Lecture #17

Ray Tracing – Secondary Effects

Computer Graphics

Winter Term 2020/21

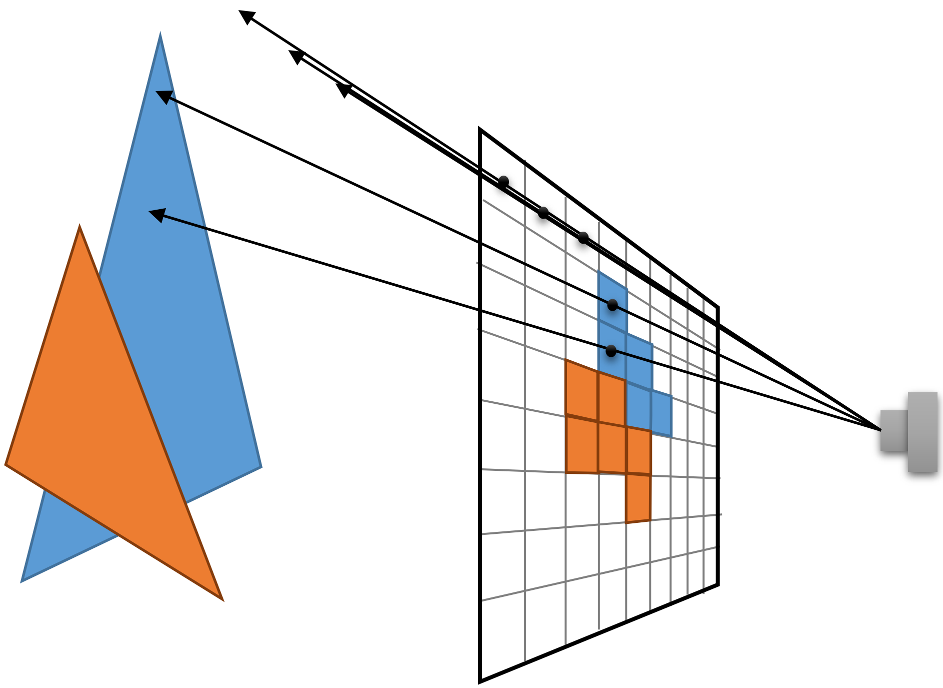
Marc Stamminger / Roberto Grosso

Ray Casting

- Ray Casting ⊂ Ray Tracing

```
for each pixel p
    cast a ray through pixel p
        shade p
```

= "find scene point visible in p"



eye rays only

Ray Tracing

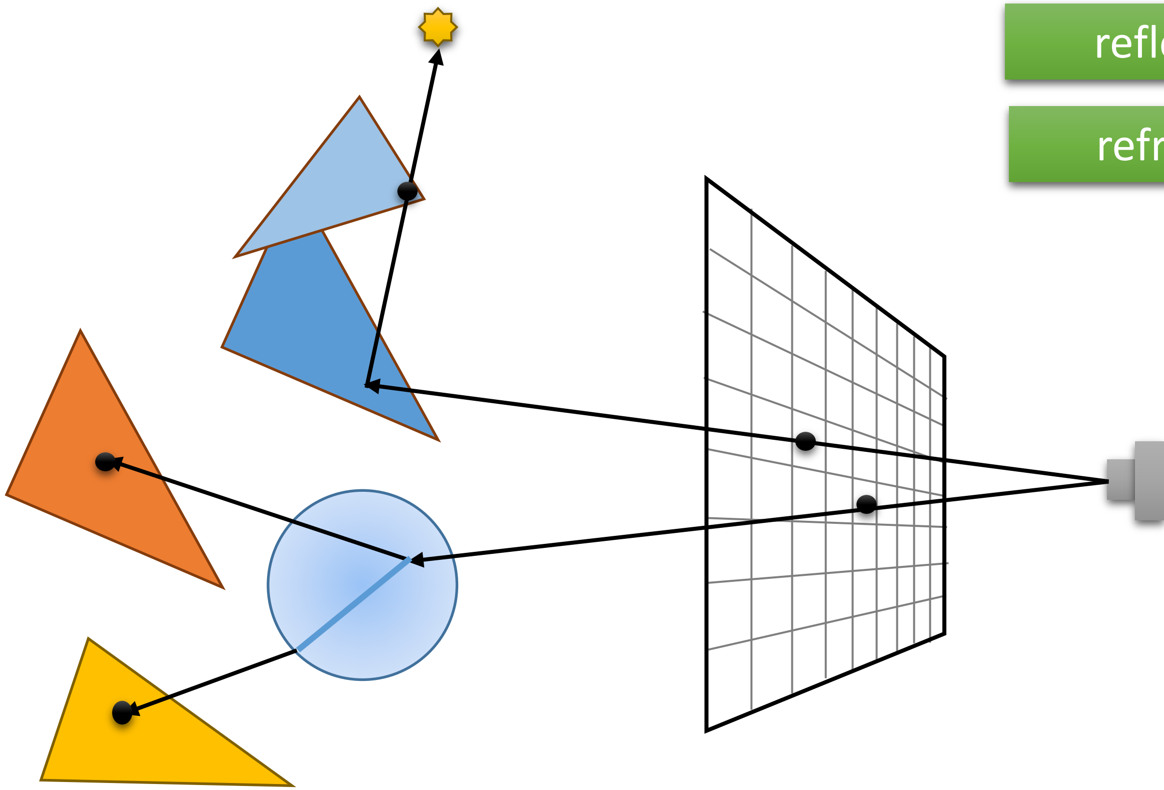
- Ray Casting → Ray Tracing

eye rays

shadow rays

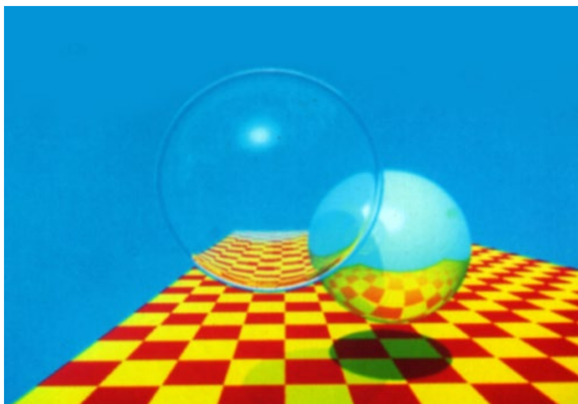
reflection rays

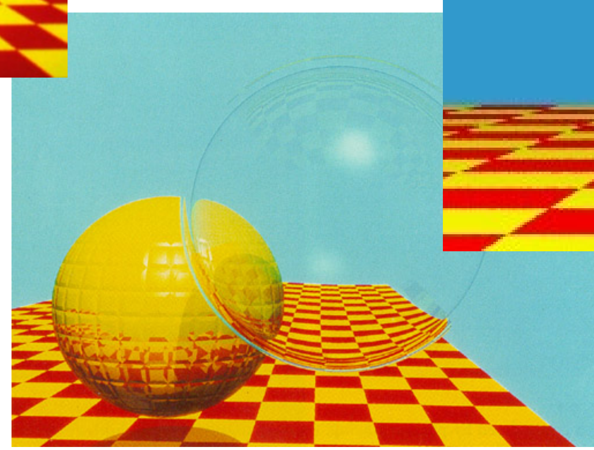
refraction rays

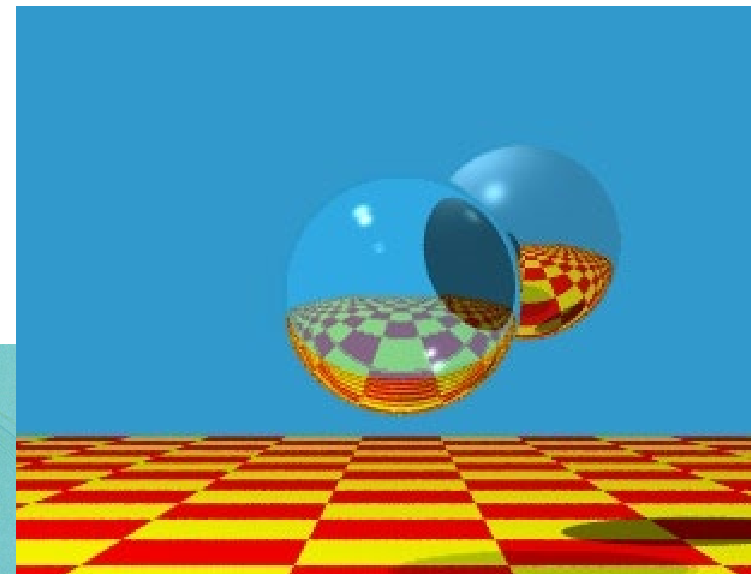


Introduction

- 1968: Ray Casting: Arthur Appel
- 1979: Recursive ray tracing: Turner Whitted







reflection
and
refraction

Images by Turner Whitted

Introduction

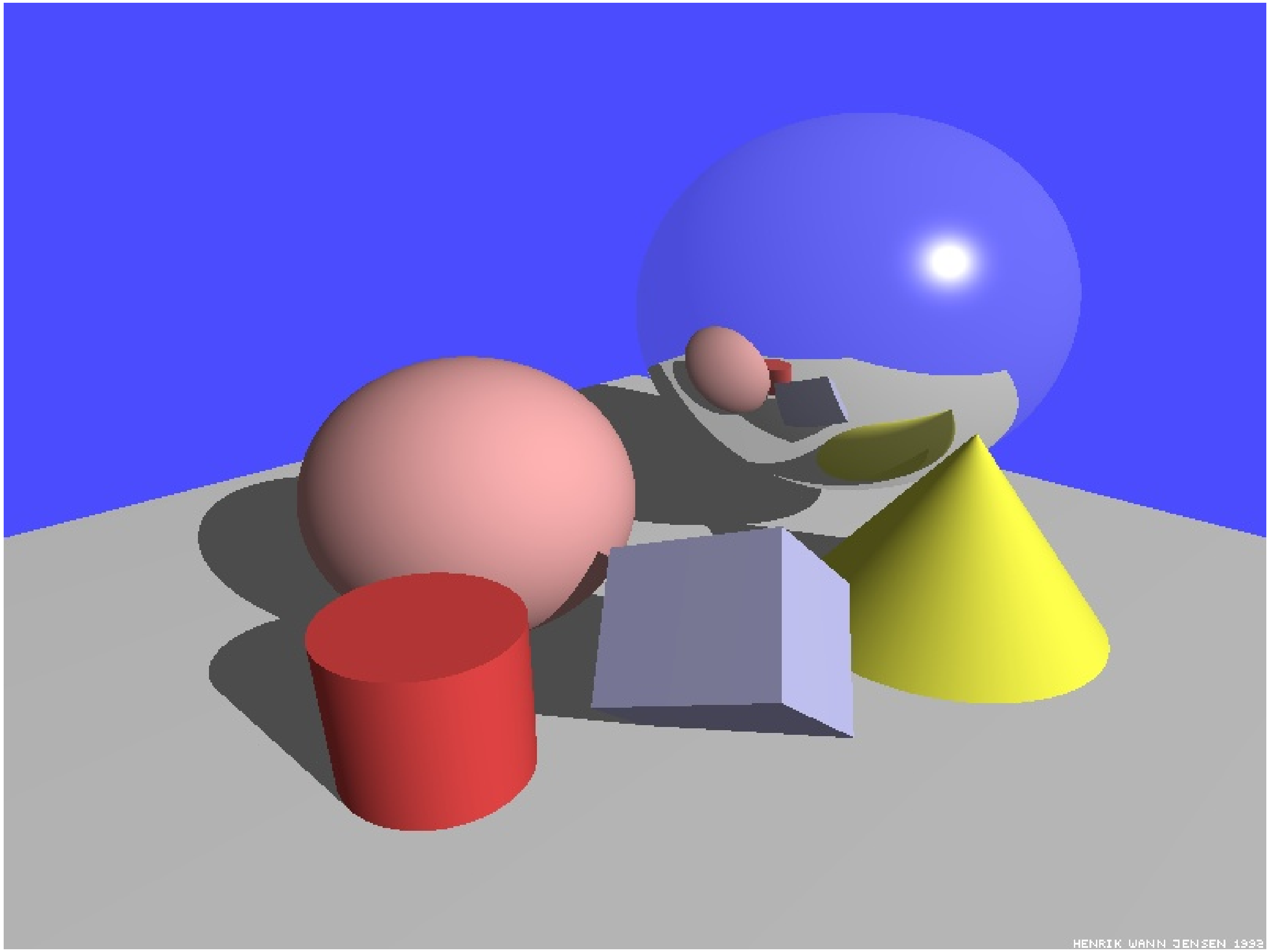


Image by Henrik Wann Jensen. He writes: One of my first ray tracing images (1990-1991).
Rendered first time on an Amiga in HAM mode (the good old days).

Basic Ray Tracing

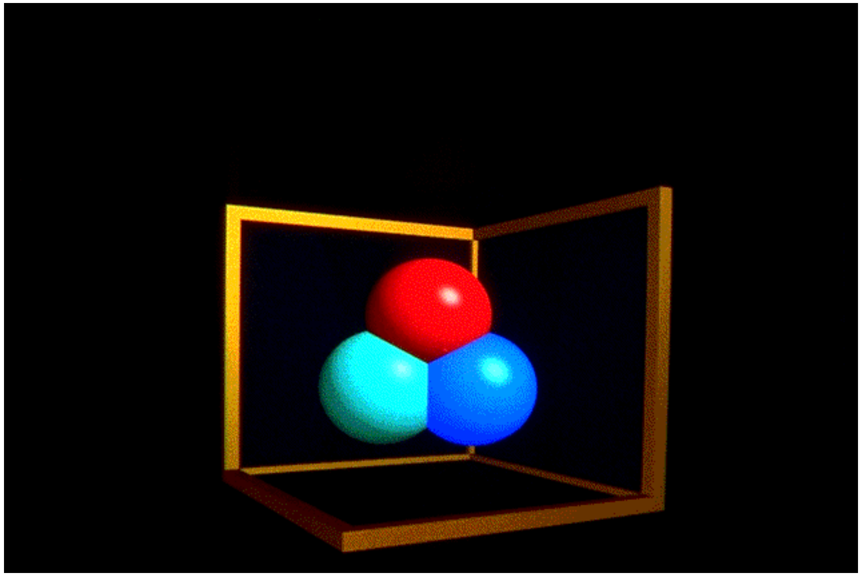
```
for each pixel p
    compute eye ray
    c = raytrace(ray,0)
    set pixel p to color c

raytrace(ray,depth)
    hit = intersect(scene,ray)
    c = black;
    if (hit.shader.isReflective() and depth < maxdepth)
        reflray = compute reflection ray
        c += raytrace(reflray,depth+1) * reflcolor
    if (hit.shader.isRefractive() and depth < maxdepth)
        refrray = compute refraction ray
        c += raytrace(refrray,depth+1) * refrcolor
    shadowRay = compute shadow ray
    if (intersect(scene,shadowRay)
         c += hit.shader.ambientColor
    else
         c += hit.shader.computePhongColor();
    return c;
```

Ray Tracing

- Reflected rays can generate other reflected rays that can generate other reflected rays, etc. (recursion)

→ Layers of reflection

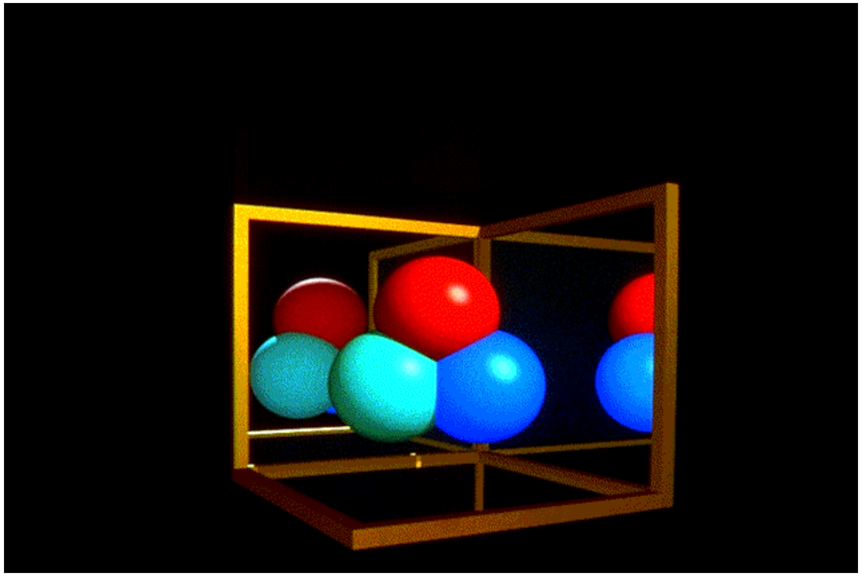
  - Scene traced with no reflection.



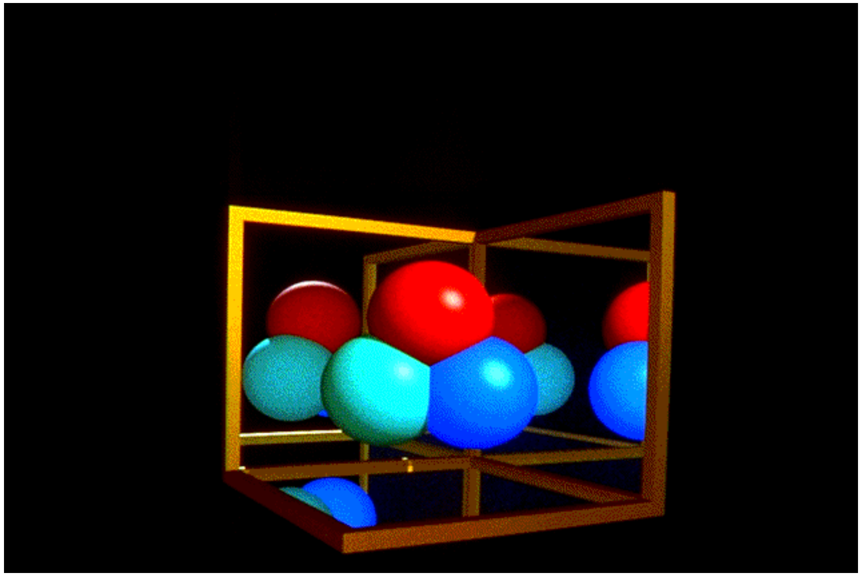
no reflection, maxdepth = 0

Source Image by Michael Sweeny, SIGGRAPH, 1991

Ray Tracing



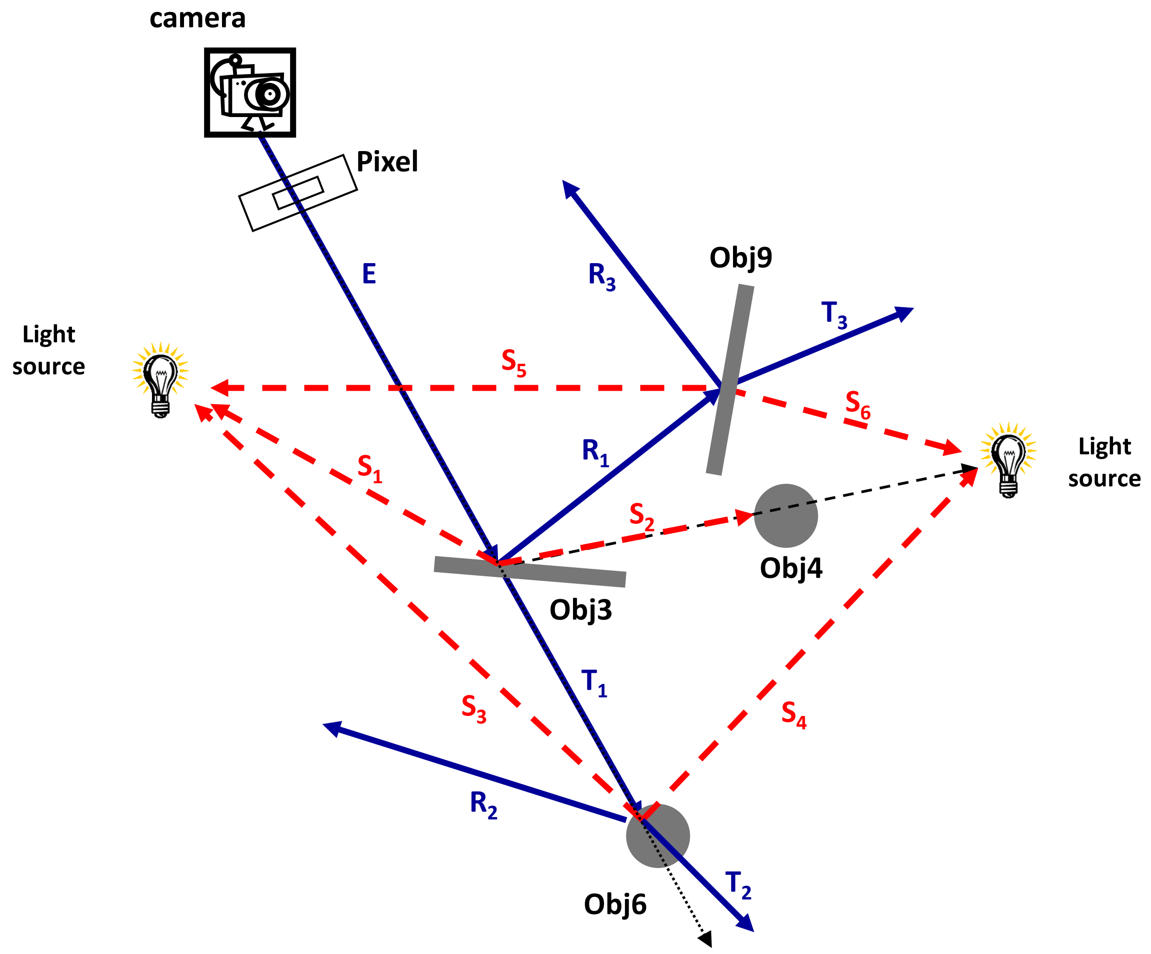
a single reflection, maxdepth =1



double reflection, maxdepth =2

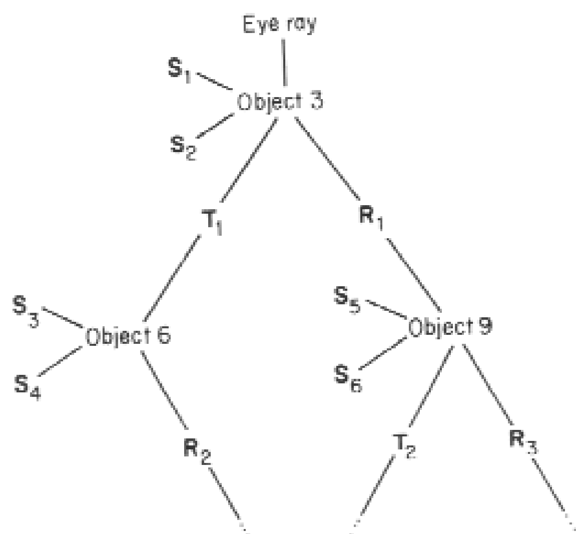
Source Michael Sweeny, SIGGRAPH, 1991

Basic Ray Tracing

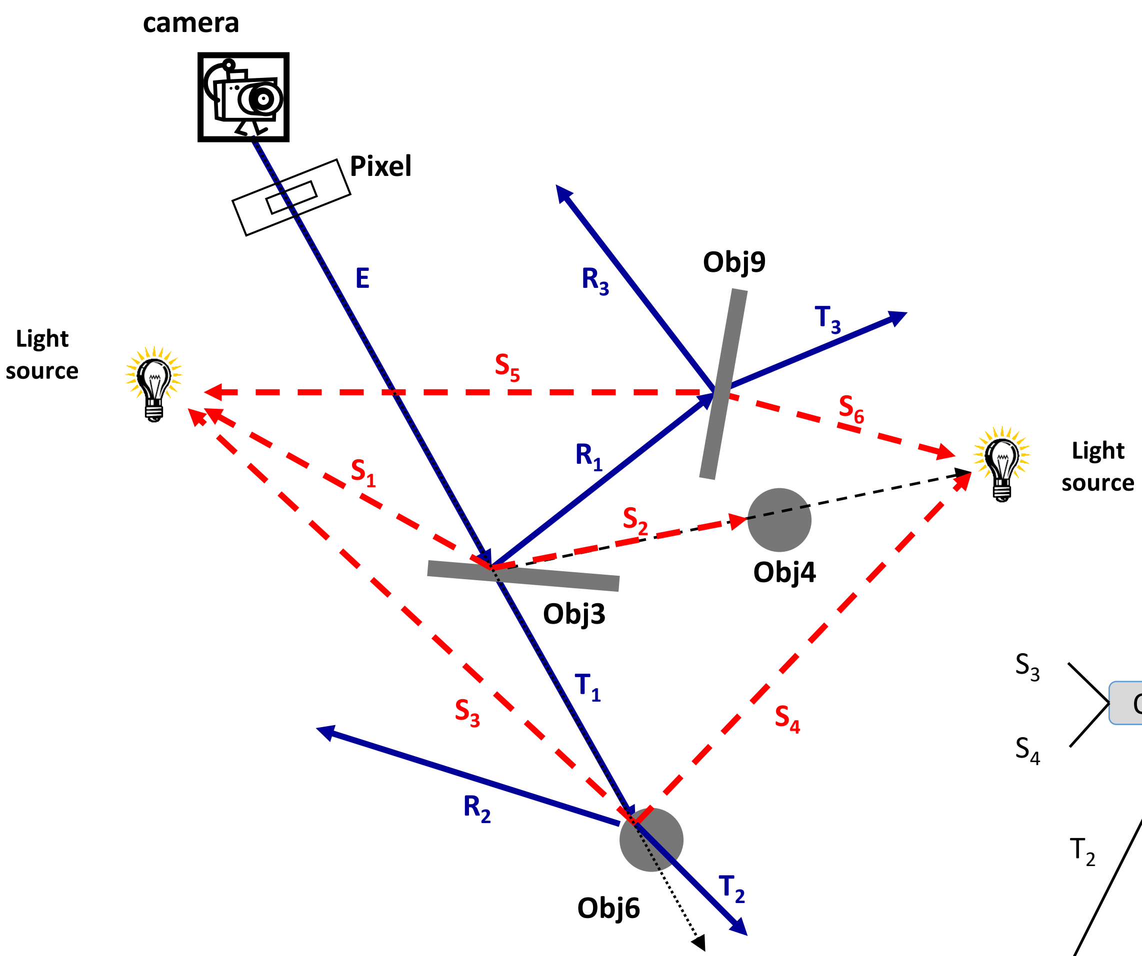


Basic Ray Tracing

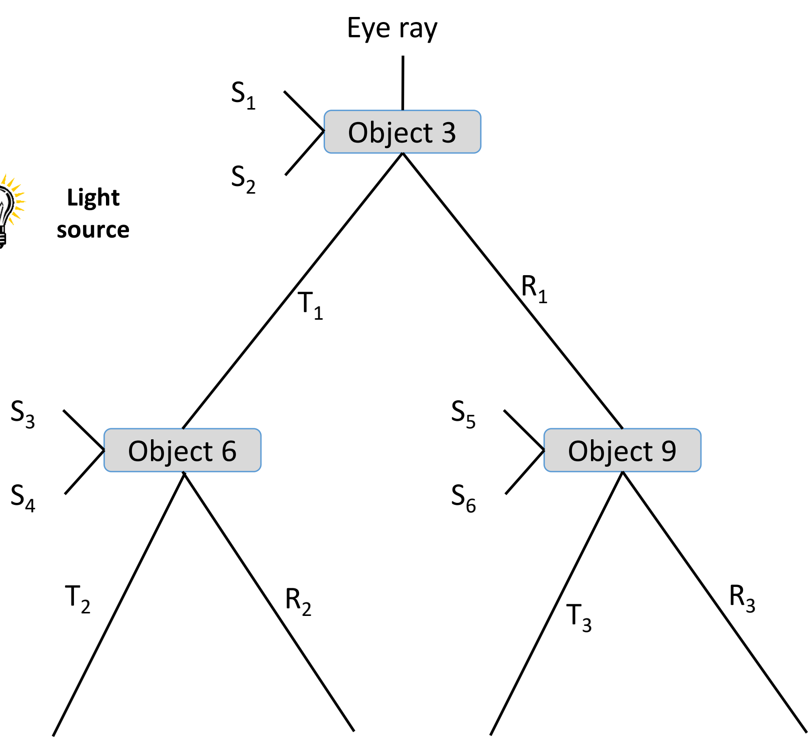
- Rays form a **Ray Tree**
- Recursive illumination calculation
- Many rays per pixel ! → millions or billions of rays…



Basic Ray Tracing

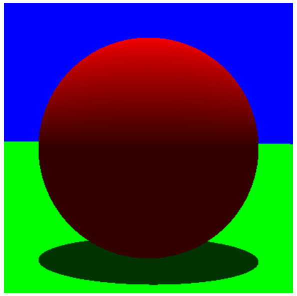


Ray tree

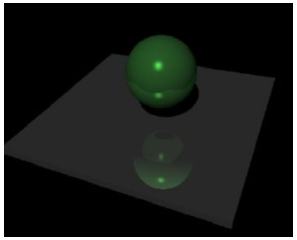


Basic Ray Tracing

- Topics today:
  - Shadows



  - Reflection

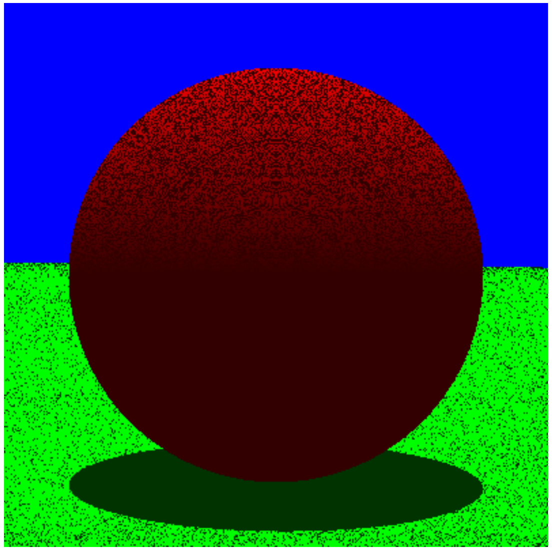


  - Transmission / Refraction

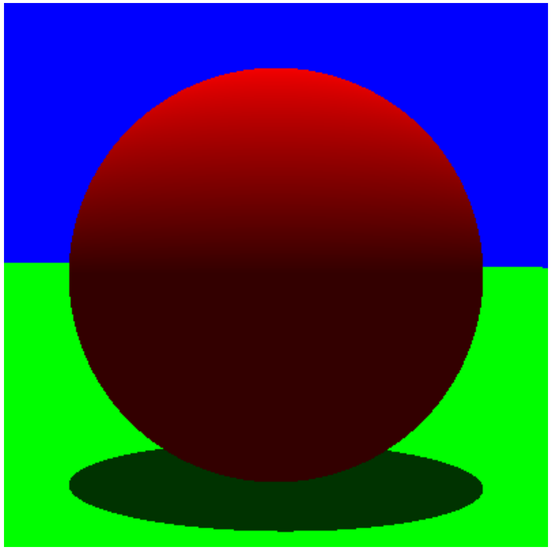
  - Ray Differentials

Shadows

- Shadow rays:
  - only intersection between surface point and light source are of interest
  - only search for intersections with $t \in [t_{min}, t_{max}]$
  - $t_{min} = 0$? → better not → **self shadowing**
    = intersection of shadow ray with object itself due to numerical issues



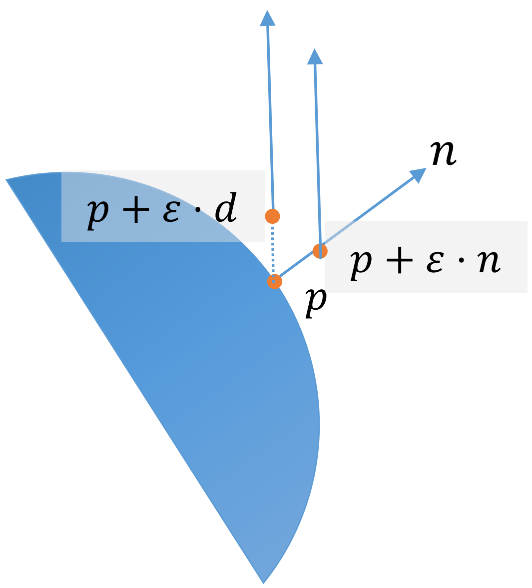
Self-shadowing



Shadow ray with epsilon

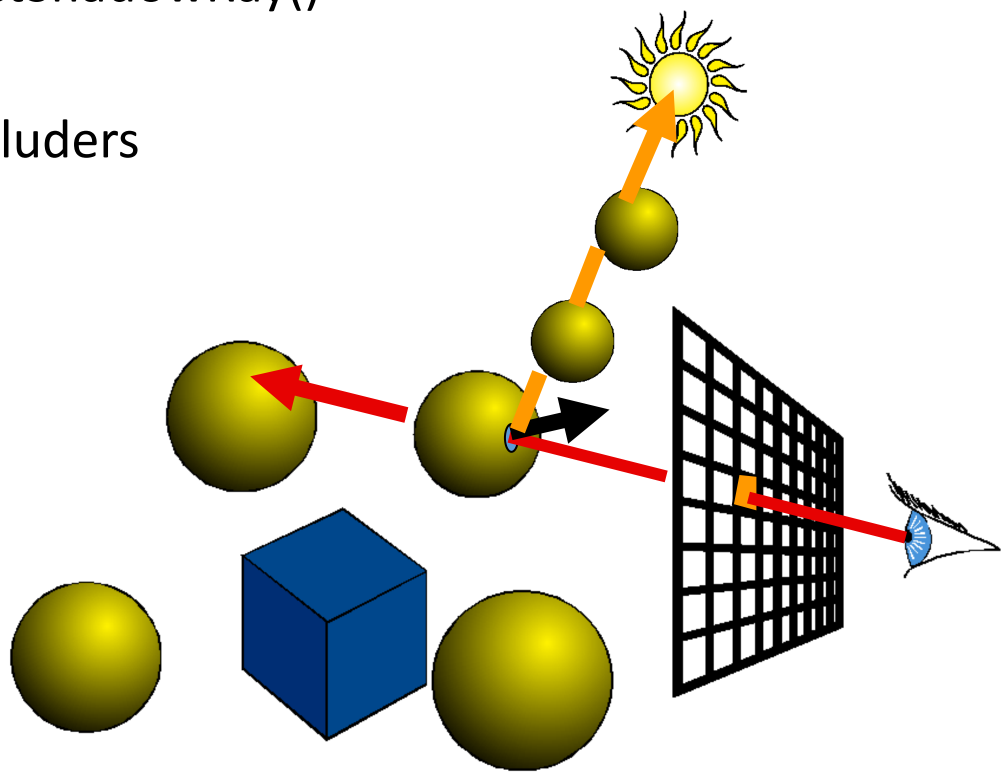
Shadows

- start secondary ray at $p = e + t \cdot d$
  → self shadowing
- Use instead $p + \varepsilon \cdot d$ or $p + \varepsilon \cdot n$ (offset along the normal)
- or use $t_{min} = \varepsilon$



Shadow optimization

- Shadow rays are special:
  → we only want to know *whether* there is an intersection, not *which* one is closest
- Special routine Object3D::intersectShadowRay()
  → Stops at first intersection
- optional: **transparent** shadow occluders
  - gather opacity along shadow ray
  - all intersections needed
  - cannot consider refraction !



Shadows

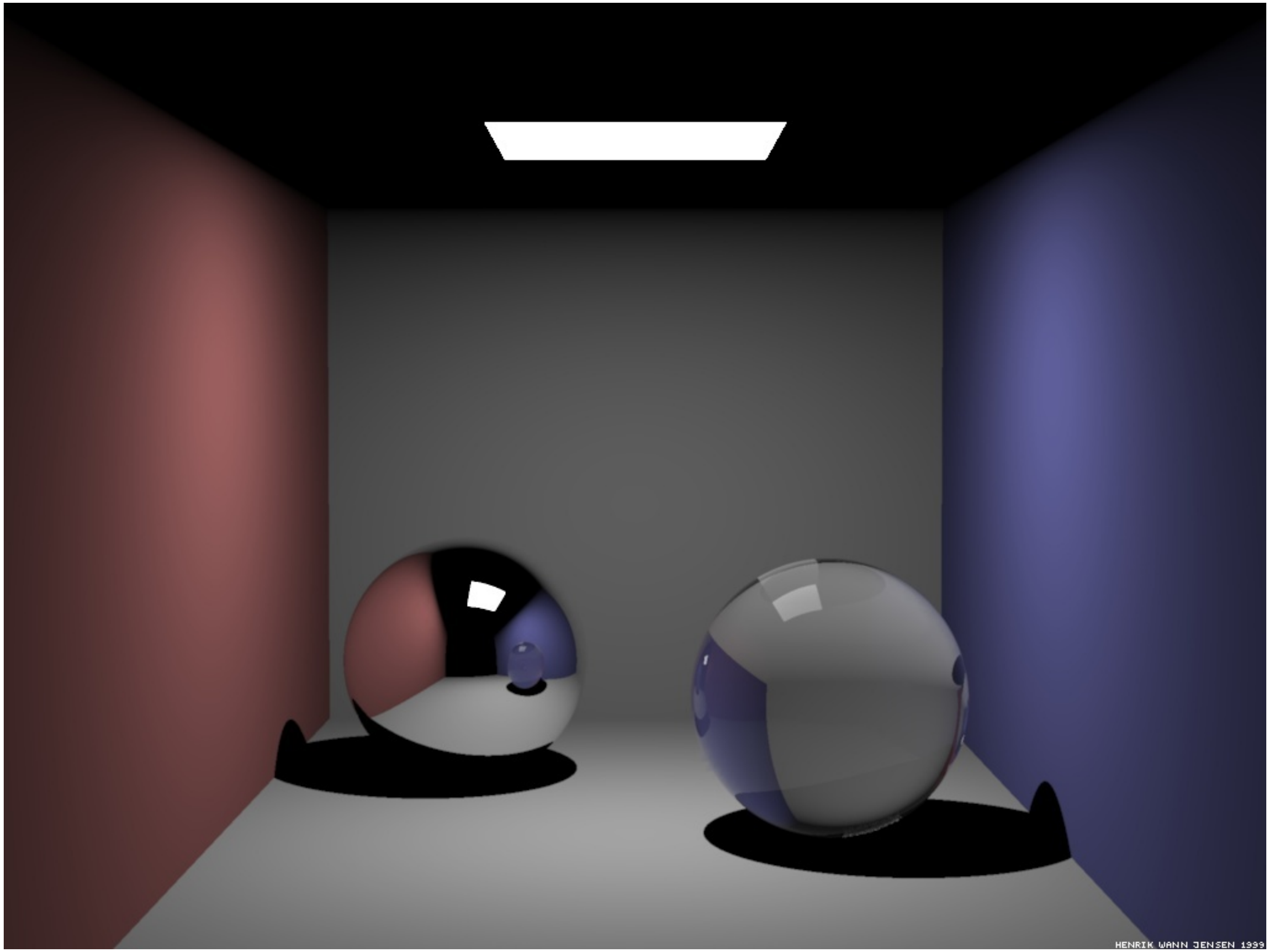
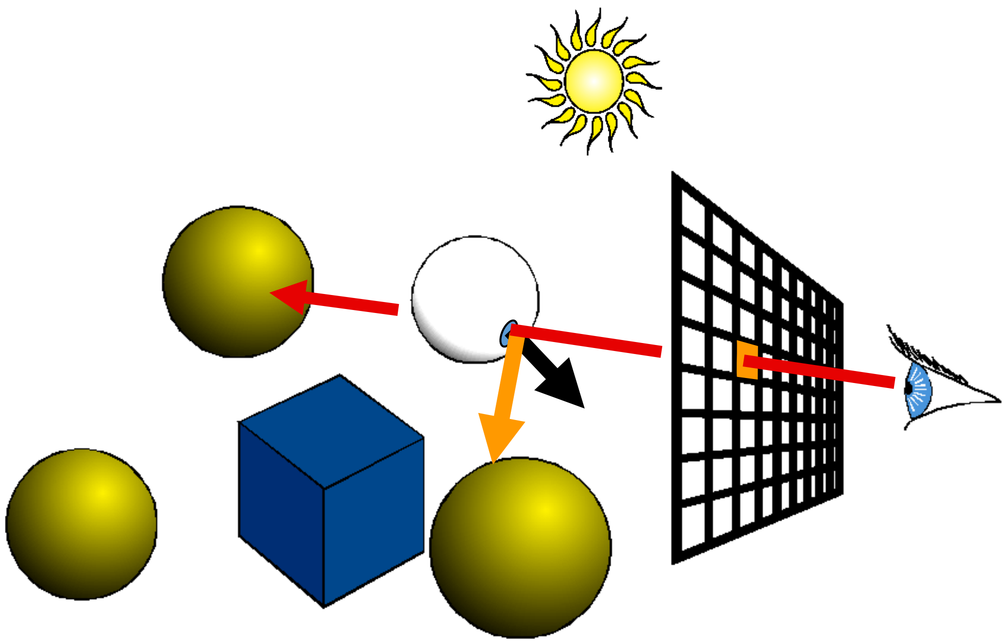
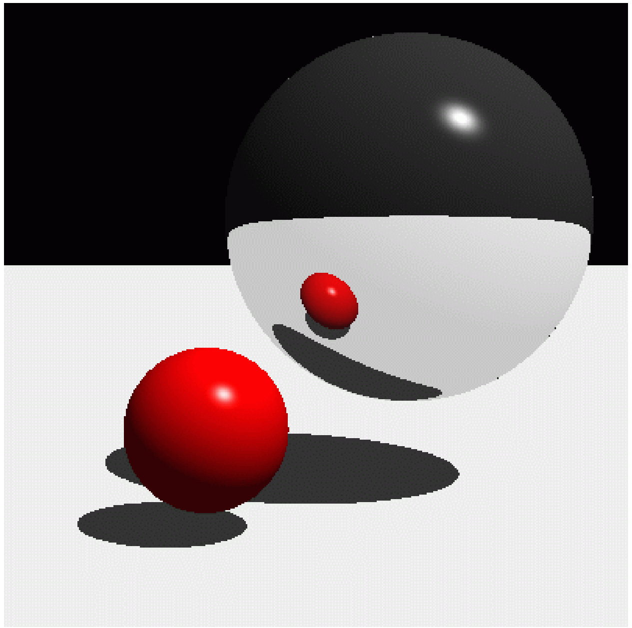


Image by Henrik Wann Jensen

Reflection

- Reflection
  - Compute mirror contribution



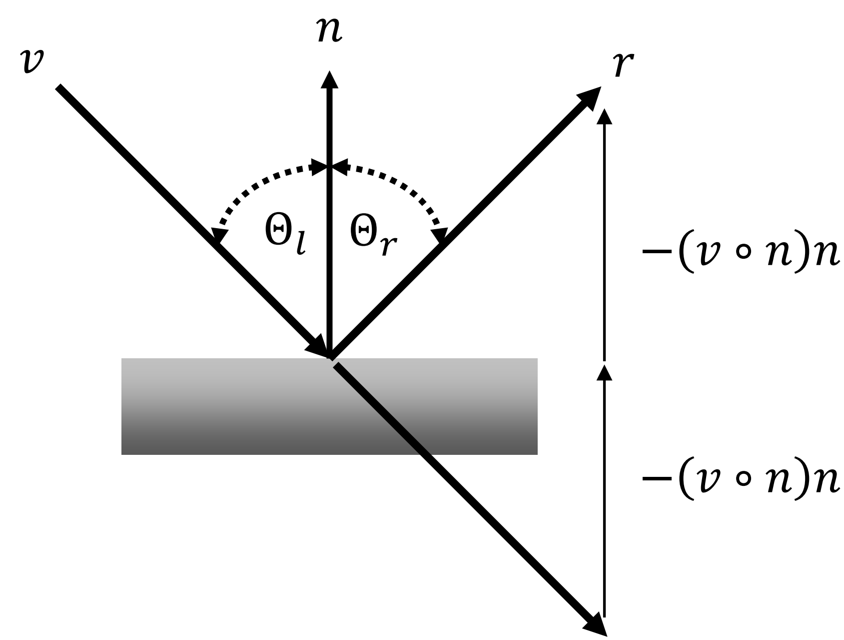


Reflection

- Reflection
  - Compute mirror contribution
  - For every hit point x cast reflection ray in direction symmetric w.r.t. normal, angle of incidence equals angle of reflection.

  - Multiply by reflection coefficient (color)
    Pixel color =
    lighting at x +
    reflection coeff. × light of reflected ray

  - In reality: reflection color varies with angle of incidence (Fresnel) → later



$\rightarrow r = v - 2(v \circ n)n$

Reflection

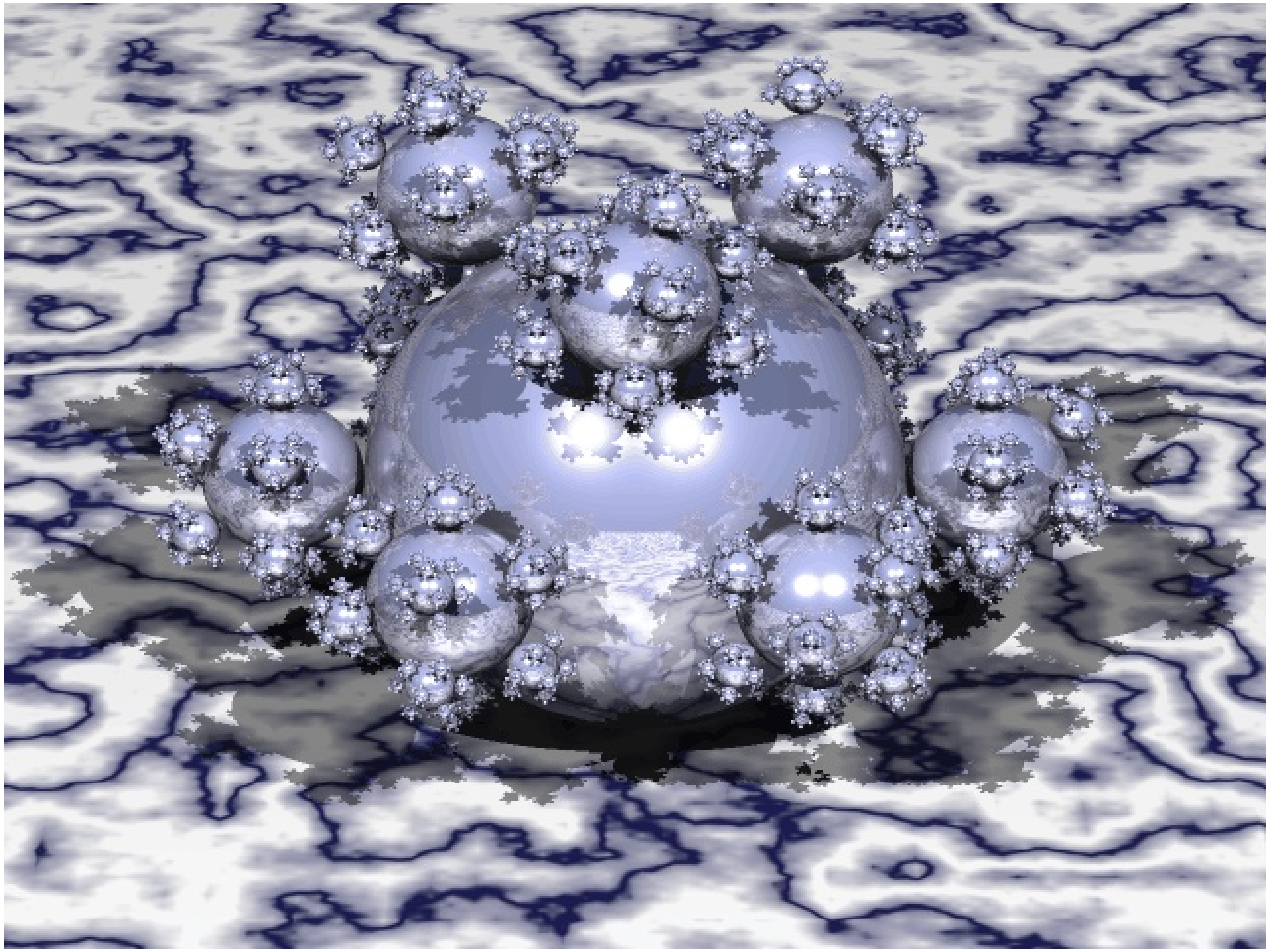
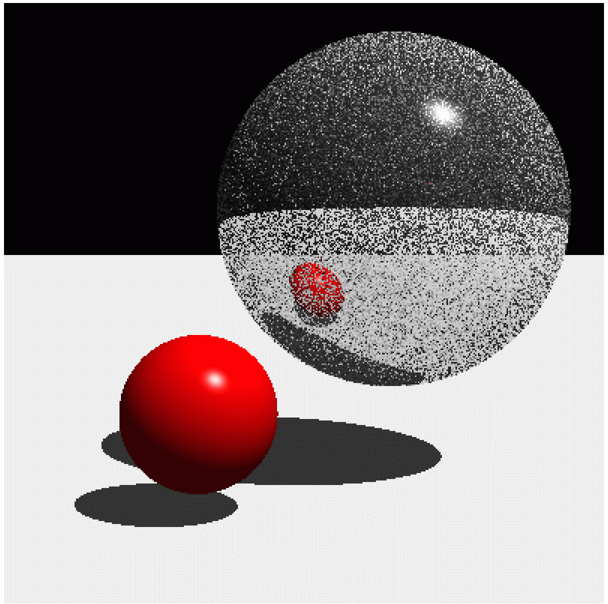


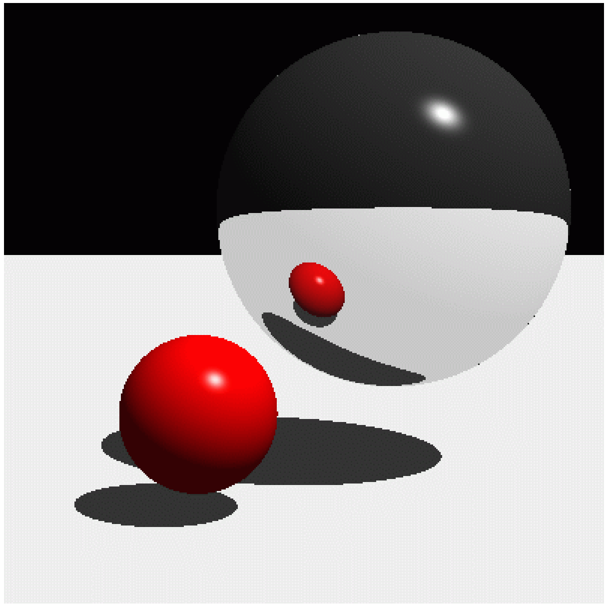
Image by Henrik Wann Jensen, 1992.
It is a procedural (fractal object). Here approximated using 500.000 spheres.

Reflection

- Don’t forget to add epsilon to the ray
  → same problem as with self shadowing



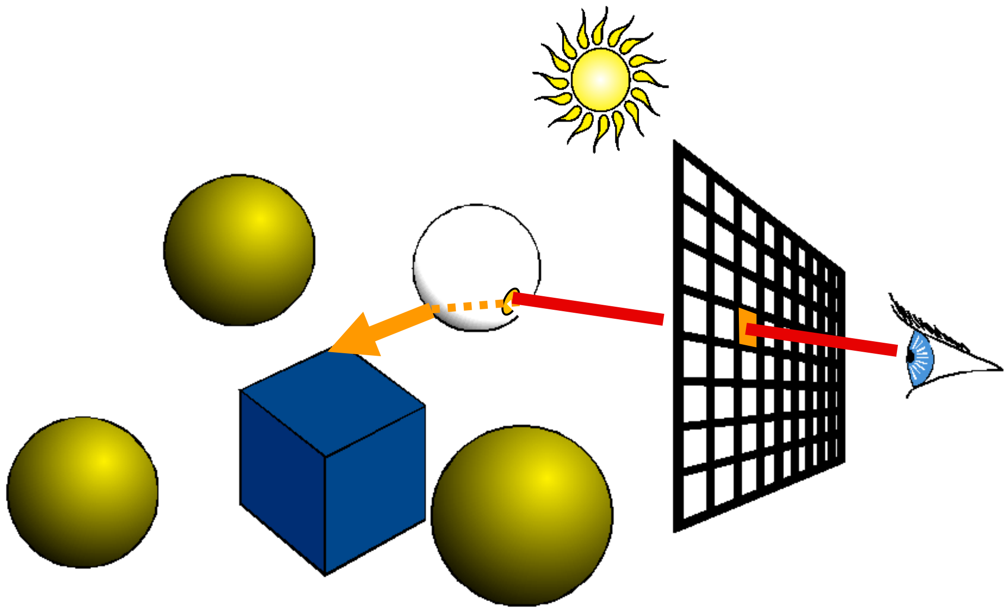
without epsilon



with epsilon

Refraction

- Compute refracted contribution



Refraction

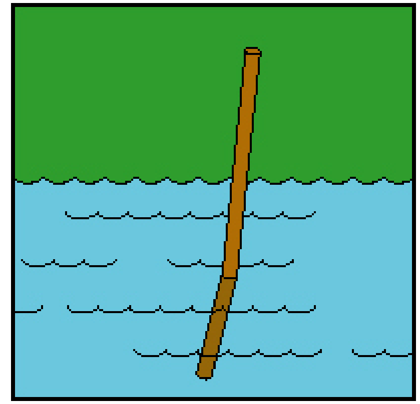
- For every hit point x, cast a ray in direction of refraction
- Pixel color = lighting at x
  + reflection coeff. × light of reflected ray
  + refraction coeff. × light of refracted ray

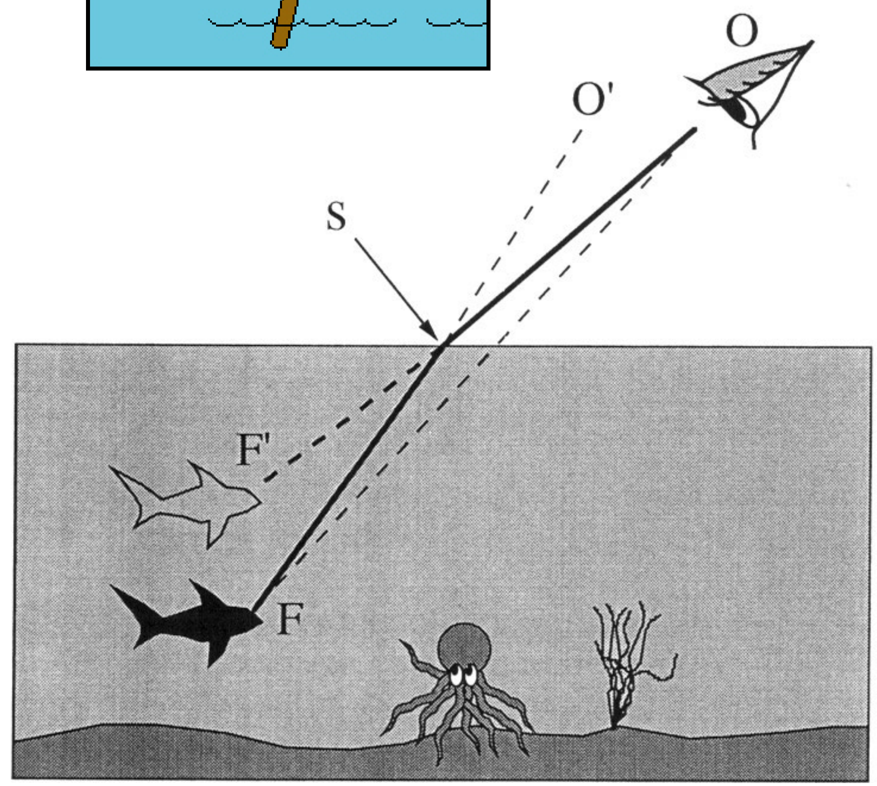
Refraction

- Straight stick in water
- Light bends at the point where it enters the water

- light passing from one transparent medium to another changes speed of light and bends trajectory

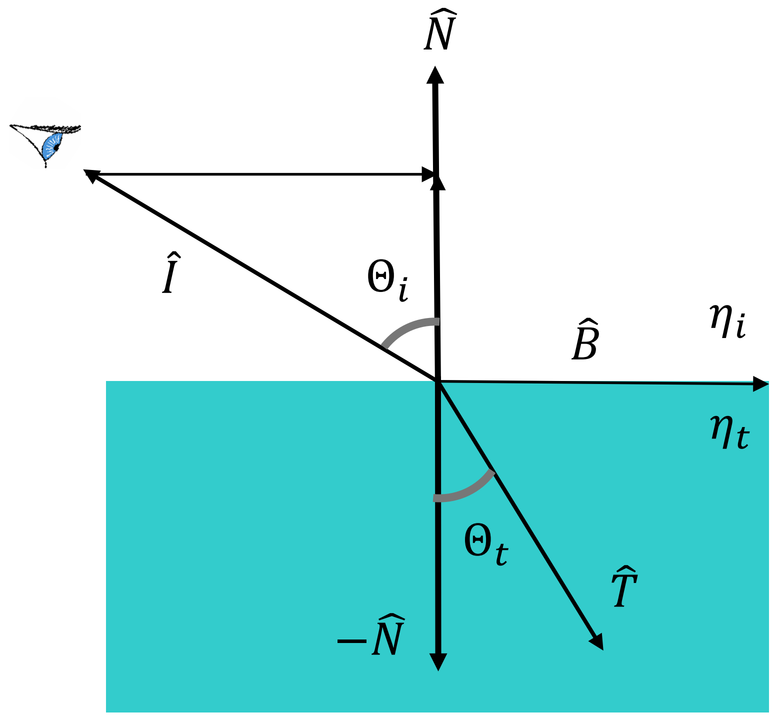
- Effect depends on
  - refractive index of mediums
  - Angle between light ray and normal





Refraction

- Snell-Descartes Law
  - two media with different refraction indices η_i and η_t
  - $\frac{\sin \Theta_i}{\sin \Theta_t} = \frac{\eta_t}{\eta_i} = \eta_r$



Reflections and Refractions: Example from ShaderToy

- ShaderToy:
  Website with lots of nice render examples, all implemented in a single shader
- Very often ray-tracer in a shader (such as this example)
- Look at the example and search for "refraction" in the code

[shadertoy.com - Buoy]

Reflection + Refraction

- Fresnel equations
  - light moves from one medium with refractive index n_1 to a medium with refractive index n_2.
  - part of the energy is **reflected** and part is **transmitted**.
  - the fraction of the power reflected is given by the reflectance R.
  - the fraction of the power transmitted is given by the transmittance T.

  - The constants R and T depend on the polarization of light.
  - The angles of the incident and refracted rays with the normal of the interface are given by the Snell-Descartes law.

Reflection + Refraction

- s-Polarization: electric field perpendicular to plane

$$R_s = \frac{\sin^2(\Theta_t - \Theta_i)}{\sin^2(\Theta_t + \Theta_i)} = \left(\frac{n_1 \cos\Theta_i - n_2 \cos\Theta_t}{n_1 \cos\Theta_i + n_2 \cos\Theta_t}\right)^2$$

- p-polarization: electric field parallel to plane

$$R_s = \frac{\tan^2(\Theta_t - \Theta_i)}{\tan^2(\Theta_t + \Theta_i)} = \left(\frac{n_1 \cos\Theta_t - n_2 \cos\Theta_i}{n_1 \cos\Theta_t + n_2 \cos\Theta_i}\right)^2$$

- un-polarized light

$$R = \frac{R_s + R_p}{2}$$

Reflection + Refraction

- Transmission coefficients

$$T_s = 1 - R, \qquad T_p = 1 - R_p, \qquad T = 1 - R$$

- If light is normal incident

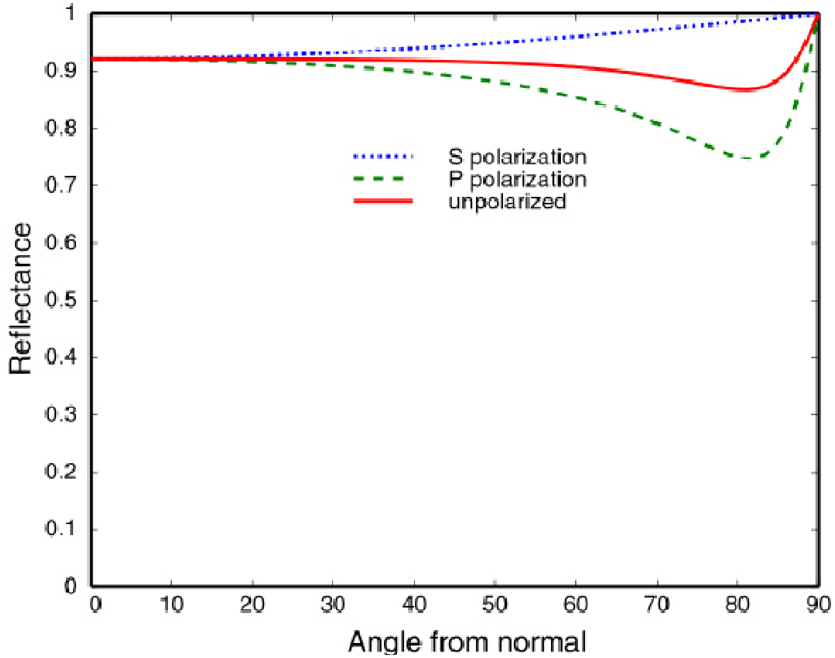
$$R_0 = R_s = R_p = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$$

$$T_0 = T_s = T_p = 1 - R = \frac{4n_1 n_2}{(n_1 + n_2)^2}$$

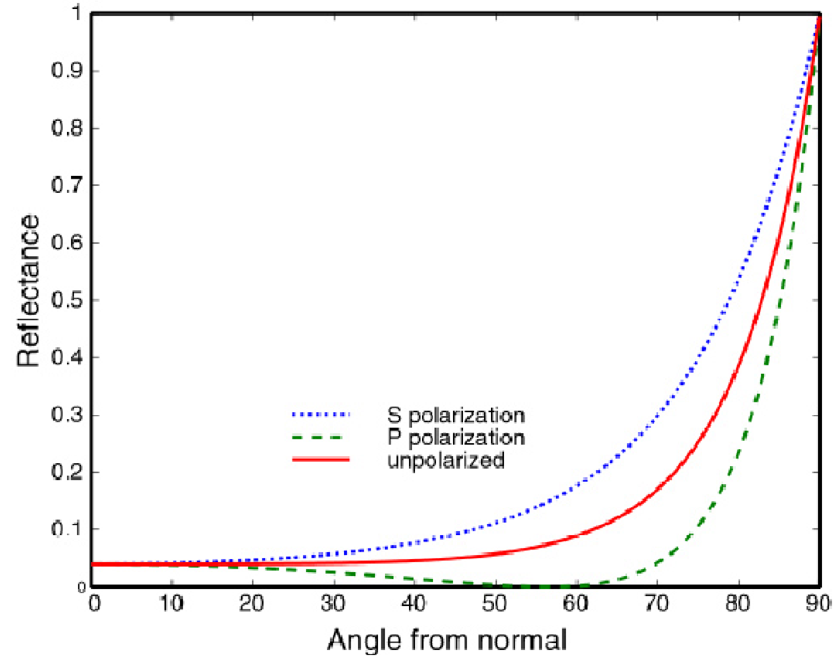
- Schlick approximation (Schlick, 1994)

$$R(\Theta_i) = R_0 + (1 - R_0)(1 - \cos\Theta_i)^5$$

Reflection + Refraction



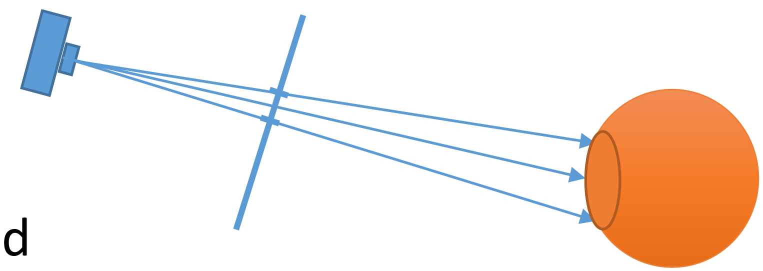
metal



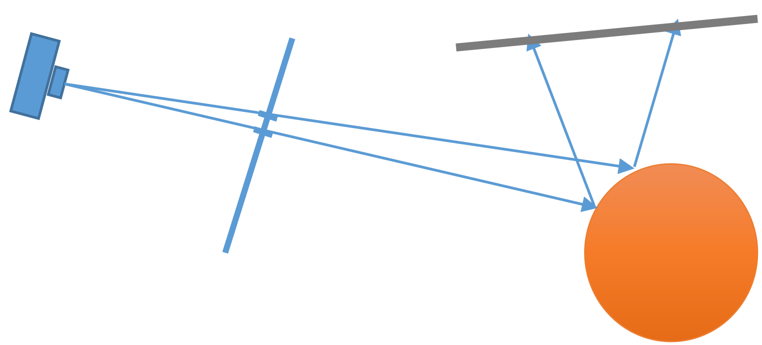
Dielectric (glass)

Ray Differentials

- Known problem: a ray is infinitely small
- Texture filtering requires size of ray bundle representing current pixel → “footprint” → see lecture “Texture aliasing”

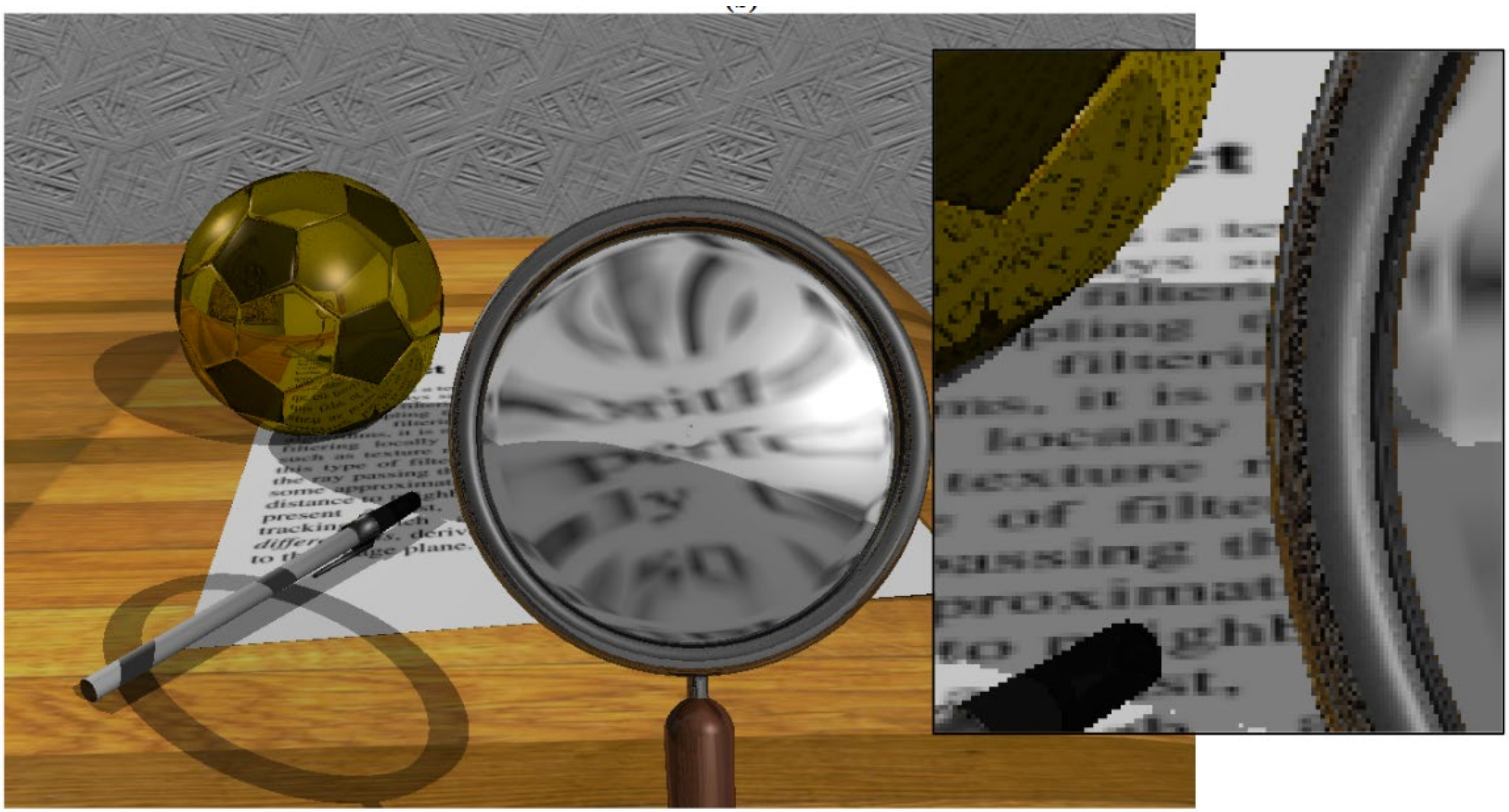


- Even worse, if rays are reflected



Ray Differentials

- Image below is rendered with MIP-Mapping based on ray distance
  → excessive blur in magnifying glass because magnification is not considered

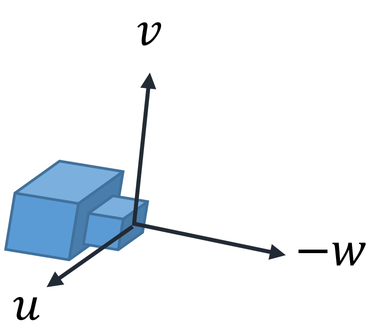


Ray Differentials

- This is how it should look like…

Ray Differentials

- Igehy: „Tracing Ray Differentials“, Siggraph 1999
- Idea
  - A ray is described by a starting point P and a direction D (notation from paper):
    $$P + tD$$
  - Starting point P and direction D of an eye ray can be expressed in terms of the image position (x, y):
    $$P(x, y) = e$$
    $$D(x, y) = normalized(-w + xu + yv)$$
    where (u, v, w) are the vectors spanning the camera coordinate system



Ray Differentials

- Additionally to the ray $[P, D]$, we can store the derivatives w.r.t. x and y:
$$\left[P, D, \frac{\partial P}{\partial x}, \frac{\partial P}{\partial y}, \frac{\partial D}{\partial x}, \frac{\partial D}{\partial y}\right]$$
- The ray differentials tell us how fast a ray changes when moving its starting point on the image plane
- We can track these differentials:
  - at a hit point: how does hit point vary
    → new differential of starting point of secondary ray
  - by reflection: how does reflection direction vary
    → new differential of direction of secondary ray
- By tracking these differentials, we can approximate the footprint of a single pixel at a hit point

Ray Differentials

- Initialization:
  - We start with a simple eye ray (without depth of field or similar):

$$P(x, y) = e$$
$$d = -w + x \cdot u + y \cdot v$$
$$D(x, y) = \frac{d}{\sqrt{d \circ d}} = normalized(d)$$

  - Ray differentials:

$$\frac{\partial P}{\partial x}(x, y) = 0$$
$$\frac{\partial D}{\partial x}(x, y) = \frac{(d \circ d)u - (d \circ u)d}{(d \circ d)^{\frac{3}{2}}}$$

(y analog)

Ray Differentials

- No let‘s assume we found a hit point at t:

$$P' = P + tD$$

- For the hit point we can derive:

$$\frac{\partial P'}{\partial x} = \left(\frac{\partial P}{\partial x} + t \cdot \frac{\partial D}{\partial x}\right) + \frac{\partial t}{\partial x} D$$

- We need to compute $\partial t / \partial x$
  - Depends on surface normal at hit point

Ray Differentials

- Assume hit point is from intersection with plane $n \circ x = b$:

$$t = \frac{b - P \circ n}{D \circ n}$$

- Derivative:

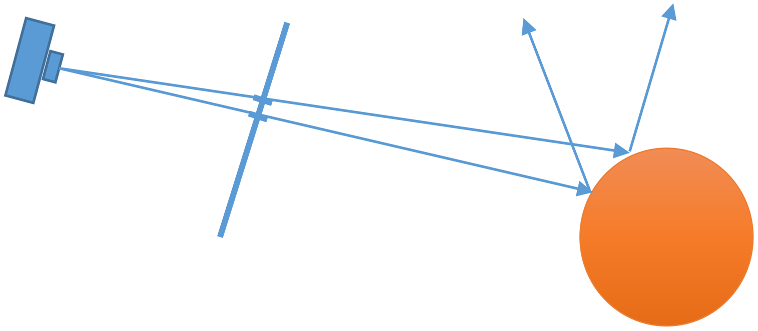
differential of ray direction

$$\frac{\partial t}{\partial x} = \frac{\left(t \frac{\partial D(x,y)}{\partial x}\right) \circ n}{D \circ n}$$

Ray Differentials

- Reflection
  - What happens with ray differentials at reflection?
    - Convex surface → opening angle increased
    - Concave surface → rays get focused

- Refraction similar



Ray Differentials

- Using ray differentials machinery for reflection:
  - Assume a ray hits a surface at P from direction D. Then reflect using
$$D' = D - 2(D \circ N)N$$
  - The ray differential of the direction is then
$$\frac{\partial D'}{\partial x} = \frac{\partial D}{\partial x} - 2\left[(D \circ N)\frac{\partial N}{\partial x} + \left(\frac{\partial D}{\partial x} \circ N + D \circ \frac{\partial N}{\partial x}\right)N\right]$$

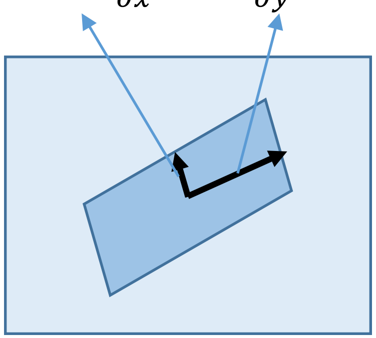
direction differential before reflection

“curvature” of surface

- Refraction similar

Ray Differentials

- Texture filtering
  - Approximate footprint of a pixel in texture space using ray differentials
  - Express texture coordinates as function of hit point P':
$$T = f(P')$$
  - How fast does texture coordinate change w.r.t. (x, y):
$$\frac{\partial T}{\partial x} = \frac{\partial f}{\partial P'}\left(P'(x)\right)\frac{\partial P'}{\partial x}$$
computed previously
  - So we can estimate the footprint
    - $T(x \pm \Delta x, y \pm \Delta y) \approx f(P') \pm \Delta x \frac{\partial T}{\partial x} \pm \Delta y \frac{\partial T}{\partial y}$

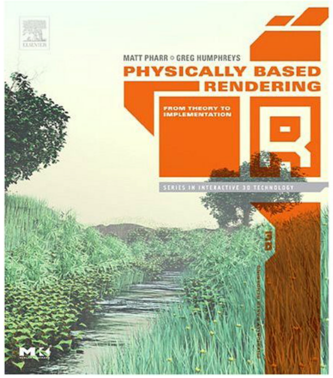


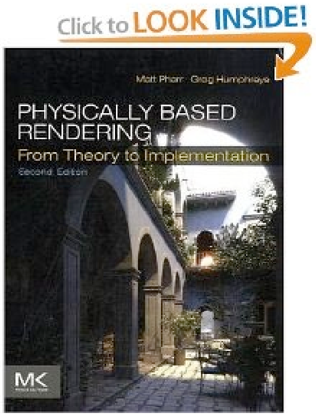
Ray Differentials

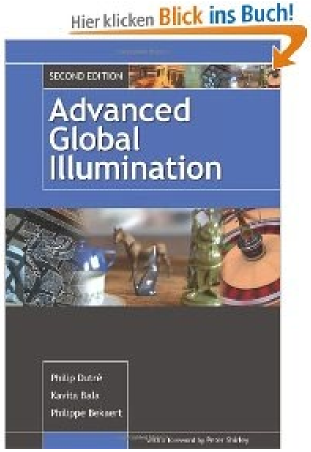
- Footprint is in general not a square, but an arbitrary quadrilateral
- → **Anisotropic filtering (**see lecture “Texture Mapping”) accounts for this and uses arbitrary rectangular filter masks instead of square ones
- Ray differentials deliver information about the footprint

Additional information

- Books:
  - Matt Pharr, Greg Humphreys:
    „PHYSICALLY BASED RENDERING“
    Morgan Kaufmann
    http://www.pbrt.org/







  - Philip Dutre, Kavita Bala,
    Philippe Bekaert:
    “Advanced Global Illumination”

  - Glassner, Andrew S.:
    „**An Introduction to Ray-Tracing**“
    Morgan Kaufmann San Diego: Academic, 1989