RADIANCE
System Overview

- Collection of over 100 individual programs
- Programs communicate via standard file types
- Basic file types:
  - Scene description files
  - Function files
  - Data files
  - Font files
  - Octree files (binary)
  - Ambient files (binary)
  - Picture files (binary)
  - Plot files
- Basic operations:
  - Generate a scene object, sky description, etc.
  - Compile scene description into an octree
  - Compute radiances, irradiances, glare, etc.
  - Manipulate and plot computed values
  - Render scene from a particular viewpoint
  - Filter picture
  - Display picture
- Import/Export options:
  - CAD import translators
  - Image import/export translators

Scene Description Files

- ASCII files containing:
  - Primitives
  - Aliases
  - Inline Commands
  - Comments
  - White Space (i.e. free format)

- A Primitive is a basic *Radiance* Object
  - surface
  - material
  - pattern
  - texture

Example

```
green sphere ball
0
0
4  10  15  -5  1.5
```

- An Alias associates a new name with a previously defined primitive

Example

```
void alias wall_color
           blue
```

Basic File Types — **Scene Description Files**

- Inline Commands simply take the output of a command as more scene input

Examples

```
!gensky 6 21 12 -a 42 -o 118

!genbox white room 15 35 5 | xform -t -5 10 0

!xform -e -rz 60 -t 15 8 2 object.rad
```

- A Comment is any line beginning with a pound sign ('#') and is generally ignored

Examples

```
# This is a comment which contains some
# information that is useful to the user
# but has no meaning for the software.
# Comments cannot appear in the middle
# of a primitive or alias or inline command.
# Sometimes a program inserts a comment for
# a special purpose, though it is usually
# just to let the user know that it
# generated the output being examined.
```

- White Space (i.e. spaces, tabs, newlines, form feeds) are used only to separate words

Example Scene:

```
# this is the material for my light source:

void light bright
0
0
3  100  100  100

# this is the material for my test ball:

void plastic red_plastic
0
0
5  .7  .05  .05  .05  .05

void alias ball_material red_plastic

# here is the light source:

bright sphere fixture
0
0
4  2  1  1.5  .125

# here is the ball:

ball_material sphere ball
0
0
4  .7  1.125  .625  .125

# the wall material:

void plastic gray_paint
0
0
5  .5  .5  .5  0  0

# a box shaped room:

!genbox gray_paint room 3 2 1.75 -i
```

Function Files

- ASCII files defining variables and functions for procedural patterns and textures, data coordinate mapping, procedural surface definitions and data manipulation

- The expression language used by *Radiance* is a Turing-equivalent functional language

  - Turing-equivalent is a fancy way of saying, "anything that can be expressed in a standard programming language can be expressed in this language"
  - The term functional language means that the code consists of definitions of things in terms of other things rather than operations to be carried out in a particular sequence
  - Execution is the result of some outside call for a particular evaluation, which precipitates a chain of evaluations yielding the desired result

- The basic data type understood and supported by the language is double precision real

- Externally defined variables and functions give the expressions meaning and power

Example Pattern Function:

```
{
     winxmit.cal - window transmittance function.

     Rdot is predefined as the cosine of the angle
     between the output direction and the surface
     normal (of the window).
}

winxmit = 1.018 * Rdot * (1 + (1 - Rdot*Rdot)^1.5);
```

Corresponding Scene Description:

```
# skyfunc definition:
!gensky 7 12 +15 -a 42

# window distribution function:
skyfunc brightfunc window_dist
2 winxmit winxmit.cal
0
0

# light for window, using 88% normal transmittance:
window_dist light window_illum
0
0
3 .88 .88 .88

# the actual window:
window_illum polygon window
0
0
12
     5   0   3
     10  0   3
     10  0   7
     5   0   7
```

Example Texture Function:

```
{
     woodtex.cal - wood grain texture

     A1 = roughness (0 < roughness < 1).
}

xgrain_dx = 0;
xgrain_dy = A1 * Rdot * sin(wztexang);
xgrain_dz = A1 * Rdot * cos(wztexang);
wxtexang = PI * fnoise3(Px/10, Py, Pz);
```

Corresponding Scene Description:

```
# A woodgrain texture:
void texfunc xwoodtex
4 xgrain_dx xgrain_dy xgrain_dz woodtex.cal -s 0.1
0
1 0.3

xwoodtex plastic xwood
0
0
5 0.333 0.173 0.072 0.019 0.04

!genbox xwood box 10 5 8 -r .25
```

Data Files

- A data file contains 1-dimensional (scalar) values on an N-dimensional grid

- Most data files are used for light fixture photometry, but they can be used to specify any pattern or texture

- Data files contain only ASCII encodings of integer and real numbers separated by white space (i.e. free format)

- The first number is an integer indicating the number of dimensions, followed by that number of dimension specifications, followed by the actual data

- A dimension specification is either a starting and ending value followed by the number of points, or two zeroes followed by the grid (ordinate) values

Example Photometry Data File (taskD.dat):

Number of dimensions

Theta values, uneven steps from 0 to 90

```
2
0 0 11
        0 5 15 25 35 45 55 65 75 85 90
0 180 5
```

Phi values, 0-180 by 45

```
229     229     229     229     229
244     240     228     219     213
280     262     222     193     181
329     284     208     165     153
386     318     190     138     126
345     316     169     109     100
211     209     140     76      70
113     99      89      45      40
65      48      29      20      21
25      18      4       6       7
1       0       0       0       0
```

Data values, running through phi at each theta

Corresponding Scene Description:

```
void brightdata light_dist
7 flatcorr taskD.dat source.cal src_theta src_phi2 -rz -90
0
0

light_dist light light_output
0
0
3 .0418 .0418 .0418

light_output polygon lens
0
0
12
        -18     11.75   -1.875
        18      11.75   -1.875
        18      6       -1.875
        -18     6       -1.875
```

The function flatcorr and the variables src_theta and src_phi2 are defined in the function file "source.cal" located in the standard library directory

Basic File Types

- Font Files
  - ASCII integer encoding, white space delimited
  - Defines polygonal glyph in [0,255] box for each character
  - Used for text patterns and mixtures and by **psign**
  - Currently only one text font provided, helvet.fnt
- Octree Files
  - Binary encoding (but portable between systems)
  - "Frozen" form includes scene data in compact format
  - Used to accelerate ray tracing and for instancing
  - Data structure dependent only on geometry
- Ambient Files
  - Binary but portable
  - Contains view-independent illumination information
  - Used for sharing data between views and processes
  - Can be converted to/from ASCII form by **lookamb**
- Picture Files
  - Binary but portable
  - Contains dimensions, orientation and RGBE 32-bit pixels
  - Generated and used by renderers and filters
  - Run-length encoding reduces file size
- Plot Files
  - Portable ASCII and binary files
  - Contains 2-dimensional point and curve data and functions
  - Used mostly by older *metafile* graphics programs
  - Converters to PostScript and Targa for convenient output

Generators

- **genbox**
  - Generates a rectangular box
  - Options for beveled or rounded edges and corners
- **gensurf**
  - Generates an arbitrary 3-dimensional surface
  - Works from functions or data
  - Optional smoothing (surface normal interpolation)
- **gensky**
  - Standard CIE clear, overcast or intermediate sky
  - Options for latitude, longitude, time zone, etc.
  - Absolute accuracy requires measured sky data
- **xform**
  - Not really a generator -- used to move objects around
  - Input is one or more *Radiance* scene files
  - Supports arrays and, with inline commands, hierarchy
- Others
  - **genblinds** Venetian blinds
  - **gencat** Catenary (e.g. hanging chain)
  - **genprism** Arbitrary prism (i.e. extruded polygon)
  - **genrev** Surface of revolution (using cones)
  - **genworm** Curve with varying thickness (cone-spheres)
  - **replmarks** Replaces small polygons with objects

Scene Compilers

- **oconv**
  - compiles scene description files into an octree
  - octree is required by *Radiance* for rendering
  - octrees also used for instancing, creating object libraries
  - **oconv** also allows incremental compilation
  - options for maximum objects in set, maximum resolution, and scene boundaries

- **getbbox**
  - computes bounding box for scene
  - **oconv** computes bounding cube, so usually unnecessary
  - much faster than **oconv** if only bounding box is needed

This is covered later in "Computing Other Values"

Compute Values

- **rtrace**
  - basic computation engine
  - computes individual radiances and irradiances
  - input and output in alternative formats
  - many other values available as general ray query
  - often run as subprocess by other programs

- **mkillum**
  - computes output distributions of "secondary" sources
  - input is *Radiance* scene description
  - output is modified scene description and data files
  - runs **rtrace** as a subprocess to do the real work

- **findglare**
  - locates and quantifies potential glare sources in a scene
  - input is *Radiance* octree and/or picture file
  - output is list of glare sources and vertical illuminances
  - output is used by glare evaluation program, **glarendx**
  - runs **rtrace** to compute luminances not found in picture
  - usually accessed through interactive front end, **glare**

Rendering

- **rview**
  - interactive rendering program
  - starts at low resolution, works to improve
  - process may be interrupted with any command
  - changes to scene require restarting program

- **rpict**
  - batch rendering program
  - most efficient in time and memory
  - highest quality output
  - can create multiple pictures for animation

- **rpiece**
  - batch parallel and distributed rendering program
  - breaks picture into small pieces
  - calls **rpict** to render each piece
  - cooperates across network file system
  - most efficient on large, expensive pictures
  - requires working NFS lock manager (rare, it seems)

Filtering Pictures

- **pfilt**
  - most basic and essential filter
  - performs anti-aliasing through size reduction
  - adjusts exposure
  - removes Monte Carlo sampling artifacts (speckle)
  - optional color correction and balancing
  - optional star filter effects
  - works on incomplete pictures

- **pcomb**
  - general programmable filter
  - takes any number of input pictures
  - performs any operation that can be defined locally
  - uses same expression language as function files

- **pinterp**
  - takes one or more pictures and creates a new view
  - usually used for interpolation of animated frames

- Others
  - **normpat** "normalize" a picture for use as a pattern
  - **pcompos** general picture cut and paste
  - **pflip** flip a picture left to right, top to bottom
  - **protate** rotate a picture 90 degrees clockwise
  - **psign** create a picture with some text

Getting Information

- **getinfo**
  - reads the standard ASCII information header in a *Radiance* binary file
  - can read bounding cube from an octree
  - can read image dimensions and orientation from a picture

- **lampcolor**
  - computes RGB radiance of light source
  - takes simple geometric dimensions and lamp type
  - useful in creating simple diffuse light sources

- **pextrem**
  - finds minimum and maximum pixels in a picture

- **raddepend**
  - finds scene file dependencies
  - calls **getbbox** then checks file access times

- **lookamb**
  - converts *Radiance* ambient files to/from ASCII form

Import/Export:

Input Translators

- CAD Internal Export Functions
  - **Vision3D** is a shareware CAD program with *Radiance* export
  - **DesignStudio** is a commercial CAD package with export
  - **torad** is an AutoLISP program that loads directly into **AutoCAD**

- CAD Data Files
  - **arch2rad** imports **Architrion** text format files
  - **archicad2rad** imports **ArchiCAD** RIB files
  - **dxfcvt** imports **AutoCAD** version 10 DXF files
  - **thf2rad** imports **GDS** things files

- Geometry Data Files
  - **tmesh2rad** converts triangle mesh data to *Radiance*

- Luminaire Data Files
  - **ies2rad** converts IES standard luminaire files to *Radiance*

Import/Export:

Image Translators

- The following translators convert from *Radiance*
  - **ra_pict** Macintosh PICT2 (24-bit) format
  - **ra_pixar** PIXAR image format
  - **ra_ps** PostScript 8-bit greyscale
  - **ra_gif** Compuserve 8-bit GIF

- The following translators convert both ways
  - **ra_bn** Barneyscan format
  - **ra_ppm** Poskanzer Portable Pixmap format
  - **ra_pr** Sun 8-bit rasterfile format
  - **ra_pr24** Sun 24-bit rasterfile format
  - **ra_rgbe** *Radiance* uncompressed format
  - **ra_t16** Targa 16-bit and 24-bit formats
  - **ra_t8** Targa 8-bit format
  - **ra_tiff** Tagged Image File Format (greyscale and 24-bit)

General Capabilities and Features

- Accurate calculation of luminance
  - most basic lighting unit
  - ∫ Luminance = Illuminance
  - luminance "map" = picture
- Models both electric light and daylight
  - uniform treatment of illumination sources
  - not limited to terrestrial simulations
- Supports a variety of reflectance models
  - reflectance can be any arbitrary BRDF
  - arbitrary transmittance functions also
  - optimized calculation for common materials
- Supports complicated geometry
  - curved surfaces and detailed geometry
  - no limit to geometric complexity other than memory
  - calculation time grows slowly with number of surfaces
- Takes unmodified input from CAD systems
  - no meshing or joining of surfaces necessary
  - no unreasonable surface-normal orientation requirements

General Capabilities and Features

- Light-backwards ray-tracing technique
  - start from point of measurement
  - work backwards towards light source(s)
  - main advantage is efficiency for radiance calculations
  - hierarchical octrees produce fast ray intersections in complicated environments

- Efficient calculation of indirect illumination
  - indirect irradiance values are calculated only as needed
  - values are stored and interpolated
  - interpolation uses gradient formula for better accuracy
  - values are view-independent and reusable

- Efficient calculation of direct illumination
  - adaptive sampling for scenes with many light sources
  - automatic subsampling of large area sources
  - automatic processing of "virtual" light sources
  - user-directed processing of "secondary" light sources

- Support for color, patterns and textures
  - uses basic RGB color model
  - patterns and textures may be procedural or data-driven
  - patterns also used for light source output distributions

- Parallel processing support

Calculation Procedures

- Monte Carlo indirect calculation
  - standard Monte Carlo techniques are too expensive, so...
  - indirect irradiance values are "cached" and interpolated

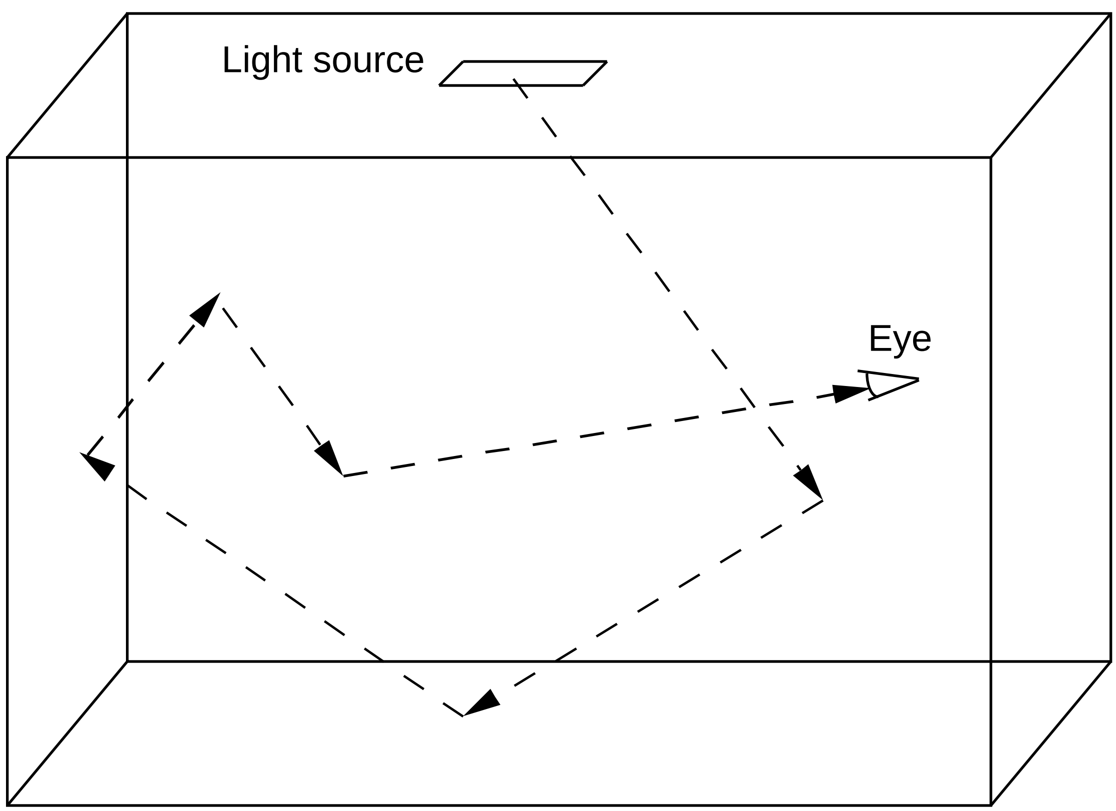
- Deterministic direct calculation
  - rays are followed directly to light sources
  - when there are many sources, only some rays are followed
  - especially large sources are subdivided for better sampling
  - reflected rays are follwed towards "virtual" light sources

- User-directed processing of "secondary" sources
  - windows, skylights, etc. pose a special sampling problem
  - better efficiency is achieved by treating them as sources
  - the user specifies which surfaces to treat, *Radiance* does the rest

Monte Carlo

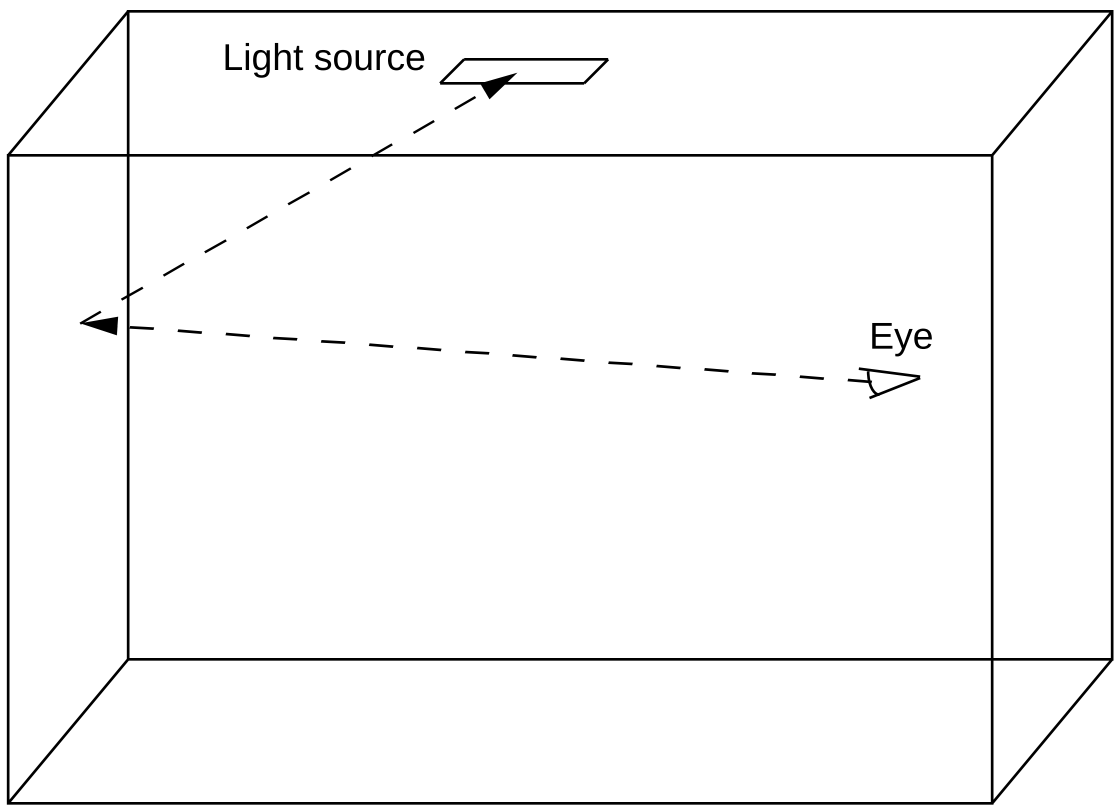
Q: What is being computed?
A: Light bouncing around until it hits a certain point.

Q: What is the most straight-forward calculation?
A: A Monte Carlo simulation of photons, i.e.



...and repeat about a billion times!

Q: Is there a faster way?
A: Almost anything would be faster. One simple trick is to start from the eye instead of the light source. That way, we only calculate those rays which are likely to affect the final result, i.e.



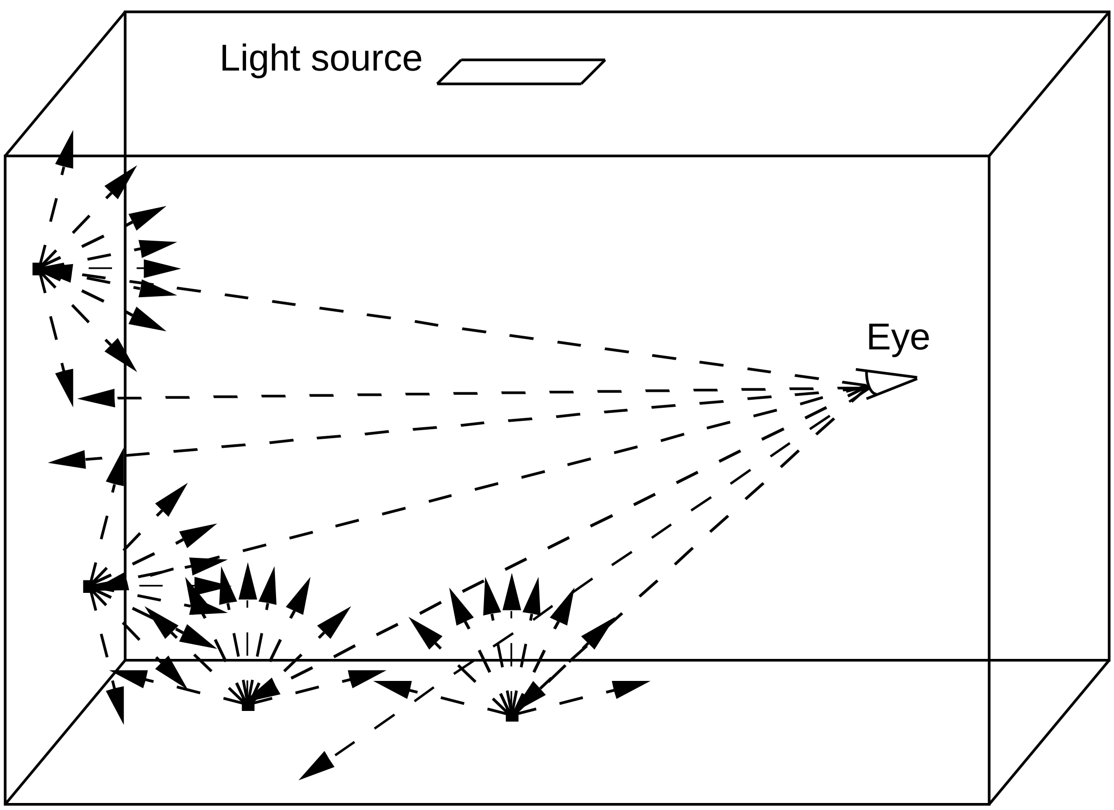
Calculation Procedures **Monte Carlo**

Q: That looks too easy. Aren't we forgetting something?

A: Yes, we are. Diffuse interreflection between surfaces is being left out.

Q: What do we do about it?

A: We go back to Monte Carlo, but only every now and then, i.e.



...notice that only <u>some</u> of the rays traced from the eye result in Monte Carlo evaluations

Direct

Q: What is meant by the "direct" component?
A: It is that component that arrives (more or less) directly from light sources.

Q: Why is the direct component treated separately?
A: Because its contribution is the most significant and if it is computed carefully, the whole calculation will be more efficient.

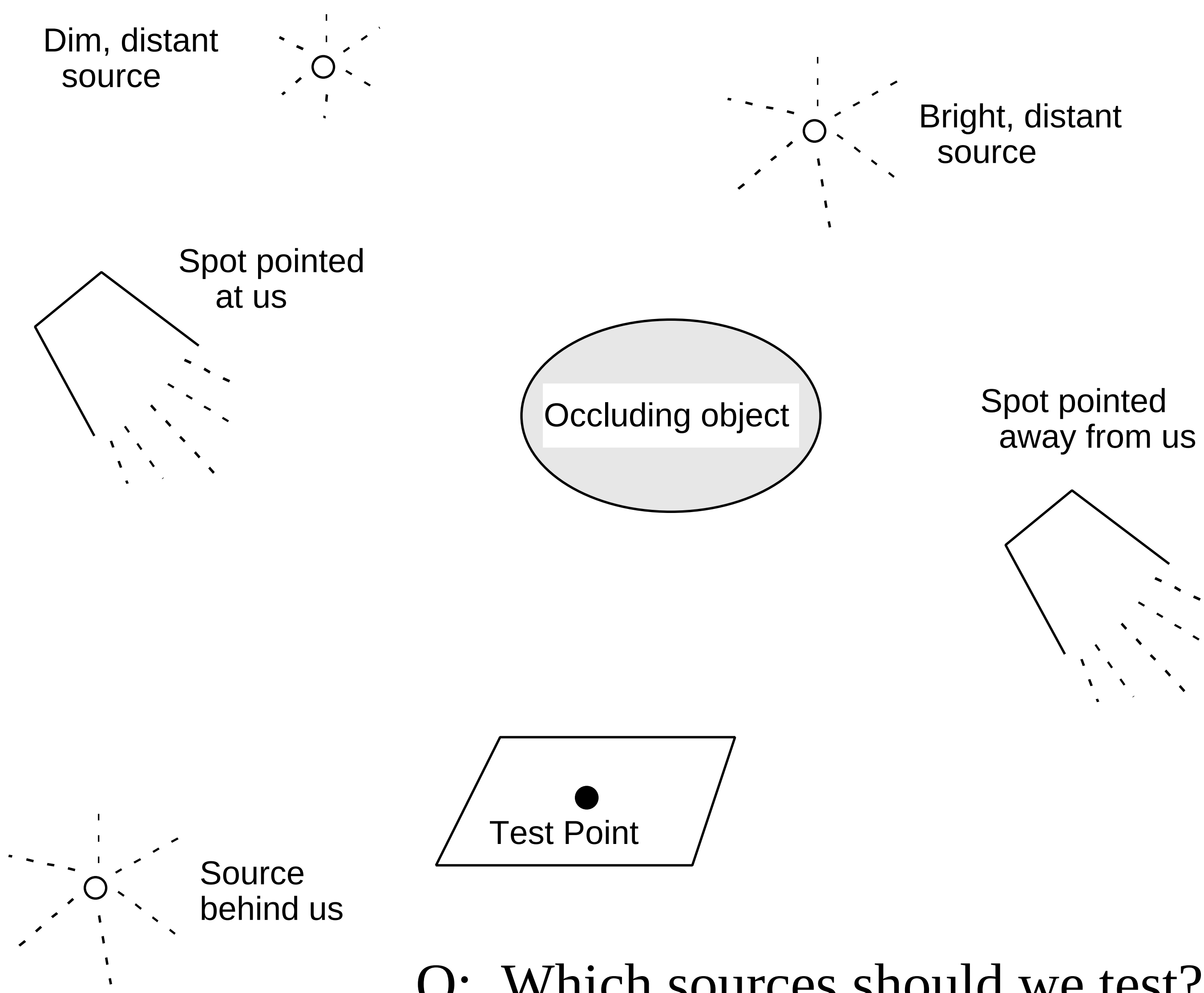
Q: What is a light source in *Radiance*?
A: Light sources include the usual electric lights, sun, certain specular reflections of light sources, and sometimes window systems and skylights. They are recognized by their material type.

Q: When is the direct component computed?
A: Every time a ray reaches a scattering surface.

Calculation Procedures **Direct**

- Source sampling would normally grow linearly with the number of light sources

- We want to avoid linear growth, so we only sample those sources which might have significant influence on the current point



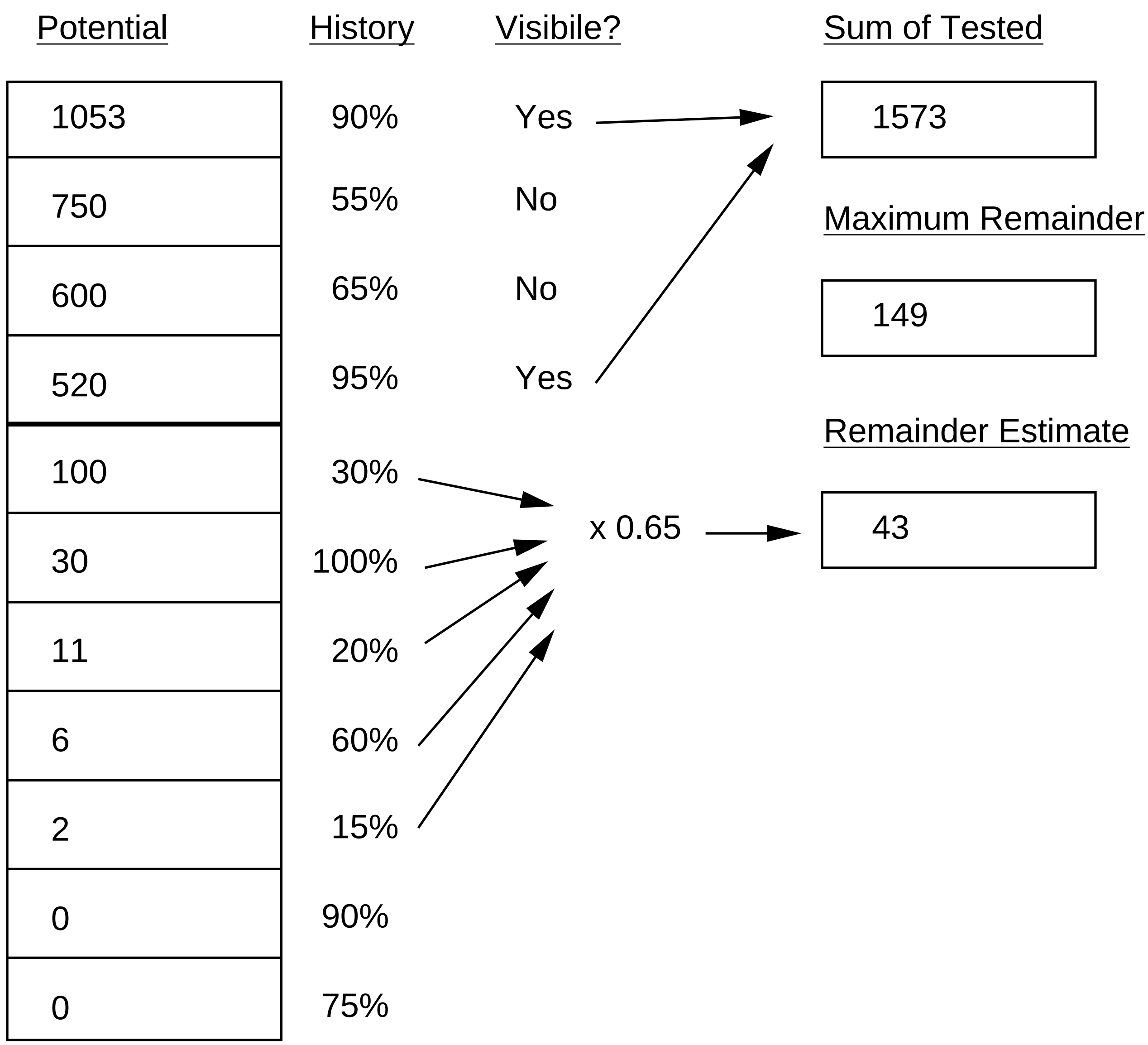
Q: Which sources should we test?

Calculation Procedures

Direct

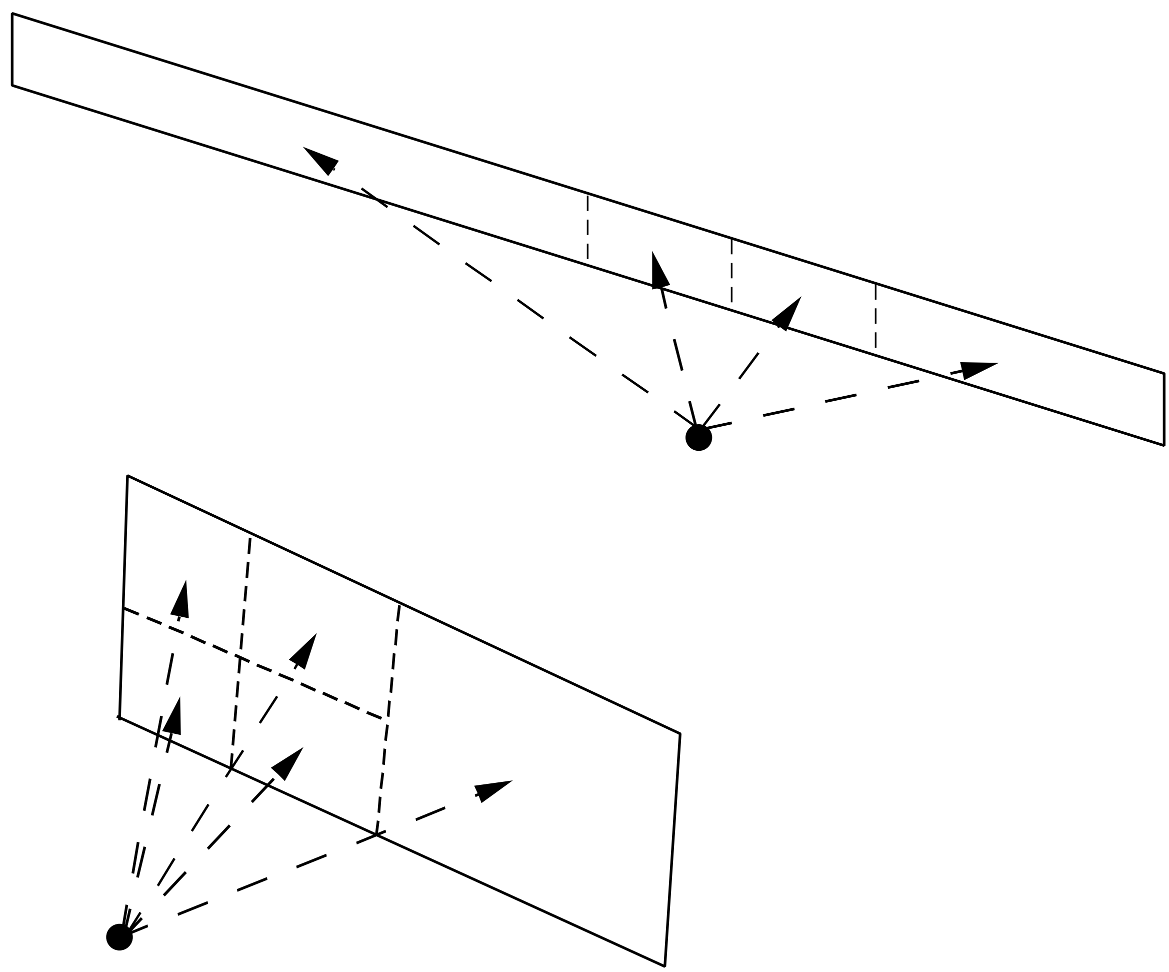
Q: How do we make these determinations in general?

A: We sort potential direct contributions and test them in order, i.e.

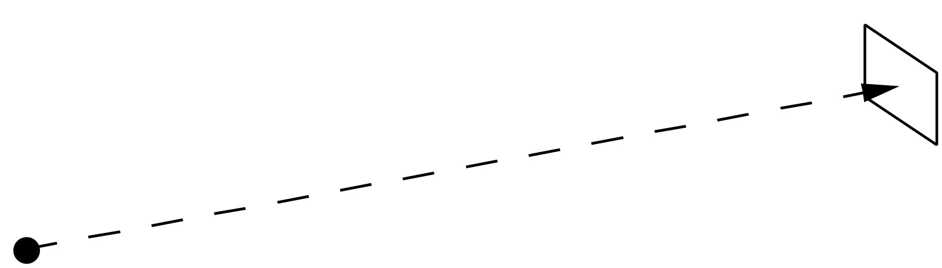


Calculation Procedures **Direct**

- Large sources are adaptively subdivided

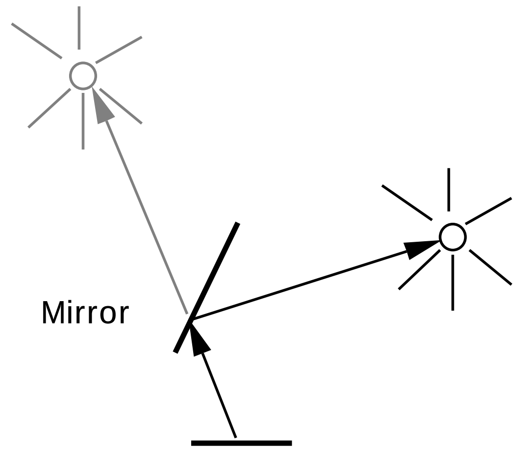


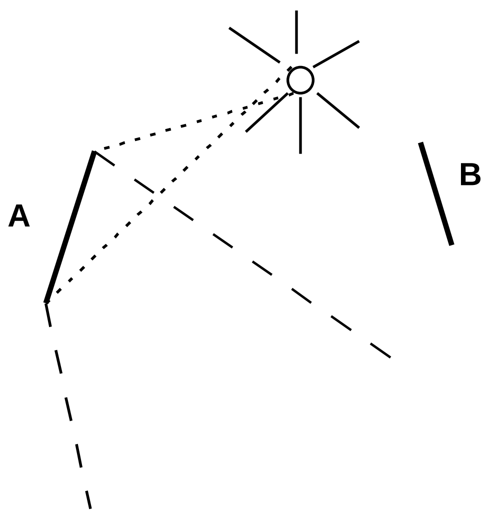
- A source that is small relative to its distance will not be subdivided



• Virtual Light Sources

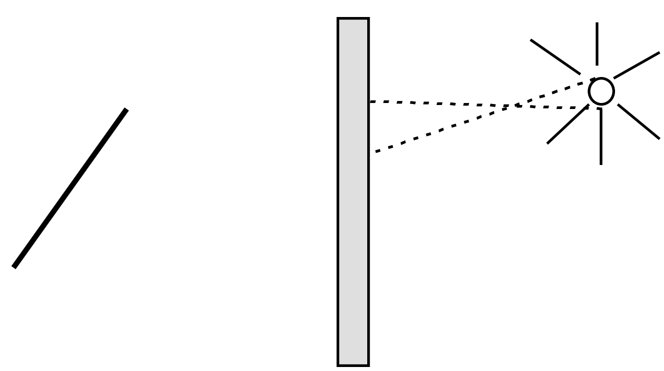
Virtual source caused
by mirror reflection.





Source reflection in mirror A
cannot intersect mirror B, so no
virtual source is created.

Source rays cannot reach
mirror surface, so no virtual
source is created.



Secondary Sources

- Secondary sources are objects that for one reason or another transfer a large amount of light into our space

- It is not technically necessary to do anything special about such objects, but it can make the calculation more efficient

- *Radiance* does not have the intelligence to figure out what objects should and shouldn't be treated as secondary sources

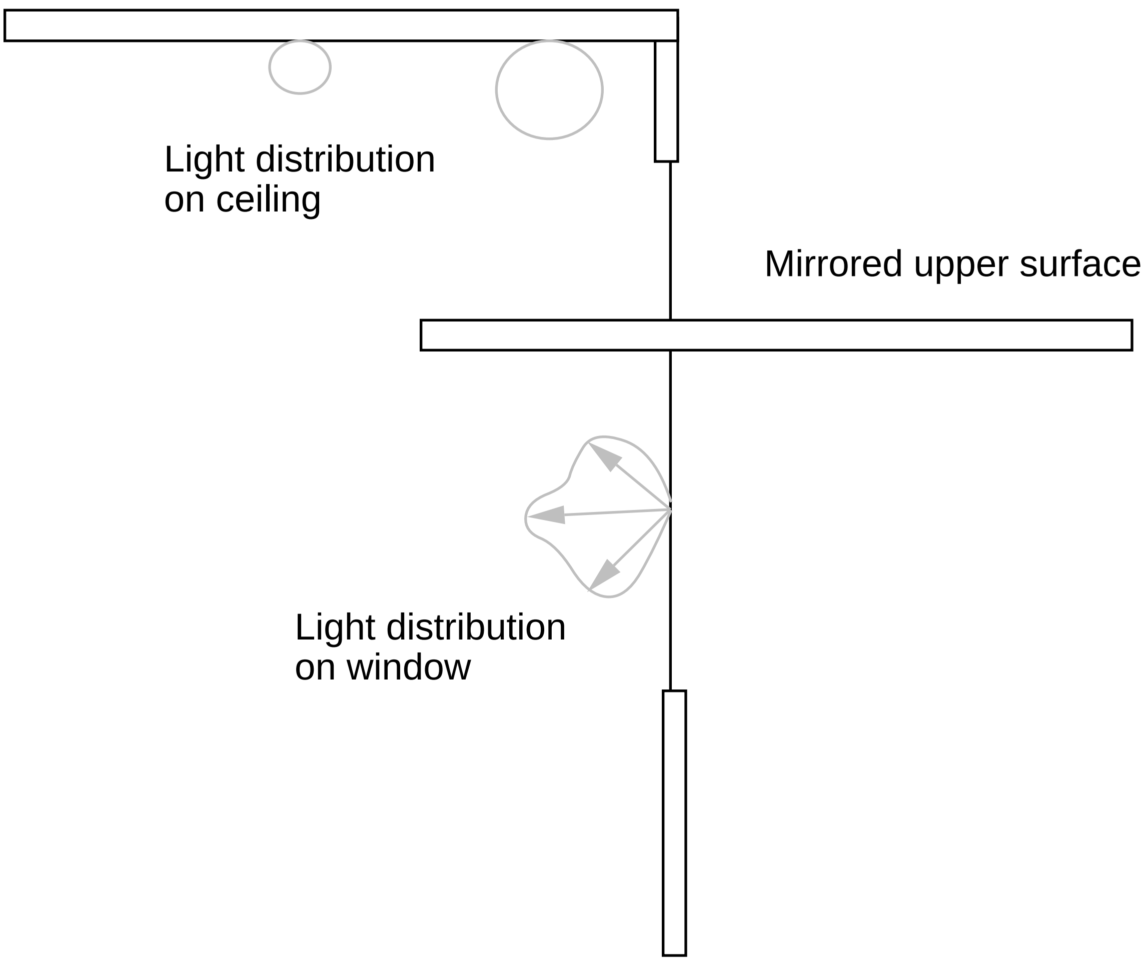
- It is therefore up to the user to decide if and when an object should be made into a secondary light source

- Once the user has submitted an object for treatment as a secondary source, **mkillum** computes its distribution and the renderers do the rest

Calculation Procedures

Secondary Sources

A crossection of office space with mirrored light shelf



Validation Work

- *Radiance* has been compared to measurements and to other simulation programs

- An initial validation study compared *Radiance* to *Lumen Micro*, *SUPERLITE* , and scale models measured in a sky simulator

- Another form of validation compared *Radiance* renderings to actual photographs

- A more recent study compared *Radiance* calculations to scale model measurements taken under beam illumination

- Others have also compared *Radiance* to their own simulations and measurements