13th International Radiance Workshop 2014
London, UK
1-3 September 2014

Ambient Calculation
Crash Course

John Mardaljevic

Professor of Building Daylight Modelling
School of Civil & Building Engineering
Loughborough University, UK

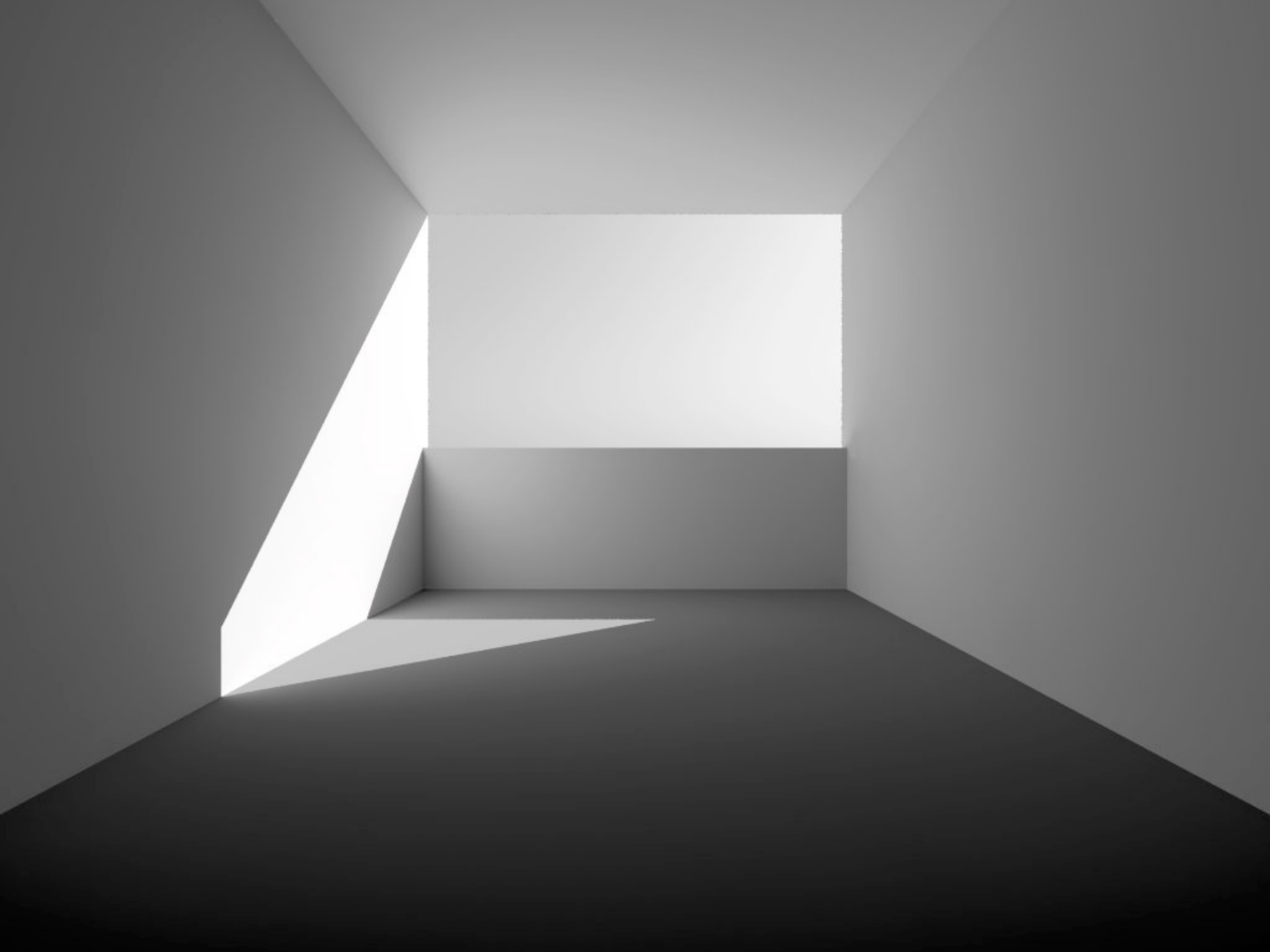
ROYAL ACADEMY OF ENGINEERING
Centre of Excellence for Sustainable Building Design

Loughborough University

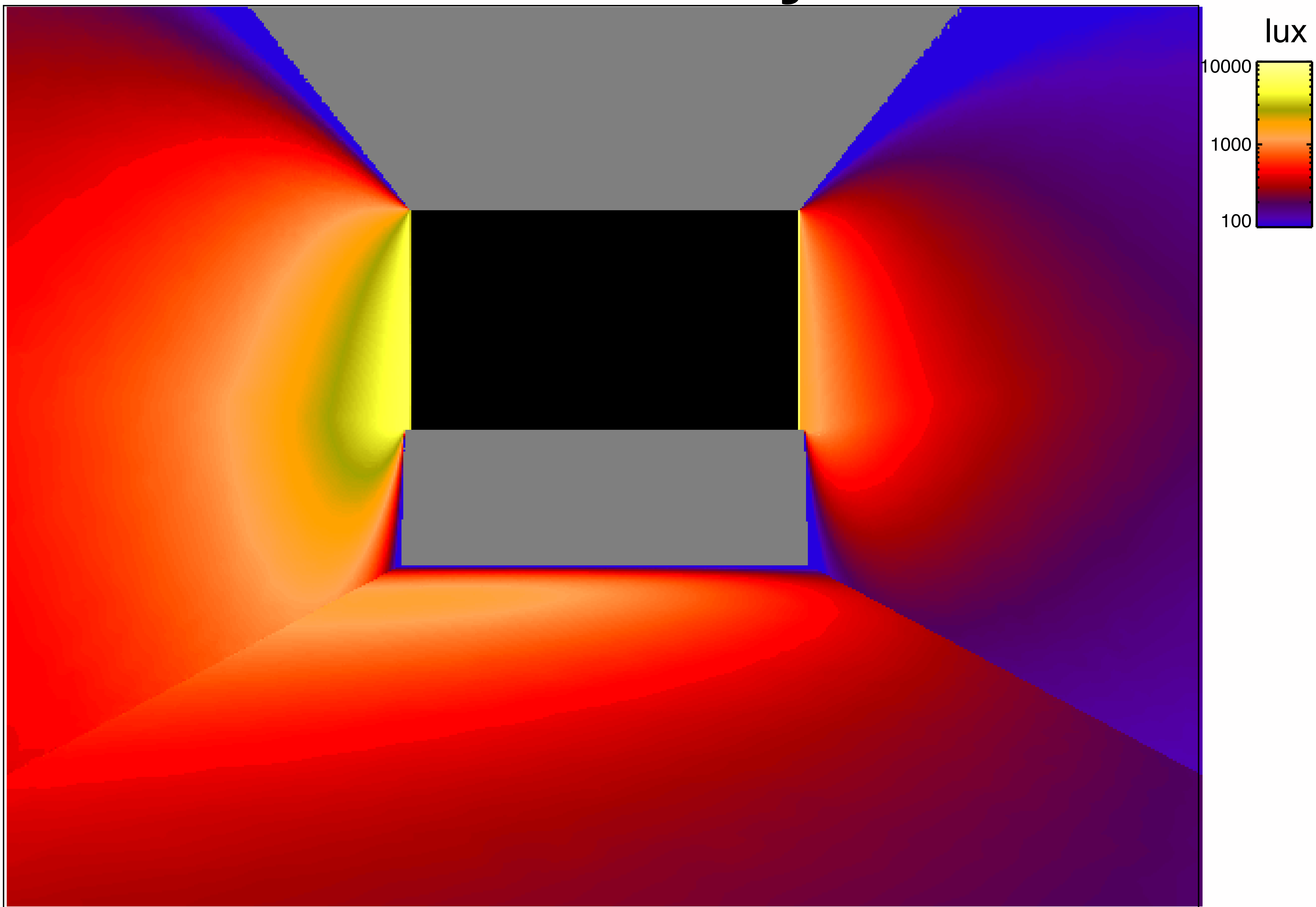
Lighting simulation is a **hunt for light**

There are several approaches we can use to hunt for light

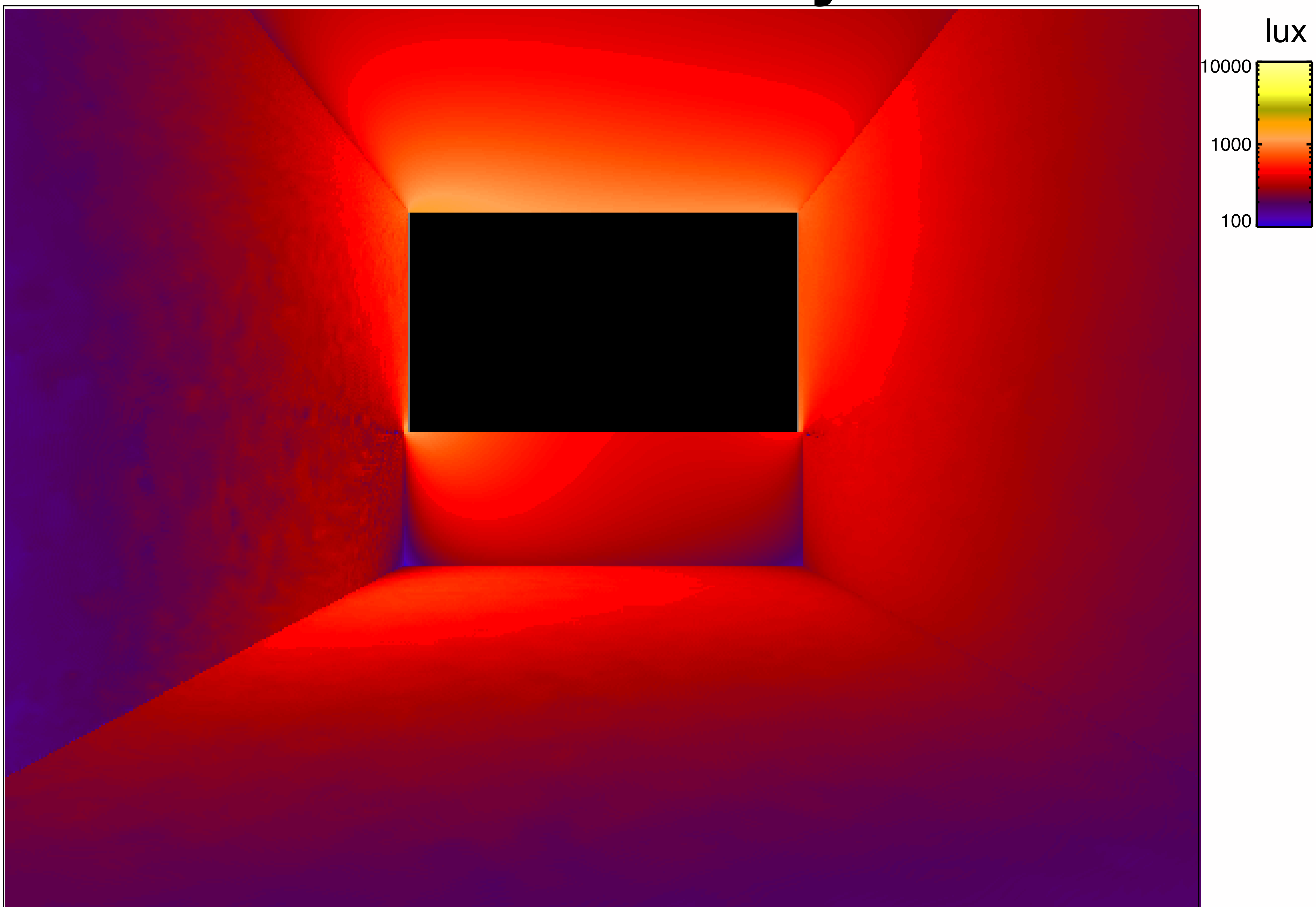
- Ray tracing - forward or backwards
- Radiosity
- Others: Photon mapping, etc.



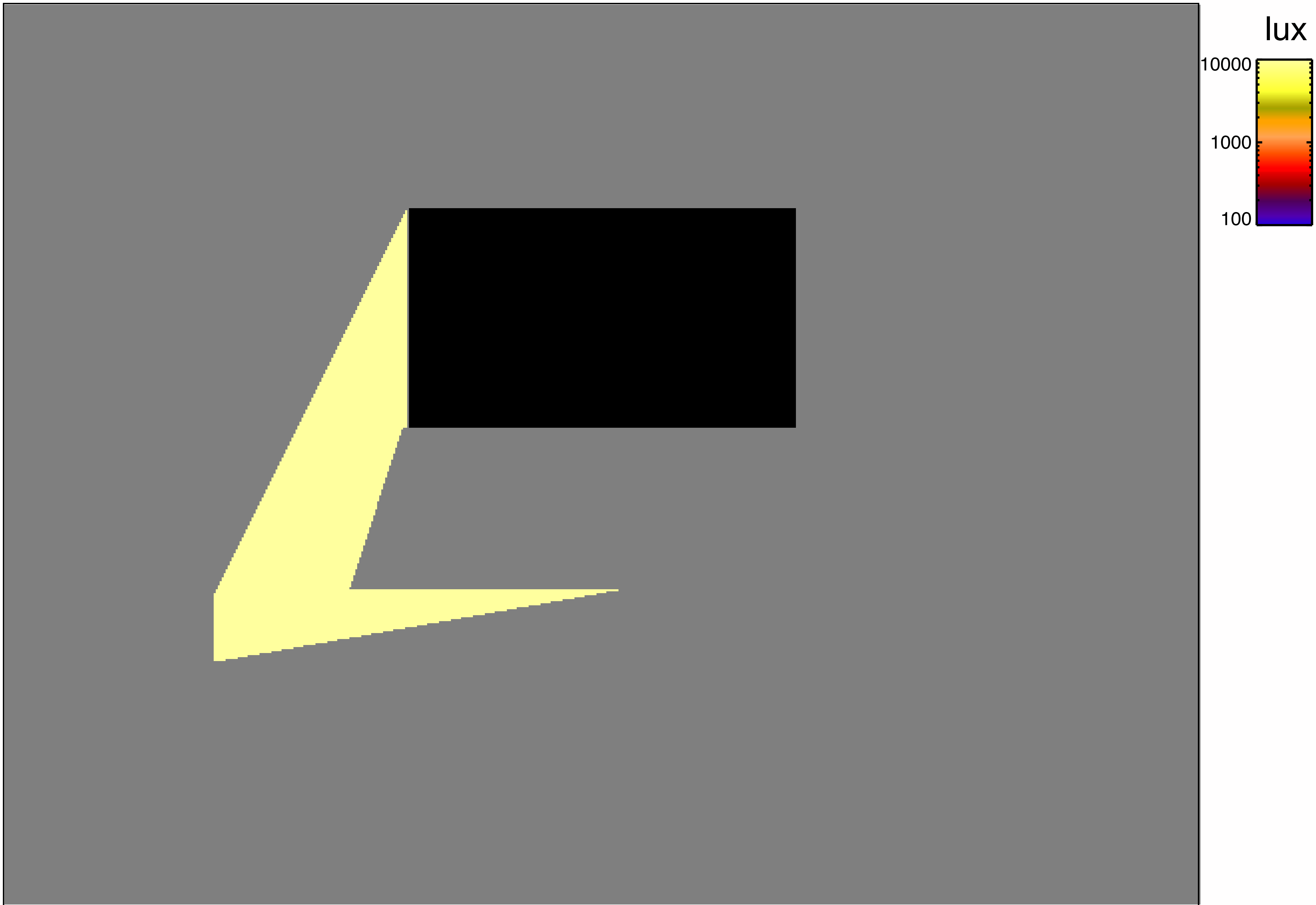
Direct sky



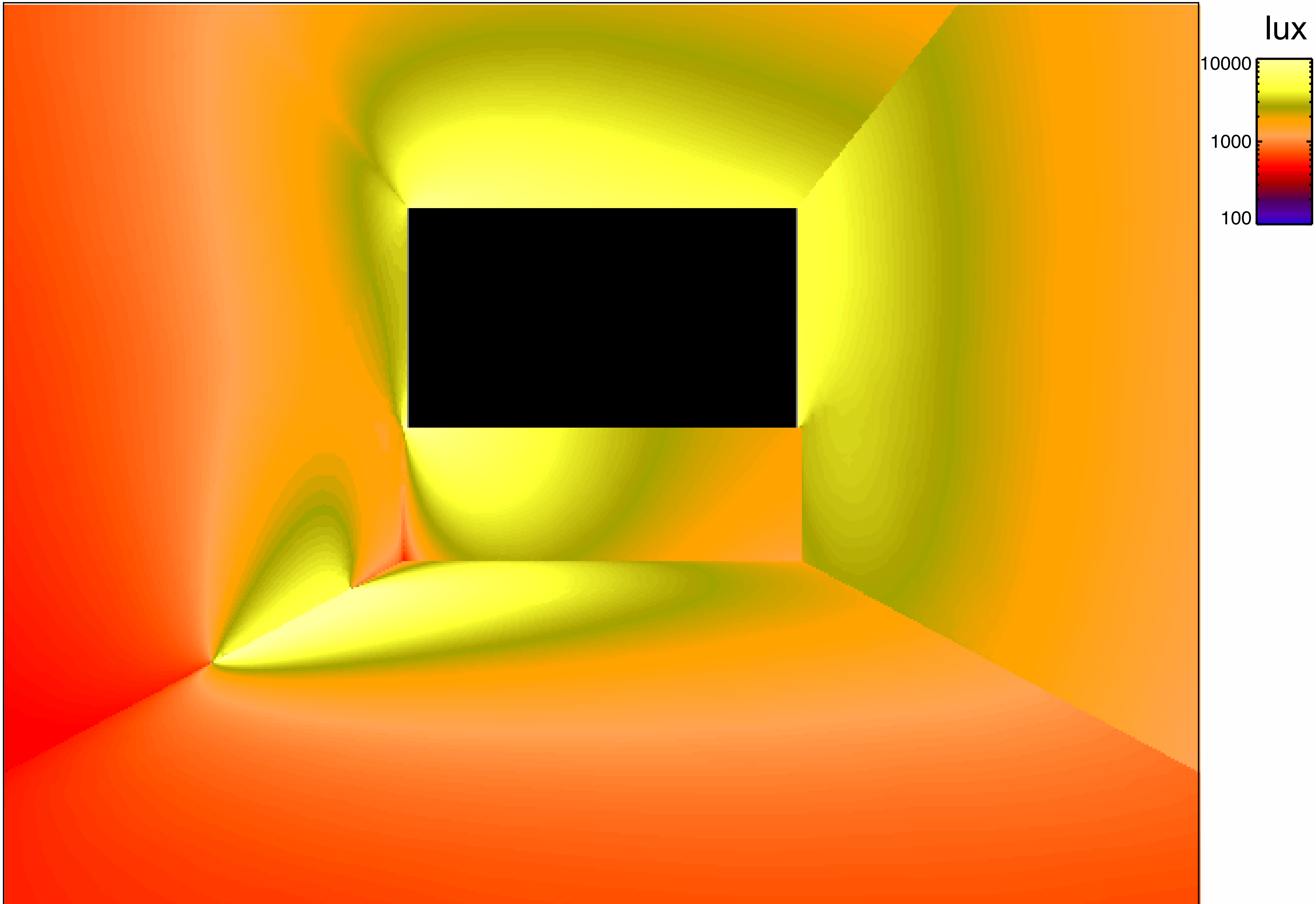
Indirect sky

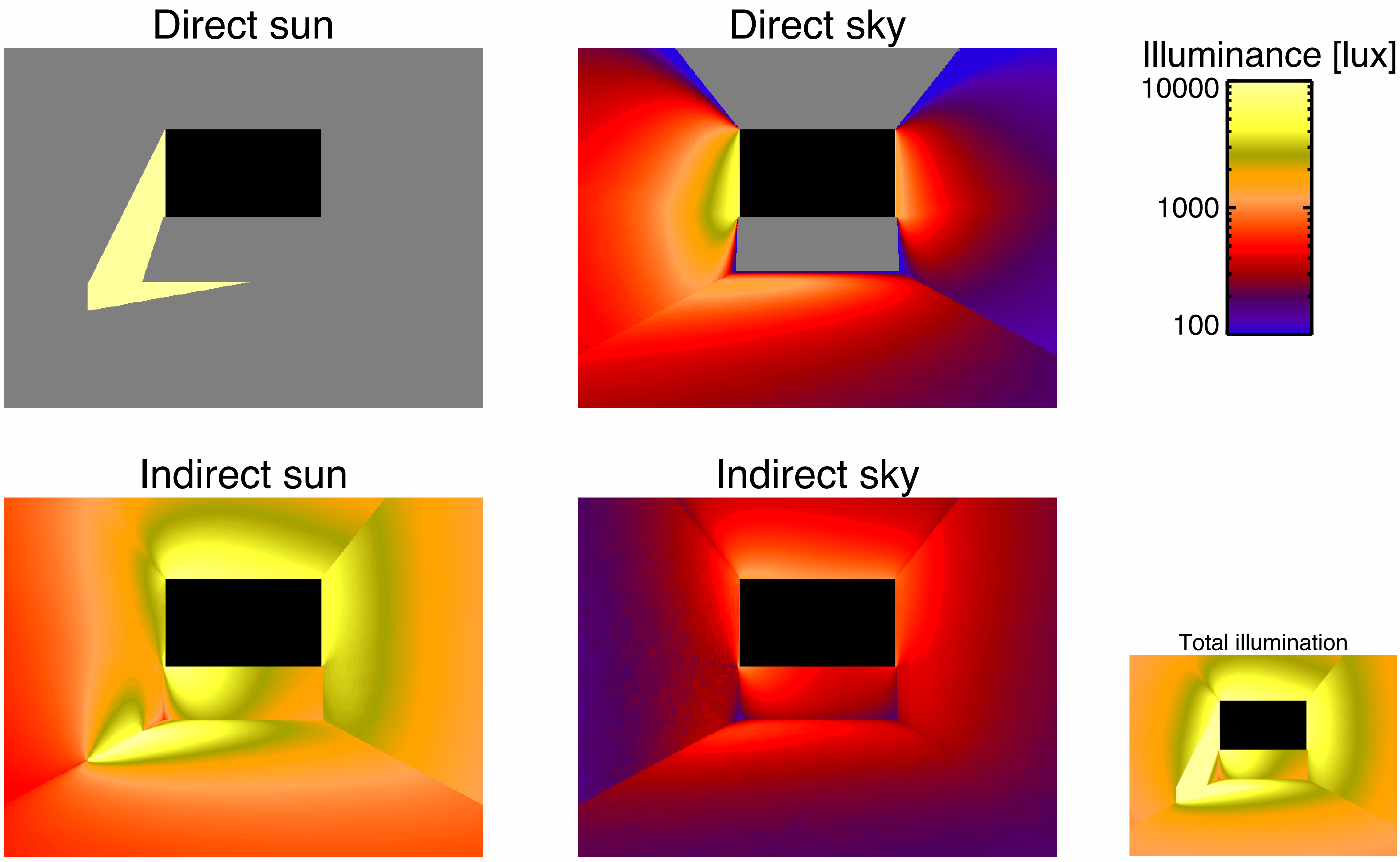


Direct sun

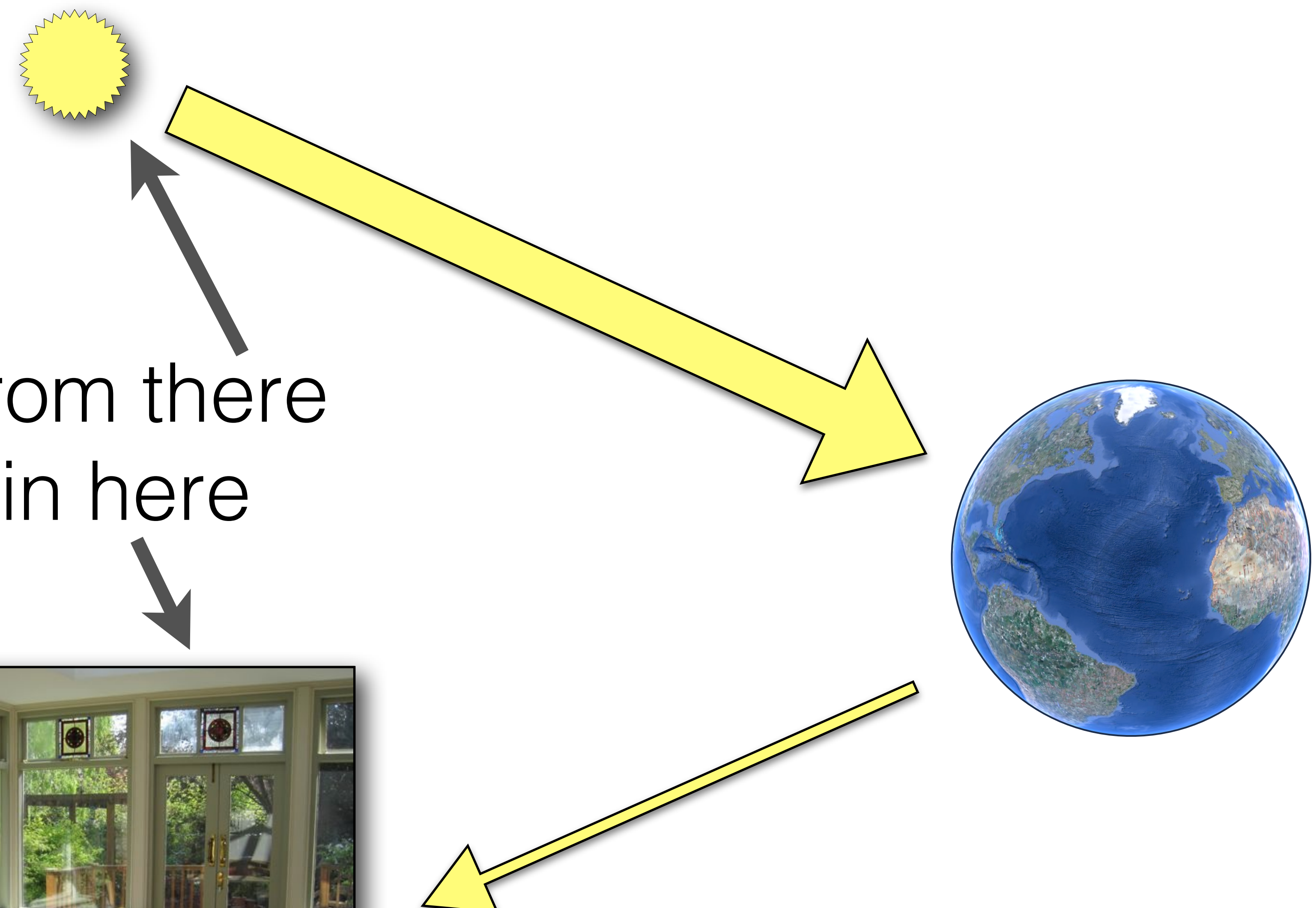


Indirect sun




Direct sun
Direct sky
Illuminance [lux]
10000
1000
100
Indirect sun
Indirect sky
Total illumination

Ray tracing: forwards or backwards?



Very little from there
ends up in here

Radiance uses backwards RT

Sunlight (beam radiation) can be intense and comes (usually) from one direction

Skylight and reflected light (from sun and sky) can come from all directions

Radiance treats the components of light differently

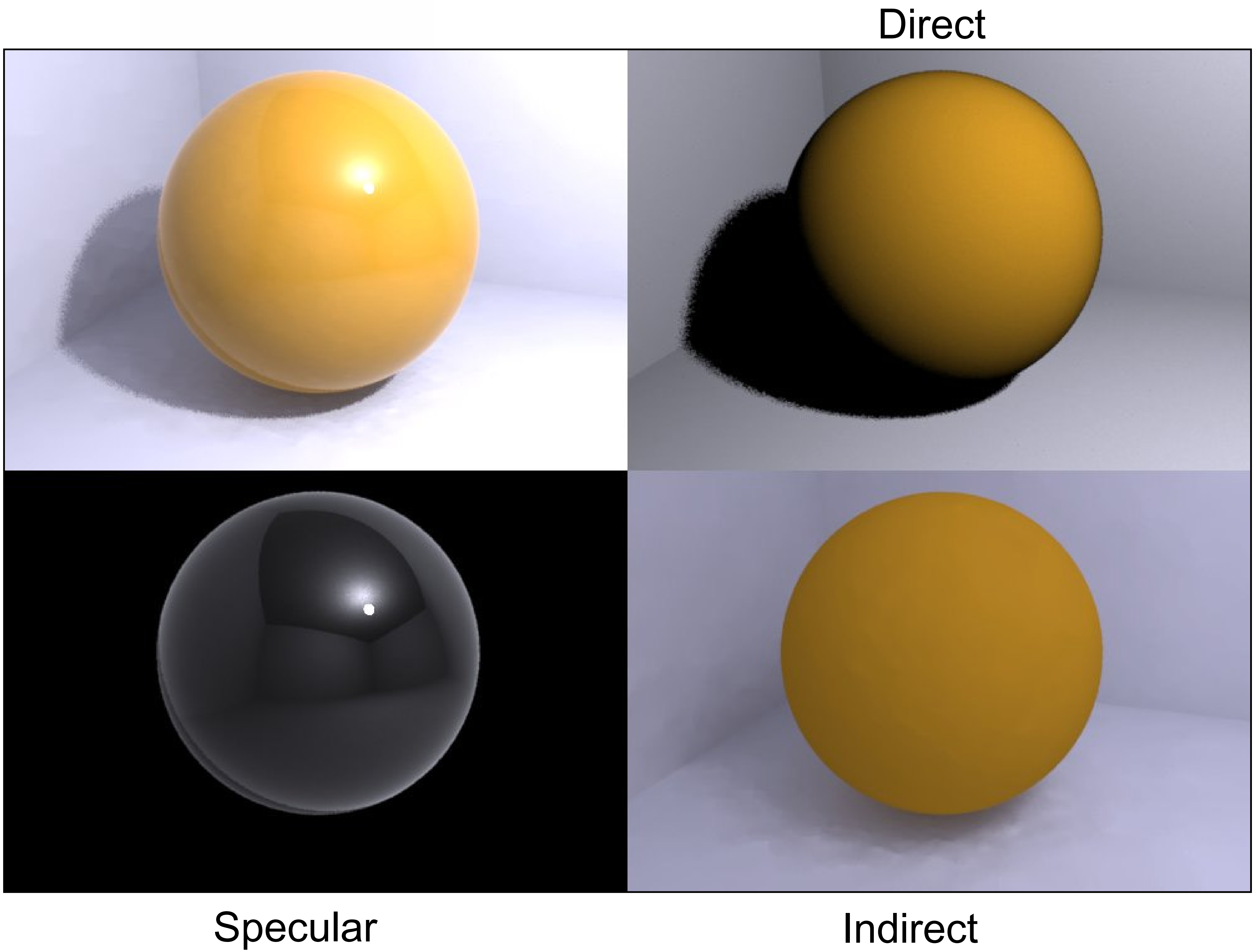


Plate 23 Rendering with Radiance

We “hunt” using different tactics depending on the source of illumination

- A deterministic method for the direct contribution from “concentrated” (i.e. **direct**) sources of light, e.g. sun or luminaire.

- A random (or stochastic or Monte-Carlo) method to “hunt” for light that could arrive from any direction (e.g. skylight or any type of reflected light). In *Radiance* this is done using **hemispherical sampling**.

Deterministic and hemispherical sampling

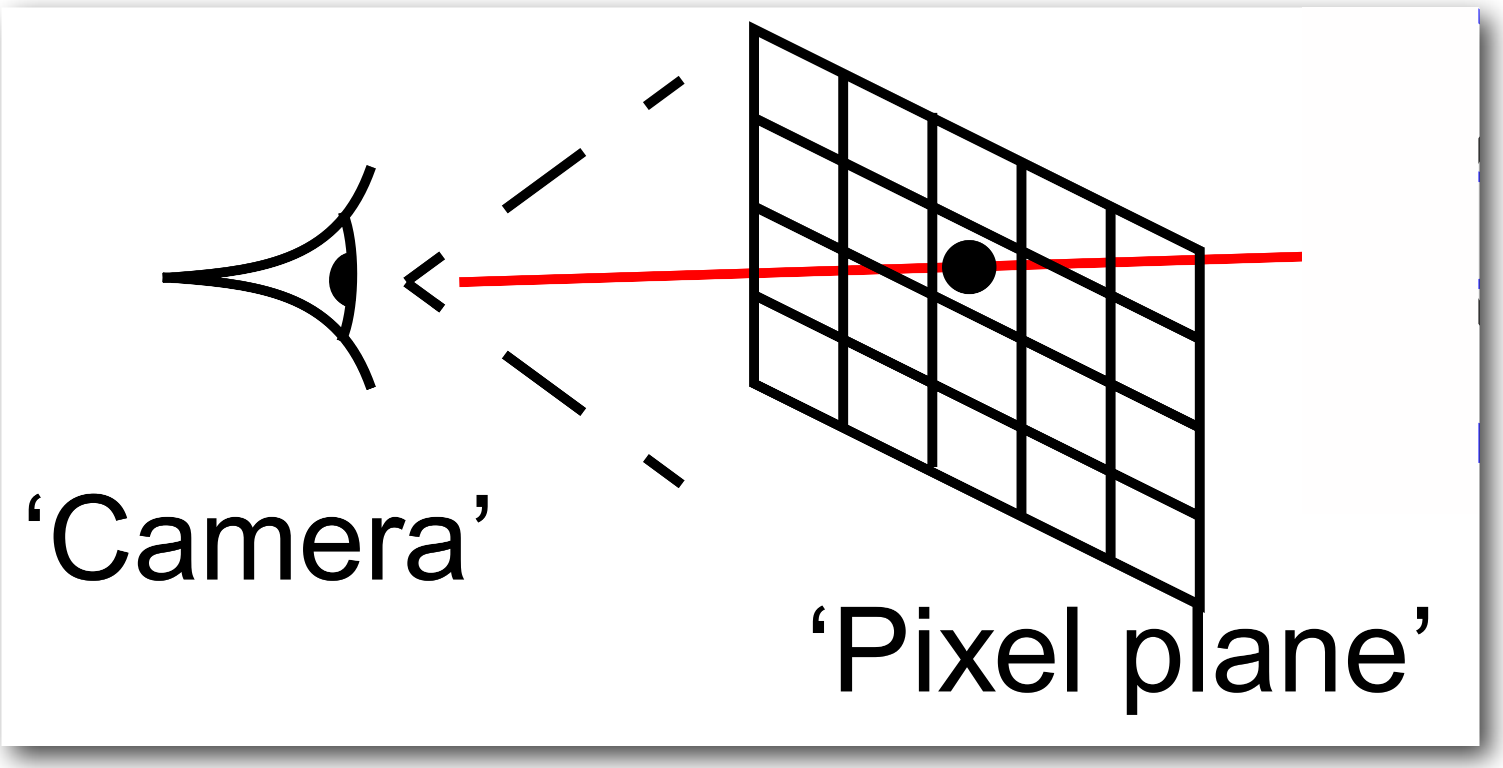
Deterministic - we know *a priori* where the light is coming from, so we send rays to the source.

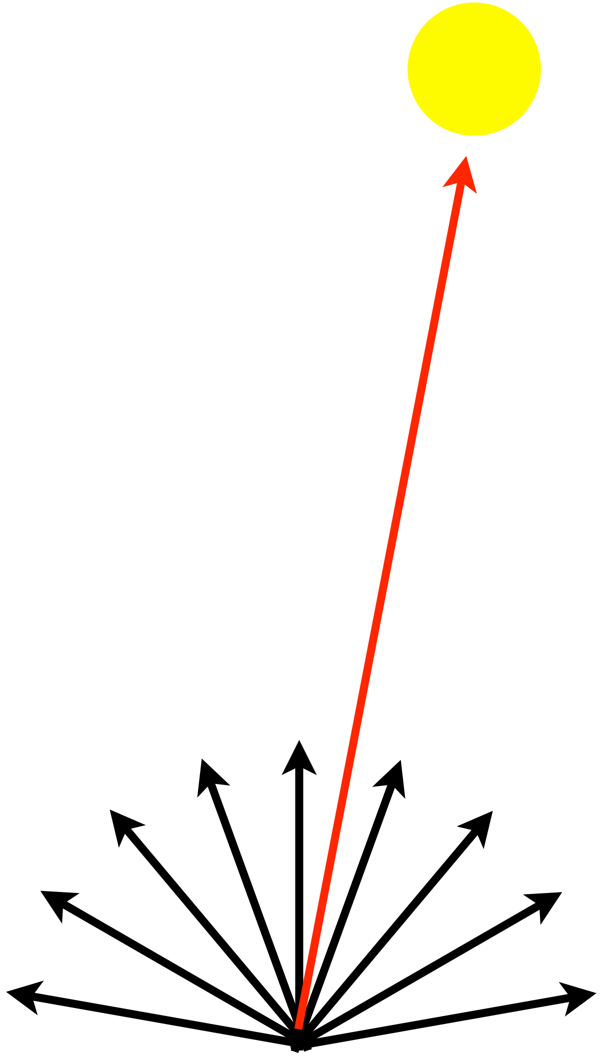
Hemispherical - we don't know in advance where the illumination is coming from, so we search (i.e. sample) every direction where it might come from.

How we define an emitting material in *Radiance* determines how it will be sampled:

- Material type **light** -> deterministic sampling
- Material type **glow** -> hemispherical sampling

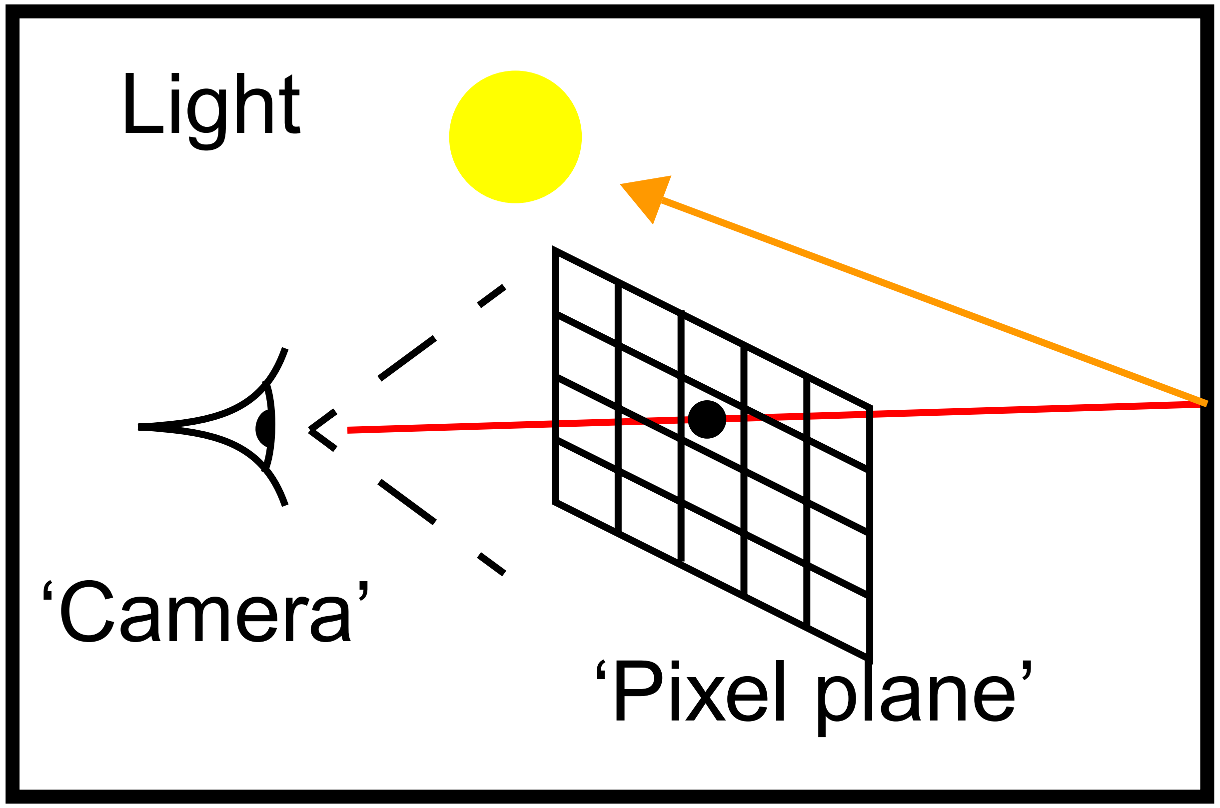
rpict





rtrace

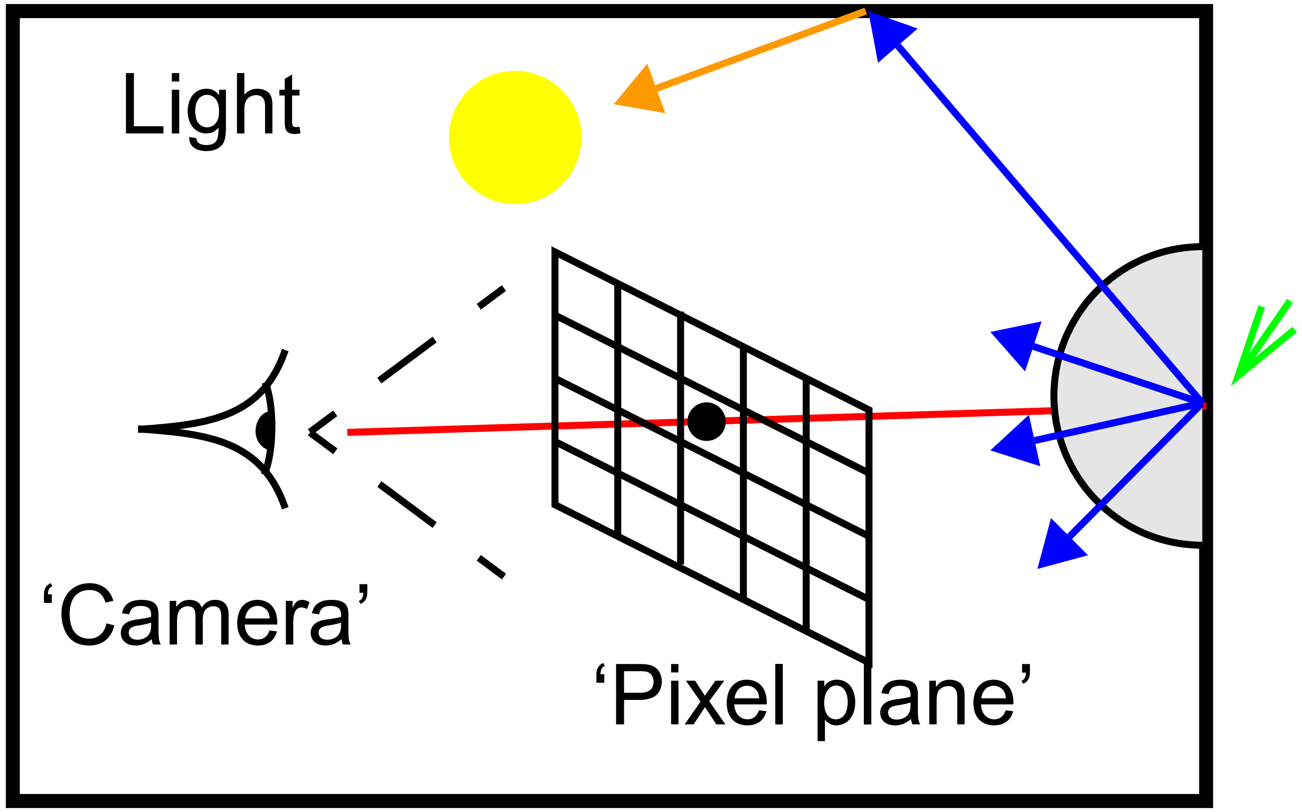
(a)



Direct

View ray intersects with scene here. A "shadow ray" is then sent to determine if this point of the scene (i.e. pixel) is illuminated by the light.

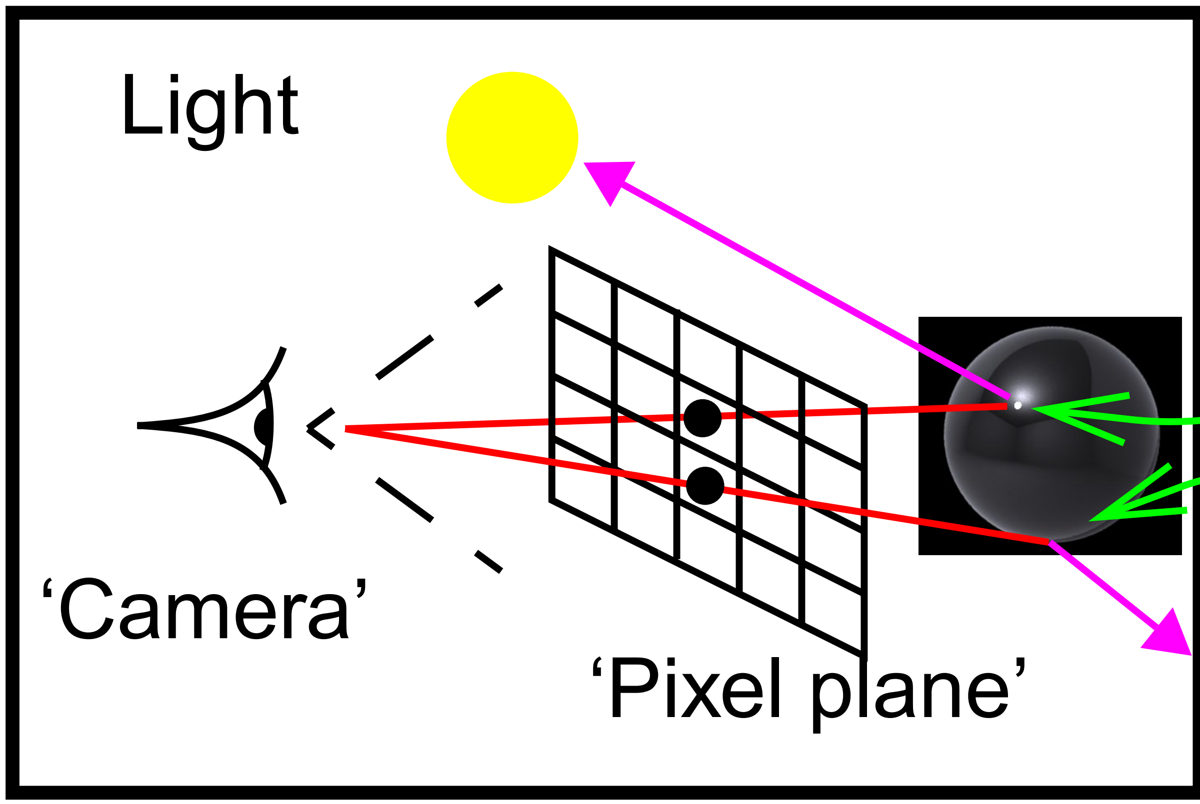
(b)



Indirect

Hemispherical sampling initiated here. Where a ray intersects with the scene, shadows rays may be sent out to determine if this point is illuminated by the light source.

(c)



Specular

Specular reflection to (direct) light source.

Specular reflection to illuminated room surfaces.

When to use **light** and **glow** sources

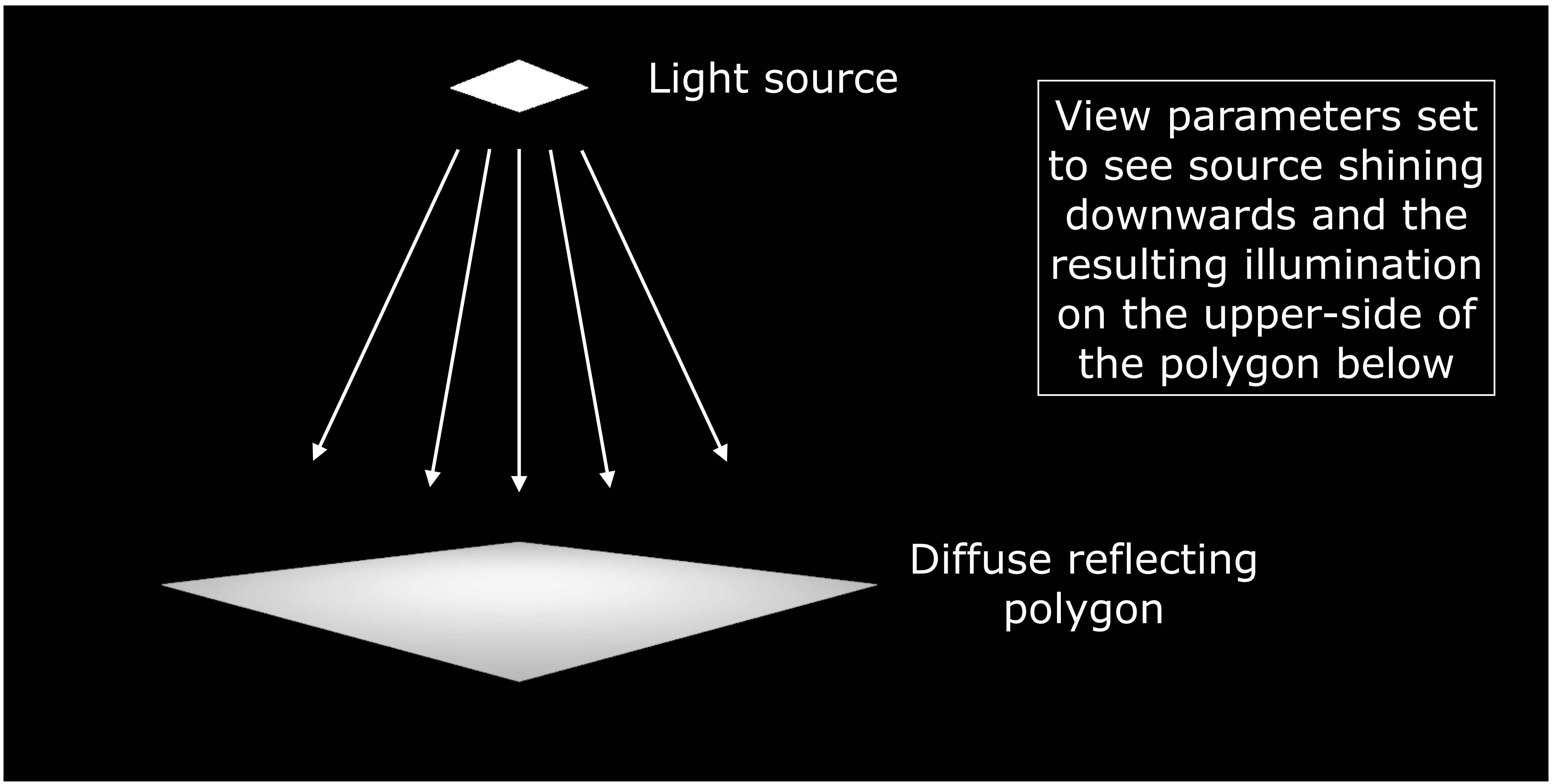
We use the material **light** for important sources of illumination, e.g. electric luminaire, the sun.

These participate in the direct calculation of illumination.

The material **glow** is used to describe extended sources of illumination (sky or ‘glowing’ ground) and also unimportant sources that may be visible to the ‘camera’ but do not contribute significantly to scene illumination. These participate in the indirect calculation of illumination.

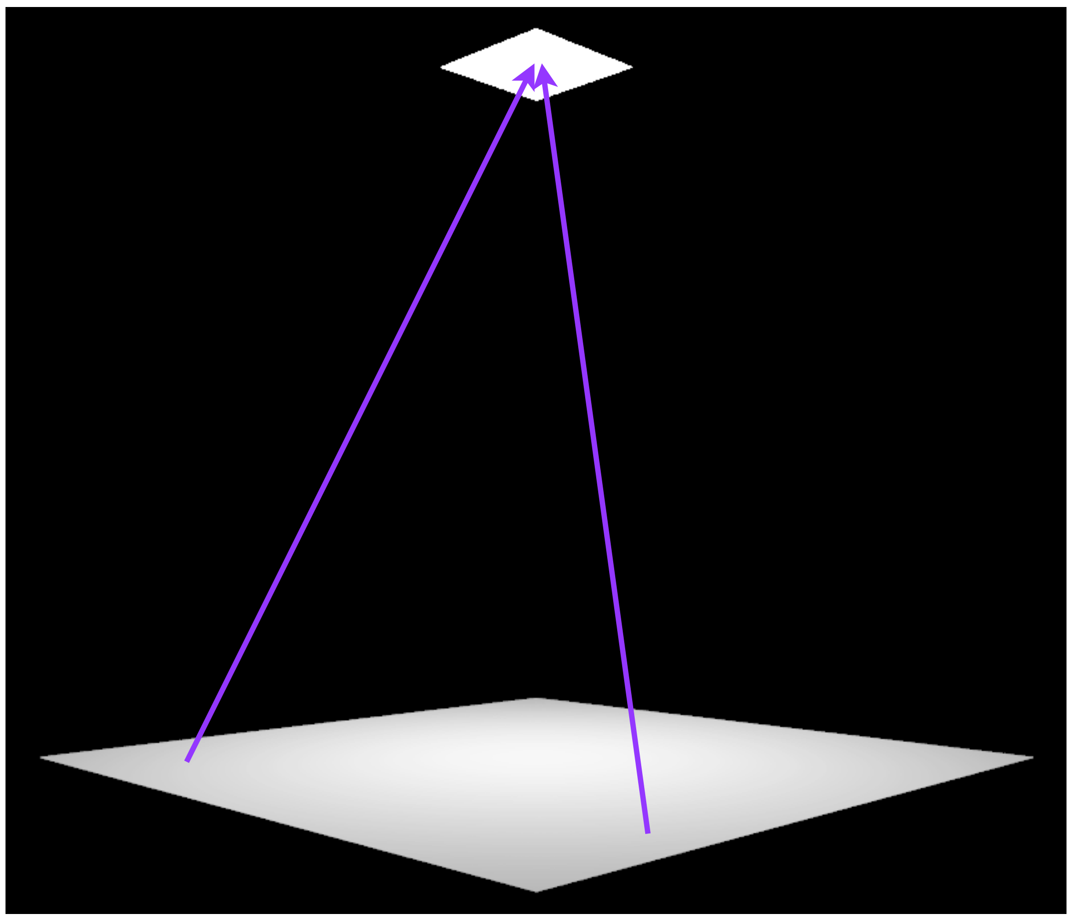
Example scene: two polygons

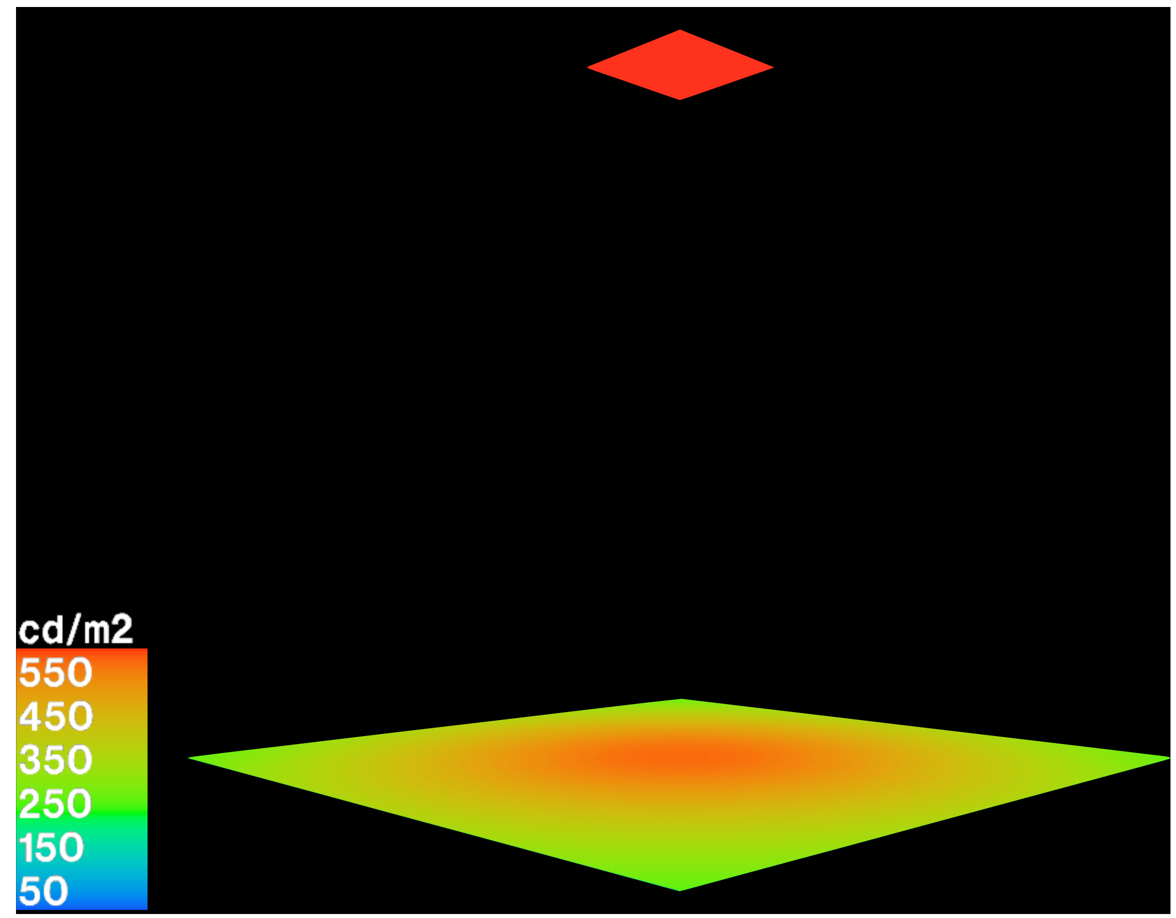
The test scene comprises two polygons - one is an emitter of light which shines onto the other



Define the emitting material as **light**

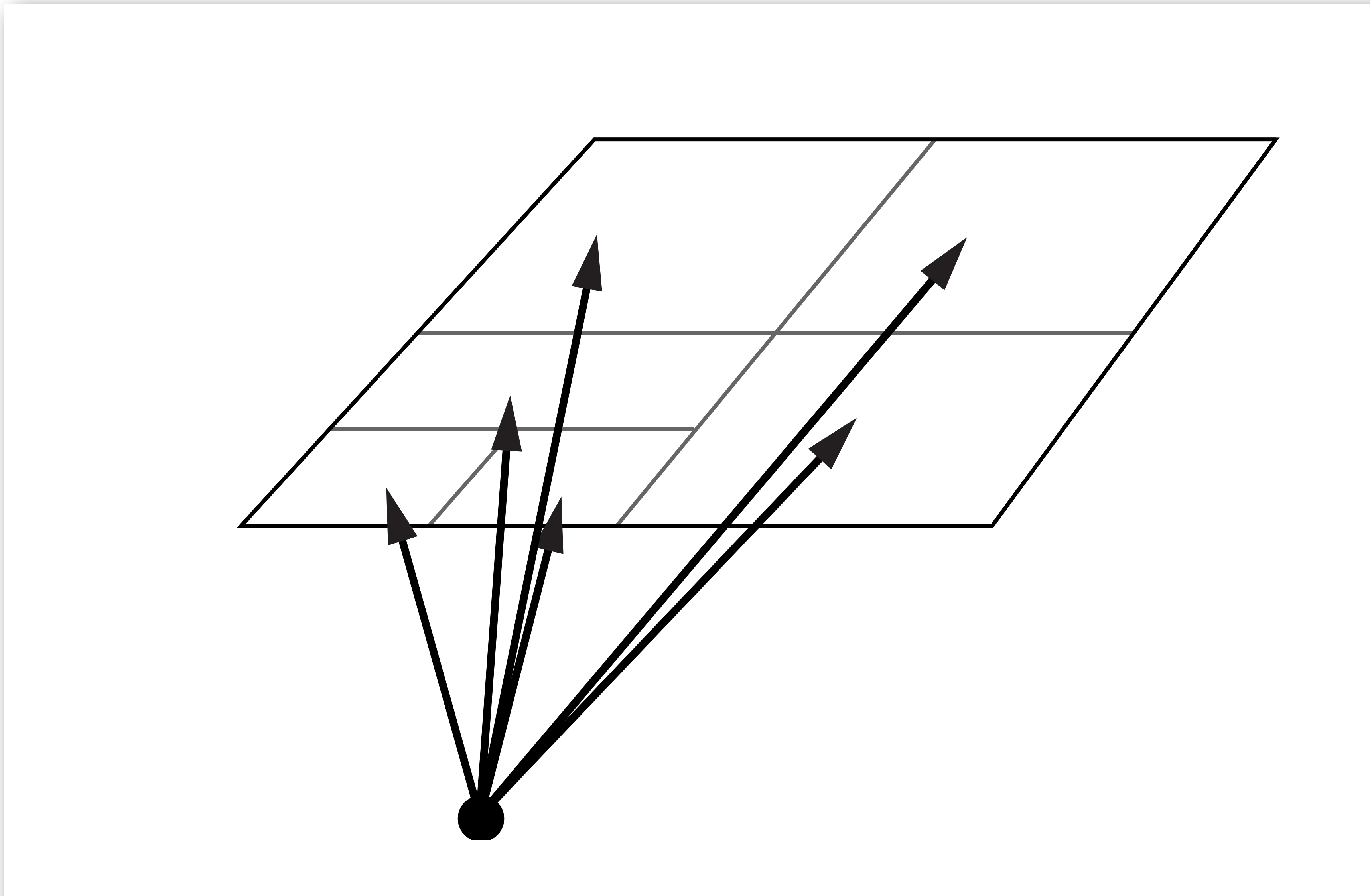
A shadow ray is sent from the reflection polygon to the source at every point in the pixel plane where the reflection polygon is visible.





The reflecting polygon is evenly illuminated by the light source.
This is clearly revealed in the false colour image.
Note: **-ab 0** setting used, i.e. inter-reflection calculation turned off.

Adaptive source subdivision

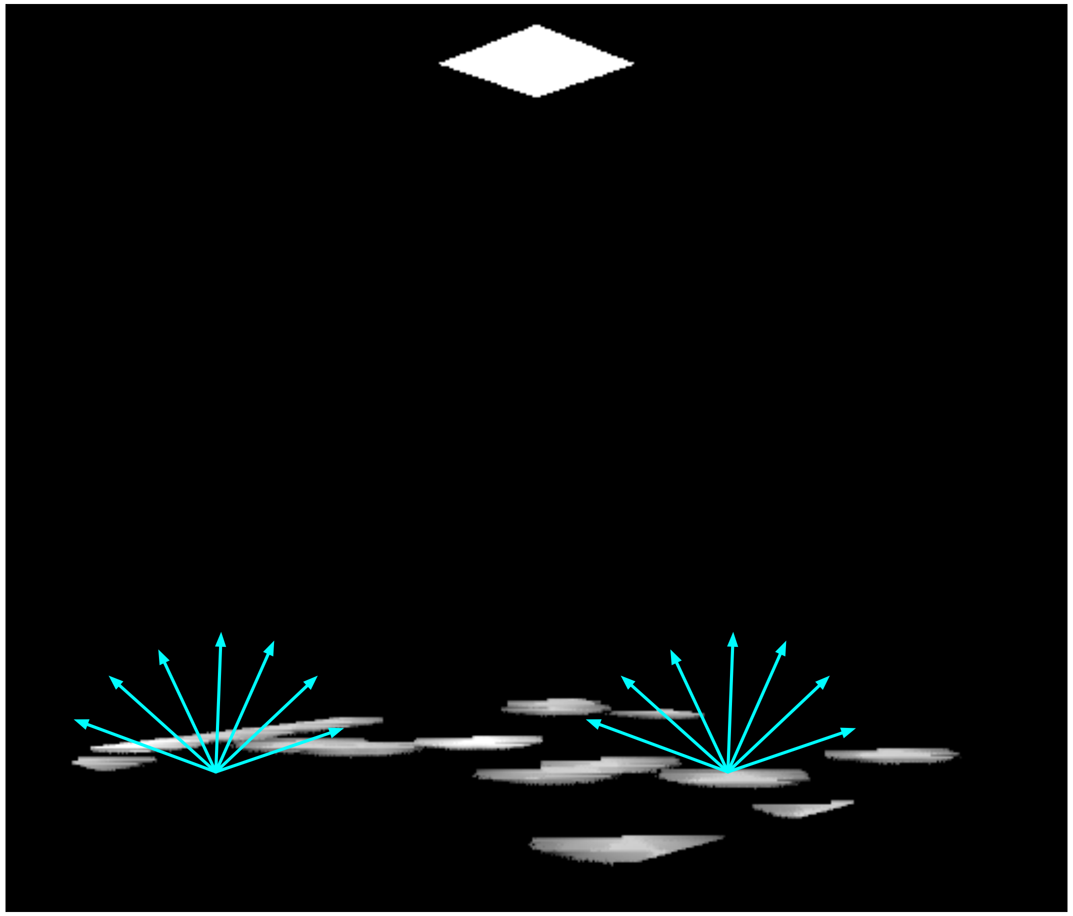


A light source will be subdivided until the width of each sample area divided by the distance to the illuminated point is below the ratio **ds** [default value = 0.2].

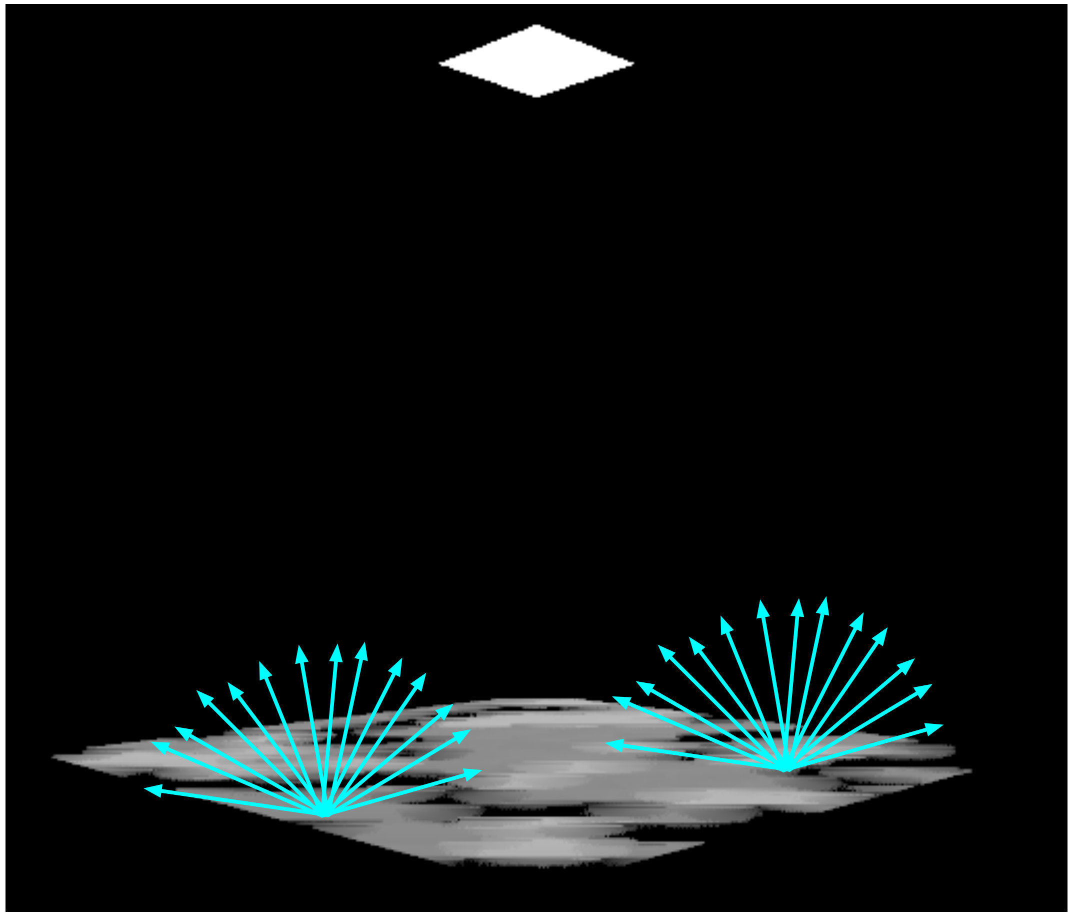
Fig 11.7 Rendering with Radiance

Define the emitting material as **glow**

Now we have to switch on the inter-reflection to hunt for the light source, i.e. set **-ab 1**. We'll hunt for the source using different numbers of hemispherical sampling rays (the **ad** parameter) to see the effect.



-ad 32



-ad 64

The sampling pattern is not evenly distributed across the hemisphere

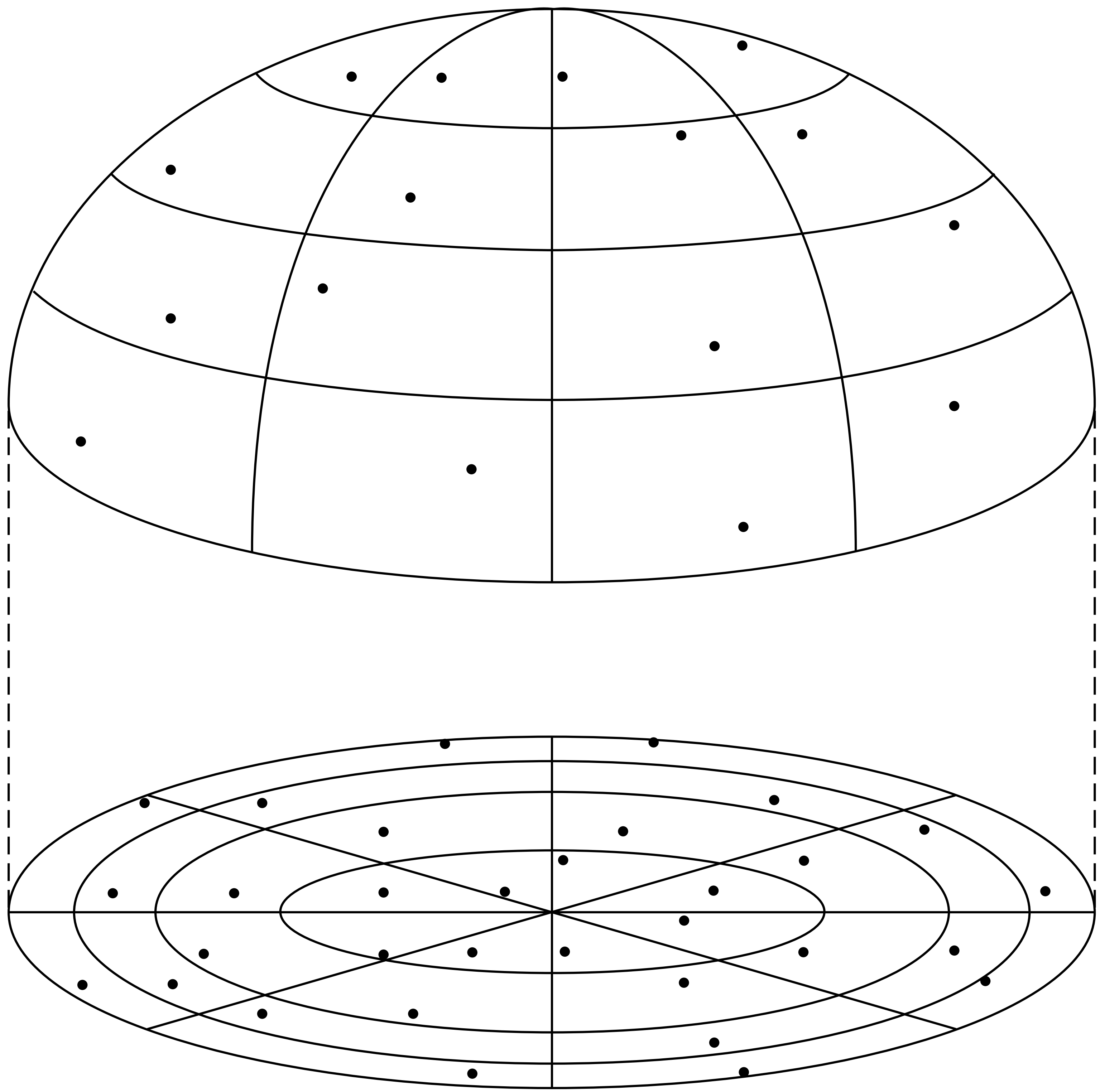
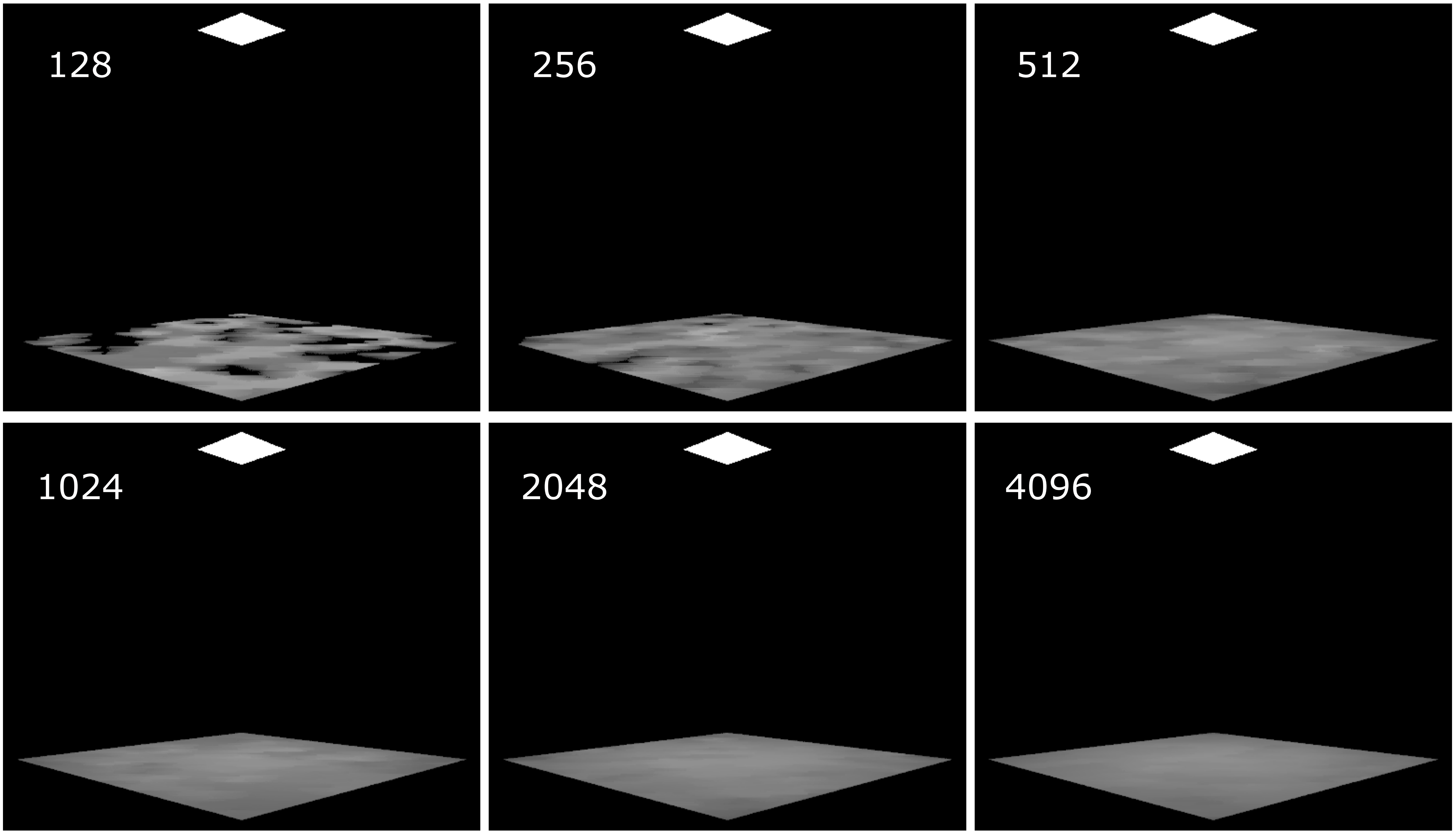
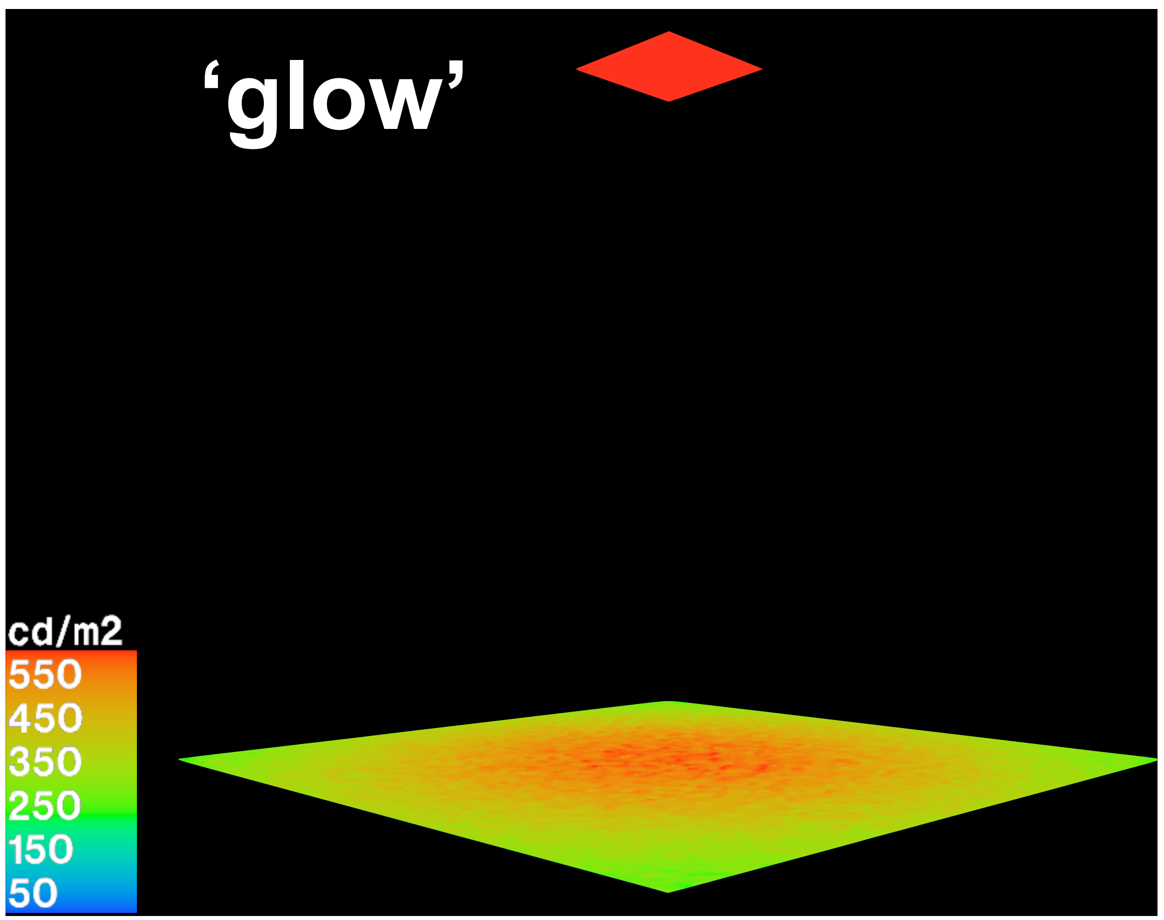


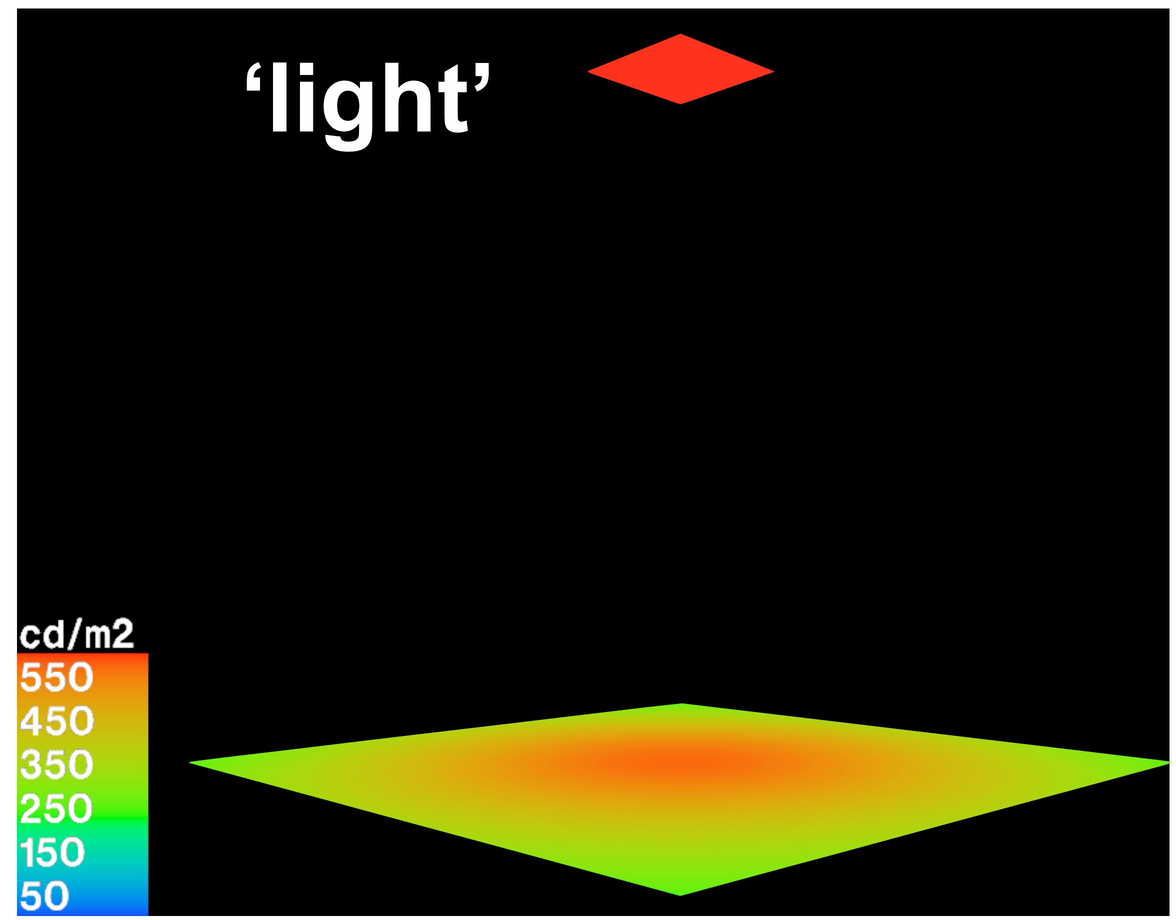
Fig 12.7 Rendering with Radiance

Increasing the number of **ad** rays does produce smoother shading (at greater computational cost)



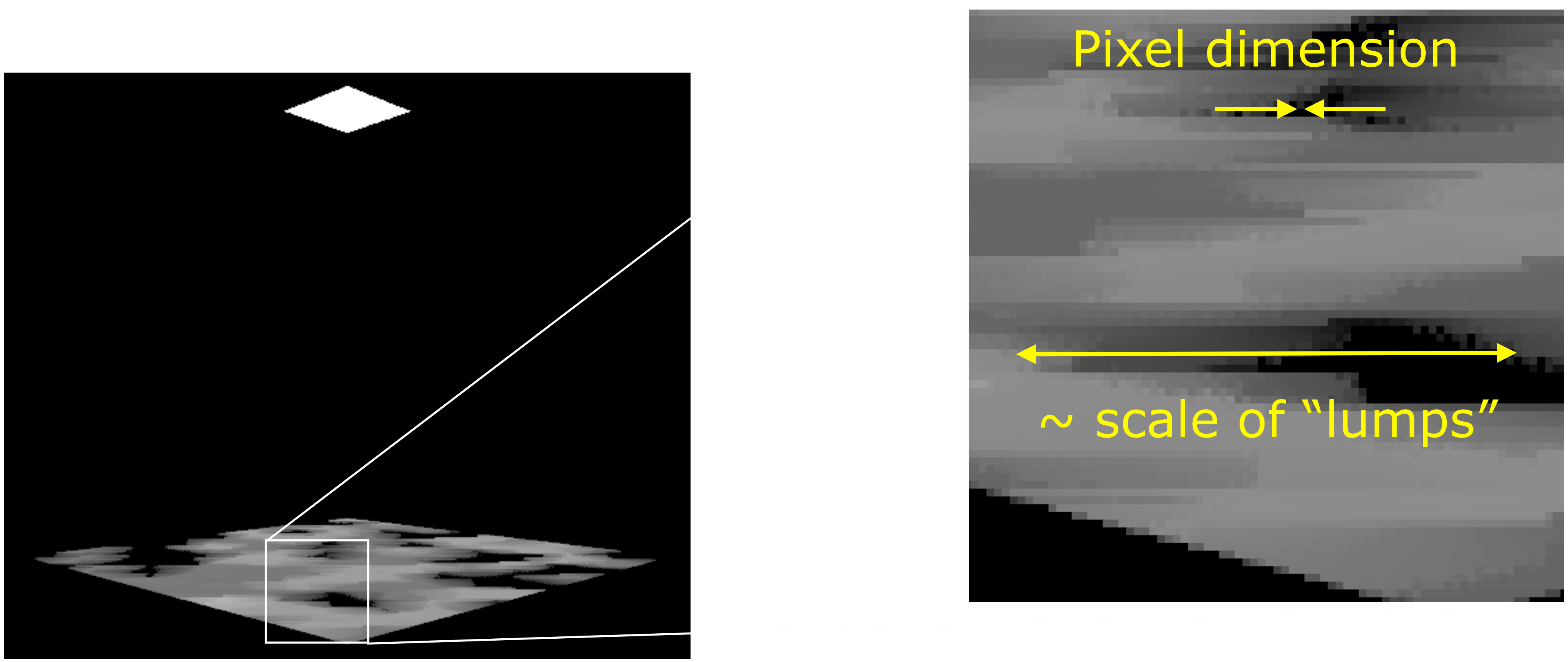
But even with **-ad 4096** the illumination from the **glow** material is not quite as smooth as with that from the **light** material.





Why are the **glow** renderings lumpy?

With a small **glow** source, sometimes the hemispherical sampling finds (i.e. “hits”) the source, and sometimes it doesn’t. Note also that there is a random (or stochastic) component to the ray direction.



Notice that the lumpiness occurs at scales <u>much larger</u> than the effective dimension of a pixel - what does that suggest about hemispherical sampling compared to deterministic?

What's the significance of the big lumps?

These suggests to us that hemispherical sampling is <u>not</u> happening for every pixel.

If it was, then the "sometimes you find the source sometimes you don't" effect would be happening from one pixel to the next - resulting in lumpiness at the pixel scale.

Usually in *Radiance*, hemispherical sampling is set to happen at points **every now and then** across a scene, and not at every pixel. *Radiance* then interpolates (i.e. estimates) values between these points.

Why use interpolation?

Simply, to be efficient. Consider, for the images used previously, the reflecting polygon comprised ~25,000 pixels. In the deterministic calculation (**light**), a shadow ray was sent to the source for each of the 25,000 pixels where a view ray intersected with the reflecting polygon.

If hemispherical sampling occurred at each of these pixels, then the number of rays sent would be 25,000 times the **ad** number:

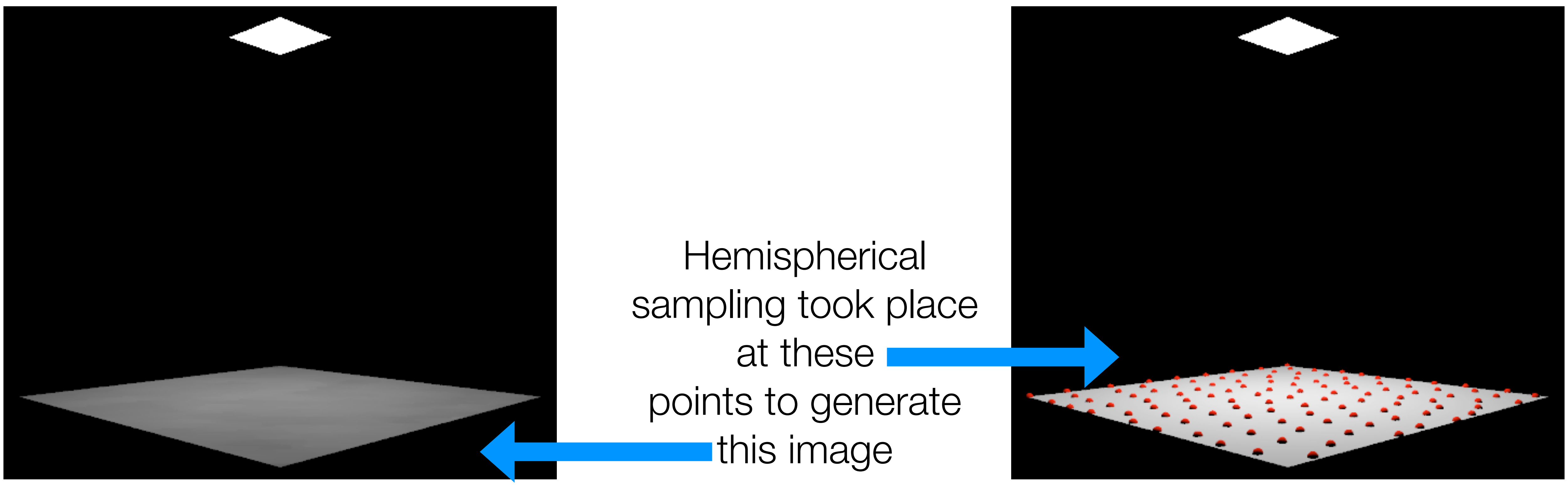
25,000 x 128 = 3,200,000 rays; or,

25,000 x 4096 = 102,400,000 rays.

Even for -ad 128 many times more hemispherical sampling rays are sent out than for the deterministic calculation, but most of those will “miss” the small source.

Where interpolation took place

The **genambpos** utility was used to place markers (red spheres •) in the scene where interpolation took place.

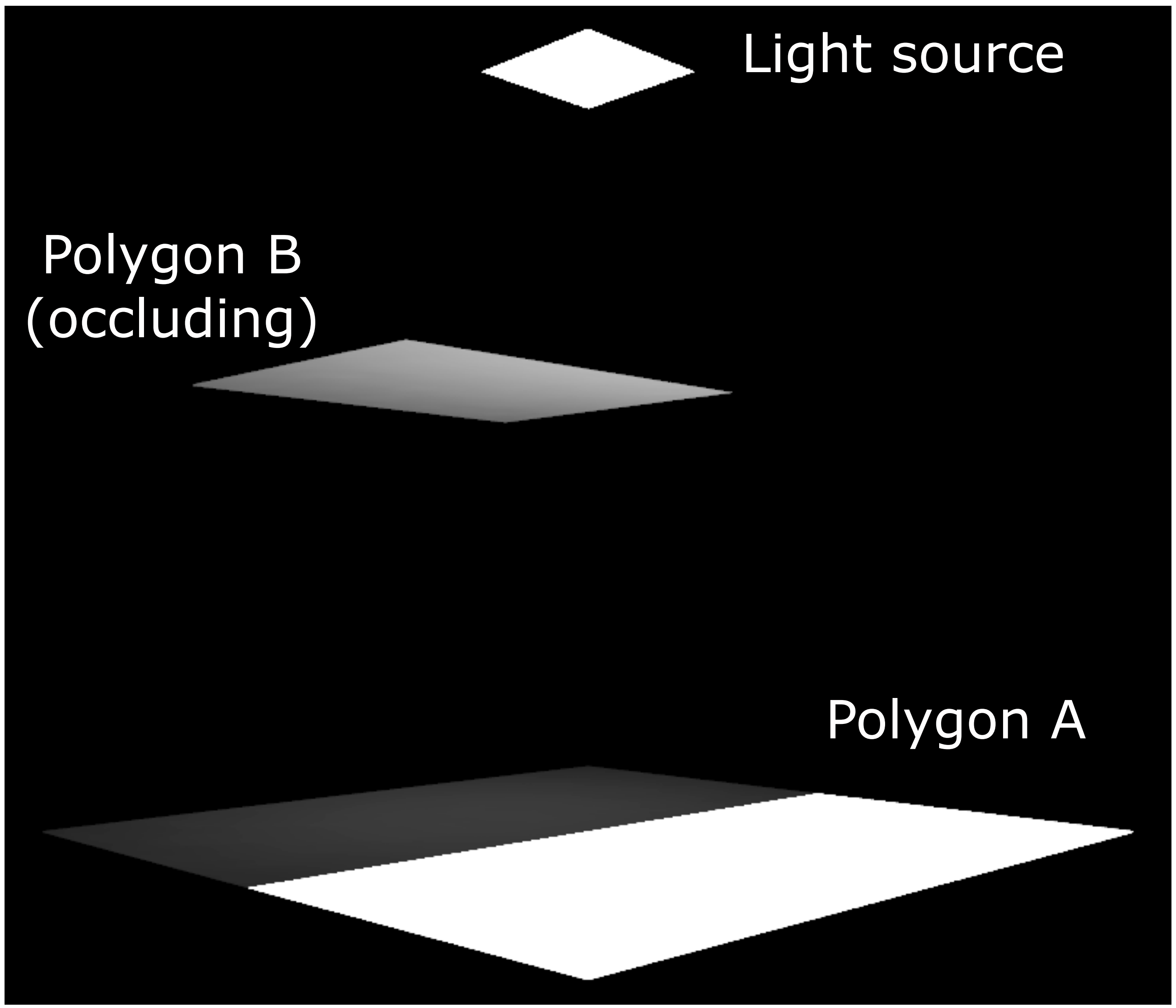


Recap

For small, important sources of illumination, we describe the emitter using the material **light** so that it is sampled using the direct (deterministic) calculation.

In the previous example, the scene didn’t allow for inter-reflection. Here, we modify the scene by adding an occluding polygon to see how hemispherical sampling is used to compute indirect or (inter-reflected) light.

Scene with partially occluding polygon



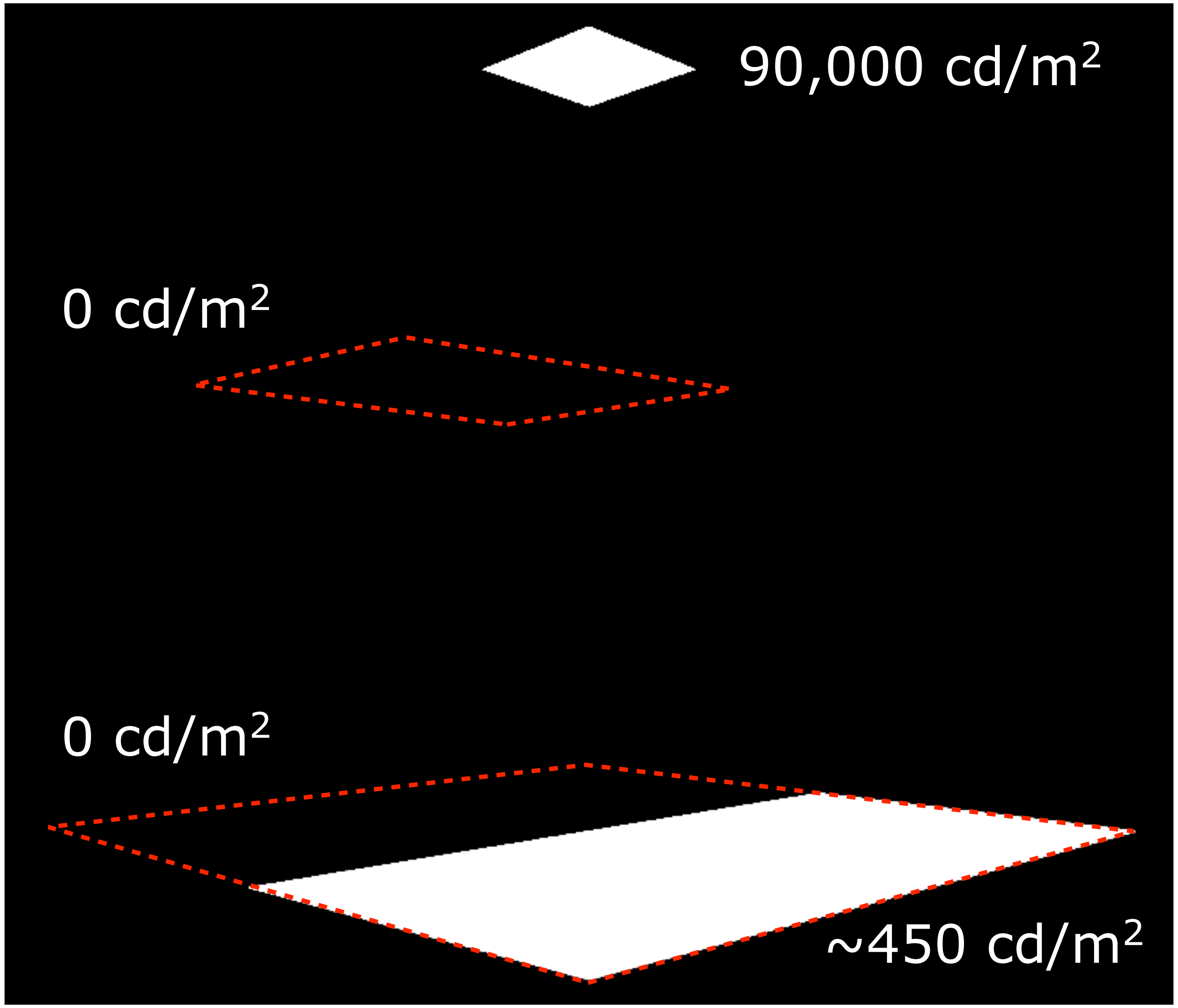
Polygon B positioned to partially shade Polygon A from the light source (material **light**)

View shows the underside of Polygon B and the topside of Polygon A

Rendering for occluding scene **-ab 0**

Underside of polygon B not illuminated

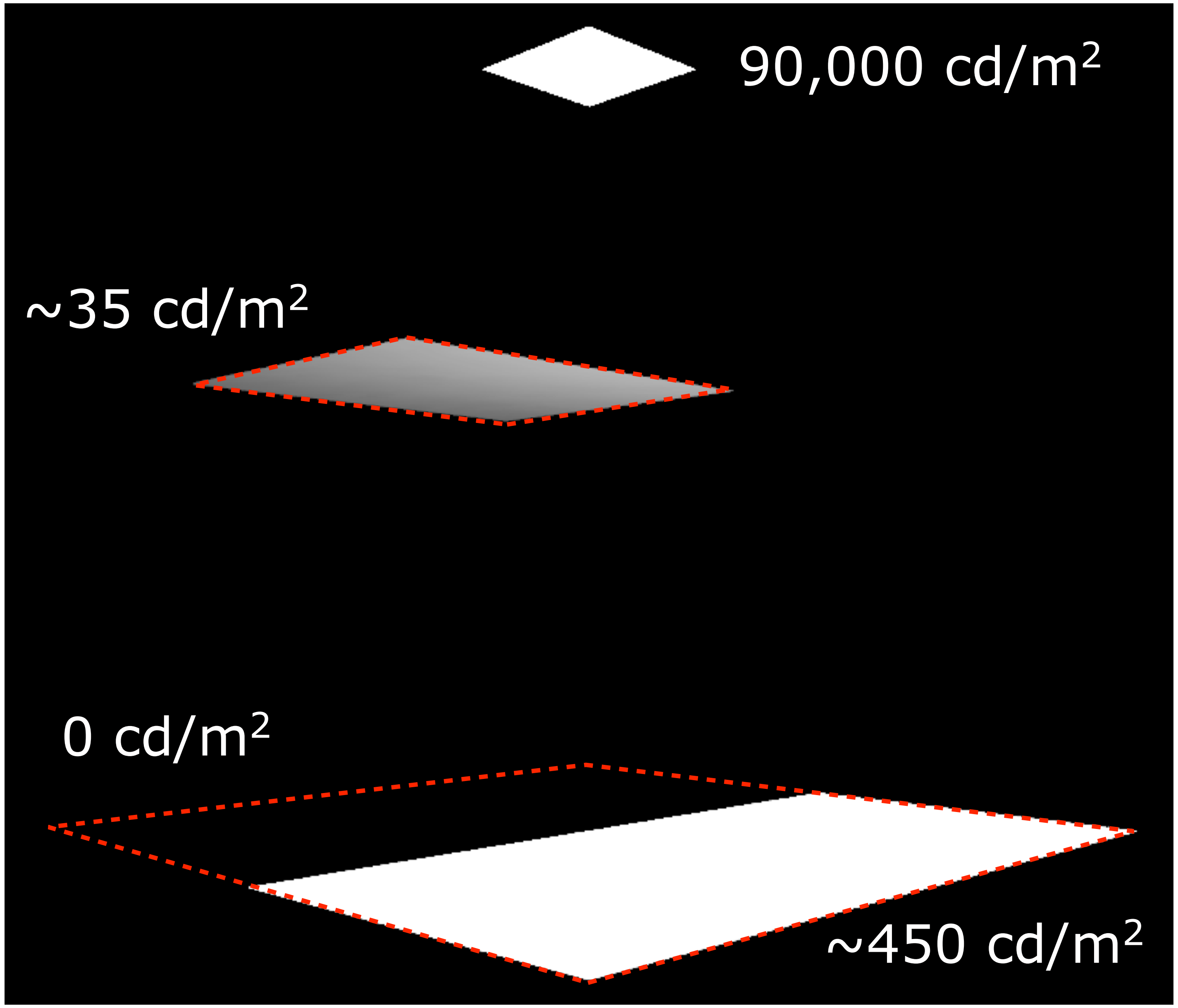
Topside of polygon A half in shade



Rendering for occluding scene **-ab 1**

Underside of polygon B now illuminated

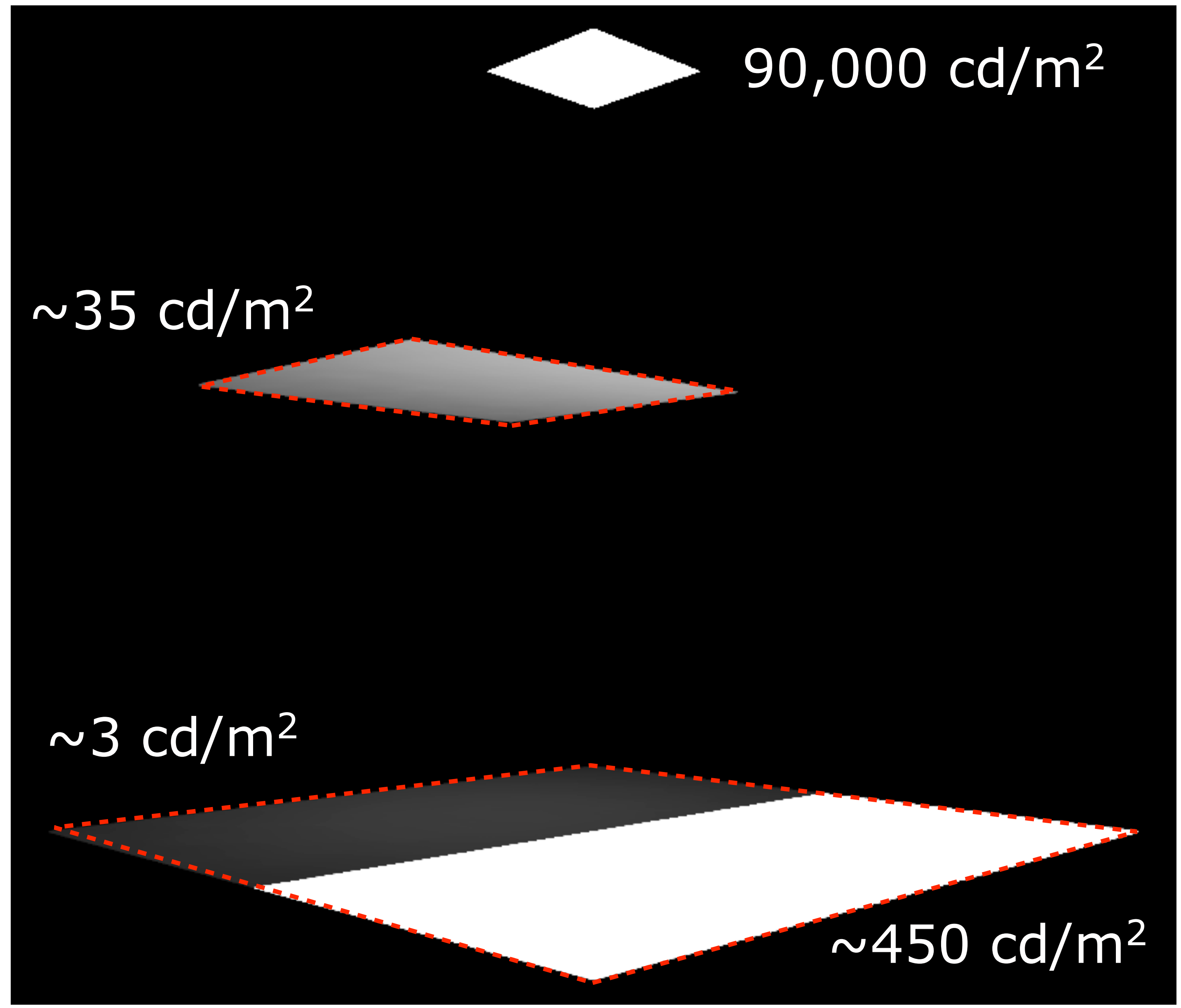
Topside of polygon A still half in shade



Rendering for occluding scene **-ab 2**

Underside of polygon B illuminated

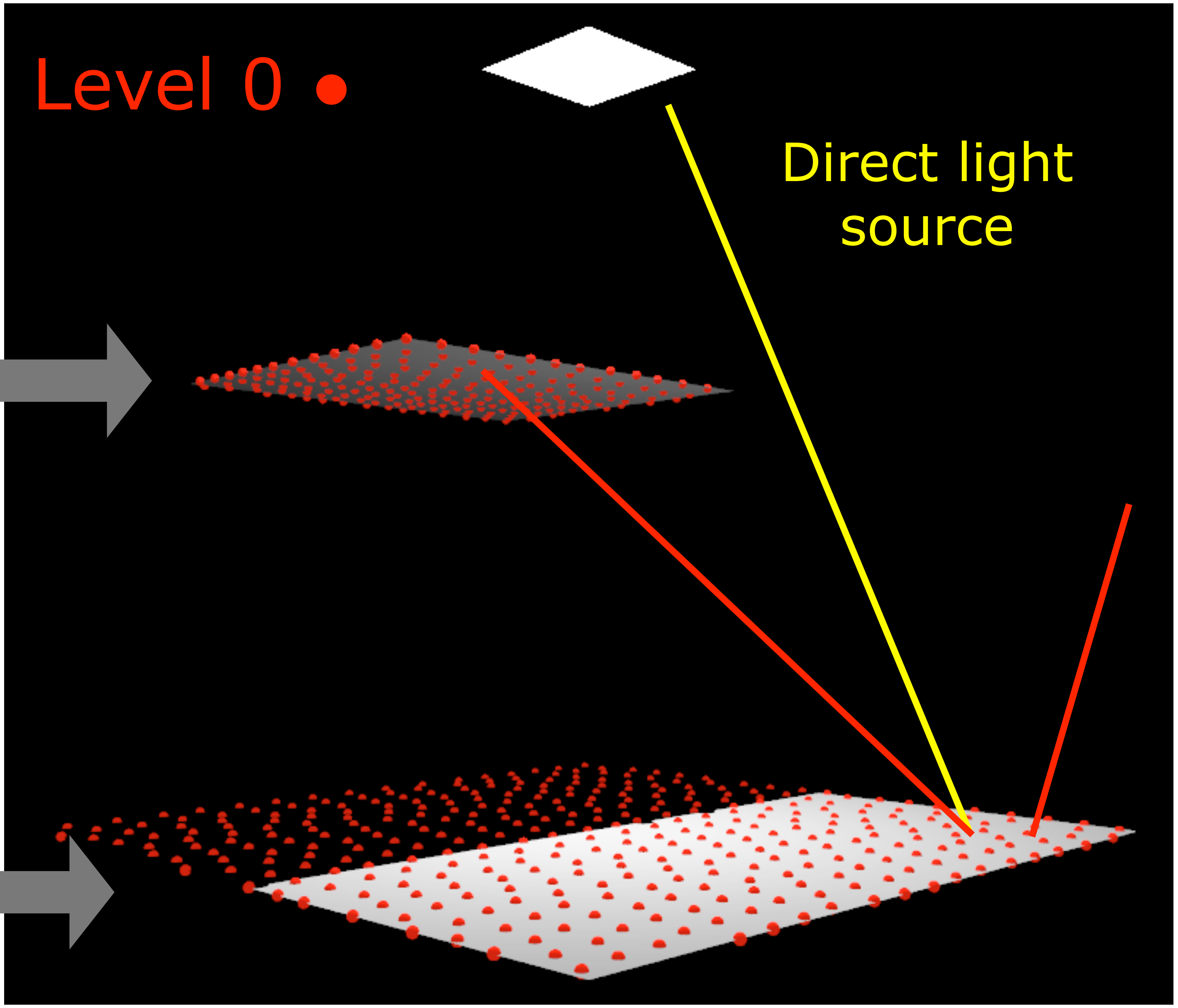
Shaded half of polygon A now illuminated by reflected light from polygon B



Hemispherical sampling (HS) took place at these locations for **-ab 1**

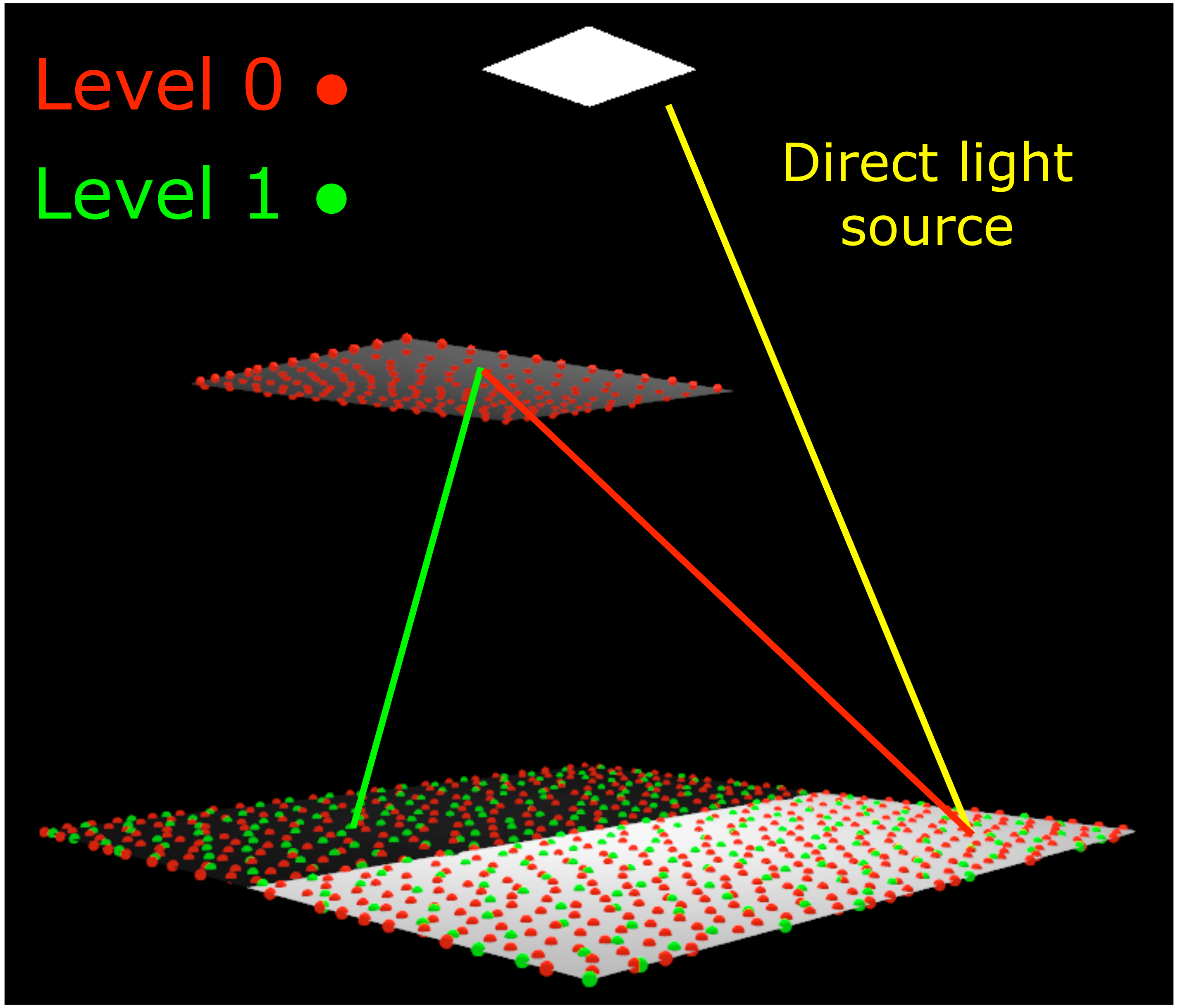
HS from here found the illuminated half of the lower polygon

But HS from here did not find any illuminated surfaces (the light source is excluded from the indirect calculation)



Hemispherical sampling took place at these locations for **-ab 2**

Level 1 HS from the lower polygon can now find the reflected light from the (underside) of the upper polygon

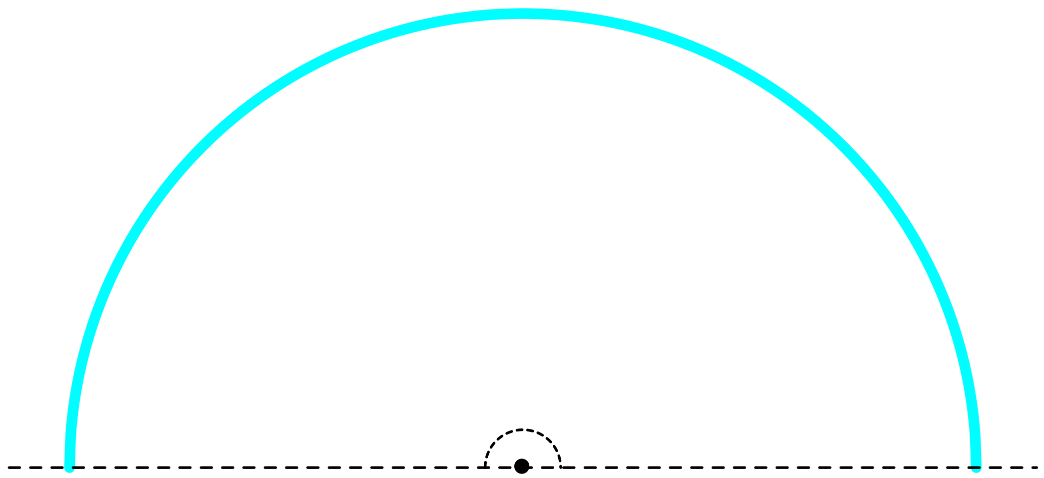


Questions?

Some quantitative examples

- Predict the illuminance under a simple sky (without sun).
- First a uniform (i.e. constant brightness sky).
- Then a CIE standard overcast sky.

```
# sky_uni.rad
# uniform brightness sky (B=1)

void glow sky_glow
0
0
4 1 1 1 0

sky_glow source sky
0
0
4 0 0 1 180
```

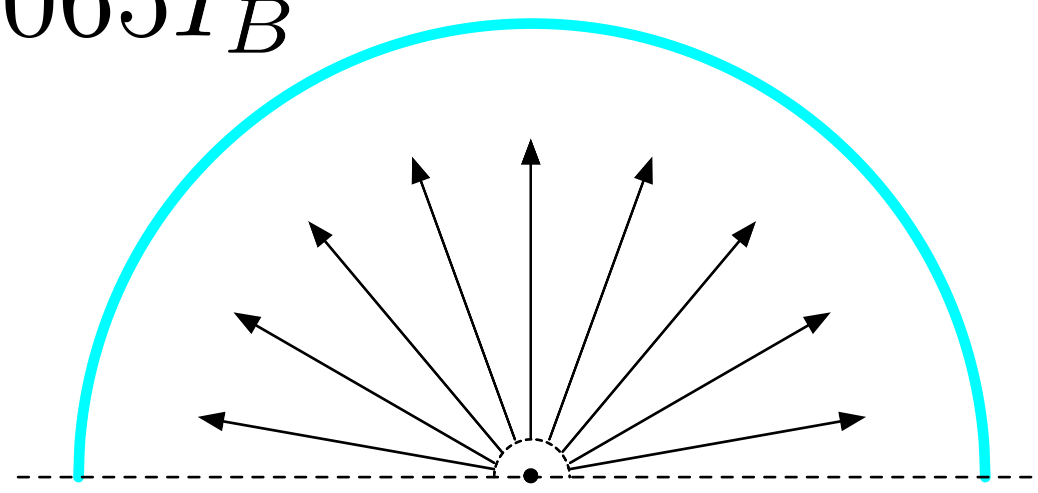


```
% oconv sky_uni.rad > sky_uni.oct
```

```
% echo "0 0 0 0 0 1" \
| rtrace -h -I+ -w -ab 1 sky_uni.oct
```

```
3.141593e+00  3.141593e+00  3.141593e+00
```

$$I = 0.265 I_R + 0.670 I_G + 0.065 I_B$$



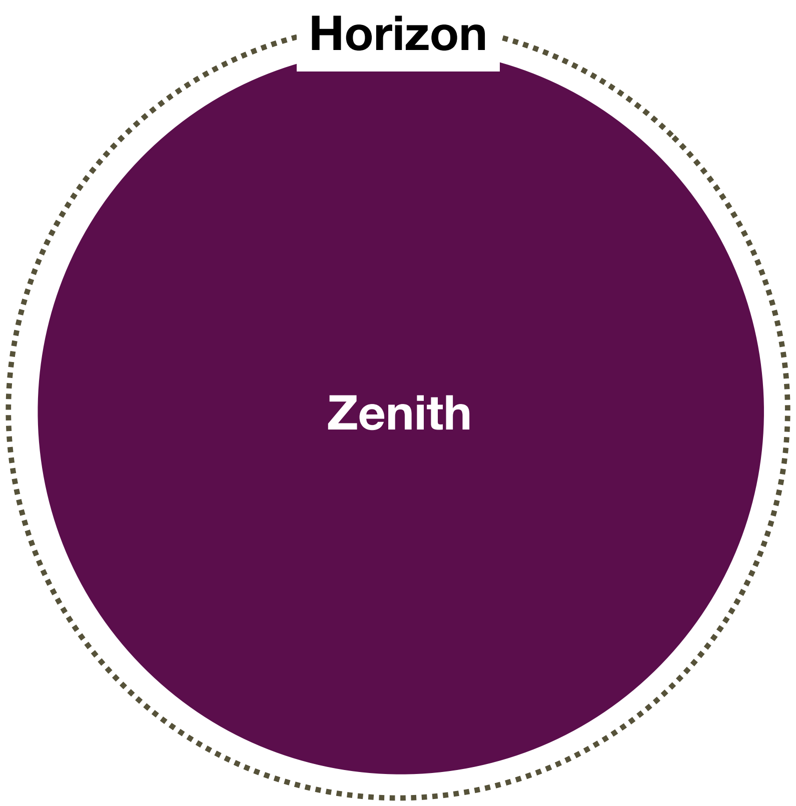
$$I = \int_0^{2\pi} \int_0^{\pi/2} B\left(\theta, \phi\right)\, \sin\theta\, \cos\theta\, d\theta\, d\phi$$

$$I = B \int_0^{2\pi} \int_0^{\pi/2} \sin\theta\, \cos\theta\, d\theta\, d\phi$$

$$I = \pi B$$

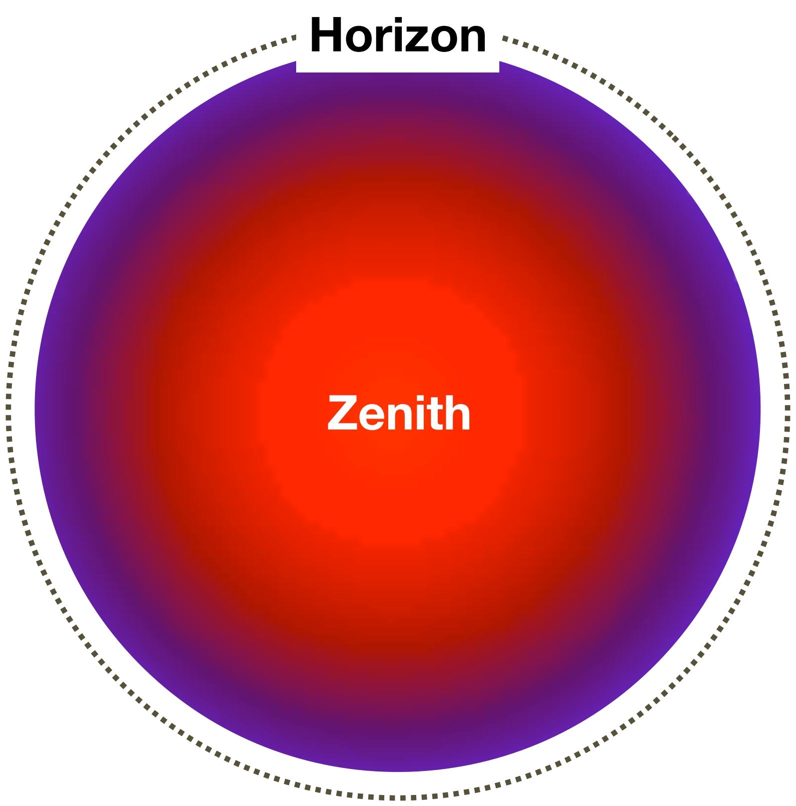
$$I = 3.1415926$$

Uniform sky



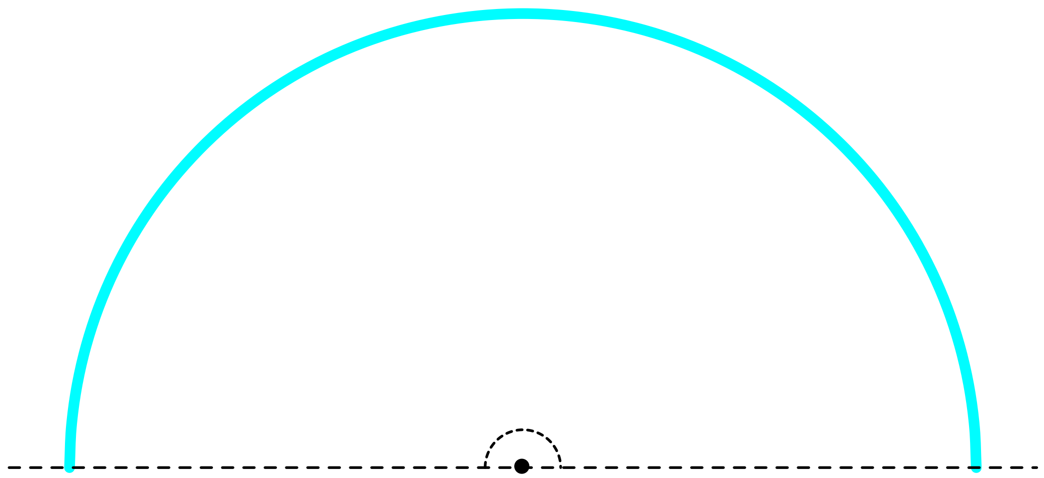
$$B_\zeta = B_z$$

CIE standard overcast sky



$$B_\zeta = \frac{B_z\,(1 + 2\cos\zeta)}{3}$$

```
# sky_ovc.rad
# CIE overcast sky (Bz = 1)
!gensky -ang 45 0 -c -b 1
skyfunc glow sky_glow
0
0
4 1 1 1 0

sky_glow source sky
0
0
4 0 0 1 180
```



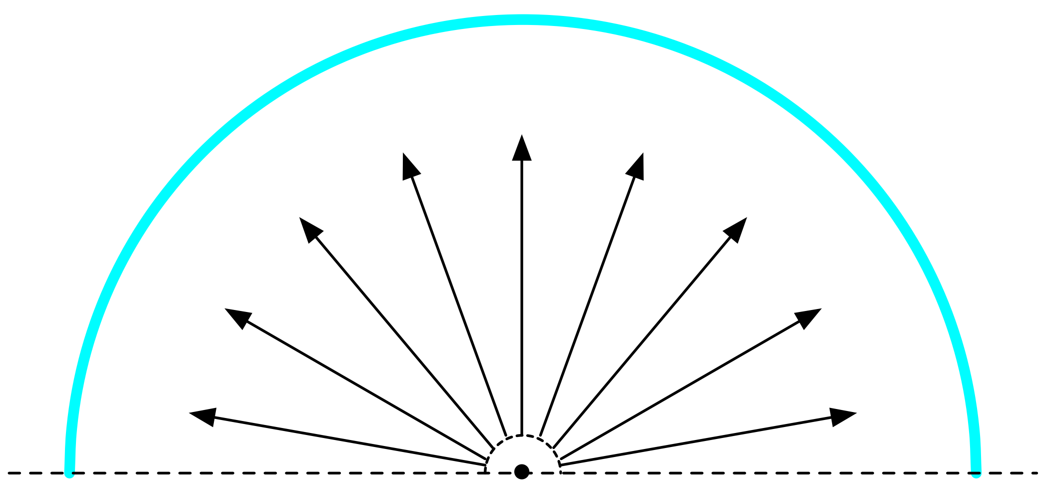
```
% oconv sky_ovc.rad > sky_ovc.oct
```

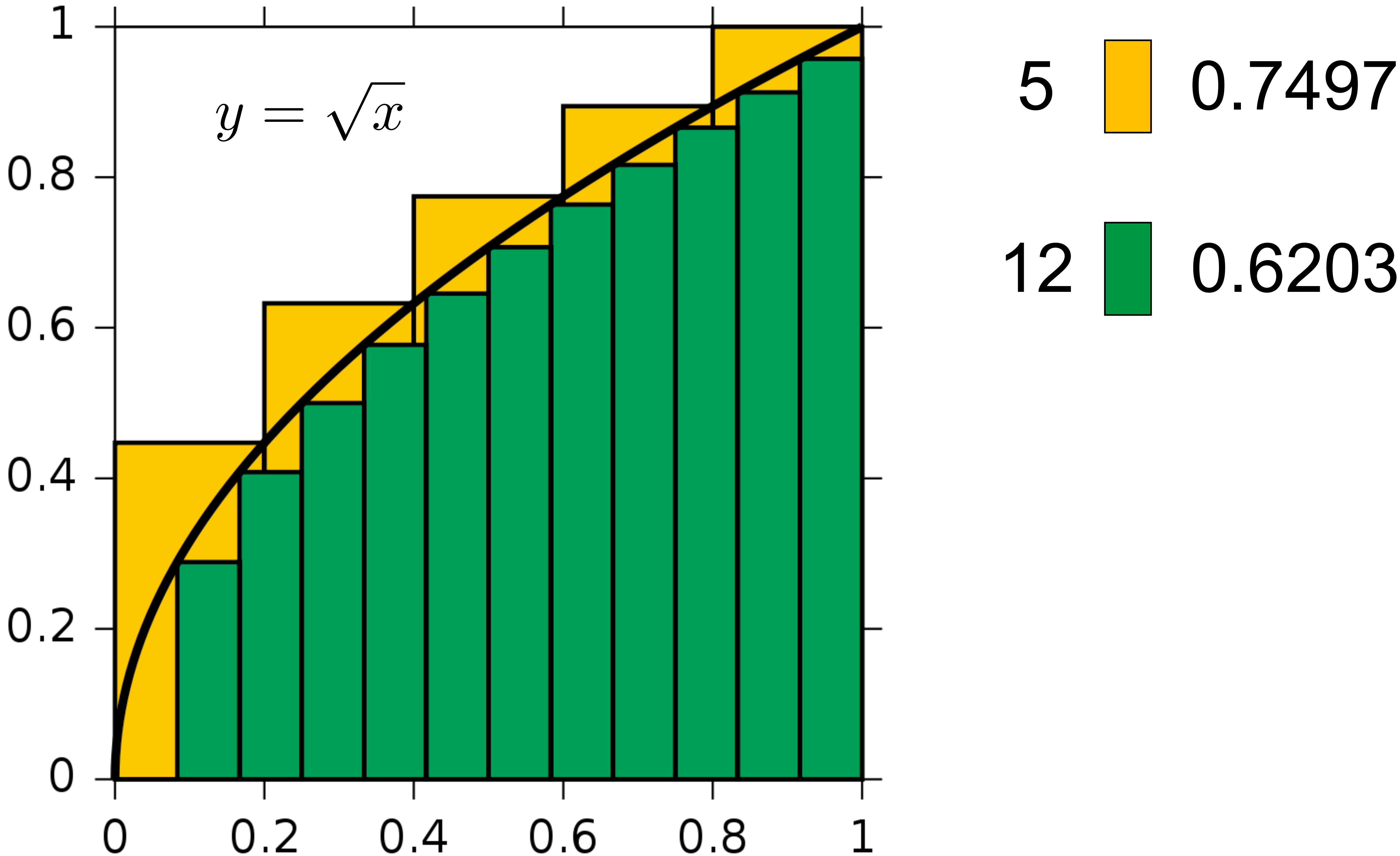
```
% rtrace -w -h -I+ -ab 1 \
   sky_ovc.oct < samp.inp \
 | rcalc -e '$1=$1*0.265+$2*0.670+$3*0.065'
```

2.434001 [default ad]

$$7\pi B_z/9 = 2.443451$$

2.443563 [higher ad]





$$area = \int_0^1 \sqrt{x}\, dx = 0.6667$$

Typical values commonly used to define the CIE overcast sky

- The CIE overcast sky is defined by its horizontal illuminance, usually given in lux.
- A convenient horizontal illuminance for a (brightish) overcast sky is 10,000lux, e.g. 500 lux corresponds to a 5% DF.
- In **gensky** we can specify either the zenith radiance (-b option) or the horizontal (diffuse) irradiance (-B option). The second option is perhaps the more direct, and we shall use that for the next **rtrace** example.

The irradiance that corresponds to this illuminance is 10,000/179 = 55.866 W/m^2.

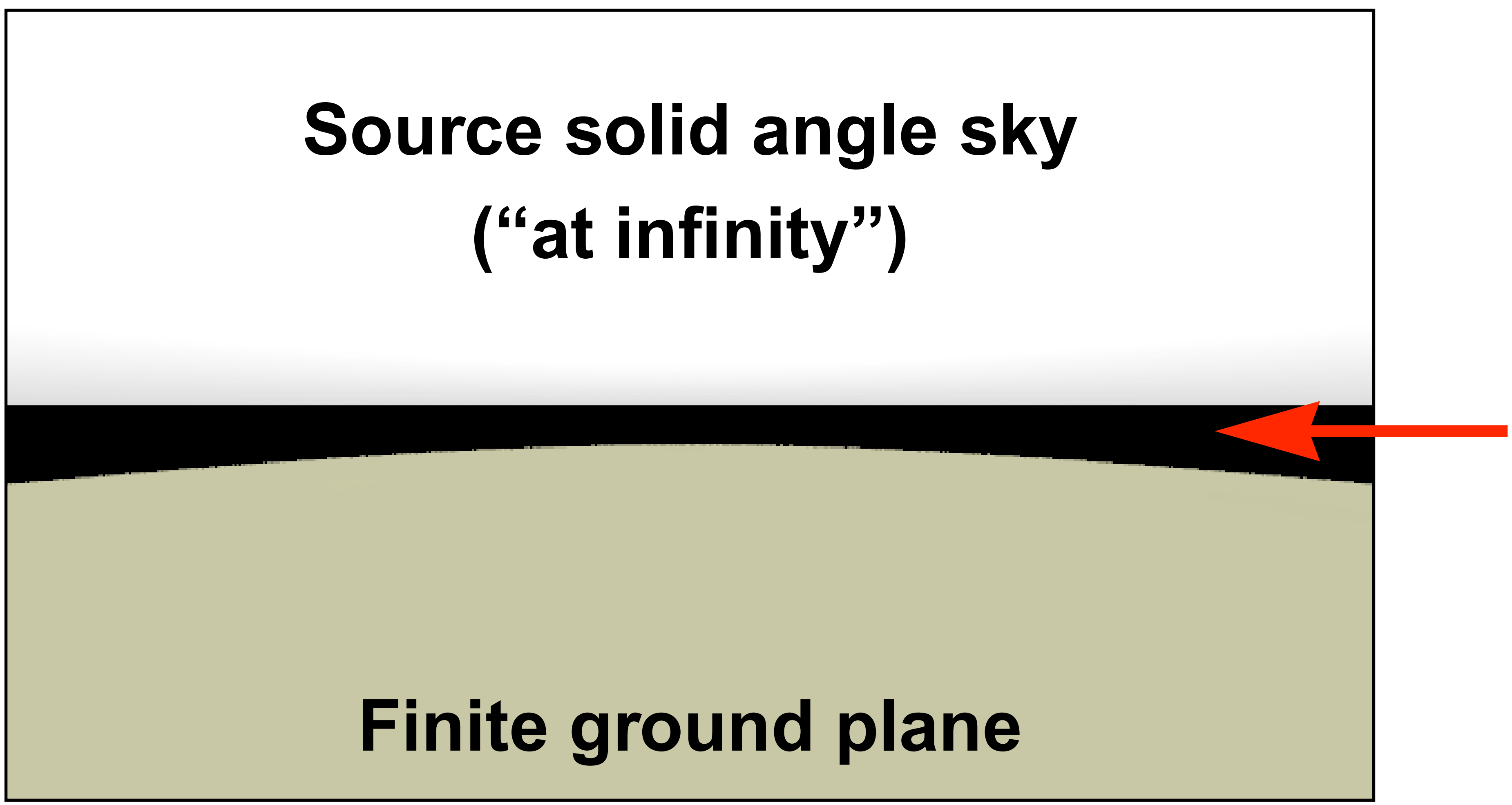
This conversion factor is the *Radiance* system's own internal value for luminous efficacy and is fixed at k_R = 179 lumens/watt (lm/W).

```
!gensky -ang 45 0 -c -B 55.866
```

```
rtrace -w -h -I+ -ab 1 \
sky_ovc.oct < samp.inp | rcalc -e \
 '$1=($1*0.265+$2*0.670+$3*0.065)*179'
```

```
9977.17002          [near enough to 10,000 lux]
```

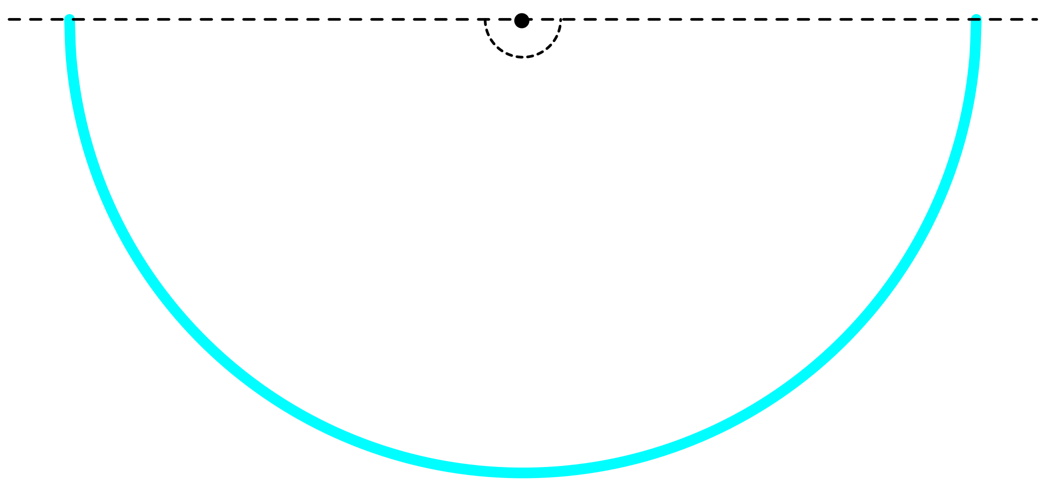
This is what we can see if we add a ground plane



Gap between "infinite" sky and finite ground

Ground glow - an upside down sky

```
skyfunc glow ground_glow
0
0
4 1 1 1 0

ground_glow source ground
0
0
4 0 0 -1 180
```



This creates a ‘seamless luminous envelope’ around our scene

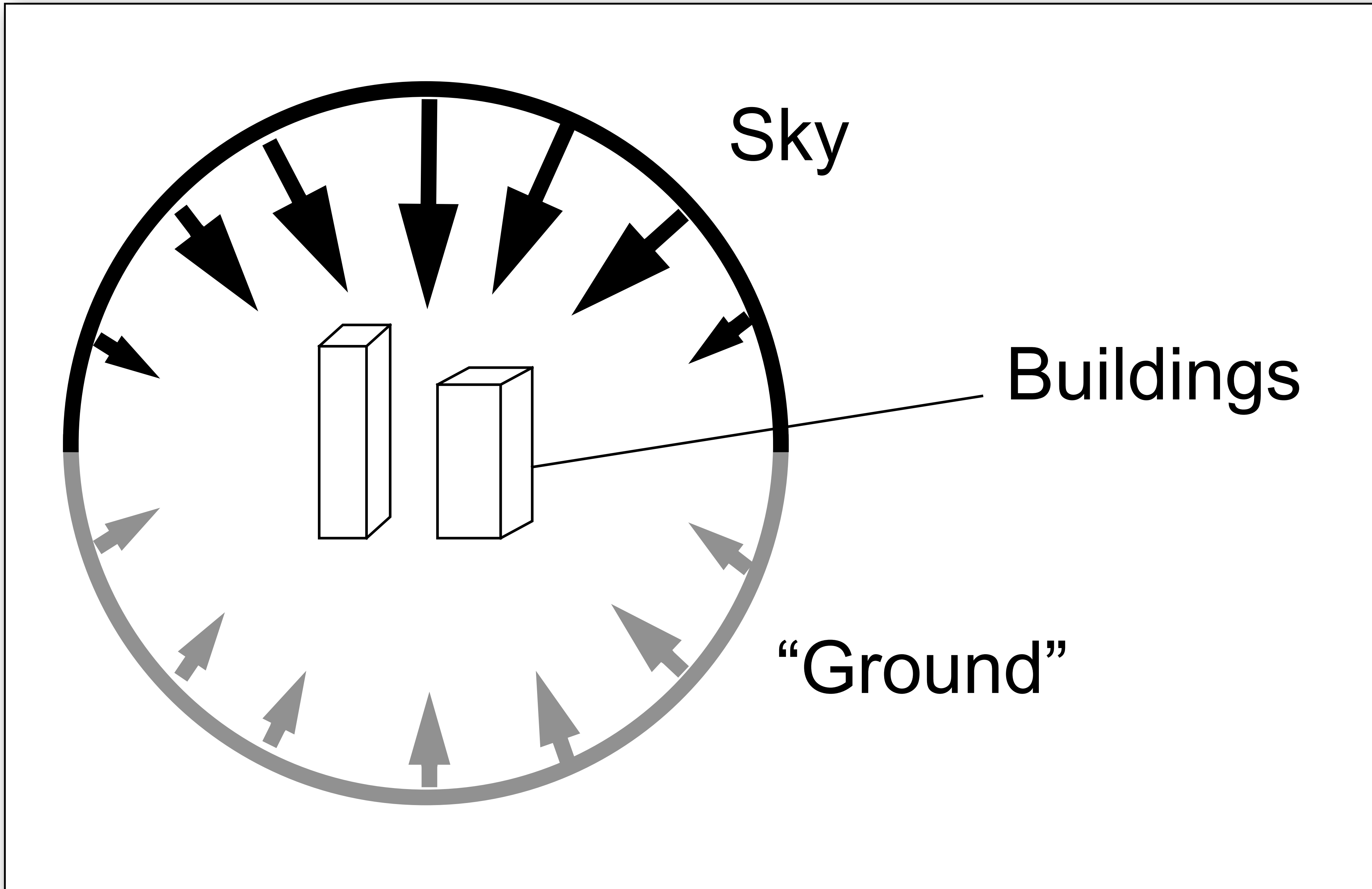
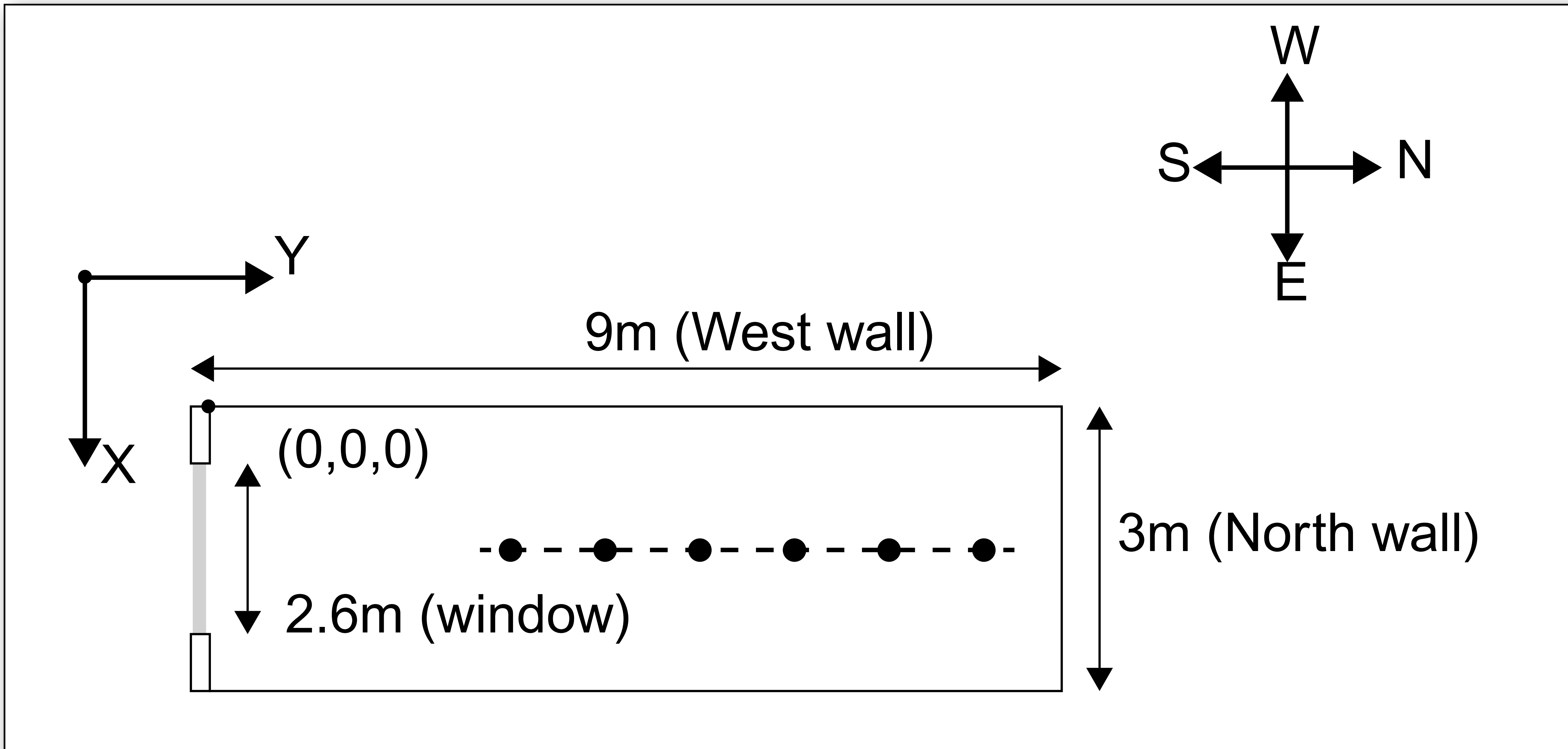


Fig 6.5 Rendering with Radiance

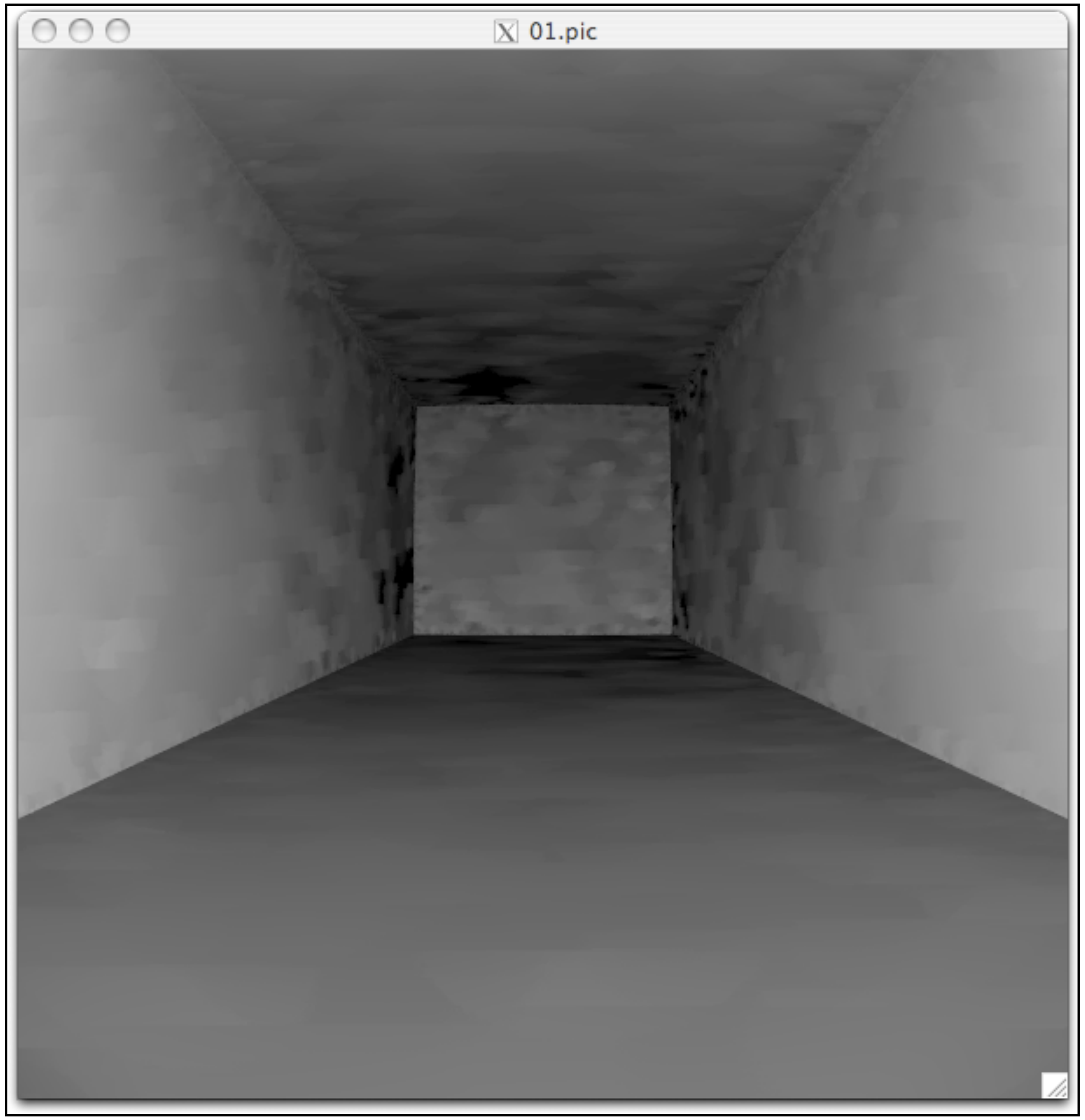
Predicting internal illuminance



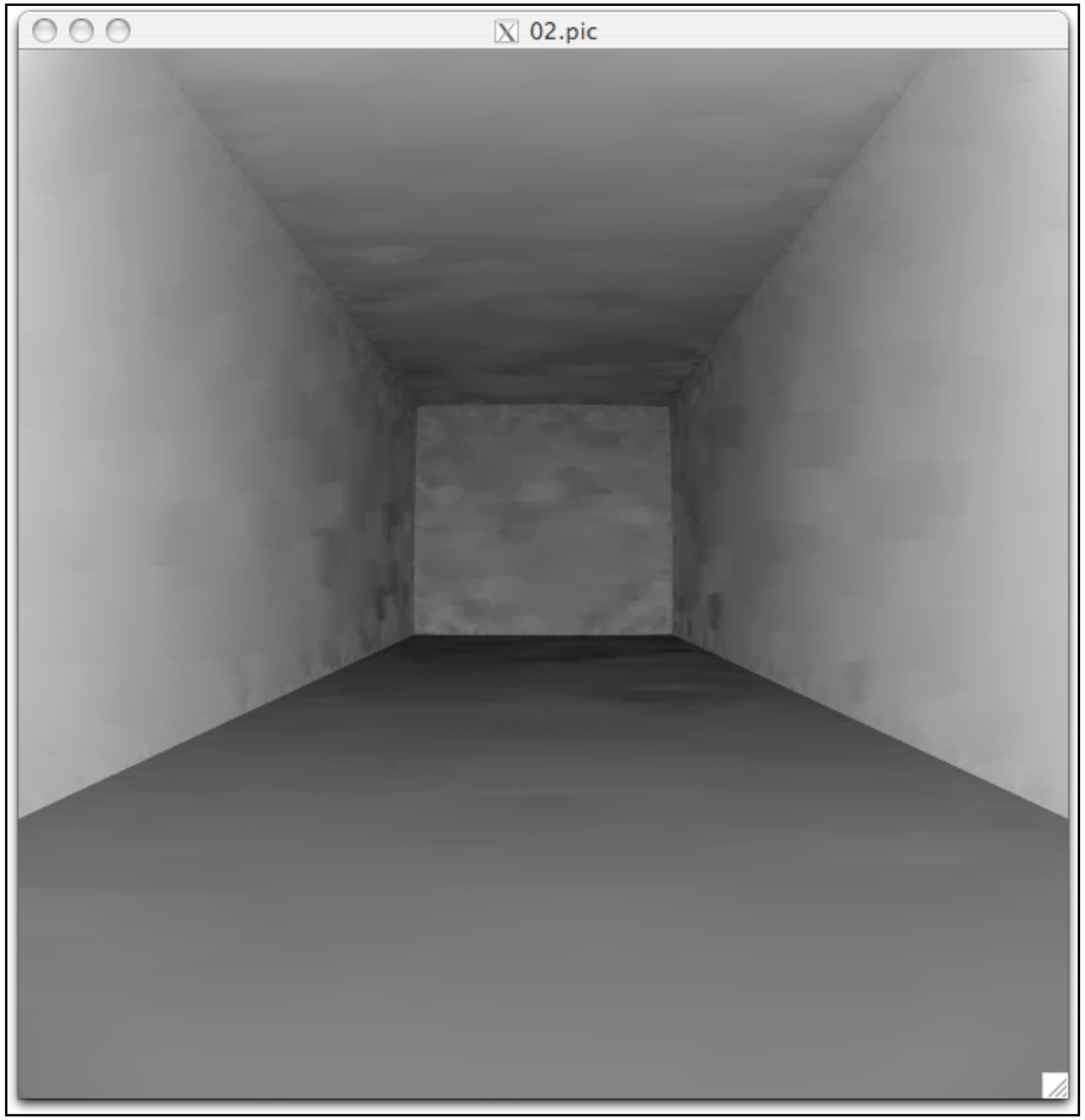
[No ground plane in this example]

```
#!/bin/csh -f
# loop through ab

foreach ab (1 2 3 4 5)
echo "Ambient bounces" $ab

# Calculate DF
  rtrace -w -h -I+ -ab $ab -aa 0.2 -ad 512 \
   -as 0 -ar 128 scene.oct \
   < samp1.inp | rcalc -e\
   '$1=($1*0.265+$2*0.670+$3*0.065*179/10000*100'

end
```

ab 1



ab 2



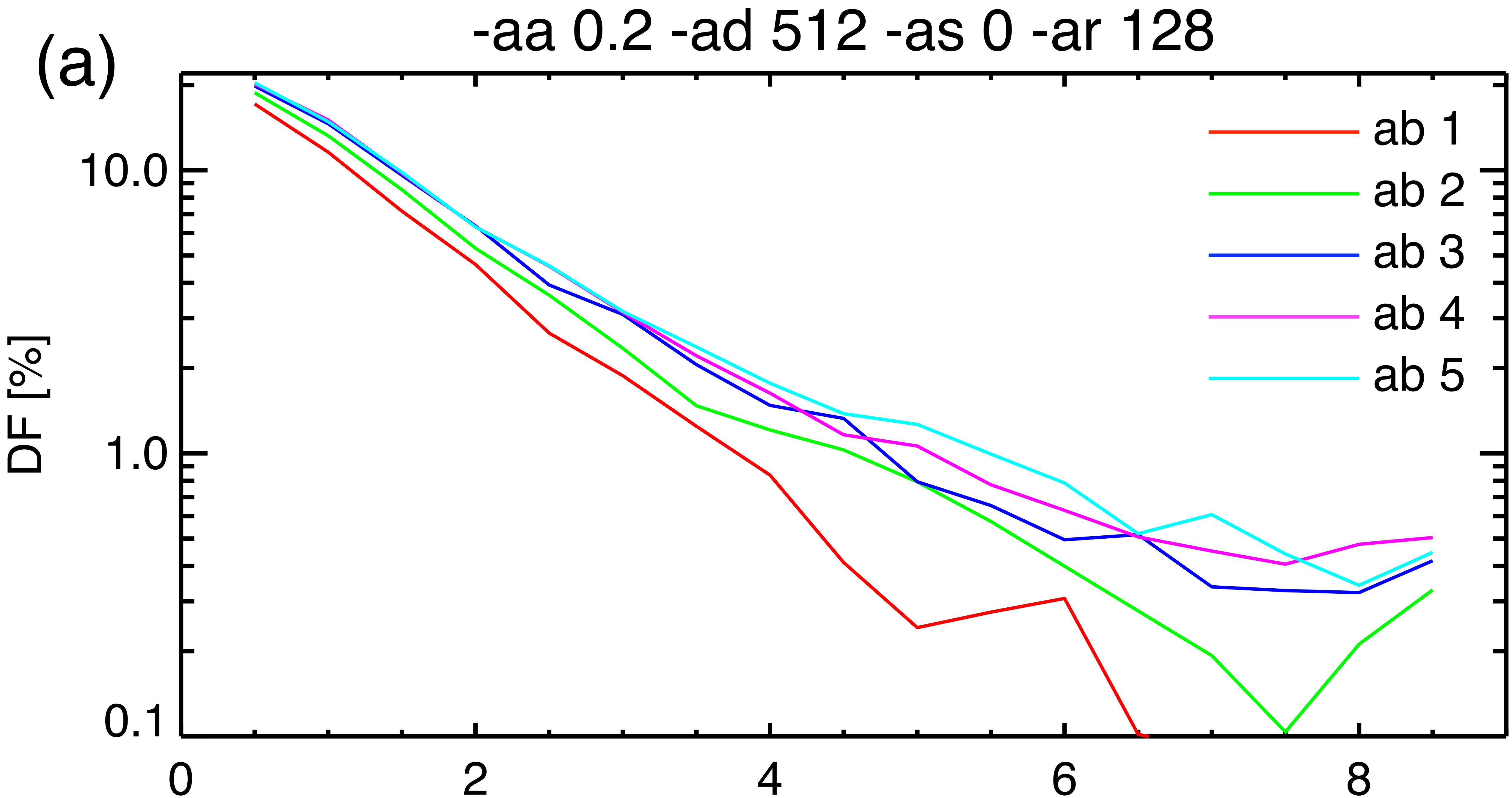


Fig 6.7 Rendering with Radiance

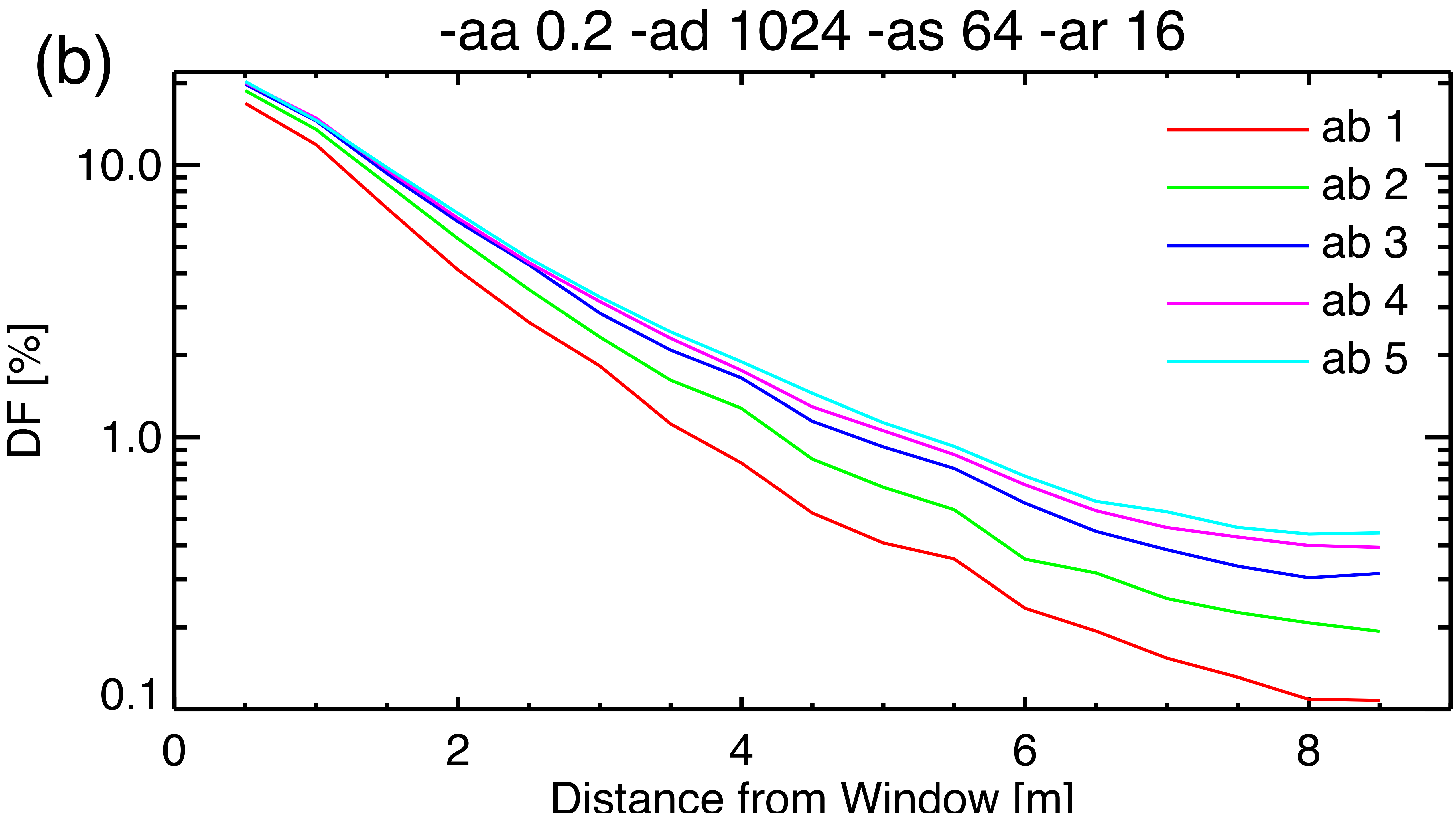


Fig 6.7 Rendering with Radiance

Questions?

Adding complexity

- Now we add a ground plane and a nearby building to our simple scene. We model the ground plane as a disc of, say, radius 20 meters, centered on the origin.

- External obstruction is a nearby building positioned so that it faces the room window and obscures much of the view of the sky from inside the room. The DF predictions are repeated as before, only now we increase the maximum -ab to 7.

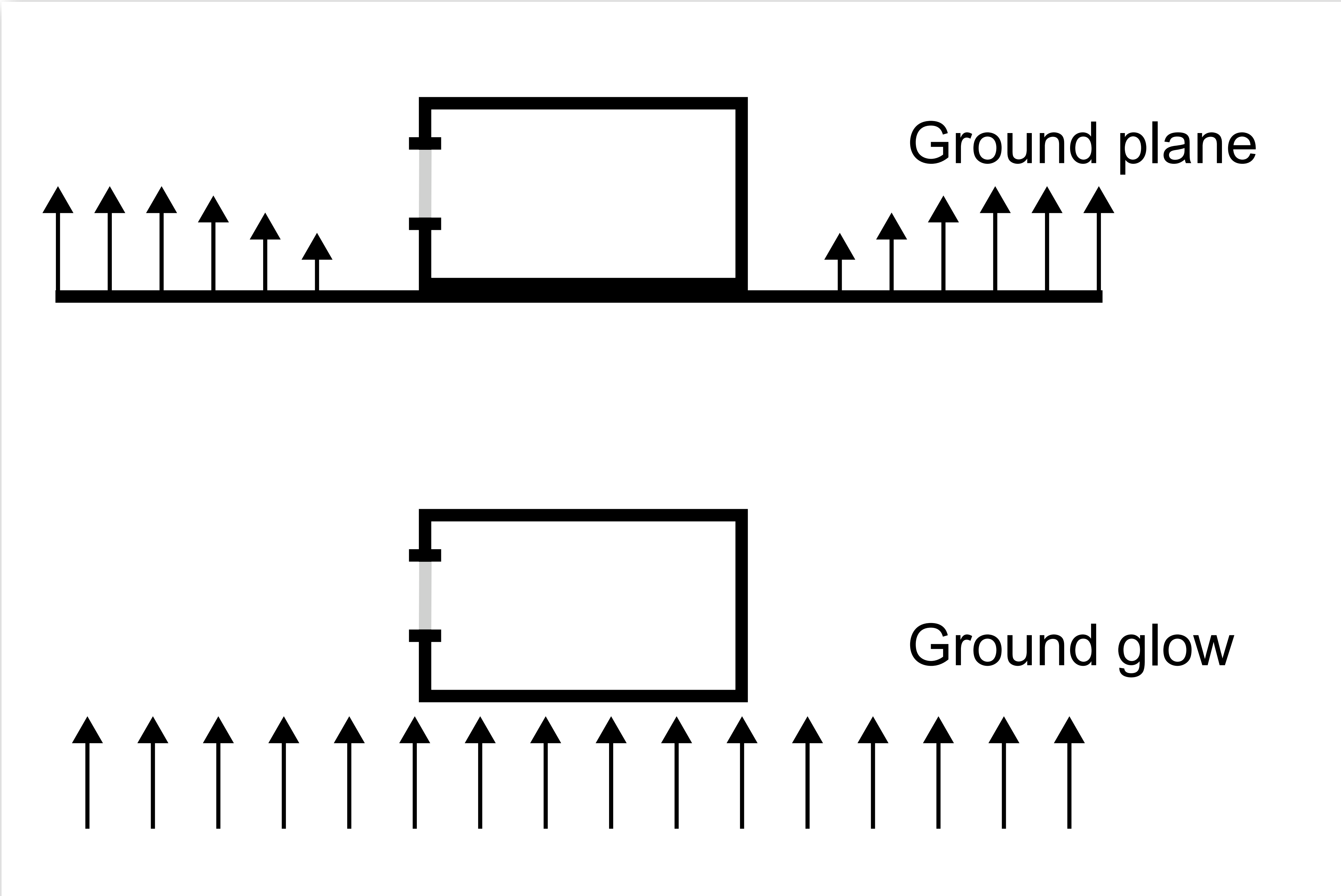


Fig 6.8 Rendering with Radiance

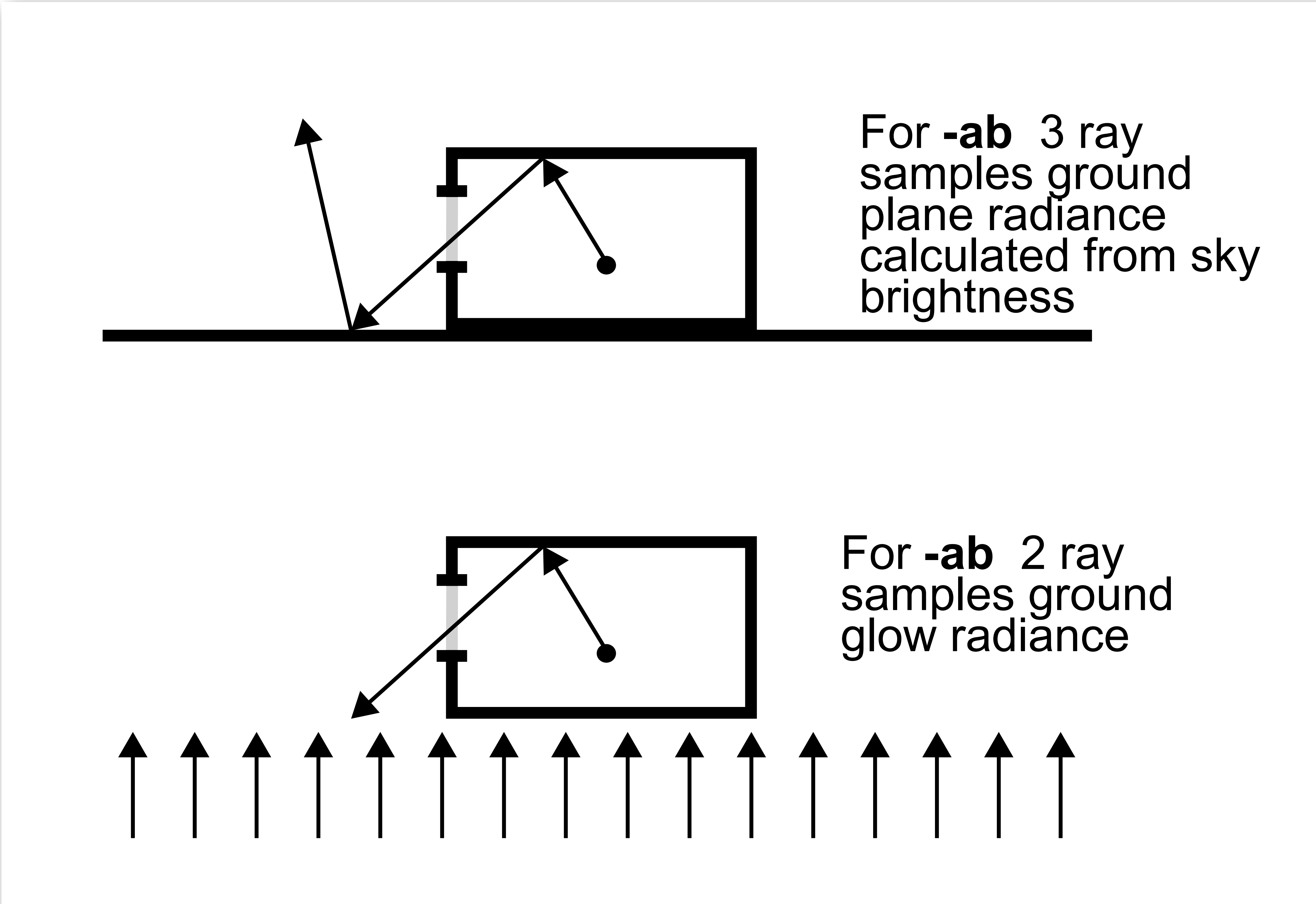
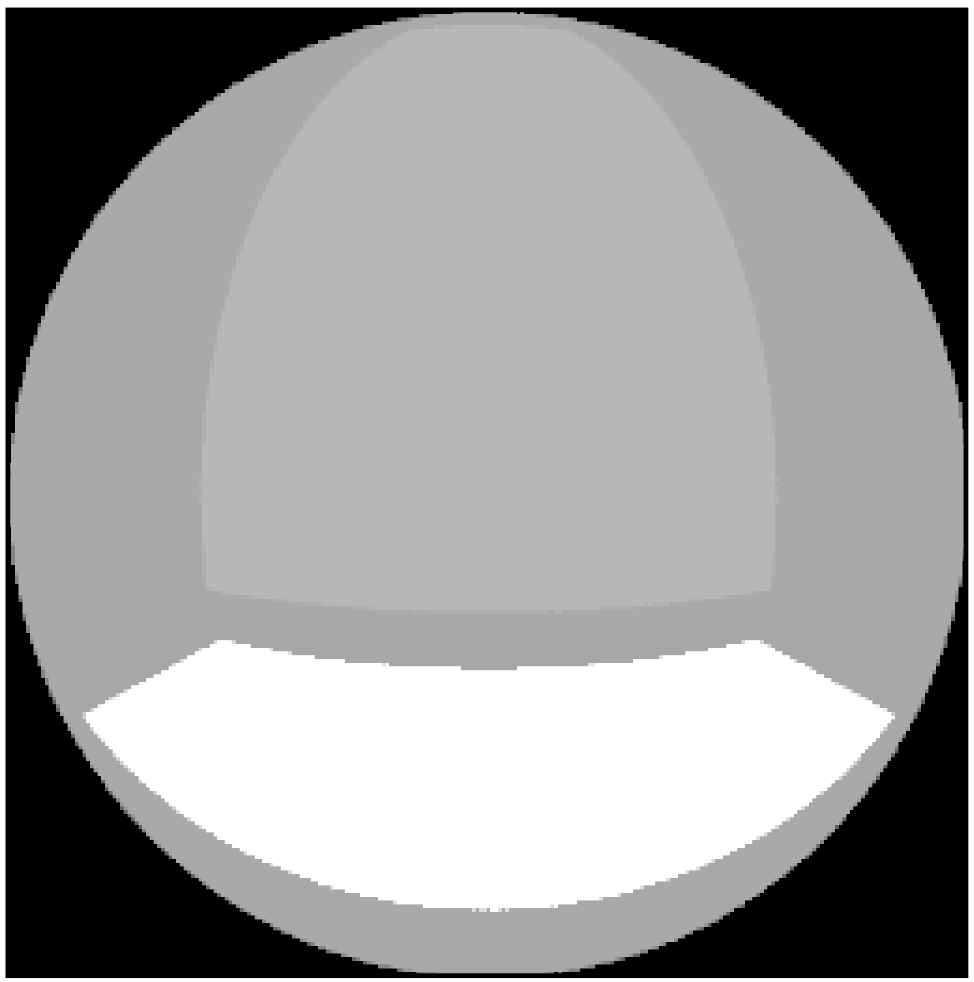
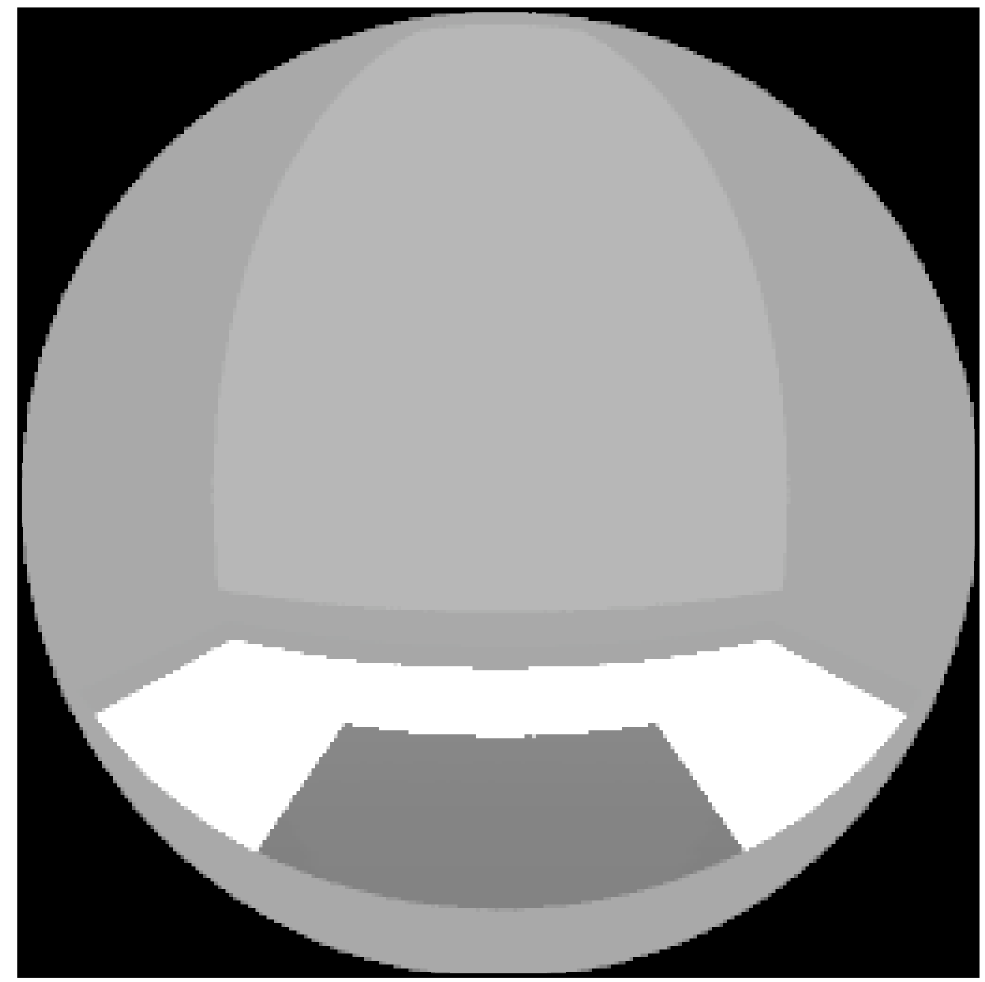


Fig 6.9 Rendering with Radiance

Photocell's 'view' from the front near the window



Previous



With obstruction

Fig 6.12 Rendering with Radiance

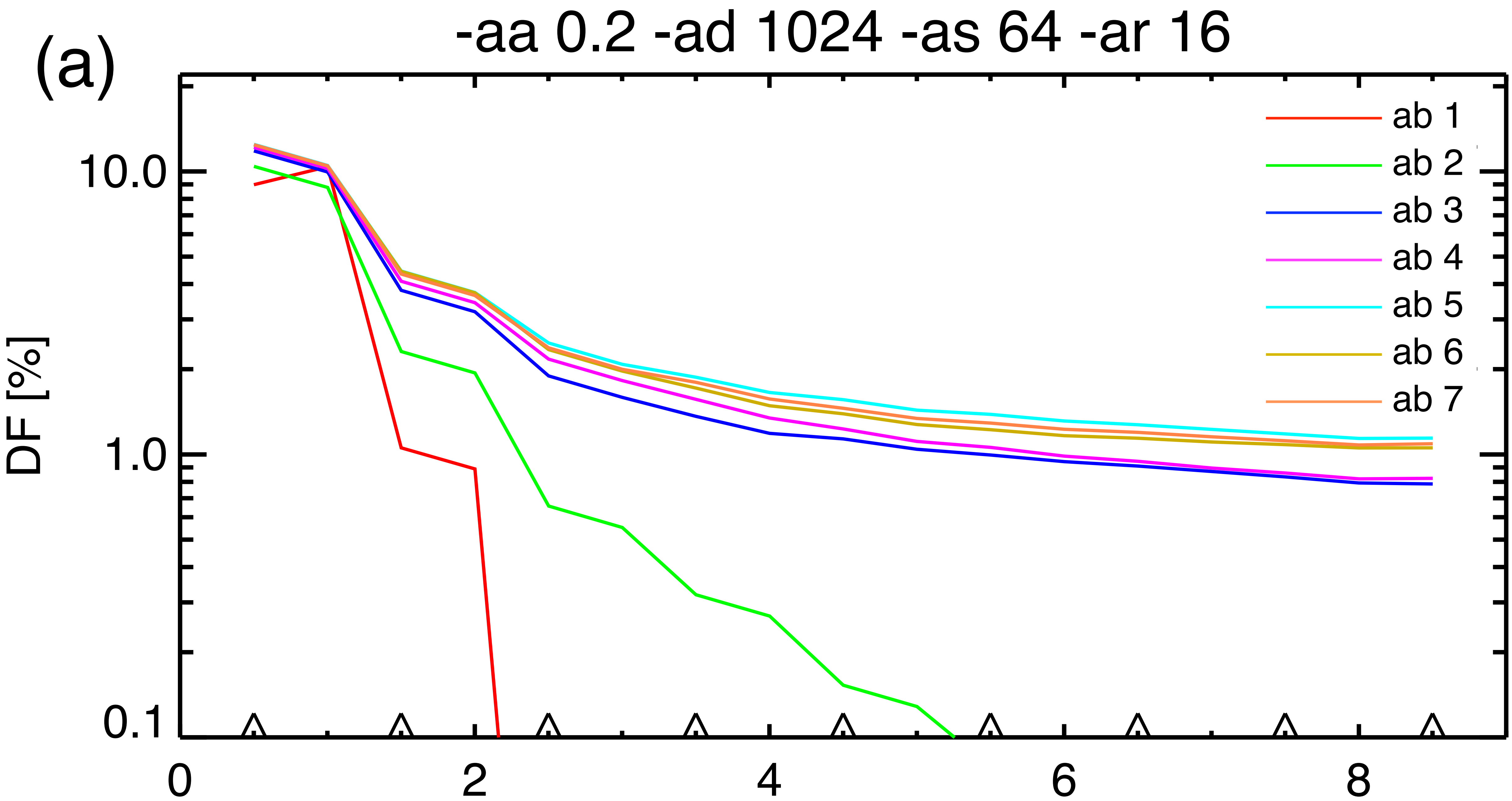


Fig 6.10 Rendering with Radiance

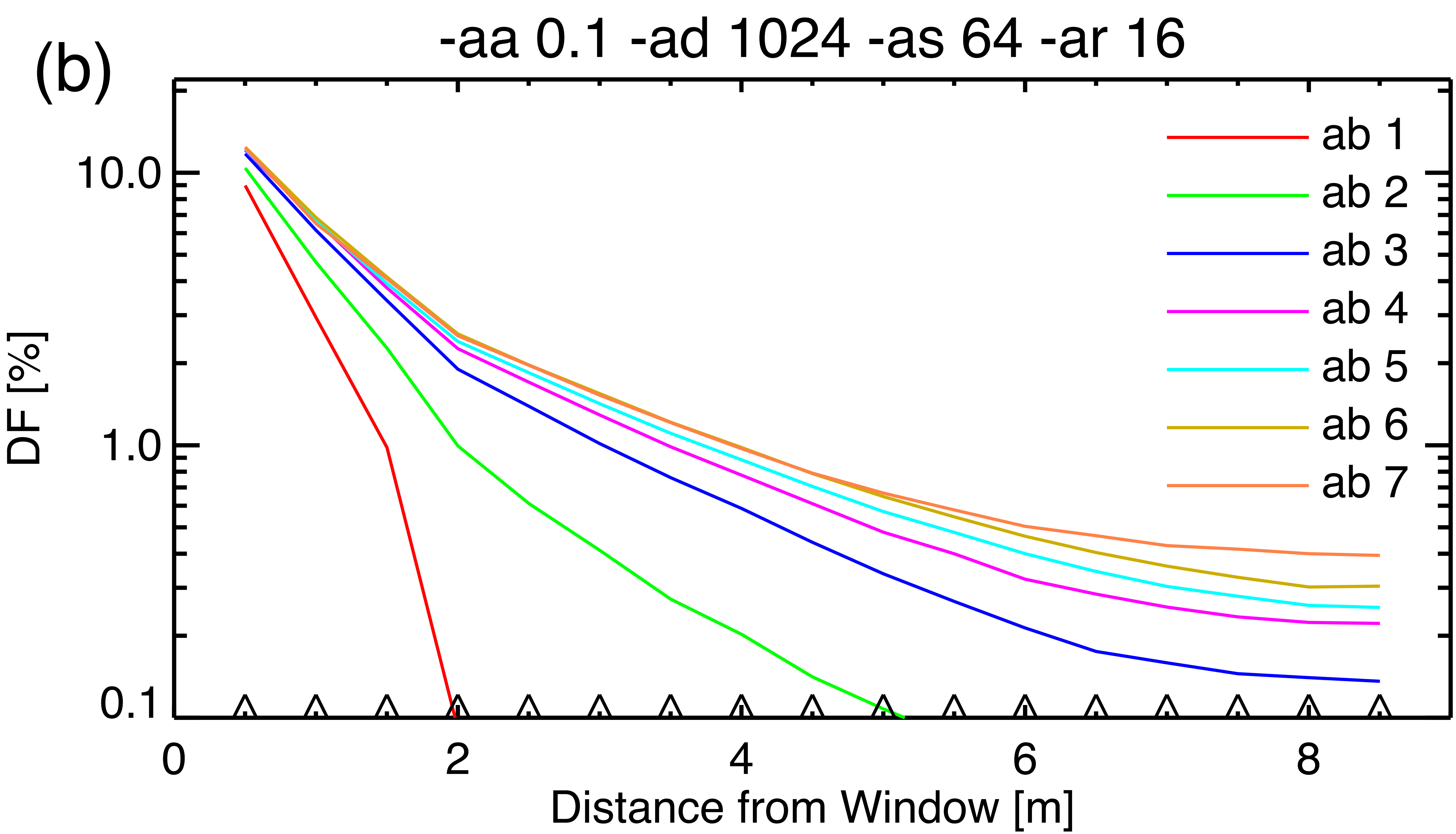
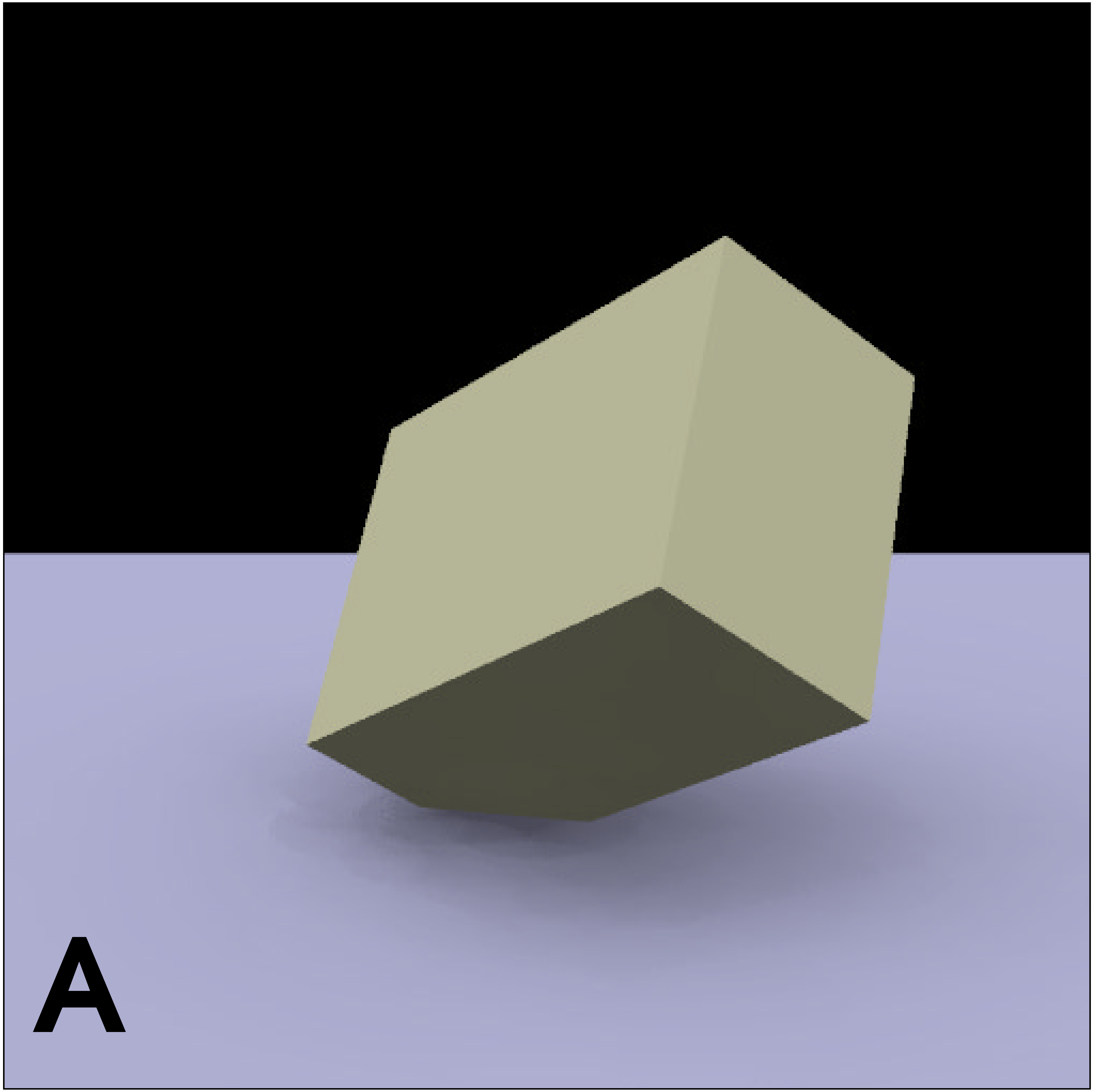
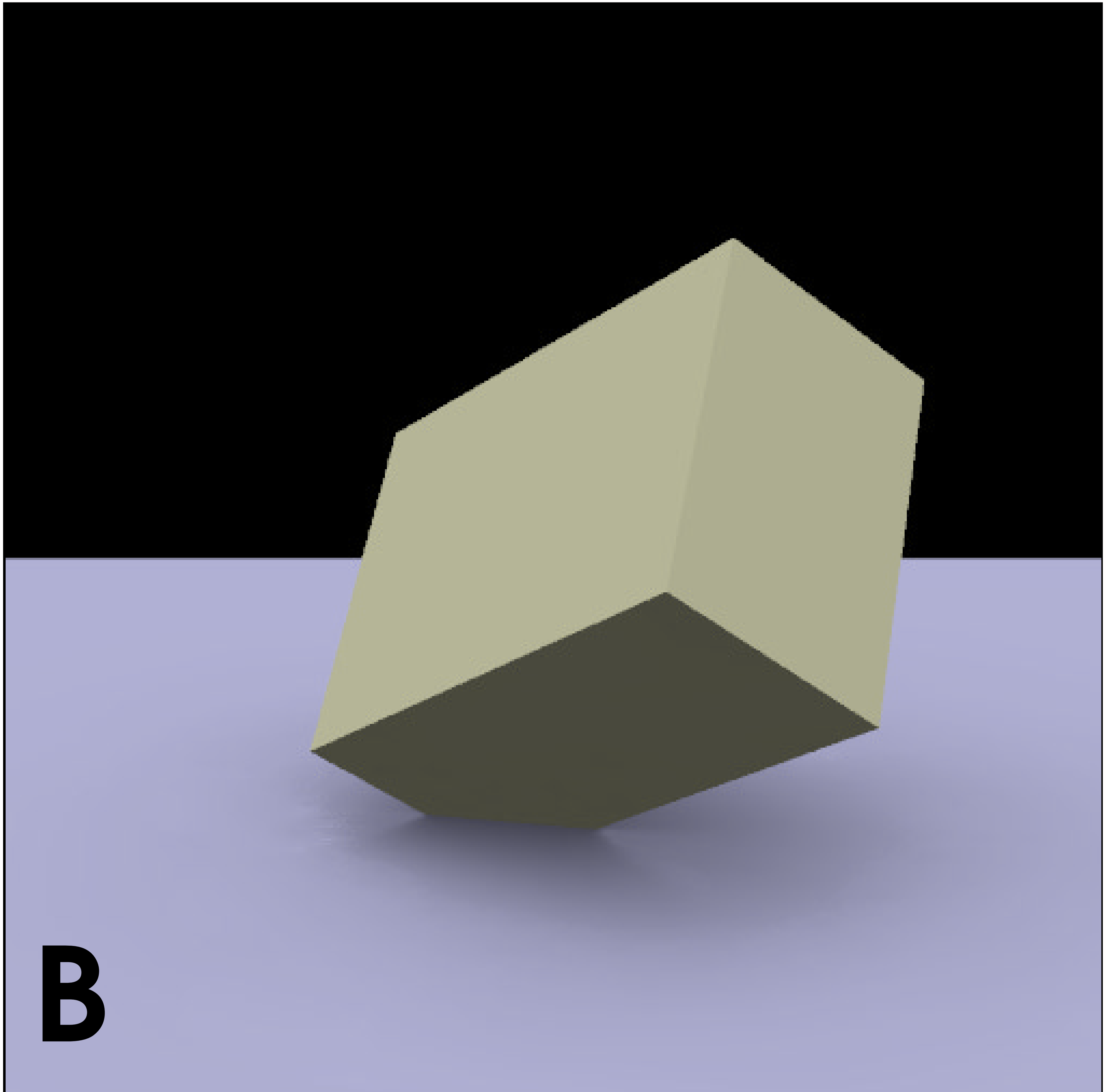


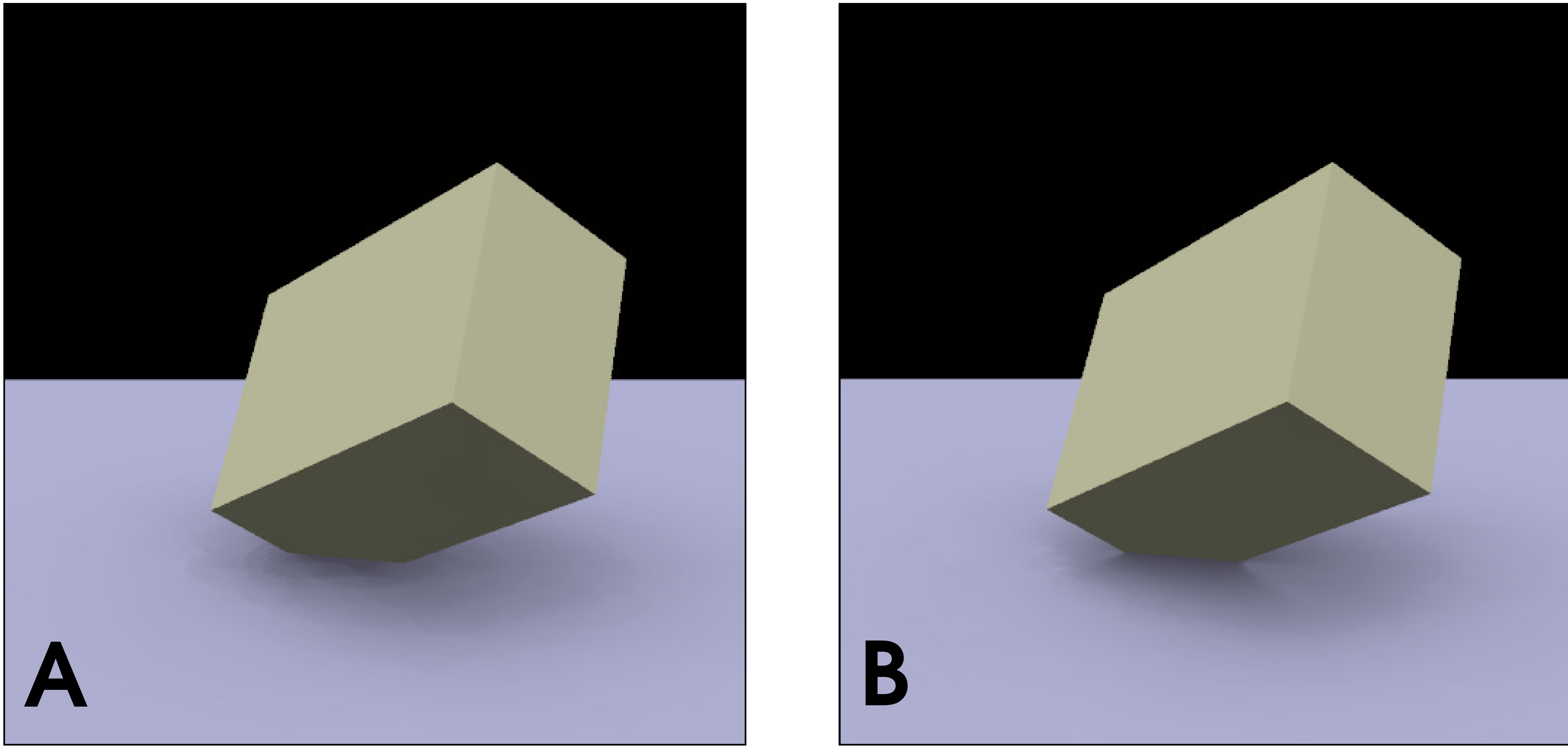
Fig 6.10 Rendering with Radiance

The ambient resolution parameter [**ar**]


A


B

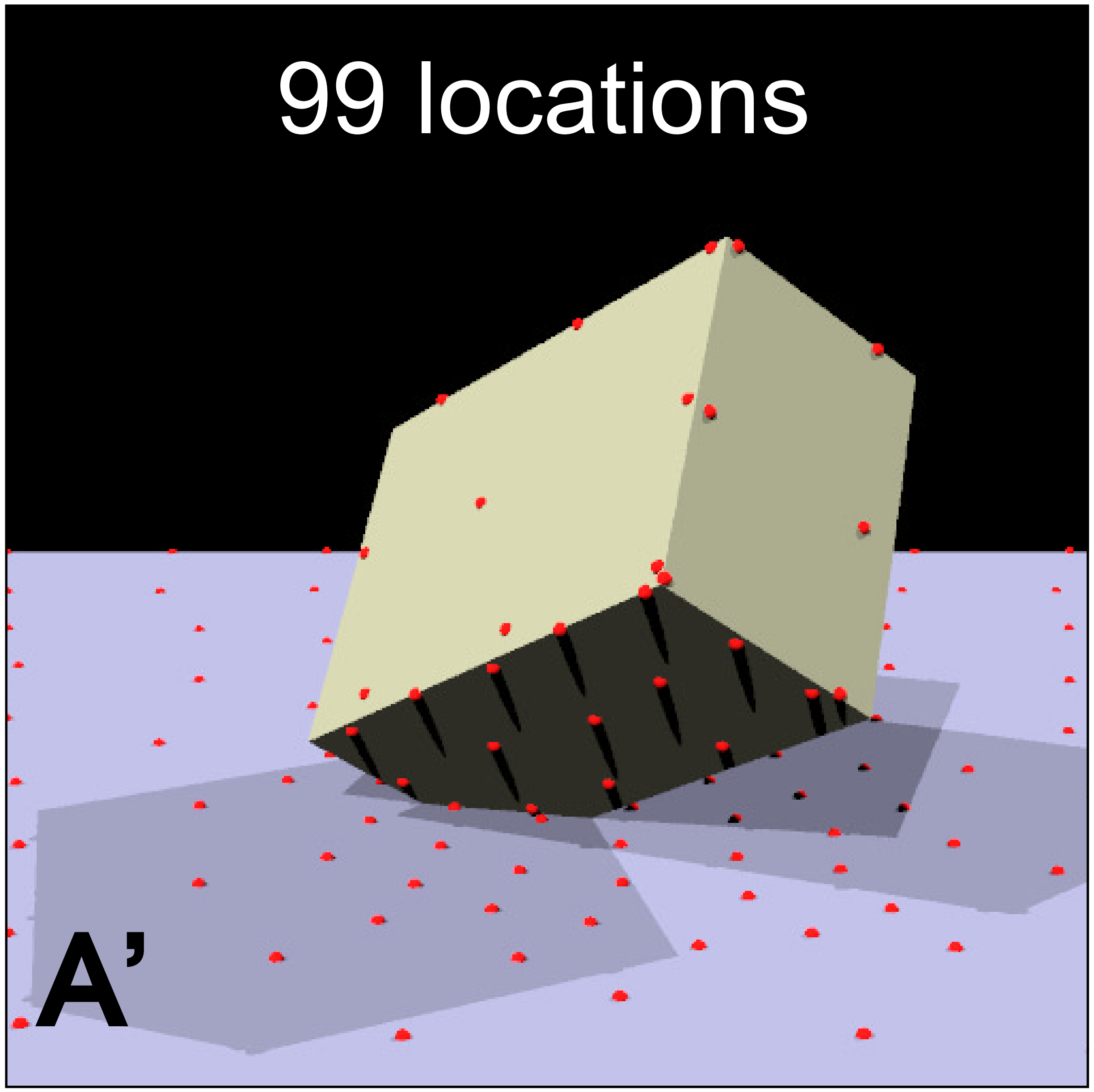
-ar 4 -ar 64

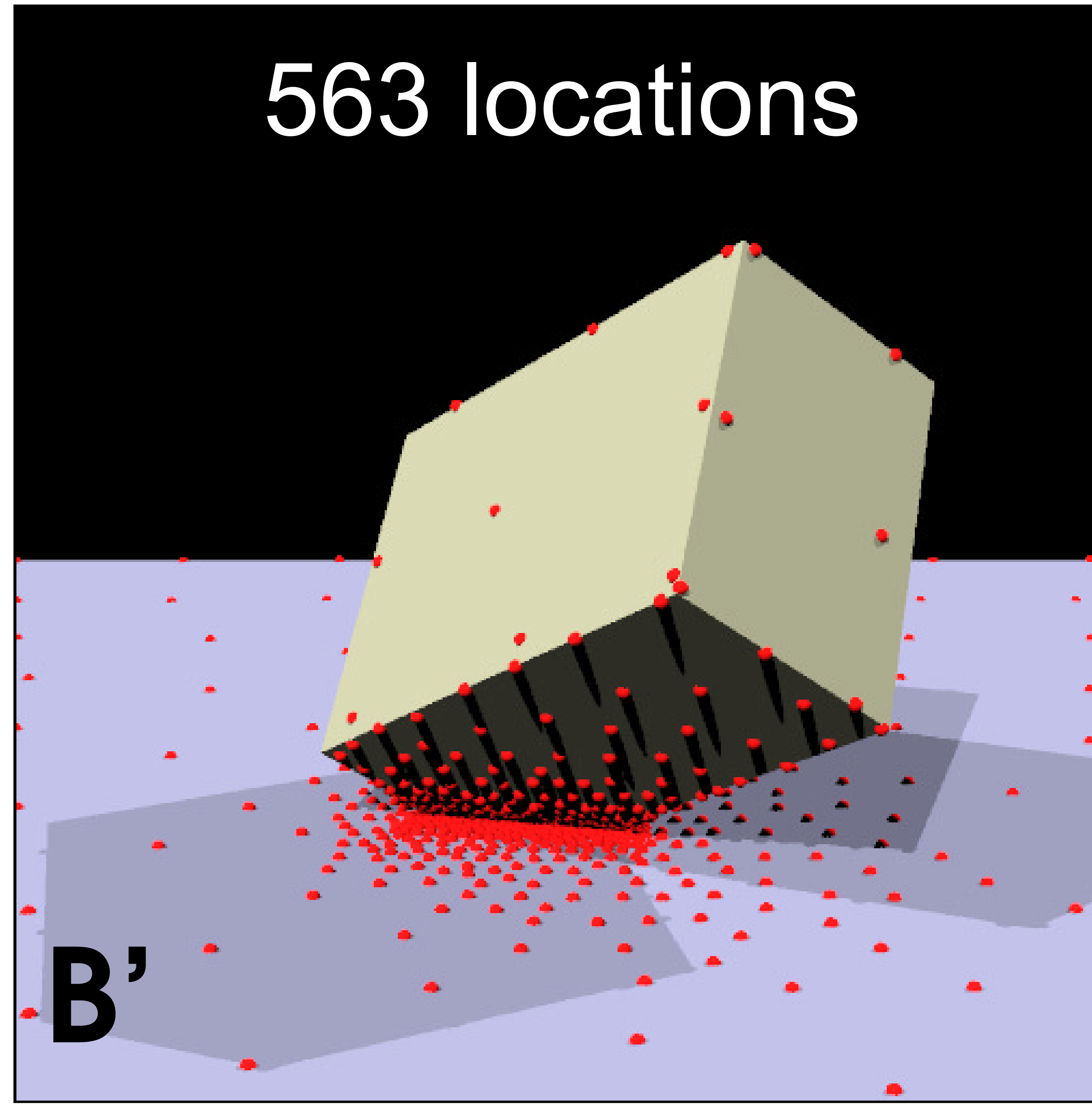


-ad 2048 -as 128 -ab 1 -aa 0.15 -av 0 0 0

```
-ar 4
```

```
-ar 64
```

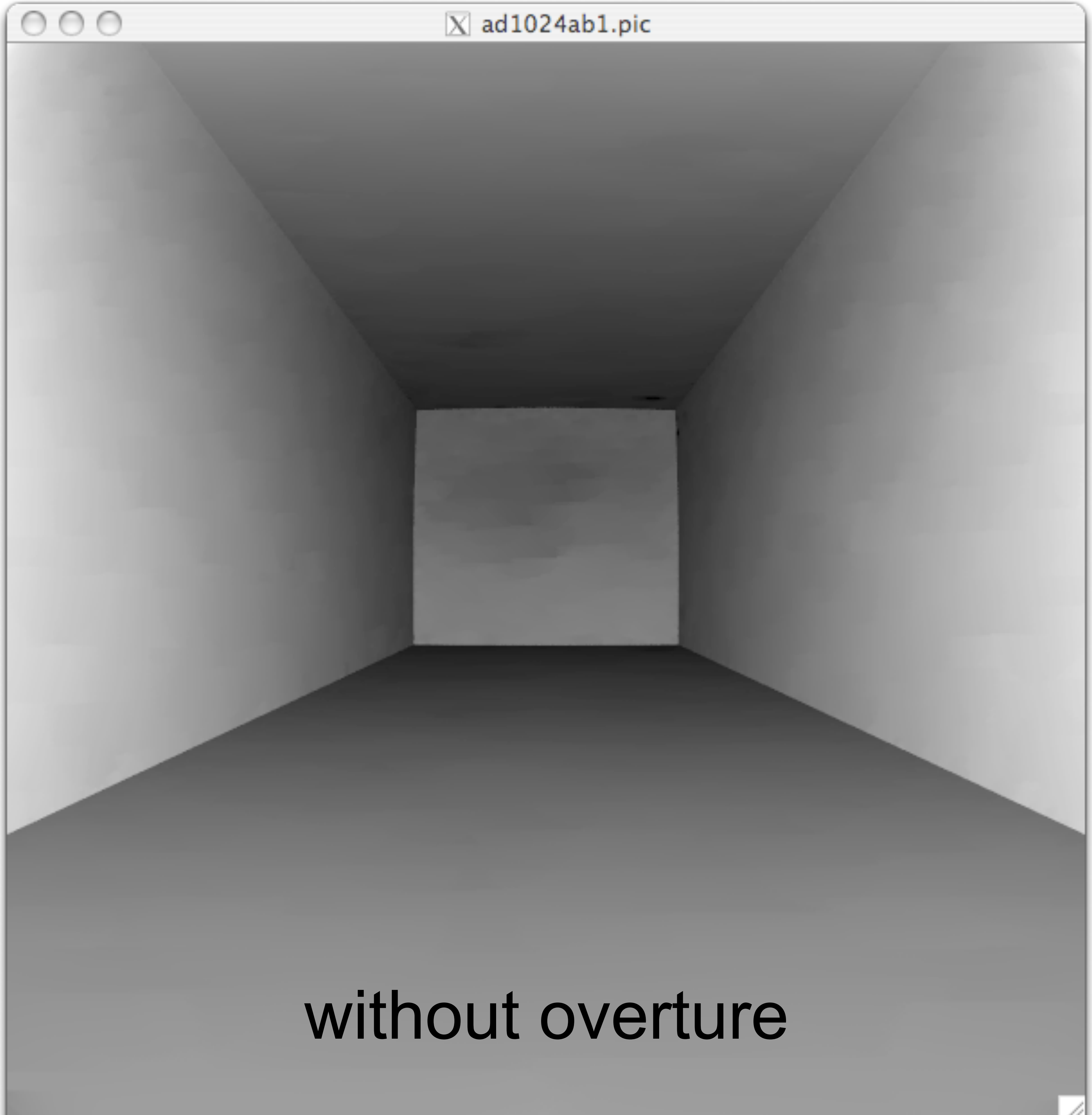




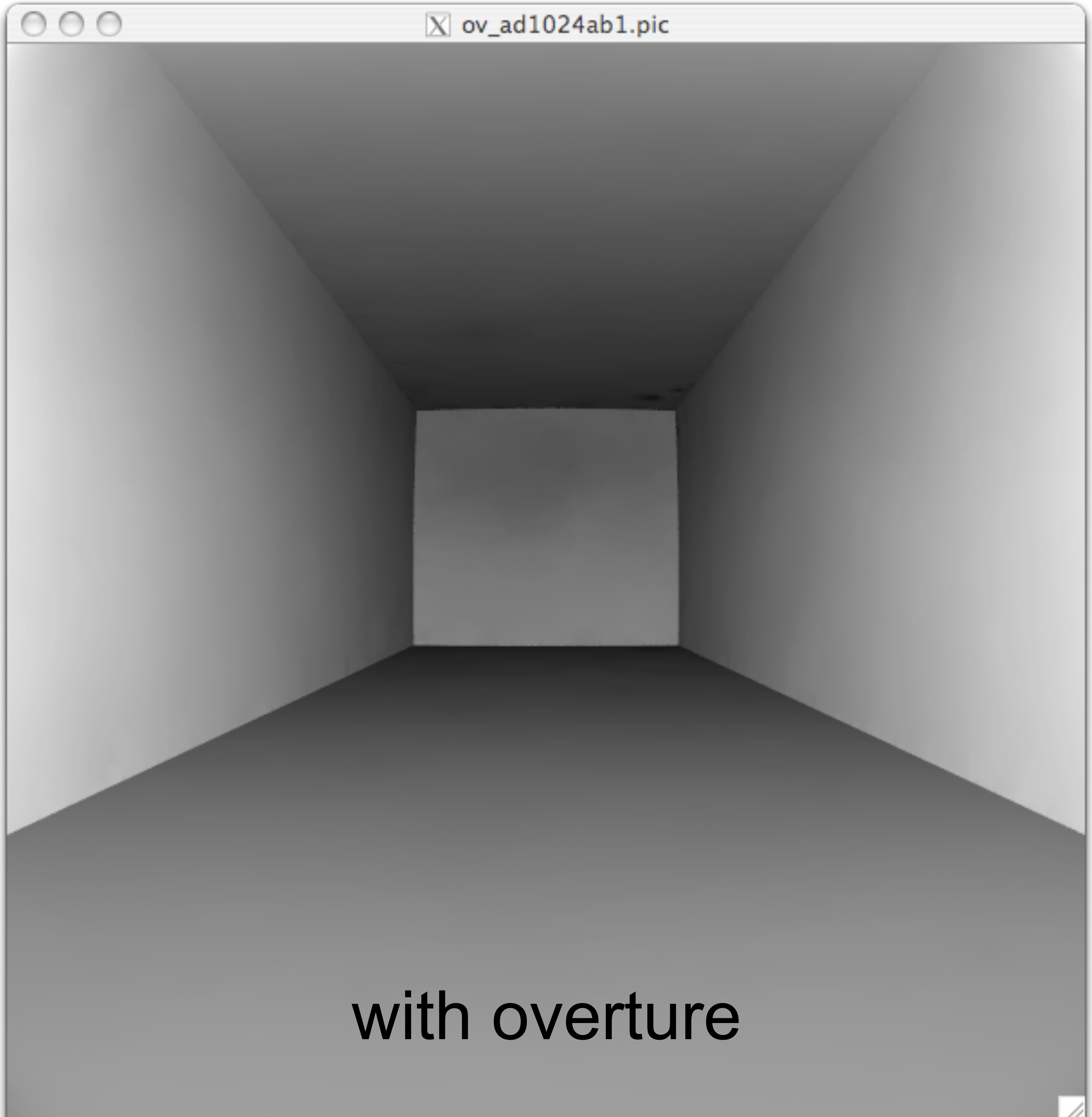
$$S_{min} = \frac{D_{max} \times aa}{ar}$$

The overture calculation

- Execute the simulation as normal, however save the ambient file (i.e. values determined from hemispherical sampling), but **don’t** keep the image.
- Then, redo the simulation using the saved ambient file and the <u>same</u> ambient parameters.



without overture


ov_ad1024ab1.pic
with overture

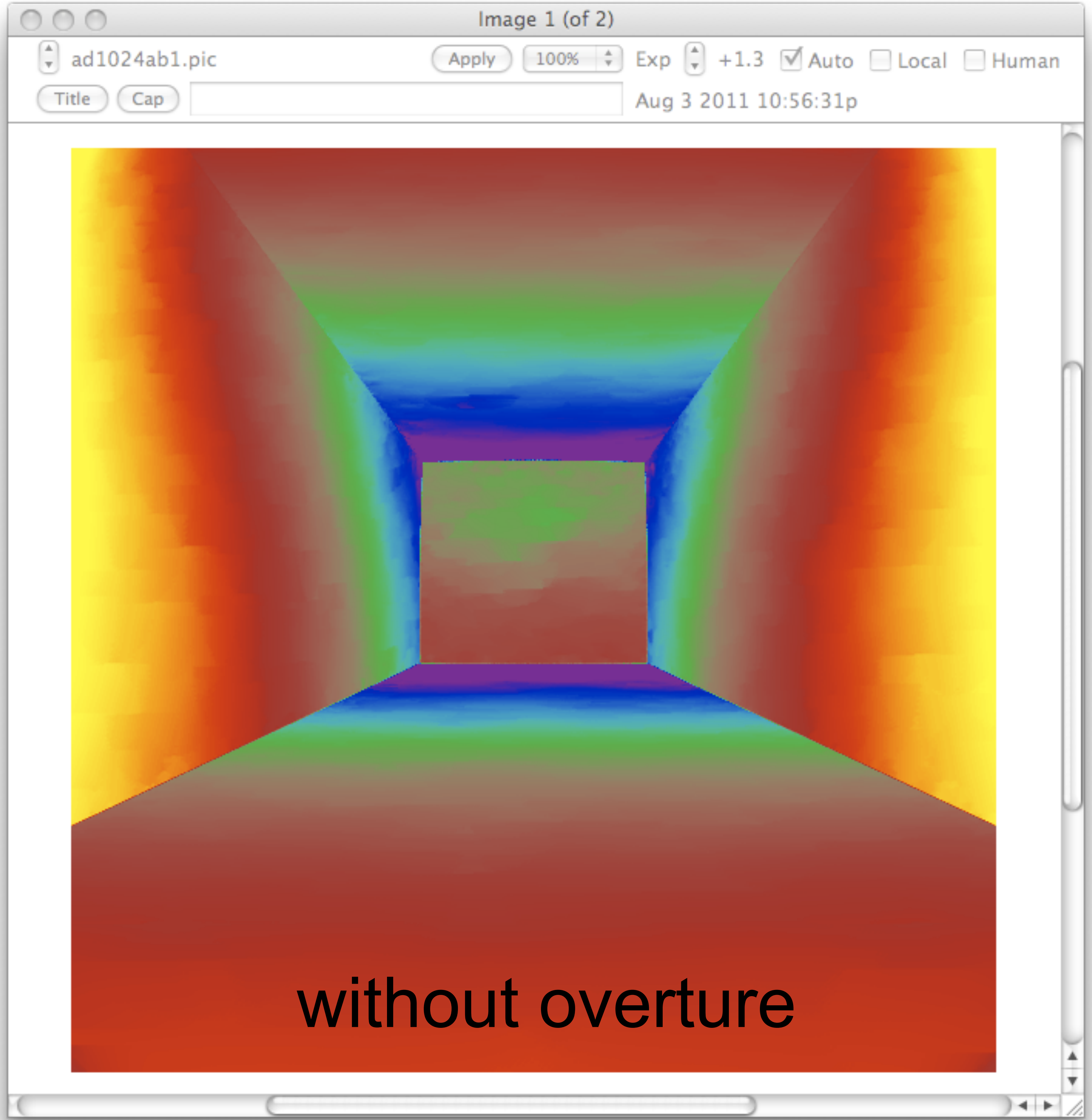


Image 1 (of 2)
ad1024ab1.pic
Apply
100%
Exp
+1.3
Auto
Local
Human
Title
Cap
Aug 3 2011 10:56:31p
without overture

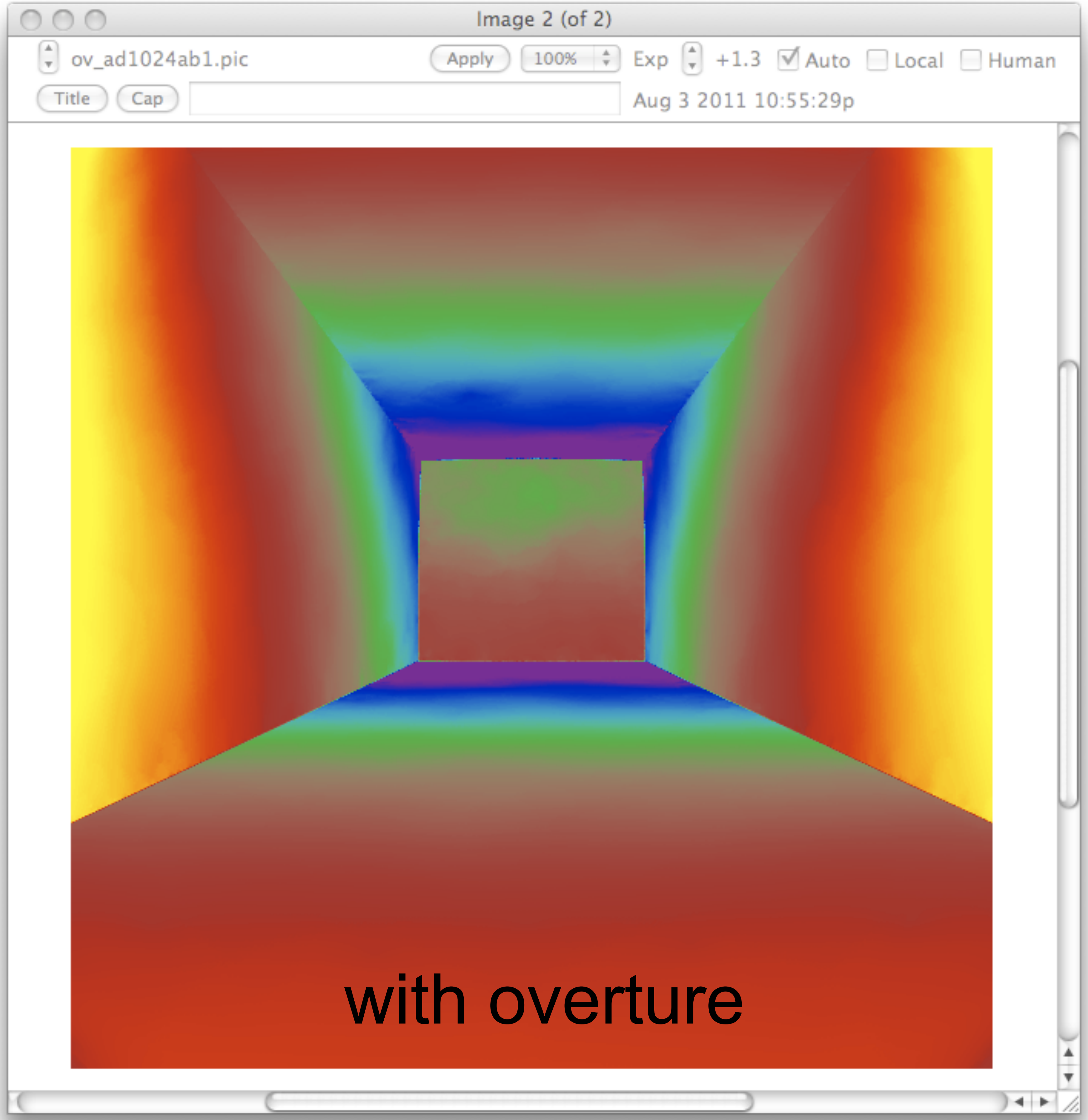
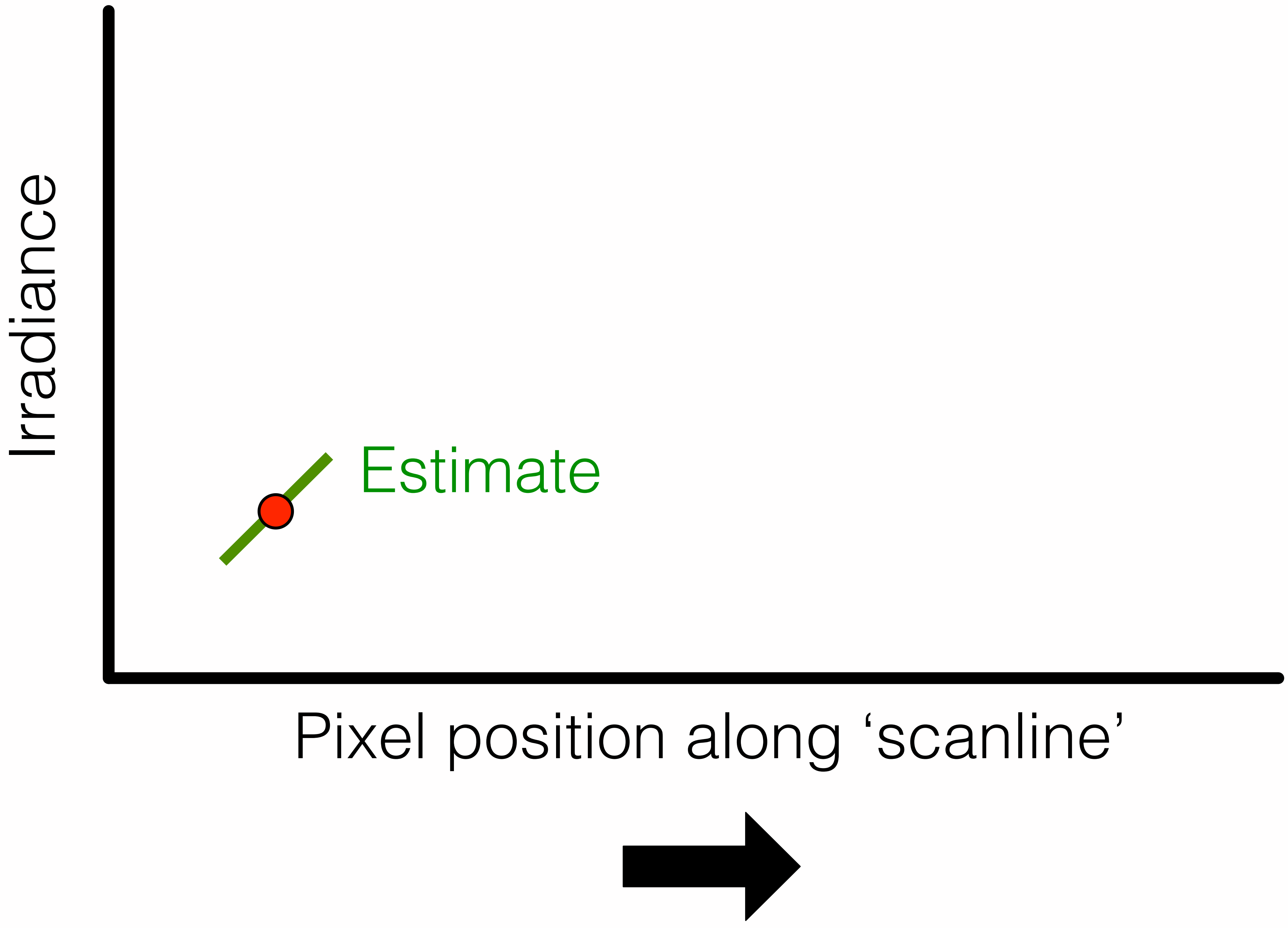
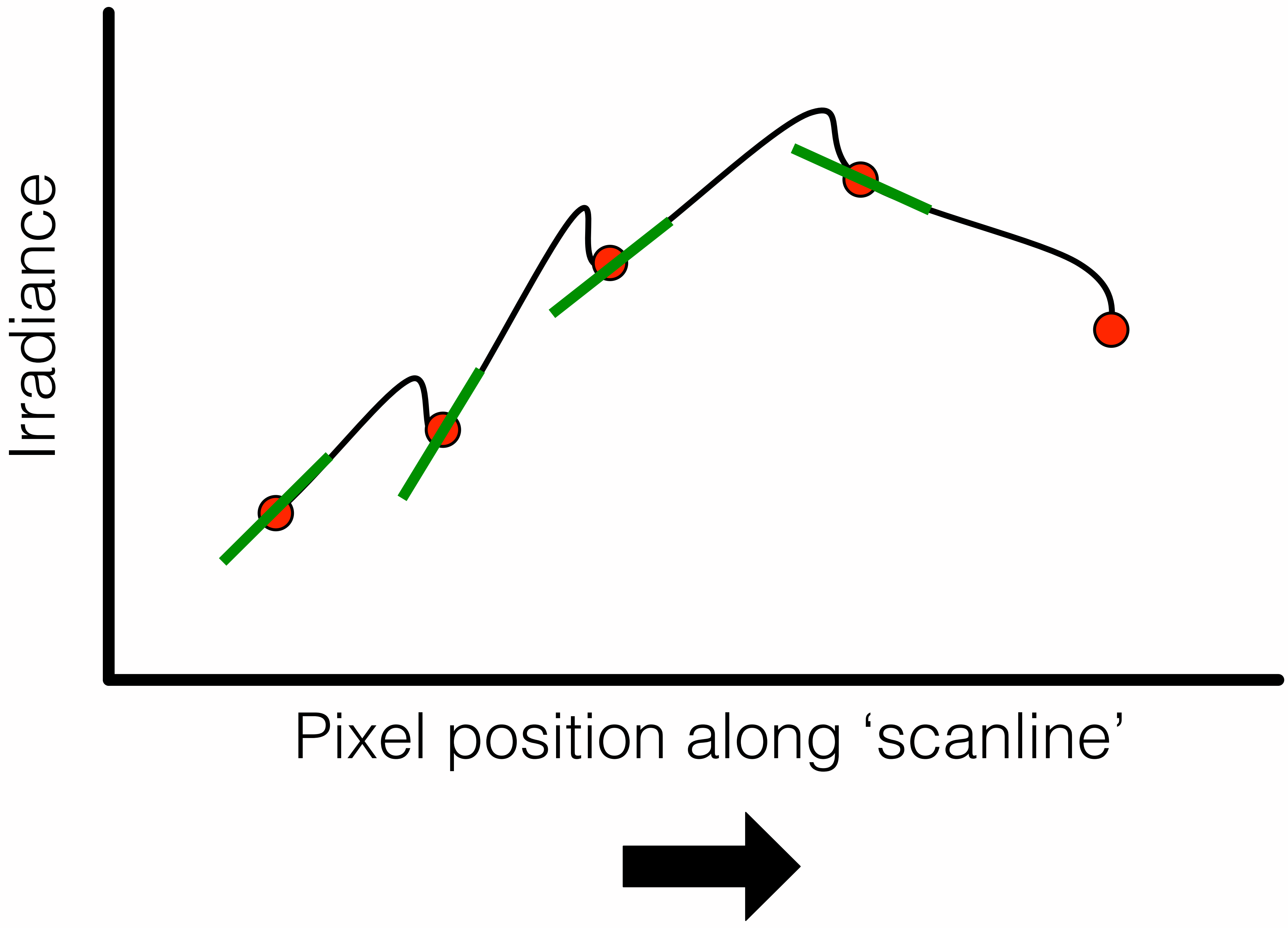
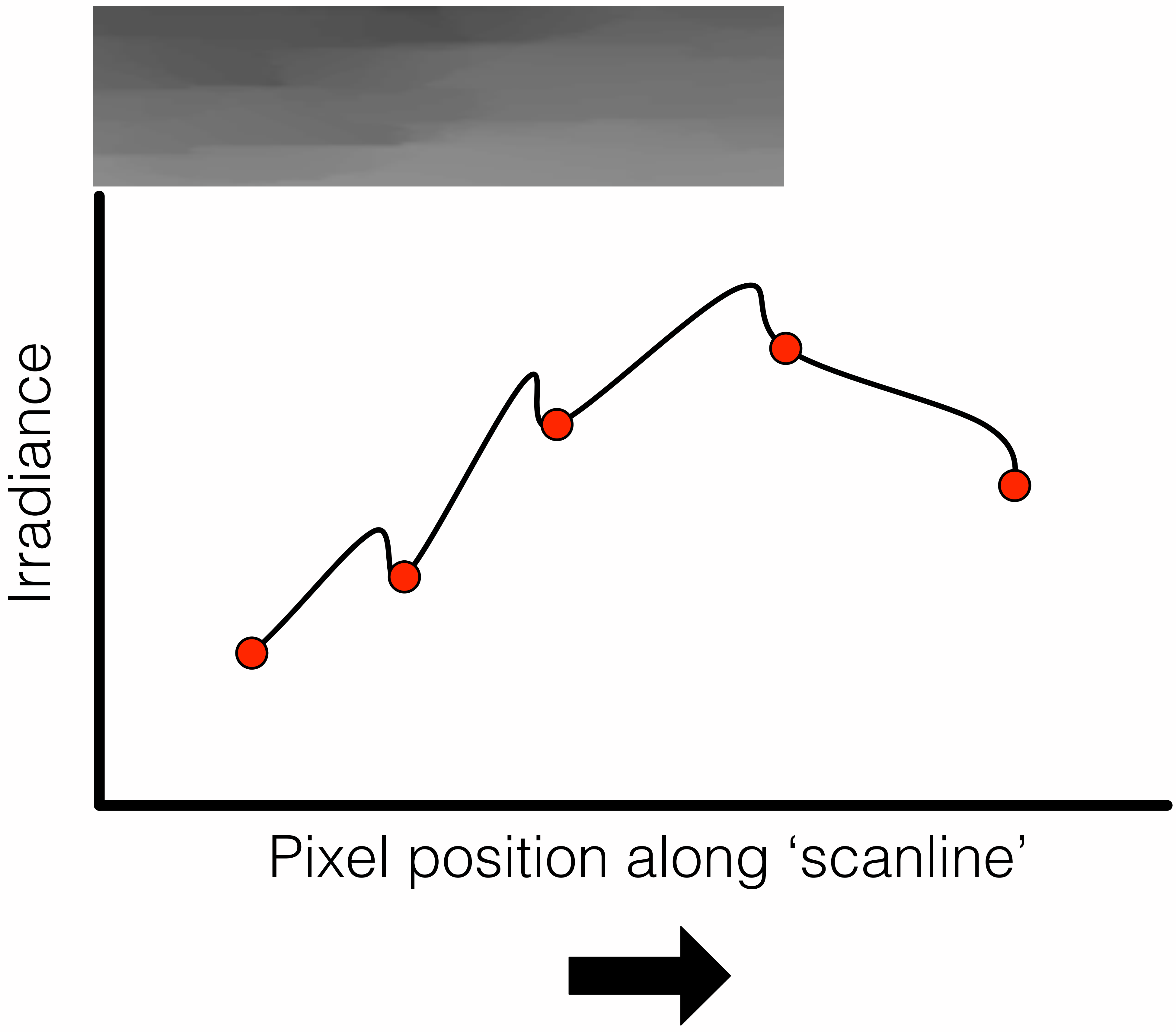


Image 2 (of 2)
ov_ad1024ab1.pic
Apply
100%
Exp
+1.3
Auto
Local
Human
Title
Cap
Aug 3 2011 10:55:29p
with overture

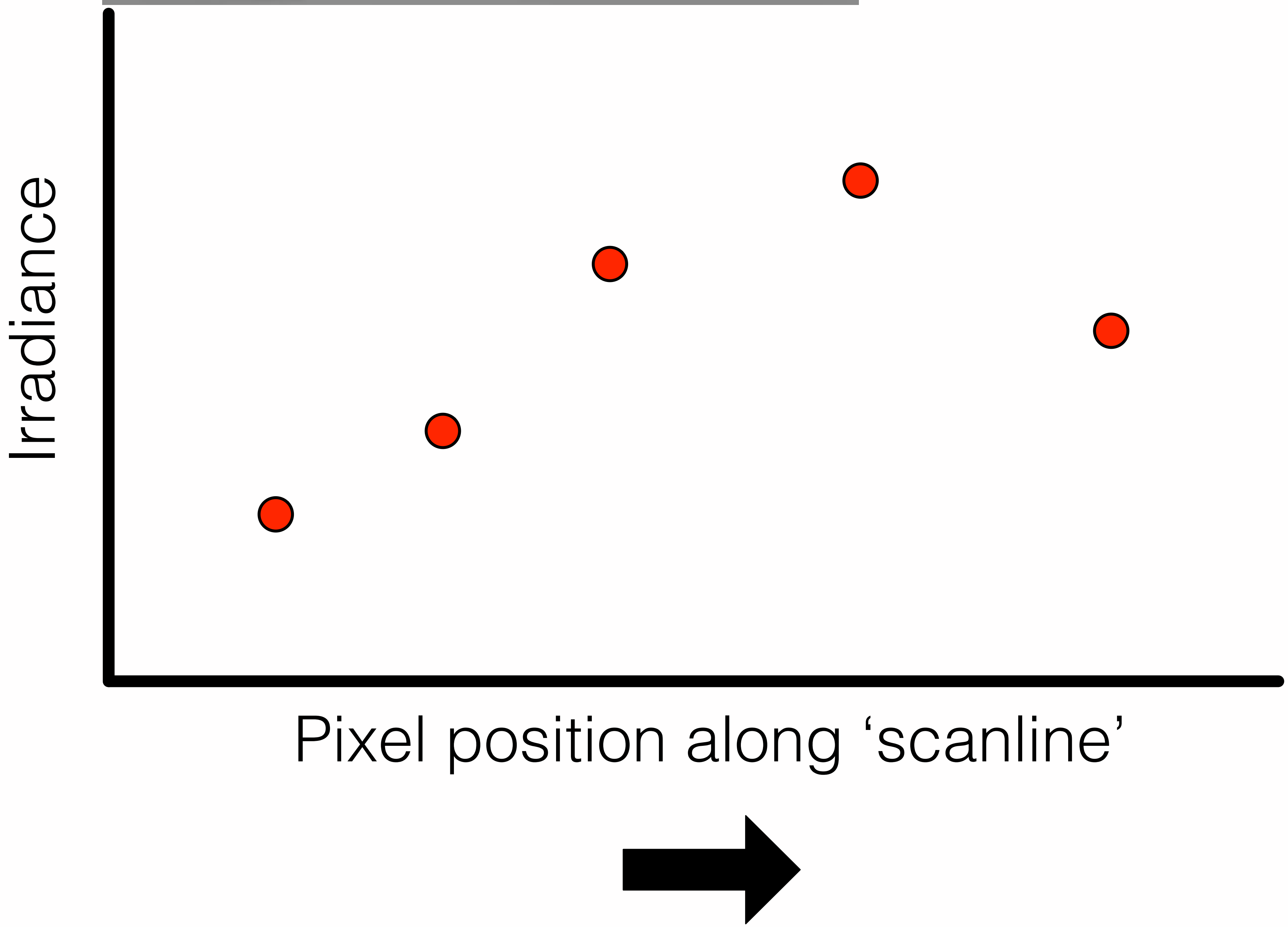

Irradiance
Estimate
Pixel position along ‘scanline’


Irradiance
Pixel position along ‘scanline’


Irradiance
Pixel position along 'scanline'

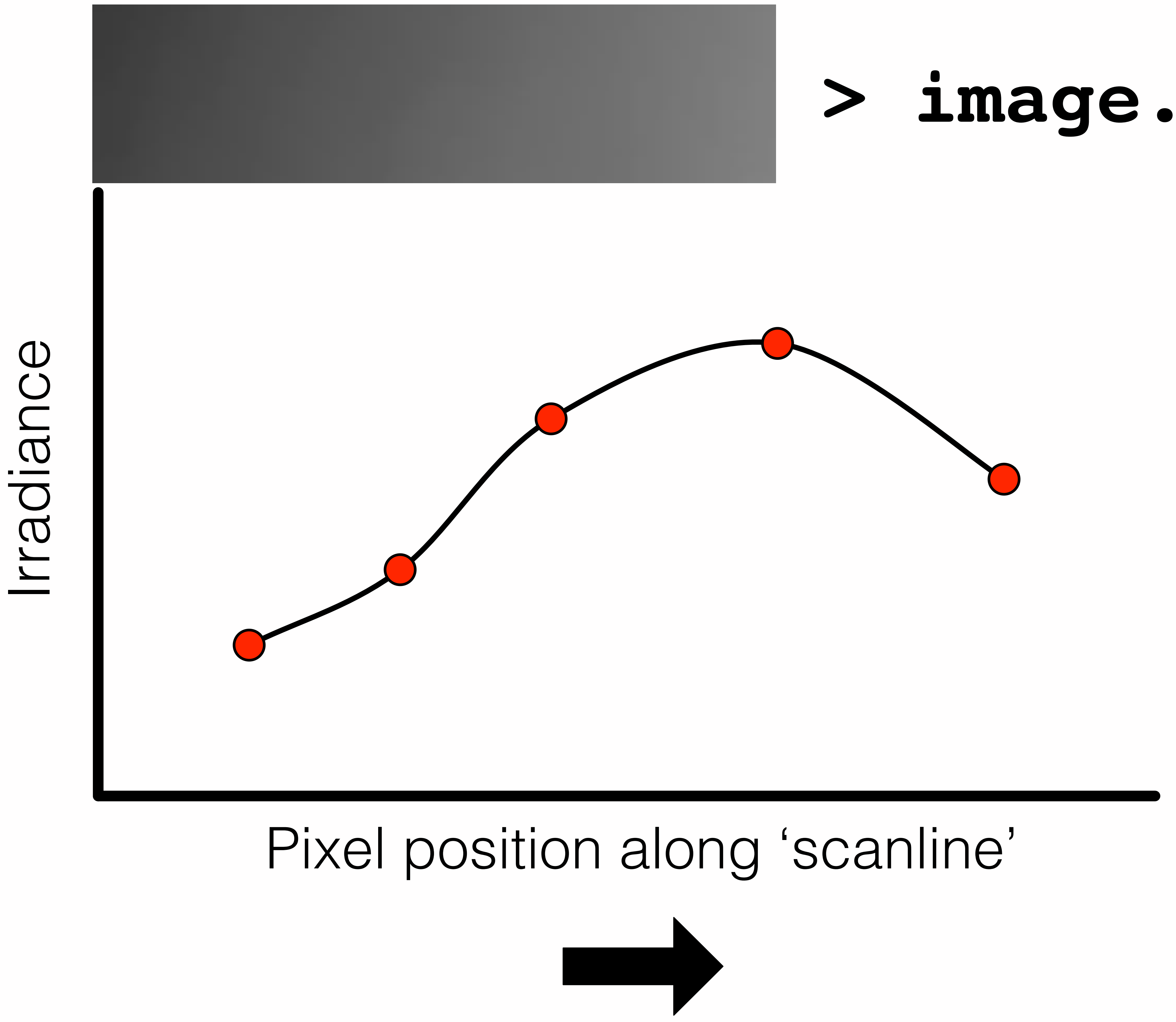
`-af keep.af`

`> /dev/null`



-af keep.af

> image.hdr



Why overture?

- In a ‘one-off’ simulation, *Radiance* has to sometimes use **extrapolation** to estimate values between sampling locations as it progresses from one sampling point to the next.

- With an overture calculation, the ambient file (aka ambient cache) is first populated with values. Thereby ensuring that - when reused to create an image - *Radiance* uses **interpolation** between already calculated values rather than less reliable extrapolation. Negligible overhead in overall computation time.

Parameter settings and CPU costs

```
% rtrace -defaults

-av 0.0 0.0 0.0          # ambient value
-aw 0                    # ambient value weight
-ab 0                    # ambient bounces
-aa 0.100000             # ambient accuracy
-ar 256                  # ambient resolution
-ad 1024                 # ambient divisions
-as 512                  # ambient super-samples
```

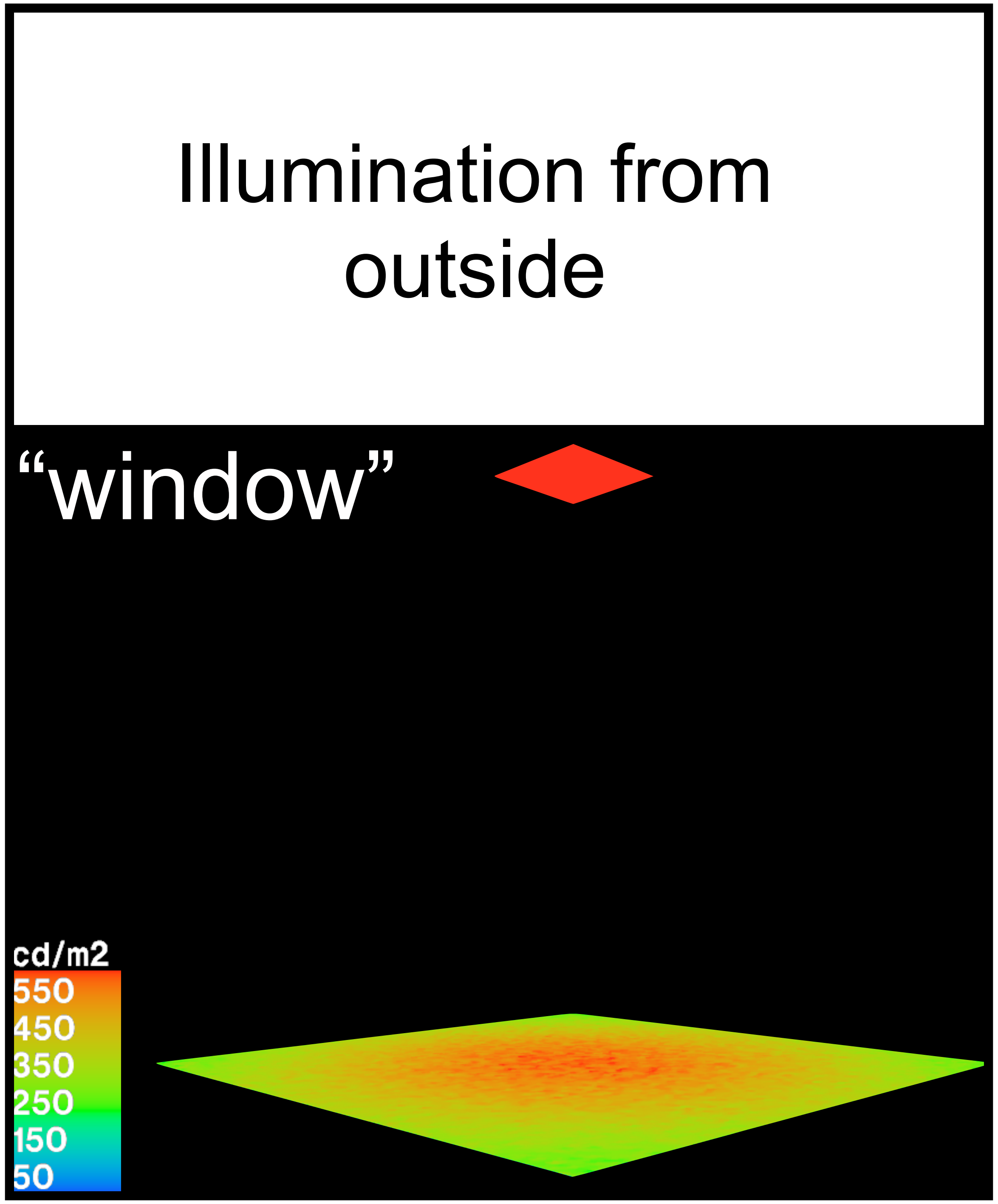
Parameter	Change	Potential CPU overhead
ad ambient divisions	512 to 1024 i.e. doubling	x 2
aa ambient accuracy	0.2 to 0.1 i.e. halving	x 4
	no interpolation 0	x a lot?
ar ambient resolution	32 to 64 i.e. doubling	x 4
	unlimited resolution 0	x a lot?

Parameter	**Change**	**Potential CPU overhead**
ad ambient divisions	512 to 1024 i.e. doubling	x 2
aa ambient accuracy	0.2 to 0.1 i.e. halving	x 4
	no interpolation 0	x a lot?
ar ambient resolution	32 to 64 i.e. doubling	x 4
	unlimited resolution 0	x a lot?

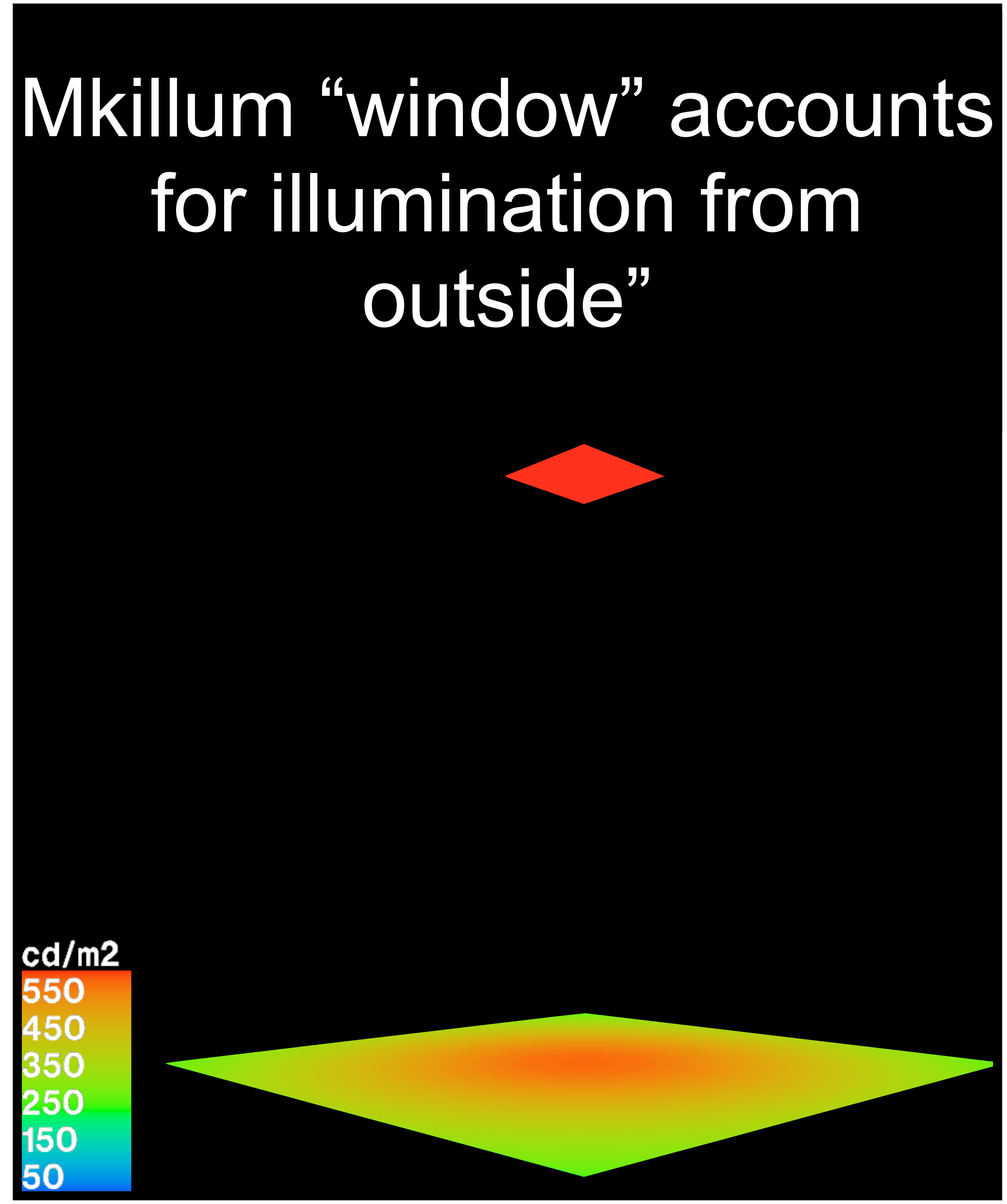
Parameter	Change	Potential CPU overhead
ad ambient divisions	512 to 1024 i.e. doubling	x 2
aa ambient accuracy	0.2 to 0.1 i.e. halving	x 4
	no interpolation 0	x a lot?
ar ambient resolution	32 to 64 i.e. doubling	x 4
	unlimited resolution 0	x a lot?

mkillum

mkillum - hunt <u>twice</u> to avoid having to search wide only to find *small* openings that lead to the light



Look ‘everywhere’ to find the light from the sky through window



Look first at the window because we know *a priori* that it is the ‘source’ of illumination

Step 1

- Create the octree as normal.
    - It is important for the **mkillum** process that follows to be able to identify the windows that need to be treated.
- Use **mkillum** to compute the <u>*window output distribution*</u> i.e. a similar specification to that used to characterise the light output distribution of a luminaire. Ambient settings as required.
    - A new window is created using the **illum** material.

Step 2

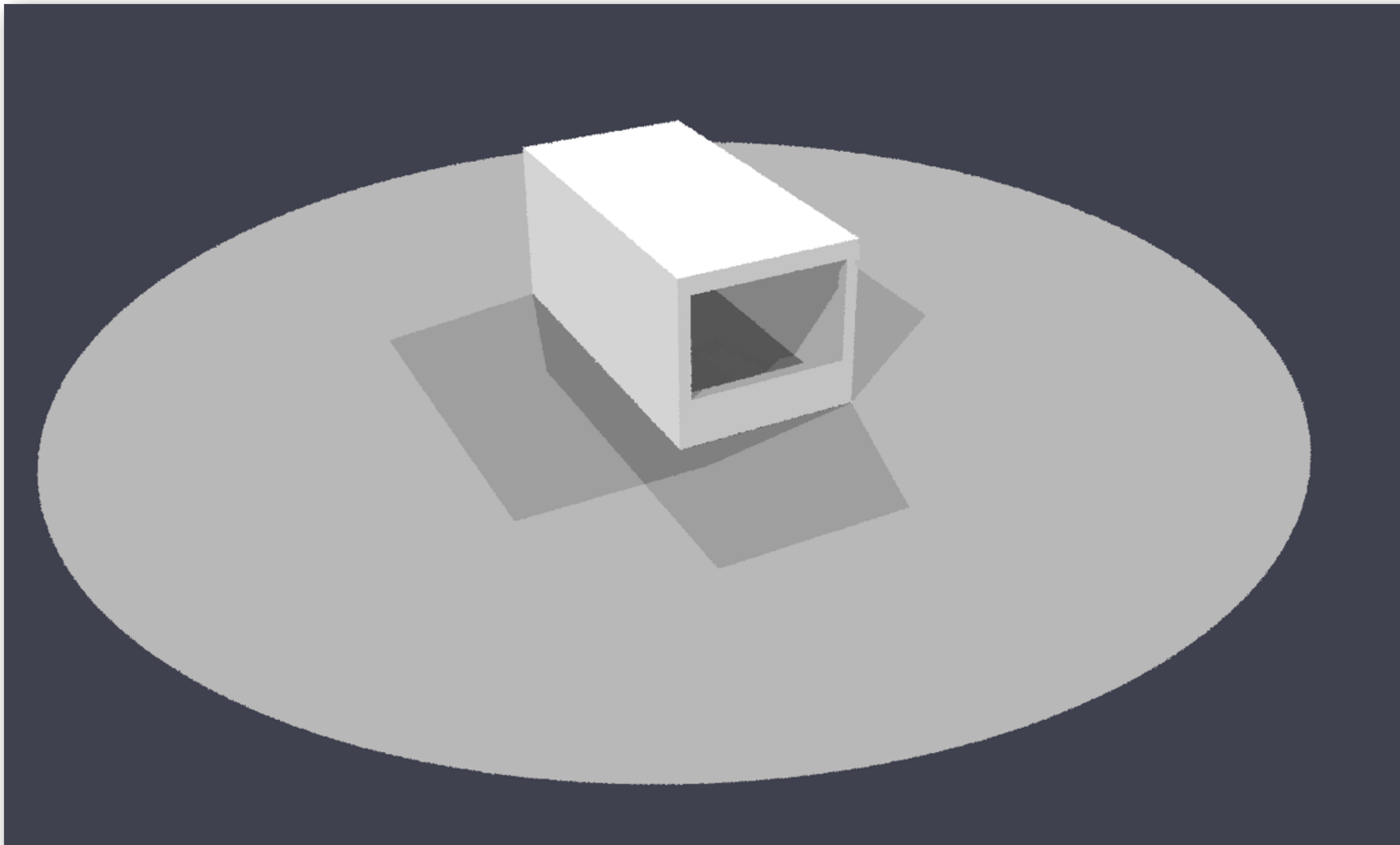
- Recreate the octree replacing the window with the new description created by **mkillum**.
  - Replace **`window.rad`** with **`mkiwin.rad`**.
- Run **rpict** or **rtrace** on the new octree with ambient settings as required.

```
oconv room.rad window.rad sky.rad \
  out.rad > scene.oct
```

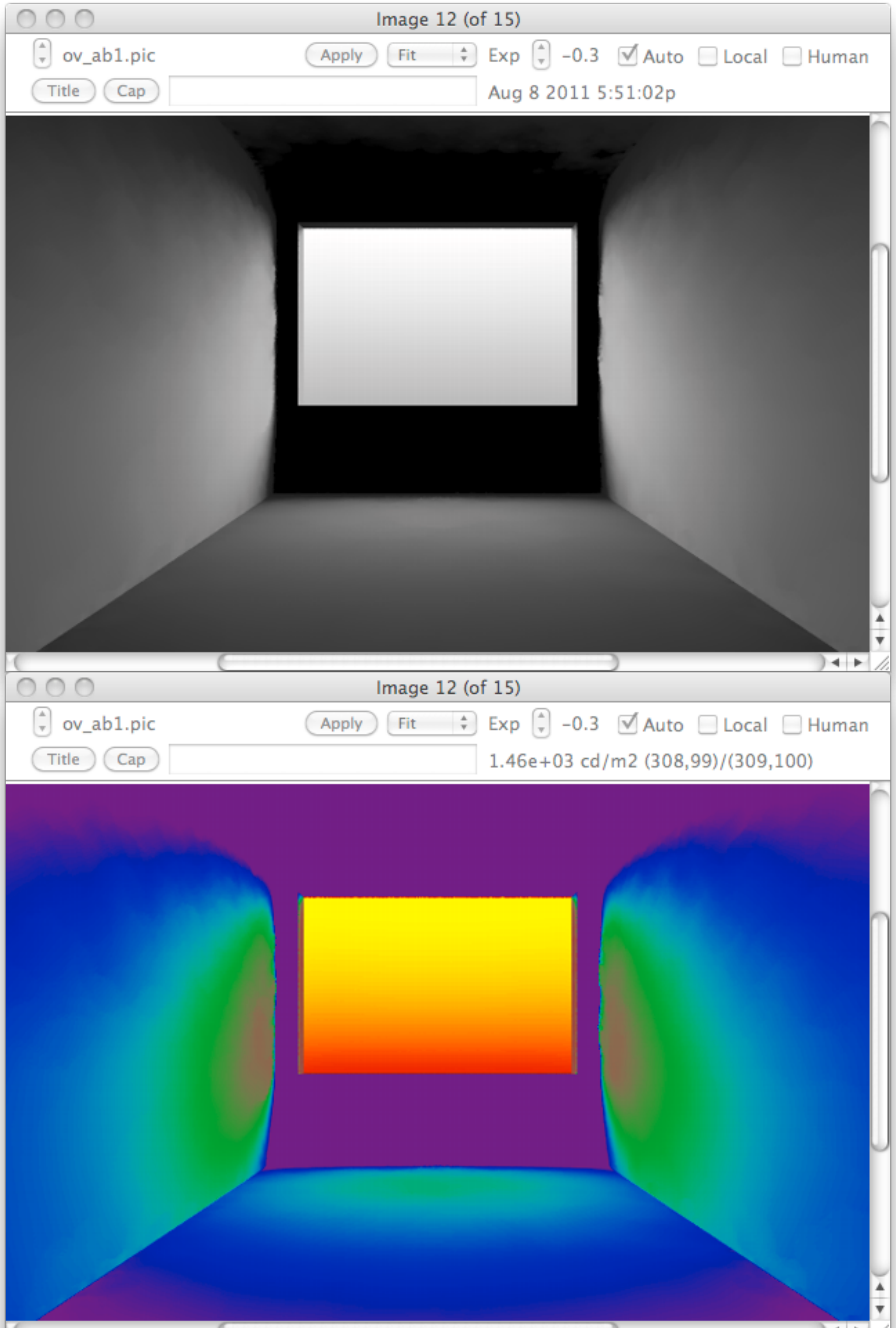

```
mkillum [options] scene.oct < window.rad > \
  mkiwin.rad
```

```
oconv office.rad mkiwin.rad sky.rad \
  out.rad >  mkiscene.oct
```

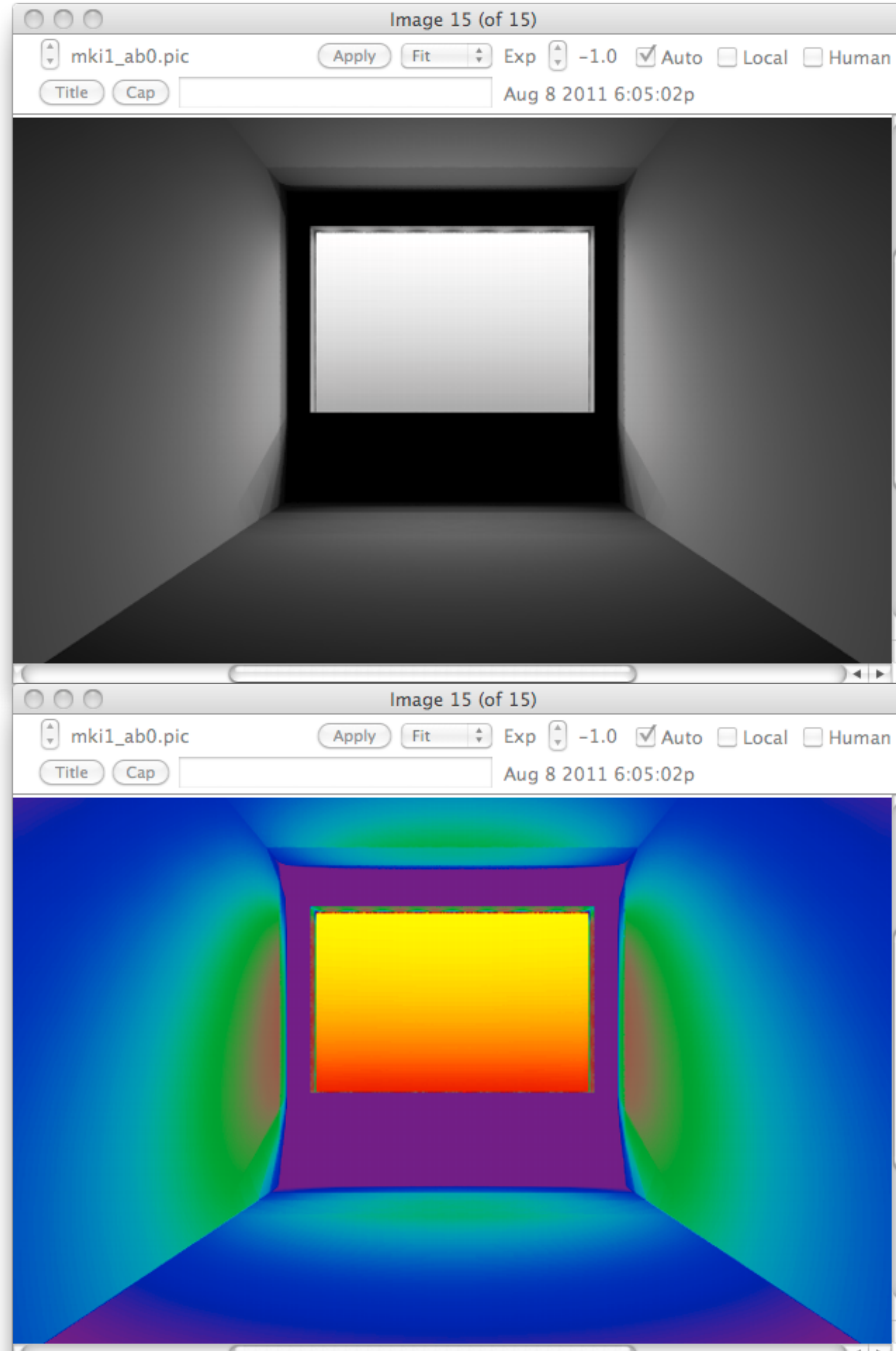

```
rpict / rtrace [options] mkiscene.oct
```



ab 1



MKI ab 1 ; ab 0



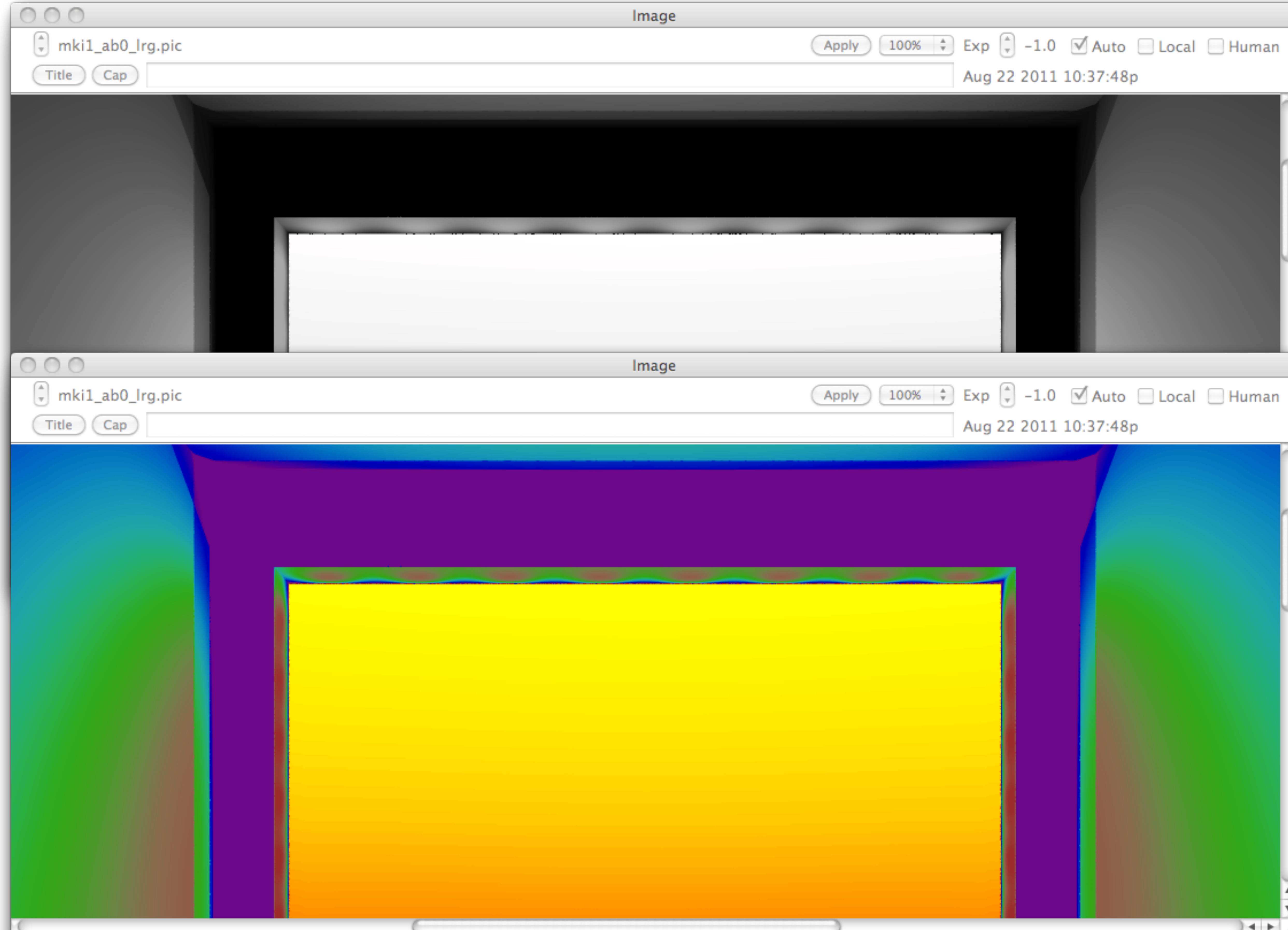
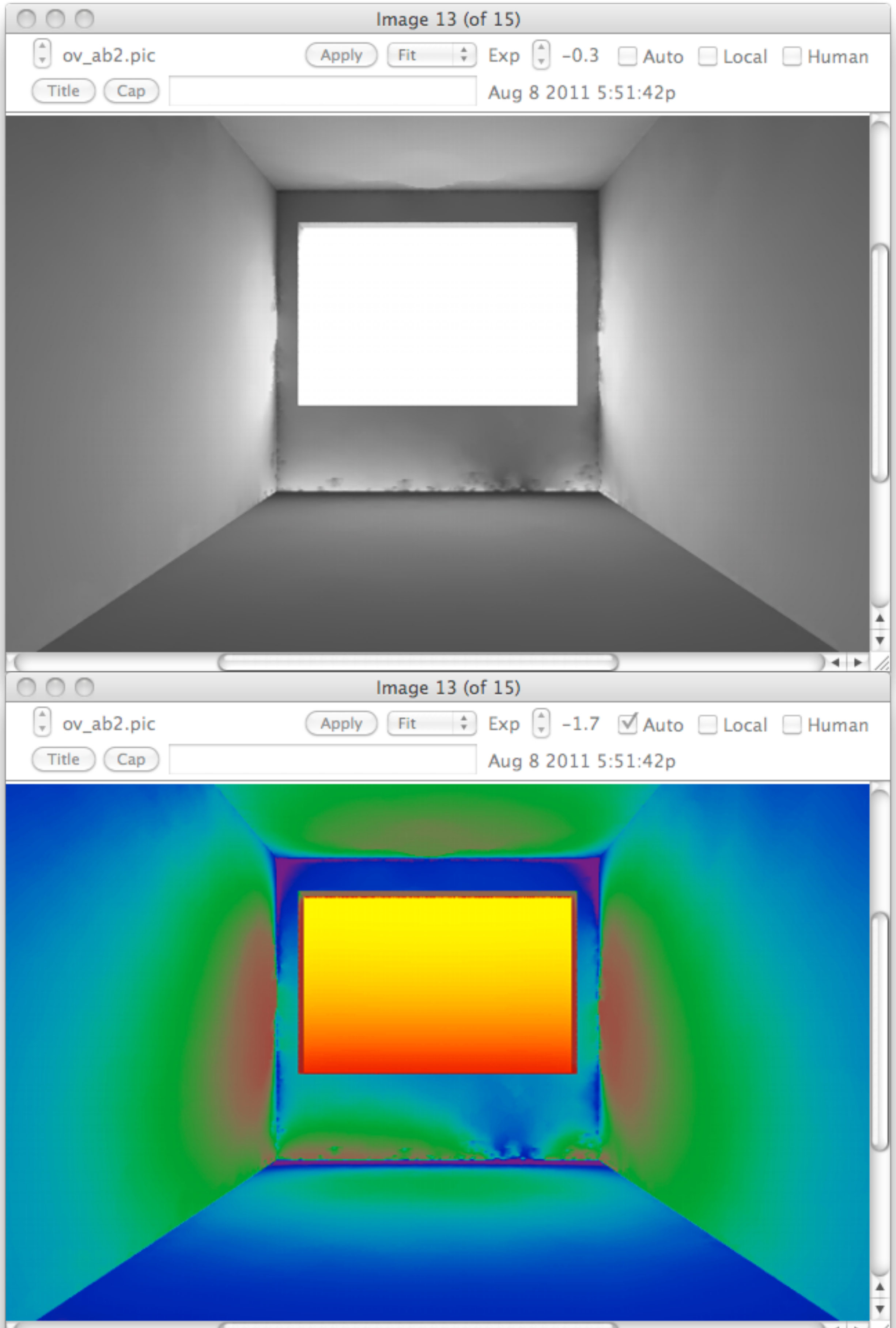
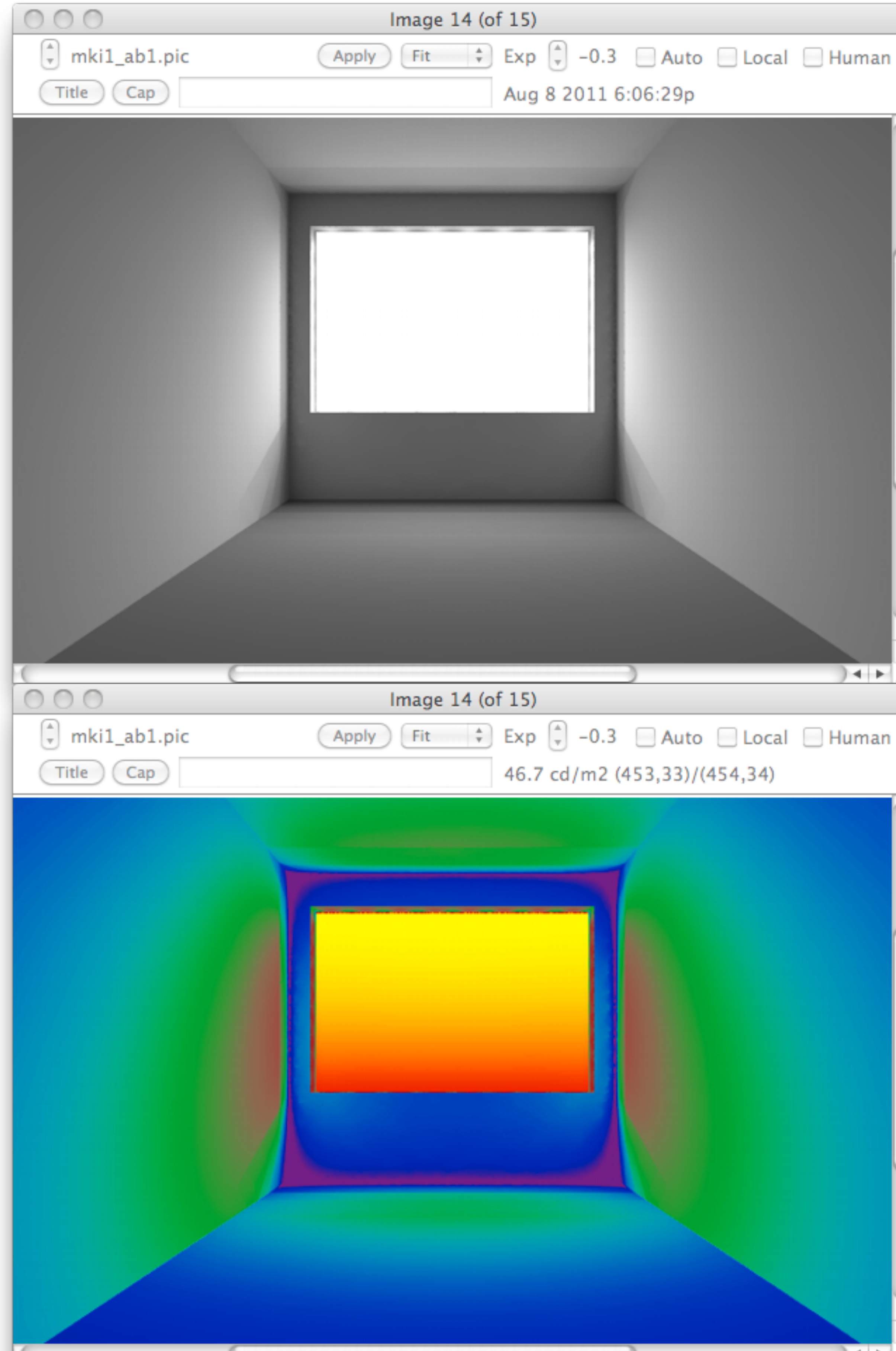


Image
mki1_ab0_lrg.pic
Apply
100%
Exp
-1.0
Auto
Local
Human
Title
Cap
Aug 22 2011 10:37:48p
Image
mki1_ab0_lrg.pic
Apply
100%
Exp
-1.0
Auto
Local
Human
Title
Cap
Aug 22 2011 10:37:48p

ab 2



MKI ab 1 ; ab 1



Issues with **mkillum**

- Many windows can results in too many light sources.
- Nearby external obstructions - subdivide window.
- CAD input - rectangles, surface normals.

Modelling venetian blinds using **mkillum**

Fig 13.4 Rendering with Radiance

A five-sided **illum** box <u>encloses</u> the blinds on the inside

Cases where the **mkillum** approach doesn't work

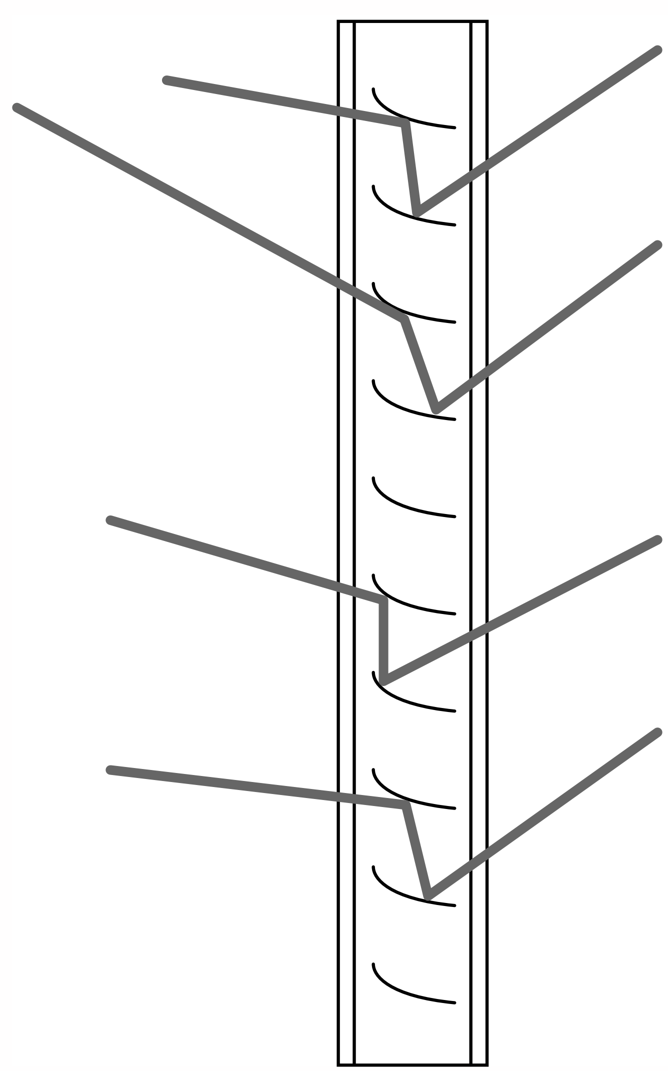
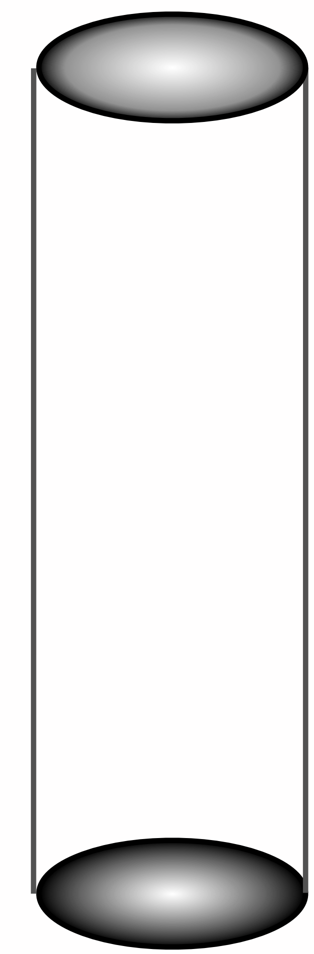


Fig 13.8 Rendering with Radiance



Curved mirror louvres

Light pipes

Questions?

Thank you

e-mail: John Mardaljevic

links: Staff page

CBDM

Professor of Building Daylight Modelling
School of Civil & Building Engineering
Loughborough University, UK