Distribution Ray Tracing

Reading

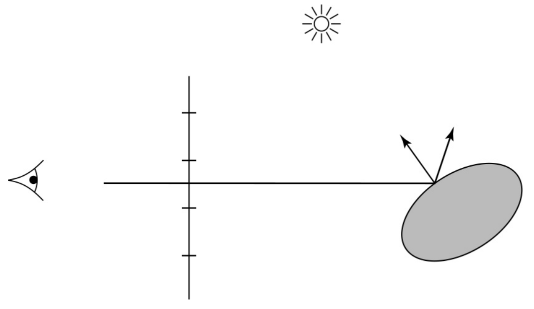
Required:

- Shirley, section 10.11

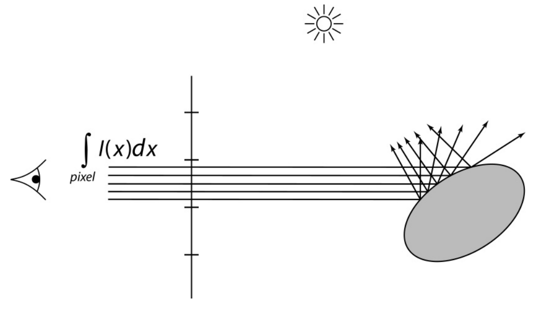
Further reading:

- Watt, sections 10.4-10.5
- A. Glassner. An Introduction to Ray Tracing. Academic Press, 1989. [In the lab.]
- Robert L. Cook, Thomas Porter, Loren Carpenter. "Distributed Ray Tracing." Computer Graphics (Proceedings of SIGGRAPH 84). *18 (3)*. pp. 137-145. 1984.
- James T. Kajiya. "The Rendering Equation." Computer Graphics (Proceedings of SIGGRAPH 86). *20 (4)*. pp. 143-150. 1986.

Pixel anti-aliasing



No anti-aliasing



Pixel anti-aliasing

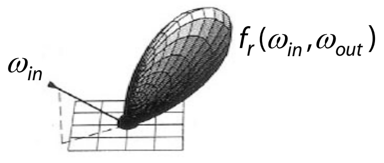
BRDF, revisited

The reflection model on the previous slide assumes that inter-reflection behaves in a mirror-like fashion.

Recall that we could view light reflection in terms of the general **Bi-directional Reflectance Distribution Function** (**BRDF**):

$$f_r(\omega_{in}, \omega_{out})$$

Which we could visualize for a given ω_{in}:



Surface reflection equation

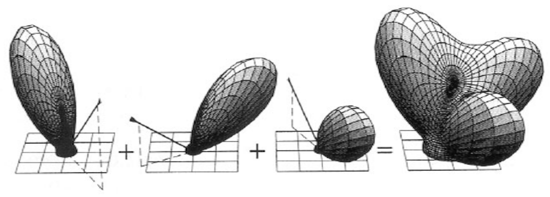
To compute the reflection from a real surface, we would actually need to solve the **surface reflection equation**:

$$I(\omega_{out}) = \int_H I(\omega_{in}) f_r(\omega_{in}, \omega_{out}) d\omega_{in}$$

For a directional light with intensity L_1 coming from direction direction, ω_1, we can view the remaining directions as contributing zero, giving:

$$I(\omega_{out}) = L_1 f_r(\omega_1, \omega_{out})$$

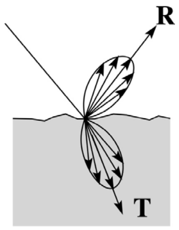
We can plot the reflected light as a function of viewing angle for multiple light source contributions:



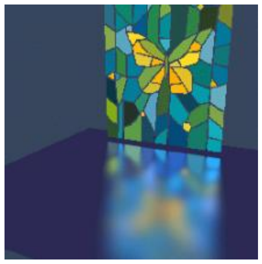
Simulating gloss and translucency

The mirror-like form of reflection, when used to approximate glossy surfaces, introduces a kind of aliasing, because we are undersampling reflection (and refraction).

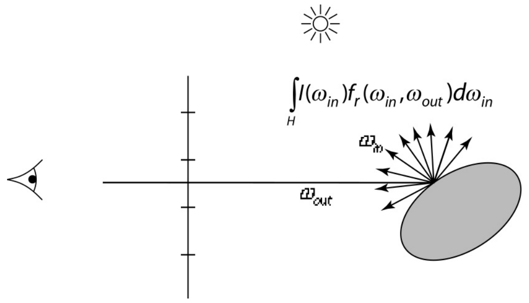
For example:



Distributing rays over reflection directions gives:

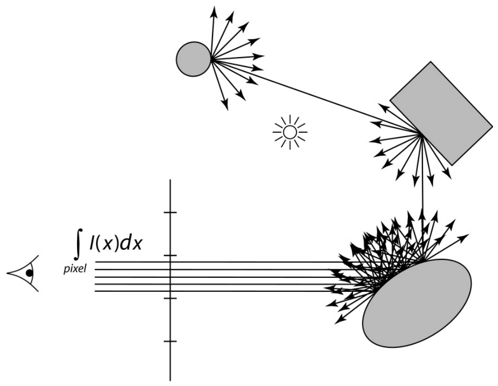


Reflection anti-aliasing


$\int_H I(\omega_{in}) f_r(\omega_{in}, \omega_{out}) d\omega_{in}$
ω_{in}
ω_{out}

Reflection anti-aliasing

Full anti-aliasing



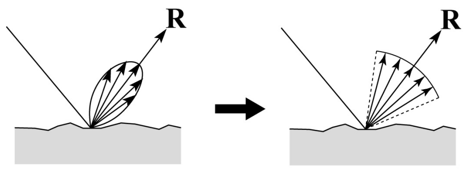
Full anti-aliasing…lots of nested integrals!

Computing these integrals is prohibitively expensive, especially after following the rays recursively.

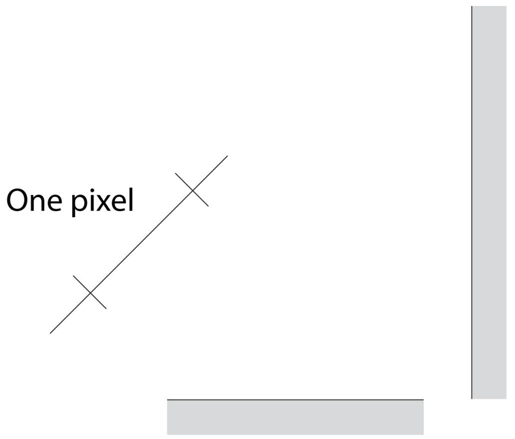
We'll look at ways to approximate high-dimensional integrals…

Glossy reflection revisited

Let' return to the glossy reflection model, and modify it – for purposes of illustration – as follows:

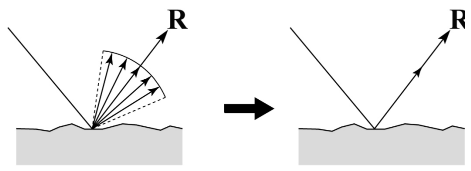


We can visualize the span of rays we want to integrate over, within a pixel:

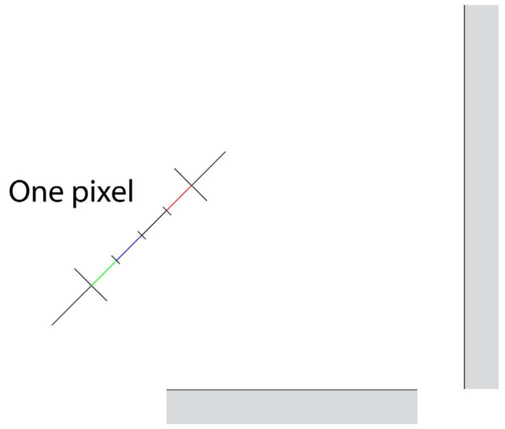


Whitted ray tracing

Returning to the reflection example, Whitted ray tracing replaces the glossy reflection with mirror reflection:

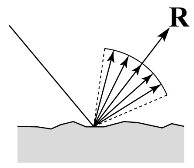


Thus, we render with anti-aliasing as follows:

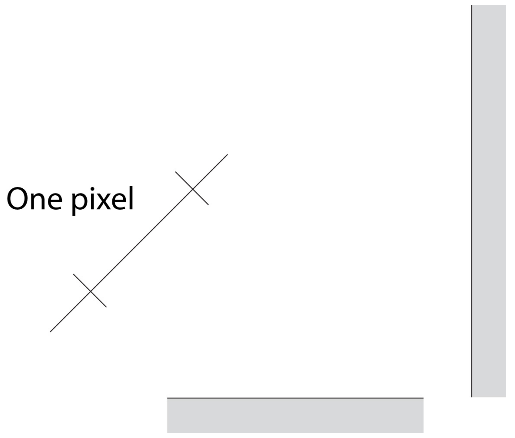


Monte Carlo path tracing

Let' return to the original glossy reflection model:

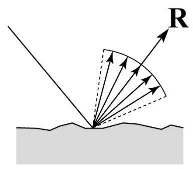


An alternative way to follow rays is by making random decisions along the way – a.k.a., Monte Carlo path tracing

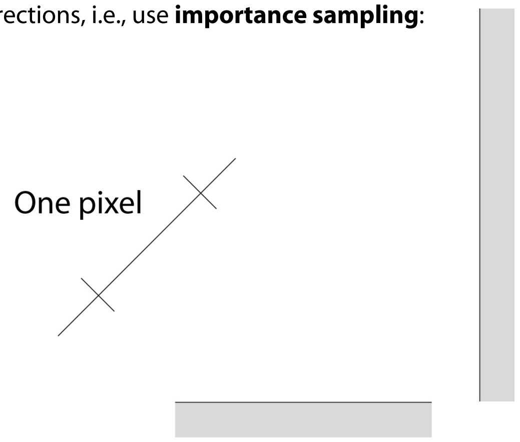


Importance sampling

The problem is that lots of samples are "wasted." Using again the glossy reflection model:

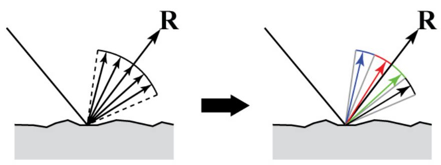


Let's now randomly choose rays, but according to a probability that favors more important reflection directions, i.e., use **importance sampling**:

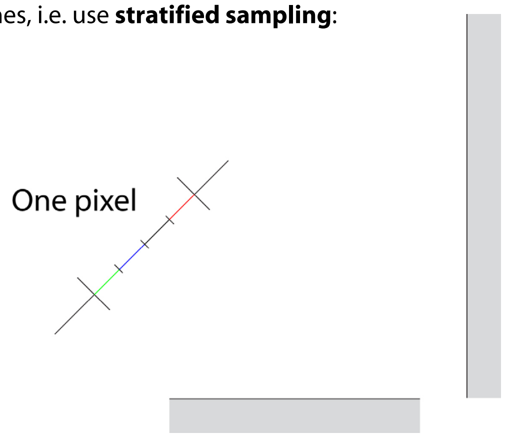


Stratified sampling

We still have a problem that rays may be clumped together. Let's simplify the reflection model some and split it into zones:

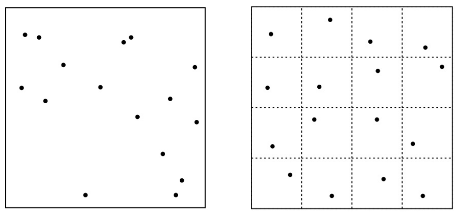


Now let's restrict our randomness to within these zones, i.e. use **stratified sampling**:



Stratified sampling of a 2D pixel

Here we see random (pure Monte Carlo) vs. stratified sampling over a 2D pixel (here 16 rays/pixel):



The stratified pattern on the right is also sometimes called a **jittered** sampling pattern.

One interesting side effect of these stochastic sampling patterns is that they actually injects noise into the solution (slightly grainier images). This noise tends to be less objectionable than aliasing artifacts.

Distribution ray tracing

Stratified, Monte Carlo importance sampling has another name, **distribution ray tracing** [Cook84]:

- uses non-uniform (jittered) samples.
- replaces aliasing artifacts with noise.
- provides additional effects by distributing rays to sample:
  - Reflections and refractions
  - Light source area
  - Camera lens area
  - Time

[Originally called "distributed ray tracing," but we will call it distribution ray tracing so as not to confuse with parallel computing.]

DRT pseudocode

TraceImage() looks basically the same, except now each pixel records the average color of jittered sub-pixel rays.

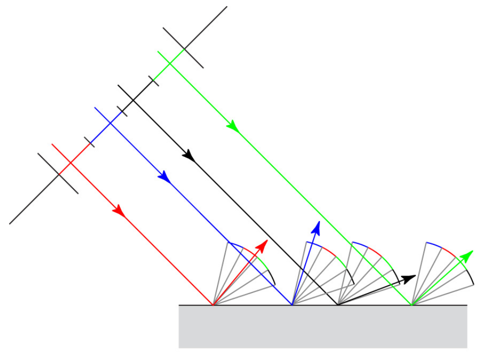
```
function traceImage (scene):
    for each pixel (i, j) in image do
        I(i, j) ← 0
        for each sub-pixel id in (i,j) do
            s ← pixelToWorld(jitter(i, j, id))
            p ← COP
            d ←(s - p).normalize()
            I(i, j) ← I(i, j) + traceRay(scene, p, d, id)
        end for
        I(i, j) ← I(i, j)/numSubPixels
    end for
end function
```

A typical choice is numSubPixels = 5*5.

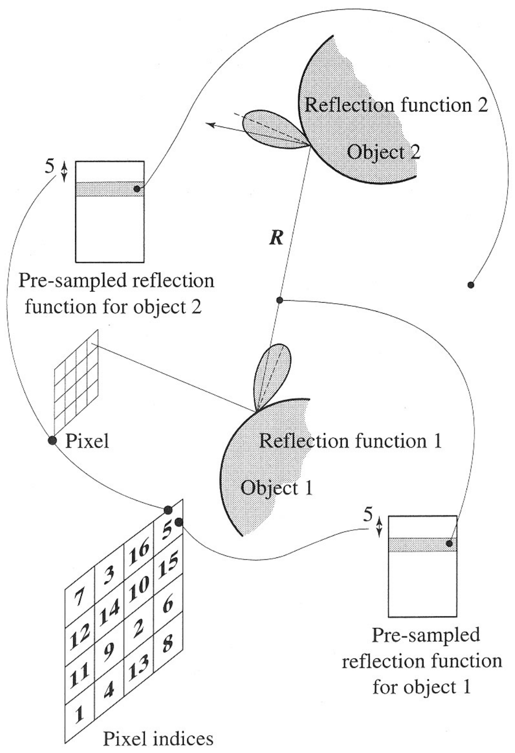
DRT pseudocode (cont'd)

Now consider *traceRay*(), modified to handle (only) opaque glossy surfaces:

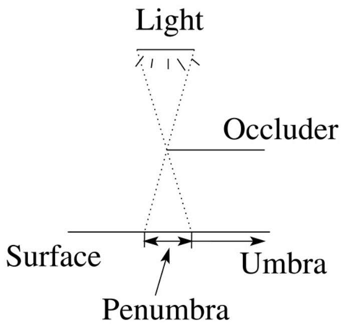
```
function traceRay(scene, p, d, id):
    (q, N, material) ← intersect (scene, p, d)
    I ← shade(…)
    R ← jitteredReflectDirection(material, N, -d, id)
    I ← I + material.$k_r$ ∗ traceRay(scene, q, R, id)
    return I
end function
```



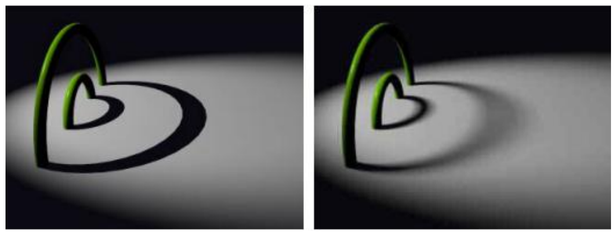
Pre-sampling glossy reflections



Soft shadows

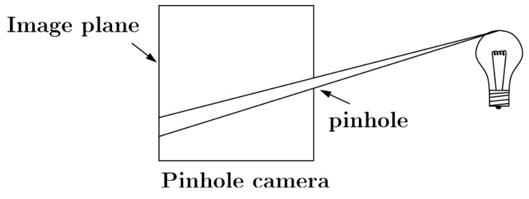


Distributing rays over light source area gives:

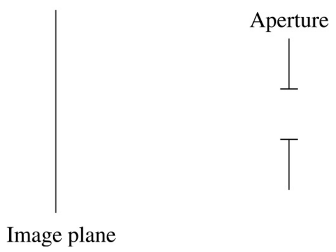


The pinhole camera, revisited

Recall the pinhole camera:



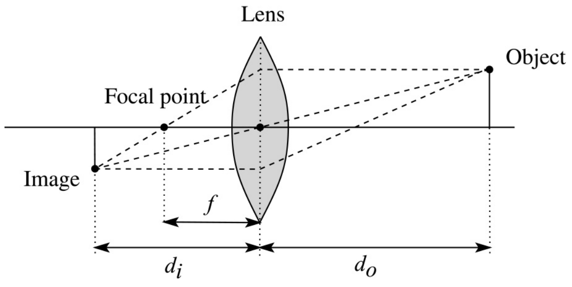
Q: How can we simulate a pinhole camera more accurately?



Lenses

Pinhole cameras in the real world require small apertures to keep the image in focus.

Lenses focus a bundle of rays to one point => can have larger aperture.



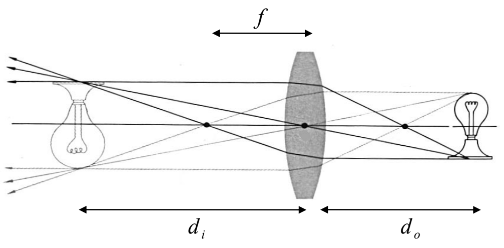
For a "thin" lens, we can approximately calculate where an object point will be in focus using the the Gaussian lens formula:

$$\frac{1}{d_i} + \frac{1}{d_o} = \frac{1}{f}$$

where f is the **focal length** of the lens.

Lenses (cont'd)

An image is formed of the whole object by collecting bundles of rays from every point on the object:

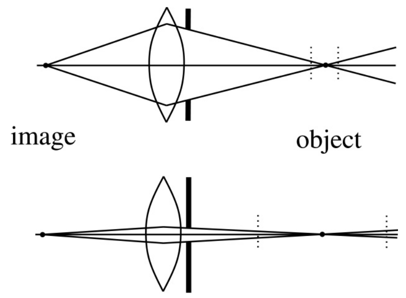


Depth of field

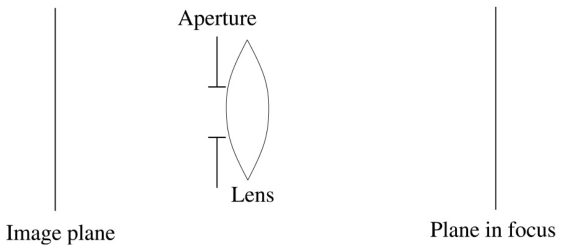
Lenses do have some limitations.

The most noticeable is the fact that points that are not in the object plane will appear out of focus.

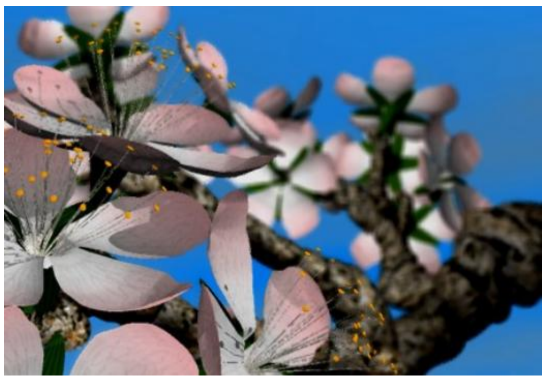
The **depth of field** is a measure of how far from the object plane points can be before appearing "too blurry."



Simulating depth of field

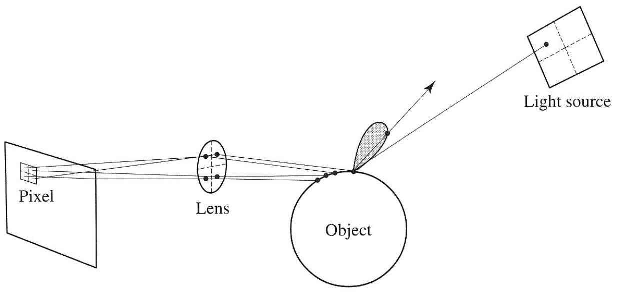


Distributing rays over a finite aperture gives:



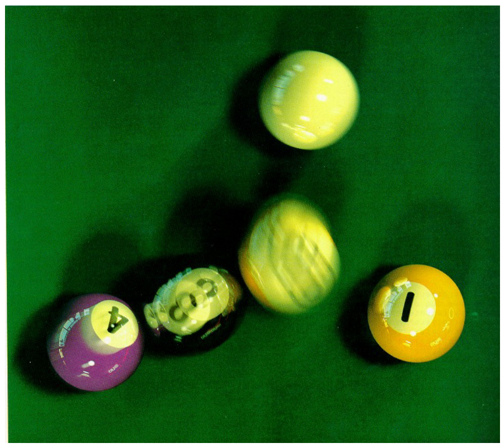
Chaining the ray id's

In general, you can trace rays through a scene and keep track of their id's to handle *all* of these effects:



DRT to simulate ___________________

Distributing rays over time gives:



Summary

What to take home from this lecture:

1. The limitations of Whitted ray tracing.
2. How distribution ray tracing works and what effects it can simulate.