11. Ray Tracing

Reading

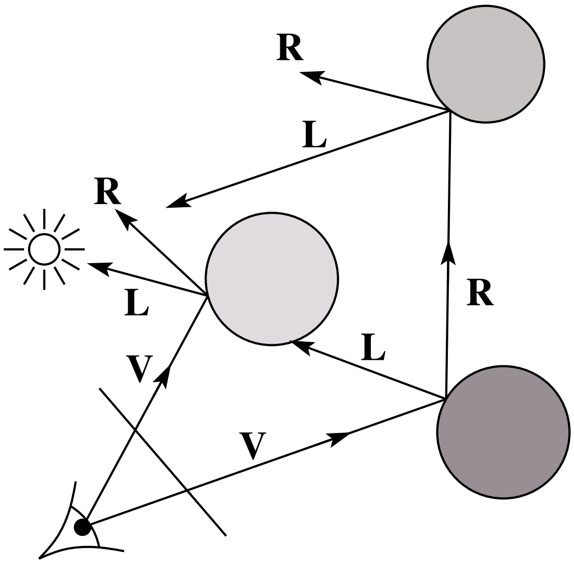
Recommended:

- Hearn & Baker, Section 14.6

Optional:

- A. Glassner. An Introduction to Ray Tracing. Academic Press, 1989.
- T. Whitted. An improved illumination model for shaded display. *Communications of the ACM* 23(6), 343-349, 1980.

Introduction



Idea: "Ray tracing" is like "ray casting," only you continue to bounce the ray around the scene.

Provides:

- Visible-surface detection
- Shadow effects
- Inter-object reflections
- Transparency
- Multiple light source illumination

Overview of lecture

1. Basic ray-tracing algorithm

2. Ray–surface intersection calculations

    · Spheres

    · Polyhedra

3. Acceleration techniques

    · Hierarchical bounding volumes

    · Space-subdivision methods

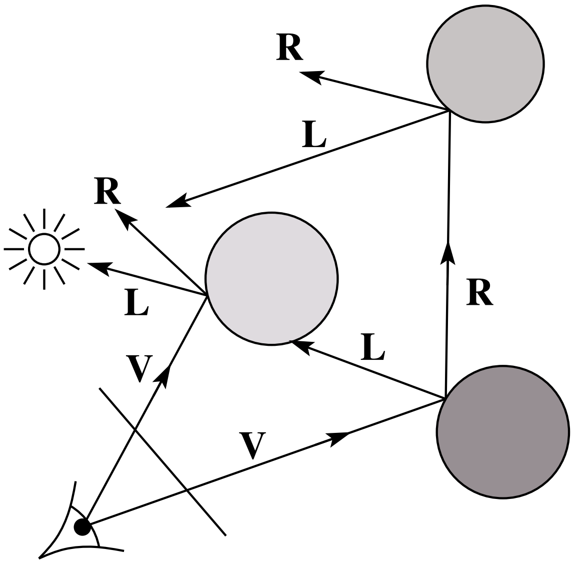
4. Antialiasing

    · Supersampling

    · Adaptive sampling

    · Distributed ray tracing

Ray-tracing algorithm



Algorithm:

1. For each pixel, trace a "primary ray" to the first visible surface (same as "ray casting").

2. For each intersection, trace "secondary rays":

   · "Shadow rays" in directions $\mathbf{L}_i$ to light sources.

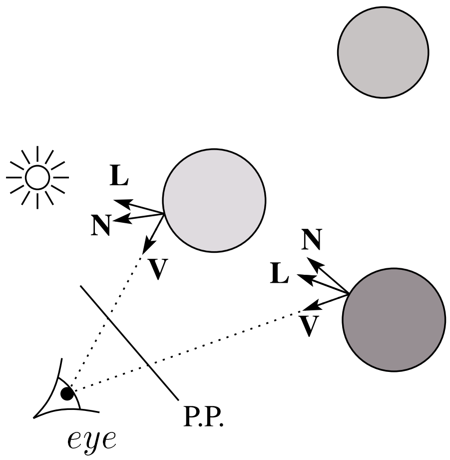
   · "Reflected ray" in direction $\mathbf{R}$.

   · "Refracted ray" in direction $\mathbf{T}$.

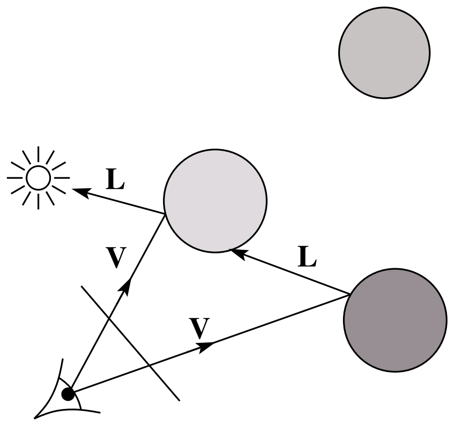
Note: this is sometimes called "backward ray tracing" because we are tracing rays from the viewer to the lights.

Ray-tracing algorithm, cont’d

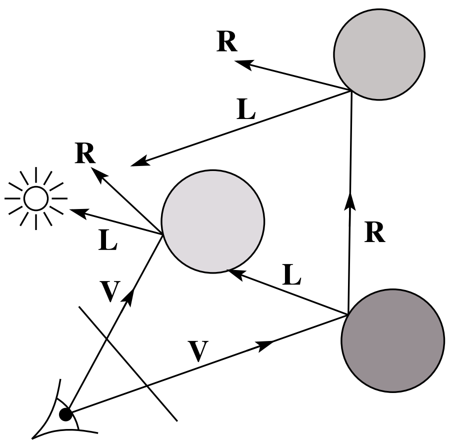
Let’s look at this in stages:



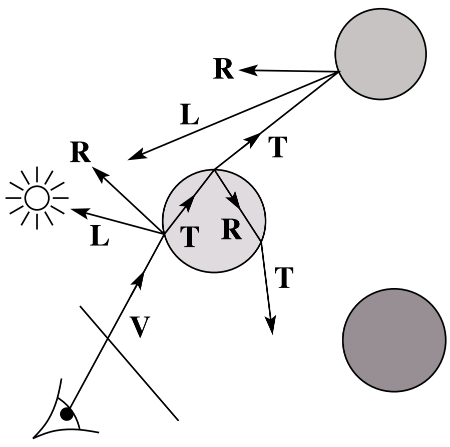
Ray casting



Shadow rays

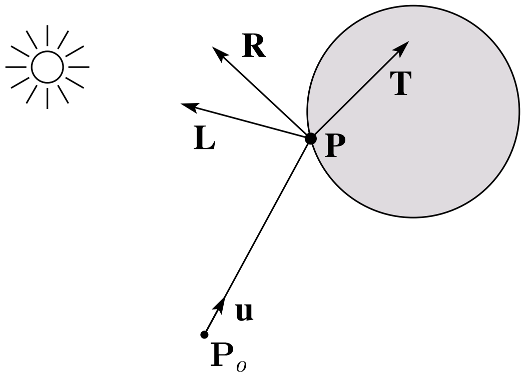


Reflection rays



Refracted rays

Shading



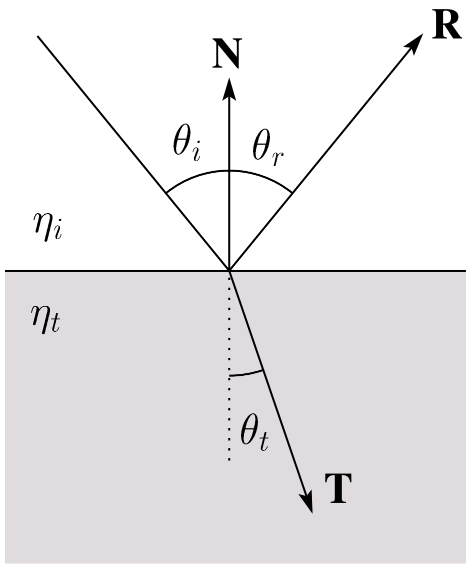
Let $I(\mathbf{P}_o, \mathbf{u})$ be the intensity seen from point $\mathbf{P}_o$ along direction $\mathbf{u}$:

$$I(\mathbf{P}_o, \mathbf{u}) \;=\; I_{\text{direct}} \;+\; I_{\text{reflected}} \;+\; I_{\text{transmitted}}$$

where

- I_{direct} is computed from the Phong model
- $I_{\text{reflected}} \;=\; k_{\text{reflected}}\, I(\mathbf{P}, \mathbf{R})$
- $I_{\text{transmitted}} \;=\; k_{\text{transmitted}}\, I(\mathbf{P}, \mathbf{T})$

Reflected and transmitted directions



Law of reflection:

$$\theta_i = \theta_r$$

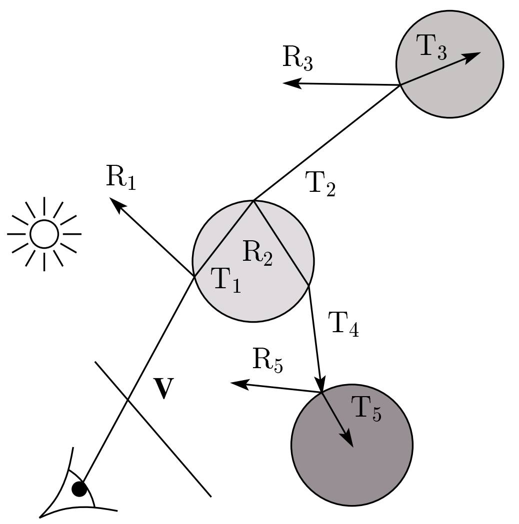
Snell's law of refraction:

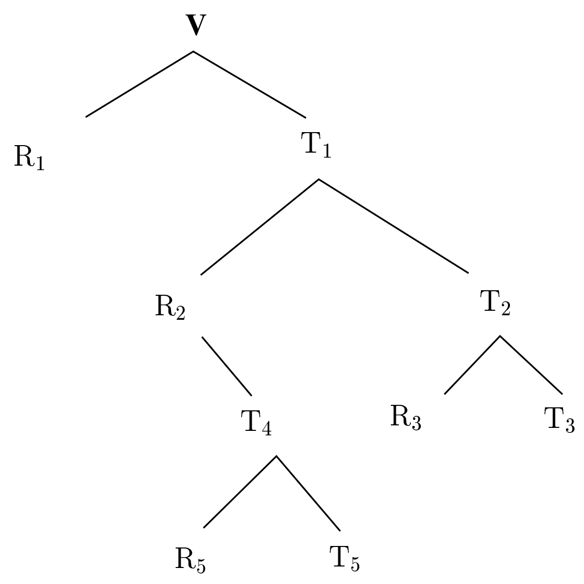
$$\eta_i \sin\theta_i = \eta_t \sin\theta_t$$

where η_i, η_t are "indices of refraction."

The ray-tracing tree

As the recursively traced rays ricochet through the scene, they are added to a “ray-tracing tree”:





Ray-tracing pseudocode

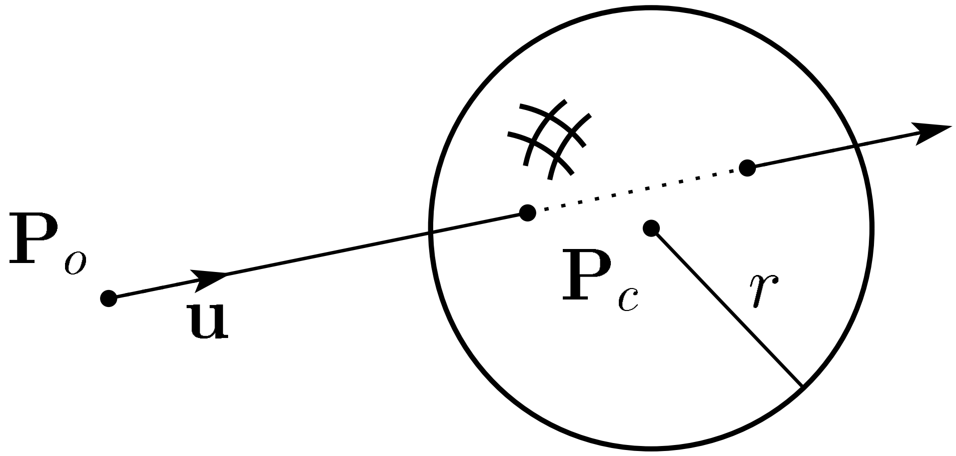
function $RayTrace(\mathbf{P}_o, \mathbf{u})$:
    $(\mathbf{P}, Obj) \leftarrow RayCast(\mathbf{P}_o, \mathbf{u}))$
    $I \leftarrow Obj.k_a * I_a$
    for each light source ℓ **do**:
        $(\mathbf{P'}, LightObj) \leftarrow RayCast(\mathbf{P}, Dir(\mathbf{P}, \ell))$
        if $LightObj = \ell$ **then**:
            $I \leftarrow I + \langle\text{diffuse term}\rangle + \langle\text{spec term}\rangle$
        end if
    end for
    $I \leftarrow I + Obj.k_{\text{reflected}} * RayTrace(\mathbf{P}, \mathbf{R})$
    $I \leftarrow I + Obj.k_{\text{transmitted}} * RayTrace(\mathbf{P}, \mathbf{T})$
    return I
end function

Terminating recursion

Q: How do you bottom out of recursive ray tracing?

Possibilities:

Intersecting rays with spheres



Given:

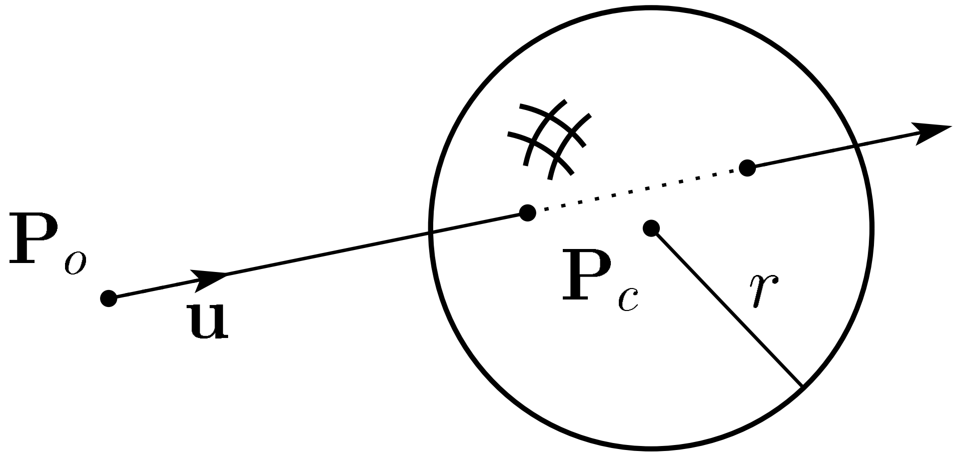
- A ray $\mathbf{P}(s)$ through initial position $\mathbf{P}_0$ in direction $\mathbf{u}$ (a unit vector):

$$\mathbf{P}(s) \;=\; \mathbf{P}_0 \;+\; s\,\mathbf{u}$$

- A sphere S centered at $\mathbf{P}_c$ with radius r

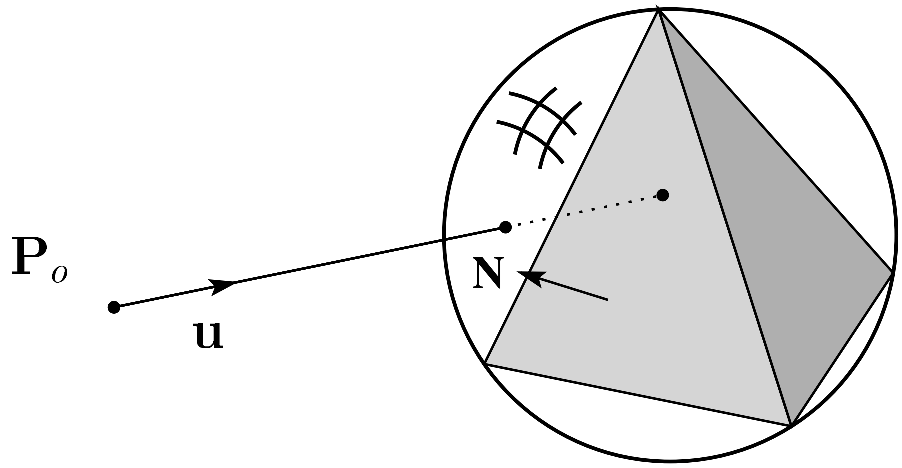
Find: The intersection of $\mathbf{P}(s)$ with S.

Intersecting rays with spheres



Solution:

Intersecting rays with polyhedra



To intersect a ray with a polyhedron:

1. Test intersection of ray with bounding sphere.
2. Locate the "front-facing" faces of the polyhedron with

$$\mathbf{u} \cdot \mathbf{N} \ < \ 0$$

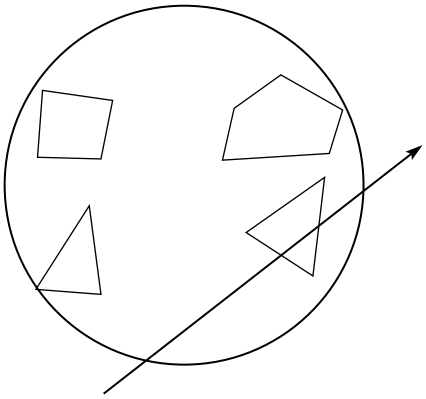
3. Intersect the ray with each front face's supporting plane.
4. Use a point-in-polygon test to see if the ray is inside the face.
5. Sort intersections according to smallest s.

Acceleration: hierarchical bounding volumes

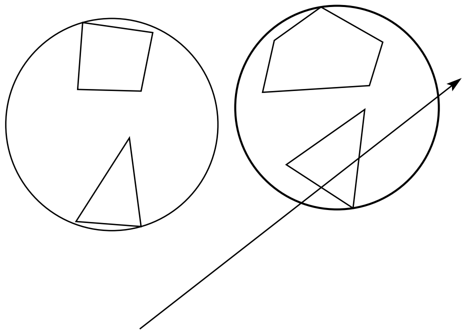
Vanilla ray tracing is really slow!

In practice, some acceleration technique is almost always used.

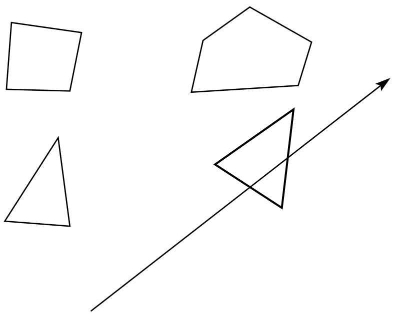
One approach is to use “hierarchical bounding volumes.”



Intersect with largest B.V...



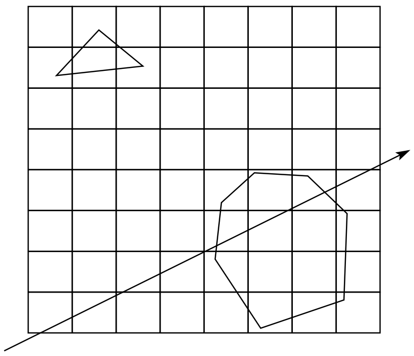
...then intersect with children...



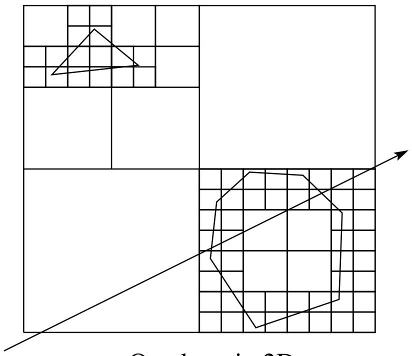
...until you reach the leaf nodes - the primitives.

Acceleration: spatial subdivision

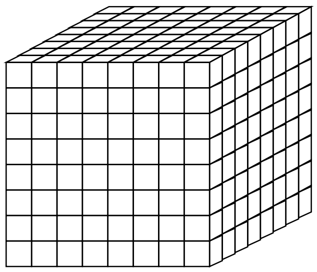
Another approach is "spatial subdivision."



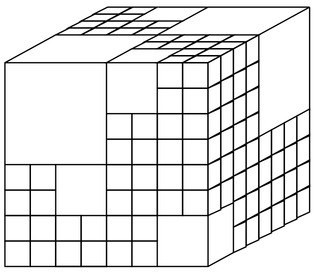
Uniform subdivion in 2D



Quadtree in 2D



Uniform subdivion in 3D



Octree in 3D

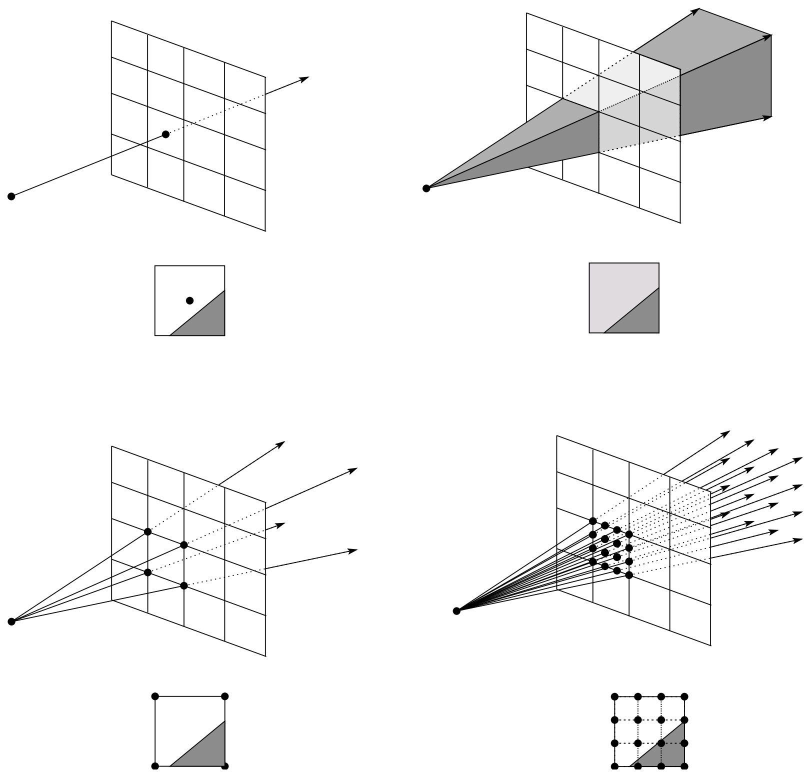
Idea:

- Partition objects spatially.
- Trace ray through voxel array.

Partition can be uniform or adaptive (e.g., octrees).

Antialiasing

When casting one ray per pixel, we are likely to have aliasing artifacts.



We would like to compute the average intensity in the neighborhood of each pixel.

To approximate this, we can cast more than one ray per pixel and average the result.

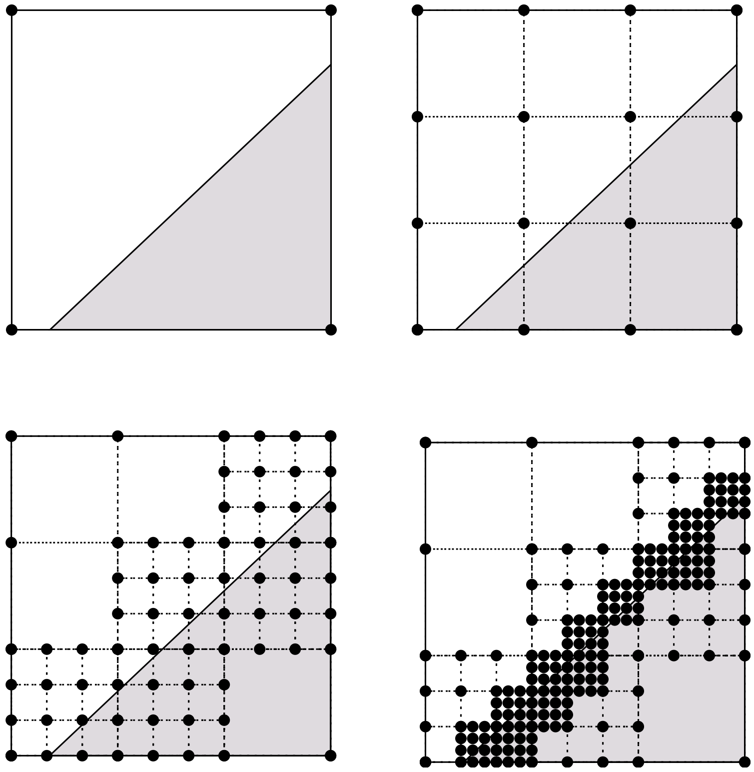
This is called "super-sampling and averaging down."

Antialiasing by adaptive sampling

Casting many rays per pixel can be unnecessarily costly.

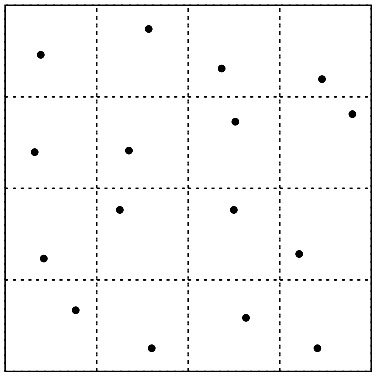
For example, if their are no rapid changes in intensity at the pixel, maybe only a few samples are needed.

Solution: adaptive sampling.



Q: When do we decide to cast more rays in a particular area?

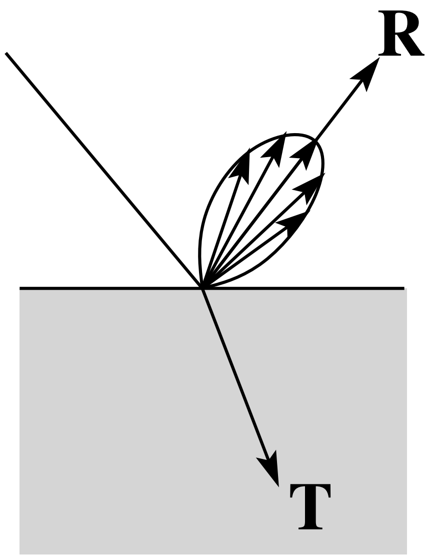
Distributed ray tracing



Idea:

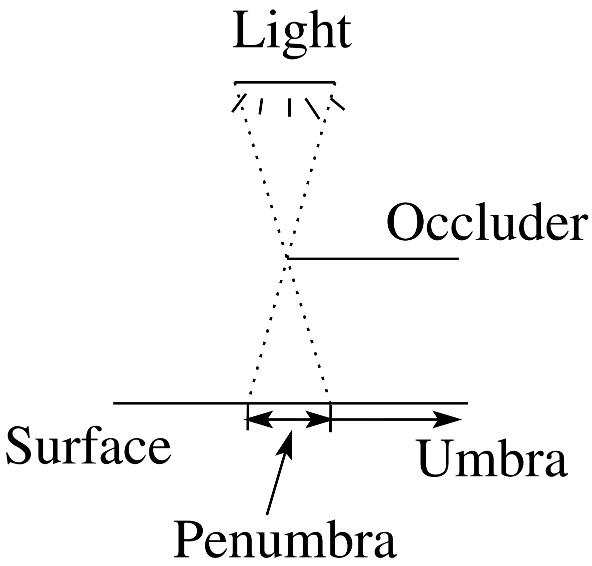
- Use non-uniform (jittered) samples.
- Replaces aliasing artifacts with noise.
- Provides additional effects if we distribute rays in other dimensions:
  - Reflection and refractions
  - Light source area
  - Camera lens area
  - Time

Distributed ray tracing, cont.



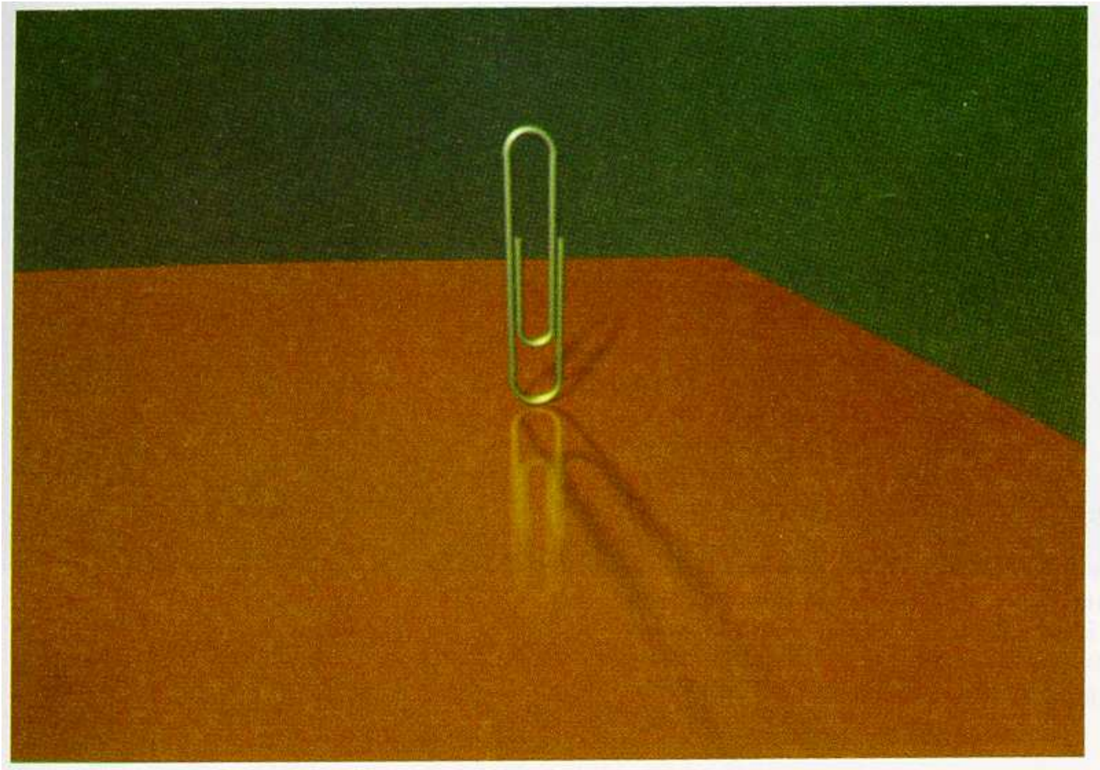
Distributing rays over reflection and refraction directions gives:

Distributed ray tracing, cont.

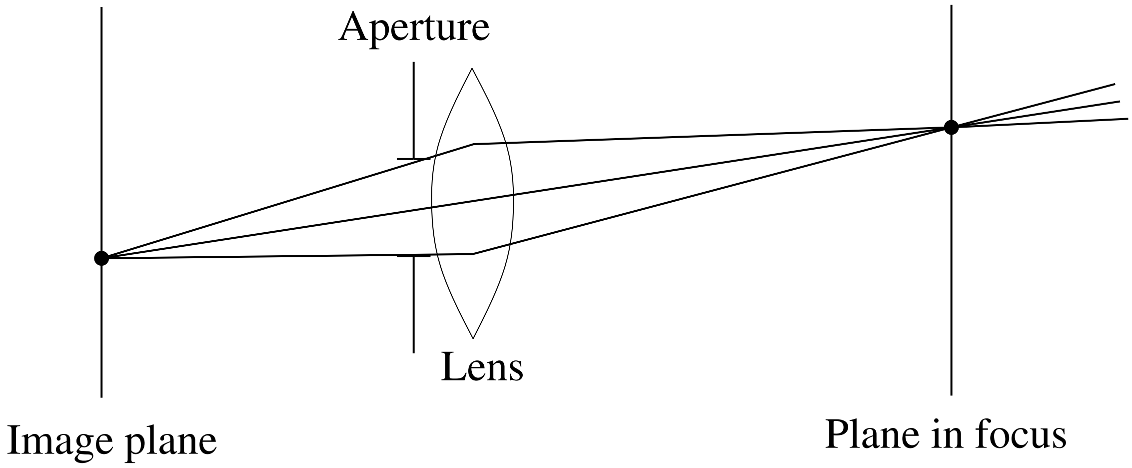


Distributing rays over light source area gives:

Distributed ray tracing, cont.

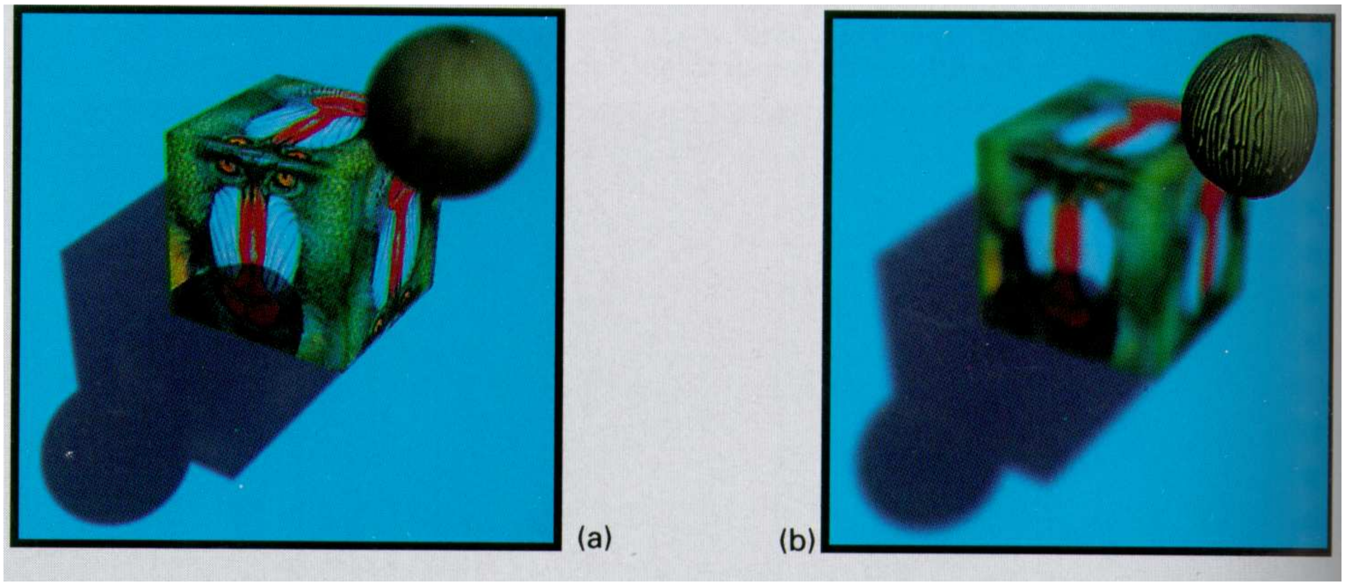


Distributed ray tracing, cont.



Distributing rays over a finite aperture gives:

Distributed ray tracing, cont.



Depth of field, (Foley, II-38)