An Introduction to 3D Computer Graphics, Stereoscopic Image, and Animation in OpenGL and C/C++

Fore June

Appendix C An Open-Source Ray Tracing Package

C.1 Radiance

RADIANCE, an open-source package, is a a suite of programs that generates highly accurate ray-tracing scenes, and is readily to be run in UNIX platforms. It was developed with primary support from the U.S. Department Of Energy and additional support from the Swiss Federal Government, and its copyright owner is the Regents of the University of California. Detailed information of the package can be found at its official web site:

http://radsite.lbl.gov/radiance/framew.html

where one can download the complete package of it. It is easy and straight forward to install and use.

Radiance yields better results over simpler lighting calculation and rendering tools as it has no limitations on the geometry or materials for simulations. Architects and engineers use Radiance to predict illumination, visual quality and appearance of innovative design spaces, and researchers use it to evaluate new lighting and daylighting technologies.

The following are some features of Radiance:

1. Input parameters such as the scene geometry, materials, luminaires, time, date and sky conditions (for daylight calculations) can be specified in files.
2. Typical ray tracing outputs such as spectral radiance (ie. luminance + color), irradiance (illuminance + color) and glare indices are calculated.
3. Simulation results may be displayed as color images, numerical values or contour plots.
4. Generator programs are provided for the creation of more complex shapes from the basic surface primitives, such as boxes, prisms and surfaces of revolution.
5. A transformation utility permits the simple duplication of objects and hierarchical construction of a scene.

Tutorials and usage of the package can be found at the sites:

1. *http://radsite.lbl.gov/radiance/refer/tutorial.html*, or
2. *http://netghost.narod.ru/gff/vendspec/radnce/tutorial.txt*

Examples of using this package presented in the next section are based on these tutorials.

C.2 Using Radiance

In this section, we present a brief tour of using Radiance. To get an image, we need a minimum input of a source of illumination and an object to reflect light to the *camera*. One can think of a Radiance renderer as an invisible camera observing a simulated world. As presented in the first example below, we begin with two spheres, one emissive and one reflective. We first define the materials, then the spheres themselves.

I. First Example

This example shows the basic formats. A scene description file represents a three-dimensional physical environment in Cartesian world coordinates, stored as ASCII text, with the following basic format:

```
# comment
```

```
modifier type identifier
n S1 S2 "S 3" .. Sn
0
m R1 R2 R3 .. Rm

modifier alias identifier reference

! command
...
```

The following are the steps to create a red plastic sphere shone by bright light. The keyword **light** defines the basic material for self-luminous; in the example, its (R, G, B) values are $(100, 100, 100)$. The **plastic** material requires 5 parameters: (red, green, blue, specularity, roughness). A **sphere** is defined by its center and radius; so $(7\ \ 1.125\ \ .625\ \ .125)$ means its center is at $(7, 1.125, .625)$ and its radius is .125.

1. Use a text editor (e.g. vi) to create a file named *room0.rad* with the following code:

```
# My first scene.

# The basic primitive format is:
#
# modifier TYPE identifier
# number_of_arguments [arguments...]
# arguments are of the same type, and
# could be strings, integers or reals
#
#The special modifier "void" means no modifier.

# Material for my light source:
void light bright
0
0
3 100 100 100
 # Material for my test ball:
 void plastic red_plastic
 0
 0
 5 .7 .05 .05 .05 .05

 # Here is the light source:
 bright sphere fixture
 0
 0
 4 2 1 1.5 .125

 # Here is the ball:
 red_plastic sphere ball
 0
 0
 4 .7 1.125 .625 .125
```

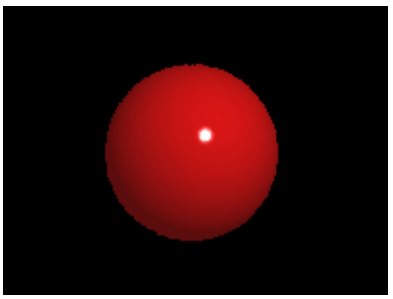


Figure C-1 First Example

2 Create Octree. As now we have a simple scene description, we can examine it using the interactive viewing program, *rview*. However, we need to first create an **octree** file, which will be used to speed up the rendering process. This is done by the command:

```
$ oconv room0.rad > test0.oct
```

Note that the file extensions *.rad* and *.oct* are not enforced, but merely a convenience for recognizing the files.

3 Get information about the Octree. The command **getinfo** can be used to retrieve information from the unviewable binary *.oct* file. Try entering the command:

```
$ getinfo test0.oct
```

4 Use the command **rview** to render the scene:

```
$ rview -vp 2.25 .375 1 -vd -.25 .125 -.125 -av .5 .5 .5 test0.oct
```

The output is shown in Figure C-1. Note that within the *rview* display menu, we can save the viewing configuration to the file *default.vp* (or a file with other file names), by entering the following command near the bottom of the display screen:
: view default.vp
We can retrieve a view file with the command:
: last *filename*
Details of the command *rview* can be found at the site:
http://www.siggraph.org/education/materials/HyperGraph/raytrace/radiance/man_html/rview.1.html

II. Second Example:

1. Modify *room0.rad* to *room1.rad* by adding the following code:

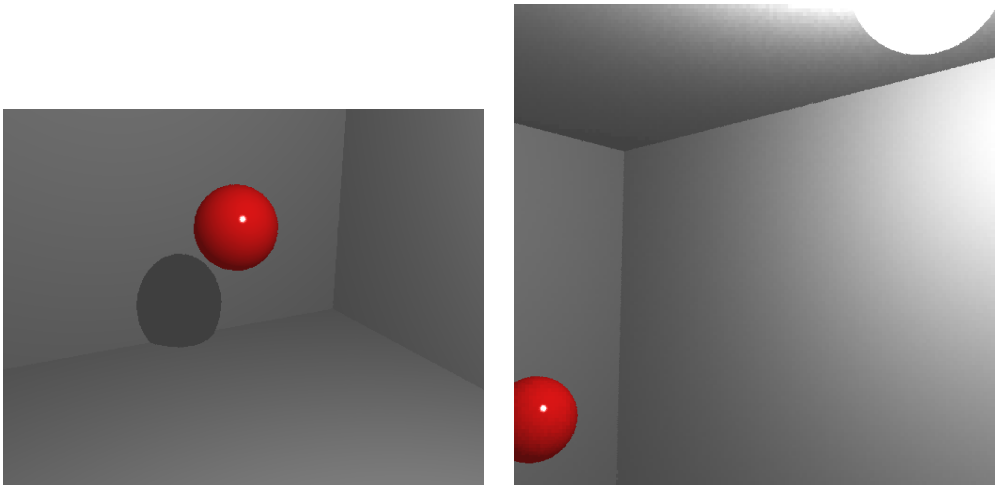
```
# the wall material:
void plastic gray_paint
0
0
5 .5 .5 .5 0 0

# a box shaped room:
!genbox gray_paint room 3 2 1.75 -i
```

The command **genbox** generates a box (6 rectangles) with lower left corner at $(0, 0, 0)$, and upper right corner at $(3, 2, 1.75)$.

2. Convert to octtree:
$ oconv room1.rad > test1.oct

3. Render the scene with configuration file *default.vp*:
$ rview -vf default.vp -av .5 .5 .5 test1.oct
An output is shown in Figure C-2(a).

4. Rotate scene with command **rotate** or **r** at the render prompt. Example:
:r 5 -20
A sample output is shown in Figure C-2(b).



(a) (b)

Figure C-2 Example 2

III. Third Example:

1. Modify *room1.rad* to *room2.rad* by adding more features:

```
# a shiny blue box:
void plastic blue_plastic
0
0
5 .1 .1 .6 .05 .1

!genbox blue_plastic box .5 .5 .5 |xform -rz 15 -t .5 .75 0

# a chrome rod to suspend the light
# from the ceiling:
void metal chrome
0
0
5 .8 .8 .8 .9 0

chrome cylinder fixture_support
0
0
7
   2    1    1.5
   2    1    1.75
   .05
#Note: cylinder has 7 parameters: x0 y0 z0
#                                 x1 y1 z1  radius
# denoting the top and bottom disc centers, and radius
```

2. Convert and render:

```
$ oconv room2.rad > test2.oct
$ rview -vf default.vp \
     -av .5 .5 .5 test2.oct
```

A sample output is shown in Figure C-3.

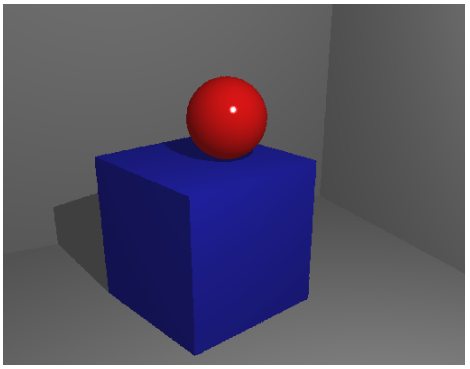


Figure C-3 Third Example

IV. Example 4:

1. Try different materials, modifying *room2.rad* to room3.rad with the following changes:

```
#
# room3.rad
#

# 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:

# solid crystal:

void dielectric crystal
0
0
5 .5 .5 .5 1.5 0

# dark brown:

void plastic brown
0
0
5 .2 .1 .1 0 0

# light gray:

void plastic white
0
0
5 .7 .7 .7 0 0

# here is the light source:
```

```
  bright sphere fixture
  0
  0
  4 2 1 1.5 .125

  # here is the ball:

  crystal 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

# a shiny blue box:

void plastic blue_plastic
0
0
5 .1 .1 .6 .05 .1
```

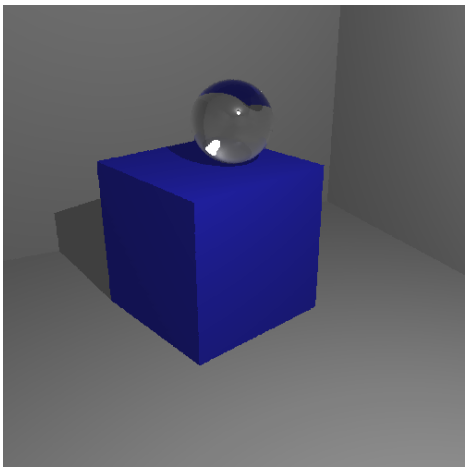


Figure C-4 Example 4

```
!genbox blue_plastic box .5 .5 .5 |xform -rz 15 -t .5 .75 0

# a chrome rod to suspend the light from the ceiling:

void metal chrome
0
0
5 .8 .8 .8 .9 0

chrome cylinder fixture_support
0
0
7
    2    1    1.5
    2    1    1.75
    .05
```

2. Convert and render:

```
$ oconv room3.rad > test3.oct
$ rview -vf default.vp -av .5 .5 .5 test3.oct
```

Note "—" means piping. The statement
xform -rz 15 -t .5 .75 0
means to transform the object by rotating the scene by 15^o about the z-axis, and translating the scene by $(.5, .75, 0)$.

3. One can create an image from the scene using the command *rpict* like:

```
$ rpict -vf default.vp -av .5 .5 .5 test3.oct > test3.pic
```

   You may convert the *.pic* file to a *.png* file using the Unix utility *convert* like:

```
$ convert test3.pic test3.png
```

   An output is shown in Figure C-4.

V. Example 5:

This example shows how to change the color of ceiling and floor.

1. Generate room externally:

```
$ genbox gray_paint room 3 2 1.75 -i >> room4.rad
```

2. Edit the file *room4.rad* to make floor (all z values = 0) with brown color:

```
genbox gray_paint room 3 2 1.75 -i >> room4.rad
```

3. and make ceiling (all z values = 1.75) white:

```
white polygon room.6457
```

4. An output of the scene is shown in Figure C-5.

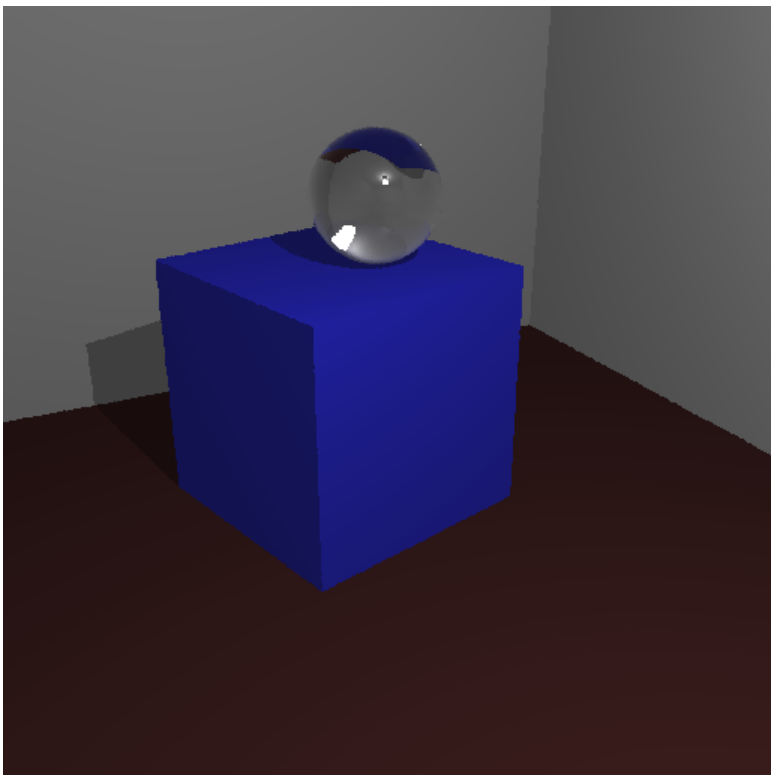


Figure C-5 Example 5

VI. Example 6:

This example shows how to add a window to the previous example. This is a two-steps process.

1. Copy *room4.rad* to *room5.rad*.
2. Step 1:
   (a) Cut a hole in the wall and put in a piece of glass. (Use coincident edges to make a seam to give the appearance of a hole.) Edit *room5.rad*, changing *room.5137* to:

```
gray_paint polygon room.5137
0
0
30

 3      2      1.75
 3      2      0
 3      0      0
 3      0      1.75
 3      .625   1.75
 3      .625   .625
 3      1.375  .625
 3      1.375  1.375
 3      .625   1.375
 3      .625   1.75
```

   (b) Create a separate file named *window.rad* for the widnow:

```
# transmittance glass window has a transmission of 96%:
void glass window_glass
0
0
3 .96 .96 .96

window_glass polygon window
0
0
12
3      .625       1.375
3      1.375      1.375
3      1.375      .625
3      .625       .625
```

3. Step 2:
   (a) Create a scene outside the window (sky and sun). Save the scene in a separate file named *sky.rad*:

```
$ gensky 3 20 10 -o 0 -m 0 -a 40 > sky.rad
```

   (b) Create a file named *outside.rad* that contains a generic background of the outside scene:

```
#Standard sky and ground to follow a gensky sun & sky distribution.
```

```
skyfunc glow sky_glow
0
0
4 .9 .9 1.15 0

sky_glow source sky
0
0
4 0 0 1 180

skyfunc glow ground_glow
0
0
4 1.4 .9 .6 0
ground_glow source ground
0
0
4 0 0-1 180
```

(c) Put everything in one octree:

```
$ oconv sky.rad outside.rad window.rad room5.rad > test5.oct
```

(d) Render with:

```
$ rview -vf default.vp -av .5 .5 .5 test5.oct
```

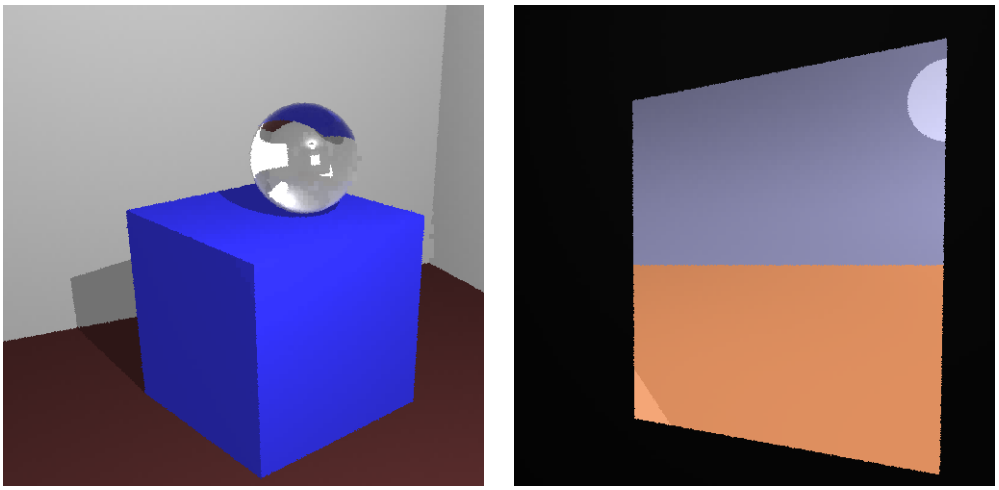
We may adjust the illumination (exposure) with the render command ":e" (e.g. e 0.8). Figure C-6(a) shows an image of the scene with exposure 0.8 and rendered with:

```
$rview -vtv -vp -2.25 -0.375 1 -vd 0.25 -0.125 -0.125\
         -vu 0 0 1 -vh 45 -vv 45 -vo 0 -va 0 -vs 0 -vl 0
```

Figure C-6(b) is with eposure 0.4 and rendered with:

```
$rview -vtv -vp 1.7823 0.15627 0.99132 -vd 0.23949 0.19006 -0.01653\
           -vu 0 0 1 -vh 45 -vv 45 -vo 0 -va 0 -vs 0 -vl 0
```

We save the configuration file for rendering Figure C-6(b) as *test5b.vp*.



(a) (b)

Figure C-6 Example 6

VII. Example 7:

This example adds illumination to the window.

1. Copy *window.rad* to *srcwindow.rad* and add the following:

```
# An emissive window

# visible glass window_glass

void glass window_glass
0
0
3 .96 .96 .96
 # window distribution function, including angular
 # transmittance:
 skyfunc brightfunc window_dist
 2 winxmit winxmit.cal
 0
 0

 # illum for window,using 88%
 # transmittance at normal
 # incidence:
 window_dist illum window_illum
 1 window_glass
 0
 3 .88 .88 .88

 # the source polygon:
 window_illum polygon window
 0
 0
 12
    3        .625     1.375
    3       1.375     1.375
    3       1.375      .625
    3        .625      .625
```

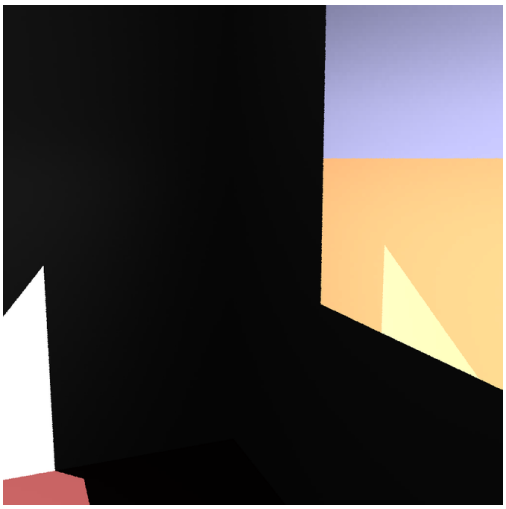


Figure C-7 Example 7

2. Compile with:

```
$oconv sky.rad outside.rad srcwindow.rad room5.rad > test5s.oct
```

3. Render with:

```
$rview -vf test5b.vp -av .5 .5 .5 test5s.oct
```

An ouput is shown in Figure C-7.