank 4 tensor tree, suitable for use in the Radiance rendering tools. The parameter given to this option is the log to the base 2 of the sampling resolution in each dimension, and must be an integer. The *−c* setting should be adjusted so that an appropriate number of samples lands in each region. A *−t4* parameter of 5 corresponds to 32x32 or 1024 output regions, so a *−c* setting of 10240 would provide 10 samples per region on average. Increasing the resolution to 6 corresponds to 64x64 or 4096 regions, so the *−c* setting would need to be increased by a factor of 4 to provide the same accuracy in each region.

The *−t3* mode is similar to *−t4* but computes a rank 3 tensor tree rather than rank 4. This provides a much faster computation, but only works in special circumstances. Specifically, do NOT use this option if the system is not in fact isotropic. I.e., only use *−t3* when you are certain that the system has a high degree of radial symmetry. Again, the parameter to this option sets the maximum resolution as a power of 2 in each dimension, but in this case there is one less dimension being sampled.

## EXAMPLE

To create a BSDF description including geometry from a set of venetian blinds:

```
genblinds blind_white blind1 .07 3 1.5 30 40 | xform -rz -90 -rx 90 > blind1.rad
genBSDF -r @rtc.opt blind_white.mat glazing.rad blind1.rad > blind1.xml
```

To create a non-uniform, anisotropic BSDF distribution with a maximum resolution of 128x128 from the same description:

```
genBSDF -r @rtc.opt -t4 7 -c 160000 blind_white.mat glazing.rad blind1.rad > blind12.xml
```

## NOTES

The variable resolution (tensor tree) BSDF representation is not supported by all software and applicatons, and should be used with caution. It provides practical, high-resolution data for use in the Radiance rendering programs, but does not work in the matrix formulation of the daylight coefficient method for example. Also, third party tools generally expect or require a fixed number of sample directions using the Klems directions or similar.

## AUTHOR

Greg Ward

## SEE ALSO

dctimestep(1), gendaymtx(1), genklemsamp(1), genskyvec(1), mkillum(1), pkgBSDF(1), rcontrib(1), rtrace(1)

## NAME

genblinds - generate a RADIANCE description of venetian blinds

## SYNOPSIS

**genblinds mat name depth width height nslats angle** [ **−r|+r rcurv** ]

## DESCRIPTION

*Genblinds* produces a RADIANCE scene description of a set of venetian blinds. The *depth* of the blinds (X dimension) is given first, followed by the *width* (Y dimension), followed by the *height* (Z dimension). The number of slats to place evenly within this height is given as *nslats*. The *angle* of the blind, where zero is perfectly horizontal and a positive angle tilts the positive X edge upwards, is given in degrees. The blinds are initially situated so that the corner of the bottom blind is *height/nslats/2* above the XY plane, and all coordinates are positive. Each new slat is placed *height/nslats* above the previous one, until the top slat is at *height - height/nslats/2* . The blinds may of course be moved from this starting point with the *xform(1)* command.

If curved blinds are desired, a radius of curvature may be given with the *+/-r* option. If given as *+r,* The curvature is upward (which is the usual configuration). If the option is given as *-r,* then the curvature is downward. The radius indicates how far from each slat its effective cylindrical center resides. Each slat will be broken into as many polygons as is necessary to keep the delta changes in angle less than 10 degrees. (Note that this may result in a rather large number of polygons.)

## EXAMPLE

To produce a curved set of blinds with 25 slats at 15 degrees:

```
genblinds white blind 1 46 88 25 15 −r 1 > blinds.rad
```

## AUTHOR

Jean-Louis Scartezzini and Greg Ward

## SEE ALSO

genbox(1), genrev(1), gensurf(1), genworm(1), rpict(1), rvu(1), xform(1)

## NAME

genbox - generate a RADIANCE description of a box

## SYNOPSIS

**genbox mat name xsiz ysiz zsiz** [ **−i** ][ **−r rad** | **−b bev** ]

## DESCRIPTION

*Genbox* produces a RADIANCE scene description of a parallelepiped with one corner at the origin and the opposite corner at *(xsiz, ysiz, zsiz)*. The sides of the box will be parallel to the three coordinate planes. The surfaces that make up the box will be modified by *mat* and their identifiers will begin with *name*. The *−i* option can be used to produce a box with inward directed surface normals. The *−r* option can be used to specify the radius for rounded edges. The *−b* option can be used to specify the indentation for beveled edges.

## EXAMPLE

To produce a rectangular box made of wood with beveled edges:

```
genbox wood box1 5 8 3 −b .5 > box1
```

## AUTHOR

Greg Ward

## BUGS

Because spheres and cylinders are used to construct boxes with rounded edges, a transparent box of this type appears quite messy.

## SEE ALSO

genrev(1), gensurf(1), genworm(1), rpict(1), rvu(1), xform(1)

## NAME

genclock - generate a RADIANCE description of a clock

## SYNOPSIS

**genclock** [ **−f face_mat** ][ **−c case_mat** ][ **−n name** ] { **HH:MM** | **hours** }

## DESCRIPTION

*Genclock* produces a RADIANCE scene description of an analog clock showing the given hour. The hour may either be given as *HH:MM* or decimal *hours*.

The face of the clock will have a radius of 1.0 units, with the surrounding case 1.1 (2.2 diameter). The origin is at the center of the back, and the face looks in the positive X-direction. The 12 o'clock direction corresponds to the positive Z-axis. (The Y-axis direction is 3 o'clock.) The *xform(1)* command may be used to resize and relocate the clock as desired.

Normally, *genclock* produces all of the materials necessary for its own description, but options are provided to specify alternate materials for the face and case. The numbers on the face are in dark lettering, so the face material must be relatively light for them to show up well. By default, the clock is given the name "clock," but this may be changed with the *−n* option.

## EXAMPLE

To produce a 12 inch diameter clock showing 10:35 and hang it at 60 on a wall facing the Y-direction at Y=10:

```
genclock 10:35 | xform −s 6 −rz 90 −t 20 10 60
```

## AUTHOR

Greg Ward

## SEE ALSO

genbox(1), genrev(1), gensurf(1), genworm(1), rpict(1), rvu(1), xform(1)

## NAME

gendaylit - generates a RADIANCE description of the daylight sources using Perez models for direct and diffuse components

## SYNOPSIS

**gendaylit month day hour [-P|-W|-L|-G|-E] input_value(s) [ options ]**
**gendaylit -ang altitude azimuth [-P|-W|-L|-G|-E] input_value(s) [ options ]**

## DESCRIPTION

*Gendaylit* produces a RADIANCE scene description based on an angular distribution of the daylight sources (direct+diffuse) for the given atmospheric conditions (direct and diffuse component of the solar radiation), date and local standard time. The default output is the radiance of the sun (direct) and the sky (diffuse) integrated over the visible spectral range (380-780 nm). We have used the calculation of the sun's position and the ground brightness models which were programmed in *gensky*.

The diffuse angular distribution is calculated using the Perez et al. sky luminance distribution model (see Solar Energy Vol. 50, No. 3, pp. 235-245, 1993) which, quoting Perez, describes "the mean instantaneous sky luminance angular distribution patterns for all sky conditions from overcast to clear, through partly cloudy, skies". The correctness of the resulting sky radiance/luminance values in this simulation is ensured through the normalization of the modelled sky diffuse to the measured sky diffuse irradiances/illuminances.

As described below, the radiation can be defined with the pairs direct-normal and diffuse-horizontal irradiance (-W option), direct-horizontal and diffuse-horizontal irradiance (-G option), direct-normal and diffuse-horizontal illuminance (-L option) or global-horizontal irradiation alone (-E option). The direct-normal radiation is understood here as the radiant flux coming from the sun and an area of approximately 3 degrees around the sun (World Meteorological Organisation specifications for measuring the direct radiation. The aperture angle of a pyrheliometer is approximately 6 degrees). To simplify the calculations for the direct radiation, the sun is represented as a disk and no circumsolar radiation is modelled in the 3 degrees around the sun. This means that all the measured/evaluated direct radiation is added to the 0.5 degree sun source.

The direct and diffuse solar irradiances/illuminances are the inputs needed for the calculation. These quantities are the commonly accessible data from radiometric measurement centres or from the Test Reference Year. The use of such data is the recommended method for achieving the most accurate simulation results.

The atmospheric conditions are modelled with the Perez et al. parametrization (see Solar Energy Vol. 44, No 5, pp. 271-289, 1990), which is dependent on the values for the direct-normal and the diffuse-horizontal irradiances. The three parameters are epsilon, delta and the solar zenith angle. "Epsilon variations express the transition from a totally overcast sky (epsilon=1) to a low turbidity clear sky (epsilon>6); delta variations reflect the opacity/thickness of the clouds". Delta can vary from 0.05 representing a dark sky to 0.5 for a very bright sky. Not every combination of epsilon, delta and solar zenith angle is possible. For a clear day, if epsilon and the solar zenith angle are known, then delta can be determined. For intermediate or overcast days, the sky can be dark or bright, giving a range of possible values for delta when epsilon and the solar zenith are fixed. The relation between epsilon and delta is represented in a figure on page 393 in Solar Energy Vol.42, No 5, 1989. Note that the epsilon parameter is a function of the solar zenith angle. It means that a clear day will not be defined by fixed values of epsilon and delta. Consequently the input parameters, epsilon, delta and the solar zenith angle, have to be determined on a graph. It might be easier to work with the measured direct and diffuse components (direct normal irradiance/illuminance and diffuse horizontal irradiance/illuminance) than with the epsilon and delta parameters.

The conversion of irradiance into illuminance for the direct and the diffuse components is determined by the luminous efficacy models of Perez et al. (see Solar Energy Vol. 44, No 5, pp. 271-289, 1990). To convert the luminance values into radiance integrated over the visible range of the spectrum, we devide the luminance by the white light efficacy factor of 179 lm/W. This is consistent with the RADIANCE calculation because the luminance will be recalculated from the radiance integrated over the visible range by:

luminance = radiance_integrated_over_visible_range * 179 or luminance = (RED*.263 + GREEN*.655 + BLUE*.082) * 179 with the capability to model colour (where radiance_integrated_over_visible_range == (RED + GREEN + BLUE)/3).

From *gensky, if the hour is preceded by a plus sign ('+'), then it is interpreted as local solar time instead of standard time*. The second form gives the solar angles explicitly. The altitude is measured in degrees above the horizon, and the azimuth is measured in degrees west of South. The x axis points east, the y axis points north, and the z axis corresponds to the zenith. The actual material and surface(s) used for the sky is left up to the user. In addition to the specification of a sky distribution function, *gendaylit* suggests an ambient value in a comment at the beginning of the description to use with the *−av* option of the RADIANCE rendering programs. (See rview(1) and rpict(1).) This value is the cosine-weighted radiance of the sky in W/sr/m^2.

*Gendaylit* can be used with the following input parameters. They offer three possibilities to run it: with the Perez parametrization, with irradiance values and with illuminance values.

**−P** *epsilon delta* (these are the Perez parameters)

**−W** *direct-normal-irradiance* (W/m^2), *diffuse-horizontal-irradiance* (W/m^2)

**−G** *direct-horizontal-irradiance* (W/m^2), *diffuse-horizontal-irradiance* (W/m^2)

**−L** *direct-normal-illuminance* (lm/m^2), *diffuse-horizontal-illuminance* (lm/m^2)

**−E** *global-horizontal-irradiance* (W/m^2)

The -E option calculates the diffuse irradiance fraction with the model of Erbs, Klein and Duffie (Solar Energy 28/4, 1982), being followed by the calculation of the -G option. Due to the high uncertainty of the model, the results have to be handled with care. A second irradiance value, if available, is definitely recommended.

The output can be set to either the radiance of the visible radiation, the solar radiance (full spectrum) or the luminance.

**−O**[0|1|2] (0=output in W/m^2/sr visible radiation (default), 1=output in W/m^2/sr solar radiation, 2=output in lm/m^2/sr luminance).

*Gendaylit* supports the following options.

**−s** The source description of the sun is not generated.

**−w** Suppress warning messages

**-g** *rfl* Average ground reflectance is *rfl*. This value is used to compute *skyfunc* when Dz is negative.

The following options do not apply when the solar altitude and azimuth are given explicitly.

**-a** *lat* The site latitude is *lat* degrees north. (Use negative angle for south latitude.) This is used in the calculation of sun angle.

**-o** *lon* The site longitude is *lon* degrees west. (Use negative angle for east longitude.) This is used in the calculation of solar time and sun angle. Be sure to give the corresponding standard meridian also! If solar time is given directly, then this option has no effect.

**-m** *mer* The site standard meridian is *mer* degrees west of Greenwich. (Use negative angle for east.) This is used in the calculation of solar time. Be sure to give the correct longitude also! If solar time is given directly, then this option has no effect.

**-i** *time_interval***[min]**
If gendaylit is used with weather files, the specified instantaneous points of time may be incorrect. This error occurs due to the fact that measurement results are frequently defined for time intervals, not for specific points of time. Although gendaylit is working correctly, this may lead to wrong outputs especially at low sun altitudes. The -i option allows to specify the time interval of the measurements in minutes, causing the solar position to be corrected for low sun altitudes. A warning message is returned if a correction has been performed.

## EXAMPLES

A clear non-turbid sky for a solar altitude of 60 degrees and an azimut of 0 degree might be defined by:

gendaylit -ang 60 0 -P 6.3 0.12 or gendaylit -ang 60 0 -W 840 135 This sky description corresponds to the clear sky standard of the CIE.

The corresponding sky with a high turbidity is:

gendaylit -ang 60 0 -P 3.2 0.24 or gendaylit -ang 60 0 -W 720 280

The dark overcast sky (corresponding to the CIE overcast standard, see CIE draft standard, Pub. No. CIE DS 003, 1st Edition, 1994) is obtained by:

gendaylit -ang 60 0 -P 1 0.08

A bright overcast sky is modelled with a larger value of delta, for example:

gendaylit -ang 60 0 -P 1 0.35

To generate the same bright overcast sky for March 2th at 3:15pm standard time at a site latitude of 42 degrees, 108 degrees west longitude, and a 110 degrees standard meridian:

gendaylit 3 2 15.25 -a 42 -o 108 -m 110 -P 1 0.35

## FILES

/usr/local/lib/ray/perezlum.cal

## AUTHOR

Jean-Jacques Delaunay, Jan Wienold, Wendelin Sprenger, Fraunhofer ISE (Freiburg i.B., Germany) (wendelin.sprenger@ise.fhg.de)

## ACKNOWLEDGEMENTS

The first work on this program was supported by the German Federal Ministry for Research and Technology (BMFT) under contract No. 0329294A, and a scholarship from the French Environment and Energy Agency (ADEME) which was co-funded by Bouygues. Many thanks to Peter Apian-Bennewitz, Arndt Berger, Christian Reetz, Ann Kovach, R. Perez, C. Gueymard and G. Ward for their help.

## SEE ALSO

gensky(1), rpict(1), rview(1), xform(1)

## NAME

gendaymtx - generate an annual Perez sky matrix from a weather tape

## SYNOPSIS

**gendaymtx** [ **−v** ][ **−d|−s** ][ **−r deg** ][ **−m N** ][ **−g r g b** ][ **−c r g b** ][ **-o{f|d}** ][ **-O{0|1}** ] [ **tape.wea** ]

## DESCRIPTION

*Gendaymtx* takes a weather tape as input and produces a matrix of sky patch values using the Perez all-weather model. The weather tape is assumed to be in the simple ASCII format understood by DAYSIM, which contains a short header with the site parameters followed by the month, day, standard time, direct normal and diffuse horizontal irradiance values, one time step per line. Each time step line is used to compute a column in the output matrix, where rows correspond to sky patch positions, starting with 0 for the ground and continuing to 145 for the zenith using the default *−m 1* parameter setting.

Increasing the *−m* parameter, typically by factors of two, yields a higher resolution sky using the Reinhart patch subdivision. For example, setting *−m 4* yields a sky with 2305 patches plus one patch for the ground. Each matrix entry is in fact three values, corresponding to red green and blue radiance channels (watts/sr/meter^2). Thus, an hourly weather tape for an entire year would yield 8760x3 (26280) values per output line (row).

The *−c* option may be used to specify a color for the sky. The gray value should equal 1 for proper energy balance The default sky color is *−c 0.960 1.004 1.118* . Similarly, the *−g* option may be used to specify a ground color. The default value is *−g 0.2 0.2 0.2* corresponding to a 20% gray.

The *−d* option may be used to produce a sun-only matrix, with no sky contributions. Alternatively, the *−s* option may be used to exclude any direct solar component from the output.

By default, *gendaymtx* assumes the positive Y-axis points north such that the first sky patch is in the Y-axis direction on the horizon, the second patch is just west of that, and so on spiraling around to the final patch near the zenith. The *−r* (or *−rz)* option rotates the sky the specified number of degrees counter-clockwise about the zenith, i.e., west of north. This is in keeping with the effect of passing the output of *gensky(1)* or *gendaylit(1)* through *xform(1)* using a similar transform.

The *−of* or *−od* option may be used to specify binary float or double output, respectively. This is much faster to write and to read, and is therefore preferred on systems that support it. (MS Windows is not one of them.) The *−O1* option specifies that output should be total solar radiance rather than visible radiance. Finally, the *−v* option will enable verbose reporting, which is mostly useful for finding out how many time steps are actually in the weather tape.

## EXAMPLES

Produce an uncolored Tregenza sky matrix without solar direct:

```
gendaymtx -m 1 -c 1 1 1 -s Detroit.wea > Detroit.mtx
```

Produce an hourly, annual Reinhart sky matrix with 2306 patches including solar contributions and send float output to *dctimestep(1)* to compute a sensor value matrix:

```
gendaymtx -m 4 -of VancouverBC.wea | dctimestep -if -n 8760 DCoef.mtx > res.dat
```

## AUTHORS

Ian Ashdown wrote most of the code, based on Jean-Jacques Delaunay's original gendaylit(1) implementation. Greg Ward wrote the final parameter parsing and weather tape conversion.

## SEE ALSO

dctimestep(1), genBSDF(1), gendaylit(1), gensky(1), genskyvec(1), rcollate(1), rcontrib(1), xform(1)

## NAME

genklemsamp - generate ray samples over surfaces using Klems BSDF basis

## SYNOPSIS

**genklemsamp** [ **−c N** ][ **−f{alfld}** ] [ **view opts** ] [ **geom.rad ..** ]

## DESCRIPTION

*Genklemsamp* generates ray origins and directions to sample planar surfaces needed to determine incident radiances or daylight coefficients. This command is typically used in conjunction with *rcontrib(1)* to analyze exterior daylight coefficients for an annual simulation.

The view options are needed to specify (at minimum) the view direction, which corresponds to the orientation of the surfaces, and the view up vector, which corresponds to the azimuth=90-degree position on the Klems sampling hemisphere. The view fore clipping distance is also quite useful, as it provides a means to pass through some thickness in a fenestration system before samples are sent out. Other view options such as the view type and aft clipping distance are overridden or ignored.

If no Radiance scene files are provided, then the specified parallel view defines the width, height, center, and orientation of the window or facade of interest. If one or more scene files are given, they are presumed to contain planar surfaces over which *genklemsamp* will originate samples. The sampled surface normals must correspond to the specified view direction, and unaligned surfaces are silently ignored.

The *−c* option specifies the number of rays to sample per Klems direction. These samples will be distributed evenly over the surface(s). The default setting is 1000.

The *−ff* option specifies that output rays should be generated as 32-bit IEEE binary float values. This may be more efficient if passed directly to *rcontrib* or *rtrace(1)* with the same option. Likewise, the *−fd* option specifies 64-bit IEEE binary double values. The default setting of *−fa* produces ASCII floating point values.

## EXAMPLE

To generate 500 samples per direction over a set of south-facing windows and pass to *rcontrib* to compute daylight coefficients in a matrix:

```
genklemsamp -c 500 -vd 0 -1 0 -vu 0 0 1 south_windows.rad | rcontrib -c 500 -e MF:1 -f reinhart.cal -b
rbin -bn Nrbins exterior.oct > Dsouth.dat
```

## AUTHOR

Greg Ward

## SEE ALSO

dctimestep(1), genBSDF(1), genskyvec(1), mkillum(1), rcontrib(1), rtrace(1), vwrays(1)

## NAME

genprism - generate a RADIANCE description of a prism

## SYNOPSIS

**genprism mat name** { **- | vfile | N v1 v2 .. vN** } [ **−l lvect** ][ **−r radius** ][ **−c** ][ **−e** ]

## DESCRIPTION

*Genprism* produces a RADIANCE scene description of a prism, or extruded polygon. The polygon to extrude lies in the z==0 plane, and is given as a list of (x,y) pairs on the standard input (-), or from the file *vfile,* or on the command line preceded by the number of vertices, *N*. The order of the vertices and the extrusion vector *lvect* (default (0,0,1)) determine the surface orientations. The surfaces that make up the prism will be modified by *mat* and their identifiers will begin with *name*. The *−r* option may be used to round the corners of the object using spheres and cylinders. The *−c* option inhibits generation of a face connecting the last vertex to the first. The *−e* option inhibits generation of the end polygons.

## EXAMPLE

To produce a equilateral triangular prism:

```
genprism clear_plastic prism 3 0 0 .5 .866 1 0
```

## AUTHOR

Greg Ward

## BUGS

The rounding option only works for opaque prisms with outward facing normals. If the normals face inward, the appearance will be bizarre.

## SEE ALSO

genbox(1), genrev(1), gensurf(1), genworm(1), rpict(1), rvu(1), xform(1)

## NAME

genrev - generate a RADIANCE description of surface of revolution

## SYNOPSIS

**genrev mat name 'z(t)' 'r(t)' nseg** [ **−e expr** ][ **−f file** ][ **−s** ]

## DESCRIPTION

*Genrev* produces a RADIANCE scene description of a surface of revolution. The object will be composed of *nseg* cones, cups, cylinders, tubes or rings following the parametric curve defined by *z(t)* (height) and *r(t)* (radius). When *z* is increasing with t, the surface normal points outward. When *z* is decreasing, the normal points inward. The variable *t* used in the function expressions varies from 0 to 1 in even steps of *1/nseg*. The expressions are of the same type used in RADIANCE function files. Auxiliary expressions and/or files may be specified in any number of *−e* and *−f* options. The *−s* option smooths the surfaces using Phong normal interpolation.

## EXAMPLE

To generate a torus with an inner radius of 1 and an outer radius of 3:

```
genrev steel torus 'sin(2*PI*t)' '2+cos(2*PI*t)' 32
```

## AUTHOR

Greg Ward

## BUGS

The −s option doesn't modify the surface normal correctly for the opposite side.

## SEE ALSO

genbox(1), gensurf(1), genworm(1), icalc(1), rpict(1), rvu(1), xform(1)

## NAME

gensky - generate a RADIANCE description of the sky

## SYNOPSIS

**gensky month day time** [ **options** ]
**gensky -ang altitude azimuth** [ **options** ]
**gensky -defaults**

## DESCRIPTION

*Gensky* produces a RADIANCE scene description for the CIE standard sky distribution at the given month, day and time. By default, the time is interpreted as local standard time on a 24-hour clock. The time value may be given either as decimal hours, or using a colon to separate hours and minutes. If the time is immediately followed (no white space) by a North American or European time zone designation, then this determines the standard meridian, which may be specified alternatively with the *−m* option. The following time zones are understood, with their corresponding hour differences from Greenwich Mean Time:

Standard time:

| YST | PST | MST | CST | EST | GMT |
|---|---|---|---|---|---|
| 9 | 8 | 7 | 6 | 5 | 0 |

| CET | EET | AST | GST | IST | JST | NZST |
|---|---|---|---|---|---|---|
| −1 | −2 | −3 | −4 | −5.5 | −9 | −12 |

Daylight savings time:

| YDT | PDT | MDT | CDT | EDT | BST |
|---|---|---|---|---|---|
| 8 | 7 | 6 | 5 | 4 | −1 |

| CEST | EEST | ADT | GDT | IDT | JDT | NZDT |
|---|---|---|---|---|---|---|
| −2 | −3 | −4 | −5 | −6.5 | −10 | −13 |

If the time is preceded by a plus sign ('+'), then it is interpreted as local solar time instead. It is very important to specify the correct latitude and longitude (unless local solar time is given) using the *−a* and *−o* options to get the correct solar angles.

The second form gives the solar angles explicitly. The altitude is measured in degrees above the horizon, and the azimuth is measured in degrees west of South.

The third form prints the default option values.

The output sky distribution is given as a brightness function, *skyfunc*. Its value is in watts/steradian/meter2. The x axis points east, the y axis points north, and the z axis corresponds to the zenith. The actual material and surface(s) used for the sky is left up to the user. For a hemispherical blue sky, the description might be:

```
!gensky 4 1 14

skyfunc glow skyglow
0
0
4 .99 .99 1.1 0

skyglow source sky
0
0
4 0 0 1 180
```

Often, *skyfunc* will actually be used to characterize the light coming in from a window.

In addition to the specification of a sky distribution function, *gensky* suggests an ambient value in a comment at the beginning of the description to use with the *−av* option of the RADIANCE rendering programs.

(See rvu(1) and rpict(1).) This value is the cosine-weighted radiance of the sky in watts/steradian/meter2.

*Gensky* supports the following options.

**−s** Sunny sky without sun. The sky distribution will correspond to a standard CIE clear day.

**+s** Sunny sky with sun. In addition to the sky distribution function, a source description of the sun is generated.

**−c** Cloudy sky. The sky distribution will correspond to a standard CIE overcast day.

**−i** Intermediate sky without sun. The sky will correspond to a standard CIE intermediate day.

**+i** Intermediate sky with sun. In addition to the sky distribution, a (somewhat subdued) sun is generated.

**−u** Uniform cloudy sky. The sky distribution will be completely uniform.

**-g** *rfl* Average ground reflectance is *rfl*. This value is used to compute *skyfunc* when Dz is negative. Ground plane brightness is the same for *−s* as for *+s*. (Likewise for *−i* and *+i*, but see the *−r* option below.)

**-b** *brt* The zenith brightness is *brt*. Zenith radiance (in watts/steradian/meter2) is normally computed from the sun angle and sky turbidity (for sunny sky). It can be given directly instead, using this option.

**-B** *irrad* Same as *−b*, except zenith brightness is computed from the horizontal diffuse irradiance (in watts/meter2).

**-r** *rad* The solar radiance is *rad*. Solar radiance (in watts/steradian/meter2) is normally computed from the solar altitude. This option may be used to override the default calculation. If a value of zero is given, no sun description is produced, and the contribution of direct solar to ground brightness is neglected.

**-R** *irrad* Same as *−r*, except solar radiance is computed from the horizontal direct irradiance (in watts/meter2).

**-t** *trb* The turbidity factor is *trb*. Greater turbidity factors correspond to greater atmospheric scattering. A turbidity factor of 1.0 indicates an ideal clear atmosphere (i.e. a completely dark sky). Values less than 1.0 are physically impossible.

The following options do not apply when the solar altitude and azimuth are given explicitly.

**-a** *lat* The site latitude is *lat* degrees north. (Use negative angle for south latitude.) This is used in the calculation of sun angle.

**-o** *lon* The site longitude is *lon* degrees west. (Use negative angle for east longitude.) This is used in the calculation of solar time and sun angle. Be sure to give the corresponding standard meridian also! If solar time is given directly, then this option has no effect.

**-m** *mer* The site standard meridian is *mer* degrees west of Greenwich. (Use negative angle for east.) This is used in the calculation of solar time. Be sure to give the correct longitude also! If a time zone or solar time is given directly, then this option has no effect.

## EXAMPLE

To produce a sunny sky for July 4th at 2:30pm Eastern daylight time at a site latitude of 42 degrees, 89 degrees west longitude:

```
gensky 7 4 14:30EDT +s −a 42 −o 89
```

To produce a sunny sky distribution for a specific sun position but without the sun description:

```
gensky −ang 23 −40 −s
```

## FILES

/usr/local/lib/ray/skybright.cal

## AUTHOR

Greg Ward

## SEE ALSO

rpict(1), rvu(1), xform(1)

## NAME

genskyvec - compute patch radiance averages for a specific sky

## SYNOPSIS

**genskyvec** [ **−m N** ][ **−c r g b** ][ **−d** ]

## DESCRIPTION

*Genskyvec* samples the *Radinace* sky description given on the standard input to generate a list of average patch radiances. If there is a sun in the description, *genskyvec* will include its contribution in the three nearest sky patches, distributing energy according to centroid proximity.

By default, *genskyvec* divides the sky into 2305 patches, plus one patch for the ground. This corresponds to Reinhart's extension of the Tregenza sky, where the original 145 patches are subdivided into 16 subpatches, except at the zenith. A different subdivision may be specified via the *−m* option. The value given will be used to subdivide each dimension, so the default of 4 yields almost 16 times as many patches as the original Tregenza sky, which can be specified with *−m 1*. A higher resolution sky is generally better for daylight coefficient analysis where solar position is important.

The *−c* option may be used to specify a color for the sky. The gray value should equal 1 for proper energy balance The default sky color is *−c 0.960 1.004 1.118* .

The *−d* option may be used to produce a sun-only vector, with no sky contributions. If a time of day is selected or sky type that has no sun, then a zero vector is produced.

## EXAMPLE

To generate 578 patches corresponding to a 2x2 subdivision of the Tregenza sky on a sunny equinox noon:

```
gensky 9 21 12 | genskyvec -m 2 > sky09_21_12.dat
```

## AUTHOR

Greg Ward

## SEE ALSO

dctimestep(1), genBSDF(1), gendaylit(1), gendaymtx(1), gensky(1), rcontrib(1), rtrace(1)

## NAME

gensurf - generate a RADIANCE or Wavefront description of a curved surface

## SYNOPSIS

**gensurf mat name 'x(s,t)' 'y(s,t)' 'z(s,t)' m n** [ **−e expr** ][ **−f file** ][ **−s** ][ **−o** ]
**gensurf mat name 'x(s,t)' 'y(s,t)' dfile m n** [ **−e expr** ][ **−f file** ][ **−s** ][ **−o** ]
**gensurf mat name dfile dfile dfile m n** [ **−s** ][ **−o** ]

## DESCRIPTION

*Gensurf* produces either a RADIANCE scene description or a Wavefront .OBJ file of a functional surface defined by the parametric equations *x(s,t), y(s,t),* and *z(s,t)*. The surface normal is defined by the right hand rule as applied to *(s,t)*. *S* will vary from 0 to 1 in steps of *1/m,* and *t* will vary from 0 to 1 in steps of *1/n*. The surface will be composed of *2*m*n* or fewer triangles and quadrilaterals. The expressions are of the same type used in RADIANCE function files. Auxiliary expressions and/or files may be specified in any number of *−e* and *−f* options. The *−s* option adds smoothing (surface normal interpolation) to the surface. The *−o* option produces a Wavefront .OBJ file rather than a RADIANCE scene description. This is most useful as input to the *obj2mesh(1)* program for producing a compiled mesh. A single "usemtl" statement will appear at the beginning of the .OBJ output, echoing the modifier given on the command line.

Rough holes may be cut in the mesh by defining a valid(s,t) function. Where this function is positive, polygon vertices will be produced. Where it is negative, no geometry will be output. Surface normal interpolation will ignore any invalid vertices.

The second invocation form reads z data values from the file *dfile*. This file must give either m*n or (m+1)*(n+1) floating point z values. If m*n values are given, then the values correspond to the centroid of each quadrilateral region. If (m+1)*(n+1) values are given, then the values correspond to the vertices of each quadrilateral region. The ordering of the data in the file is such that the s values are changing faster than the t values. If a minus ('-') is given for *dfile,* then the values are read from the standard input.

The third invocation form is used to read coordinate triplets from a file or the standard input. The three *dfile* arguments must all be the same, and the corresponding file must contain three floating point values for each point location. The ordering and other details are the same as those described for z value files above.

## EXAMPLE

To generate a tesselated sphere:

```
gensurf crystal ball 'sin(PI*s)*cos(2*PI*t)' 'cos(PI*s)' 'sin(PI*s)*sin(2*PI*t)' 7 10
```

To generate a 10x20 smoothed height field from 12 recorded vertex z values:

```
gensurf dirt ground '10*s' '20*t' height.dat 2 3 −s
```

## AUTHOR

Greg Ward

## BUGS

The smoothing operation requires that functions be defined beyond the [0,1] boundaries of s and t.

## SEE ALSO

genbox(1), genrev(1), genworm(1), icalc(1), obj2mesh(1), obj2rad(1), rpict(1), rvu(1), xform(1)

## NAME

genworm - generate a RADIANCE description of a functional worm

## SYNOPSIS

**genworm mat name 'x(t)' 'y(t)' 'z(t)' 'r(t)' nseg** [ **−e expr** ][ **−f file** ]

## DESCRIPTION

*Genworm* produces a RADIANCE scene description of a worm defined by the parametric equations *x(t), y(t), z(t),* and *r(t)* (the radius). *T* will vary from 0 to 1 in steps of *1/nseg*. The surface will be composed of *nseg* cones or cylinders and *nseg+1* spheres. The expressions are of the same type used in RADIANCE function files. Auxiliary expressions and/or files may be specified in any number of *−e* and *−f* options.

## EXAMPLE

To generate a banana:

```
genworm yellow banana '0' '5*sin(t)' '5*cos(t)' '.4-(.5-t)*(.5-t)' 20
```

## AUTHOR

Greg Ward

## BUGS

Since the worm is constructed of intersecting surfaces, only opaque materials should be used with this object. Also, a worm cannot double back inside itself without making a mess.

## SEE ALSO

genbox(1), genrev(1), gensurf(1), icalc(1), rpict(1), rvu(1), xform(1)

## NAME

getbbox - compute bounding box for RADIANCE scene

## SYNOPSIS

**getbbox** [ **−w** ][ **−h** ] [ **input ..** ]

## DESCRIPTION

*Getbbox* reads each scene description *input* and computes the minimum axis-aligned parallelopiped that will enclose all of the objects. Each *input* can be either a file name, or a command (enclosed in quotes and preceded by a '!'). If no arguments are given, the standard input is read. A hyphen ('-') can also be used to indicate the standard input.

The *−w* option suppresses warnings. The *−h* option suppresses the header line "xmin xmax ymin ymax zmin zmax".

## EXAMPLE

To compute the bounding box for the object ''thingy'':

```
getbbox thingy
```

To preview ''scene'':

```
preview −v FOUR −b 'getbbox −h scene' scene
```

## NOTES

Since expanding a scene can require considerable overhead, it is better to use the bounding cube produced by *oconv(1)* and read by *getinfo(1)* if an octree exists for the scene. However, there are certain circumstances, such as foreign object placement, that require knowing the bounding box rather than just the bounding cube.

## AUTHOR

Greg Ward

## ACKNOWLEDGEMENT

Work on this program was sponsored by the LESO group at EPFL in Switzerland.

## SEE ALSO

getinfo(1), oconv(1), xform(1)

## NAME

getinfo - get header information from a RADIANCE file

## SYNOPSIS

**getinfo** [ **-d** ][ **file ..** ]
**getinfo** -

## DESCRIPTION

*Getinfo* reads the header of each RADIANCE *file* and writes it to the standard output. Octree and picture files are in a binary format, which makes it difficult to determine their content. Therefore, a few lines of text are placed at the beginning of each file by the RADIANCE program that creates it. The end of the header information and the start of the data is indicated by an empty line. The *−d* option can be used to print the dimensions of an octree or picture file instead. For an octree, *getinfo −d* prints the bounding cube (xmin ymin zmin size). For a picture, *getinfo −d* prints the y and x resolution (−Y yres +X xres). If no *file* is given, the standard input is read.

The second form of getinfo with a hyphen simply removes the header and copies the body of the file from the standard input to the standard output.

## EXAMPLE

To print the header information from scene1.oct and scene2.hdr:

```
getinfo scene1.oct scene2.hdr
```

## AUTHOR

Greg Ward

## SEE ALSO

oconv(1), pfilt(1), rhinfo(1), rpict(1), rvu(1)

## NAME

glare - perform glare and visual comfort calculations

## SYNOPSIS

**glare** [ **glarefile** [ **picture** [ **octree** ] ] ]

## DESCRIPTION

*Glare* is an interactive script for executing programs used to locate glare sources and compute glare indices and visual comfort probability. If no *glarefile* is given, the program prompts the user for a one. If the file does not exist, *glare* asks the user some questions about the scene in question then runs *findglare(1)* to compute values to store in the file. *Glare* then presents the user a menu of available glare index calculations. After choosing a calculation, *glare* offers to store the result (usually not useful) or plot the information (but only for multiple glare angles).

If you are creating a new *glarefile,* it usually works best to start with a displayed image for reference during the interrogation.

## AUTHOR

Greg Ward

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland.

## SEE ALSO

dayfact(1), findglare(1), glarendx(1), igraph(1), rpict(1), xglaresrc(1), ximage(1)

## NAME

glarendx - calculate glare index

## SYNOPSIS

**glarendx –t type** [ **–h** ] [ **glarefile** ]

## DESCRIPTION

*Glarendx* computes the selected glare index *type* from the given *glarefile* produced by *findglare(1)*. *Glarendx* computes one value for each (indirect illuminance) angle in the input file. If no *glarefile* is given, *glarendx* reads from the standard input. The *–h* option can be used to remove the information header from the output.

*Glarendx* understands the following arguments for *type:*

| | |
|---|---|
| guth_vcp | Guth Visual Comfort Probability |
| cie_cgi | CIE Glare Index (Einhorn) |
| ugr | Unified Glare Rating |
| brs_gi | BRS Glare Index |
| dgi | Daylight Glare Index |
| guth_dgr | Guth Disability Glare Rating |
| vert_dir | Direct Vertical Illuminance |
| vert_ill | Total Vertical Illuminance |
| vert_ind | Indirect Vertical Illuminance |

Alternatively, a symbolic or hard link to the program with any of the above names may be used in place of the *type* argument.

## AUTHORS

Greg Ward, Raphael Compagnon

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland.

## SEE ALSO

findglare(1), glare(1), igraph(1), rpict(1), xglaresrc(1), ximage(1)

## NAME

glrad - render a RADIANCE scene using OpenGL

## SYNOPSIS

**glrad** [ **−w** ][ **−b** ][ **−s** ][ **−S** ][ **−v view** ] **rfile** [ **VAR=value ..** ]

## DESCRIPTION

*Glrad* renders a Radiance scene description in OpenGL. Its syntax and behavior is similar to *rad(1)* with the *−o* option, where the output device is assumed to be an X11 server with GLX extensions.

The *−w* option turns off warnings. The *−s* option tells *glrad* to run *rad* silently, not echoing *oconv(1)* command. The *−b* option turns off back face visibility (i.e., enables back face culling). This is equivalent to the *−bv* option of *rpict(1)* and *rvu(1)*. The *−S* option turns on full-screen stereo for displays that support it. (Be sure to run */usr/gfx/setmon(1)* or its equivalent to set STR_TOP or STR_BOT, first.) The *−v* option may be used to specify a starting view, either by symbolic name as entered in the *view* assignments in *rfile*, or by a complete view specification, enclosed in quotes. If no view is specified, then the first standard view from *rfile* is used to start.

Variables permitted in *rfile* are described in the *rad* manual page. Additional or overriding assignments may be given on the command line following *rfile*.

The view is controlled via the mouse and simple one-character commands, listed below:

**(mouse)** Modify the current view. The mouse is used to control the current view in the following ways:

| CONTROL | | MOUSE ACTION |
|---|---|---|
| (none) | left | Move forward towards cursor position |
| (none) | right | Move backward away from cursor position |
| (none) | middle | Rotate in place (usually safe) |
| shift | left | Orbit left around cursor position |
| shift | right | Orbit right around cursor position |
| shift | middle | Orbit skyward |
| cntl | middle | Orbit earthward |

For all movements but rotating in place, the cursor must be placed over some bit of visible geometry, otherwise the program has no reference point from which to work. It is best to just experiment with these controls until you learn to fly safely in your model. And if you run into trouble, the 'l' command is very useful. (See below.)

**'+'** Zoom in on the current cursor position. (Beware of repeating keys that go faster than the display updates.)

**'-'** Zoom out from the current cursor position.

**'l'** Return to the last saved view. Each time a new command changes the current view, the last view is saved, and may be recalled with this command. Multiple uses of the same command (e.g., rotation, zoom) will save only the view before the first such command. This way, it is easy to get back to where you were before a sequence of view changes.

**'h'** Fix the head height. All mouse-controlled view motions will be adjusted so that the head height does not change (where vertical is determined by the current view up vector).

**'H'** Release the head height, allowing it to change again during mouse-controlled movements.

**'v'** Print the current view parameters to the standard output. This is useful for finding out where you are, or for saving specific views in a keyframe file for animations or returning to later.

**'V'** Append the current view to the original *rfile*. This view will be unnamed, but can be referred to by number or the user may add a name later with a text editor. The current view number becomes the last standard view. (See the 'n' and 'p' commands, below.)

**'n'** Go to the next standard view stored in *rfile*. If the last view is currently displayed, then cycle to the first one.

**'p'** Go to the previous standard view stored in *rfile*. If the first view is currently displayed, then cycle to the last one.

**'q'** Quit *glrad*. This is the normal way to exit the program.

## AUTHOR

Greg Ward Larson

## BUGS

It would be nice if *glrad* set the appropriate video format for stereo viewing automatically, but the process is different on different systems and there is no single, sure-fire way to do it for all systems. On systems that do not support stereo extensions, the program may be compiled with the –DNOSTEREO option, which will avoid undefined symbol errors.

## SEE ALSO

chmod(1), getinfo(1), ls(1), objview(1), oconv(1), ps(1), rad(1), ranimate(1), rhcopy(1), rholo(1), rpict(1), rtrace(1), rvu(1), setmon(1)

## NAME

histo - compute 1-dimensional histogram of N data columns

## SYNOPSIS

**histo [-c][-p] xmin xmax nbins**
**histo [-c][-p] imin imax**

## DESCRIPTION

*Histo* bins columnular data on the standard input between the given minimum and maximum values. If three command line arguments are given, the third is taken as the number of data bins between the first two real numbers. If only two arguments are given, they are both assumed to be integers, and the number of data bins will be equal to their difference plus one. The bins are always of equal size.

The output is N+1 columns of data (for N columns input), where the first column is the centroid of each division, and each row corresponds to the frequencies for each column around that value.

If the *−c* option is present, then *histo* computes the cumulative histogram for each column instead of the straight frequencies. The upper value of each bin is printed also instead of the centroid. This may be useful in computing percentiles, for example. Values below the minimum specified are still counted in the cumulative total.

The *−p* option tells *histo* to report the percentage of the total number of input lines rather than the absolute counts. In the case of a cumulative total, this yields the percentile values directly. Values above the maximum are counted as well as values below in this case.

All input data is interpreted as real values, and columns must be white-space separated. If any value is less than the minimum or greater than the maximum, it will be ignored unless the *−c* option is specified.

## EXAMPLE

To count data values between −1 and 1 in 50 bins:

```
histo −1 1 50 < input.dat
```

To count frequencies of integers between 0 and 255:

```
histo 0 255 < input.dat
```

## AUTHOR

Greg Ward

## SEE ALSO

cnt(1), neaten(1), rcalc(1), rcollate(1), rlam(1), tabfunc(1), total(1)

## NAME

icalc - interactive calculator

## SYNOPSIS

**icalc** [ **file** ]

## DESCRIPTION

*Icalc* is a algebraic calculator designed primarily for interactive use. Each formula definition *file* is read and compiled from the RADIANCE library where it is found. The standard input is then read, expressions are evaluated and results are sent to the standard output. If a newline is escaped using a backslash, input is continued on the next line.

An expression contains real numbers, variable names, function calls, and the following operators:

```
+ - * / ^
```

Operators are evaluated left to right, except '^', which is right associative. Exponentiation has the highest precedence; multiplication and division are evaluated before addition and subtraction. Expressions can be grouped with parentheses. Each result is assigned a number, which can be used in future expressions. For example, the expression ($3*10) is the result of the third calculation multiplied by ten. A dollar sign by itself may be used for the previous result. All values are double precision real.

In addition, variables and functions can be defined by the user. A variable definition has the form:

```
var = expression ;
```

Any instance of the variable in an expression will be replaced with its definition. A function definition has the form:

```
func(a1, a2, ..) = expression ;
```

The expression can contain instances of the function arguments as well as other variables and functions. Function names can be passed as arguments. Recursive functions can be defined using calls to the defined function or other functions calling the defined function.

To define a constant expression, simply replace the equals sign ('=') with a colon (':') in a definition. Constant expressions are evaluated only once, the first time they are used. This avoids repeated evaluation of expressions whose values never change. Ideally, a constant expression contains only numbers and references to previously defined constant expressions and functions. Constant function definitions are are replaced by their value in any expression that uses them with constant arguments. All predefined functions and variables have the constant attribute. Thus, "sin(PI/4)" in an expression would be immediately replaced by ".707108" unless sin() or PI were redefined by the user. (Note that redefining constant expressions is not a recommended practice!)

A variable or function's definition can be displayed with the '?' command:

```
? name
```

If no name is given, all definitions are printed. The '>' command writes definitions to a file:

```
> file
```

Similarly, the '<' command loads definitions.

The following library of predefined functions and variables is provided:

**PI** the ratio of a circle's circumference to its diameter.

**if(cond, then, else)**
if cond is greater than zero, then is evaluated, otherwise else is evaluated. This function is necessary for recursive definitions.

**select(N, a1, a2, ..)**
return aN (N is rounded to the nearest integer). This function provides array capabilities. If *N* is zero, the number of available arguments is returned.

**rand(x)** compute a random number between 0 and 1 based on x.

**floor(x)** return largest integer not greater than x.

**ceil(x)** return smallest integer not less than x.

**sqrt(x)** return square root of x.

**exp(x)** compute e to the power of x (e approx = 2.718281828).

**log(x)** compute the logarithm of x to the base e.

**log10(x)** compute the logarithm of x to the base 10.

**sin(x), cos(x), tan(x)**
trigonometric functions.

**asin(x), acos(x), atan(x)**
inverse trigonometric functions.

**atan2(y, x)** inverse tangent of y/x (range –pi to pi).

## ENVIRONMENT

RAYPATH the directories to check for auxiliary files.

## AUTHOR

Greg Ward

## SEE ALSO

ev(1), rcalc(1), tabfunc(1)

## NAME

ies2rad - convert IES luminaire data to RADIANCE description

## SYNOPSIS

**ies2rad** [ **options** ] [ **input ..** ]

## DESCRIPTION

*Ies2rad* converts one or more IES luminaire data files to the equivalent RADIANCE scene description. The light source geometry will always be centered at the origin aimed in the negative Z direction, with the 0 degree plane along the X axis. (Note, this means that the IES "width" is actually along the Y axis, while "length" corresponds to the X axis.) Usually, two output files will be created for every input file, one scene file (with a ".rad" suffix) and one data file (with a ".dat" suffix). If the IES input file includes tilt data, then another data file will be created (with a "+.dat" suffix). If the *−s* option is used, the scene data will be sent to the standard output instead of being written to a file. Since the data file does not change with other options to *ies2rad,* this is a convenient way to specify different lamp colors and multipliers inline in a scene description. If the *−g* option is used, then an octree file will be created (with the ".oct" suffix). The root portion of the output file names will be the same as the corresponding input file, unless the *−o* option is used. The output files will be created in the current directory (no matter which directory the input files came from) unless the *−l* or *−p* options are used.

*Ies2rad* assigns light source colors based on information in a lamp lookup table. Since most lamps are distinctly colored, it is often desirable to override this lookup procedure and use a neutral value that will produced color-balanced renderings. In general, it is important to consider lamp color when an odd assortment of fixture types is being used to illuminate the same scene, and the rendering can always be balanced by pfilt(1) to a specific white value later.

**-l** *libdir* Set the library directory path to *libdir.* This is where all relative pathnames will begin for output file names. For light sources that will be used by many people, this should be set to some central location included in the RAYPATH environment variable. The default is the current working directory.

**-p** *prefdir* Set the library subdirectory path to *prefdir.* This is the subdirectory from the library where all output files will be placed. It is often most convenient to use a subdirectory for the storage of light sources, since there tend to be many files and placing them all in one directory is very messy. The default value is the empty string.

**-o** *outname*
Set the output file name root to *outname*. This overrides the default output file name root which is the same as the input file. This option may be used for only one input file, and is required when reading data from the standard input.

**-s** Send the scene information to the standard output rather than a separate file. This is appropriate when calling *ies2rad* from within a scene description via an inline command. The data file(s) will still be written based on the output file name root, but since this information is unaffected by command line options, it is safe to have multiple invocations of *ies2rad* using the same input file and different output options. The *−s* option may be used for only one input file.

**-d***units* Output dimensions are in *units,* which is one of the letters 'm', 'c', 'f', or 'i' for meters, centimeters, feet or inches, respectively. The letter specification may be followed by a slash ('/') and an optional divisor. For example, *−dm/1000* would be millimeters. The default output is in meters, regardless of the original units in the IES input file. Note that there is no space in this option.

**-i** *rad* Ignore the crude geometry given by the IES input file and use instead an illum sphere with radius *rad*. This option may be useful when the user wishes to add a more accurate geometric description to the light source model, though this need is obviated by the recent LM-63-1995 specification, which uses MGF detail geometry. (See *−g* option below.)

**-g** If the IES file contains MGF detail geometry, compile this geometry into a separate octree and create a single instance referencing it instead of including the converted geometry directly in

the Radiance output file. This can result in a considerable memory savings for luminaires which are later duplicated many times in a scene, though the appearance may suffer for certain luminaires since the enclosed glow sources will not light the local geometry as they would otherwise.

**-f** *lampdat* Use *lampdat* instead of the default lamp lookup table (lamp.tab) to map lamp names to xy chromaticity and lumen depreciation data. It is often helpful to have customized lookup tables for specific manufacturers and applications.

**-t** *lamp* Use the given lamp type for all input files. Normally, *ies2rad* looks at the header lines of the IES file to try and determine what lamp is being used in the fixture. If any of the lines is matched by a pattern in the lamp lookup table (see the –f option above), that color and depreciation factor will be used instead of the default (see the –c and –u options). The *lamp* specification is also looked up in the lamp table unless it is set to "default", in which case the default color is used instead.

**–c** *red grn blu*
Use the given color if the type of the lamp is unknown or the –t option is set to "default". If unspecified, the default color will be white.

**–u** *lamp* Set the default lamp color according to the entry for *lamp* in the lookup table (see the –f option). This is the color that will be used if the input specification does not match any lamp type patterns. This option is used instead of the –c option.

**–m** *factor* Multiply all output quantities by *factor*. This is the best way to scale fixture brightness for different lamps, but care should be taken when this option is applied to multiple files.

## EXAMPLE

To convert a single IES data file in inches with color balanced output and 15% lumen depreciation, creating the files "fluor01.rad" and "fluor01.dat" in the current directory:

```
ies2rad –di –t default –m .85 fluor01.ies
```

To convert three IES files of various types to tenths of a foot and put them in the library "/usr/local/lib/ray" subdirectory "source/ies":

```
ies2rad –df/10 –l /usr/local/lib/ray –p source/ies ies01 ies02 ies03
```

To convert a single file and give the output a different name:

```
ies2rad –o fluorescent ies03
```

## ENVIRONMENT

RAYPATH directories to search for lamp lookup table

## AUTHOR

Greg Ward

## BUGS

In pre–1991 standard IES files, all header lines will be examined for a lamp table string match. In post–1991 standard files, only those lamps with the [LAMP] or [LAMPCAT] keywords will be searched. The first match found in the file is always the one used. This method of assigning colors to fixtures is less than perfect, and the IES would do well to include explicit spectral information somehow in their specification.

The IESNA LM–63 specification prior to 1995 provided three basic source shapes, rectangular, round, and elliptical. The details of these shapes is vague at best. Rectangular sources will always be rectangular, but ies2rad will approximate round sources as spherical if the height is close to or greater than the width and length, and as a ring otherwise. Elliptical sources are treated the same as round sources. The 1995 standard rectifies this problem by including detailed luminaire geometry as MGF data, though nothing in the standard requires manufacturers to provide this information.

## SEE ALSO

mgf2rad(1), oconv(1), pfilt(1), rad2mgf(1), rpict(1), xform(1)

## NAME

igraph - interactive graphing program

## DESCRIPTION

*Igraph* is a crude interactive program for creating line and scatter plots. Provisions for reading and writing graph files as well as changing variables and getting output are provided from a menu.

Graph files and variables are as described in bgraph(1).

## AUTHOR

Greg Ward

## BUGS

There is no mechanism provided for undefining a variable.

## SEE ALSO

bgraph(1), icalc(1), metafile(5)

## NAME

lampcolor - compute spectral radiance for diffuse emitter

## SYNOPSIS

**lampcolor** [ lamptable ]

## DESCRIPTION

*Lampcolor* is an interactive program for computing the appropriate color for a diffuse emitter. From the total lumen output and a simple description of the fixture geometry, the radiance for a diffuse fixture is calculated. Since most light fixtures are not really diffuse, the preferred method is to use luminaire distribution data (see *ies2rad(1))*.

A table of lamp colors and depreciation factors is used to get the color for a specific lamp type. If "white" is specified, the lamp will be uncolored. This is probably preferable for scenes with only a single variety of lamp in order to produce color-balanced images.

## ENVIRONMENT

| | |
|---|---|
| RAYPATH | Directories to check for lamp table |

## FILES

| | |
|---|---|
| lamp.tab | Default lamp lookup table |

## AUTHOR

Greg Ward

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland.

## SEE ALSO

ies2rad(1)

## NAME

lookamb - examine ambient file values

## SYNOPSIS

**lookamb** [ **−d** ][ **−r** ][ **−h** ] [ **ambfile** ]

## DESCRIPTION

*Lookamb* examines an ambient file produced by *rpict(1), rtrace(1),* or *rvu(1)*. The default mode prints the position, direction, level, weight, radius, value, position gradient and direction gradient for each stored ambient value preceded by a label appropriate for human interpretation. The *−d* option prints the same quantities in 18 columns without any expanatory text, and is more appropriate as input to another program.

The *−r* option performs the reverse conversion, taking the output of lookamb and reproducing the original ambient file on the standard output.

The *−h* option causes lookamb not to print out the header information on the output, or not to expect it on the input with the *−r* option.

If no file is given, *lookamb* reads from the standard input.

## NOTES

Before release 2.2 of Radiance, ambient files were in a machine-dependent format. Since that is no longer the case, ambient files can now be moved freely between machines without any conversions. Thus, the only reason to use *lookamb* now is to examine the contents of an ambient file.

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), oconv(1), pfilt(1), pinterp(1), rpict(1), rtrace(1), rvu(1)

## NAME

macbethcal - compute color compensation based on measured Macbeth chart

## SYNOPSIS

**macbethcal** [ **−d debug.hdr** ][ **−p xul yul xur yur xll yll xlr ylr** ] **scannedin.hdr** [ **calibout.cal** ]
**macbethcal −c** [ **−d debug.hdr** ] [ **measured.xyY** [ **calibout.cal** ] ]

## DESCRIPTION

*Macbethcal* takes a scanned image or measurement set of a Macbeth ColorChecker™ color rendition chart and computes a color mapping function suitable as input to *pcomb(1)*.

In the first form, *macbethcal* takes a scanned image of a Macbeth chart that has been converted into a Radiance picture using a fixed procedure. When used properly as input to *pcomb,* the computed calibration file will adjust the brightness and color of any similarly scanned and converted image so as to best match the original. If the lighting conditions are carefully controlled (as in the case of a flatbed scanner), it is even possible to get reliable reflectance values this way, at least within 10% or so. The input picture is named on the command line. The output calibration file will be written to the standard output if no file name is given on the command line.

In the second form, the input is from a file containing measured values for each Macbeth color. This file must contain entries of the form:

N x y Y

Where *N* is the number of the corresponding Macbeth color. (See back of ColorChecker chart for color names and indexing, but it basically starts from the upper left with 1 and proceeds in English text order to the lower right, which is 24.) The values *x, y* and *Y* are the 1931 CIE (x,y) chromaticity coordinates followed by the luminance for that color, which can be in any units. If a white value is known (i.e. maximum output level), then it may be given as entry number 0. The entries may be in any order, and comments may be included delimited by a pound sign ('#') and continuing to the end of line. It is recommended that measurements be done for all 24 colors, but the only required entries are the 6 neutral values on the bottom row of the chart.

Computing a mapping from measured colors is usually more convenient when calibrating a particular output device. This is accomplished by printing the picture *macbeth_spec.hdr* (which may be found in the standard RADIANCE library directory in the lib subdirectory) and measuring the output with a chroma meter or spectrophotometer.

For a scanned image, the locations of the 24 Macbeth patches in the input picture must be known. If the chart borders are not at the edges of the input picture, or the chart has been reversed or rotated or is uncentered or at an oblique angle, then it is necessary to specify the pixel locations of the corners of the chart with the *−p* option. The corner postions (x,y pixel addresses as given by the *ximage(1)* "p" command) are ordered on the command line: upper-left, upper-right, lower-left, lower-right (i.e. English text ordering). These coordinates should be the outside corner positions of the following patches:

| | |
|---|---|
| upper-left | = 1. dark skin |
| upper-right | = 6. bluish green |
| lower-left | = 19. white |
| lower-right | = 24. black |

If the chart has been flipped or rotated, simply give the pixel positions of the appropriate patch corners, wherever they are in the image. (Note: if the Radiance picture has been flipped or rotated with *pflip(1)* or *protate(1), ximage* will report the original pixel positions if the *−c* option was not used by the reorienting program(s). This will be wrong, so be sure to use the *−c* option.) *Macbethcal* can handle a chart with any orientation or perspective warping if the corner coordinates are given correctly. The debug picture output is the best way to check for consistency. (See the *−d* option, below.)

The *−d* option may be used to specify an additional output file, which will be a picture comparing the scanned image processed according to the computed mapping against the standard Macbeth colors. It is a

good idea to use the debug option to check that the color patches are being located correctly, and to see how well *macbethcal* does at matching colors. The center of each patch will show the target color; the left side of each patch will show the original color, and the right side will show the corrected value. If the match works well, the debug picture should have a sort of "notch on the left" look in each patch. Macbeth colors that could not be matched because they were out of gamut on this device are indicated with diagonal lines drawn through the associated target colors.

## METHOD

*Macbethcal* computes the color mapping in two stages. The first stage uses the six neutral color patches at the bottom of the Macbeth chart to compute a piecewise linear approximation to the brightness mapping of each RGB primary. The second stage looks at all the colors that are within the device's gamut to compute a least-sqaures fit for a linear color transformation from the measured space into the standard Radiance RGB space (as defined by the three primaries in src/common/color.h).

Thanks to the nature of inverse mappings, this method should work either for converting scanned data to match the original, or for preconditioning pictures to be sent to specific output devices. In other words, the same calibration file works either for correcting scanned images OR precorrecting images before printing.

A warning is printed if some unsaturated colors are determined to be out of gamut, as this may indicate a poor rendition or improper picture alignment. The debug picture will show which colors were excluded by drawing diagonal lines through their entries.

## NOTE

It is very important that the same settings be applied when scanning or printing other images to be calibrated with the computed file. In particular, all exposure adjustments should be fixed manually, and no tweaking of the settings should be done along the way. The final result will be best if the original scanned image is not too far off from what it should be. In the case of slide and negative scanners, it is best to apply the recommended calibration file for the type of film used, so long as this calibration is fixed and not adjusted on a per-image basis.

## CHART AVAILABILITY

The Macbeth chart is available at most photographic supply stores, or may be ordered directly from Macbeth:

Macbeth
Munsell Color
405 Little Britain Rd.
New Windsor, NY 12553-6148
tel. 1-800-622-2384 (USA)
fax. 1-914-561-0267

The chart sells for under $50 US at the time of this writing.

## EXAMPLES

To compute a calibration for a FunkyThing scanner and check the results:

```
ra_tiff -r mbscan.tif mbscan.hdr
macbethcal -d debug.hdr mbscan.hdr FunkyThing.cal
ximage debug.hdr
```

To apply this computed calibration to another scanned image:

```
ra_tiff -r another.tif | pcomb -f FunkyThing.cal - > another_calib.hdr
```

To compute a calibration file for the BigWhiz film recorder, after taking measurements of a slide made from macbeth_spec.hdr:

```
macbethcal -c macbeth_spec.xyY BigWhiz.cal
```

To prepare a picture prior to output on the same film recorder:

```
pcomb -f BigWhiz.cal standard.hdr > toprint.hdr
```

To use *pcond(1)* to also adjust the image for human response:

```
pcond -f BigWhiz.cal -h standard.hdr > toprint.hdr
```

## AUTHOR

Greg Ward
Paul Heckbert supplied code for perspective projective mapping

## SEE ALSO

icalc(1), pcomb(1), pcond(1), pfilt(1), ximage(1)

## NAME

**cbreak**, **nocbreak**, **echo**, **noecho**, **halfdelay**, **intrflush**, **keypad**, **meta**, **nodelay**, **notimeout**, **raw**, **noraw**, **noqiflush**, **qiflush**, **timeout**, **wtimeout**, **typeahead** - **curses** input options

## SYNOPSIS

```
#include <curses.h>

int cbreak(void);
int nocbreak(void);
int echo(void);
int noecho(void);
int halfdelay(int tenths);
int intrflush(WINDOW *win, bool bf);
int keypad(WINDOW *win, bool bf);
int meta(WINDOW *win, bool bf);
int nodelay(WINDOW *win, bool bf);
int raw(void);
int noraw(void);
void noqiflush(void);
void qiflush(void);
int notimeout(WINDOW *win, bool bf);
void timeout(int delay);
void wtimeout(WINDOW *win, int delay);
int typeahead(int fd);
```

## DESCRIPTION

Normally, the tty driver buffers typed characters until a newline or carriage return is typed. The **cbreak** routine disables line buffering and erase/kill character-processing (interrupt and flow control characters are unaffected), making characters typed by the user immediately available to the program. The **nocbreak** routine returns the terminal to normal (cooked) mode.

Initially the terminal may or may not be in **cbreak** mode, as the mode is inherited; therefore, a program should call **cbreak** or **nocbreak** explicitly. Most interactive programs using **curses** set the **cbreak** mode. Note that **cbreak** overrides **raw**. [See **curs_getch**(3X) for a discussion of how these routines interact with **echo** and **noecho**.]

The **echo** and **noecho** routines control whether characters typed by the user are echoed by **getch** as they are typed. Echoing by the tty driver is always disabled, but initially **getch** is in echo mode, so characters typed are echoed. Authors of most interactive programs prefer to do their own echoing in a controlled area of the screen, or not to echo at all, so they disable echoing by calling **noecho**. [See **curs_getch**(3X) for a discussion of how these routines interact with **cbreak** and **nocbreak**.]

The **halfdelay** routine is used for half-delay mode, which is similar to **cbreak** mode in that characters typed by the user are immediately available to the program. However, after blocking for *tenths* tenths of seconds, ERR is returned if nothing has been typed. The value of **tenths** must be a number between 1 and 255. Use **nocbreak** to leave half-delay mode.

If the **intrflush** option is enabled, (*bf* is **TRUE**), when an interrupt key is pressed on the keyboard (interrupt, break, quit) all output in the tty driver queue will be flushed, giving the effect of faster response to the interrupt, but causing **curses** to have the wrong idea of what is on the screen. Disabling (*bf* is **FALSE**), the option prevents the flush. The default for the option is inherited from the tty driver settings. The window argument is ignored.

The **keypad** option enables the keypad of the user's terminal. If enabled (*bf* is **TRUE**), the user can press a function key (such as an arrow key) and **wgetch** returns a single value representing the function key, as in **KEY_LEFT**. If disabled (*bf* is **FALSE**), **curses** does not treat function keys specially and the program has to interpret the escape sequences itself. If the keypad in the terminal can be turned on (made to transmit) and off (made to work locally), turning on this option causes the terminal keypad to be turned on when **wgetch** is called. The default value for keypad is false.

Initially, whether the terminal returns 7 or 8 significant bits on input depends on the control mode of the tty driver [see termio(7)]. To force 8 bits to be returned, invoke **meta**(*win*, **TRUE**); this is equivalent, under POSIX, to setting the CS8 flag on the terminal. To force 7 bits to be returned, invoke **meta**(*win*, **FALSE**); this is equivalent, under POSIX, to setting the CS7 flag on the terminal. The window argument, *win*, is always ignored. If the terminfo capabilities **smm** (meta_on) and **rmm** (meta_off) are defined for the terminal, **smm** is sent to the terminal when **meta**(*win*, **TRUE**) is called and **rmm** is sent when **meta**(*win*, **FALSE**) is called.

The **nodelay** option causes **getch** to be a non-blocking call. If no input is ready, **getch** returns **ERR**. If disabled (*bf* is **FALSE**), **getch** waits until a key is pressed.

While interpreting an input escape sequence, **wgetch** sets a timer while waiting for the next character. If **notimeout**(*win*, **TRUE**) is called, then **wgetch** does not set a timer. The purpose of the timeout is to differentiate between sequences received from a function key and those typed by a user.

The **raw** and **noraw** routines place the terminal into or out of raw mode. Raw mode is similar to **cbreak** mode, in that characters typed are immediately passed through to the user program. The differences are that in raw mode, the interrupt, quit, suspend, and flow control characters are all passed through uninterpreted, instead of generating a signal. The behavior of the BREAK key depends on other bits in the tty driver that are not set by **curses**.

When the **noqiflush** routine is used, normal flush of input and output queues associated with the **INTR**, **QUIT** and **SUSP** characters will not be done [see termio(7)]. When **qiflush** is called, the queues will be flushed when these control characters are read. You may want to call **noqiflush()** in a signal handler if you want output to continue as though the interrupt had not occurred, after the handler exits.

The **timeout** and **wtimeout** routines set blocking or non-blocking read for a given window. If *delay* is negative, blocking read is used (i.e., waits indefinitely for input). If *delay* is zero, then non-blocking read is used (i.e., read returns **ERR** if no input is waiting). If *delay* is positive, then read blocks for *delay* milliseconds, and returns **ERR** if there is still no input. Hence, these routines provide the same functionality as **nodelay**, plus the additional capability of being able to block for only *delay* milliseconds (where *delay* is positive).

The **curses** library does ‘‘line-breakout optimization’’ by looking for typeahead periodically while updating the screen. If input is found, and it is coming from a tty, the current update is postponed until **refresh** or **doupdate** is called again. This allows faster response to commands typed in advance. Normally, the input FILE pointer passed to **newterm**, or **stdin** in the case that **initscr** was used, will be used to do this typeahead checking. The **typeahead** routine specifies that the file descriptor *fd* is to be used to check for typeahead instead. If *fd* is -1, then no typeahead checking is done.

## RETURN VALUE

All routines that return an integer return **ERR** upon failure and OK (SVr4 specifies only "an integer value other than **ERR**") upon successful completion, unless otherwise noted in the preceding routine descriptions.

X/Open does not define any error conditions. In this implementation, functions with a window parameter will return an error if it is null. Any function will also return an error if the terminal was not initialized. Also,

**halfdelay**
: returns an error if its parameter is outside the range 1..255.

## PORTABILITY

These functions are described in the XSI Curses standard, Issue 4.

The ncurses library obeys the XPG4 standard and the historical practice of the AT&T curses implementations, in that the echo bit is cleared when curses initializes the terminal state. BSD curses differed from this slightly; it left the echo bit on at initialization, but the BSD **raw** call turned it off as a side-effect. For best portability, set echo or noecho explicitly just after initialization, even if your program remains in cooked mode.

## NOTES

Note that **echo**, **noecho**, **halfdelay**, **intrflush**, **meta**, **nodelay**, **notimeout**, **noqiflush**, **qiflush**, **timeout**, and **wtimeout** may be macros.

The **noraw** and **nocbreak** calls follow historical practice in that they attempt to restore to normal ('cooked') mode from raw and cbreak modes respectively. Mixing raw/noraw and cbreak/nocbreak calls leads to tty driver control states that are hard to predict or understand; it is not recommended.

## SEE ALSO

**curses**(3X), **curs_getch**(3X), **curs_initscr**(3X), **termio**(7)

## NAME

meta2bmp - convert metafile to Windows Bitmap (BMP) File

## SYNOPSIS

**meta2bmp** [ **−c | −r** ][ **−x width** ][ **−y height** ][ **−m minrad** ][ **−o outname** ] file ..

## DESCRIPTION

*Meta2bmp* reads each metafile *file* in sequence and converts it to a compressed, color-mapped BMP file. The result is sent to the standard output (which must be redirected) unless the −*o* option is used. The argument to the −*o* option specifies the base file name, to which a page number and ".bmp" is added as a suffix. Note that this option must be present in order to produce more than a single page of output.

The default output resolution is 400 by 400, but a different resolution can be given with the −*x* and −*y* options.

The −*m* option can be used to set a minimum value for the line radius in pixels. This may be helpful for improving the readability of high resolution output. The default value is 0, which allows lines of one pixel thickness.

If the option −*c* is specified, the input files are only conditioned for output, ie. expanded (see pexpand). This is useful if many copies of the same output is desired. If the option −*r* is instead specified, the input is assumed already to be conditioned. If no input files are specified, the standard input is read.

## EXAMPLE

To convert the plots examp1.plt and examp2.plt to 1024x1024 Targa files:

```
bgraph examp1.plt examp2.plt | meta2bmp −o examp −x 1024 −y 1024
```

## FILES

see pexpand(1) and psort(1)

## AUTHOR

Greg Ward

## SEE ALSO

bgraph(1), igraph(1), meta2tga(1), pexpand(1), psmeta(1), psort(1), ra_t8(1)

## NAME

meta2tga - convert metafile to Targa image format

## SYNOPSIS

**meta2tga** [ **−c** | **−r** ][ **−x width** ][ **−y height** ][ **−m minrad** ][ **−o outname** ] file ..

## DESCRIPTION

*Meta2tga* reads each metafile *file* in sequence and converts it to a compressed, color-mapped Targa file. The result is sent to the standard output (which must be redirected) unless the −*o* option is used. The argument to the −*o* option specifies the base file name, to which a page number and ".tga" is added as a suffix. Note that this option must be present in order to produce more than a single page of output.

The default output resolution is 400 by 400, but a different resolution can be given with the −*x* and −*y* options.

The −*m* option can be used to set a minimum value for the line radius in pixels. This may be helpful for improving the readability of high resolution output. The default value is 0, which allows lines of one pixel thickness.

If the option −*c* is specified, the input files are only conditioned for output, ie. expanded (see pexpand). This is useful if many copies of the same output is desired. If the option −*r* is instead specified, the input is assumed already to be conditioned. If no input files are specified, the standard input is read.

## EXAMPLE

To convert the plots examp1.plt and examp2.plt to 1024x1024 Targa files:

```
bgraph examp1.plt examp2.plt | meta2tga −o examp −x 1024 −y 1024
```

## FILES

see pexpand(1) and psort(1)

## AUTHOR

Greg Ward

## SEE ALSO

bgraph(1), igraph(1), meta2bmp(1), pexpand(1), psmeta(1), psort(1), ra_t8(1)

## NAME

metafile - graphics command interface, similar to plot(5)

## DESCRIPTION

The *metafile* graphics format was designed with the primary goal of serving as a temporary file for routines which output to dot-matrix and other line-at-a-time devices. As a result, all of the "primitives" are completely self-contained to facilitate sorting.

A primitive is a command which can itself be plotted. Into this catagory fall line segments, rectangle and triangle fills, matrix and vector strings. Every primitive has a zeroeth argument which contains bundled attribute information, and an extent. The extent gives the x and y minimum and maximum values which enclose the primitive. The extent is used in sorting, and typically also in describing the primitive. For example, a line segment will be described completely by its enclosing rectangle and attributes including specification of which diagonal the segment falls on. Other primitives will have additional arguments, such as vector string, which must specify the string to be output within its extent.

"Global" commands separate the primitives and allow functions which affect all commands. These are commands such as end of page, pause, open and close segment, set, unset and reset, and a special global, end of file. The end of file command is included to facilitate finding the end of file on systems which do not keep track exactly. Global commands sometimes have arguments. The open command, for instance, specifies the name of the segment. Global commands never have extents.

The metafile commands are as follows:

**F** end of file: no arguments.
When end of file is reached, all processing stops.

**E** end of page: no arguments.
This causes the device to advance to the next screen or page. If the output device is a terminal, it will beep and wait for the user to hit return before clearing the screen.

**P** pause: arguments specify the message to be printed.
This causes output to be flushed and the controlling terminal to be opened. The user is then prompted with the specified string followed by the message "- (hit return to continue)". If no string is specified, the bell is sounded without a message. After the user hits return, output continues. This command is useful when the user is required for some part of the output, such as changing paper or pens.

**D** draw global: no arguments.
This global forces flushing of output and updating of device.

**I** include file: arg0 TRUE if standard file.
The include global causes the contents of the named file to be substituted in the include command's location. If arg0 is 1 (TRUE), a standard location is searched if the file is not found in the working directory. If arg0 is 0 (FALSE), the file must be in the working directory. Include files can be nested to the number of allowed open files.

**S** set: arg0 specifies what to set (from meta.h):
SALL: place context mark on current settings.
SPAT0: set pattern 0 to the specified value.
SPAT1: set pattern 1 to the specified value.
SPAT2: set pattern 2 to the specified value.
SPAT3: set pattern 3 to the specified value.
The set command is used to globally affect certain attributes. The zeroeth argument specifies the variable to set, and the arguments following specify the value. Pattern values can have two forms. The first form begins with the letter 'P', immediately followed by an integer between 0 and 11. This selects one from the following patterns: solid, thick \\\, thin \\\, mixed \\\, thick ///, thin ///, mixed ///, crisscross, web. The default pattern settings are: 0=P0, 1=P1, 2=P2, 3=P3. The second form gives the explicit values for a pattern. The set all command makes a context mark with the current settings. All settings which follow can be undone with the unset all command.

**U** unset: arg0 specifies what to unset (from meta.h):
SALL: return to previous context.
SPAT0: set pattern 0 to the previous value.
SPAT1: set pattern 1 to the previous value.
SPAT2: set pattern 2 to the previous value.
SPAT3: set pattern 3 to the previous value.
The unset command returns a variable to its previous value. The unset all command returns the settings to the values they had in the previous context. If no context has been marked by set all, variables are returned to their default values.

**R** reset: arg0 specifies what to reset (from meta.h):
SALL: reset all variables.
SPAT0: set pattern 0 to the default value.
SPAT1: set pattern 1 to the default value.
SPAT2: set pattern 2 to the default value.
SPAT3: set pattern 3 to the default value.
The reset command returns a variable to its default setting. The reset all command returns all variables to their initial state.

**O** open segment: arguments specify segment name.
The commands following up to a C (close segment) are not to be output, but are to be stored in the named segment. Segment names can contain any ascii character (except newline) in any sequence of reasonable length. Segment definitions are local to the enclosing segment. Side effects should be avoided in segments by balancing calls to set and unset. A segment cannot reference itself.

**C** close segment: no arguments.
The current segment is closed, which completes its usable definition.

**l** line segment: fields of arg0 are:
100: orientation: positive slope, negative slope.
060: type: solid, dashed, dotted, dotted-dashed.
014: width: 0, 12, 24, 48, 96 units.
003: color: black, red, green, blue.

**r** rectangle fill: fields of arg0 are:
100: toggle: OR fill, XOR fill.
014: pattern: choice of 4 (see set).
003: color: black, red, green, blue.
Fills the given extent with the specified pattern. Toggle (XOR) fill allows the reversal of previous fills to an area.

**t** triangle fill: fields of arg0 are:
100: toggle: OR fill, XOR fill.
060: orientation: right (& down), up, left, down.
014: pattern: choice of 4 (see set).
003: color: black, red, green, blue.
Fills the given half-rectangle with the specified pattern. A triangle is oriented to the right if the the area between the positive-sloped diagonal and the lower right corner of the extent is filled. Rotating this triangle ccw successively yields up, left and down triangles. Toggle (XOR) fill allows the reversal of previous fills to an area.

**p** polygon fill: fields of arg0 are:
100: border: no border, line border.
060: orientation: right (& down), up, left, down.
014: pattern: choice of 4 (see set).
003: color: black, red, green, blue.
The argument string gives a blank separated list of the polygon vertices in the form: "x0 y0 x1 y1 x2 y2 ... ". The coordinates must be integers ranging between 0 and 16383. The bounding box and orientation will be used to fit the original polygon into a scaled and rotated position. The last vertex will be

connected to the first, and the polygon will be filled in with the specified pattern. If a border is requested, one will be drawn of solid black zero width lines. All polygon fills will toggle, therefore other polygon and toggled triangle and rectangle fills will affect the final appearance of the image. For example, a polygon drawn inside another polygon of the same pattern will make a hole.

**m** matrix string: fields of arg0 are:
100: strike: single, double.
060: density: 10 cpi, 12 cpi, 17 cpi, 20 cpi.
014: size: normal, double width, double height, double both.
003: color: black, red, green, blue.
The upper left corner of the extent is used to place the beginning of the string specified after the command. More sophisticated drivers will use the extent for clipping, but the size of the characters will not be altered.

**v** vector string: fields of arg0 are:
060: orientation: right, up, left, down.
014: thickness: 0, 12, 24, 48, 96 units.
003: color: black, red, green, blue.
The string specified following the command will be made to fit within the given extent.

**s** print segment: fields of arg0 are:
060: orientation: right, up, left, down.
014: thickness: 0, 12, 24, 48, 96 units.
003: color: black, red, green, blue.
The segment whose name is specified in the arguments will be oriented according to arg0 and made to fit in the given extent. The thickness and color of the lines in the segment will be changed also according to arg0. In the case of area fill, it is the pattern rather than the width which will change. The segment must have been previously defined using the open segment global. Note that matrix strings will not transfer well since they cannot be oriented or scaled.

The metafile has two basic formats. The first format is meant to be user readable, and has the form:

```
c arg0 xmin ymin xmax ymax ‘args
```

Where c is the single letter command, arg0 is the octal value for arg0, xmin ymin xmax ymax are the extent (ranging from 0 to 16283), and the optional args following the backquote are additional arguments, terminated by a newline. If the command is a global, the extent is not present. If the global has no arg0, 0200 is appropriate. Any global which has a following string must have a value for arg0 (< 0200). Comments are permitted on lines beginning with a pound sign ('#').

The second format is roughly equivalent, but packs the extrema into two bytes each. It takes between one quarter and one third as much space, and much less processing to use this type of file, hence it is the default format for all of the programs. Conversion between formats is accomplished with cv(1).

## FILES

The standard location for metafiles used by the programs is /usr/lib/meta/, but can be changed by setting the environment variable MDIR. This is useful for systems where the owner does not have access to the /usr/lib/ directory. It also allows the user to create his own metafiles for vector characters and other symbols.

## BUGS

The command for line segment ('l') is awkward at best.

## AUTHOR

Greg Ward

## SEE ALSO

cv(1), meta(3), pexpand(1), primout(3), psort(1)

## NAME

mgf2meta - convert Materials and Geometry Format file to Metafile graphics

## SYNOPSIS

**mgf2meta** [ **-t threshold** ] **{x|y|z} xmin xmax ymin ymax zmin zmax** [ **input ..** ]

## DESCRIPTION

*Mgf2meta* converts one or more Materials and Geometry Format (MGF) files to a 2-D orthographic projection along the selected axis in the *metafile(1)* graphics format. All geometry is clipped to the specified bounding box, and the resulting orientation is as follows:

| Projection | Orientation |
|---|---|
| ========== | ========== |
| x | Y-axis right, Z-axis up |
| y | Z-axis right, X-axis up |
| z | X-axis right, Y-axis up |

If multiple input files are given, the first file prints in black, the second prints in red, the third in green and the fourth in blue. If more than four input files are given, they cycle through the colors again in three other line types: dashed, dotted and dot-dashed.

The *−t* option may be used to randomly throw out line segments that are shorter than the given *threshold* (given as a fraction of the plot width). Segments are included with a probability equal to the square of the line length over the square of the threshold. This can greatly reduce the number of lines in the drawing (and therefore improve the drawing speed) with only a modest loss in quality. A typical value for this parameter is 0.005.

All MGF material information is ignored on the input.

## EXAMPLE

To project two MGF files along the Z-axis and display them under X11:

```
mgf2meta z 0 10 0 15 0 9 building1.mgf building2.mgf | x11meta −r
```

To convert a RADIANCE scene to a line drawing in RADIANCE picture format:

```
rad2mgf scene.rad | mgf2meta x ‘getbbox −h scene.rad‘ | meta2tga | ra_t8 −r > scene.hdr
```

## AUTHOR

Greg Ward

## SEE ALSO

getbbox(1), meta2bmp(1), meta2tga(1), metafile(5), mgf2rad(1), pflip(1), protate(1), psmeta(1), ra_t8(1), rad2mgf(1), x11meta(1)

MGF web site "http://radsite.lbl.gov/mgf/HOME.html"

## NAME

mgf2rad - convert Materials and Geometry Format file to RADIANCE description

## SYNOPSIS

**mgf2rad** [ **−m matfile** ][ **−e mult** ][ **−g dist** ] [ **input ..** ]

## DESCRIPTION

*Mgf2rad* converts one or more Materials and Geometry Format (MGF) files to a RADIANCE scene description. By definition, all output dimensions are in meters. The material names and properties for the surfaces will be those assigned in MGF. Any materials not defined in MGF will result in an error during translation. Light sources are described inline as IES luminaire files, and *mgf2rad* calls the program *ies2rad(1)* to translate these files. If an IES file in turn contains an MGF description of the local fixture geometry, this may result in a recursive call to *mgf2rad,* which is normal and should be transparent. The only side-effect of this additional translation is the appearance of other RADIANCE scene and data files produced automatically by *ies2rad*.

The *−m* option may be used to put all the translated materials into a separate RADIANCE file. This is not always advisable, as any given material name may be reused at different points in the MGF description, and writing them to a separate file loses the contextual association between materials and surfaces. As long as unique material names are used throughout the MGF description and material properties are not redefined, there will be no problem. Note that this is the only way to get all the translated materials into a single file, since no output is produced for unreferenced materials; i.e. translating just the MGF materials does not work.

The *−e* option may be used to multiply all the emission values by the given *mult* factor. The *−g* option may be used to establish a glow distance (in meters) for all emitting surfaces. These two options are employed principally by *ies2rad,* and are not generally useful to most users.

## EXAMPLE

To translate two MGF files into one RADIANCE materials file and one geometry file:

```
mgf2rad −m materials.rad building1.mgf building2.mgf > building1+2.rad
```

To create an octree directly from two MGF files and one RADIANCE file:

```
oconv '\!mgf2rad materials.mgf scene.mgf' source.rad > scene.oct
```

## FILES

| | |
|---|---|
| tmesh.cal | Used to smooth polygonal geometry |
| *.rad | RADIANCE source descriptions created by ies2rad |
| *.dat | RADIANCE source data created by ies2rad |
| source.cal | Used for IES source coordinates |

## AUTHOR

Greg Ward

## SEE ALSO

ies2rad(1), mgf2meta(1), obj2rad(1), oconv(1), rad2mgf(1), xform(1)

MGF web site "http://radsite.lbl.gov/mgf/HOME.html"

## NAME

mkillum - compute illum sources for a RADIANCE scene

## SYNOPSIS

**mkillum** [ **−n nprocs** ][ **rtrace options** ] **octree** [ **< file ..** ]
**mkillum** [ **rtrace options** ] **−defaults**

## DESCRIPTION

*Mkillum* takes a prepared RADIANCE scene description and an octree and computes light source distributions for each surface, replacing them with secondary sources whose contributions can be computed more efficiently by *rpict(1)* and *rvu(1)*. This type of optimization is most useful for windows and skylights which represent concentrated sources of indirect illumination. *Mkillum* is not appropriate for very large sources or sources with highly directional distributions. These are best handled respectively by the ambient calculation and the secondary source types in RADIANCE.

If the *−n* option is specified with a value greater than 1, multiple ray tracing processes will be used to accelerate computation on a shared memory machine. Note that there is no benefit to using more processes than there are local CPUs available to do the work.

Remaining arguments to *mkillum* are interpreted as rendering options for *rtrace(1),* to compute the light distributions for the input surfaces. These surfaces can be any combination of polygons, spheres and rings. Other surfaces may be included, but *mkillum* cannot compute their distributions.

By default, *mkillum* reads from its standard input and writes to its standard output. It is possible to specify multiple input files in a somewhat unconventional fashion by placing a lesser-than symbol ('<') before the file names. (Note that this character must be escaped from most shells.) This is necessary so *mkillum* can tell where the rendering arguments end and its own input files begin.

## VARIABLES

*Mkillum* has a number of parameters that can be changed by comments in the input file of the form:

```
#@mkillum variable=value option switch{+|-} ..
```

String or integer variables are separated from their values by the equals sign ('='). Options appear by themselves. Switches are followed either by a plus sign to turn them on or a minus sign to turn them off.

Parameters are usually changed many times within the same input file to tailor the calculation, specify different labels and so on. The parameters and their meanings are described below.

**o**=*string* Set the output file to *string*. All subsequent scene data will be sent to this file. If this appears in the first comment in the input, nothing will be sent to the standard output. Note that this is not recommended when running *mkillum* from *rad(1),* which expects the output to be on the standard output.

**m**=*string* Set the material identifier to *string*. This name will be used not only as the new surface modifier, but it will also be used to name the distribution pattern and the data files. The distribution name will be *string* plus the suffix ".dist". The data file will be named *string* plus possibly an integer plus a ".dat" suffix. The integer is used to avoid accidently writing over an existing file. If overwriting the file is desired, use the *f* variable below.

**f**=*string* Set the data file name to *string*. The next data file will be given this name plus a ".dat" suffix. Subsequent files will be named *string* plus an integer plus the ".dat" suffix. An existing file with the same name will be clobbered. This variable may be unset by leaving off the value. (See also the *m* variable above.)

**a** Produce secondary sources for all of the surfaces in the input. This is the default.

**e**=*string* Produce secondary sources for all surfaces except those modified by *string*. Surfaces modified by *string* will be passed to the output unchanged.

**i**=*string* Only produce secondary sources for surfaces modified by *string*.

**n** Do not produce any secondary sources. All input will be passed to the output unaffected, except any void surfaces will be removed.

**b**=*real* Do not produce a secondary source for a surface if its average brightness (radiance) is less than the value *real*.

**c**=*{d|a|n}* Use color information according to the given character. If the character is *d,* then color information will be used in three separate data files and the distribution will be fully characterized in terms of color. If the character is *a,* then only the average color is computed and the distribution will not contain color information. If the character is *n,* even the average distribution color will be thrown away, producing secondary sources that are completely uncolored. This may be desirable from a color-balancing point of view.

**d**=*integer* Set the number of direction samples per projected steradian to *integer.* The number of directions stored in the associated data file will be approximately this number multiplied by pi for polygons and rings, and by 4pi for spheres. If *integer* is zero, then a diffuse source is assumed and no distribution is created.

**s**=*integer* Set the number of ray samples per direction to *integer.* This variable affects the accuracy of the distribution value for each direction as well as the computation time for *mkillum*.

**l{+|-}** Switch between light sources and illum sources. If this switch is enabled *(l+), mkillum* will use the material type "light" to represent surfaces. If disabled *(l-), mkillum* will use the material type "illum" with the input surface modifier as its alternate material. The default is *l-*.

## EXAMPLES

The following command generates illum's corresponding to geometry in the files "it1.rad" and "it2.rad":

```
mkillum −ab 2 −ad 1024 −av .1 .1 .1 basic.oct "<" it1.rad it2.rad > illums.rad
```

The output file "illums.rad" would then be combined with the original scene geometry to create a more easily rendered composite.

## ENVIRONMENT

RAYPATH the directories to check for auxiliary files.

## AUTHOR

Greg Ward

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland.

## SEE ALSO

genBSDF(1), oconv(1), rad(1), rpict(1), rtrace(1), rvu(1)

## NAME

mksource - compute distant sources for RADIANCE image-based lighting

## SYNOPSIS

**mksource** [ **−d nsamps** ][ **−t thresh** ][ **−a maxang** ] [ **octree** ]

## DESCRIPTION

*Mksource* produces a RADIANCE scene description of distant illum sources corresponding to bright, concentrated regions in the given ambient environment. Any local geometry is ignored in the input octree, which should be derived from a captured light probe and modeled as a distant hemispherical or spherical glow source. The output sources may then be combined with this environment to produce a more efficient scene for rendering, faster and less prone to sampling artifacts.

The *−d* option may be used to specify the number of ray samples, which defaults to 262,000. Calculation time is roughly proportional to this setting, and the default is fine enough to resolve sources the size of the sun (half a degree) or larger.

The *−t* option may be used to manually set the radiance threshold for sources, in watts/sr/meter^2. The default uses the top 2 percentile of the environment, which is usally a good value.

The *−a* option may be used to specify a maximum source diameter, which defaults to 15 degrees. *Mkillum* silently enforces a maximum of 180 degrees for this option.

If no input octree is specified, the standard input is read.

## EXAMPLE

To add sources with a maximum size of 20 degrees to the enviornment described in environ.oct:

```
mksource −a 20 environ.oct > srcs.rad
oconv −i environ.oct srcs.rad > env+srcs.oct
```

## AUTHOR

Greg Ward

## SEE ALSO

mkillum(1), rpict(1), rvu(1)

## NAME

neaten - neaten up output columns

## SYNOPSIS

**neaten** [ **format** ]

## DESCRIPTION

*Neaten* reads from its standard input and neatens up columns separated by white space using the specified format. The format is a string consisting of a positive integer followed by an alignment character and another integer. The alignment character is usually a decimal point ('.'), but it can be any non-digit.

The alignment character is used as the central point of each column. The total column field width will be the number to the left of the alignment character plus one for the alignment character itself plus the number to the right of the alignment character.

If a field does not contain the alignment character, it will be printed to the left of where the alignment character would have appeared. If a field is too long to print within the specified format, the entire field will be printed and that row will not be aligned with the rest.

The default format is "8.8".

## EXAMPLE

To examine a file with columns of numbers:

```
neaten 10.8 < input | more
```

## BUGS

Columns wider than the total width of the format specification will be printed without any separating white space.

The program does not do anything special with tabs on the input.

## AUTHOR

Greg Ward

## SEE ALSO

cnt(1), rcalc(1), rcollate(1), rlam(1), total(1)

## NAME

normpat - normalize RADIANCE pictures for use as patterns.

## SYNOPSIS

**normpat** [ **−v** ][ **−b** ][ **−f** ][ **−r maxres** ] **picture ..**

## DESCRIPTION

*Normpat* normalizes one or more RADIANCE pictures to an average brightness of 1.0 and optionally removes fundamental frequencies and blends the edges of the image. The original images are overwritten during this process, and it is recommended that the program work on copies of the pictures for this reason.

The *−r* option can be used to set the maximum horizontal or vertical resolution of the final result, which should not be greater than 256 for most patterns (due to the associated memory burden during rendering). The *−f* option uses a Fourier transform to remove the lowest frequencies from the image, reducing the noticeability of pattern repitition. The *−b* option can be used to blend the edges of the image so that when it is tiled, the seams are less apparent. The *−v* option turns on the verbose flag, which prints on the standard output progress messages as the script runs.

*Normpat* is a shell script that makes calls to other RADIANCE programs that do the actual work.

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), pcomb(1), pcompos(1), pfilt(1), pflip(1), protate(1), psign(1), ra_bn(1), ra_pr(1), ra_t8(1), ra_t16(1), rpict(1)

## NAME

normtiff - tone-map and convert RADIANCE picture or HDR TIFF to standard TIFF

## SYNOPSIS

**normtiff** [ **options** ] **input output.tif**

## DESCRIPTION

*Normtiff* prepares a Radiance picture or high dynamic-range TIFF for output to a display or hard copy device. If the dynamic range of the scene exceeds that of the display (as is usually the case), *normtiff* will compress the dynamic range of the picture such that both dark and bright regions are visible. In addition, certain limitations in human vision may be mimicked in order to provide an appearance similar to the experience one might have in the actual scene.

Output is always an uncompressed RGB TIFF, which must be named on the command line along with the input file. If the input file has a ".tif" or ".tiff" extension, *normtiff* attempts to read it as a TIFF. Otherwise, *normtiff* first tries opening it as a RADIANCE picture, only opening it as a TIFF if it fails header inspection. (See the *getinfo(1)* program.) If the input is neither a RADIANCE picture nor a high dynamic-range TIFF, the program reports an error and exits.

The following command line options are understood. Since this program is very similar to *pcond(1),* several of the switches are identical.

**-b** Toggle 8-bit black and white (grayscale) TIFF output. If the input is a grayscale TIFF, this switch is automatically selected. Otherwise, the output defaults to 24-bit RGB.

**-z** Output LZW-compressed TIFF (smaller file).

**-h** Mimic human visual response in the output. The goal of this process is to produce output that correlates strongly with a person's subjective impression of a scene. This switch turns on both the *−s* and *−c* switches, described below.

**-s** Toggle the use of the human contrast sensitivity function in determining the exposure for the image. A darker scene will have relatively lower exposure with lower contrast than a well-lit scene.

**-c** Toggle mesopic color correction. If parts of the image are in the mesopic or scotopic range where the cone photoreceptors lose their efficiency, this switch will cause a corresponding loss of color visibility in the output and a shift to a scotopic (blue-dominant) response function.

**-l** Toggle the use of a linear response function versus the standard dynamic range compression algorithm. This may make some parts of the resulting image too dark or too bright to see.

**-u** *Ldmax* Specifies the top of the luminance range for the target output device. That is, the luminance (in candelas/m^2) for an output pixel value of (R,G,B)=(255,255,255). This parameter affects tone mapping only when the *−s* switch is on. The default value is 100 cd/m^2.

**-d** *Lddyn* Specifies the dynamic range for the target output device, which is the ratio of the maximum and minimum usable display luminances. The default value is 32, which is typical for CRT monitors.

**-p** *xr yr xg yg xb yb xw yw*
Specifies the RGB primaries for the target output device. These are the 1931 CIE (x,y) chromaticity values for red, green, blue and white, respectively.

**-g** *gamma* Specifies the output device gamma correction value. The default value is 2.2, which is appropriate for most CRT monitors. (A value of 1.8 is common in color prepress and color printers.)

## EXAMPLES

To convert a RADIANCE picture to an 8-bit grayscale TIFF:

```
normtiff −b scene.hdr sceneb.tif
```

To condition a high dynamic-range TIFF for a particular film recorder with known color primaries, dynamic range and gamma response:

```
pcond -d 50 -g 2.5 -p .580 .340 .281 .570 .153 .079 .333 .333 orig.tif filmrgb.tif
```

To simulate human visual response on a monitor with known maximum luminance:

```
normtiff -h -u 80 scene.hdr sceneh.tif
```

## REFERENCE

Greg Ward Larson, Holly Rushmeier, Christine Piatko, "A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes," *IEEE Transactions on Visualization and Computer Graphics* , December 1997.

http://positron.cs.berkeley.edu/gwlarson/pixformat/

## AUTHOR

Greg Ward Larson

## ACKNOWLEDGMENT

This work was supported by Silicon Graphics, Inc.

## SEE ALSO

getinfo(1), pcond(1), pflip(1), pvalue(1), protate(1), ra_xyze(1), rpict(1), ximage(1)

## NAME

obj2mesh - create a compiled RADIANCE mesh file from Wavefront .OBJ input

## SYNOPSIS

**obj2mesh** [ **−a matfile** ][ **−l matlib** ][ **−n objlim** ][ **−r maxres** ][ **−w** ] [ **input.obj** [ **output.rtm** ] ]

## DESCRIPTION

*Obj2mesh* reads a Wavefront .OBJ file from *input.obj* (or the standard input) and compiles it into a RADIANCE triangle mesh, which is sent to *output.rtm* (or standard output). Any RADIANCE material descriptions included via one or more *−a* options will be compiled and stored in the mesh as well. If the *−l* option is used to specify a material file, the RADIANCE library locations are searched. This mesh may be included in a RADIANCE scene description via the *mesh* primitive, thus:

```
mod mesh id
1+ output.rtm [xform args]
0
0
```

The syntax and semantics are identical to the RADIANCE *instance* primitive. If *mod* is "void", then the stored mesh materials will be applied during rendering. Otherwise, the given material will be substituted on all the mesh surfaces.

The *−n* option specifies the maximum surface set size for each voxel. Larger numbers result in quicker mesh generation, but potentially slower rendering. Values below 6 are not recommended, since this is the median valence for a mesh vertex (the number of adjacent faces), and smaller values will result in pointless octree subdivision. The default setting is 9.

The *−r* option specifies the maximum octree resolution. This should be greater than or equal to the ratio of the mesh bounding box to the smallest triangle. The default is 16384.

The *−w* option suppresses warnings.

Although the mesh file format is binary, it is meant to be portable between machines. The only limitation is that machines with radically different integer sizes will not work together.

## DETAILS

The following Wavefront statements are understood and compiled by *obj2mesh*.

**v** *x y z* A vertex location, given by its Cartesian coordinates. The final mesh position may of course be modified by the transform arguments given to the *mesh* primitive in the Radiance scene description.

**vn** *dx dy dz*
A vertex normal vector, given by its three direction components, which will be normalized by *obj2mesh*. Normals will be interpolated over the mesh during rendering to produce a smooth surface. If no vertex normals are present, the mesh will appear tesselated. A zero length normal (i.e., 0 0 0) will generate a syntax error.

**vt** *u v* A local vertex texture coordinate. These coordinates will be interpolated and passed to the "Lu" and "Lv" variables during rendering. Local coordinates can extend over any desired range of values.

**usemtl** *mname*
A material name. The following faces will use the named material, which is taken from the material definitions in the *−a* input file(s).

**g** *gname* Group association. The following faces are associated with the named group. If no "usemtl" statement has been encountered, the current group is used for the surface material identifier.

**f** *v1/t1/n1 v2/t2/n2 v3/t3/n3* **..**
A polygonal face. Polygon vertices are specified as three indices separated by slashes ('/'). The first index is the vertex location, the second index is the local (u,v) texture coordinate, and the third index is the vertex surface normal. Positive indices count from the beginning of the

input, where the first vertex position ( *v* statement) is numbered 1, and likewise for the first texture coordinate and the first surface normal. Negative indices count backward from the current position in the input, where –1 is the last vertex encountered, –2 is the one before that, etc. An index of 0 may be used for the vertex texture or normal to indicate none, or these may be left off entirely. All faces will be broken into triangles in the final mesh.

All other statement types will be ignored on the input. Statements understood by *obj2rad(1)* will be ignored silently; other statements will generate a warning message after translation to indicate how much was missed.

## DIAGNOSTICS

There are four basic error types reported by obj2mesh:

warning - a non-fatal input-related error

fatal - an unrecoverable input-related error

system - a system-related error

internal - a fatal error related to program limitations

consistency - a program-caused error

Most errors are self-explanatory. However, the following internal errors should be mentioned:

Set overflow in addobject (id)
: This error occurs when too many surfaces are close together in a scene. Sometimes a dense mesh can be accommodated by increasing the maximum resolution (by powers of two) using the *–r* option, but usually this error indicates something is wrong. Either too many surfaces are lying right on top of each other, or the bounding cube is inflated from disparate geometry in the input. Chances are, the face number "id" is near those causing the problem.

Hash table overflow in fullnode
: This error is caused by too many surfaces, and there is little hope of compiling this mesh.

## EXAMPLES

To create a compiled triangle mesh from the scene file mesh.obj using materials from the file mesh.mat:

```
obj2mesh –a mesh.mat mesh.obj mesh.rtm
```

To use local coordinates to place a square tiled image on a mesh object:

```
void colorpict tiled_pat
7 red green blue mytile.hdr . frac(Lu) frac(Lv)
0
0

tiled_pat plastic tiled_mat
0
0
5 .9 .9 .9 0 0

tiled_mat mesh tiled_mesh
1 mymesh.rtm
0
0
```

## ENVIRONMENT

| | |
|---|---|
| RAYPATH | the directories to search for material files. |

## AUTHOR

Greg Ward

## SEE ALSO

gensurf(1), getinfo(1), make(1), obj2rad(1), oconv(1), rpict(1), rvu(1), rtrace(1), xform(1)

## NAME

obj2rad - convert Wavefront .obj file to RADIANCE description

## SYNOPSIS

**obj2rad** [ **−n** ][ **−f** ][ **−m mapfile** ][ **−o objname** ] [ **input** ]

## DESCRIPTION

*Obj2rad* converts a Wavefront .obj file to a RADIANCE scene description. The material names for the surfaces will assigned based on the mapping rules file given in the *−m* option. If no mapping file is given, the identifiers given by the "usemtl" statements will be used as the material names. If no "usemtl" statements are found, the group names (given by the "g" statement) will be used instead. Failing this, the default material "white" will be used.

A mapping file contains a list of materials followed by the conditions a surface must satisfy in order to have that material. For example, if we wanted all faces in the Group "thingy" with texture Map "pine" to use the material "wood", and all other surfaces to use the material "default", we would create the following mapping file:

```
default ;
wood (Group "thingy") (Map "pine") ;
```

All faces would satisfy the first set of conditions (which is empty), but only the faces in the Group "thingy" with texture Map "pine" would satisfy the second set of conditions.

Each rule can have up to one condition per qualifier, and different translators use different qualifiers. In *obj2rad,* the valid qualifiers are *Material, Map, Group, Object* and *Face*. A condition is either a single value for a specific attribute, or an integer range of values. (Integer ranges are specified in brackets and separated by a colon, eg. [−15:27], and are always inclusive.) A semicolon is used to indicate the end of a rule, which can extend over several lines if necessary.

The semantics of the rule are such that "and" is the implied conjunction between conditions. Thus, it makes no sense to have more than one condition in a rule for a given qualifier. If the user wants the same material to be used for surfaces that satisfy different conditions, they simply add more rules. For example, if the user also wanted faces between 50 and 175 in the Group "yohey" to use "wood", they would add the following rule to the end of the example above:

```
wood (Face [50:175]) (Group "yohey") ;
```

Note that the order of conditions in a rule is irrelevant. However, the order of rules is very important, since the last rule satisfied determines which material a surface is assigned.

By convention, the identifier "void" is used to delete unwanted surfaces. A surface is also deleted if it fails to match any rule. Void is used in a rule as any other material, but it has the effect of excluding all matching surfaces from the translator output. For example, the following mapping would delete all surfaces in the Object "junk" except those with the Group name "beige", to which it would assign the material "beige_cloth", and all other surfaces would be "tacky":

```
tacky ;
void (Object "junk") ;
beige_cloth (Object "junk") (Group "beige") ;
```

The *−n* option may be used to produce a list of qualifiers from which to construct a mapping for the given .obj file. This is also useful for determining which materials must be defined when no mapping is used.

The *−f* option is used to flatten all faces, effectively ignoring vertex normal information. This is sometimes desirable when a smaller model or more robust rendering is desired, since interpolating vertex normals takes time and is not always reliable.

The *−o* option may be used to specify the name of this object, though it will be overriden by any "o"

statements in the input file. If this option is absent, and there are no "o" statements, *obj2rad* will attempt to name surfaces based on their group associations.

If no input files are given, the standard input is read.

## DETAILS

The following Wavefront statements are understood and translated by *obj2rad*.

**#** A comment. This statement is passed to the output verbatim. It has no effect.

**f** A polygonal face. If the vertices have associated surface normals, the face will be broken into quadrilaterals and triangles with the appropriate Radiance textures to interpolate them. Likewise, if the face is non-planar, it will be broken into triangles. Each face in the input file is assigned a number, starting with 1, and this number may be used in the material mapping rules.

**g** Group association. The following faces are associated with the named group(s). These may be used in the mapping rules, where a rule is matched if there is an association with the named Group. (I.e. since there may be multiple group associations, any match is considered valid.) If a mapping file is not used and no "usemtl" statement has been encountered, the main group is used for the surface material identifier.

**o** Object name. This is used to name the following faces, and may be used in the mapping rules.

**usemap** A texture map (i.e. Radiance pattern) name. The name may be used in the material mapping rules, but the indexing of Radiance patterns is not yet supported.

**usemtl** A material name. The name may be used in mapping rules, or will be used as the Radiance material identifier if no mapping is given.

**v** A vertex, given by its x, y and z coordinates.

**vn** A vertex normal, given by its x, y and z direction components. This vector will be normalized by *obj2rad,* and an error will result if it has length zero.

**vt** A vertex texture coordinate. Not currently used, but will be if we ever get around to supporting Wavefront textures.

All other statement types will be ignored on the input. A final comment at the end of the Radiance output file will give some indication of how successful the translation was, since it will mention the number of statements *obj2rad* did not recognize.

## EXAMPLE

To create a qualifier list for triceratops.obj:

```
obj2rad –n triceratops.obj > triceratops.qual
```

To translate triceratops.obj into a RADIANCE file using the mapping triceratops.map:

```
obj2rad –m triceratops.map triceratops.obj > triceratops.rad
```

## NOTES

Many good and useful Wavefront object files are available by anonymous ftp from "avalon.chinalake.navy.mil" in the /pub/objects/obj directory.

## FILES

tmesh.cal - used for triangle normal interpolation
surf.cal - used for quadrilateral normal interpolation

## AUTHOR

Greg Ward

## SEE ALSO

arch2rad(1), ies2rad(1), obj2mesh(1), oconv(1), thf2rad(1), xform(1)

## NAME

objline - create metafile line drawings of RADIANCE object(s)

## SYNOPSIS

**objline** [input ..]

## DESCRIPTION

*Objline* takes one or more RADIANCE scene files and produces four parallel line projections using calls to *rad2mgf(1)* and *mgf2meta(1)*. The output must be redirected to a suitable destination for *metafile(5)* 2-d graphics, such as *x11meta(1)* or *psmeta(1)*.

The four projections presented are along the X-axis (displayed in the upper left quadrant), along the Y-axis (upper right), along the Z-axis (lower left) and an oblique view (lower right). If multiple RADIANCE input files are given, they are shown in different colors and line styles. (Materials are ignored, so materials files are best left out.) If no input files are given on the command line, the standard input is read.

## EXAMPLES

To create a line drawing of the RADIANCE file "myfile.rad" and display under X11:

```
objline myfile.rad | x11meta –r &
```

To create a line drawing of three objects in different colors and send to the printer:

```
objline obj1.rad obj2.rad obj3.rad | psmeta | lpr
```

To create a line drawing of a room and convert into a 1024x1024 RADIANCE picture:

```
objline room.rad | meta2tga –x 1024 –y 1024 | ra_t8 –r > drawing.hdr
```

## AUTHOR

Greg Ward

## SEE ALSO

meta2tga(1), metafile(5), mgf2meta(1), psmeta(1), ra_t8(1), rad2mgf(1), x11meta(1)

## NAME

objview - view RADIANCE object(s)

## SYNOPSIS

**objview** [ **−u updirection** ][ rad options ] input ..
**objview** [ **−g** ][ **−u updirection** ][ glrad options ] input ..
**objview.rb** [ **−l** ][ **−r roomsiz** ][ **−n nprocs** ] input ..

## DESCRIPTION

*Objview* renders a RADIANCE object interactively using *rad(1)* or *glrad(1)*. This program is merely a shell script that adds some light sources to a scene then calls *rad(1)* or *glrad(1)* to make an octree and view the scene interactively.

If the default up vector (+Z) is inappropriate for this object, then specify a different one using the *−u* option to *objview*.

Any number of material and scene files may be given, but no in-line commands or standard input.

There is also a Ruby version of objview (objview.rb) available, primarily for Windows compatibility, but the Ruby version also adds a luminaire viewer option.

If the user specifies the -l option, then a Radiance luminaire description is the input to the program. objview.rb will place the centroid of the luminiare at 0,0,0 and surround the distribution with a room that is 4' square (in meters), with a view along the positive y. Scale the room size with the -r option.

The number of processors to use may be specified with the -n option.

## AUTHOR(s)

Greg Ward Larson, Rob Guglielmetti

## SEE ALSO

glrad(1), oconv(1), rad(1), rvu(1)

## NAME

oconv - create an octree from a RADIANCE scene description

## SYNOPSIS

**oconv** [ **−i octree** | **−b xmin ymin zmin size** ][ **−n objlim** ][ **−r maxres** ][ **−f** ][ **−w** ][ - ] [ **input ..** ]

## DESCRIPTION

*Oconv* adds each scene description *input* to *octree* and sends the result to the standard output. Each *input* can be either a file name, or a command (enclosed in quotes and preceded by a '!'). Similarly, the octree input may be given as a command preceded by a '!'. If any of the surfaces will not fit in *octree,* an error message is printed and the program aborts. If no *octree* is given, a new one is created large enough for all of the surfaces.

The *−b* option allows the user to give a bounding cube for the scene, starting at *xmin ymin zmin* and having a side length *size*. If the cube does not contain all of the surfaces, an error results. The *−b* and *−i* options are mutually exclusive.

The *−n* option specifies the maximum surface set size for each voxel. Larger numbers result in quicker octree generation, but potentially slower rendering. Smaller values may or may not produce faster renderings, since the default number (6) is close to optimal for most scenes.

The *−r* option specifies the maximum octree resolution. This should be greater than or equal to the ratio of the largest and smallest dimensions in the scene (ie. surface size or distance between surfaces). The default is 16384.

The *−f* option produces a frozen octree containing all the scene information. Normally, only a reference to the scene files is stored in the octree, and changes to those files may invalidate the result. The freeze option is useful when the octree file's integrity and loading speed is more important than its size, or when the octree is to be relocated to another directory, and is especially useful for creating library objects for the "instance" primitive type. If the input octree is frozen, the output will be also.

The *−w* option suppresses warnings.

A hyphen by itself ('-') tells *oconv* to read scene data from its standard input. This also implies the *−f* option.

The only scene file changes that do not require octree regeneration are modifications to non-surface parameters. If the coordinates of a surface are changed, or any primitives are added or deleted, oconv must be run again. Programs will abort with a "stale octree" message if they detect any dangerous inconsistencies between the octree and the input files.

Although the octree file format is binary, it is meant to be portable between machines. The only limitation is that machines with radically different integer sizes will not work together. For the best results, the −f option should be used if an octree is to be used in different environments.

## DIAGNOSTICS

There are four basic error types reported by oconv:

warning - a non-fatal input-related error

fatal - an unrecoverable input-related error

system - a system-related error

internal - a fatal error related to program limitations

consistency - a program-caused error

Most errors are self-explanatory. However, the following internal errors should be mentioned:

Too many scene files

Reduce the number of scene files by combining them or using calls to xform(1) within files to create a hierarchy.

Set overflow in addobject (id)

This error occurs when too many surfaces are close together in a scene. Either too many surfaces are lying right on top of each other, or the bounding cube is inflated from an oversized object or an improper *−b* specification. If hundreds of triangles come together at a common vertex, it may not be possible to create an octree from the object. This happens most often when inane CAD systems create spheres using a polar tessellation. Chances are, the surface "id" is near one of those causing the problem.

Hash table overflow in fullnode

This error is caused by too many surfaces. If it is possible to create an octree for the scene at all, it will have to be done in stages using the *−i* option.

## EXAMPLE

To add book1, book2 and a transformed book3 to the octree ‘‘scene.oct’’:

```
oconv –i scene.oct book1 book2 '\!xform –rz 30 book3' > newscene.oct
```

## AUTHOR

Greg Ward

## NOTES

In the octree, the names of the scene files are stored rather than the scene information. This means that a new octree must be generated whenever the scene files are changed or moved. Also, an octree that has been moved to a new directory will not be able to find scene files with relative pathnames. The freeze option avoids these problems. *make(1)* or *rad(1)* can be used to automate octree creation and maintenance.

## SEE ALSO

getbbox(1), getinfo(1), make(1), obj2mesh(1), rad(1), rpict(1), rvu(1), rtrace(1), xform(1)

## NAME

pcomb - combine RADIANCE pictures

## SYNOPSIS

**pcomb** [ **-h** ][ **-w** ][ **−x xres** ][ **−y yres** ][ **−f file** ][ **−e expr** ] [ [ **-o** ][ **−s factor** ][ **−c r g b** ] **input ..** ]

## DESCRIPTION

*Pcomb* combines equal-sized RADIANCE pictures and sends the result to the standard output. By default, the result is just a linear combination of the input pictures multiplied by *−s* and *−c* coefficients, but an arbitrary mapping can be assigned with the *−e* and *−f* options. Negative coefficients and functions are allowed, and *pcomb* will produce color values of zero where they would be negative.

The variables *ro, go* and *bo* specify the red, green and blue output values, respectively. Alternatively, the single variable *lo* can be used to specify a brightness value for black and white output. The predefined functions *ri(n), gi(n)* and *bi(n)* give the red, green and blue input values for picture *n*. To access a pixel that is nearby the current one, these functions also accept optional x and y offsets. For example, *ri(3,-2,1)* would return the red component of the pixel from picture 3 that is left 2 and up 1 from the current position. Although x offsets may be as large as width of the picture, y offsets are limited to a small window (+/- 32 pixels) due to efficiency considerations. However, it is not usually necessary to worry about this problem -- if the requested offset is not available, the next best pixel is returned instead.

For additional convenience, the function *li(n)* is defined as the input brightness for picture *n*. This function also accepts x and y offsets.

The constant *nfiles* gives the number of input files present, and *WE* gives the white efficacy (lumens/brightness) for pixel values, which may be used with the *−o* option or the le(n) values to convert to absolute photometric units (see below). The variables *x* and *y* give the current output pixel location for use in spatially dependent functions, the constants *xmax* and *ymax* give the input resolution, and the constants *xres* and *yres* give the output resolution (usually the same, but see below). The constant functions *re(n), ge(n), be(n),* and *le(n)* give the exposure values for picture *n,* and *pa(n)* gives the corresponding pixel aspect ratio. Finally, for pictures with stored view parameters, the functions *Ox(n), Oy(n)* and *Oz(n)* return the ray origin in world coordinates for the current pixel in picture *n,* and *Dx(n), Dy(n)* and *Dz(n)* return the normalized ray direction. In addition, the function *T(n)* returns the distance from the origin to the aft clipping plane (or zero if there is no aft plane), and the function *S(n)* returns the solid angle of the current pixel in steradians (always zero for parallel views). If the current pixel is outside the view region, *T(n)* will return a negative value, and *S(n)* will return zero.

The *−h* option may be used to reduce the information header size, which can grow disproportionately after multiple runs of *pcomb* and/or *pcompos(1)*. The *−w* option can be used to suppress warning messages about invalid calculations. The *−o* option indicates that original pixel values are to be used for the next picture, undoing any previous exposure changes or color correction.

The *−x* and *−y* options can be used to specify the desired output resolution, *xres* and *yres,* and can be expressions involving other constants such as *xmax* and *ymax*. The constants *xres* and *yres* may also be specified in a file or expression. The default output resolution is the same as the input resolution.

The *−x* and *−y* options must be present if there are no input files, when the definitions of *ro, go* and *bo* will be used to compute each output pixel. This is useful for producing simple test pictures for various purposes. (Theoretically, one could write a complete renderer using just the functional language...)

The standard input can be specified with a hyphen ('-'). A command that produces a RADIANCE picture can be given in place of a file by preceeding it with an exclamation point ('!').

## EXAMPLES

To produce a picture showing the difference between pic1 and pic2:

```
pcomb −e 'ro=ri(1)−ri(2);go=gi(1)−gi(2);bo=bi(1)−bi(2)' pic1 pic2 > diff
```

Or, more efficiently:

```
pcomb pic1 −s −1 pic2 > diff
```

To precompute the gamma correction for a picture:

```
pcomb -e 'ro=ri(1)^.4;go=gi(1)^.4;bo=bi(1)^.4' inp.hdr > gam.hdr
```

To perform some special filtering:

```
pcomb -f myfilt.cal -x xmax/2 -y ymax/2 input.hdr > filtered.hdr
```

To make a picture of a dot:

```
pcomb -x 100 -y 100 -e 'ro=b;go=b;bo=b;b=if((x-50)^2+(y-50)^2-25^2,0,1)' > dot
```

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), icalc(1), pcompos(1), pfilt(1), rpict(1)

## NAME

pcompos - composite RADIANCE pictures.

## SYNOPSIS

**pcompos** [ **−h** ][ **−x xres** ][ **−y yres** ][ **−b r g b** ][ **−lh h** ][ **−la** ] [ **−t min1** ][ **+t max1** ][ **−l lab** ][ **=SS** ] **pic1 x1 y1 ..**
or
**pcompos** [ **−a ncols** ][ **−s spacing** ][ **−o x0 y0** ][ options ] **pic1 pic2 ..**

## DESCRIPTION

*Pcompos* arranges and composites RADIANCE pictures and sends the result to the standard output. Each input picture must be accompanied by an anchor point (unless the *−a* option is used, see below). This anchor point is the usually position of the picture's left lower corner in the final output, but can be changed for individual pictures with an *=SS* option, where *S* is one of '-', '+' or '0', indicating the minimum, maximum or center of the image, respectively. (For example, =+- would indicate the anchor is relative to the right lower corner, and *=-0* would indicate the anchor is relative to the center of the left edge.) Negative anchor coordinates result in the input being cropped at the origin. By default, the size of the output picture will be just large enough to encompass all the input files. By specifying a smaller dimension using the *−x* and *−y* options, input files can be cropped at the upper boundary. Specifying a larger dimension produces a border. The *−b* option specifies a background color to appear wherever input files do not cover. The default value is black (0 0 0). The *−h* option may be used to reduce the information header size, which can grow disproportionately after multiple runs of *pcompos* and/or *pcomb(1)*.

If input files overlap, later pictures will overwrite earlier ones. By default, input files are copied unconditionally within the output boundaries. The *−t* option specifies a lower threshold intensity under which input pixels will not be copied to the output. The *+t* option specifies an upper threshold. These options are useful for cutting around irregular boundaries in the input.

The *−l* option can be used to specify a label for a specific picture, which will be given a height determined by the *−lh* option (default 24 pixels) and placed in the upper left corner of the picture. This label is generated by the program *psign(1)*. The *−la* option instructs *pcompos* to label each picture automatically by its name. This is particularly useful in conjunction with the *−a* option for producing a catalog of images (see example below). The *−l* option may still be used to override the default label for a picture.

The *−a* option can be used to automatically compute anchor points that place successive pictures next to each other in *ncols* columns. The ordering will place the first picture in the lower left corner, the next just to the right of it, and so on for *ncols* pictures. Then, the next row up repeats the pattern until all the input pictures have been added to the output. If the pictures are of different size, *pcompos* will end up leaving some background areas in the output picture. There will also be an unfinished row at the top if the number of pictures is not evenly divided by *ncols*. The *−s N* option will cause each image to be separated by at least N pixels. The *−o x0 y0* option specifies a nonzero anchor point for the bottom left image.

The standard input can be specified with a hyphen ('-'). A command that produces a RADIANCE picture can be given in place of a file by preceeding it with an exclamation point ('!').

## EXAMPLE

To put a copyright label at the bottom of a picture:

```
psign Copyright 1987 | pcompos inp.hdr 0 0 +t .5 - 384 64 > out.hdr
```

To make a catalog of images separated by white 10-pixel borders:

```
pcompos −la −a 4 −s 10 −b 1 1 1 dog*.hdr > alldogs.hdr
```

## NOTES

Since there is a limit to the number of open files and processes, large collections of images must be created in stages. Even if the system limit on open files is large, *pcompos* places an artificial limit of 1024 on the number of open files and/or processes.

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), pcomb(1), pfilt(1), psign(1), rpict(1)

## NAME

pcond - condition a RADIANCE picture for output

## SYNOPSIS

**pcond** [ **options** ] **input** [ **output** ]

## DESCRIPTION

*Pcond* conditions a Radiance picture for output to a display or hard copy device. If the dynamic range of the scene exceeds that of the display (as is usually the case), *pcond* will compress the dynamic range of the picture such that both dark and bright regions are visible. In addition, certain limitations in human vision may be mimicked in order to provide an appearance similar to the experience one might have in the actual scene.

Command line switches turn flags off and on, changing program behavior. A switch given by itself toggles the flag from off to on or on to off depending on its previous state. A switch followed by a '+' turns the option on explicitly. A switch followed by a '-' turns the option off. The default is all switches off. Other options specify output device parameters in order to get more accurate color and contrast.

**-h***[+-]* Mimic human visual response in the output. The goal of this process is to produce output that correlates strongly with a person's subjective impression of a scene. This switch is a bundle of the *−a, −v, −s* and *−c* options.

**-a***[+-]* Defocus darker regions of the image to simulate human visual acuity loss. This option will not affect well-lit scenes.

**-v***[+-]* Add veiling glare due to very bright regions in the image. This simulates internal scattering in the human eye, which results in a loss of visible contrast near bright sources.

**-s***[+-]* Use the human contrast sensitivity function in determining the exposure for the image. A darker scene will have relatively lower exposure with lower contrast than a well-lit scene.

**-c***[+-]* If parts of the image are in the mesopic or scotopic range where the cone photoreceptors lose their efficiency, this switch will cause a corresponding loss of color visibility in the output and a shift to a scotopic (blue-dominant) response function.

**-w***[+-]* Use a center-weighted average for the exposure rather than the default uniform average. This may improve the exposure for scenes with high or low peripheral brightness.

**-i** *fixfrac* Set the relative importance of fixation points to *fixfrac,* which is a value between 0 and 1. If *fixfrac* is zero (the default), then no fixation points are used in determining the local or global adaptation. If *fixfrac* is greater than zero, then a list of fixation points is read from the standard input. These points are given as tab-separated (x,y) picture coordinates, such as those produced by the *−op* option of *ximage(1)*. The foveal samples about these fixation points will then be weighted together with the global averaging scheme such that the fixations receive *fixfrac* of the total weight. If *fixfrac* is one, then only the fixation points are considered for adaptation.

**-I***[+-]* Rather than computing a histogram of foveal samples from the source picture, use the precomputed histogram provided on the standard input. This data should be given in pairs of the base-10 logarithm of world luminance and a count for each bin in ascending order, as computed by the *phisto(1)* script. This option is useful for producing identical exposures of multiple pictures (as in an animation), and provides greater control over the histogram computation.

**-l***[+-]* Use a linear response function rather than the standard dynamic range compression algorithm. This will prevent the loss of usable physical values in the output picture, although some parts of the resulting image may be too dark or too bright to see.

**-e** *expval* Set the exposure adjustment for the picture to *expval*. This may either be a real multiplier, or a (fractional) number of f-stops preceeded by a '+' or '-'. This option implies a linear response (see the *−l* option above).

**-u** *Ldmax* Specifies the top of the luminance range for the target output device. That is, the luminance (in candelas/m^2) for an output pixel value of (R,G,B)=(1,1,1). The default value is 100 cd/m^2.

**-d** *Lddyn* Specifies the dynamic range for the target output device, which is the ratio of the maximum and minimum usable display luminances. The default value is 32.

**-p** *xr yr xg yg xb yb xw yw*
Specifies the RGB primaries for the target output device. These are the 1931 CIE (x,y) chromaticity values for red, green, blue and white, respectively.

**-f** *macbeth.cal*
Use the given output file from *macbethcal(1)* to precorrect the color and contrast for the target output device. This does a more thorough job than a simple primary correction using the *–p* option. Only one of *–f* or *–p* may be given.

**-x** *mapfile* Put out the final mapping from world luminance to display luminance to *mapfile*. This file will contain values from the minimum usable world luminance to the maximum (in candelas/m^2) in one column, and their corresponding display luminance values (also in candelas/m^2) in the second column. This file may be used for debugging purposes, or to plot the mapping function created by *pcond*.

## EXAMPLES

To display an image as a person might perceive it in the actual scene:

```
pcond –h final.hdr > display.hdr
ximage display.hdr ; rm display.hdr &
```

To do the same on a 24–bit display with known primary values:

```
setenv DISPLAY_PRIMARIES ".580 .340 .281 .570 .153 .079 .333 .333"
pcond –h –p $DISPLAY_PRIMARIES final.hdr | ximage &
```

To prepare a picture to be sent to a film recorder destined eventually for a slide projector with a minimum and maximum screen luminance of 1.5 and 125 candelas/m^2, respectively:

```
pcond –d 83 –u 125 final.hdr > film.hdr
```

To do the same if the output colors of the standard image "ray/lib/lib/macbeth_spec.hdr" have been measured:

```
macbethcal –c mbfilm.xyY > film.cal
pcond –d 83 –u 125 –f film.cal final.hdr > film.hdr
```

To further tweak the exposure to bring out certain areas indicated by dragging the right mouse button over them in *ximage:*

```
ximage –op –t 75 final.hdr | pcond –i .5 –d 83 –u 125 –f film.cal final.hdr > film.hdr
```

To use a histogram computed on every 10th animation frame:

```
phisto frame*0.hdr > global.hist
pcond –I –s –c frame0352.hdr < global.hist | ra_tiff – frame0352.tif
```

## REFERENCE

Greg Ward Larson, Holly Rushmeier, Christine Piatko, “A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes,” *IEEE Transactions on Visualization and Computer Graphics* , December 1997.

http://www.sgi.com/Technology/pixformat/Larsonetal.html

## AUTHOR

Greg Ward Larson

## SEE ALSO

getinfo(1), macbethcal(1), normtiff(1), pcompos(1), pflip(1), phisto(1), pinterp(1), pvalue(1), protate(1), ra_xyze(1), rad(1), rpict(1), ximage(1)

## NAME

pdfblur - generate views for depth-of-field blurring

## SYNOPSIS

**pdfblur aperture nsamp viewfile**

## DESCRIPTION

*Pdfblur* takes the given *viewfile* and computes *nsamp* views based on an aperture diameter of *aperture* (in world coordinate units) and a focal distance equal to the length of the *−vd* view direction vector. When rendered and averaged together, these views will result in a picture with the specified depth of field. Either *pinterp(1)* or *rpict(1)* may be called to do the actual work. (The given *viewfile* must also be passed on the command line to the chosen renderer, since *pdfblur* provides supplemental view specifications only.)

For *pinterp,* feed the output of *pdfblur* to the standard input of *pinterp* and apply the *−B* option to blur views together. In most cases, a single picture with z-buffer is all that is required to get a satisfactory result, though the perfectionist may wish to apply three pictures arranged in a triangle about the aperature, or alternatively apply the *−ff* option together with the *−fr* option of *pinterp*. (The latter may actually work out to be faster, since rendering three views takes three times as long as a single view, and the *−fr* option will end up recomputing relatively few pixels by comparison.)

To use *pdfblur* with *rpict,* apply the *−S* option to indicate a rendering sequence, and set the *−o* option with a formatted file name to save multiple output pictures. When all the renderings are finished, combine them with the *pcomb(1)* program, using appropriate scalefactors to achieve an average. Note that using *rpict* is MUCH more expensive than using *pinterp,* and it is only recommended if the scene and application absolutely demand it (e.g. there is prominent refraction that must be modeled accurately).

For both *pinterp* and *rpict,* the computation time will be proportional to the number of views from *pdfblur.* We have found a *nsamp* setting somewhere between 5 and 10 to be adequate for most images. Relatively larger values are appropriate for larger aperatures.

The *−pd* option of *rpict* may be used instead or in combination with or instead of *pdfblur* to blur depth-of-field. If used in combination, it is best to set the *−pd* option to the overall *aperture* divided by *nsamp* to minimize ghosting in the output.

To simulate a particular camera's aperture, divide the focal length of the lens by the f-number, then convert to the corresponding world coordinate units. For example, if you wish to simulate a 50mm lens at f/2.0 in a scene modeled in meters, then you divide 50mm by 2.0 to get 25mm, which corresponds to an effective aperture of 0.025 meters.

## EXAMPLES

To use *pinterp* to simulate an aperture of 0.5 inches on a lens focused at a distance of 57 inches:

```
rpict −vf myview −x 640 −y 480 −z orig.zbf scene.oct > orig.hdr
pdfblur 0.5 57 8 orig.hdr | pinterp −B −vf orig.hdr −x 640 −y 480 orig.hdr orig.zbf > blurry.hdr
```

To use *rpict* exclusively to do the same:

```
pdfblur .5 57 5 myview | rpict −S 1 −vf myview −x 640 −y 480 −o view%d.hdr scene.oct
pcomb −s .2 view1.hdr −s .2 view2.hdr −s .2 view3.hdr −s .2 view4.hdr −s .2 view5.hdr > blurry.hdr
```

## AUTHOR

Greg Ward

## BUGS

This program really only works with perspective views.

## SEE ALSO

pcomb(1), pinterp(1), pmblur(1), pmblur2(1), pmdblur(1), rcalc(1), rpict(1), vwright(1)

## NAME

pexpand - expand requested commands in metafile

## SYNOPSIS

**pexpand** [ **+/−EPDOCSURIlrtmvsp** ] file ..

## DESCRIPTION

*Pexpand* reads each metafile *file* in sequence and expands any commands specified in a '+' option, and deletes any commands specified in a '-' option. This is necessary because most drivers will not support many metafile commands such as 'v' (vector string) and 'O', 'C', and 's' (segments). *Pexpand* will expand '+' instances into the corresponding primitives which are supported, and delete all '-' instances.

Certain commands are currently considered basic, and must be supported by all drivers. The commands 'D', 'E', 'S', 'U', 'R', 'l', 'r', and 't' are basic (see metafile(5)).

If no input files are specified, the standard input is read.

**+EPDOCSURIlrtmvsp**
: Expand the requested command(s).

**−EPDOCSURIlrtmvsp**
: Delete the requested command(s).

## EXAMPLE

To expand vector strings and segements, and delete pauses from "meta":

```
pexpand +vOCs −P meta
```

## FILES

/usr/lib/meta/vchars.mta (see metafile(5))

## AUTHOR

Greg Ward

## SEE ALSO

metafile(5), psort(1)

## NAME

pextrem - find minimum and maximum values in RADIANCE picture

## SYNOPSIS

**pextrem** [ **−o** ] [ picture ]

## DESCRIPTION

*Pextrem* locates the minimum and maximum values for the input *picture,* and prints their pixel locations and color values. The first line printed contains the x and y pixel location (x measured from the left margin, y measured from the bottom), followed by the red, green and blue values. The second line printed contains the same information for the maximum value.

The *−o* option prints the original (radiance) values, undoing any exposure or color correction done on the picture.

If no input *picture* is given, the standard input is read.

## AUTHOR

Greg Ward

## BUGS

The luminance value is used for comparison of pixels, although in certain anomolous cases (ie. highly saturated colors) it is possible that *pextrem* will not pick the absolute minimum or maximum luminance value. This is because a fast integer-space comparison is used. A more reliable floating-point comparison would be slower by an order of magnitude.

## SEE ALSO

falsecolor(1), getinfo(1), pcomb(1), pcompos(1), pextrem(1), pfilt(1), pflip(1), protate(1), psign(1), rpict(1), ximage(1)

## NAME

pfilt - filter a RADIANCE picture

## SYNOPSIS

**pfilt** [ **options** ] [ **file** ]

## DESCRIPTION

*Pfilt* performs anti-aliasing and scaling on a RADIANCE picture. The program makes two passes on the picture file in order to set the exposure to the correct average value. If no *file* is given, the standard input is read.

**-x** *res* Set the output x resolution to *res*. This must be less than or equal to the x dimension of the target device. If *res* is given as a slash followed by a real number, the input resolution is divided by this number to get the output resolution. By default, the output resolution is the same as the input.

**-y** *res* Set the output y resolution to *res,* similar to the specification of the x resolution above.

**-p** *rat* Set the pixel aspect ratio to *rat*. Either the x or the y resolution will be reduced so that the pixels have this ratio for the specified picture. If *rat* is zero, then the x and y resolutions will adhere to the given maxima. Zero is the default.

**-c** Pixel aspect ratio is being corrected, so do not write PIXASPECT variable to output file.

**-e** *exp* Adjust the exposure. If *exp* is preceded by a '+' or '-', the exposure is interpreted in f-stops (ie. the power of two). Otherwise, *exp* is interpreted as a straight multiplier. The individual primaries can be changed using *−er, −eg* and *−eb*. Multiple exposure options have a cumulative effect.

**-t** lamp Color-balance the image as if it were illuminated by fixtures of the given type. The specification must match a pattern listed in the lamp lookup table (see the −f option below).

**-f** lampdat Use the specified lamp lookup table rather than the default (lamp.tab).

**−1** Use only one pass on the file. This allows the exposure to be controlled absolutely, without any averaging. Note that a single pass is much quicker and should be used whenever the desired exposure is known and star patterns are not required.

**−2** Use two passes on the input. This is the default.

**−b** Use box filtering (default). Box filtering averages the input pixels corresponding to each separate output pixel.

**-r** *rad* Use Gaussian filtering with a radius of *rad* relative to the output pixel size. This option with a radius around 0.6 and a reduction in image width and height of 2 or 3 produces the highest quality pictures. A radius greater than 0.7 results in a defocused picture.

**-m** *frac* Limit the influence of any given input pixel to *frac* of any given output pixel. This option may be used to mitigate the problems associated with inadequate image sampling, at the expense of a slightly blurred image. The fraction given should not be less than the output picture dimensions over the input picture dimensions (x_o*y_o/x_i/y_i), or blurring will occur over the entire image. This option implies the *−r* option for Gaussian filtering, which defaults to a radius of 0.6.

**-h** *lvl* Set intensity considered "hot" to *lvl*. This is the level above which areas of the image will begin to exhibit star diffraction patterns (see below). The default is 100 watts/sr/m2.

**-n** *N* Set the number of points on star patterns to *N*. A value of zero turns star patterns off. The default is 0. (Note that two passes are required for star patterns.)

**-s** *val* Set the spread for star patterns to *val*. This is the value a star pattern will have at the edge of the image. The default is .0001.

**−a** Average hot spots as well. By default, the areas of the picture above the hot level are not used in setting the exposure.

## ENVIRONMENT

RAYPATH directories to search for lamp lookup table

## FILES

/tmp/rt??????

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), ies2rad(1), pcompos(1), pflip(1), pinterp(1), pvalue(1), protate(1), rad(1), rpict(1), ximage(1)

## NAME

pflip - flip a RADIANCE picture.

## SYNOPSIS

**pflip** [ **−h** ][ **−v** ][ **−c** ] **input** [ **output** ]

## DESCRIPTION

*Pflip* flips a RADIANCE picture horizontally and/or vertically. The *−h* option results in a horizontal exchange, and the *−v* option results in a vertical exchange. Both options may be applied.

The *−c* option indicates that the action is to correct an improper original image orientation, thus the recorded scanline ordering should not be changed.

If the *−v* option is not specified, the input may be piped from stdin.

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), pcompos(1), pfilt(1), protate(1), psign(1), rpict(1)

## NAME

phisto - compute a luminance histogram from one or more RADIANCE pictures

## SYNOPSIS

**phisto picture ..**

## DESCRIPTION

*Phisto* is a script that calls *pfilt(1), rcalc(1)* and *histo(1)* to compute a histogram of log luminance values for foveal samples in the given picture files. A foveal sample covers approximately 1 degree, though this script does not use this exact area. The minimum and maximum values are determined, and 100 histogram bins are uniformly divided between these extrema. Foveal samples less than 1e-7 candelas/sq.meter are silently ignored. If no picture is named on the command line, the standard input is read.

The primary function of this script is to precompute histograms for the *pcond(1)* program, which may then be used to compute multiple, identical exposures. This is especially useful for animations and image comparisons.

## EXAMPLE

To compute two identical tone mappings for image1.hdr and image2.hdr:

```
phisto image1.hdr image2.hdr > both.histo
pcond –I –h image1.hdr < both.histo > image1m.hdr
pcond –I –h image2.hdr < both.histo > image2m.hdr
```

## AUTHOR

Greg Ward Larson

## SEE ALSO

histo(1), pcond(1), pfilt(1), pvalue(1), rcalc(1), total(1)

## NAME

pinterp - interpolate/extrapolate view from pictures

## SYNOPSIS

**pinterp** [ view options ][ **−t threshold** ][ **−z zout** ][ **−f** *type* ][ **−B** ][ **−alq** ][ **−e exposure** ][ **−n** ] **pictfile zspec ..**

## DESCRIPTION

*Pinterp* interpolates or extrapolates a new view from one or more RADIANCE pictures and sends the result to the standard output. The input picture files must contain correct view specifications, as maintained by *rpict(1), rvu(1), pfilt(1)* and *pinterp*. Specifically, *pinterp* will not work on pictures processed by *pcompos(1)* or *pcomb(1)*. Each input file must be accompanied by a z specification, which gives the distance to each pixel in the image. If *zspec* is an existing file, it is assumed to contain a short floating point number for each pixel, written in scanline order. This file is usually generated by the *−z* option of *rpict(1)*. If *zspec* is a positive number rather than a file, it will be used as a constant value for the corresponding image. This may be useful for certain transformations on "flat" images or when the viewpoint remains constant.

The *−n* option specifies that input and output z distances are along the view direction, rather than absolute distances to intersection points. This option is usually appropriate with a constant z specification, and should not be used with *rpict(1)* z files.

The *−z* option writes out interpolated z values to the specified file. Normally, this information is thrown away.

*Pinterp* rearranges the pixels from the input pictures to produce a reasonable estimate of the desired view. Pixels that map within the *−t* threshold of each other (.02 times the z distance by default) are considered coincident. With the *−a* option, image points that coincide will be averaged together, giving a smooth result. The *−q* option turns averaging off, which means that the first mapped pixel for a given point will be used. This makes the program run faster and take less memory, but at the expense of image quality. By default, two or more pictures are averaged together, and a single picture is treated with the faster algorithm. This may be undesirable when a quick result is desired from multiple input pictures in the first case, or a single picture is being reduced in size (anti-aliased) in the second case.

Portions which were hidden or missing in the input pictures must be "filled in" somehow, and a number of methods are provided by the *−f* option. The default value for this option is *−fa,* which results in both foreground and background filling. The foreground fill algorithm spreads each input pixel to cover all output pixels within a parallelogram corresponding to that pixel's projection in the new view. Without it, each input pixel contributes to at most one output pixel. The background algorithm fills in those areas in the final picture that have not been filled with foreground pixels. It does this by looking at the boundary surrounding each blank area and picking the farthest pixels to each side, assuming that this will make a suitable background. The *−ff* option tells the program to use only the foreground fill, the *−fb* option says use only background fill, and the *−f0* option says not to use either fill algorithm.

Even when both fill algorithms are used, there may still be some unfilled pixels. By default, these pixels are painted black and assigned a z distance of zero. The *−fc* option can be used to change the color used for unfilled pixels, and the *−fz* option can be used to set the z distance (always along the view direction). Alternatively, the *−fr* option can be used to compute these pixels using *rtrace(1)*. The argument to this option is a quoted string containing arguments for *rtrace*. It must contain the octree used to generate the input pictures, along with any other options necessary to match the calculation used for the input pictures. The *−fs* option can be used to place a limit on the distance (in pixels) over which the background fill algorithm is used. The default value for this option is 0, which is interpreted as no limit. A value of 1 is equivalent to turning background fill off. When combined with the *−fr* option, this is roughly equivalent to the *−ps* option of *rpict(1)*.

In order of increasing quality and cost, one can use the *−fa* option alone, or the *−fr* option paired with *−fs* or *−ff* or *−f0*. The last combination will result in the recalculation of all pixels not adequately accounted for in the input pictures, with an associated computational expense. It is rare that the *−fs* option results in appreciable image degradation, so it is usually the second combination that is used when the background fill algorithm results in objectionable artifacts.

The *−B* option may be used to average multiple views read from the standard input into a single, blurred output picture. This is similar to running *pinterp* multiple times and averaging the output together with a program like *pcomb(1)*. This option is useful for simulating motion blur and depth of field. (See *pmdblur(1).)* The input views are reported in the information header of the output file, along with the averaged view. The picture dimensions computed from the first view will be the ones used, regardless whether or not the subsequent views agree. (The reported pixel aspect ratio in the output is determined from these original dimensions and the averaged view.) Note that the expense of the *−fr* option is proportional to the number of views computed, and the *−z* output file will be the z-buffer of the last view interpolated rather than an averaged distance map.

In general, *pinterp* performs well when the output view is flanked by two nearby input views, such as might occur in a walk-through animation sequence. The algorithms start to break down when there is a large difference between the view desired and the view(s) provided. Specifically, obscured objects may appear to have holes in them and large areas at the image borders may not be filled by the foreground or background algorithms. Also, specular reflections and highlights will not be interpolated very well, since their view-dependent appearance will be incompletely compensated for by the program. (The *−a* option offers some benefit in this area.)

The *−e* option may be used to adjust the output image exposure, with the same specification given as for *pfilt*. The actual adjustment will be rounded to the nearest integer f-stop if the *−q* option is in effect (or there is only a single input picture).

## EXAMPLE

To interpolate two frames of a walk-through animation, anti-alias to 512x400 and increase the exposure by 2.5 f-stops:

```
pinterp −vf 27.vf −a −x 512 −y 400 −e +2.5 30.hdr 30.z 20.hdr 20.z > 27.hdr
```

To extrapolate a second eyepoint for a stereo pair and recalculate background regions:

```
pinterp −vf right.vf −ff −fr "−av .1 .1 .1 scene.oct" left.hdr left.z > right.hdr
```

To convert an angular fisheye to a hemispherical fisheye:

```
pinterp −vf fish.hdr −vth -ff fish.hdr 1 > hemi.hdr
```

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), pdfblur(1), pfilt(1), pmblur(1), pmdblur(1), rpict(1), ranimate(1), rtrace(1), rvu(1)

## NAME

pkgBSDF - package BSDFs provided as XML for Radiance

## SYNOPSIS

**pkgBSDF** [ **−i** ][ **−s** ] **bsdf.xml ..**

## DESCRIPTION

*PkgBSDF* takes one or more XML files, each describing the bidirectional scattering distribution function of a surface or geometric system, and packages them into RADIANCE scene descriptions. If the input file(s) do not begin with a '.' or '/', the directories in the RAYPATH environment variable will be searched. One rectangle is produced for each BSDF that has no detail geometry and/or no rear-side reflectance distribution. If both front and back reflectance distributions are present with detail geometry, then two rectangles will be used to sandwich the geometry extracted from the BSDF file. The front surface will always be centered just in front of the origin facing the positive Z-direction, with "up" in the positive Y-direction. The back surface, if present, will parallel the front surface at a distance slightly greater than the BSDF system thickness. Scene units will always be in meters.

The *−i* option causes *pkgBSDF* to produce frozen octree instances for any detail geometry, which may reduce memory requirements in scenes with many references to the same complex BSDF object.

Normally, *pkgBSDF* produces one RADIANCE scene file for each XML input, replacing the required '.xml' suffix with '.rad' in the same directory. The *−s* option sends the RADIANCE scene description to the standard output rather than writing it to a file, but only works for a single XML input. Any instance octree is still written out to a file generated by replacing the '.xml' suffix with '.oct' in the same directory.

## EXAMPLE

Package all the XML files in the current directory as Radiance scene files, employing frozen octree instances for any detail geometry:

```
pkgBSDF -i *.xml
```

## AUTHOR

Greg Ward

## SEE ALSO

genBSDF(1), oconv(1), xform(1)

## NAME

plotin - convert plot(5) to metafile(5) primitives

## SYNOPSIS

**plotin** file ..

## DESCRIPTION

*Plotin* reads each plot(5) *file* in sequence and converts it to output suitable for use by the metafile filters. If no input files are specified, the standard input is read.

## EXAMPLE

To plot the graph example.grf to the imagen:

```
graph < example.grf | plotin | impress | ipr
```

## FILES

see pexpand(1)

## AUTHOR

Greg Ward

## SEE ALSO

graph(1G), metafile(5), plot(1), plot(5), plotout(1)

## NAME

pmblur - generate views for camera motion blurring

## SYNOPSIS

**pmblur speed nsamp v0file v1file**

## DESCRIPTION

*Pmblur* takes two viewfiles and generates *nsamp* views starting from *v0file* and moving towards *v1file*. When rendered and averaged together, these views will result in a picture with motion blur due to a camera changing from v0 to v1 in a relative time unit of 1, whose shutter is open starting at v0 for *speed* of these time units. Either *pinterp(1)* or *rpict(1)* may be called to do the actual work. (The given *v0file* must also be passed on the command line to the chosen renderer, since *pmblur* provides supplemental view specifications only.)

For *pinterp,* feed the output of *pmblur* to the standard input of *pinterp* and apply the *−B* option to blur views together. In most cases, two pictures with z-buffers at v0 and v1 will get a satisfactory result, though the perfectionist may wish to apply the *−ff* option together with the *−fr* option of *pinterp*.

To use *pmblur* with *rpict,* apply the *−S* option to indicate a rendering sequence, and set the *−o* option with a formatted file name to save multiple output pictures. When all the renderings are finished, combine them with the *pcomb(1)* program, using appropriate scalefactors to achieve an average. Note that using *rpict* is MUCH more expensive than using *pinterp,* and it is only recommended if the scene and application absolutely demand it (e.g. there is prominent refraction that must be modeled accurately).

For both *pinterp* and *rpict,* the computation time will be proportional to the number of views from *pmblur.* We have found a *nsamp* setting somewhere between 5 and 10 to be adequate for most images. Relatively larger values are appropriate for faster camera motion.

The *−pm* option of *rpict* may be used instead or in combination to blur animated frames, with the added advantage of blurring reflections and refractions according to their proper motion. However, this option will result in more noise and expense than using *pmblur* with *pinterp* as a post-process. If both blurring methods are used, a smaller value should be given to the *rpict −pm* option equal to the shutter speed divided by the number of pmblur views. This will be just enough to blur the boundaries of the ghosts which may appear using *pmblur* with a small number of time samples.

## EXAMPLES

To use *pinterp* to simulate motion blur between two frames of a walk-through animation, where the camera shutter is open for 1/4 of the interframe distance:

```
pmblur .25 8 fr1023.hdr fr1024.hdr | pinterp −B −vf fr1023.hdr −x 640 −y 480 fr1023.hdr fr1023.zbf
fr1024.hdr fr1024.zbf > fr1023b.hdr
```

## AUTHOR

Greg Ward

## BUGS

Changes in the view shift and lift vectors or the fore and aft clipping planes are not blurred.

## SEE ALSO

pcomb(1), pdfblur(1), pinterp(1), pmblur2(1), pmdblur(1), rcalc(1), rpict(1), vwright(1)

## NAME

pmblur2 - apply motion blur to a frame from an animation

## SYNOPSIS

**pmblur2 f0,f1/n HDRspec ZBFspec MVOspec**

## DESCRIPTION

*Pmblur2* Computes a square-shutter blur from frame position *f0* to frame position *f1* using the given picture, depth buffer, and motion vector offset files produced by *ranimove(1)*. These arguments should be set similarly to the *BASENAME, ZNAME,* and *MNAME* variables in the *ranimove* input file, but should include the suffixes as well.

The starting and ending frame number are usually given as floating-point values. For example, the range "10.75,11.25" would simulate a 180-degree shutter around frame 11. An optional time slice count may be given to override the one that is computed by default. The range "5,15/30" would blur all the frames from 5 to 15 continuously using 30 time slices. The blurred image is always sent to stdout.

## EXAMPLES

To blur and downsample frame 95 with a 360-degree shutter:

```
pmblur2 94.5,95.5 frame%03d.hdr frame%03d.zbf frame%03d.mvo | pfilt -1 -x /2 -y /2 > final95.hdr
```

## AUTHOR

Greg Ward

## SEE ALSO

pdfblur(1), pfilt(1), pinterp(1), pmblur(1), pmdblur(1), ranimove(1)

## NAME

pmdblur - generate views for combined camera motion and depth blurring

## SYNOPSIS

**pmdblur speed aperture nsamp v0file v1file**

## DESCRIPTION

*Pmdblur* takes two viewfiles and generates *nsamp* views starting from *v0file* and moving towards *v1file,* simulating an aperture of diameter *aperture* in world coordinate units. When rendered and averaged together, these views will result in a picture with motion and depth-of-field blur due to a camera changing from v0 to v1 in a relative time unit of 1, whose shutter is open starting at v0 for *speed* of these time units. Either *pinterp(1)* or *rpict(1)* may be called to do the actual work. (The given *v0file* must also be passed on the command line to the chosen renderer, since *pmdblur* provides supplemental view specifications only.)

For *pinterp,* feed the output of *pmdblur* to the standard input of *pinterp* and apply the *−B* option to blur views together. In most cases, two pictures with z-buffers at v0 and v1 will get a satisfactory result, though the perfectionist may wish to apply the *−ff* option together with the *−fr* option of *pinterp*.

To use *pmdblur* with *rpict,* apply the *−S* option to indicate a rendering sequence, and set the *−o* option with a formatted file name to save multiple output pictures. When all the renderings are finished, combine them with the *pcomb(1)* program, using appropriate scalefactors to achieve an average. Note that using *rpict* is MUCH more expensive than using *pinterp,* and it is only recommended if the scene and application absolutely demand it (e.g. there is prominent refraction that must be modeled accurately).

For both *pinterp* and *rpict,* the computation time will be proportional to the number of views from *pmdblur.* We have found a *nsamp* setting somewhere between 7 and 15 to be adequate for most images. Relatively larger values are appropriate for faster camera motion.

The *−pm* and/or *−pd* options of *rpict* may be used instead or in combination to blur animated frames, with the added advantage of blurring reflections and refractions according to their proper motion. However, this option will result in more noise and expense than using *pmdblur* with *pinterp* as a post-process. If both blurring methods are used, a smaller value should be given to the *rpict −pm* option equal to the shutter speed divided by the number of samples, and the *−pd* option equal to the aperture divided by the number of samples. This will be just enough to blur the boundaries of the ghosts which may appear using *pmdblur* with a small number of time samples.

To simulate a particular camera's aperture, divide the focal length of the lens by the f-number, then convert to the corresponding world coordinate units. For example, if you wish to simulate a 50mm lens at f/2.0 in a scene modeled in meters, then you divide 50mm by 2.0 to get 25mm, which corresponds to an effective aperture of 0.025 meters.

## EXAMPLES

To use *pinterp* to simulate motion blur between two frames of a walk-through animation, where the camera shutter is open for 1/4 of the interframe distance with an aperture of 0.1 world units:

```
pmdblur .25 .1 8 fr1023.hdr fr1024.hdr | pinterp −B −vf fr1023.hdr −x 640 −y 480 fr1023.hdr
fr1023.zbf fr1024.hdr fr1024.zbf > fr1023b.hdr
```

## AUTHOR

Greg Ward

## SEE ALSO

pcomb(1), pdfblur(1), pinterp(1), pmblur(1), pmblur2(1), rcalc(1), rpict(1), vwright(1)

## NAME

protate - rotate a RADIANCE picture.

## SYNOPSIS

**protate** [ **−c** ][ **−r** ] **input** [ **output** ]

## DESCRIPTION

*Protate* rotates a RADIANCE picture 90 degrees. This is useful for output on hardcopy devices with aspect ratios opposite to the input picture. By default, the image is rotated clockwise. The *−r* option may be used to rotate the image counter-clockwise instead.

The *−c* option indicates that the action is to correct an improper original image orientation, thus the recorded scanline ordering should not be changed.

## NOTES

To rotate an image 180 degrees, use *pflip(1)* with both the *-h* and *-v* options.

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), pcompos(1), pfilt(1), pflip(1), psign(1), rpict(1)

## NAME

psign - produce a RADIANCE picture from text.

## SYNOPSIS

**psign** [ **options** ] [ **text** ]

## DESCRIPTION

*Psign* produces a RADIANCE picture of the given *text*. The output dimensions are determined by the character height, aspect ratio, number of lines and line length. (Also the character size if text squeezing is used.) If no *text* is given, the standard input is read.

**-cb** *r g b* Set the background color to *r g b* The default is white (1 1 1).

**-cf** *r g b* Set the foreground color to *r g b* The default is black (0 0 0).

**–dr** Text reads to the right (default).

**–du** Text reads upwards.

**–dl** Text reads to the left (upside down).

**–dd** Text reads downwards.

**-h** *cheight* Set the character height to *cheight*. The default is 32 pixels.

**-a** *aspect* Set the character aspect ratio (height/width) to *aspect*. The default value is 1.67.

**-x** *xsize* Set the horizontal image size to *xsize*. Use with *–y* option (below) in place of the *–h* specification to control output image size directly. If the character aspect ratio *(–a* option, above) is non-zero, then one of the specified x or y output dimensions may be reduced to maintain this ratio. If direction is right *(–dr)* or left *(–dl),* then it is not necessary to give the *–y* option, since it can be computed from the character height *(–h)*.

**-y** *ysize* Set the vertical image size to *ysize*. Use with the *–x* option (described above). If direction is up *(–du)* or down *(–dd),* then it is not necessary to give the *–x* option, since it can be computed from the character height *(–h)*.

**-s** *spacing* Set the intercharacter spacing to *spacing*. The magnitude of this value is multiplied by the character height over the aspect ratio (ie. the character width) to compute the desired distance between characters in the output. The sign of the value, positive or negative, determines how this ideal spacing is used in the actual placement of characters. If *spacing* is positive, then the overall width of the line will not be affected, nor will indentation of textual elements. Thus, the text format will be mostly unaffected. However, spacing between characters will reflect their relative size for a more natural appearance. If *spacing* is negative, characters will be squeezed together to meet the spacing critereon, regardless of how it might affect the format of the output. The default value for *spacing* is zero, which is interpreted as uniformly spaced characters.

**-f** *fontfile* Load the font from *fontfile*. The default font is helvet.fnt

## EXAMPLE

To put a big "Hi!" on the terminal:

```
psign –h 22 –a 1 –cb 0 0 0 –cf 1 1 1 Hi\! | ttyimage
```

## ENVIRONMENT

RAYPATH path to search for font files

## AUTHOR

Greg Ward

## BUGS

The entire bitmap is stored in memory, which can be a problem for large and/or high-resolution signs.

## SEE ALSO

getinfo(1), pcompos(1), pfilt(1), ttyimage(1)

## NAME

psmeta - convert metafile to PostScript

## SYNOPSIS

**psmeta** file ..

## DESCRIPTION

*Psmeta* reads each metafile *file* in sequence and converts it to PostScript output suitable for a standard letter-size page. The file produced may also be read into programs that can handle Encapsulated PostScript. If no input files are specified, the standard input is read.

## EXAMPLE

To print the plot file example.plt to the ap5 printer:

```
bgraph example.plt | psmeta | lpr –P ap5
```

## AUTHOR

Greg Ward

## SEE ALSO

bgraph(1), igraph(1), lpr(1), meta2bmp(1), meta2tga(1), x11meta(1)

## NAME

psort - sort primitives in metafile as requested

## SYNOPSIS

**psort** [ **+/−x** ][ **+/−y** ][ **+/−X** ][ **+/−Y** ] file ..

## DESCRIPTION

*Psort* reads each metafile *file* in sequence and sorts primitives between globals according to the option specification. Lower case options mean the corresponding minimum, upper case indicates the maximum value. A '+' before the option means sort in order of increasing values, '-' means decreasing. The order the options appear on the command line is the order in which the extrema are examined. For example, the options $-Y\ +x$ would mean "sort on decreasing ymax, then increasing xmin values".

If no input files are specified, the standard input is read.

## EXAMPLE

To sort the file "meta" in order of increasing xmax, then decreasing ymin:

```
psort +X −y meta
```

## FILES

/tmp/psXXXXa /tmp/psXXXXb

## BUGS

Aborting the program will sometimes leave files in /tmp.

## AUTHOR

Greg Ward

## SEE ALSO

metafile(5), pexpand(1)

## NAME

pvalue - convert RADIANCE picture to/from alternate formats

## SYNOPSIS

**pvalue** [ **options** ] [ **file** ]
**pvalue -r** [ **options** ] [ **file1** [ **file2 file3** ] ]

## DESCRIPTION

*Pvalue* converts the pixels of a RADIANCE picture to or from another format. In the default mode, pixels are sent to the standard output, one per line, in the following ascii format:

xpos ypos red green blue

If no *file* is given, the standard input is read.

The reverse conversion option *(-r)* may be used with a single input file or when reading from the standard input, but if the second form is used with three separate input files, the three primaries are presumed to be separated in these files.

**−u** Print only unique values in the output, skipping runs of equal pixels. Specifying *+u* turns this option off, which is the default.

**−o** Print original values, before exposure compensation. If the input file is in XYZE format, the Y (green) channel will correspond to units of candelas/meter^2. Otherwise, the RGB values should be in spectral watts/steradian/meter^2. Specifying *+o* uses final exposed values, which is the default.

**−h** Do not print header. Specifying *+h* causes the header to be printed, which is the default.

**−H** Do not print the resolution string. (See also the *−r* option below.) Specifying an input resolution for reverse conversion also turns this option off. Specifying *+H* causes the resolution string to be printed, which is the default.

**−s** *nbytes* Skip the specified number of bytes on the input header. This option is useful for skipping unintelligible headers in foreign file formats. (Does not work when reading from standard input.)

**−e** *exposure*
Adjust the exposure by the amount specified. If the exposure is being given as a conversion factor, use *+e* instead, so an EXPOSURE line will not be recorded in the header (if any).

**−g** *gamma* Set gamma correction for conversion. When converting from a RADIANCE picture to another format, the inverse gamma is applied to correct for monitor response. When converting to a RADIANCE picture *(−r* option), the gamma is applied directly to recover the linear values. By default, *gamma* is set to 1.0, meaning no gamma correction is performed.

**−d** Data only, do not print x and y pixel position.

**−da** Same as *−d*.

**−di** Print ascii integer values from 0 to 255+. If *+di* is given, the integer values will be preceded by the x and y pixel locations.

**−db** Output binary byte values from 0 to 255.

**−dw** Output binary 16-bit words from 0 to 65535.

**−dW** Output binary 16-bit words from 0 to 65535, byte-swapped.

**−df** Output binary float values.

**−dF** Output byte-swapped binary float values.

**−dd** Output binary double values.

**−dD** Output byte-swapped binary double values.

**−R** Reverse ordering of colors so that the output is blue then green then red. The default ordering (specified with *+R)* is red then green then blue.

**−n** The RGB values are non-interleaved, meaning that all the red, green and blue data are stored together in separate chunks. Interleaving may be turned on with the *+n* option, which is the default.

**−b** Print brightness values rather than RGB. Specifying *+b* turns this option off, which is the default.

**−p***P* Put out only the primary *P,* where *P* is one of upper or lower case 'R', 'G' or 'B' for red, green or blue, respectively. This option may be used to separate the Radiance primaries into three files with three separate runs of *pvalue,* or only one file when only one primary is needed. Note that there is no space between this option and its argument.

**−r** Perform reverse conversion. Input is in the format given by the other options. The x and y resolution must be specified on the command line, unless the image file contains a Radiance resolution string at the beginning (see *−H* option above and *−y* option below). Specifying *+r* converts from a Radiance picture to other values, which is the default.

**−p** *xr yr xg yg xb yb xw yw*
On reverse conversion, RGB input uses the given set of color primaries. These are written into the output header with the PRIMARIES variable.

**−pXYZ** On reverse conversion, input is in CIE XYZ coordinates, rather than RGB. The Y channel is assumed to be in candelas/meter^2.

**-y** *res* Set the output y resolution to *res*. If *+y* is specified, then the scanlines are assumed to be in increasing order (ie. bottom to top). The default value for this option is 0, which means that the picture size and scanline order must appear as the first line after the header (if any) in the input file. Either an upper or lower case 'Y' may be used for this option. Since Radiance files always contain such a line, this option has no effect for forward conversions.

**+x** *res* Set the output x resolution to *res*. If *−x* is specified, then the scanlines are assumed to be in decreasing order (ie. right to left). The ordering of the *−y* and *+x* options determines whether the scanlines are sorted along x or along y. Most Radiance pictures are sorted top to bottom, then left to right. This corresponds to a specification of the form "−y yres +x xres". Either an upper or lower case 'X' may be used for this option. Like the *−y* option, *−x* options have no effect for forward conversions.

## EXAMPLE

To look at the original, unique pixels in picture:

```
pvalue −o −u picture | more
```

To convert from a 512x400 8-bit greyscale image in bottom to top, left to right scanline ordering:

```
pvalue −r −db −b −h +y 400 +x 512 input.im > flipped.hdr
pflip −v flipped.hdr > final.hdr
```

## AUTHOR

Greg Ward

## BUGS

The *−r* option does not work with the *−u* option. Also, input pixel locations are ignored during a reverse conversion, so this information is not used in determining the scanline ordering or picture size.

## SEE ALSO

getinfo(1), pcompos(1), pfilt(1), pflip(1), protate(1), ra_xyze(1), rpict(1), rtrace(1), rvu(1)

## NAME

ra_bmp - convert RADIANCE picture to/from Windows BMP image

## SYNOPSIS

**ra_bmp** [ **−b** ][ **−g gamma** ][ **−e spec** ][ **-p xr yr xg yg xb yb xw yw** ] [ [ **input|-** [ **output** ] ]
**ra_bmp −r** [ **−g gamma** ][ **−e +/-stops** ] [ [ **input|-** [ **output** ] ]

## DESCRIPTION

*Ra_bmp* converts between RADIANCE and a Windows BMP image. During forward conversion, the *−b* option produces 8-bit grayscale output rather than 24-bit RGB. The *−g* option specifies the exponent used in monitor gamma correction; the default value is 2.2. The *−e* option specifies a tone-mapping method or exposure compensation. If the argument is one of the special words "auto", "human", or "linear", the image will be tone-mapped with histogram adjustment, human vision simulation, or an averaged linear exposure, respectively. (These words may be abbreviated by one or more letters.) Otherwise, the exposure specification is interpreted as a linear compensation value in integer f-stops, which must be proceeded by a '+' or '-'. The CRT color output primaries may be specified with the *−p* option.

The *−r* option invokes a reverse conversion, from a Windows BMP image to a RADIANCE picture. Tone-mapping and monitor primaries are not supported for reverse conversion.

*Ra_bmp* can accept 16-bit or color-mapped BMP files on input, but cannot not produce them on output.

A hyphen ('-') may be specified to indicate standard input for either forward or reverse conversion so that an output file may be specified. If the BMP input is taken from the standard input or sent to the standard output without tone-mapping, then the scanlines may be reversed from their usual ordering. Reversed scanlines sometimes causes difficulties for programs attempting to interpret the resulting RADIANCE picture or BMP image, which may be rejected or displayed inverted.

## AUTHOR

Greg Ward

## SEE ALSO

pfilt(1), ra_ppm(1), ra_pr(1), ra_pr24(1), ra_t8(1), ra_t16(1), ra_tiff(1), ximage(1)

## NAME

ra_gif - convert RADIANCE picture to Compuserve GIF

## SYNOPSIS

**ra_gif** [ **−b** ][ **−d** ][ **−c ncolors** ][ **−g gamma** ][ **−e +/-stops** ][ **-n sampfac** ] **input** [ **output** ]

## DESCRIPTION

*Ra_gif* converts from RADIANCE to Compuserve GIF color-mapped, compressed image files. In the default mode, a RADIANCE picture is converted to a color-mapped GIF file of the same horizontal and vertical dimensions with 8-bits per pixel. The *−b* option converts the image to black and white. The *−d* option turns off dithering. The *−c* option allows fewer than 256 colors (and fewer than 8 bits per pixel). The *−g* option specifies the exponent used in gamma correction; the default value is 2.2. An exponent of 1.0 turns gamma correction off. The *−e* option specifies an exposure compensation in f-stops (powers of two). Only integer stops are allowed, for efficiency. The *−n* option specifies a sampling factor for neural network color quantization. This value should be between 1 and 80, where 1 takes the longest and produces the best results in small areas of the image. If no value is given, a faster median cut algorithm is used. If the output file is missing, the standard output is used.

## AUTHORS

Greg Ward
Paul Haeberli
David Rowley
Anthony Dekker provided the code for neural network color quantization

## SEE ALSO

pfilt(1), ra_bmp(1), ra_bn(1), ra_ppm(1), ra_pr(1), ra_pr24(1), ra_t8(1), ra_t16(1), ra_tiff(1), ximage(1)

## NAME

ra_pict - convert Radiance pictures to Macintosh PICT files

## SYNOPSIS

**ra_pict** [ **–e +/-** *stops* ] [ **–v** ] [ **–g** *gamma* ] [ *infile* [ *outfile* ]]

## DESCRIPTION

**Ra_pict** converts a Radiance picture, as produced by **rpict** (1) to a Macintosh PICT file. The picture will be a 24 bit PICT 2 picture, using a single bit map (DirectRect).

## OPTIONS

**–g** *gamma* This sets an explict gamma correction for the image. If it is not specified, the default value of 2.0 is used.

**–v** Invokes verbose mode, and gives line on standard error giving the size of the picture and the gamma correction used.

**–e +/–** *stops*
Adjusts the exposure by stops.

*infile* Specifies the file to read the picture from. If none is specified, it takes it from standard input. If standard input is used, then the picture is sent to standard output.

*outfile* Specifies the file to send the PICT file to. If none is specified, it is sent to standard output.

## EXAMPLES

```
ra_pict mypict.hdr mypict.pict
```

Will convert the Radiance picture mypict.hdr, giving the Macintosh PICT mypict.pict.

```
ra_pict –g 2.2 mypict.hdr mypict.pict
```

Will convert the file using a gamma of 2.2.

## SEE ALSO

rpict(1)

## BUGS

Does not yet do Macintosh PICT to Radiance PIC, as this PICT files are a lot more complex than just one bit map. This is left as an exercise for the reader :-)

## AUTHOR

Russell Street

## NAME

ra_ppm - convert RADIANCE picture to/from a Poskanzer Portable Pixmap

## SYNOPSIS

**ra_ppm** [ **−r** ][ **−a** ][ **−b** ][ **−s maxv** ][ **−g gamma** ][ **−e +/-stops** ] [ **input** [ **output** ] ]

## DESCRIPTION

*Ra_ppm* converts between RADIANCE and Poskanzer Portable Pixmap formats. The *−g* option specifies the exponent used in gamma correction; the default value is 2.2. An exponent of 1 turns gamma correction off. The *−e* option specifies an exposure compensation in f-stops (powers of two). Only integer stops are allowed, for efficiency. The *−r* option invokes a reverse conversion, from a Pixmap to a RADIANCE picture. If the output file is missing, the standard output is used. If the input file is missing as well, the standard input is used.

The *−a* option produces a standard ASCII Pixmap representation instead of the default binary file. The file is much larger and the conversion is much slower, which is why this format is not normally used. The *−b* option forces greyscale output. The *−s* option controls the output scale, which is 255 by default. If this value is set above 255, then two bytes will be output for each component in binary mode. This may not be understood by some PPM readers, which do not understand files with maximum values greater than 255. The maximum allowed setting for this parameter is 65535.

With the *−r* option, the type of the Pixmap input file is determined automatically. *Ra_ppm* will read either greyscale or color Pixmaps, with any precision up to a maximum scale of 65535.

## NOTES

The Poskanzer Portable Bitmap Plus package contains translators between the Pixmap format and many of the dozen or so image file "standards" that exist. At the time of this writing, the software is free and available by anonymous ftp from export.lcs.mit.edu (18.30.0.238) in the file "contrib/pbmplus.tar.Z".

## AUTHOR

Greg Ward

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland and Silicon Graphics, Inc.

## SEE ALSO

pfilt(1), ra_bmp(1), ra_bn(1), ra_pr(1), ra_pr24(1), ra_t8(1), ra_t16(1), ra_tiff(1), ximage(1)

## NAME

ra_ps - convert RADIANCE picture to a PostScript file

## SYNOPSIS

**ra_ps** [ **−b|c** ][ **−A|B|C** ][ **−n ncopies** ][ **−e +/-stops** ][ **−g gamma** ][ **−p paper** ][ **−m[h|v] margin** ][ **−d dpi** ] [ **input** [ **output** ] ]

## DESCRIPTION

*Ra_ps* translates a RADIANCE picture to a color or greyscale Adobe PostScript file for printing on a laser printer or importing to a page layout program. The *−b* option tells *ra_ps* to produce greyscale output. (The default is color, which may be specified explicitly with the *−c* option.)

The *−A* option specifies that the output should be in uncompressed ASCII hexstring format (the default). The *−B* option specifies that the output should be in uncompressed binary string format. The file size will be roughly half that of the ASCII equivalent, but some printers and especially some printer connections do not support binary transfer, so this option should be used with caution. The *−C* option specifies that the output should be in run-length compressed ASCII format. The file size will be one half to one tenth as large as the hexstring equivalent and can be sent over any network or by e-mail. The only disadvantage is that it will actually take longer to print on some printers, since the "readhexstring" procedure is generally faster than a custom replacement.

The *−n* option specifies the number of copies to print of this image. It is often preferable to use this option instead of the multiple copy option of the print spooler program, since the latter often results in duplication of the input file with a large associated cost.

The *−e* option specifies an exposure compensation in f-stops (powers of two). Only integer stops are allowed, for efficiency. The *−g* option specifies a power law for the printer transfer function. The default gamma setting for greyscale printers is 1.0 (linear), and the default gamma for color printers is 1.8 (commonly used in prepress). If your output seems to have too much contrast relative to its screen equivalent, print out the file "ray/lib/lib/gamma.hdr" to your printer without any gamma correction and using the *−d* option to set the dots-per-inch (see below). The best match between the small lines and the grey patch next to it indicate the approximate gamma of your printer, which you should use with the *−g* option for best contrast reproduction in subsequent conversions.

The standard print area assumes 8.5 by 11 inch (U.S. letter) paper, with 0.5 inch margins on all sides. The image will be rotated 90 degrees if it fits better that way in the available print area, and it will always be centered on the page. The *−p* and *−m* options to control the paper size and margins, respectively. The argument to the *−p* option is the common name for a given paper size, or WWxHH, where WW is the width (in inches) and HH is the height. If millimeters or centimeters are the preferred measurement unit, the 'x' may be replaced by 'm' or 'c', respectively. The WW and HH values are decimal quantities, of course. The current paper identifiers understood by the program may be discovered by giving a 0 argument to the *−p* option. They are currently:

```
_Name________Width_Height_(inches)
envelope     4.12   9.50
executive    7.25  10.50
letter       8.50  11.00
lettersmall  7.68  10.16
legal        8.50  14.00
monarch      3.87   7.50
statement    5.50   8.50
tabloid     11.00  17.00
A3          11.69  16.54
A4           8.27  11.69
A4small      7.47  10.85
A5           6.00   8.27
A6           4.13   6.00
B4          10.12  14.33
B5           7.17  10.12
```

| | | |
|---|---|---|
| C5 | 6.38 | 9.01 |
| C6 | 4.49 | 6.38 |
| DL | 4.33 | 8.66 |
| hagaki | 3.94 | 5.83 |

The paper size name may be abbreviated with three or more letters, and character case is ignored. The argument to the *−m* option is the margin width, which is 0.5 inches by default. A millimeter or centimeter quantity may be given instead of inches by immediately following the value with a 'm' or 'c' character, respectively. (Leave no space between the quantity and its unit letter.) If you wish to specify the horizontal and vertical margins separately, use the *−mh* and *−mv* options, instead.

The *−d* option may be used to explicitly set the print density (in dots per inch). If the input picture is lower resolution than the printer and has square pixels, then *ra_ps* will adjust the image size so that pixels map to dot regions exactly. This may improve the appearance of fine detail, and may speed up the printing process as well, at the expense of a slightly smaller image area. If you wish to maximize print area and the input image contains no fine detail, then do not specify this option.

The output from *ra_ps* is designed to be compatible with the Encapsulated PostScript standard, which means that the resulting file may be incorporated into documents by page layout programs that can read in EPS files. Unfortunately, there is currently no option for generating a preview bitmap, so the image will show up on the screen as a rectangular area only. To control the EPS image size directly, use the *−p* option as explained above with the WWxHH specification, and set *−m 0* to turn off the margins.

## AUTHOR

Greg Ward

## SEE ALSO

pfilt(1), ra_bmp(1), ra_bn(1), ra_pr(1), ra_pr24(1), ra_t8(1), ra_t16(1), ra_ppm(1), ra_tiff(1), ximage(1)

## NAME

ra_rgbe - convert between different RADIANCE picture types

## SYNOPSIS

**ra_rgbe** [ **−r** ][ **−e +/-stops** ][ **−f** ][ **−n frameno** ] [ **input** [ **outspec** ] ]

## DESCRIPTION

*Ra_rgbe* converts between RADIANCE run-length encoded and flat formats, and separates concatanated animation frames produced by *rpict(1)*. The *−e* option specifies an exposure compensation in f-stops (powers of two). Only integer stops are allowed, for efficiency. By default, *ra_rgbe* produces a flat RADIANCE picture file from any type of RADIANCE input picture. The *−r* option causes *ra_rgbe* to produce a run-length encoded file instead.

If the input file consists of multiple animation frames as produced by *rpict* with the *−S* option, *ra_rgbe* will read each frame and write it to the output file created by calling *printf(3)* with the output specification and frame number as arguments. If the output specification begins with an exclamation mark ('!'), then this interpreted as a command spec., which is also run through *printf* with the frame number to get the final command. This command must accept a Radiance picture on its standard input, and may write results either to a file or to the standard output. The *−n* option may be used to select a specific frame for output, and other frames in the input will be skipped. Normally, all frames will be read and written.

*Ra_rgbe* will report an error and exit if the target output file already exists, unless the *−f* option is given. If the output file is missing, the standard output is used. If the input file is missing or set to '-', the standard input is used.

## NOTES

The file format for RADIANCE pictures was changed between release 1.4 and release 2.0. The older format can still be read by all the programs, but only the newer format is produced. This newer format cannot be read by RADIANCE software prior to release 2.0.

*Ra_rgbe* provides some downward compatibility by producing files that can be read by older RADIANCE software. The resultant files are also easier to manipulate with programs designed to read raw raster data.

The other use for *ra_rgbe* is as a quicker way to adjust the exposure of a RADIANCE picture than *pfilt(1),* since *ra_rgbe* only allows integer f-stop changes. In this mode, *ra_rgbe* should be used with the *−r* option.

## AUTHOR

Greg Ward

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland.

## SEE ALSO

pfilt(1), printf(1), ra_xyze(1), rpict(1)

## NAME

ra_t16 - convert RADIANCE picture to/from Targa 16 or 24-bit image file

## SYNOPSIS

**ra_t16** [ **−2** ][ **−3** ][ **−g gamma** ][ **−e +/-stops** ] [ **input** [ **output** ] ]
**ra_t16 −r** [ **−g gamma** ][ **−e +/-stops** ] [ **input** [ **output** ] ]

## DESCRIPTION

*Ra_t16* converts between RADIANCE and Targa 16-bit or 24-bit RGB image files (type 2 in Targa's documentation). In the default mode, a RADIANCE picture is converted to an RGB file of the same horizontal and vertical dimensions with 16-bits per pixel. The *−3* option tells the program to produce a 24-bit image file instead. The *−g* option specifies the exponent used in gamma correction; the default value is 2.2. An exponent of 1.0 turns gamma correction off. The *−e* option specifies an exposure compensation in f-stops (powers of two). Only integer stops are allowed, for efficiency. The *−r* option invokes a reverse conversion, from a 16-bit or 24-bit Targa file to a RADIANCE picture. The determination of depth is made automatically on reverse translation, so the *−2* and *−3* options are not necessary. If the output file is missing, the standard output is used.

## AUTHORS

Greg Ward

## BUGS

Run-length encoded files can be read but not written with this program.

## SEE ALSO

pfilt(1), ra_bmp(1), ra_bn(1), ra_ppm(1), ra_pr(1), ra_pr24(1), ra_t8(1), ra_tiff(1), ximage(1)

## NAME

ra_t8 - convert RADIANCE picture to/from Targa 8-bit image file

## SYNOPSIS

**ra_t8** [ **−d** ][ **−b** ][ **−c ncolors** ][ **−g gamma** ][ **−e +/-stops** ][ **-n sampfac** ] **input** [ **output** ]
**ra_t8 −r** [ **−g gamma** ][ **−e +/-stops** ] [ **input** [ **output** ] ]

## DESCRIPTION

*Ra_t8* converts between RADIANCE and Targa 8-bit color-mapped image files (type 1 in Targa's documentation). In the default mode, a RADIANCE picture is converted to a color-mapped Targa file of the same horizontal and vertical dimensions with 8-bits per pixel. The *−d* option turns off dithering. The *−b* option converts the image to black and white. Only with this option can the input be taken from stdin. The *−c* option allows fewer than 256 colors. The *−g* option specifies the exponent used in gamma correction; the default value is 2.2. An exponent of 1.0 turns gamma correction off. The *−e* option specifies an exposure compensation in f-stops (powers of two). Only integer stops are allowed, for efficiency. The *−n* option specifies a sampling factor for neural network color quantization. This value should be between 1 and 80, where 1 takes the longest and produces the best results in small areas of the image. If no value is given, a faster median cut algorithm is used. The *−r* option invokes a reverse conversion, from an 8-bit Targa file to a RADIANCE picture. If the output file is missing, the standard output is used.

## AUTHORS

Greg Ward
Anthony Dekker provided the code for neural network color quantization

## BUGS

Run-length encoded files can be read but not written with this program.

## SEE ALSO

pfilt(1), ra_bmp(1), ra_bn(1), ra_ppm(1), ra_pr(1), ra_pr24(1), ra_t16(1), ra_tiff(1), ximage(1)

## NAME

ra_tiff - convert RADIANCE picture to/from a TIFF color or greyscale image

## SYNOPSIS

**ra_tiff** [ **-z|-L|-l|-f|-w** ][ **-b** ][ **-e +/-stops** ][ **-g gamma** ] { **in.hdr|-** } **out.tif**
**ra_tiff -r** [ **−x** ][ **−g gamma** ][ **−e +/-stops** ] **in.tif** [ **out.hdr|-** ]

## DESCRIPTION

*Ra_tiff* converts between RADIANCE and TIFF image formats. The *−g* option specifies the exponent used in gamma correction; the default value is 2.2, which is the recommended value for TIFF images.

The *−b* option can be used to specify an 8-bit greyscale TIFF output file. The type of input file is determined automatically.

The *−z* option will result in LZW compression of the TIFF output file. The *−L* option specifies SGILOG compression, which is recommended to capture the full dynamic range of the Radiance picture. However, since many TIFF readers do not yet support this format, use this option under advisement. The *−l* option specifies SGILOG24 compressed output, which has less dynamic range than SGILOG, but may be smaller in some cases. (It is usually larger.) The *−f* option specifies 32-bit IEEE floating-point/primary output, which is the highest resolution format but results in very large files, since each RGB pixel takes 96 bits (12 bytes) and does not compress well. The *−w* option specifies 16-bit/primary output, which is understood by some photo editing software, such as Adobe Photoshop. Decompression is automatically determined for TIFF input.

The *−e* option specifies an exposure compensation in f-stops (powers of two). Only integer stops are allowed, for efficiency.

The *−r* option invokes a reverse conversion, from a TIFF image to a RADIANCE picture. The RADIANCE picture file can be taken from the standard input or sent to the standard output by using a hyphen ('-') in place of the file name, but the TIFF image must be to or from a file. The *−x* option can be used to specify an XYZE Radiance output file, rather than the default RGBE.

## EXAMPLES

To convert a Radiance picture to SGILOG-compressed TIFF format:

```
ra_tiff –L scene1.hdr scene1.tif
```

To later convert this image back into Radiance and display using human visibility tone-mapping:

```
ra_tiff –r scene1.tif scene1.hdr
ximage –e human scene1.hdr
```

## AUTHOR

Greg Ward Larson
Sam Leffler

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland. Additions for the SGILOG data encoding were sponsored by Silicon Graphics, Inc.

## BUGS

Many TIFF file subtypes are not supported.
A gamma value other than 2.2 is not properly recorded or understood if recorded in the TIFF file.

## SEE ALSO

pfilt(1), ra_bmp(1), ra_bn(1), ra_ppm(1), ra_pr(1), ra_pr24(1), ra_t8(1), ra_t16(1), ximage(1)

## NAME

ra_xyze - convert between RADIANCE RGBE and XYZE formats

## SYNOPSIS

**ra_xyze** [ **−r** ][ **−e exposure** ][ **−o** ][ **−c** | **−u** ][ **−p xr yr xg yg xb yb xw yw** ] [ **input** [ **output** ] ]

## DESCRIPTION

*Ra_xyze* converts between RADIANCE RGBE (red,green,blue,exponent) and XYZE (CIE X,Y,Z,exponent) formats. The *−e* option specifies an exposure compensation, which may be given as a decimal multiplier or in f-stops (powers of two). The *−o* option may be used to specify original units, to which the exposure compensation is applied. Otherwise, the multiplier is in addition to any previous exposure adjustment. By default, *ra_xyze* produces a flat XYZE RADIANCE picture file from any type of RADIANCE input picture. To override these defaults, the *−c* option may be used to specify run-length encoded output, or the *−u* option may be used to specify a flat output.

The *−r* option causes *ra_xyze* to produce a run-length encoded RGBE file instead, unless *−u* is given, also, when it will produce a flat RGBE file. The *−p* option may be used to override the standard RADIANCE RGB primary colors to tailor the image for a particular output device or representation. The eight floating-point arguments to this option are the 1931 CIE (x,y) chromaticity coordinates of the three RGB primaries plus the white point, in that order. The new primaries will be recorded in the header of the output file, so that the original information may be fully recovered later. It is not necessary that the input file be in XYZE format. Th *−r* option may therefore be used to convert from one RGB primary representation to another using the *−p* option.

If the output file is missing, the standard output is used. If the input file is missing as well, the standard input is used.

## NOTES

The CIE standard used is the 1931 2-degree observer, and the correct output representation relies on the original RADIANCE input description being defined properly in terms of the standard RADIANCE RGB primaries, whose CIE (x,y) chromaticity values are defined in the header file in src/common/color.h. In this same file is a standard for the luminous efficacy of white light (WHTEFFICACY), which is used as a conversion between lumens and watts throughout RADIANCE. This same factor is applied by *ra_xyze* when converting between the radiometric units of the RGBE format and the photometric units of the XYZE format. The purpose of this factor is to ensure that the Y component of the CIE representation is luminance in units of candelas/meter^2.

Most of the RADIANCE picture filters should work uniformly on either RGBE or XYZE files, so it is not necessary to convert back to RGBE format except for conversion or display, in which case the correct primaries for the chosen output device should be specified with the *−p* option if they are known.

## EXAMPLES

To convert RGBE output from *rpict(1)* into run-length encoded XYZE format:

```
rpict [options] scene.oct | ra_xyze −c > scene_xyz.hdr
```

To prepare a RADIANCE picture for display on a calibrated NTSC monitor:

```
ra_xyze −r −p .670 .330 .210 .710 .140 .080 .333 .333 stand.hdr ntsc.hdr
```

## AUTHOR

Greg Ward

## BUGS

Any color correction applied to the original image is not removed or translated by *ra_xyze,* and it may result in color shifts in the output. If color preservation is important and the correction is unwanted, it is best to remove it with *pfilt(1)* using the *−er, −eg* and *−eb* options first. (Simply look at the header and apply the reciprocal primaries of all COLORCORR= lines multiplied together.) Better still, get the picture before color correction is applied.

## SEE ALSO

pfilt(1), pvalue(1), ra_rgbe(1), rpict(1)

## NAME

rad - render a RADIANCE scene

## SYNOPSIS

**rad** [ **−s** ][ **-n** | **-N npr** ][ **−t** ][ **−e** ][ **−V** ][ **−w** ][ **−v view** ][ **−o device** ] **rfile** [ **VAR=value ..** ]

## DESCRIPTION

*Rad* is an executive program that reads the given *rfile* and makes appropriate calls to *oconv(1), mkillum(1), rpict(1), pfilt(1),* and/or *rvu(1)* to render a specific scene. Variables in *rfile* give input files and qualitative information about the rendering(s) desired that together enable *rad* to intelligently set parameter values and control the simulation.

Normally, commands are echoed to the standard output as they are executed. The *−s* option tells *rad* to do its work silently. The *−n* option tells *rad* not to take any action (ie. not to actually execute any commands). The *−N* option instructs *rad* to run as many as *npr* rendering processes in parallel. The *−t* option tells *rad* to bring rendering files up to date relative to the input (scene description) files, without performing any actual calculations. If no octree exists, it is still necessary to run *oconv(1)* to create one, since the *−t* option will not create invalid (i.e. empty) files, and a valid octree is necessary for the correct operation of *rad*. The *−e* option tells *rad* to explicate all variables used for the simulation, including default values not specified in the input file, and print them on the standard output.

Normally, *rad* will produce one picture for each view given in *rfile*. The *−v* option may be used to specify a single desired view. The *view* argument may either be a complete view specification (enclosed in quotes and beginning with an optional identifier) or a number or single-word identifier to match a view defined in *rfile*. If the argument is one of the standard view identifiers, it may or may not be further elaborated in *rfile*. (See "view" variable description, below.) If the argument does not match any views in *rfile* and is not one of the standard views, no rendering will take place. This may be convenient when the only action desired of *rad* is the rebuilding of the octree. In particular, the argument "0" will never match a view.

If the *−V* option is given, each view will be printed on the standard output before being applied, in a form suitable for use in a view file or *rpict* rendering sequence. This is helpful as feedback or for accessing the *rad* view assignments without necessarily starting a rendering.

By default, *rad* will run *rpict* and *pfilt* to produce a picture for each view. The *−o* option specifies an output device for *rvu* (usually "x11") and runs this interactive program instead, using the first view in *rfile* or the view given with the *−v* option as the starting point.

Additional variable settings may be added or overridden on the command line following *rfile*. Upper case variables specified more than once will result in a warning message (unless the *−w* option is present), and the last value given will be the one used.

The *−w* option turns off warnings about multiply and misassigned variables.

Rendering variable assignments appear one per line in *rfile*. The name of the variable is followed by an equals sign ('=') and its value(s). The end of line may be escaped with a backslash ('\'), though it is not usually necessary since additional variable values may be given in multiple assignments. Variables that should have only one value are given in upper case. Variables that may have multiple values are given in lower case. Variables may be abbreviated by their first three letters. Comments in *rfile* start with a pound sign ('#') and proceed to the end of line.

The rendering variables, their interpretations and default values are given below.

**OCTREE** The name of the octree file. The default name is the same as *rfile* but with any suffix replaced by ".oct". (The octree must be a file -- *rad* cannot work with commands that produce octrees.)

**ZONE** This variable specifies the volume of interest for this simulation. The first word is either "Interior" or "Exterior", depending on whether the zone is to be observed from the inside or the outside, respectively. (A single letter may be given, and case does not matter.) The following six numbers are the minimum and maximum X coordinates, minimum and maximum Y, and minimum and maximum Z for the zone perimeter. It is important to give the zone as it is used to determine many of the rendering parameters. The default exterior zone is the bounding cube

for the scene as computed by *oconv*.

**EXPOSURE**

This variable tells *rad* how to adjust the exposure for display. It is important to set this variable properly as it is used to determine the ambient value. An appropriate setting may be discovered by running *rvu* and noting the exposure given by the "exposure =" command. As in *rvu* and *pfilt*, the exposure setting may be given either as a multiplier or as a number of f–stop adjustments (eg. +2 or −1.5). There is no default value for this variable. If it is not given, an average level will be computed by *pfilt* and the ambient value will be set to 10 for exterior zones and 0.01 for interior zones.

**EYESEP** The interocular spacing for stereo viewing. I.e., the world distance between the pupils of the left and right eyes. The default value is the sum of the three "ZONE" dimensions divided by 100.

**scene** This variable is used to specify one or more scene input files. These files will be given together with the materials file(s) and any options specified by the "oconv" variable to *oconv* to produce the octree given by the "OCTREE" variable. In-line commands may be specified in quotes instead of a file, beginning with an exclamation mark ('!'). If the "scene" variable is not present, then the octree must already exist in order for *rad* to work. Even if this variable is given, *oconv* will not be run unless the octree is out of date with respect to the input files. Note that the order of files in this variable is important for *oconv* to work properly, and files given in later variable assignments will appear after previous ones on the *oconv* command line.

**materials** This variable is used to specify files that, although they must appear on the *oconv* command line, do not affect the actual octree itself. Keeping the materials in separate files allows them to be modified without requiring the octree to be rebuilt (a sometimes costly procedure). These files should not contain any geometry, and the *−f* option must not be given in the "oconv" variable for this to work.

**illum** This variable is used to specify files with surfaces to be converted into illum sources by *mkillum(1)*. When this variable is given, additional octree files will be created to contain the scene before and after illum source conversion. These files will be named according to the (default) value of the *OCTREEE* variable, with either a '0' or a '1' appearing just before the file type suffix (usually ".oct").

**objects** This variable is used for files that, although they do not appear on the *oconv* command line, contain geometric information that is referenced indirectly by the scene files. If any of these files is changed, the octree will be rebuilt. (The *raddepend(1)* command may be used to find these dependencies automatically.)

**view** This variable is used to specify a desired view for this zone. Any number of "view" lines may be given, and each will result in a rendered picture (unless the *−v* or *−o* option is specified). The value for this variable is an optional identifier followed by any number of view options (see *rpict(1)* for a complete listing). The identifier is used in file naming and associating a desired view with the *−v* command line option. Also, there are several standard view identifiers defined by *rad*. These standard views are specified by strings of the form "[Xx]?[Yy]?[Zz]?[vlcahs]?". (That is, an optional upper or lower case X followed by an optional upper or lower case Y followed by an optional upper or lower case Z followed by an optional lower case V, L, C, A or H.) The letters indicate the desired view position, where upper case X means maximum X, lower case means minimum and so on. The final letter is the view type, where 'v' is perspective (the default), 'l' is parallel, 'c' is a cylindrical panorama, is a planisphere (stereographic) fisheye. A perspective view from maximum X, minimum Y would be "Xy" or "Xyv". A parallel view from maximum Z would be "Zl". If "ZONE" is an interior zone, the standard views will be inside the perimeter. If it is an exterior zone, the standard views will be outside. Note that the standard views are best used as starting points, and additional arguments may be given after the identifier to modify a standard view to suit a particular model. The default view is "X" if no views are specified. A single specified view of "0" means no views will be automatically generated.

**UP** The vertical axis for this scene. A negative axis may be specified with a minus sign (eg. "−Y"). There is no default value for this variable, although the standard views assume Z is up if no other axis is specified.

**RESOLUTION**
This variable specifies the desired final picture resolution. If only a single number is given, this value will be used for both the horizontal and vertical picture dimensions. If two numbers are given, the first is the horizontal resolution and the second is the vertical resolution. If three numbers are given, the third is taken as the pixel aspect ratio for the final picture (a real value). If the pixel aspect ratio is zero, the exact dimensions given will be those produced. Otherwise, they will be used as a frame in which the final image must fit. The default value for this variable is 512.

**QUALITY**
This variable sets the overall rendering quality desired. It can have one of three values, "LOW", "MEDIUM" or "HIGH". These may be abbreviated by their first letter, and may be in upper or lower case. Most of the rendering options will be affected by this setting. The default value is "L".

**PENUMBRAS**
This is a boolean variable indicating whether or not penumbras are desired. A value of "TRUE" will result in penumbras (soft shadows), and a value of "FALSE" will result in no penumbras (sharp shadows). True and false may be written in upper or lower case, and may be abbreviated by a single letter. Renderings generally proceed much faster without penumbras. The default value is "F".

**INDIRECT**
This variable indicates how many diffuse reflections are important in the general lighting of this zone. A direct lighting system (eg. fluorescent troffers recessed in the ceiling) corresponds to an indirect level of 0. An indirect lighting system (eg. hanging fluorescents directed at a reflective ceiling) corresponds to an indirect level of 1. A diffuse light shelf reflecting sunlight onto the ceiling would correspond to an indirect level of 2. The setting of this variable partially determines how many interreflections will be calculated. The default value is 0.

**PICTURE** This is the root name of the output picture file(s). This name will have appended the view identifier (or a number if no id was used) and a ".hdr" suffix. If a picture corresponding to a specific view exists and is not out of date with respect to the given octree, it will not be re-rendered. The default value for this variable is the root portion of *rfile*.

**RAWFILE**
This is the root name of the finished, raw *rpict* output file. If specified, *rad* will rename the original *rpict* output file once it is finished and filtered rather than removing it, which is the default action. The given root name will be expanded in the same way as the "PICTURE" variable, and if the "RAWFILE" and "PICTURE" variables are identical, then no filtering will take place.

**ZFILE** This is the root name of the raw distance file produced by the −*z* option of *rpict*. To this root name, an underscore plus the view name plus a ".zbf" suffix will be added. If no "ZFILE" is specified, none will be produced.

**AMBFILE**
This is the name of the file where "ambient" or diffuse interreflection values will be stored by *rpict* or *rvu*. Although it is not required, an ambient file should be given whenever an interreflection calculation is expected. This will optimize successive runs and minimize artifacts. An interreflection calculation will take place when the "QUALITY" variable is set to HIGH, or when the "QUALITY" variable is set to MEDIUM and "INDIRECT" is positive. There is no default value for this variable.

**DETAIL** This variable specifies the level of visual detail in this zone, and is used to determine image sampling rate, among other things. If there are few surfaces and simple shading, then this

should be set to LOW. For a zone with some furniture it might be set to MEDIUM. If the space is very cluttered or contains a lot of geometric detail and textures, then it should be set to HIGH. The default value is "M".

**VARIABILITY**
This variable tells *rad* how much light varies over the surfaces of this zone, and is used to determine what level of sampling is necessary in the indirect calculation. For an electric lighting system with uniform coverage, the value should be set to LOW. For a space with spot lighting or a window with sky illumination only, it might be set to MEDIUM. For a space with penetrating sunlight casting bright patches in a few places, it should be set to HIGH. The default value is "L".

**OPTFILE** This is the name of a file in which *rad* will place the appropriate rendering options. This file can later be accessed by *rpict* or *rvu* in subsequent manual runs using the at-sign ('@') file insert option. (Using an "OPTFILE" also reduces the length of the rendering command, which improves appearance and may even be necessary on some systems.) There is no default value for this variable.

**REPORT** This variable may be used to specify a reporting interval for batch rendering. Given in minutes, this value is multiplied by 60 and passed to *rpict* with the *−t* option. If a filename is given after the interval, it will be used as the error file for reports and error messages instead of the standard error. (See the *−e* option of *rpict(1)*. There is no default value for this variable.

**oconv** This variable may be used to specify special options to *oconv*. If the first word of the first instance of this variable is not an option, it will be used in place of the default command path, "oconv". See the *oconv(1)* manual page for a list of valid options.

**mkillum** This variable may be used to specify additional options to *mkillum*. If the first word of the first instance of this variable is not an option, it will be used in place of the default command path, "mkillum". See the *rtrace(1)* manual page for a list of valid options.

**render** This variable may be used to specify additional options to *rpict* or *rvu*. These options will appear after the options set automatically by *rad,* and thus will override the default values.

**rpict** This variable may be used to specify overriding options specific to *rpict*. If the first word of the first instance of this variable is not an option, it will be used in place of the default command path, "rpict". See the *rpict(1)* man page for a list of valid options.

**rvu** This variable may be used to specify overriding options specific to *rvu*. If the first word of the first instance of this variable is not an option, it will be used in place of the default command path, "rvu". See the *rvu(1)* man page for a list of valid options.

**pfilt** This variable may be used to specify additional options to *pfilt*. If the first word of the first instance of this variable is not an option, it will be used in place of the default command path, "pfilt". See the *pfilt(1)* manual page for details.

## EXAMPLES

A minimal input file for *rad* might look like this:

```
::::::::::
sample.rif
::::::::::
# The octree we want to use:
OCTREE= tutor.oct              # w/o this line, name would be "sample.oct"
# Our scene input files:
scene= sky.rad outside.rad room.rad srcwindow.rad
# The interior zone cavity:
ZONE= I  0 3  0 2  0 1.75      # default would be scene bounding cube
# The z-axis is up:
UP= Z                          # no default - would use view spec.
```

```
# Our exposure needs one f-stop boost:
EXPOSURE= +1                    # default is computed ex post facto
```

Note that we have not specified any views in the file above. The standard default view "X" would be used if we were to run *rad* on this file. If we only want to see what default values *rad* would use without actually executing anything, we can invoke it thus:

```
rad -n -e sample.rif
```

This will print the variables we have given as well as default values *rad* has assigned for us. Also, we will see the list of commands that *rad* would have executed had the *-n* option not been present. (Note if the octree, "tutor.oct", is not present, an error will result as it is needed to determine some of the opiton settings.)

Different option combinations have specific uses, ie:

```
rad -v 0 sample.rif OPT=samp.opt  # build octree, put options in "sample.opt"
rad -n -e -s sample.rif > full.rif     # make a complete rad file
rad -n sample.rif > script.sh          # make a script of commands
rad -V -v Zl -n -s sample.rif > plan.vf       # make a plan view file
rad -t sample.rif              # update files after minor change to input
rad -s sample.rif &                    # execute silently in the background
rad -N 2 sample.rif            # render views using two parallel rpict calls
rad -N 4 -v 1 sample.rif       # render first view with four rpiece calls
```

If we decide that the default values *rad* has chosen for our variables are not all appropriate, we can add some more assignments to the file:

```
QUAL= MED                # default was low
DET= low                 # default was medium - our space is almost empty
PEN= True                # we want to see soft shadows from our window
VAR= hi            # daylight can result in fairly harsh lighting
view= XYa -vv 120        # let's try a fisheye view
PICT= tutor              # our picture name will be "tutor_XYa.hdr"
```

Note the use of abbreviations, and the modification of a standard view. Now we can invoke *rad* to take a look at our scene interactively with *rvu:*

```
rad -o x11 sample.rif
```

*Rad* will run *oconv* first to create the octree (assuming it doesn't already exist), then *rvu* with a long list of options. Let's say that from within *rvu,* we wrote out the view files "view1.vp" and "view2.vp". We could add these to "sample.rif" like so:

```
view= vw1 -vf view1.vp          # Our first view
view= vw2 -vf view2.vp          # Our second view
RESOLUTION= 1024                # Let's go for a higher resolution result
```

To start *rvu* again using vw2 instead of the default, we use:

```
rad -o x11 -v vw2 sample.rif
```

Once we are happy with the variable settings in our file, we can run *rad* in the background to produce one image for each view:

```
rad sample.rif REP=5 >& errfile &
```

This will report progress every five minutes to "errfile".

## FILES

$(PICTURE)_$(view).unf Unfinished output of *rpict*

## AUTHOR

Greg Ward

## BUGS

You cannot run more than one *rad* process at a time on the same input file, as the second process will attempt to recover the output files of the first process, damaging the results. The exceptions to this are running interactively via the *−o* option, or rendering different views using the *−v* option.

Incremental building of octrees is not supported as it would add considerable complexity to *rad*. Complicated scene builds should still be left to *make(1),* which has a robust mechanism for handling hierarchical dependencies. If *make* is used in this fashion, then only the "OCTREE" variable of *rad* is needed.

The use of some *pfilt* options is awkward, since the "EXPOSURE" variable results in a single pass invocation (the *−1* option of *pfilt* and two passes are necessary for certain effects, such as star patterns. The way around this problem is to specify a "RAWFILE" that is the same as the "PICTURE" variable so that no filtering takes place, then call *pfilt* manually. This is preferable to leaving out the "EXPOSURE" variable, since the exposure level is needed to accurately determine the ambient value for *rpict*.

The use of upper and lower case naming for the standard views may be problematic on systems that don't distinguish case in filenames.

## SEE ALSO

glrad(1), make(1), mkillum(1), objview(1), oconv(1), pfilt(1), raddepend(1), ranimate(1), rholo(1), rpict(1), rpiece(1), rtrace(1), rvu(1), touch(1), vgaimage(1), ximage(1)

## NAME

rad2mgf - convert RADIANCE scene description to Materials and Geometry Format

## SYNOPSIS

**rad2mgf** [ **−dU** ] [ **input ..** ]

## DESCRIPTION

*Rad2mgf* converts one or more RADIANCE scene files to the Materials and Geometry Format (MGF). Input units are specified with the *−mU* option, where *U* is one of 'm' (meters), 'c' (centimeters), 'f' (feet) or 'i' (inches). The assumed unit is meters, which is the required output unit for MGF (thus the need to know). If the input dimensions are in none of these units, then the user should apply *xform(1)* with the *−s* option to bring the units into line prior to translation.

The MGF material names and properties for the surfaces will be those assigned in RADIANCE. If a referenced material has not been defined, then its name will be invoked in the MGF output without definition, and the description will be incomplete.

## LIMITATIONS

Although MGF supports all of the geometric types and the most common material types used in RADIANCE, there is currently no support for advanced BRDF materials, patterns, textures or mixtures. Also, the special types "source" and "antimatter" are not supported, and all light source materials are converted to simple diffuse emitters (except "illum" materials, which are converted to their alternates). These primitives are reproduced as comments in the output and must be replaced manually if necessary.

The RADIANCE "instance" and "mesh" types aretreated specially. *Rad2mgf* converts each instance or mesh to an MGF include statement, using the corresponding transformation and a file name derived from the octree or mesh name. (The original octree/mesh suffix is replaced by ".mgf".) For this to work, the user must separately create the referenced MGF files from the original RADIANCE descriptions. The description file names can usually be determined using the *getinfo(1)* command run on the octrees in question.

## EXAMPLE

To convert three RADIANCE files (in feet) to one MGF file:

```
rad2mgf −df file1.rad file2.rad file3.rad > scene.mgf
```

To translate a RADIANCE materials file to MGF:

```
rad2mgf materials.rad > materials.mgf
```

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), ies2rad(1), mgf2meta(1), mgf2rad(1), obj2rad(1), oconv(1), xform(1)

MGF web site "http://radsite.lbl.gov/mgf/HOME.html"

## NAME

raddepend - find RADIANCE scene dependencies

## SYNOPSIS

**raddepend** file ..

## DESCRIPTION

*Raddepend* uses *getbbox(1)* to expand scene file arguments and find file dependencies for *make(1)* or *rad(1)*. *Raddepend* looks only in the current directory, so dependencies hidden elsewhere in the filesystem will not be found or named.

The output is the name of files, one per line, that were accessed during the expansion of the input file arguments. The file arguments are excluded from the list. If no input files are given, the standard input is read.

## AUTHOR

Greg Ward

## BUGS

On some older NFS systems, the file access dates are not updated promptly. As a result, *raddepend* may not be 100% reliable on these systems. If the output seems to be missing essential files, this is no doubt why. The only fix is to put in a longer sleep time between the *getbbox* call and the final *ls(1)*.

## SEE ALSO

make(1), oconv(1), rad(1), xform(1)

## NAME

ran2tiff - expose and convert a RADIANCE animation to TIFF frames

## SYNOPSIS

**ran2tiff** [ **-W prev_frame_wt** ][ **-H histo** ][ **-D dir** ] [ **pcond opts** ] [ **ra_tiff opts** ] **frame1.hdr frame2.hdr ..**

## DESCRIPTION

*Ran2tiff* takes an animated sequence of Radiance pictures and converts them by calling *pcond(1)* and *ra_tiff(1)* in such a way that exposure changes gradually from one frame to the next. It accomplishes this by computing a histogram for each image and blending it with a weighted histogram from the previous frame, where the weight is set by the *−W* option, or 0.9 by default. A value of 0.5 would mean that the previous frame carries 1/2 the weight of the current one, and the frame before that 1/4, 1/8 before that, and so on.

The *−H* option specifies a histogram file to be used to ensure continuous exposure between the current sequence and one before it. If the file does not exist, it will be created at the end of the conversion. If it does exist, its contents will be used as the histogram at the end of the previous sequence.

The *−D* option may be used to specify an output directory for the resulting TIFF frames. Otherwise, the current directory will be used. The output file names will correspond to the input names, but with a ".tif" suffix.

Ran2tiff also accepts the standard options for *pcond* and *ra_tiff*, though output to LogLuv TIFF formats is not supported. (There would be no point in adjusting the exposure if the output were HDR.)

## EXAMPLE

To convert a sequence of frames using a rapid human exposure response and LZW compression on the output:

```
ran2tiff −W 0.7 −h −z frame*.hdr
```

Similar to *fieldcomb(1),* this program works well with the "TRANSFER" facility of *ranimate(1)*.

## AUTHOR

Greg Ward

## SEE ALSO

fieldcomb(1), getinfo(1), pcond(1), pfilt(1), phisto(1), ra_tiff(1), ranimate(1), rpict(1)

## NAME

ranimate - compute a RADIANCE animation

## SYNOPSIS

**ranimate** [ **−s** ][ **−n** ][ **−e** ][ **−w** ] **ranfile**

## DESCRIPTION

*Ranimate* is an executive program that reads the given *ranfile* and makes appropriate calls to *rad(1), rpict(1), pinterp(1),* and/or *pfilt(1)* to render an animation. Variables in *ranfile* indicate input files, process servers (execution hosts), output directories and file names, and various other controls and options.

Normally, commands are echoed to the standard output as they are executed. The *−s* option tells *ranimate* to do its work silently. The *−n* option tells *ranimate* not to take any action (ie. not to actually execute any commands). The *−e* option tells *ranimate* to explicate all variables used for the animation, including default values not specified in the input file, and print them on the standard output.

The *−w* option turns off warnings about multiply and misassigned variables.

Normally, *ranimate* will produce one animation frame for each view given in the specified view file. If an animation has ended or been killed in an incomplete state, however, *ranimate* will attempt to pick up where the earlier process left off. If the process is still running, or was started on another machine, *ranimate* will report this information and exit.

Animation variable assignments appear one per line in *ranfile*. The name of the variable is followed by an equals sign ('=') and its value(s). The end of line may be escaped with a backslash ('\'), though it is not usually necessary since additional variable values may be given in multiple assignments. Variables that should have only one value are given in upper case. Variables that may have multiple values are given in lower case. Variables may be abbreviated by their first three letters, except for "host", which must have all four. Comments in *ranfile* start with a pound sign ('#') and proceed to the end of line.

The animation variables, their interpretations and default values are given below.

**DIRECTORY**
: The name of the animation directory. All temporary files generated during the animation will be placed in this directory, which will be created by *ranimate* if it does not exist. A file named "STATUS" will also be created there, and will contain current information about the animation process. This variable has no default value, and its setting is required.

**OCTREE**
: The name of the octree file for a static scene walk-through animation. There is no default value for this variable, and any setting will be ignored if the *ANIMATE* variable is also set (see below).

**ANIMATE**
: The scene generation command for a dynamic animation. This command, if given, will be executed with the frame number as the final argument, and on its standard output it must produce the complete octree for that frame. Care must be taken that this command does not create any temporary files that might collide with same-named files created by other animation commands running in parallel. Also, the command should produce no output to the standard error, unless there is a fatal condition. (I.e., switch all warnings off; see the BUGS section, below.) There is no default animation command, and either this variable or the *OCTREE* variable must be set.

**VIEWFILE**
: This variable names a file from which *ranimate* may extract the view for each frame in the animation. This file should contain one valid view per frame, starting with frame 1 on line 1, regardless of the setting of the *START* variable. An exception is made for a view file with only a single view, which is used for every frame of a dynamic scene animation. This variable is required, and there is no default value.

**START**
: The initial frame number in this animation sequence. The minimum value is 1, and if a later starting frame is given, *ranimate* assumes that the earlier frames are included in some other *ranfile,* which has been previously executed. (See the *NEXTANIM* variable, below.) The

default value is 1.

**END** The final frame number in this sequence. The minimum value is equal to the *START* frame, and the default value is computed from the number of views in the given *VIEWFILE*.

**EXPOSURE**
This variable tells *ranimate* how to adjust the exposure for each frame. As in *pfilt,* the exposure setting may be given either as a multiplier or as a number of f–stop adjustments (eg. +2 or –1.5). Alternatively, a file name may be given, which *ranimate* will interpret as having one exposure value per line per frame, beginning with frame 1 at line 1. (See also the *VIEWFILE* variable, above.) There is no default value for this variable. If it is not given, an average level will be computed by *pfilt* for each frame.

**BASENAME**
The base output file name for the final frames. This string will be passed to the *–o* and *–z* options of rpict, along with appropriate suffixes, and thus should contain a *printf(3)* style integer field to distinguish one frame number from another. The final frames will use this name with a ".hdr" suffix. The default value is the assigned *DIRECTORY* followed by "/frame%03d".

**host** A host to use for command execution. This variable may be assigned a host name, followed by an optional number of parallel processes, followed by an optional directory (relative to the user's home directory on that machine), followed by an alternate user name. Multiple *host* assignments may appear. It is not advisable to specify more than one process on a single-CPU host, as this just tends to slow things down. The default value is "localhost", which starts a single process in the current directory of the local machine.

**RIF** This variable specifies a *rad* input file to use as a source of rendering options and other variable settings. If given, *ranimate* will execute *rad* and create an options file to later pass to *rpict* or *rtrace*. Besides prepending the *render* variable, *ranimate* will also extract default settings for the common variables: *OCTREE, RESOLUTION, EXPOSURE* and *pfilt*. Following the file name, overriding variable settings may be given, which will be passed to *rad* on the command line. Settings with spaces in them should be enclosed in quotes. The execution of *rad* will also update the contents of the octree, if necessary. There is no default value for this variable.

**DISKSPACE**
Specify the amount of disk space (in megabytes) available on the destination file system for temporary file storage. *Ranimate* will coordinate its batch operations based on this amount of storage, assuming that there is either enough additional space for all the final frames, or that the given *TRANSFER* command will move the finished frames to some other location (see below). The default value is 100 megabytes.

**ARCHIVE**
After each batch rendering is finished and checked for completeness, *ranimate* will execute the given command, passing the names of all the original pictures and z-buffer files generated by *rpict*. (The command is executed in the destination directory, and file names will be simple.) Normally, the archive command copies the original files to a tape device or somewhere that they can be retrieved in the event of failure in the frame interpolation stages. After the archive command has successfully completed, the original renderings are removed. There is no default value for this variable, meaning that the original unfiltered frames will simply be removed. Note that the last one or two rendered frames may not be copied, archived or removed in case there is a another sequence picking up where this one left off.

**TRANSFER**
The command to transfer the completed animation frames. The shell changes to the destination directory and appends the names of all the finished frames to this command before it is executed. Normally, the transfer command does something such as convert the frames to another format and/or copy them to tape or some other destination device before removing them. The *fieldcomb(1)* script may be used to conveniently combine fields into frames for

field-based animations. If this variable is not given, the final frames are left where they are. (See *BASENAME,* above.)

**RSH** The command to use instead of *ssh(1)* to execute commands remotely on another machine. The arguments and behavior of this program must be identical to the UNIX *ssh* command, except that the *-l* option will always be used to specify an alternate user name rather than the *user@host* convention. The *-l* option may or may not appear, but the *-n* option will always be used, and the expected starting directory will be that of the remote user, just as with *ssh*.

**NEXTANIM**
This variable specifies the next *ranfile* to use after this sequence is completed. This offers a convenient means to continue an animation that requires different control options in different segments. It is important in this case to correctly set the *START* and *END* variables in each *ranfile* so that the segments do not overlap frames.

**OVERSAMPLE**
This variable sets the multiplier of the original image size relative to the final size given by the *RESOLUTION* variable. This determines the quality of anti-aliasing in the final frames. A value of 1 means no anti-aliasing, and a value of 3 produces very good anti-aliasing. The default value is 2. (A fractional value may be used for previews, causing low resolution frames with large, blocky pixels to be produced.)

**INTERPOLATE**
This variable sets the number of frames to interpolate between each rendered frame in a static scene walk-through. Z-buffers for each rendered frame will be generated by *rpict,* and *pinterp* will be called to perform the actual "tweening." This results in a potentially large savings in rendering time, but should be used with caution since certain information may be lost or inaccurate, such as specular highlights and reflections, and objects may even break apart if too few renderings are used to interpolate too much motion. The default value for this variable is 0, meaning no interpolation. Interpolation is also switched off if the *ANIMATE* variable is specified.

**MBLUR** This variable specifies the fraction of a frame time that the shutter is simulated as being open for motion blur. A number of samples may be given as a second argument, which controls the number of additional frames computed and averaged together by *pinterp*. If this number is less than 2, then bluring is performed by *rpict* only, resulting in greater noise than the combination of *rpict* and *pinterp* used otherwise. (The default value for number of samples is 5.) The default fraction is 0, meaning no motion blurring. This option does not currently work with the *ANIMATE* variable, since pinterp only works for static environments.

**DBLUR** This variable specifies the aperture diameter for depth-of-field blurring, in world units. A number of samples may be given as a second argument, which controls the number of additional frames computed and averaged together by *pinterp*. If this number is less than 2, then blurring is performed by *rpict* only, resulting in greater noise than the combination of *rpict* and *pinterp* used otherwise. (The default value for number of samples is 5.) To simulate a particular camera's aperture, divide the focal length of the lens by the f-number, then convert to the corresponding world coordinate units. For example, if you wish to simulate a 50mm lens at f/2.0 in a scene modeled in meters, then you divide 50mm by 2.0 to get 25mm, which corresponds to an effective aperture of 0.025 meters. The default aperture is 0, meaning no depth-of-field blurring. This option does not currently work with the *ANIMATE* variable, since pinterp only works for static environments.

**RTRACE** This boolean variable tells *ranimate* whether or not to employ *rtrace* during frame interpolation using the *–fr* option to *pinterp*. If set to True, then the same rendering options and static octree are passed to *rtrace* as are normally used by *rpict*. The default value is False. Note that this variable only applies to static environment walk-throughs (i.e., no *ANIMATE* command).

**RESOLUTION**

This variable specifies the desired final picture resolution. If only a single number is given, this value will be used for both the horizontal and vertical picture dimensions. If two numbers are given, the first is the horizontal resolution and the second is the vertical resolution. If three numbers are given, the third is taken as the pixel aspect ratio for the final picture (a real value). If the pixel aspect ratio is zero, the exact dimensions given will be those produced. Otherwise, they will be used as a frame in which the final image must fit. The default value for this variable is 640.

**render** This variable may be used to specify additional options to *rpict* or *rtrace*. These options will appear after the options set automatically by *rad,* and thus will override the default values.

**pinterp** This variable may be used to specify additional options to *pinterp,* which is used to interpolate frames for a static scene walk-through. (See the *pinterp* man page, and the *INTERPOLATE* variable.) Do not use this variable to set the *pinterp −fr* option, but use the *RTRACE* setting instead.

**pfilt** This variable may be used to specify additional options to *pfilt*. If this variable is given in the *ranfile,* then *pfilt* will always be used. (Normally, *pfilt* is called only if *pinterp* is not needed or automatic exposure is required.) See the *pfilt* manual page for details.

## EXAMPLES

A minimal input file for *ranimate* might look like this:

```
::::::::::
sample.ran
::::::::::
# The rad input file for our static scene:
RIF= tutor.rif
# The spool directory:
DIRECTORY= anim1
# The view file containing one view per frame:
VIEWFILE= anim1.vf
# The amount of temporary disk space available:
DISKSPACE= 50# megabytes
```

Note that most of the variables are not set in this file. If we only want to see what default values *ranimate* would use without actually executing anything, we can invoke it thus:

```
ranimate −n −e sample.ran
```

This will print the variables we have given as well as default values *ranimate* has assigned for us. Also, we will see the list of commands that *ranimate* would have executed had the *−n* option not been present.

Usually, we execute *ranimate* in the background, redirecting the standard output and standard error to a file:

```
ranimate sample.ran >& sample.err &
```

If we decide that the default values *ranimate* has chosen for our variables are not all appropriate, we can add some more assignments to the file:

```
host= rays 3 ~greg/obj/tutor ray      # execute as ray on multi-host "rays"
host= thishost                        # execute one copy on this host also
INTERP= 3                             # render every fourth frame
RES= 1024                             # shoot for 1024x resolution
MBLUR= .25                            # apply camera motion blur
EXP= anim1.exp                        # adjust exposure according to file
pfilt= −r .9                          # use Gaussian filtering
ARCHIVE= tar cf /dev/nrtape           # save original renderings to tape
```

Note the use of abbreviation for variable names.

## FILES

$(DIRECTORY)/STATUS animation status file $(DIRECTORY)/* other temporary files
$(BASENAME).hdr final animation frames

## AUTHOR

Greg Ward

## BUGS

Due to the difficulty of controlling processes on multiple execution hosts, the *−n* option of *ranimate* is not useful in the same way as *rad* for generating a script of executable commands to render the sequence. It may give an idea of the sequence of events, but certain temporary files and so forth will not be in the correct state if the user attempts to create a separate batch script.

If multiple processors are available on a given host and the *RTRACE* variable is set to True, then the *−PP* option of *rtrace* should be employed, but it is not. There is no easy way around this problem, but it has only minor consequences in most cases. (The *−PP* option is used for *rpict,* however.)

The current implementation of the remote shell does not return the exit status of the remote process, which makes it difficult to determine for sure if there has been a serious error or not. Because of this, *ranimate* normally turns off warnings on all rendering processes, and takes any output to standard error from a remote command as a sign that a fatal error has occurred. (This also precludes the use of the *−t* option to report rendering progress.) If the error was caused by a process server going down, the server is removed from the active list and frame recovery takes place. Otherwise, *ranimate* quits at that point in the animation.

The current execution environment, in particular the RAYPATH variable, will not be passed during remote command execution, so it is necessary to set whatever variables are important in the remote startup script (e.g., ".cshrc" for the C-shell). This requirement may be circumvented by substituting the *on(1)* command for *ssh(1)* using the *RSH* control variable, or by writing a custom remote execution script.

If a different remote user name is used, *ranimate* first attempts to change to the original user's directory with a command of the form *cd uname* . This works under *csh(1),* but may fail under other shells such as *sh(1).*

If multiple hosts with different floating point formats are used, *pinterp* will fail because the Z-buffer files will be inconsistent. (Recall that *pinterp* is called if INTERPOLATE > 0 and/or MBLUR is assigned.) Since most modern machines use IEEE floating point, this is not usually a problem, but it is something to keep in mind.

## SEE ALSO

fieldcomb(1), pfilt(1), pinterp(1), pmblur(1), rad(1), ran2tiff(1), ranimove(1), rpict(1), ssh(1), rtrace(1)

## NAME

ranimove - render a RADIANCE animation with motion

## SYNOPSIS

**ranimove** [ **−s** ][ **−e** ][ **−w** ][ **−f beg,end** ][ **−n nprocs** ][ **−t sec** ][ **−d jnd** ] **rnmfile**

## DESCRIPTION

*Ranimove* is a program for progressive animation rendering. Variables in the given *rnmfile* indicate input files, output file names, and various other controls and options.

Normally, progress reports are written to the standard output, but the *−s* option tells *ranimove* to do its work silently. The *−e* option tells *ranimove* to explicate all variables used for the animation, including default values not specified in the input file, and print them on the standard output. The *−w* option turns off warnings about multiply and misassigned variables and non-fatal rendering problems.

Normally, *ranimove* will produce one animation frame for each view given in the specified view file. If the *−f* option is specified, the animation will resume at the given frame, and continue to the end of the sequence, or to the second frame if one is given (separated from the first by a comma but no space).

The *−n* option specifies the number of processes to use for rendering. The default value is 1, which is appropriate for most machines that have a single central processing unit (CPU). If you are running on a machine with multiple CPUs, a larger value up to the number of processors may be used to improve rendering speed, depending on the system load.

Because *ranimove* renders each frame progressively, it needs some criteria for when to proceed to the next frame in the animation. The *−t* option is used to specify the maximum number of seconds to spend on any one frame. The default value for this option is 60 seconds. Additionally, the *−d* option may be used to specify a termination threshold in just-noticeable-differences. If the error can be reduced below this number of JNDs over the whole frame before the time allocation is spent, *ranimove* will then proceed to the next frame. A value of 2.0 JNDs is the point at which 75% of the people will notice a difference, and this is the level usually selected for such a termination test. There is no default value for this option, which means that rendering will proceed until the time allocation is spent for each frame, regardless. If *−t* is set to 0, *ranimove* will spend as much time as it takes to reduce the visible error below the value set by the *−d* option.

*Ranimove* renders each frame in three stages. In the first stage, a low-quality image is rendered using one ray sample per 16 pixels. In the second stage, pixels from the previous frame are extrapolated to their corresponding positions in this one, based on the given camera and object movements. A set of heuristics is applied to prevent errors in specular highlights and shadows, avoiding some of the errors typical with the *pinterp(1)* program. In the third stage, additional high-quality samples are used to refine important regions of the image that are judged to have visible errors. This proceeds until the stopping criteria specified by the *−t* and *-d* options are met, when the frame is filtered and written to the designated picture file.

The chief differences between this program and *ranimate(1)* are that motion blur is computed for objects as well as camera movement, and its progressive rendering allows better control over the tradeoff between frame accuracy and rendering time. Fewer controls are provided for managing the picture files produced by *ranimove,* and no facilities for distributed rendering are available other than executing *ranimove* on different machines using the *−f* option to manually partition the work.

Animation variable assignments appear one per line in *rnmfile*. The name of the variable is followed by an equals sign ('=') and its value(s). The end of line may be escaped with a backslash ('\'), though it is not usually necessary since additional variable values may be given in multiple assignments. Variables that should have only one value are given in upper case. Variables that may have multiple values are given in lower case. Variables may be abbreviated by their first three letters. Comments in *rnmfile* start with a pound sign ('#') and proceed to the end of line.

The animation variables, their interpretations and default values are given below.

**OCTREE** The name of the base octree file, which should be generated by the *oconv(1)* command using the *−f* option. There is no default value for this variable. If no *RIF* variable is given, the octree must be specified.

**RIF** This variable specifies a *rad(1)* input file to use as a source of rendering options and other variable settings. If given, *ranimate* will execute *rad* and create an options file to control rendering parameters. *Ranimate* will also extract default settings for the common variables: *OCTREE, RESOLUTION,* and *EXPOSURE*. Following the file name, overriding variable settings may be given, which will be passed to *rad* on the command line. Settings with spaces in them should be enclosed in quotes. The execution of *rad* will also update the contents of the octree, if necessary. There is no default value for this variable.

**move** This variable specifies an object (or objects) with a specific motion and/or rendering priority. Four value arguments are expected for each appearance of this variable. The first is the name of a parent move object, or "void" if none. If given, the object's transformation will be prepended to that of its parent. The second argument is the name of this object, which will be used to name surfaces it contains, and as a parent identifier for any child objects that reference it. The third argument is the transformation string or file for this object. If this argument is enclosed in quotes and begins with a hyphen ('-'), then it will be interpreted as a static transform specification a la *xform(1)*. Otherwise, the argument will be taken as the name of a file that contains one such transform specification per line, corresponding to frames in the animation. A period ('.') may be given if no object transformation is desired. The fourth argument is the name of a *RADIANCE* scene file (or files) to be given to *xform* for transformation. If this argument begins with an exclamation point ('!'), then it will be interpreted as a command rather than a file. A final word corresponding to the frame number will be appended to the command, and its output will be passed to the input of *xform* for each frame. An optional fifth argument specifies the rendering priority for this object. Values greater than 1.0 will result in preferential rendering of this object over other portions of the image when it appears in a frame. Values less than 1.0 will cause the rendering to neglect this object in favor of other parts of the image. A value of 3.0 can be interpreted as saying the viewer is three times more likely to look at this object than the background. A file may be given rather than a floating point value, and this file must contain one floating point number per line, corresponding to priorities for frames in the animation.

**VIEWFILE**
This variable names a file from which *ranimove* may extract the view for each frame in the animation. This file should contain one valid view per frame, starting with frame 1 on line 1. An exception is made for a view file with only a single view, which is used for every frame of the animation. In this case, the *END* variable must also be specified. This variable is required, and there is no default value.

**END** The final frame number in the animation. The default value is computed from the number of views in the given *VIEWFILE*. Normally, this variable will only be given if the view is static.

**EXPOSURE**
This variable tells *ranimate* how to adjust the exposure for each frame. As in *pfilt,* the exposure setting may be given either as a multiplier or as a number of f–stop adjustments (eg. +2 or –1.5). Alternatively, a file name may be given, which *ranimate* will interpret as having one exposure value per line per frame, beginning with frame 1 at line 1. (See also the *VIEWFILE* variable, above.) There is no default value for this variable. If it is not given, no exposure adjustments will be made.

**BASENAME**
The base output file name for the final frames. This string should contain a *printf(3)* style integer field to distinguish one frame number from another. The final frames will use this name with a ".hdr" suffix. The default value is "frame%03d".

**MBLUR** This variable specifies the fraction of a frame time that the shutter is simulated as being open for motion blur. Motion blur is computed by *ranimove* using image-based rendering methods in a rather crude fashion. A much better result is achieved by setting *MBLUR=0* and using the *pmblur2(1)* command as a post-process, assigning the *ZNAME* and *MNAME* variables described below. The default motion blur is 0.

**RATE** This variable specifies the animation frame rate, in frames per second. This is needed to compute the animation error visibility. The default value is 8.

**RESOLUTION**
This variable specifies the desired final picture resolution. If only a single number is given, this value will be used for both the horizontal and vertical picture dimensions. If two numbers are given, the first is the horizontal resolution and the second is the vertical resolution. If three numbers are given, the third is taken as the pixel aspect ratio for the final picture (a real value). If the pixel aspect ratio is zero, the exact dimensions given will be those produced. Otherwise, they will be used as a frame in which the final image must fit. The default value for this variable is 640.

**lowq** This variable may be used to specify rendering options for the initial, low-quality ray samples. It may be given either as a list of rendering parameter settings, or as variable settings for the *rad* command, in which case the *RIF* variable must also be specified.

**highq** This variable may be used to specify rendering options for the final, high-quality ray samples. It may be given either as a list of rendering parameter settings, or as variable settings for the *rad* command, in which case the *RIF* variable must also be specified.

**oconv** This variable may be used to specify special options for *oconv*. See the *oconv(1)* manual page for a list of valid options. (The *–f* option is specified by default.)

**ZNAME** The base file name for the depth buffer output frames. This string should contain a *printf(3)* style integer field to distinguish one frame number from another. The final frames will use this name with a ".zbf" suffix. This is needed by the *pmblur2* command to compute a better motion blur. There is no default setting for this variable.

**MNAME** The base file name for the motion vector output frames. This string should contain a *printf(3)* style integer field to distinguish one frame number from another. The final frames will use this name with a ".mvo" suffix. This file is used together with the image and depth buffer by *pmblur2* to compute a better motion blur. In this case, the *MBLUR* variable should be unset or zero. The file will contain 3 16-bit unsigned values per pixel. The first two are the x and y offsets to each pixel in the previous frame, with an offset of 32768 being no movement. The third value is 0 if the previous frame's pixel was on a different object, and 32768 if it was on the same object. There is no default setting for this variable.

## EXAMPLES

A minimal input file for *ranimove* might look like this:

```
::::::::::
sample.rnm
::::::::::
# The rad input file for our static scene:
RIF= tutor.rif
# The view file containing one view per frame:
VIEWFILE= anim1.vf
# Our central character and its motion:
move= void myguy myguy.xf myguy.rad 2.0
```

Note that most of the variables are not set in this file. If we only want to see what default values *ranimove* would use without actually executing anything, we can invoke it thus:

```
ranimove –n 0 –e sample.rnm
```

This will print the variables we have given as well as default values *ranimove* has assigned for us.

Usually, we execute *ranimove* in the background, redirecting the standard output and standard error to a file:

```
ranimove sample.rnm >& sample.err &
```

If we decide that the default values *ranimove* has chosen for our variables are not all appropriate, we can add some more assignments to the file:

```
RES= 1024                    # shoot for 1024x resolution
MBLUR= .25                   # apply camera motion blur
RATE= 15                     # 15 frames/second
EXP= anim1.exp               # adjust exposure according to file
lowq= QUAL=Low                     # low quality ray sampling
highq= QUAL=Med                    # high quality ray sampling
```

Note the use of abbreviation for variable names.

## AUTHOR

Greg Ward

## SEE ALSO

fieldcomb(1), oconv(1), pfilt(1), pinterp(1), pmblur2(1), pvalue(1), rad(1), ran2tiff(1), ranimate(1), rpict(1), xform(1)

## NAME

rcalc - record calculator

## SYNOPSIS

**rcalc** [ **−b** ][ **−l** ][ **−p** ][ **−n** ][ **−w** ][ **−u** ][ **−tS** ][ **−i format** ][ **−o format** ][ **−f source** ][ **−e expr** ][ **−s svar=sval** ] file ..

## DESCRIPTION

*Rcalc* transforms ‘‘records’’ from each *file* according to the given set of literal and relational information. By default, records are separated by newlines, and contain numeric fields separated by tabs. The *−tS* option is used to specify an alternate tab character.

A *−i format* option specifies a template for an alternate input record format. *Format* is interpreted as a specification string if it contains a dollar sign ’$’. Otherwise, it is interpreted as the name of the file containing the format specification. In either case, if the format does not end with a newline, one will be added automatically. A special form of the *−i* option may be followed immediately by a ’d’ or an ’f’ and an optional count, which defaults to 1, indicating the number of double or float binary values to read per record on the input file. If the input is byte-swapped, the *−iD* or *−iF* options may be substituted. If binary input is specified, no format string or file is needed.

A *−o format* option specifies an alternate output record format. It is interpreted the same as an input specification, except that the special *−od* or *−of* options do not require a count, as this will be determined by the number of output channels in the given expressions. If byte-swapped output is desired, the *−oD* or *−oF* options may be substituted.

The *−p* option specifies "passive mode," where characters that do not match the input format are passed unaltered to the output. This option has no effect unless *−i* is also specified, and does not make much sense unless *−o* is also given. With both input and output formats, the passive mode can effectively substitute information in the middle of a file or stream without affecting the rest of the data.

The variable and function definitions in each *−f source* file are read and compiled from the RADIANCE library where it is found. The *−e expr* option can be used to define variables on the command line. Since many of the characters in an expression have special meaning to the shell, it should usually be enclosed in single quotes. The *−s svar=sval* option can be used to assign a string variable a string value. If this string variable appears in an input format, only records with the specified value will be processed.

The *−b* option instructs the program to accept only exact matches. By default, tabs and spaces are ignored except as field separators. The *−l* option instructs the program to ignore newlines in the input, basically treating them the same as tabs and spaces. Normally, the beginning of the input format matches the beginning of a line, and the end of the format matches the end of a line. With the *−l* option, the input format can match anywhere on a line.

The *−w* option causes non-fatal error messages (such as division by zero) to be supressed. The *−u* option causes output to be flushed after each record. The *−n* option tells the program not to get any input, but to produce a single output record. Otherwise, if no files are given, the standard input is read.

Format files associate names with string and numeric fields separated by literal information in a record. A numeric field is given in a format file as a dollar sign, followed by curly braces enclosing a variable name:

This is a numeric field: ${vname}

A string variable is enclosed in parentheses:

This is a string field: $(sname)

The program attempts to match literal information in the input format to its input and assign string and numeric fields accordingly. If a string or numeric field variable appears more than once in the input format, input values for the corresponding fields must match (ie. have the same value) for the whole record to match. Numeric values are allowed some deviation, on the order of 0.1%, but string variables must match exactly. Thus, dummy variables for "don’t care" fields should be given unique names so that they are not all required to take on the same value.

For each valid input record, an output record is produced in its corresponding format. Output field widths

are given implicitly by the space occupied in the format file, including the dollar sign and braces. This makes it impossible to produce fields with fewer than four characters. If the *−b* option is specified, input records must exactly match the template. By default, the character following each input field is used as a delimiter. This implies that string fields that are followed by white space cannot contain strings with white space. Also, numeric fields followed but not preceded by white space will not accept numbers preceded by white space. Adjacent input fields are advisable only with the *−b* option. Numeric output fields may contain expressions as well as variables. A dollar sign may appear in a literal as two dollar signs ($$).

The definitions specified in *−e* and *−f* options relate numeric output fields to numeric input fields. For the default record format, a field is a variable of the form $N, where N is the column number, beginning with 1. Output columns appear on the left-hand side of assignments, input columns appear on the right-hand side.

A variable definition has the form:

```
var = expression ;
```

Any instance of the variable in an expression will be replaced with its definition.

An expression contains real numbers, variable names, function calls, and the following operators:

```
+ - * / ^
```

Operators are evaluated left to right, except '^', which is right associative. Powers have the highest precedence; multiplication and division are evaluated before addition and subtraction. Expressions can be grouped with parentheses. All values are double precision real.

A function definition has the form:

```
func(a1, a2, ..) = expression ;
```

The expression can contain instances of the function arguments as well as other variables and functions. Function names can be passed as arguments. Recursive functions can be defined using calls to the defined function or other functions calling the defined function.

The variable *cond,* if defined, will determine whether the current input record produces an output record. If *cond* is positive, output is produced. If *cond* is less than or equal to zero, the record is skipped and no other expressions are evaluated. This provides a convenient method for avoiding inappropriate calculations. The following library of pre-defined functions and variables is provided:

Return the value for input column *N*. If an input format is given, using a channel number generates an error.

**in(n)** Return the value for input column *n,* or the number of columns available in this record if *n* is 0. This is an alternate way to get a column value instead of using the $N notation, and is more flexible since it is programmable. This function is disabled if an input format is used.

**if(cond, then, else)**
if cond is greater than zero, then is evaluated, otherwise else is evaluated. This function is necessary for recursive definitions.

**select(N, a1, a2, ..)**
return aN (N is rounded to the nearest integer). This function provides array capabilities. If *N* is zero, the number of available arguments is returned.

**rand(x)** compute a random number between 0 and 1 based on x.

**floor(x)** return largest integer not greater than x.

**ceil(x)** return smallest integer not less than x.

**sqrt(x)** return square root of x.

**exp(x)** compute e to the power of x (e approx = 2.718281828).

**log(x)** compute the logarithm of x to the base e.

**log10(x)** compute the logarithm of x to the base 10.

**PI** the ratio of a circle's circumference to its diameter.

**recno** the number of records recognized thus far.

**outno** the number or records output thus far (including this one).

**sin(x), cos(x), tan(x)**
trigonometric functions.

**asin(x), acos(x), atan(x)**
inverse trigonometric functions.

**atan2(y, x)** inverse tangent of y/x (range –pi to pi).

## EXAMPLE

To print the square root of column two in column one, and column one times column three in column two:

```
rcalc –e '$1=sqrt($2);$2=$1*$3' inputfile > outputfile
```

## ENVIRONMENT

RAYPATH the directories to check for auxiliary files.

## AUTHOR

Greg Ward

## BUGS

String variables can only be used in input and output formats and –*s* options, not in definitions.

Tabs count as single spaces inside fields.

## SEE ALSO

cnt(1), ev(1), getinfo(1), icalc(1), rcollate(1), rlam(1), tabfunc(1), total(1)

## NAME

rcollate - resize or transpose matrix data

## SYNOPSIS

**rcollate** [ **−h** ][ **−w** ][ **−f[afdb][N]]** ][ **−t** ][ **−ic in_col** ][ **−ir in_row** ][ **−oc out_col** ][ **−or out_row** ] [ **input.dat** ]

## DESCRIPTION

*Rcollate* reads in a single matrix file (table) and reshapes it to have the number of columns specified by the *−oc* option. By default, the file is assumed to include an information header, which is copied to the standard output along with the command name, but the *−h* option may be used to turn this behavior off. The *−w* option turns off non-fatal warning messages, such as unexpected EOD.

The input format is assumed to be ASCII, with three white-space separated words (typically numbers) in each record. A different input format may be specified with the *−f* option. The suboptions are *−fa, −ff, −fd,* and *−fb* for ASCII, float, double, and binary, respectively. An optional count may be attached to specify the number of data elements per record, which defaults to 1. Thus, the default setting is *−fa3*. Since *rcollate* does not interpret the fields, all binary options of the same length have the same result. On most architectures, *−ff6, −fd3,* and *−fb24* would all be equivalent. Note that the lack of row separators in binary files means that *rcollate* does not actually do anything for binary files unless the transpose option is given, also.

The transpose option, *−t* swaps rows and columns on the input. For binary files, the user must specify at least one input or output dimension to define the matrix size. For ASCII files, *rcollate* will automatically determine the number of columns based on the position of the first EOL (end-of-line) character, and the number of rows based on the total count of records in the file. The user may override these determinations, allowing the matrix to be resized as well as transposed. If input and output dimensions are given, the number of input rows must equal the number of output columns, and the number of input columns must equal the number of output rows. For large transpose operations on Unix systems, it is most efficient to specify the input file on the command line, rather than reading from the standard input, since *rcollate* can map the file directly into virtual memory.

## EXAMPLE

To change put 8760 color triplets per row in a matrix with no header:

```
rcollate -h −oc 8760 input.dat > col8760.dat
```

To transpose a binary file with 145 float triplets per input row:

```
rcollate -ff3 -ic 145 -t orig.flt > transpose.flt
```

## AUTHOR

Greg Ward

## NOTES

The *rcollate* command is rather inflexible when it comes to output field and record separators for ASCII data. It accepts any amount of white space between fields on input, but only produces spaces as field separators between words and tabs as record separators on output. Output row separtors will always be an EOL, which may differ between systems.

If no options are given on the command line, or a binary file is specified without a transpose, *rcollate* issues a warning and simply copies its input to its standard output.

## SEE ALSO

cnt(1), histo(1), neaten(1), rcalc(1), rlam(1), tabfunc(1), total(1)

## NAME

rcontrib - compute contribution coefficients in a RADIANCE scene

## SYNOPSIS

**rcontrib** [ **−n nprocs** ][ **−V** ][ **−c count** ][ **−fo** | **−r** ][ **−e expr** ][ **−f source** ][ **−o ospec** ][ **−b binv** ][ **−bn nbins** ] { **−m mod** | **−M file** } [ **$EVAR** ] [ **@file** ] [ rtrace options ] **octree**
**rcontrib [ options ] −defaults**

## DESCRIPTION

*Rcontrib* computes ray coefficients for objects whose modifiers are named in one or more *−m* settings. These modifiers are usually materials associated with light sources or sky domes, and must directly modify some geometric primitives to be considered in the output. A modifier list may also be read from a file using the *−M* option. The RAYPATH environment variable determines directories to search for this file. (No search takes place if a file name begins with a '.', '/' or '~' character.)

If the *−n* option is specified with a value greater than 1, multiple processes will be used to accelerate computation on a shared memory machine. Note that there is no benefit to using more processes than there are local CPUs available to do the work, and the *rcontrib* process itself may use a considerable amount of CPU time.

By setting the boolean *−V* option, you may instruct *rcontrib* to report the contribution from each material rather than the ray coefficient. This is particularly useful for light sources with directional output distributions, whose value would otherwise be lost in the shuffle. With the default *-V-* setting, the output of rcontrib is a coefficient that must be multiplied by the radiance of each material to arrive at a final contribution. This is more convenient for computing daylight coefficeints, or cases where the actual radiance is not desired. Use the *-V+* setting when you wish to simply sum together contributions (with possible adjustment factors) to obtain a final radiance value. Combined with the *−i* or *−I* option, irradiance contributions are reported by *−V+* rather than radiance, and *−V-* coefficients contain an additonal factor of PI.

The *−c* option tells *rcontrib* how many rays to accumulate for each record. The default value is one, meaning a full record will be produced for each input ray. For values greater than one, contributions will be averaged together over the given number of input rays. If set to zero, only a single record will be produced at the very end, corresponding to the sum of all rays given on the input (rather than the average). This is equivalent to passing all the output records through a program like *total(1)* to sum RGB values together, but is much more efficient. Using this option, it is possible to reverse sampling, sending rays from a parallel source such as the sun to a diffuse surface, for example. Note that output flushing via zero-direction rays is disabled with *−c* set to zero.

The output of *rcontrib* has many potential uses. Source contributions can be used as components in linear combination to reproduce any desired variation, e.g., simulating lighting controls or changing sky conditions via daylight coefficients. More generally, *rcontrib* can be used to compute arbitrary input-output relationships in optical systems, such as luminaires, light pipes, and shading devices.

*Rcontrib* sends the accumulated rays tallies to one or more destinations according to the given *−o* specification. If a destination begins with an exclamation mark ('!'), then a pipe is opened to a command and data is sent to its standard input. Otherwise, the destination is treated as a file. An existing file of the same name will not be clobbered, unless the *−fo* option is given. If instead the *−r* option is specified, data recovery is attempted on existing files. (If *−c 0* is used together with the *−r* option, existing files are read in and new ray evaluations are added to the previous results, providing a convenient means for progressive simulation.) If an output specification contains a "%s" format, this will be replaced by the modifier name. The *−b* option may be used to further define a "bin number" within each object if finer resolution is needed, and this will be applied to a "%d" format in the output file specification if present. The actual bin number is computed at run time based on ray direction and surface intersection, as described below. The number of bins must be specified in advance with the *−bn* option, and this is critical for output files containing multiple values per record. A variable or constant name may be given for this parameter if it has been defined via a previous *−f* or *−e* option. Since bin numbers start from zero, the bin count is always equal to the last bin plus one. Set the this value to zero if the bin count is unknown (the default). The most recent *−b, −bn* and *−o* options to the left of each *−m* setting are the ones used for that modifier. The ordering of other options is unimportant, except for *−x* and *−y* if the *−c* is zero, when they control the resolution string

produced in the corresponding output.

If a *−b* expression is defined for a particular modifier, the bin number will be evaluated at run time for each ray contribution. Specifically, each ray's world intersection point will be assigned to the variables Px, Py, and Pz, and the normalized ray direction will be assigned to Dx, Dy, and Dz. These parameters may be combined with definitions given in *−e* arguments and files read using the *−f* option. The computed bin value will be rounded to the nearest whole number. This mechanism allows the user to define precise regions or directions they wish to accumulate, such as the Tregenza sky discretization, which would be otherwise impossible to specify as a set of RADIANCE primitives. The rules and predefined functions available for these expressions are described in the *rcalc(1)* man page. Unlike *rcalc, rcontrib* will search the RADIANCE library directories for each file given in a *−f* option.

If no *−o* specification is given, results are written on the standard output in order of modifier (as given on the command line) then bin number. Concatenated data is also sent to a single destination (i.e., an initial *−o* specification without formatting strings). If a "%s" format appears but no "%d" in the *−o* specification, then each modifier will have its own output file, with multiple values per record in the case of a non-zero *−b* definition. If a "%d" format appears but no "%s", then each bin will get its own output file, with modifiers output in order in each record. For text output, each RGB coefficient triple is separated by a tab, with a newline at the end of each ray record. For binary output formats, there is no such delimiter to mark the end of each record.

Input and output format defaults to plain text, where each ray's origin and direction (6 real values) are given on input, and one line is produced per output file per ray. Alternative data representations may be specified by the *−f[io]* option, which is described in the *rtrace* man page along with the associated *−x* and *−y* resolution settings. In particular, the color ('c') output data representation together with positive dimensions for *−x* and *−y* will produce an uncompressed RADIANCE picture, suitable for manipulation with *pcomb(1)* and related tools.

Options may be given on the command line and/or read from the environment and/or read from a file. A command argument beginning with a dollar sign ('$') is immediately replaced by the contents of the given environment variable. A command argument beginning with an at sign ('@') is immediately replaced by the contents of the given file.

## EXAMPLES

To compute the proportional contributions from sources modified by "light1" vs. "light2" on a set of illuminance values:

```
rcontrib −I+ @render.opt −o c_%s.dat −m light1 −m light2 scene.oct < test.dat
```

To generate a pair of images corresponding to these two lights' contributions:

```
vwrays −ff −x 1024 −y 1024 −vf best.vf | rcontrib −ffc ‘vwrays −d −x 1024 −y 1024 −vf best.vf‘ @render.opt −o c_%s.hdr −m light1 −m light2 scene.oct
```

These images may then be recombined using the desired outputs of light1 and light2:

```
pcomb −c 100 90 75 c_light1.hdr −c 50 55 57 c_light2.hdr > combined.hdr
```

To compute an array of illuminance contributions according to a Tregenza sky:

```
rcontrib −I+ −f tregenza.cal −b tbin −bn Ntbins −o sky.dat −m skyglow −b 0 −o ground.dat −m groundglow @render.opt scene.oct < test.dat
```

## ENVIRONMENT

RAYPATH path to search for −f and −M files

## AUTHOR

Greg Ward

## SEE ALSO

cnt(1), genklemsamp(1), getinfo(1), pcomb(1), pfilt(1), ra_rgbe(1), rcalc(1), rpict(1), rsensor(1), rtrace(1), total(1), vwrays(1), ximage(1)

## NAME

replmarks - replace triangular markers in a RADIANCE scene description

## SYNOPSIS

**replmarks** [ **−e** ][ **−m newmod** ][ **−s scale** ] { **−x objfile** | **−i octree** | **−I mesh** } **modname ..** [ **file ..** ]

## DESCRIPTION

*Replmarks* replaces triangular markers identified by the modifier *modname* in each scene description *file* and writes the result to the standard output. The *−x* option indicates that each marker should be replaced by an appropriate *xform(1)* command on *objfile*. The *−i* option indicates that each marker should be replaced by an instance of *octree*. The *−I* option indicates that each marker should be replaced by an instance of the Radiance mesh *mesh*. One of these three options must appear on the command line, along with *modname,* the modifier used by markers in the file.

Multiple modifiers may be given, as long as each one is preceded by its own *−x* or *−i* option.

The transformation for each marker is determined by its location and orientation. A marker should be a right triangle pointing like a half-arrow in the direction of the transformed x-axis, x'. The longest side is the hypoteneuse, the second longest side is the x'-axis, and the third longest side indicates the direction of the y'-axis. Any additional sides will be ignored (ie. a quadrilateral may be used instead of a triangle if the extra side is small). The z'-axis is determined by the cross product of the x' and y' axes, and the origin is the common vertex between x' and y'.

The size of the marker is ignored unless the *−s* option is used, where *scale* is a multiplier for the x'-axis length to indicate the total scale factor. For example, a *scale* value of 5 with a marker length of .5 would result in a total scale factor of 2.5 to be used in the transformation.

The *−e* option causes commands in the file to be expanded, and is required to replace markers from commands in the input file. Even with this option, *replmarks* will not examine objects for markers. Specifically, an object included by *replmarks* as a result of a *−x* expansion will be transferred verbatim, without regard to any surfaces therein that might have been considered as marks if they were on the main input.

The *−m* option causes all replaced objects to be given the modifier *newmod*. Otherwise, the new object surfaces will use their originally defined modifiers. A different replacement modifier may be given for each marker type. The marker modifier name itself is only used to identify markers, and will not appear in the output in any form.

If no input *file* is given, the standard input is read.

## EXAMPLE

To replace all polygons with the modifier "knobs" in the file input with a transformed "knob.rad" and write the result to output:

```
replmarks −x knob.rad knobs input > output
```

To use instances of "tree.oct" with scaling set to three times the tree marker length:

```
replmarks −s 3 −i tree.oct tree input > output
```

## AUTHOR

Greg Ward

## SEE ALSO

arch2rad(1), ies2rad(1), xform(1)

## NAME

rhcopy - copy ray information into a holodeck

## SYNOPSIS

**rhcopy dest_holo** [ **−u** ][ **−d** ] **−h src_holo ..**
or
**rhcopy dest_holo** [ **−u** ][ **−d** ] **−p src_pic src_zbf ..**

## DESCRIPTION

*Rhcopy* adds ray sample data to the existing holodeck *dest_holo*. In the first form, the ray samples are taken from one or more holodeck files given after the *−h* option. In the second form, the ray samples are taken from one or more RADIANCE picture files and their depth buffers, which must be paired on the command line after the *−p* option.

The *−u* option turns on duplicate ray checking. In some cases, the same ray may already exist in the destination holodeck, and it would be redundant to add it. By default, *rhcopy* does not check for duplicates, because it takes extra time, and in many invocations is not necessary, as when copying into an empty holodeck.

The *−d* option turns off depth checking. Normally, *rhcopy* checks the OBSTRUCTIONS variable of the destination holodeck, and if it is set to True, makes sure that all contributing rays start outside each section. If OBSTRUCTIONS is set to False, then *rhcopy* makes sure that any contributing rays end outside each section. If OBSTRUCTIONS is not set, then this option has no effect. (See the *rholo(1)* man page for a definition of the OBSTRUCTIONS variable.)

*Rcopy* cannot be used to create a holodeck -- use *rholo* for this purpose. For example, to create an empty holodeck, run *rholo* without either the *−n* or *−o* option. Whatever variables are set by *rholo* when the new holodeck is created are the ones that will have effect when later rendering or viewing. Since the ray sample data may be taken from any source, *rholo* and *rhcopy* may be used together to change certain unalterable holodeck parameters, such as the section grid geometry.

## EXAMPLE

To take data from an existing holodeck after changing the section grid:

```
rholo new.hdk new.hif
rhcopy new.hdk −h old.hdk
```

To add ray samples from two pictures to the new holodeck:

```
rhcopy new.hdk −p view1.hdr view1.zbf view2.hdr view2.zbf
```

## NOTES

*Rhcopy* attempts to place the beams in the holodeck in a good order for quick access, but if the data comes from multiple sources, the results may not be optimal. For large holodecks, it is sometimes useful to run the *rhoptimize(1)* program once all runs of *rhcopy* are through.

## AUTHOR

Greg Ward Larson

## ACKNOWLEDGMENT

This work was supported by Silicon Graphics, Inc.

## SEE ALSO

getinfo(1), pfilt(1), psign(1), rhinfo(1), rholo(1), rhoptimize(1), rpict(1)

## NAME

rhinfo - print information about a RADIANCE holodeck file

## SYNOPSIS

**rhinfo input.hdk**

## DESCRIPTION

*Rhinfo* reads the RADIANCE holodeck file *input.hdk* and writes out the header and section information, including the grid sizes, number of beams and a histogram of samples/beam for each section.

## EXAMPLE

To print the header and section information from scene1.hdk:

```
rhinfo scene1.hdk
```

## AUTHOR

Greg Ward Larson

## ACKNOWLEDGMENT

This work was supported by Silicon Graphics, Inc.

## SEE ALSO

getinfo(1), rhcopy(1), rholo(1), rhoptimize(1)

## NAME

rholo - generate/view a RADIANCE holodeck

## SYNOPSIS

**rholo** [ **−n npr** ][ **−o dev** ][ **−w** ][ **−i** ][ **−f** | **−r** ] **hdkfile** [ **varfile** | **+** | **-** [ **VAR=value ..** ] ]

## DESCRIPTION

*Rholo* is a program for generating and viewing holodeck files. Similar to *rvu(1), rholo* can compute views interactively, but unlike *rvu,* it reuses any and all information that was previously computed in this or earlier runs using the given holodeck file, *hdkfile*.

The *−n* option sets the number of *rtrace(1)* processes to start for the calculation. It defaults to zero, which means that no new rays will be calculated. In general, it is unwise to start more processes than there are processors on the system. On a multiprocessing system with 4 or more processors, a value one less than the total should yield optimal interactive rates on a lightly loaded system.

The −o option sets the output device to use for display. Currently, there are at least two display drivers available, *x11* and *glx*. If no output device is specified, then *rholo* will start a global calculation of the holodeck, filling it in as time goes by. The quality of the final holodeck will depend on how long *rholo* runs before it is interrupted or runs out of file space or time, according to the variable settings described in the control variable section, below. If no output device and no processes are specified, *rholo* creates an empty holodeck using the given *varfile,* if present.

The *−i* option provides for reading from the standard input. Without a display driver, the input should consist only of views, which will be used to limit which parts of the holodeck are rendered in a batch calculation. With a display driver, most of the commands understood by the driver can be issued either from the operating window or the standard input. These commands are described together with their window equivalents in the display driver section following the control variable section.

The *−f* option permits the given holodeck to be clobbered. Without this option, giving both the holodeck file and a variable file (or "-") will result in an error message if the holodeck exists, since giving both implies that a new holodeck is being created. (When reusing an existing holodeck, the variable values are taken from the holodeck header, though some may be overriden by giving a "+" in place of the variable file.) Also, attempts to clear the holodeck using the interactive "clobber" command will be permitted only if the *−f* option is given on the initial command line.

The *−r* option tells *rholo* to open the holodeck file read-only, which is the default if there are no ray calculation processes. If one or more *rtrace* processes are started with the *−n* option and the *−r* option is given or the specified holodeck is not writable by the user, then any additional rays computed during the session will be discarded rather than saved to the holodeck file.

One or more holodeck section boundaries are defined along with other parameters in the holodeck file or, if the holodeck is being created, the *rholo* control variable file, *varfile*. These section boundaries define where you may move, or at least, where you will be able to see, since they determine where computed rays are stored. Additional variable settings may be added or overridden on the command line following *varfile*. If no *varfile* is needed, a holodeck may still be created by giving a "-" on the command line in place of the variable file. If you wish to override some of the variable settings in an existing holodeck, use a "+", followed by the new settings on the command line. Upper case variables specified more than once will result in a warning message (unless the *−w* option is present), and the last value given will be the one used, unless it would conflict with something in an existing holodeck that cannot be changed, such as the section boundaries. Changing section boundaries requires creating a new holodeck using *rholo* without a *−n* or *−o* option, then running *rhcopy(1)* to fill the new holodeck with the old holodeck's contents.

The *−w* option turns off warnings about multiply and misassigned variables.

Rendering variable assignments appear one per line in *varfile*. The name of the variable is followed by an equals sign ('=') and its value(s). The end of line may be escaped with a backslash ('\'), though it is not usually necessary. Variables that should have only one value are given in upper case. Variables that may have multiple values are given in lower case. Variables may be abbreviated by their first three letters. Comments in *varfile* start with a pound sign ('#') and proceed to the end of line.

## CONTROL VARIABLES

The control variables, their interpretations and default values are given below.

**OCTREE** The name of the octree file. The default name is the same as *hdkfile* but with any suffix replaced by ".oct". This variable may also be read from *rad(1)* if the "RIF" variable is set. (See below.)

**RIF** This variable specifies a *rad* input file to use as a source of rendering options and other variable settings. If given, *rholo* will execute *rad* and get the rendering options to later pass to *rtrace*. Besides prepending the *render* variable, *rholo* will also extract default settings for the common "OCTREE" variable, and the "EYESEP" variable. Following the file name, overriding variable settings may be given, which will be passed to *rad* on the command line. Settings with spaces in them should be enclosed in quotes. The execution of *rad* will also update the contents of the octree, if necessary. There is no default value for this variable.

**EYESEP** The interocular spacing for stereo viewing. I.e., the world distance between the pupils of the left and right eyes. There is no default value for this variable.

**section** A section is a parallelepiped given by an origin and three axis vectors (i.e., 12 floating point values in world coordinates). The axis vectors define the three edges attached to the origin vertex, and the other edges and vertices are determined by the parallel wall constraint. A holodeck section is a region in which the user may freely move about to obtain a view of what is outside that region. In object rendering mode, a section may instead contain a detailed object to be viewed from the outside. The grid dimensions for each axis may also be given by three additional integer arguments following the three axes. Otherwise, if the grid dimensions are left out or any are unspecified or zero, the "GRID" variable will be used to determine them from the section axis lengths. (See below.) There is no default value for this variable, and it is required. If multiple values are given, they will be used for multiple rendering sections, which may or may not be connected, but should generally not overlap. The starting view for interactive display will be the center of the first section facing the positive X direction unless "OBSTRUCTIONS" is set to True, when the view will be placed outside the first section. (See below for this variable's definition.) The third axis of the first section is also used as the default "view up" vector.

**geometry** This variable is used to associate geometry from an octree file with one or more sections. The specified octree will be used by certain drivers (e.g., the "ogl" driver) to display simplified geometry using hardware lighting during motion. If this variable is not set, such drivers will use the main octree file, which contains all the scene geometry. This can be slow if the scene is complex, so use simplified geometry with portals (described below) or specify a non-existent file to turn geometry rendering off. If there is just one setting of this variable, it will be used for all sections. If there are multiple settings, they will correspond to multiple sections.

**portals** This variable is used to associate portal geometry with one or more sections, as required for simplified geometry in some drivers (e.g., "ogl"). The portal geometry itself is given in one or more RADIANCE scene files or quoted commands beginning with an exclamation mark ('!'), and the input may or may not include material definitons. (I.e., the surfaces may be modified by "void" if there are no materials.) A portal is an imaginary surface that intervenes between a view and some detailed geometry not included in the current section. (See the "geometry" variable definition, above.) Portals are often placed in doorways, windows and in front of mirrors. Portal geometry may also be placed around local geometry that has been culled due to its complexity. This specification is necessary in order that the detail geometry be drawn correctly, and that mirrors will work with virtual distances. (See the definition of "VDISTANCE," below.) The orientation of the portal surface geometry is ignored, so they have effect no matter which way they are facing. If there is just one setting of this variable, it will be used for all sections. If there are multiple settings, they will correspond to multiple sections.

**GRID** The default section grid size in world distance units. If any section axis grid is unspecified, the length of the axis will be divided by this number and rounded up to the next larger integer. The grid size is a very important determiner of holodeck performance, since the holodeck beam

index is proportional to average axis grid dimension to the fourth power! If the beam index is too large, poor file and memory performance will result. If the beam index is too small, the holodeck resolution will suffer and objects will tend to break up. In general, the grid size should divide each section wall into 64 or fewer cells for optimal performance. The default value for this variable is the maximum section axis length divided by 8.

**OBSTRUCTIONS**
This boolean variable tells *rholo* whether or not to compute intersections with objects inside holodeck sections. If it is set to "False", then only objects outside the holodeck sections will be visible. This is appropriate when you know all sections to be devoid of geometry, or when some secondary method is available for rendering geometry inside each section. If it is set to "True," all inside geometry will be visible. There is no default for this variable, which means that rays will be started at random points within each holodeck section, allowing interior geometry to be partially sampled.

**VDISTANCE**
This boolean variable determines whether the actual distance to objects is computed, or the virtual distance. If it is set to "True," the virtual distance will be used, which will make reflections and refractions through smooth, flat objects clear, but will blur the boundaries of those objects. Note that some drivers cannot render virtual samples without the proper placement of "portals" in the scene. (See above for the definition of the "portals" variable.) If it is set to "False," the reflections and refractions will be blurred, but object boundaries will remain sharp. The default value for this variable is "False."

**CACHE** The memory cache size to use for ray samples during interactive rendering, in Megabytes. This tuning parameter determines the tradeoff between memory use and disk access time for interactive display. This value will not affect memory use or performance for global holodeck rendering if there is no display process. The default cache is effectively set to 16 Megabytes. If this variable is set to zero, no limit will be placed on memory use and the process will grow to accommodate all the beams that have been accessed.

**DISKSPACE**
Specifies the maximum holodeck file size, in Megabytes. Once the holodeck file reaches this size, *rtrace* will exit. If there is no display process, *rholo* will also exit. The default value for this variable is 0, which is interpreted as no size limit.

**TIME** Sets the maximum time to run rtrace, in decimal hours. After this length of time, *rtrace* will exit. If there is no display process, *rholo* will also exit. If there is a display process, and *rtrace* is restarted with the "restart" command, then the time clock will be restarted as well. The default value for this variable is 0, which is interpreted as no time limit.

**REPORT** This variable may be used to specify a interval for progress reports in minutes. If this value is zero, then progress reports will not be given in intervals, but a final report of the file size and fragmentation will be issued when the program terminates, along with the number of rays and packets computed. If a filename is given after the interval, it will be used as the error file for reports and error messages instead of the standard error. There is no default value for this variable.

**render** This variable may be used to specify additional options to *rtrace*. These options will appear after the options set automatically by *rad,* and thus will override the default values.

## DISPLAY DRIVER

*Rholo* may be started in interactive mode using the *−o* option to specify an output display driver. Currently, three drivers are supported on most machines, *glx, ogl* and *x11*. (In addition, there are variations on the first two drivers for stereo displays, local objects and human tone mapping. These are accessed with some combination of the 's', 'o' and 'h' suffixes, always in that order. E.g., the OpenGL stereo driver with human tone mapping would be "oglsh".) Each driver accepts simple one-character commands and mouse view control in its operating window. If the *−i* option is also given, then the driver will also listen for commands entered on the standard input. (It is unwise to use the *−i* option when rholo is run in the

background, because it will occassionally stop the process when input is available on the controlling terminal.) The commands and their single-key window equivalents are given below.

**VIEW= (mouse)**

Modify the current view with the specified parameters. (See the *−v** view options in the *rpict(1)* manual page for parameter details.) There is no one-character equivalent for this command in the display window. Instead, the mouse is used to control the current view in the following ways:

| CONTROL | | MOUSE ACTION |
|---|---|---|
| (none) | left | Move forward towards cursor position |
| (none) | middle | Rotate in place (usually safe) |
| (none) | right | Move backward away from cursor position |
| shift | left | Orbit left around cursor position |
| shift | middle | Orbit skyward |
| cntl | middle | Orbit earthward |
| shift | right | Orbit right around cursor position |
| cntl+shift | any | Frame focus by dragging rectangle |

For all movements but rotating in place, the cursor must be placed over some bit of visible geometry, otherwise the driver has no reference point from which to work. It is best to just experiment with these controls until you learn to fly safely in your model. And if you run into trouble, the "last" command is very useful. (See below.)

**last 'l'** Return to the previous view. Some drivers will save up multiple views in a history, but you are guaranteed at least one.

**where 'v'**

Print the current view parameters to the standard output. This is useful for finding out where you are, or for saving specific views in a keyframe file for animations or returning to later.

**frame 'f'**

Change the calculation focus. If the "frame" command is given with no arguments on the standard input, it is equivalent to the interactive 'F' command, which releases the current calculation focus. If the "frame" command is followed by a relative horizontal and vertical position (specified as floating point values between 0 and 1), then the new focus is about this position on the screen (where 0 0 is at the lower left of the display). This is equivalent to the interactive 'f' command, which sets the focus about the current window cursor position. If four relative coordinates are given, they are assumed to mean the minimum horizontal and vertical positon, and the maximum horizontal and vertical position, in that order. This is equivalent to dragging the mouse over a rectangular area with the 'cntl+shift' keys held down.

**pause 'p'**

Pause the ray calculation temporarily.

**resume <cr>**

Resume the ray calculation.

**redraw ^L**

Redraw the current view from values calculated and stored in the holodeck. When executed from the display window via '^L', the effect may be slightly different, since all stored information will be flushed.

**kill 'K'** Terminate the ray calculation process. This is usually unnecessary, but is provided for special purpose applications.

**restart 'R'**

Restart the ray calculation process. If the "RIF" variable has been set, *rad* will be run first to assure that the octree is up to date.

**clobber 'C'**
Clobber the holodeck contents, deleting all that has been calculated before. To get an interactive dissolve of changes to the scene description, use the sequence "kill," "clobber," "restart." This command will be honored by *rholo* only if it was started with the *–f* option.

**quit 'q'** Quit *rholo*. The ray tracing calculation is terminated and all values are flushed to the holodeck file. This is the normal way to exit the program.

In addition to these standard commands, all drivers offer the following supplimentary controls.

**'h'** Fix the head height. All mouse-controlled view motions will be adjusted so that the head height does not change (where vertical is determined by the current view up vector).

**'H'** Release the head height, allowing it to change again during mouse-controlled movements.

**^R** Redraw the current view, recomputing the tone mapping in the process. This is useful if the current view is too light or too dark. (On an 8-bit display, it may be necessary to redraw the screen a couple of times to get the best image.) The "^L" command is a stronger type of redraw, since it will use only rays in the current view to determine the tone mapping, rather than a history of rays drawn from the *rholo* server.

## EXAMPLES

The following shows a minimal holodeck control variable file:

```
RIF= sample.rif                # rad input file
section= 2 2 4  5 0 0  0 7 0  0 0 3   # section parallelepiped origin & vectors
```

Technically, the "RIF" setting is not necessary, but the results are much better when *rholo* is used in association with *rad* to control the rendering parameters.

Here is a slightly more sophisticated example:

```
RIF=electric.rif
section= 7 4 3.5  15 0 0  0 10 0  0 0 5
GRID= .75
CACHE= 20       # cache size in megabytes
TIME= 120       # maximum time in hours
DISK= 200       # maximum file size in megabytes
REPORT= 60 elect.her
OBST= False
VDIST= False
```

We can invoke *rholo* on the above file to compute a hologram overnight in batch mode:

```
rholo –n 1 elect.hdk elect.hif TIME=12 &
```

This will report progress every hour to "elect.her".

The next morning, we can look at the holodeck interactively:

```
rholo–n 1 –o x11 elect.hdk &
```

If the previous command were still running, the above command would fail because the permissions on the holodeck would not grant access. To terminate *rholo* without losing any computed information, use the *kill(1)* command to send an interrupt or terminate signal to the *rholo* process listed by *ps(1)*. If the system goes down or something dire happens to *rholo,* it may be necessary to restore read/write permission on the holodeck using *chmod(1)*. Do not do this, however, unless you are absolutely sure that *rholo* is no longer running on the holodeck. (See the *ps* man page on how to check for running processes. The file modification date as reported by *ls(1)* is another clue.)

To view the holodeck without invoking a new ray calculation, leave off the *–n* option. To compute the holodeck with multiple processes on a multiprocessing system, use a higher number for the *–n* option. (Don't use more processes than you have processors, though, because you'll only slow things down.)

To allow interactive control of *rholo* from another process, the following invocation will override the file size limit and permit the holodeck to be clobbered by a command entered on the standard input:

```
rholo −n 1 −o x11 −i −f elect.hdk + DISK=0
```

To create an empty holodeck from settings on the command line:

```
rholo new.hdk - RIF=sample.rif "section=2 2 4  8 0 0  0 10 0  0 0 3"
```

## NOTES

Each time rays are added to a beam, that beam's position in the holodeck file is released and a new position is found. After substantial computation on a holodeck, especially over several runs, the holodeck file may become fragmented, leaving holes that take up space without contributing useful information. The percentage fragmentation is reported when the REPORT variable is set and some calculation has taken place. When this percentage gets high on a large holodeck (above 15% or so), it is a good idea to run the *rhoptimize(1)* program once batch rendering is complete to close the gaps and collect beams into groups for quicker rendering access. Rholo will print periodic warnings when the fragmentation exceeds 20%.

## AUTHOR

Greg Ward Larson

## ACKNOWLEDGMENT

This work was supported by Silicon Graphics, Inc.

## BUGS

Global participating media are not handled correctly, though local participating media will usually work.

## SEE ALSO

chmod(1), ls(1), ps(1), rad(1), ranimate(1), rhcopy(1), rhinfo(1), rhoptimize(1), rhpict(1), rpict(1), rtrace(1), rvu(1)

## NAME

rhoptimize - optimize beam locations in holodeck file

## SYNOPSIS

**rhoptimize** [ **−u** ] **src_holo** [ **dest_holo** ]

## DESCRIPTION

*Rhoptimize* optimizes beam positions and eliminates fragment waste in the holodeck file *src_holo,* writing the result to *dest_holo,* or back into *src_holo* if only one argument is given. This may improve rendering speed on large holodecks, which tend to have widely dispersed beam positions that cause delays due to long file seeks. It may also reduce the size of the file, since large holodecks become fragmented as they fill up with new ray samples. (Use the *rhinfo(1)* program to determine holodeck file fragmentation.) The *−u* option adds a check to make sure that each ray sample is unique, i.e., the same sample is never given twice. This check is usually unnecessary, but may eliminate redundant samples from some holodeck files.

## EXAMPLE

To optimize the beam order in old.hdk and write the results to new.hdk:

```
rhoptimize old.hdk new.hdk
```

To optimize beam order in scene.hdk:

```
rhoptimize scene.hdk
```

## NOTES

If *rhoptimize* is given only one file argument, it creates a temporary file in the same directory, and moves it onto the original file once it has successfully completed its operation. If the operation fails for some reason, the temporary file is removed and the original holodeck is left unchanged.

This program generally takes several minutes to run and offers no progress reports.

## FILES

rho????.hdk temporary file to hold new holodeck

## AUTHOR

Greg Ward Larson

## ACKNOWLEDGMENT

This work was supported by Silicon Graphics, Inc.

## SEE ALSO

getinfo(1), pfilt(1), psign(1), rhcopy(1), rhinfo(1), rholo(1), rpict(1)

## NAME

rhpict - render a RADIANCE picture from a holodeck file

## SYNOPSIS

**rhpict** [ **options** ] **holodeck**

## DESCRIPTION

*Rhpict* generates one or more pictures from the RADIANCE holodeck file *holodeck* and sends them to the standard output. The *−o* option may be used to specify an alternate output file. Other options specify the viewing parameters and provide some control over the calculation.

The view as well as some of the other controls are shared in common with the *rpict(1)* command. The options that are unique to *rhpict* are given first, followed by the more familiar ones.

**-s** Use the smooth resampling algorithm, which amounts to linear interpolation between ray samples with additional edge detection along color and object boundaries. This is the default.

**-r** *rf* Use random resampling, where *rf* is a fraction from 0 to 1 indicating the desired degree of randomness. A random fraction of 0 is not the same as smooth resampling, because there is no linear interpolation, just Voronoi regions. Values greater than 1 produce interesting underwater effects.

**-x** *res* Set the maximum x resolution to *res*.

**-y** *res* Set the maximum y resolution to *res*.

**-pa** *rat* Set the pixel aspect ratio (height over width) to *rat*. Either the x or the y resolution will be reduced so that the pixels have this ratio for the specified view. If *rat* is zero, then the x and y resolutions will adhere to the given maxima.

**-pe** *expval* Set the exposure value for the output pictures to *expval*. Since filtering is performed by *rhpict,* there is little sense in passing the output through *pfilt(1),* other than changing the exposure. This option eliminates that need. The value may be specified either as a multiplier, or as a number f-stops preceeded by a '+' or '-' character.

**-vt***t* Set view type to *t*. If *t* is 'v', a perspective view is selected. If *t* is 'l', a parallel view is used. A cylindrical panorma may be selected by setting *t* to the letter 'c'. This view is like a standard perspective vertically, but projected on a cylinder horizontally (like a soupcan's-eye view). Three fisheye views are provided as well; 'h' yields a hemispherical fisheye view, 'a' results in angular fisheye distortion, and 's' results in a planisphere (stereographic) projection. A hemispherical fisheye is a projection of the hemisphere onto a circle. The maximum view angle for this type is 180 degrees. An angular fisheye view is defined such that distance from the center of the image is proportional to the angle from the central view direction. An angular fisheye can display a full 360 degrees. A planisphere fisheye view maintains angular relationships between lines, and is commonly used for sun path analysis. This is more commonly known as a "stereographic projection," but we avoid the term here so as not to confuse it with a stereoscopic pair. A planisphere fisheye can display up to (but not including) 360 degrees, although distortion becomes extreme as this limit is approached. Note that there is no space between the view type option and its single letter argument.

**-vp** *x y z* Set the view point to *x y z* . This is the focal point of a perspective view or the center of a parallel projection.

**-vd** *xd yd zd*
Set the view direction vector to *xd yd zd* .

**-vu** *xd yd zd*
Set the view up vector (vertical direction) to *xd yd zd* .

**-vh** *val* Set the view horizontal size to *val*. For a perspective projection (including fisheye views), *val* is the horizontal field of view (in degrees). For a parallel projection, *val* is the view width in world coordinates.

**-vv** *val* Set the view vertical size to *val*.

**-vo** *val* Set the view fore clipping plane at a distance of *val* from the view point. The plane will be perpendicular to the view direction for perspective and parallel view types. For fisheye view types, the clipping plane is actually a clipping sphere, centered on the view point with radius *val*. Objects in front of this imaginary surface will not be visible. This may be useful for seeing through walls (to get a longer perspective from an exterior view point) or for incremental rendering. A value of zero implies no foreground clipping. A negative value produces some interesting effects, since it creates an inverted image for objects behind the viewpoint. This possibility is provided mostly for the purpose of rendering stereographic holograms.

**-va** *val* Set the view aft clipping plane at a distance of *val* from the view point. Like the view fore plane, it will be perpendicular to the view direction for perspective and parallel view types. For fisheye view types, the clipping plane is actually a clipping sphere, centered on the view point with radius *val*. Objects behind this imaginary surface will not be visible. A value of zero means no aft clipping, and is the only way to see infinitely distant objects such as the sky.

**-vs** *val* Set the view shift to *val*. This is the amount the actual image will be shifted to the right of the specified view. This is option is useful for generating skewed perspectives or rendering an image a piece at a time. A value of 1 means that the rendered image starts just to the right of the normal view. A value of −1 would be to the left. Larger or fractional values are permitted as well.

**-vl** *val* Set the view lift to *val*. This is the amount the actual image will be lifted up from the specified view, similar to the −*vs* option.

**-vf** *file* Get view parameters from *file,* which may be a picture or a file created by rvu (with the "view" command).

**-S** *seqstart* Instead of generating a single picture based only on the view parameters given on the command line, this option causes *rhpict* to read view options from the standard input and for each line containing a valid view specification, generate a corresponding picture. *Seqstart* is a positive integer that will be associated with the first output frame, and incremented for successive output frames. By default, each frame is concatenated to the output stream, but it is possible to change this action using the −*o* option (described below). Multiple frames may be later extracted from a single output stream using the *ra_rgbe(1)* command.

**-o** *fspec* Send the picture(s) to the file(s) given by *fspec* instead of the standard output. If this option is used in combination with −*S* and *fspec* contains an integer field for *printf(3)* (eg., "%03d") then the actual output file name will include the current frame number.

**−w** Turn off warning messages.

## EXAMPLE

rhpict −vp 10 5 3 −vd 1 −.5 0 scene.hdk > scene.hdr

rpict −S 1 −o frame%02d.hdr scene.hdk < keyframes.vf

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), pfilt(1), pinterp(1), printf(3), ra_rgbe(1), rholo(1), rpict(1), rvu(1)

## NAME

rlam - laminate records from multiple files

## SYNOPSIS

**rlam** [ **–tS** ][ **–u** ][ **–iaN** | **–ifN** | **–idN** | **–iiN** | **–iwN** | **–ibN** ] input1 input2 ..

## DESCRIPTION

*Rlam* simply joins records (or lines) from multiple inputs, separating them with the given string (TAB by default). Different separators may be given for different files by specifying additional *–t* options in between each file name. Note that there is no space between this option and its argument. If none of the input files uses an ASCII separator, then no end-of-line character will be printed, either.

An input is either a stream or a command. Commands are given in quotes, and begin with an exclamantion point ('!'). If the inputs do not have the same number of lines, then shorter files will stop contributing to the output as they run out.

The *–ia* option may be used to specify ASCII input (the default), or the *–if* option may be used to indicated binary IEEE 32-bit floats on input. Similarly, the *–id* and *–ii* options may be used to indicate binary 64-bit doubles or integer words, respectively. The *–iw* option specifies 2-byte short words, and the *–ib* option specifies bytes. If a number is immediately follows any of these options, then it indicates that multiple such values are expected for each record. For example, *–if3* indicates three floats per input record for the next named input. In the case of the *–ia* option, no number indicates one line per input record, and numbers greater than zero indicate that many characters exactly per record. For binary input formts, no number implies one value per record. For anything other than EOL-separated input, the default tab separator is reset to the empty string.

A hyphen ('-') by itself can be used to indicate the standard input, and may appear multiple times. The *–u* option forces output after each record (i.e., one run through inputs).

## EXAMPLE

To join files output1 and output2, separated by a comma:

```
rlam –t, output1 output2
```

To join a file with line numbers (starting at 0) and its reverse:

```
cnt ‘wc –l < lam.c‘ | rlam – –t: lam.c –t  ’!tail –r lam.c’
```

To join four data files, each having three doubles per record:

```
rlam –id3 file1.dbl file2.dbl file3.dbl file4.dbl > combined.dbl
```

## AUTHOR

Greg Ward

## SEE ALSO

cnt(1), histo(1), neaten(1), rcalc(1), tabfunc(1), total(1)

## NAME

rpict - generate a RADIANCE picture

## SYNOPSIS

**rpict** [ **options** ] [ **$EVAR** ] [ **@file** ] [ **octree** ]
**rpict** [ **options** ] **−defaults**

## DESCRIPTION

*Rpict* generates a picture from the RADIANCE scene given in *octree* and sends it to the standard output. If no *octree* is given, the standard input is read. (The octree may also be specified as the output of a command enclosed in quotes and preceded by a '!'.) Options specify the viewing parameters as well as giving some control over the calculation. Options may be given on the command line and/or read from the environment and/or read from a file. A command argument beginning with a dollar sign ('$') is immediately replaced by the contents of the given environment variable. A command argument beginning with an at sign ('@') is immediately replaced by the contents of the given file.

In the second form shown above, the default values for the options (modified by those options present) are printed with a brief explanation.

Most options are followed by one or more arguments, which must be separated from the option and each other by white space. The exceptions to this rule are the *−vt* option and the boolean options. Normally, the appearance of a boolean option causes a feature to be "toggled", that is switched from off to on or on to off depending on its previous state. Boolean options may also be set explicitly by following them immediately with a '+' or '-', meaning on or off, respectively. Synonyms for '+' are any of the characters "yYtT1", and synonyms for '-' are any of the characters "nNfF0". All other characters will generate an error.

**-vt***t* Set view type to *t*. If *t* is 'v', a perspective view is selected. If *t* is 'l', a parallel view is used. A cylindrical panorma may be selected by setting *t* to the letter 'c'. This view is like a standard perspective vertically, but projected on a cylinder horizontally (like a soupcan's-eye view). Three fisheye views are provided as well; 'h' yields a hemispherical fisheye view, 'a' results in angular fisheye distortion, and 's' results in a planisphere (stereographic) projection. A hemispherical fisheye is a projection of the hemisphere onto a circle. The maximum view angle for this type is 180 degrees. An angular fisheye view is defined such that distance from the center of the image is proportional to the angle from the central view direction. An angular fisheye can display a full 360 degrees. A planisphere fisheye view maintains angular relationships between lines, and is commonly used for sun path analysis. This is more commonly known as a "stereographic projection," but we avoid the term here so as not to confuse it with a stereoscopic pair. A planisphere fisheye can display up to (but not including) 360 degrees, although distortion becomes extreme as this limit is approached. Note that there is no space between the view type option and its single letter argument.

**-vp** *x y z* Set the view point to *x y z* . This is the focal point of a perspective view or the center of a parallel projection.

**-vd** *xd yd zd*
Set the view direction vector to *xd yd zd* . The length of this vector indicates the focal distance as needed by the *−pd* option, described below.

**-vu** *xd yd zd*
Set the view up vector (vertical direction) to *xd yd zd* .

**-vh** *val* Set the view horizontal size to *val*. For a perspective projection (including fisheye views), *val* is the horizontal field of view (in degrees). For a parallel projection, *val* is the view width in world coordinates.

**-vv** *val* Set the view vertical size to *val*.

**-vo** *val* Set the view fore clipping plane at a distance of *val* from the view point. The plane will be perpendicular to the view direction for perspective and parallel view types. For fisheye view types, the clipping plane is actually a clipping sphere, centered on the view point with radius *val*. Objects in front of this imaginary surface will not be visible. This may be useful for

seeing through walls (to get a longer perspective from an exterior view point) or for incremental rendering. A value of zero implies no foreground clipping. A negative value produces some interesting effects, since it creates an inverted image for objects behind the viewpoint. This possibility is provided mostly for the purpose of rendering stereographic holograms.

**-va** *val* Set the view aft clipping plane at a distance of *val* from the view point. Like the view fore plane, it will be perpendicular to the view direction for perspective and parallel view types. For fisheye view types, the clipping plane is actually a clipping sphere, centered on the view point with radius *val*. Objects behind this imaginary surface will not be visible. A value of zero means no aft clipping, and is the only way to see infinitely distant objects such as the sky.

**-vs** *val* Set the view shift to *val*. This is the amount the actual image will be shifted to the right of the specified view. This is option is useful for generating skewed perspectives or rendering an image a piece at a time. A value of 1 means that the rendered image starts just to the right of the normal view. A value of −1 would be to the left. Larger or fractional values are permitted as well.

**-vl** *val* Set the view lift to *val*. This is the amount the actual image will be lifted up from the specified view, similar to the −*vs* option.

**-vf** *file* Get view parameters from *file,* which may be a picture or a file created by rvu (with the "view" command).

**-x** *res* Set the maximum x resolution to *res*.

**-y** *res* Set the maximum y resolution to *res*.

**-pa** *rat* Set the pixel aspect ratio (height over width) to *rat*. Either the x or the y resolution will be reduced so that the pixels have this ratio for the specified view. If *rat* is zero, then the x and y resolutions will adhere to the given maxima.

**-ps** *size* Set the pixel sample spacing to the integer *size*. This specifies the sample spacing (in pixels) for adaptive subdivision on the image plane.

**-pt** *frac* Set the pixel sample tolerance to *frac*. If two samples differ by more than this amount, a third sample is taken between them.

**-pj** *frac* Set the pixel sample jitter to *frac*. Distributed ray-tracing performs anti-aliasing by randomly sampling over pixels. A value of one will randomly distribute samples over full pixels. A value of zero samples pixel centers only. A value between zero and one is usually best for low-resolution images.

**-pm** *frac* Set the pixel motion blur to *frac*. In an animated sequence, the exact view will be blurred between the previous view and the next view as though a shutter were open this fraction of a frame time. (See the −*S* option regarding animated sequences.) The first view will be blurred according to the difference between the initial view set on the command line and the first view taken from the standard input. It is not advisable to use this option in combination with the *pmblur(1)* program, since one takes the place of the other. However, it may improve results with *pmblur* to use a very small fraction with the −*pm* option, to avoid the ghosting effect of too few time samples.

**-pd** *dia* Set the pixel depth-of-field aperture to a diameter of *dia* (in world coordinates). This will be used in conjunction with the view focal distance, indicated by the length of the view direction vector given in the −*vd* option. It is not advisable to use this option in combination with the *pdfblur(1)* program, since one takes the place of the other. However, it may improve results with *pdfblur* to use a very small fraction with the −*pd* option, to avoid the ghosting effect of too few samples.

**-dj** *frac* Set the direct jittering to *frac*. A value of zero samples each source at specific sample points (see the −*ds* option below), giving a smoother but somewhat less accurate rendering. A positive value causes rays to be distributed over each source sample according to its size, resulting in more accurate penumbras. This option should never be greater than 1, and may even cause

problems (such as speckle) when the value is smaller. A warning about aiming failure will issued if *frac* is too large. It is usually wise to turn off image sampling when using direct jitter by setting –ps to 1.

**-ds** *frac* Set the direct sampling ratio to *frac*. A light source will be subdivided until the width of each sample area divided by the distance to the illuminated point is below this ratio. This assures accuracy in regions close to large area sources at a slight computational expense. A value of zero turns source subdivision off, sending at most one shadow ray to each light source.

**-dt** *frac* Set the direct threshold to *frac*. Shadow testing will stop when the potential contribution of at least the next and at most all remaining light source samples is less than this fraction of the accumulated value. (See the –*dc* option below.) The remaining light source contributions are approximated statistically. A value of zero means that all light source samples will be tested for shadow.

**–dc** *frac* Set the direct certainty to *frac*. A value of one guarantees that the absolute accuracy of the direct calculation will be equal to or better than that given in the –*dt* specification. A value of zero only insures that all shadow lines resulting in a contrast change greater than the –*dt* specification will be calculated.

**-dr** *N* Set the number of relays for secondary sources to *N*. A value of 0 means that secondary sources will be ignored. A value of 1 means that sources will be made into first generation secondary sources; a value of 2 means that first generation secondary sources will also be made into second generation secondary sources, and so on.

**-dp** *D* Set the secondary source presampling density to D. This is the number of samples per steradian that will be used to determine ahead of time whether or not it is worth following shadow rays through all the reflections and/or transmissions associated with a secondary source path. A value of 0 means that the full secondary source path will always be tested for shadows if it is tested at all.

**–dv** Boolean switch for light source visibility. With this switch off, sources will be black when viewed directly although they will still participate in the direct calculation. This option may be desirable in conjunction with the –*i* option so that light sources do not appear in the output.

**-ss** *samp* Set the specular sampling to *samp*. For values less than 1, this is the degree to which the highlights are sampled for rough specular materials. A value greater than one causes multiple ray samples to be sent to reduce noise at a commmesurate cost. A value of zero means that no jittering will take place, and all reflections will appear sharp even when they should be diffuse. This may be desirable when used in combination with image sampling (see –*ps* option above) to obtain faster renderings.

**-st** *frac* Set the specular sampling threshold to *frac*. This is the minimum fraction of reflection or transmission, under which no specular sampling is performed. A value of zero means that highlights will always be sampled by tracing reflected or transmitted rays. A value of one means that specular sampling is never used. Highlights from light sources will always be correct, but reflections from other surfaces will be approximated using an ambient value. A sampling threshold between zero and one offers a compromise between image accuracy and rendering time.

**-bv** Boolean switch for back face visibility. With this switch off, back faces of all objects will be invisible to view rays. This is dangerous unless the model was constructed such that all surface normals face outward. Although turning off back face visibility does not save much computation time under most circumstances, it may be useful as a tool for scene debugging, or for seeing through one-sided walls from the outside.

**-av** *red grn blu*
Set the ambient value to a radiance of *red grn blu* . This is the final value used in place of an indirect light calculation. If the number of ambient bounces is one or greater and the ambient value weight is non-zero (see *-aw* and *-ab* below), this value may be modified by the computed

indirect values to improve overall accuracy.

**-aw** *N* Set the relative weight of the ambient value given with the *-av* option to *N*. As new indirect irradiances are computed, they will modify the default ambient value in a moving average, with the specified weight assigned to the initial value given on the command and all other weights set to 1. If a value of 0 is given with this option, then the initial ambient value is never modified. This is the safest value for scenes with large differences in indirect contributions, such as when both indoor and outdoor (daylight) areas are visible.

**-ab** *N* Set the number of ambient bounces to *N*. This is the maximum number of diffuse bounces computed by the indirect calculation. A value of zero implies no indirect calculation.

**-ar** *res* Set the ambient resolution to *res*. This number will determine the maximum density of ambient values used in interpolation. Error will start to increase on surfaces spaced closer than the scene size divided by the ambient resolution. The maximum ambient value density is the scene size times the ambient accuracy (see the *−aa* option below) divided by the ambient resolution. The scene size can be determined using *getinfo(1)* with the *−d* option on the input octree. A value of zero is interpreted as unlimited resolution.

**-aa** *acc* Set the ambient accuracy to *acc*. This value will approximately equal the error from indirect illuminance interpolation. A value of zero implies no interpolation.

**-ad** *N* Set the number of ambient divisions to *N*. The error in the Monte Carlo calculation of indirect illuminance will be inversely proportional to the square root of this number. A value of zero implies no indirect calculation.

**-as** *N* Set the number of ambient super-samples to *N*. Super-samples are applied only to the ambient divisions which show a significant change.

**-af** *fname* Set the ambient file to *fname*. This is where indirect illuminance will be stored and retrieved. Normally, indirect illuminance values are kept in memory and lost when the program finishes or dies. By using a file, different invocations can share illuminance values, saving time in the computation. Also, by creating an ambient file during a low resolution rendering, better results can be obtained in a second high resolution pass. The ambient file is in a machine-independent binary format which may be examined with *lookamb(1)*.

The ambient file may also be used as a means of communication and data sharing between simultaneously executing processes. The same file may be used by multiple processes, possibly running on different machines and accessing the file via the network (ie. *nfs(4))*. The network lock manager *lockd(8)* is used to insure that this information is used consistently.

If any calculation parameters are changed or the scene is modified, the old ambient file should be removed so that the calculation can start over from scratch. For convenience, the original ambient parameters are listed in the header of the ambient file. *Getinfo(1)* may be used to print out this information.

**-ae** *mod* Append *mod* to the ambient exclude list, so that it will not be considered during the indirect calculation. This is a hack for speeding the indirect computation by ignoring certain objects. Any object having *mod* as its modifier will get the default ambient level rather than a calculated value. Any number of excluded modifiers may be given, but each must appear in a separate option.

**-ai** *mod* Add *mod* to the ambient include list, so that it will be considered during the indirect calculation. The program can use either an include list or an exclude list, but not both.

**-aE** *file* Same as *−ae,* except read modifiers to be excluded from *file*. The RAYPATH environment variable determines which directories are searched for this file. The modifier names are separated by white space in the file.

**-aI** *file* Same as *−ai,* except read modifiers to be included from *file*.

**-me** *rext gext bext*
Set the global medium extinction coefficient to the indicated color, in units of 1/distance (distance in world coordinates). Light will be scattered or absorbed over distance according to this value. The ratio of scattering to total scattering plus absorption is set by the albedo parameter, described below.

**-ma** *ralb galb balb*
Set the global medium albedo to the given value between 0 0 0 and 1 1 1. A zero value means that all light not transmitted by the medium is absorbed. A unitary value means that all light not transmitted by the medium is scattered in some new direction. The isotropy of scattering is determined by the Heyney-Greenstein parameter, described below.

**−mg** *gecc* Set the medium Heyney-Greenstein eccentricity parameter to *gecc*. This parameter determines how strongly scattering favors the forward direction. A value of 0 indicates perfectly isotropic scattering. As this parameter approaches 1, scattering tends to prefer the forward direction.

**−ms** *sampdist*
Set the medium sampling distance to *sampdist,* in world coordinate units. During source scattering, this will be the average distance between adjacent samples. A value of 0 means that only one sample will be taken per light source within a given scattering volume.

**−i** Boolean switch to compute irradiance rather than radiance values. This only affects the final result, substituting a Lambertian surface and multiplying the radiance by pi. Glass and other transparent surfaces are ignored during this stage. Light sources still appear with their original radiance values, though the *−dv* option (above) may be used to override this.

**−u** Boolean switch to control uncorrelated random sampling. When "off", a low-discrepancy sequence is used, which reduces variance but can result in a dithered appearance in specular highlights. When "on", pure Monte Carlo sampling is used in all calculations.

**-lr** *N* Limit reflections to a maximum of *N,* if N is a positive integer. If *N* is zero, then Russian roulette is used for ray termination, and the *-lw* setting (below) must be positive. If N is a negative integer, then this sets the upper limit of reflections past which Russian roulette will be used. In scenes with dielectrics and total internal reflection, a setting of 0 (no limit) may cause a stack overflow.

**-lw** *frac* Limit the weight of each ray to a minimum of *frac*. During ray-tracing, a record is kept of the estimated contribution (weight) a ray would have in the image. If this weight is less than the specified minimum and the *-lr* setting (above) is positive, the ray is not traced. Otherwise, Russian roulette is used to continue rays with a probability equal to the ray weight divided by the given *frac*.

**-S** *seqstart* Instead of generating a single picture based only on the view parameters given on the command line, this option causes *rpict* to read view options from the standard input and for each line containing a valid view specification, generate a corresponding picture. This option is most useful for generating animated sequences, though it may also be used to control rpict from a remote process for network-distributed rendering. *Seqstart* is a positive integer that will be associated with the first output frame, and incremented for successive output frames. By default, each frame is concatenated to the output stream, but it is possible to change this action using the *−o* option (described below). Multiple frames may be later extracted from the output using *ra_rgbe(1)*.

Note that the octree may not be read from the standard input when using this option.

**-o** *fspec* Send the picture(s) to the file(s) given by *fspec* instead of the standard output. If this option is used in combination with *−S* and *fspec* contains an integer field for *printf(3)* (eg. "%03d") then the actual output file name will include the current frame number. *Rpict* will not allow a picture file to be clobbered (overwritten) with this option. If an image in a sequence already exists *(−S* option), *rpict* will skip until it reaches an image that doesn't, or the end of the sequence. This is useful for running rpict on multiple machines or processors to render the

same sequence, as each process will skip to the next frame that needs rendering.

**-r** *fn* Recover pixel information from the file *fn*. If the program gets killed during picture generation, the information may be recovered using this option. The view parameters and picture dimensions are also recovered from *fn* if possible. The other options should be identical to those which created *fn,* or an inconsistent picture may result. If *fn* is identical to the file specification given with the *−o* option, *rpict* will rename the file prior to copying its contents. This insures that the old file is not overwritten accidentally. (See also the *−ro* option, below.)

If *fn* is an integer and the recover option is used in combination with the *−S* option, then *rpict* skips a number of view specifications on its input equal to the difference between *fn* and *seqstart*. *Rpict* then performs a recovery operation on the file constructed from the frame number *fn* and the output file specification given with the *−o* option. This provides a convenient mechanism for recovering in the middle of an aborted picture sequence.

The recovered file will be removed if the operation is successful. If the recover operation fails (due to lack of disk space) and the output file and recover file specifications are the same, then the original information may be left in a renamed temporary file. (See FILES section, below.)

**-ro** *fspec* This option causes pixel information to be recovered from and subsequently returned to the picture file *fspec*. The effect is the same as specifying identical recover and output file names with the *−r* and *−o* options.

**-z** *fspec* Write pixel distances out to the file *fspec*. The values are written as short floats, one per pixel in scanline order, as required by *pinterp(1)*. Similar to the *−o* option, the actual file name will be constructed using *printf* and the frame number from the *−S* option. If used with the *−r* option, *−z* also recovers information from an aborted rendering.

**−P** *pfile* Execute in a persistent mode, using *pfile* as the control file. This option must be used together with *−S,* and is incompatible with the recover option *(−r)*. Persistent execution means that after reaching end-of-file on its input, *rpict* will fork a child process that will wait for another *rpict* command with the same *−P* option to attach to it. (Note that since the rest of the command line options will be those of the original invocation, it is not necessary to give any arguments besides *−P* for subsequent calls.) Killing the process is achieved with the *kill(1)* command. (The process ID in the first line of *pfile* may be used to identify the waiting *rpict* process.) This option may be less useful than the *−PP* variation, explained below.

**−PP** *pfile* Execute in continuous-forking persistent mode, using *pfile* as the control file. The difference between this option and the *−P* option described above is the creation of multiple duplicate processes to handle any number of attaches. This provides a simple and reliable mechanism of memory sharing on most multiprocessing platforms, since the *fork(2)* system call will share memory on a copy-on-write basis. This option may be used with *rpiece(1)* to efficiently render a single image using multiple processors on the same host.

**-t** *sec* Set the time between progress reports to *sec*. A progress report writes the number of rays traced, the percentage completed, and the CPU usage to the standard error. Reports are given either automatically after the specified interval, or when the process receives a continue (−CONT) signal (see *kill(1))*. A value of zero turns automatic reporting off.

**-e** *efile* Send error messages and progress reports to *efile* instead of the standard error.

**−w** Boolean switch for warning messages. The default is to print warnings, so the first appearance of this option turns them off.

## EXAMPLE

```
rpict −vp 10 5 3 −vd 1 −.5 0 scene.oct > scene.hdr

rpict −S 1 −o frame%02d.hdr scene.oct < keyframes.vf
```

## ENVIRONMENT

RAYPATH the directories to check for auxiliary files.

## FILES

| | |
|---|---|
| /tmp/rtXXXXXX | common header information for picture sequence |
| rfXXXXXX | temporary name for recover file |

## DIAGNOSTICS

If the program terminates from an input related error, the exit status will be 1. A system related error results in an exit status of 2. If the program receives a signal that is caught, it will exit with a status of 3. In each case, an error message will be printed to the standard error, or to the file designated by the $-e$ option.

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), lookamb(1), oconv(1), pdfblur(1), pfilt(1), pinterp(1), pmblur(1), printf(3), ra_rgbe(1), rad(1), rtrace(1), rvu(1)

## NAME

rpiece - render pieces of a RADIANCE picture

## SYNOPSIS

**rpiece** [ **−v** ][ **−x xres** ][ **−y yres** ][ **−X xdiv** ][ **−Y ydiv** ][ **−F|R syncfile** ][ **−T timelim** ] [ **$EVAR** ] [ **@file** ] [ rpict options ] **−o picture octree**

## DESCRIPTION

*Rpiece* renders a RADIANCE picture a piece at a time, calling *rpict(1)* to do the actual work. This is useful for running multiple *rpict* processes on cooperating machines to render a single picture, which is a shared file specified with the *−o* option. The overall picture dimensions will be *xres* by *yres* (or smaller, depending on the *−pa* option and other view options), and the picture will be rendered in *xdiv* by *ydiv* pieces.

There are two basic methods for telling *rpiece* which piece(s) of a picture to render. The explicit method is to write on the standard input the *X* and *Y* position of the desired piece(s), where *X* runs from zero to *xdiv−1* and *Y* runs from zero to *ydiv−1*. (The lower left piece of a picture corresponds to (0,0) in this system.) Alternatively, the implicit specification method uses a synchronization file to determine which piece is to be rendered next. Specified with the *−F* option, *syncfile* initially contains the values for *xdiv* and *ydiv,* so the *−X* and *−Y* options are unnecessary. (However, they are used if *syncfile* does not exist.) The first *rpiece* process puts a lock on *syncfile* and modifies its contents before starting work on the first piece of the image. It writes the *X* and *Y* position of the piece it will work on, so the next *rpiece* process to modify *syncfile* will start on the next piece. (When it finishes with its piece, it appends the index to the end of *syncfile.)* This procedure continues until all the pieces are done, at which point all of the *rpiece* processes will terminate.

The *−R* option may be used instead of *−F* if some of the pieces were not properly finished by previous (killed) runs of *rpiece*. This option should be used by at most one *rpiece* process, which must be started first and with *no other rpiece processes running* or else it will rerender the same pieces other processes have begun. Once the recover process is started, you may start other *rpiece* processes using the *−F* option to run simultaneously. If some processes die during execution, leaving one or more half-finished pieces in the picture even though the other processes think the work is done, you may run a single *rpiece* with the *−R* option by itself to repair the holes.

The *−v* flag switches on verbose mode, where *rpiece* reports to the standard output after each piece begins and after each piece is finished.

Options may be given on the command line and/or read from the environment and/or read from a file. A command argument beginning with a dollar sign ('$') is immediately replaced by the contents of the given environment variable. A command argument beginning with an at sign ('@') is immediately replaced by the contents of the given file.

## EXAMPLE

First *rpiece* process is started on the machine "goober":

```
goober% echo 1 8 > syncfile
goober% echo −F syncfile −x 1024 −y 1024 −vf view −o picture octree > args
goober% rpiece @args &
```

Second *rpiece* processes is started on the machine "sucker":

```
sucker% rpiece @args &
```

## NOTES

Due to NFS file buffering, the network lock manager is employed to guarantee consistency in the output file even though non-overlapping writes are used. This would tend to slow the process down if *rpiece* were to wait for this I/O to complete before starting on the next piece, so *rpiece* forks separate processes to hang around waiting for I/O completion. The number of processes thus designated is set by the MAXFORK macro in the program (compiled in the src/util directory). If the fork call is slow on a system, it may actually be better to set MAXFORK to zero. In other cases, the network lock manager may be so slow that this value should be increased to get the best utilization.

The output picture is not run-length encoded, and can be quite large. The approximate size (in kilobytes) can be computed by the simple formula:

filesize = xres*yres/256

Make sure that there is enough space on the filesystem to hold the entire picture before beginning. Once the picture is finished, the *ra_rgbe(1)* program with the −r option may be used to convert to a run−length encoded picture for more efficient storage, although *pfilt(1)* or any of the other Radiance picture filters will do the same thing.

The ALRM signal may be used to gracefully terminate an *rpiece* process after it finishes the current piece. This permits other currently running or subsequently started *rpiece* process(es) to continue rendering the picture without loss. The −*T* option will send the ALRM signal to *rpiece* after the specified number of (decimal) hours. This is the best way to force a time limit on the computation, since information will not be lost, though the process may continue for some time afterwards to finish its current piece.

## BUGS

This program may not work on some systems whose NFS lock manager is unreliable. In particular, some System V derivative UNIX systems often have problems with the network lock manager. If the output is scrambled or rpict aborts with some ambient file related problem, you should just remove the ambient file and go back to normal rendering.

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), pfilt(1), ra_rgbe(1), rpict(1), ximage(1)

## NAME

rsensor - compute sensor signal from a RADIANCE scene

## SYNOPSIS

**rsensor** [ **-n nprocs** ][ **-h** ][ **render options** ] [ **$EVAR** ] [ **@file** ] { [ **-rd nrays** ][ **-dn nsrc** ][ **sensor_view** ] **sensor_file ..** } **octree**
**rsensor** [ **-h** ] { [ **-rd nrays** ][ **sensor_view** ] **sensor_file ..** } **.**
**rsensor [ options ] –defaults**

## DESCRIPTION

*Rsensor* traces rays outward from one or more specified illumination sensors into the RADIANCE scene given by *octree,* sending the computed sensor value to the standard output. (The octree may be given as the output of a command enclosed in quotes and preceded by a '!'.) In the second form, a single period ('.') is given in place of an octree, and the origin and directions of the specified number of rays will be printed on the standard output. If these rays are later traced and added together, the results will sum to a signal proportional to the given sensor distribution. In the third form, the default values for the options (modified by those options present) are printed with a brief explanation.

Options may be given on the command line and/or read from the environment and/or read from a file. A command argument beginning with a dollar sign ('$') is immediately replaced by the contents of the given environment variable. A command argument beginning with an at sign ('@') is immediately replaced by the contents of the given file.

The sensor files themselves will be searched for in the path locations specified by the *RAYPATH* environment variable, similar to other types of Radiance auxiliary files. If the sensor file path begins with '/', '.' or '~', no search will take place. Before each sensor file, a separate view may be specified. In this case, the view origin and direction will correspond to the position and orientation of the sensor, and the view up vector will determine the zero azimuthal direction of the sensor. The fore clipping distance may be used as well, but other view options will be ignored. (See *rpict(1)* for details on how to specify a view.) The actual data contained in the sensor file corresponds to the *SPOT* tab-separated matrix specification, where the column header has "degrees" in the leftmost column, followed by evenly-spaced azimuthal angles. Each row begins with the polar angle, and is followed by the relative sensitivity values for each direction. A low-resolution example of a sensor file is given below:

| degrees | 0 | 90 | 180 | 270 |
|---|---|---|---|---|
| 0 | .02 | .04 | .02 | .04 |
| 45 | .01 | .02 | .01 | .02 |
| 90 | .001 | .002 | .001 | .002 |

As well as different views, the number of samples may be changed between sensors, where the *–rd* option controls the number of ray samples sent at random, and the *–dn* option controls the number of rays sent to each light source per sensor.

The *–h* option toggles header output, which defaults to "on." The *–n* option may be used to specify multiple calculation processes on systems with more than one CPU. For additional options, consult the *rtrace(1)* man page. The final *octree* argument must be given, as the octree cannot be read from the standard input.

## EXAMPLES

To compute values for the same sensor with two different positions:

```
rsensor -ab 2 -vf posA.vf mysens.dat -vf posB.vf mysens.dat scene.oct
```

To generate a set of rays corresponding to a given sensor and compute the resulting signal with rtrace:

```
rsensor -h -vf posC.vf mysens.dat . | rtrace -h scene.oct | total -m
```

## ENVIRONMENT

RAYPATH the directories to check for auxiliary files.

## AUTHOR

Greg Ward for Architectural Energy Corporation

## SEE ALSO

oconv(1), rpict(1), rcontrib(1), rtrace(1)

## NAME

rtrace - trace rays in RADIANCE scene

## SYNOPSIS

**rtrace** [ **options** ] [ **$EVAR** ] [ **@file** ] **octree**
**rtrace** [ **options** ] **–defaults**

## DESCRIPTION

*Rtrace* traces rays from the standard input through the RADIANCE scene given by *octree* and sends the results to the standard output. (The octree may be given as the output of a command enclosed in quotes and preceded by a '!'.) Input for each ray is:

xorg yorg zorg xdir ydir zdir

If the direction vector is (0,0,0), a bogus record is printed and the output is flushed if the *-x* value is one or zero. (See the notes on this option below.) This may be useful for programs that run *rtrace* as a separate process. In the second form, the default values for the options (modified by those options present) are printed with a brief explanation.

Options may be given on the command line and/or read from the environment and/or read from a file. A command argument beginning with a dollar sign ('$') is immediately replaced by the contents of the given environment variable. A command argument beginning with an at sign ('@') is immediately replaced by the contents of the given file. Most options are followed by one or more arguments, which must be separated from the option and each other by white space. The exceptions to this rule are the boolean options. Normally, the appearance of a boolean option causes a feature to be "toggled", that is switched from off to on or on to off depending on its previous state. Boolean options may also be set explicitly by following them immediately with a '+' or '-', meaning on or off, respectively. Synonyms for '+' are any of the characters "yYtT1", and synonyms for '-' are any of the characters "nNfF0". All other characters will generate an error.

**-f***io* Format input according to the character *i* and output according to the character *o*. *Rtrace* understands the following input and output formats: 'a' for ascii, 'f' for single-precision floating point, and 'd' for double-precision floating point. In addition to these three choices, the character 'c' may be used to denote 4-byte floating point (Radiance) color format for the output of values only *(–ov* option, below). If the output character is missing, the input format is used.

Note that there is no space between this option and its argument.

**-o***spec* Produce output fields according to *spec*. Characters are interpreted as follows:

| | |
|---|---|
| o | origin (input) |
| d | direction (normalized) |
| v | value (radiance) |
| V | contribution (radiance) |
| w | weight |
| W | color coefficient |
| l | effective length of ray |
| L | first intersection distance |
| c | local (u,v) coordinates |
| p | point of intersection |
| n | normal at intersection (perturbed) |
| N | normal at intersection (unperturbed) |
| s | surface name |

m modifier name

M material name

~ tilde (end of trace marker)

If the letter 't' appears in *spec,* then the fields following will be printed for every ray traced, not just the final result. If the capital letter 'T' is given instead of 't', then all rays will be reported, including shadow testing rays to light sources. Spawned rays are indented one tab for each level. The tilde marker ('~') is a handy way of differentiating the final ray value from daughter values in a traced ray tree, and usually appears right before the 't' or 'T' output flags. E.g., *−ov~TmW* will emit a tilde followed by a tab at the end of each trace, which can be easily distinguished even in binary output.

Note that there is no space between this option and its argument.

**-te** *mod* Append *mod* to the trace exclude list, so that it will not be reported by the trace option *(−o*t*)*. Any ray striking an object having *mod* as its modifier will not be reported to the standard output with the rest of the rays being traced. This option has no effect unless either the 't' or 'T' option has been given as part of the output specifier. Any number of excluded modifiers may be given, but each must appear in a separate option.

**-ti** *mod* Add *mod* to the trace include list, so that it will be reported by the trace option. The program can use either an include list or an exclude list, but not both.

**-tE** *file* Same as *−te,* except read modifiers to be excluded from *file*. The RAYPATH environment variable determines which directories are searched for this file. The modifier names are separated by white space in the file.

**-tI** *file* Same as *−ti,* except read modifiers to be included from *file*.

**−i** Boolean switch to compute irradiance rather than radiance values. This only affects the final result, substituting a Lambertian surface and multiplying the radiance by pi. Glass and other transparent surfaces are ignored during this stage. Light sources still appear with their original radiance values, though the *−dv* option (below) may be used to override this. This option is especially useful in conjunction with ximage(1) for computing illuminance at scene points.

**−u** Boolean switch to control uncorrelated random sampling. When "off", a low-discrepancy sequence is used, which reduces variance but can result in a brushed appearance in specular highlights. When "on", pure Monte Carlo sampling is used in all calculations.

**−I** Boolean switch to compute irradiance rather than radiance, with the input origin and direction interpreted instead as measurement point and orientation.

**−h** Boolean switch for information header on output.

**-x** *res* Set the x resolution to *res*. The output will be flushed after every *res* input rays if *−y* is set to zero. A value of one means that every ray will be flushed, whatever the setting of *−y*. A value of zero means that no output flushing will take place.

**-y** *res* Set the y resolution to *res*. The program will exit after *res* scanlines have been processed, where a scanline is the number of rays given by the *−x* option, or 1 if *−x* is zero. A value of zero means the program will not halt until the end of file is reached.

If both *−x* and *−y* options are given, a resolution string is printed at the beginning of the output. This is mostly useful for recovering image dimensions with *pvalue(1),* and for creating valid Radiance picture files using the color output format. (See the *−f* option, above.)

**-n** *nproc* Execute in parallel on *nproc* local processes. This option is incompatible with the *−P* and *−PP,* options. Multiple processes also do not work properly with ray tree output using any of the *−o*t** options. There is no benefit from specifying more processes than there are cores available on the system or the *−x* setting, which forces a wait at each flush.

**-dj** *frac* Set the direct jittering to *frac*. A value of zero samples each source at specific sample points (see the *−ds* option below), giving a smoother but somewhat less accurate rendering. A

positive value causes rays to be distributed over each source sample according to its size, resulting in more accurate penumbras. This option should never be greater than 1, and may even cause problems (such as speckle) when the value is smaller. A warning about aiming failure will issued if *frac* is too large.

**-ds** *frac* Set the direct sampling ratio to *frac*. A light source will be subdivided until the width of each sample area divided by the distance to the illuminated point is below this ratio. This assures accuracy in regions close to large area sources at a slight computational expense. A value of zero turns source subdivision off, sending at most one shadow ray to each light source.

**-dt** *frac* Set the direct threshold to *frac*. Shadow testing will stop when the potential contribution of at least the next and at most all remaining light sources is less than this fraction of the accumulated value. (See the *−dc* option below.) The remaining light source contributions are approximated statistically. A value of zero means that all light sources will be tested for shadow.

**−dc** *frac* Set the direct certainty to *frac*. A value of one guarantees that the absolute accuracy of the direct calculation will be equal to or better than that given in the *−dt* specification. A value of zero only insures that all shadow lines resulting in a contrast change greater than the *−dt* specification will be calculated.

**-dr** *N* Set the number of relays for secondary sources to *N*. A value of 0 means that secondary sources will be ignored. A value of 1 means that sources will be made into first generation secondary sources; a value of 2 means that first generation secondary sources will also be made into second generation secondary sources, and so on.

**-dp** *D* Set the secondary source presampling density to D. This is the number of samples per steradian that will be used to determine ahead of time whether or not it is worth following shadow rays through all the reflections and/or transmissions associated with a secondary source path. A value of 0 means that the full secondary source path will always be tested for shadows if it is tested at all.

**−dv** Boolean switch for light source visibility. With this switch off, sources will be black when viewed directly although they will still participate in the direct calculation. This option is mostly for the program *mkillum(1)* to avoid inappropriate counting of light sources, but it may also be desirable in conjunction with the *−i* option.

**-ss** *samp* Set the specular sampling to *samp*. For values less than 1, this is the degree to which the highlights are sampled for rough specular materials. A value greater than one causes multiple ray samples to be sent to reduce noise at a commmesurate cost. A value of zero means that no jittering will take place, and all reflections will appear sharp even when they should be diffuse.

**-st** *frac* Set the specular sampling threshold to *frac*. This is the minimum fraction of reflection or transmission, under which no specular sampling is performed. A value of zero means that highlights will always be sampled by tracing reflected or transmitted rays. A value of one means that specular sampling is never used. Highlights from light sources will always be correct, but reflections from other surfaces will be approximated using an ambient value. A sampling threshold between zero and one offers a compromise between image accuracy and rendering time.

**-bv** Boolean switch for back face visibility. With this switch off, back faces of all objects will be invisible to view rays. This is dangerous unless the model was constructed such that all surface normals face outward. Although turning off back face visibility does not save much computation time under most circumstances, it may be useful as a tool for scene debugging, or for seeing through one-sided walls from the outside.

**-av** *red grn blu*
Set the ambient value to a radiance of *red grn blu* . This is the final value used in place of an indirect light calculation. If the number of ambient bounces is one or greater and the ambient value weight is non-zero (see *-aw* and *-ab* below), this value may be modified by the computed indirect values to improve overall accuracy.

**-aw** *N* Set the relative weight of the ambient value given with the *-av* option to *N*. As new indirect irradiances are computed, they will modify the default ambient value in a moving average, with the specified weight assigned to the initial value given on the command and all other weights set to 1. If a value of 0 is given with this option, then the initial ambient value is never modified. This is the safest value for scenes with large differences in indirect contributions, such as when both indoor and outdoor (daylight) areas are visible.

**-ab** *N* Set the number of ambient bounces to *N*. This is the maximum number of diffuse bounces computed by the indirect calculation. A value of zero implies no indirect calculation.

**-ar** *res* Set the ambient resolution to *res*. This number will determine the maximum density of ambient values used in interpolation. Error will start to increase on surfaces spaced closer than the scene size divided by the ambient resolution. The maximum ambient value density is the scene size times the ambient accuracy (see the *−aa* option below) divided by the ambient resolution. The scene size can be determined using *getinfo(1)* with the *−d* option on the input octree.

**-aa** *acc* Set the ambient accuracy to *acc*. This value will approximately equal the error from indirect illuminance interpolation. A value of zero implies no interpolation.

**-ad** *N* Set the number of ambient divisions to *N*. The error in the Monte Carlo calculation of indirect illuminance will be inversely proportional to the square root of this number. A value of zero implies no indirect calculation.

**-as** *N* Set the number of ambient super-samples to *N*. Super-samples are applied only to the ambient divisions which show a significant change.

**-af** *fname* Set the ambient file to *fname*. This is where indirect illuminance will be stored and retrieved. Normally, indirect illuminance values are kept in memory and lost when the program finishes or dies. By using a file, different invocations can share illuminance values, saving time in the computation. The ambient file is in a machine-independent binary format which can be examined with *lookamb(1)*.

The ambient file may also be used as a means of communication and data sharing between simultaneously executing processes. The same file may be used by multiple processes, possibly running on different machines and accessing the file via the network (ie. *nfs(4))*. The network lock manager *lockd(8)* is used to insure that this information is used consistently.

If any calculation parameters are changed or the scene is modified, the old ambient file should be removed so that the calculation can start over from scratch. For convenience, the original ambient parameters are listed in the header of the ambient file. *Getinfo(1)* may be used to print out this information.

**-ae** *mod* Append *mod* to the ambient exclude list, so that it will not be considered during the indirect calculation. This is a hack for speeding the indirect computation by ignoring certain objects. Any object having *mod* as its modifier will get the default ambient level rather than a calculated value. Any number of excluded modifiers may be given, but each must appear in a separate option.

**-ai** *mod* Add *mod* to the ambient include list, so that it will be considered during the indirect calculation. The program can use either an include list or an exclude list, but not both.

**-aE** *file* Same as *−ae,* except read modifiers to be excluded from *file*. The RAYPATH environment variable determines which directories are searched for this file. The modifier names are separated by white space in the file.

**-aI** *file* Same as *−ai,* except read modifiers to be included from *file*.

**-me** *rext gext bext*
Set the global medium extinction coefficient to the indicated color, in units of 1/distance (distance in world coordinates). Light will be scattered or absorbed over distance according to this value. The ratio of scattering to total scattering plus absorption is set by the albedo parameter, described below.

**-ma** *ralb galb balb*
Set the global medium albedo to the given value between 0 0 0 and 1 1 1. A zero value means that all light not transmitted by the medium is absorbed. A unitary value means that all light not transmitted by the medium is scattered in some new direction. The isotropy of scattering is determined by the Heyney-Greenstein parameter, described below.

**−mg** *gecc* Set the medium Heyney-Greenstein eccentricity parameter to *gecc*. This parameter determines how strongly scattering favors the forward direction. A value of 0 indicates perfectly isotropic scattering. As this parameter approaches 1, scattering tends to prefer the forward direction.

**−ms** *sampdist*
Set the medium sampling distance to *sampdist,* in world coordinate units. During source scattering, this will be the average distance between adjacent samples. A value of 0 means that only one sample will be taken per light source within a given scattering volume.

**-lr** *N* Limit reflections to a maximum of *N,* if N is a positive integer. If *N* is zero or negative, then Russian roulette is used for ray termination, and the *-lw* setting (below) must be positive. If N is a negative integer, then this sets the upper limit of reflections past which Russian roulette will be used. In scenes with dielectrics and total internal reflection, a setting of 0 (no limit) may cause a stack overflow.

**-lw** *frac* Limit the weight of each ray to a minimum of *frac*. During ray-tracing, a record is kept of the estimated contribution (weight) a ray would have in the image. If this weight is less than the specified minimum and the *-lr* setting (above) is positive, the ray is not traced. Otherwise, Russian roulette is used to continue rays with a probability equal to the ray weight divided by the given *frac*.

**-ld** Boolean switch to limit ray distance. If this option is set, then rays will only be traced as far as the magnitude of each direction vector. Otherwise, vector magnitude is ignored and rays are traced to infinity.

**-e** *efile* Send error messages and progress reports to *efile* instead of the standard error.

**−w** Boolean switch to suppress warning messages.

**−P** *pfile* Execute in a persistent mode, using *pfile* as the control file. Persistent execution means that after reaching end-of-file on its input, *rtrace* will fork a child process that will wait for another *rtrace* command with the same *−P* option to attach to it. (Note that since the rest of the command line options will be those of the original invocation, it is not necessary to give any arguments besides *−P* for subsequent calls.) Killing the process is achieved with the *kill(1)* command. (The process ID in the first line of *pfile* may be used to identify the waiting *rtrace* process.) This option may be used with the *−fr* option of *pinterp(1)* to avoid the cost of starting up *rtrace* many times.

**−PP** *pfile* Execute in continuous-forking persistent mode, using *pfile* as the control file. The difference between this option and the *−P* option described above is the creation of multiple duplicate processes to handle any number of attaches. This provides a simple and reliable mechanism of memory sharing on most multiprocessing platforms, since the *fork(2)* system call will share memory on a copy-on-write basis.

## EXAMPLES

To compute radiance values for the rays listed in samples.inp:

```
rtrace −ov scene.oct < samples.inp > radiance.out
```

To compute illuminance values at locations selected with the 't' command of *ximage(1):*

```
ximage scene.hdr | rtrace −h −x 1 −i scene.oct | rcalc −e '$1=47.4*$1+120*$2+11.6*$3'
```

To record the object identifier corresponding to each pixel in an image:

```
vwrays −fd scene.hdr | rtrace −fda `vwrays −d scene.hdr` −os scene.oct
```

To compute an image with an unusual view mapping:

```
cnt 480 640 | rcalc -e 'xr:640;yr:480' -f unusual_view.cal | rtrace -x 640 -y 480 -fac scene.oct > unusual.hdr
```

## ENVIRONMENT

| | |
|---|---|
| RAYPATH | the directories to check for auxiliary files. |

## FILES

| | |
|---|---|
| /tmp/rtXXXXXX | common header information for picture sequence |

## DIAGNOSTICS

If the program terminates from an input related error, the exit status will be 1. A system related error results in an exit status of 2. If the program receives a signal that is caught, it will exit with a status of 3. In each case, an error message will be printed to the standard error, or to the file designated by the *-e* option.

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), lookamb(1), oconv(1), pfilt(1), pinterp(1), pvalue(1), rpict(1), rcontrib(1), rvu(1), vwrays(1), ximage(1)

## NAME

rvu - generate RADIANCE images interactively

## SYNOPSIS

**rvu** [ **rpict options** ][ **−n nproc** ][ **−o dev** ][ **−b** ][ **−pe exposure** ] [ **$EVAR** ] [ **@file** ] **octree**
**rvu** [ **options** ] **−defaults**
**rvu −devices**

## DESCRIPTION

*Rvu* generates RADIANCE images using *octree*. (The octree may be given as the output of a command enclosed in quotes and preceded by a '!'.) Options specify the viewing parameters as well as giving some control over the calculation. Options may be given on the command line and/or read from the environment and/or read from a file. A command argument beginning with a dollar sign ('$') is immediately replaced by the contents of the given environment variable. A command argument beginning with an at sign ('@') is immediately replaced by the contents of the given file. The options are the same as for rpict(1), with a few notable exceptions. The *−pd, −r, −z, −S, −P, −PP* and *−t* options are not supported, and *−o* specifies which output device is being used instead of the output file. The *−x, −y* and *−pa* options are unnecessary, since *rvu* scales the display image to the specified output device. Additionally, the *−b* option improves the display on greyscale monitors, and *−pe* may be used to set an initial exposure value.

The *−n* option may be used to specify multiple processes, to accelerate rendering.

In the second form, the default values for the options are printed with a brief explanation. In the third form, the list of supported output devices is displayed.

*rvu* starts rendering the image from the selected viewpoint and gradually improves the resolution of the display until interrupted by keyboard input. *rvu* then issues a prompt (usually ':') and accepts a command line from the user. *rvu* may also stop its calculation and wait for command input if the resolution of the display has reached the resolution of the graphics device. At this point, it will give the 'done:' prompt and await further instructions. If *rvu* runs out of memory due to lack of resources to store its computed image, it will give the 'out of memory:' prompt. At this prompt, the user can save the image, quit, or even restart a new image, although this is not generally recommended on virtual memory machines for efficiency reasons.

*rvu* is not meant to be a rendering program, and we strongly recommend that *rpict(1)* be used instead for that purpose. Since *rpict(1)* does not store its image in memory or update any display of its output, it is much faster and less wasteful of its resources than *rvu*. *rvu* is intended as a quick interactive program for deciding viewpoints and debugging scene descriptions and is not suited for producing polished images.

## COMMANDS

Once the program starts, a number of commands can be used to control it. A command is given by its name, which can be abbreviated, followed by its arguments.

**aim** *[ mag [ x y z ] ]*
: Zoom in by *mag* on point *x y z* . The view point is held constant; only the view direction and size are changed. If *x y z* is missing, the cursor is used to select the view center. A negative magnification factor means zoom out. The default factor is one.

**^C**
: Interrupt. Go to the command line.

**exposure** *[ spec ]*
: Adjust exposure. The number *spec* is a multiplier used to compensate the average exposure. A value of 1 renormalizes the image to the computed average, which is usually done immediately after startup. If *spec* begins with a '+' or '-', the compensation is interpreted in f-stops (ie. the power of two). If *spec* begins with an '=', an absolute setting is performed. An '=' by itself permits interactive display and setting of the exposure. If *spec* begins with an '@', the exposure is adjusted to present similar visibility to what would be experienced in the real environment. If *spec* is absent, or an '@' is followed by nothing, then the cursor is used to pick a specific image location for normalization.

**focus** *[distance]*
Set focus distance for depth-of-field sampling. If a distance in world coordinates is absent, then the cursor is used to choose a point in the scene on which to focus. (The focus distance setting does not affect rendering in *rvu,* but can be used in *rpict* with the *−pd* option to simulate depth-of-field on views saved from *rvu.)*

**frame** *[ xmin ymin xmax ymax ]*
Set frame for refinement. If coordinates are absent, the cursor is used to pick frame boundaries. If ‘‘all’’ is specified, the frame is reset to the entire image.

**free** Free cached object structures and associated data. This command may be useful when memory is low and a completely different view is being generated from the one previous.

**last** *[ file ]* Restore the previous view. If a view or picture *file* is specified, the parameters are taken from the last view entry in the file.

**L** *[ vw [ rfile ] ]*
Load parameters for view *vw* from the *rad(1)* input file, *rfile*. Both *vw* and *rfile* must be given the first call, but subsequent calls will use the last *rfile* as a default, and "1" as the default view (ie. the first view appearing in *rfile)*. If *rvu* was started by *rad,* then the *rfile* parameter will initially default to the rad input file used.

**move** *[ mag [ x y z ] ]*
Move camera *mag* times closer to point *x y z* . For a perspective projection (or fisheye view), only the view point is changed; the view direction and size remain constant. The view size must be modified in a parallel projection since it determines magnification. If *x y z* is missing, the cursor is used to select the view center. A negative magnification factor decreases the object size. The default factor is one. Care must be taken to avoid moving behind or inside other objects.

**new** *[ nproc ]*
Restart the image, using the specified number of rendering processes. Usually used after the "set" command.

**origin** *[ xo yo zo [ xd yd zd ] ]*
Change view origin to the indicated world position *xo yo zo* looking in the direction *xd yd zd*. If the direction is missing, the current view direction is used. If the origin is missing, the cursor is used to select the view origin, and the direction will be determined by the (reoriented) surface normal. The view type and size will not be altered, but the up vector may be changed if the new direction coincides.

**pivot** *angle [ elev [ mag [ x y z ] ] ]*
Similar to the "move" command, but pivots the view about a selected point. The *angle* is measured in degrees around the view up vector using the right hand rule, so a positive value pivots the viewer to the right of the selected point. The optional *elev* is the elevation in degrees from the pivot point; positive raises the view point to look downward and negative lowers the view point to look upward.

**quit** Quit the program.

**ˆR** Redraw the image. Use when the display gets corrupted. On some displays, occassionally forcing a redraw can improve appearance, as more color information is available and the driver can make a better color table selection.

**rotate** *angle [ elev [ mag ] ]*
Rotate the camera horizontally by *angle* degrees using the right-hand rule. A positive value rotates the view towards the left, and a negative value looks to the right. If an elevation is specified, the camera looks upward *elev* degrees. (Negative means look downward.)

**set** *[ var [ val ] ]*
Check/change program variable. If *var* is absent, the list of available variables is displayed. If *val* is absent, the current value of the variable is displayed and changed interactively.

Otherwise, the variable *var* assumes the value *val*. Variables include: ambient value (av), ambient value weight (aw), ambient bounces (ab), ambient accuracy (aa), ambient divisions (ad), ambient radius (ar), ambient samples (as), black&white (b), back face visibility (bv), direct jitter (dj), direct sampling (ds), direct threshold (dt), direct visibility (dv), irradiance (i), limit weight (lw), limit recursion (lr), medium extinction (me), medium albedo (ma), medium eccentricity (mg), medium sampling (ms), pixel sample (ps), pixel threshold (pt), specular jitter (sj), specular threshold (st), and uncorrelated sampling (u). Once a variable has been changed, the "new" command can be used to recompute the image with the new parameters. If a program variable is not available here, it may show up under some other command or it may be impossible to change once the program is running.

**trace** *[ xbeg ybeg zbeg xdir ydir zdir ]*
Trace a ray. If the ray origin and direction are absent, the cursor is used to pick a location in the image to trace. The object intersected and its material, location and value are displayed.

**view** *[ file [ comments ] ]*
Check/change view parameters. If *file* is present, the view parameters are appended to a file, followed by *comments* if any. Alternatively, view options may be given directly on the command line instead of an output view file. Otherwise, view parameters are displayed and changed interactively.

**V** *[ vw [ rfile ] ]*
Append the current view as view *vw* in the rad file *rfile*. Compliment to *L* command. Note that the view is simply appended to the file, and previous views with the same name should be removed before using the file with *rad*.

**write** *[ file ]*
Write picture to *file*. If argument is missing, the current file name is used.

**^Z** Stop the program. The screen will be redrawn when the program resumes.

## ENVIRONMENT

RAYPATH the directories to check for auxiliary files.
DISPLAY_GAMMA the value to use for monitor gamma correction.

## AUTHOR

Greg Ward

## SEE ALSO

getinfo(1), lookamb(1), oconv(1), pfilt(1), rad(1), rpict(1), rtrace(1)

## NAME

tabfunc - convert table to functions for rcalc, etc.

## SYNOPSIS

**tabfunc** [ **–i** ] func1 [func2 ..]

## DESCRIPTION

*Tabfunc* reads a table of numbers from the standard input and converts it to an expression suitable for *icalc(1), rcalc(1)* and their cousins. The input must consist of a M x N matrix of real numbers, with exactly one row per line. The number of columns must always be the same in each line, separated by whitespace and/or commas, with no missing values. The first column is always the independent variable, whose value indexes all of the other elements. This value does not need to be evenly spaced, but it must be either monotonically increasing or monotonically decreasing. (I.e. it cannot go up and then down, or down and then up.) Maximum input line width is 4096 characters and the maximum number of data rows is 1024. Input lines not beginning with a numerical value will be silently ignored.

The command-line arguments given to *tabfunc* are the names to be assigned to each column. *Tabfunc* then produces a single function for each column given. If there are some columns which should be skipped, the dummy name "0" may be given instead of a valid identifier. (It is not necessary to specify a dummy name for extra columns at the end of the matrix.)

The *–i* option causes *tabfunc* to produce a description that will interpolate values in between those given for the independent variable on the input.

## EXAMPLE

To convert a small data table and feed it to rcalc for some calculation:

```
rcalc –e ‘tabfunc f1 f2 < table.dat‘ –f com.cal
```

## AUTHOR

Greg Ward

## SEE ALSO

cnt(1), icalc(1), neaten(1), rcalc(1), rlam(1), total(1)

## NAME

tmesh2rad - convert a triangular mesh to a RADIANCE scene description

## SYNOPSIS

**tmesh2rad** [ **−o obj** ][ **−m mat** ][ **−p pat** ] [ **input ..** ]

## DESCRIPTION

*Tmesh2rad* converts one or more triangle-mesh files to a RADIANCE scene description. The *−o* option may be used to assign a default object name. The single letter "T" is used if no name is given on the command line or in the file. The *−m* option may be used to assign a default material name. The non-material "void" is used as a default if none is given on the command line or in the file. The *−p* option may be used to assign a default picture for a surface pattern. If none is given on the command line or in the file, the surface will not have an associated pattern.

## FILE FORMAT

A triangle-mesh is a free-format ASCII file composed of the following eight primitive types. Each primitive is begun with a single, white-space-delimited letter:

*# Comment*
: Whatever follows up until the end of line is passed as a comment to the output. Note that there must be at least one space or tab following the pound-sign.

**o** *name*
: The white-space-delimited string *name* is used as a prefix for all following output triangles.

**m** *material*
: The white-space-delimited string *material* is used as the modifier name for all following output triangles.

**p** *picture*
: The white-space-delimited string *picture* is used as the name of the RADIANCE picture file to be used as a pattern for all following output triangles with properly defined vertices. (See *i* primitive below.)

**v** *id x y z*
: Defines the vertex *id* with 3-dimensional coordinates *x, y* and *z*. The identifier, *id* must be some small, non-negative integer value. If the same integer is used for a later vertex definition, this definition will be lost, though any triangles using the vertex prior to its redefinition will be unaffected.

**n** *nx ny nz*
: Defines a surface normal vector with the 3-dimensional components *nx, ny* and *nz*. This vector will be associated with the most recently defined vertex, and is often placed on the same line as the vertex definition for clarity. The vector need not be normalized.

**i** *u v*
: Defines a picture index for the most recently defined vertex. The *u* value will be used to lookup the horizontal pixel coordinate in the currently defined picture. The *v* value will be used to lookup the vertical pixel coordinate. (See the RADIANCE reference manual for details on picture coordinate values.) As with associated surface normals, picture indices are interpolated using barycentric coordinates based on the triangle vertices. If these coordinates are calculated correctly, this should result in a smooth mapping of a pattern onto the surface mesh.

**t** *id1 id2 id3*
: Create a triangle connecting the three vertices identified by *id1, id2* and *id3*. The right-hand rule is used to determine the default surface normal orientation, and this should not be too far from the associated vertex normals (if any). All three vertices must have an associated normal if the triangle is to be smoothed. If a picture file is defined and all three vertices have pattern indices associated with them, then this picture will be used as a pattern to modify the triangle's color.

We realize there are many similar T-mesh file formats in existence, and that it would have been just as easy to support one of these formats directly. The disadvantage to supporting an existing format is that conversion from other formats might prove difficult. It was our hope to provide a "greatest common multiple" format that would support all similar T-mesh formats, rather than supporting WaveFront's .obj format (for example) and being unable to associate a pattern with an object. Converting from other formats should be relatively straightforward. In many cases, an *awk(1), rcalc(1)* or even a *sed(1)* script should be sufficient.

## EXAMPLE

Here is an example T-mesh file:

```
# Our object name:
o test_object
# Our material:
m puce
# Our vertices:
v 1     10     15     5
v 2     10     –15    5
v 3     0      –15    0
v 4     –10    15     –5
# Two triangles joined together:
t 1 2 3
t 2 3 4
```

Which generates the following output:

```
## T-mesh read from: <stdin>

# Our material:

# Our vertices:

# Two triangles joined together:

puce polygon test_object.1
0
0
9
        10        15        5
        10        –15        5
        0         –15        0

puce polygon test_object.2
0
0
9
        10        –15        5
        0         –15        0
        –10        15        –5
```

Here is another example:

```
# A partial cylinder:
m BluePlastic
v 1 –14.673 –3.119    50 n –0.95677 –0.203374 1.17936e–10
v 2 –12.136 –8.817   –50 n –0.791363 –0.574922 4.84915e–10
v 3 –12.136 –8.817    50 n –0.791363 –0.574922 4.84915e–10
t 1 2 3
m OrangePlastic
v 1 –7.501 –12.991    50 n –0.549094 –0.812427 –1.45812e–09
v 2 –12.136 –8.817    50 n –0.791363 –0.574922 4.84915e–10
v 3 –12.136 –8.817   –50 n –0.791363 –0.574922 4.84915e–10
```

```
t 1 2 3
m BluePlastic
v 1 –1.568 –14.918    50 n –0.171094 –0.965568 –5.69788e–09
v 2 –7.501 –12.991    50 n –0.549094 –0.812427 –1.45812e–09
v 3 –7.501 –12.991   –50 n –0.429001 –0.881759 –3.6502e–09
t 1 2 3
```

Note that the same three vertices were used repeatedly, and intermingled with the triangle definitions.

## AUTHOR

Greg Ward

## BUGS

Triangle smoothing doesn't work very well for glass or trans material types in Radiance, since textures cause distorted transmission through these materials. It is best to use the dielectric material type if smooth transmission is desired.

## SEE ALSO

arch2rad(1), awk(1), ies2rad(1), thf2rad(1), oconv(1), rcalc(1), sed(1), xform(1)

## NAME

total - sum up columns

## SYNOPSIS

**total** [ **−m** ][ **−sE** | **−p** | **−u** | **−l** ][ **−i{fld}[N]** ][ **−o{fld}** ][ **−tC** ][ **−N** [ **−r** ]] [ file .. ]

## DESCRIPTION

*Total* sums up columns of real numbers from one or more files and prints out the result on its standard output.

By default, *total* computes the straigt sum of each input column, but multiplication can be specified instead with the *−p* option. Likewise, the *−u* option means find the upper limit (maximum), and *−l* means find the lower limit (minimum).

Sums of powers can be computed by giving an exponent with the *−s* option. (Note that there is no space between the *−s* and the exponent.) This exponent can be any real number, positive or negative. The absolute value of the input is always taken before the power is computed in order to avoid complex results. Thus, *−s1* will produce a sum of absolute values. The default power (zero) is interpreted as a straight sum without taking absolute values.

The *−m* option can be used to compute the mean rather than the total. For sums, the arithmetic mean is computed. For products, the geometric mean is computed. (A logarithmic sum of absolute values is used to avoid overflow, and zero values are silently ignored.)

If the input data is binary, the *−id* or *−if* option may be given for 64-bit double or 32-bit float values, respectively. Either option may be followed immediately by an optional count, which defaults to 1, indicating the number of double or float binary values to read per record on the input file. (There can be no space between the option and this count.) Similarly, the *−od* and *−of* options specify binary double or float output, respectively. These options do not need a count, as this will be determined by the number of input channels.

A count can be given as the number of lines to read before computing a result. Normally, *total* reads each file to its end before producing its result, but this behavior may be overridden by inserting blank lines in the input. For each blank input line, total produces a result as if the end-of-file had been reached. If two blank lines immediately follow each other, total closes the file and proceeds to the next one (after reporting the result). The *−N* option (where N is a decimal integer) tells *total* to produce a result and reset the calculation after every N input lines. In addition, the *−r* option can be specified to override reinitialization and thus give a running total every N lines (or every blank line). If the end of file is reached, the current total is printed and the calculation is reset before the next file (with or without the *−r* option).

The *−tC* option can be used to specify the input and output tab character. The default tab character is TAB.

If no files are given, the standard input is read.

## EXAMPLE

To compute the RMS value of colon-separated columns in a file:

```
total −t: −m −s2 input
```

To produce a running product of values from a file:

```
total −p −1 −r input
```

## BUGS

If the input files have varying numbers of columns, mean values will certainly be off. *Total* will ignore missing column entries if the tab separator is a non-white character, but cannot tell where a missing column should have been if the tab character is white.

## AUTHOR

Greg Ward

## SEE ALSO

cnt(1), neaten(1), rcalc(1), rcollate(1), rlam(1), tabfunc(1)

## NAME

trad - graphical user interface to Radiance rad(1) program

## SYNOPSIS

**trad** [ **rfile** ]

## DESCRIPTION

*Trad* is a graphical user interface to *rad(1),* which controls the operation of the basic Radiance scene compiling, rendering and picture filtering programs. Trad also includes links to a few utilities for displaying and converting results, but most of what it does can be done by editing a small text file, called the "rad input file". Scene creation still requires the use of a text or graphical editor, or translation from some external CAD format.

*Trad* is based on the Tcl/Tk *wish(1)* "windowing shell" written by John Ousterhout. (See below for instructions on installing this package if you do not have it already.)

The *trad* interface divides the rendering problem into seven screens: File, Scene, Zone, Views, Options, Action and Results. The File screen is used to load and save rad input files (a.k.a. project files). The Scene screen is used to name the Radiance input files associated with a particular project. The Zone screen is used to assign *rad* variables specific to the section of the model being rendered. The Views screen is used to define specific views to be rendered and set the picture file names and dimensions. The Options screen is used to adjust rendering quality and other parameters. The Action screen is used to initiate interactive and batch renderings. The Results screen is used to display, convert and print the rendered Radiance pictures.

If *trad* is called with no rad input file name on the command line, it will start with the File screen and you must enter a valid project file before you will be allowed to continue. If *rfile* is given, then *trad* attempts to open this file. If no such file exists, *trad* assumes you are creating a new file by this name and goes to the Scene screen so you may identify the appropriate Radiance input files. If the file exists but not all renderings have been finished, *trad* goes first to the Action screen, assuming you will want to do something. If the file exists and all renderings have completed and are up-to-date, *trad* goes to the Results screen so that you may examine the final pictures.

*Trad* includes an extensive help facility, which may be accessed either by pressing the "HELP" button and searching through the category and topic menus, or by holding the Control key and pressing the left mouse button on the mysterious widget (i.e. the button, entry window, or list box you are curious about).

## INSTALLING TCL/TK

The Tcl/Tk package is available by anonymous ftp from ftp.smli.com (204.153.12.45) in the /pub/tcl directory. (Tcl stands for "Tool Command Language," and is pronounced "tickle".) *Trad* is based on Tk release 4.0, which in turn is based on Tcl release 7.4. Although *trad* uses only the *wish* program from this package, *wish* itself depends on additional Tcl and Tk libraries, and the two toolkits must be compiled one after the other.

To compile the Tcl/Tk package, download the files "tcl7.4.tar.Z" and "tk4.0.tar.Z" and uncompress and untar them. Then, follow the instructions in the README file in the tcl7.4/ directory to configure the Makefile for your system and install the software. (Usually, it is best to do this as root or have it done by your system administrator if you do not have root privileges.) Then, do the same in the tk4.0/ directory.

*Trad* should run without modification once this is done correctly.

## AUTHOR

Greg Ward
*wish* and Tcl/Tk language by John Ousterhout

## SEE ALSO

oconv(1), pfilt(1), rad(1), rpict(1), rvu(1), wish(1), ximage(1)

## NAME

ttyimage - RADIANCE driver for dumb ASCII terminal

## SYNOPSIS

**ttyimage** [ **−c resolu** ][ **−r** ] [ **pixfile** ]

## DESCRIPTION

*Ttyimage* takes the RADIANCE picture file *pixfile* and displays it on a dumb terminal. If no *pixfile* is given, the standard input is read.

## AUTHOR

Greg Ward

## SEE ALSO

pfilt(1), rpict(1), ximage(1)

## NAME

vwrays - compute rays for a given picture or view

## SYNOPSIS

**vwrays [ -i -u -f{alfld} -c rept | -d ] { view opts .. | picture [zbuf] }**

## DESCRIPTION

*Vwrays* takes a picture or view specification and computes the ray origin and direction corresponding to each pixel in the image. This information may then be passed to *rtrace(1)* to perform other calculations. If a given pixel has no corresponding ray (because it is outside the legal view boundaries), then six zero values are sent instead.

The *−i* option may be used to specify desired pixel positions on the standard input rather than generating all the pixels for a given view. If the *−u* option is also given, output will be unbuffered.

The *−f* option may be used to set the record format to something other than the default ASCII. Using raw float or double records for example can reduce the time requirements of transferring and interpreting information in *rtrace*.

The *−c* option repeats each pixel the given number of times (default is 1). This is most useful when sending rays to *rcontrib(1)* with the same *−c* setting, providing a much faster way to average pixels over image sets. The *−pj* option should be used to jitter sample postions in most cases.

View options may be any combination of standard view parameters described in the *rpict(1)* manual page, including input from a view file with the *−vf* option. Additionally, the target X and Y dimensions may be specified with *−x* and *−y* options, and the pixel aspect ratio may be given with *−pa*. The default dimensions are 512x512, with a pixel aspect ratio of 1.0. Just as in *rpict,* the X or the Y dimension will be reduced if necessary to best match the specified pixel aspect ratio, unless this ratio is set to zero. The *−pj* option may be used to jitter samples. The default value of 0 turns off ray jittering.

If the *−d* option is given, then *vwrays* just prints the computed image dimensions, which are based on the view aspect and the pixel aspect ratio just described. The *−ld* switch will also be printed, with *−ld+* if the view file has an aft clipping plane, and *−ld-* otherwise. This is useful for passing options to the *rtrace* command line. (See below.)

If the view contains an aft clipping plane *(-va* option), then the magnitudes of the ray directions will equal the maximum distance for each pixel, which will be interpreted correctly by *rtrace* with the *−ld+* option. Note that this option should not be given unless there is an aft clipping plane, since the ray direction vectors will be normalized otherwise, which would produce a uniform clipping distance of 1.

If a picture is given on the command line rather than a set of view options, then the view and image dimensions are taken from the picture file, and the reported ray origins and directions will match the center of each pixel in the picture (plus optional jitter).

If a depth buffer file is given as well, then *vwrays* computes the intersection point of each pixel ray (equal to the ray origin plus the depth times the ray direction), and reports this instead of the ray origin. The reported ray direction will also be reversed. The interpretation of this data is an image of origins and directions for light rays leaving the scene surfaces to strike each pixel.

## EXAMPLES

To compute the ray intersection points and returned directions corresponding to a picture and its depth buffer:

```
vwrays scene_v2.hdr scene_v2.zbf > scene_v2.pts
```

To determine what the dimensions of a given view would be:

```
vwrays –d –vf myview.vf –x 2048 –y 2048
```

To generate a RADIANCE picture using *rtrace* instead of *rpict:*

```
vwrays –ff –vf view1.vf –x 1024 –y 1024 | rtrace ‘vwrays –d –vf view1.vf –x 1024 –y 1024‘ –ffc
scene.oct > view1.hdr
```

## AUTHOR

Greg Ward Larson

## ACKNOWLEDGMENT

This work was supported by Silicon Graphics, Inc.

## BUGS

Although *vwrays* can reproduce any pixel ordering (i.e., any image orientation) when given a rendered picture, it will only produce standard scanline-ordered rays when given a set of view parameters.

## SEE ALSO

rcalc(1), rpict(1), rcontrib(1), rtrace(1)

## NAME

vwright - normalize a RADIANCE view, shift it to the right

## SYNOPSIS

**vwright [ view options ] distance**
**vwright [ view options ] name**

## DESCRIPTION

In the first form, *vwright* shifts a RADIANCE view the specified distance to the right, putting out a complete set of view parameters in a single line on the standard output. This utility is most often used to compute a right-eyed view from a left-eye view for stereo imaging.

If no options are specified on the command line, *vwright* reads a view from its standard input.

The *distance* given is in world coordinate units. A negative value indicates a shift to the left rather than the right.

The second form substitutes a name prefix in place of the shift distance, and produces constant assignments on the standard output suitable for passing directly to *rcalc(1)*. For a given prefix *N,* the constant names are as follows:

| | |
|---|---|
| Nt: | view type ('v'==1,'l'==2,'a'==3,'h'==4,'c'==5,'s'==6) |
| Npx: | view point x value |
| Npy: | view point y value |
| Npz: | view point z value |
| Ndx: | view direction x value (normalized) |
| Ndy: | view direction y value (normalized) |
| Ndz: | view direction z value (normalized) |
| Nd: | view focal distance |
| Nux: | view up vector x value (normalized) |
| Nuy: | view up vector y value (normalized) |
| Nuz: | view up vector z value (normalized) |
| Nh: | view horizontal size |
| Nv: | view vertical size |
| Ns: | view shift |
| Nl: | view lift |
| No: | view fore clipping distance |
| Na: | view aft clipping distance |
| Nhx: | derived horizontal image vector x value (normalized) |
| Nhy: | derived horizontal image vector y value (normalized) |
| Nhz: | derived horizontal image vector z value (normalized) |
| Nhn: | derived horizontal image vector multiplier |
| Nvx: | derived vertical image vector x value (normalized) |
| Nvy: | derived vertical image vector y value (normalized) |
| Nvz: | derived vertical image vector z value (normalized) |
| Nvn: | derived vertical image vector multiplier |

## EXAMPLES

To start *rpict(1)* on a view .06 meters left of the view in the file "right.vf":

```
rpict `vwright -.06 < right.vf` scene.oct > right.hdr &
```

To move the *rad(1)* view named "left" 2.5 inches to the right and render from there:

```
rad –v "right `rad –n –s –V –v left examp.rif | vwright 2.5`" examp.rif &
```

To pass a view to *rcalc* for conversion to some other view:

```
rcalc –n –e `vwright orig < orig.vf` –f viewmod.cal –o view.fmt > new.vf
```

## AUTHOR

Greg Ward

## SEE ALSO

pdfblur(1), rad(1), rcalc(1), rpict(1), rvu(1)

## NAME

x11meta - output metafile graphics to X11

## SYNOPSIS

**x11meta** [ **−c** | **−r** ] file ..

## DESCRIPTION

*X11meta* reads each metafile *file* in sequence and sends it to the default X window system. If the option *c* is specified, the input files are only conditioned for output, ie. expanded and sorted (see pexpand and psort). If the option *r* is instead specified, the input is assumed already to be conditioned. If no input files are specified, the standard input is read.

**−c** Condition the input only.

**−r** Input is already conditioned, output only.

## EXAMPLE

To plot the chart example.bar:

```
bgraph example.plt | x11meta
```

## FILES

see pexpand(1) and psort(1)

## AUTHOR

Greg Ward

## SEE ALSO

bgraph(1), cv(1), igraph(1), meta2bmp(1), meta2tga(1), metafile(5), pexpand(1), psmeta(1), psort(1)

## NAME

xform - transform a RADIANCE scene description

## SYNOPSIS

**xform** [ **−c** ][ **−I** ][ **−n name** ][ **−m newmod** ][ **−f argfile** ][ **xf0** ][ **−a n1 xf1 ..** ][ **−i 1 xff** ] **file ..**

## DESCRIPTION

*Xform* transforms each scene description *file* according to the options given. If no *file* is specified, the standard input is read. The *−c* option causes commands in the input not to be expanded. The default is to execute all in line commands. (See note below about file names.) The *−n* option causes all identifiers to be prefixed with *name*. The *−m* option causes all surfaces to be given the modifier *newmod*. The *−I* option causes all surfaces to be inverted, reversing their surface normal orientations. These options are followed by the transformation options, which are described below.

The *−f* option causes the *xform* command line to be constructed from the given file, by inserting each line of the file at the current point in the command argument list. Each line in the file will result in a logically separate invocation of *xform,* and may may contain any valid *xform* arguments, including nested *−f* options. This is a convenient way to specify multiple copies of an object that do not fit a regular array pattern, without having to actually execute *xform* many times. Separate scene files may be specified this way as well, but remember that the constructed command line must fit the format of initial options *(-n, -m, -c, -I)* followed by the transform then the scene files. No initial options may appear after the first transform option, and no transform options will be understood after the first named file. In the special case where the argument to the *−f* option is a hyphen ('-'), *xform* will take its arguments from the standard input. Note that *xform* cannot simultaneously take its scene information from the standard input if the option is used in this way. Competely empty lines and lines beginning with a pound sign ('#') will be silently ignored. Beginning "!xform" or "xform" command names will also be ignored.

If one or more scene files are given on the command line, *xform* will search the RADIANCE library directories for each file. (No search takes place if a file name begins with a '.', '/' or '~' character.) Unless the *−c* option is present, *xform* will also change to that file's directory before loading it. Thus, any commands executed within that file will happen in that file's directory, which simplifies object hierarchy construction.

The transformation consists of a sequence of operations which are executed in the order they appear.

## OPTIONS

**-t** *x y z* Translate the scene along the vector *x y z* .

**-rx** *degrees*
Rotate the scene *degrees* about the x axis. A positive rotation corresponds to counter-clockwise when looking down the axis.

**-ry** *degrees*
Rotate the scene *degrees* about the y axis.

**-rz** *degrees*
Rotate the scene *degrees* about the z axis.

**-s** *factor* Scale the scene by *factor.*

**−mx** Mirror the scene about the yz plane.

**−my** Mirror the scene about the xz plane.

**−mz** Mirror the scene about the xy plane.

**-i** *count* Iterate (repeat) the following transformation (up to the next *−i* option) *count* times. This option is primarily to support the *−a* option, which is described below.

### Arrays

An array is a repeated transformation that results in a repeated object. It is specified using the *−a* option, which takes the number to repeat as its argument. The objects will step by the transformation given between this *−a* option and the next *−a* or *−i* option. The first object will have zero applications of the transform. A two-dimensional array is given by two different transformations each preceded by an array

count.

## EXAMPLE

To rotate "book" 30 degrees about the x axis then move 20 in y, prepending the name book1:

```
xform –n book1 –rx 30 –t 0 20 0 book > book1
```

To expand all commands and see what information is actually used by RADIANCE:

```
xform scene | more
```

To create a two-dimensional array of 20 lights, after an initial rotation and followed by a global translation (no command expansion):

```
xform –c –rz 90 –a 5 –t 2 0 0 –a 4 –t 0 1.5 0 –i 1 –t 0 0 10 light
```

## ENVIRONMENT

| | |
|---|---|
| RAYPATH | path to search for scene files |

## AUTHOR

Greg Ward

## BUGS

Only regular (distortion-free) transformations are allowed.

## SEE ALSO

genbox(1), gensurf(1), oconv(1), replmarks(1), rpict(1), rvu(1)

## NAME

xglaresrc - dislpay glare sources under X11

## SYNOPSIS

**xglaresrc** [ **−n windowname** ][ **−c r g b** ] **pictfile** [ **glarefile** ]

## DESCRIPTION

*Xglaresrc* displays the sources located by *findglare(1)* in the picture *pictfile* by displaying their average luminance and drawing circles around them in the image. If *pictfile* is already being displayed by *ximage(1), xglaresrc* will raise (or deiconify) this window. If *pictfile* is not being displayed, *xglaresrc* will start *ximage(1)* in the background automatically. (To quit from *ximage(1),* you can type 'q' in its display window.) Usually, *pictfile* will be the image that was used by *findglare(1)* to locate glare sources.

If the desired image is being displayed under a different name, perhaps even by a different display program than *ximage(1),* the *−n* option can be used to give an alternative *windowname* for locating the correct display window, and the *−c* option can be used to specify a different line and text color (default 1 0 0). If the inpute *glarefile* is not given, *xglaresrc* reads from its standard input.

## AUTHOR

Greg Ward

## ACKNOWLEDGEMENT

Work on this program was initiated and sponsored by the LESO group at EPFL in Switzerland.

## SEE ALSO

findglare(1), glare(1), glarendx(1), ximage(1)

## NAME

ximage - RADIANCE picture display for X window system

## SYNOPSIS

**ximage** [ **=geometry** ][ **−di display** ][ **−c ncolors** ][ **−d** ][ **−b** ][ **−m** ][ **−g gamma** ][ **−f** ][ **−e spec** ][ **-o***spec* ][ **−t intvl** ][ **−s** ] **picture ..**

## DESCRIPTION

*Ximage* takes one or more RADIANCE picture files and displays them on an X server. The *−c* option specifies the number of colors to use (default fills color table). The *−d* option turns off color dithering. The *−b* option displays the image in black and white (greyscale). The *−m* option forces monochrome output. The *−g* option specifies the exponent used in gamma correction; the default value is 2.2. The *−f* option stores a Pixmap on the server side for faster refresh. This may not work with large images on some servers. The *−o* option specifies a sequence of information to print to the standard output for the 't' command (see below). The *−t* option specifies a minimum interval (in milliseconds) between successive ray outputs in mouse tracking mode (right button pressed).

The *−e* option specifies an exposure compensation in f-stops (powers of two). Only integer stops are allowed, for efficiency. If the special word, *auto* is given instead of a number of stops, then *ximage* performs an automatic exposure adjustment similar to *pcond(1),* compressing the dynamic range of the image to fit within the dynamic range of the display. If the special word, *human* is given instead, then *ximage* performs an exposure adjustment similar to *pcond* with the *−s* and *−c* options, which compensate for human contrast and color sensitivity at the corresponding scene luminance levels. This option yields and appearance of the scene on the display that closely matches what would be experienced in the real world.

The *−s* option tells *ximage* to display multiple pictures sequentially, rather than all at once. If no *picture* is given, input is read from stdin provided either the *−b* or *−m* option is in effect, or the X server is capable of 24-bit color. However, many of the commands given below will not work.

## COMMANDS

Once a picture is displayed, the user may perform a number of operations. Some of the operations make use of an area of interest, defined by pressing the left mouse button and dragging the cursor over a section of the image. Pressing the button and immediately releasing it defines a single point as the area of interest. A command is a single character.

**q** Quit picture. (Also Q or ^D.)

**<space>** Redraw the area of interest.

**^R** Redraw the entire image.

**<return>** Display the radiance averaged over the area of interest. The maximum of the three (RGB) channels is reported.

**l** Display the photometrically-weighted luminance value in the area of interest. This assumes that the image was correctly computed in terms of luminance.

**c** Display the color in the area of interest, as adjusted by the current exposure setting.

**p** Display the x and y location of the cursor.

**i** Identify identical pixels by assigning a random color at the cursor position. This is useful for displaying contours, especially when combined with the −b option.

**t** Print information about the pixel under the cursor according to the string following the *−o* command line option. The valid characters for this option correspond roughly to the other *ximage* commands:

- o ray origin
- d ray direction
- v radiance value

l luminance value

p pixel position

The default output is "−ood", which prints the ray origin and direction. This can be used as input to rtrace(1) to get additional information about the image (ie. pipe the output of ximage into rtrace). Pressing the middle mouse button is equivalent to typing the 't' key. Pressing and holding the right mouse button is equivalent to continuously pressing the 't' key.

**=** Adjust the exposure to the area of interest. A crude adjustment is made immediately, and the number of stops is printed while the colors are resampled. After a few seconds to a minute, the final image is redisplayed. If the area of interest is already within 1/2 stop of the ideal, no adjustment is made.

**@** Same as '=' command, only the exposure is adjusted to provide roughly the same visibility for the selected region on screen as a viewer would experience in the actual space. Like the 'l' command, this adjustment assumes that the image has been correctly computed in terms of luminance. (See also the 'h' command, below.)

**a** Perform automatic exposure compensation, as if *ximage* were started with the *−e auto* option. If a rectangular area has been selected, the pixels in this region will be emphasized in the histogram, offering this area exposure preference. (Each pixel within the rectangle will be weighted as 21 outside pixels.)

**h** Perform human expsoure compensation, as if *ximage* were started with the *−e human* option. See the 'a' command above regarding pixel weighting.

**0** Reset the origin to the upper left corner of the image. This command is used to restore the original image position after using the shift or control key with the mouse to relocate the image within the frame (see below).

**f** Switch on the fast redraw option *(−f),* loading the image pixmap over to the server side. This command is useful when network delays are causing slow image refresh, and the user didn't notice it until after *ximage* was started.

**F** Switch off the fast redraw option. This frees up some memory on the server, as well as the color table for other windows.

In addition to the commands listed above, the control or shift key may be held while the cursor is dragged to reposition the image within the window.

## X RESOURCES

radiance.gamma the default gamma correction value

## ENVIRONMENT

DISPLAY_GAMMA the default gamma correction value

## AUTHORS

Greg Ward
Anat Grynberg (Paris)
Philip Thompson (MIT)

## SEE ALSO

aedimage(1), normtiff(1), pcond(1), pfilt(1), rpict(1), rtrace(1), rvu(1), xglaresrc(1), xshowtrace(1)

## NAME

xshowtrace - interactively show rays traced on RADIANCE image under X11

## SYNOPSIS

**xshowtrace** [ **−s** ][ **−T** ][ **rtrace options** ] **octree picture**

## DESCRIPTION

*Xshowtrace* takes a RADIANCE octree and a picture file and displays it on an X11 window server using *ximage(1)*. The picture should have been created from a previous *rpict(1)* or *rvu(1)* calculation using the given octree. Once the image is displayed, the user can use the 't' command of *ximage* to select points on the image to display the ray tree. *Rtrace* then produces a ray tree, which *xshowtrace* will display (in red on a color screen). The *−s* option slows the display of each ray traced to make it easier to follow the process. The *−T* option traces rays to light sources, which are normally hidden.

## AUTHOR

Greg Ward

## BUGS

If the pointer is moved between the time 't' is pressed and *xshowtrace* starts drawing rays, the rays will be displaced.

## SEE ALSO

oconv(1), rpict(1), rtrace(1), rvu(1), ximage(1)