Recursive ray tracing (fixed up slide)

- Pixel brightness =
  radiance along ray to pinhole =
  (diffuse) +
  (specular) +
  (reflected) +
  (transmitted)
- Diffuse component:
  - from sources alone (local shading model), usual case
  - from global illumination model
  - typically Lambertian, but better models do exist
- Specular (lobe) component
  - Typically use Phong approximation (text does it this way, see equation 14.32).
- Reflected component is due to radiance along ray from intersection along specular spike direction
- Transmitted component is due to radiance along ray from intersection along transmitted direction

Recursive ray tracing rendering algorithm

- Cast ray from pinhole (projection center) through pixel, determine nearest intersection
- Compute components by casting rays
  - to sources = shadow ray AND specular component
  - along perfect reflection dir = reflected ray
  - along transmitted dir = refracted ray
- Each of the components has a weight
  - The main processes do not affect color much but a vector of weights for may be more convenient, or accurate depending on a color model
- Determine component and form sum, but ...
- To determine some of the components, the ray tracer must be called **recursively**.

Recursive ray tracing rendering (cont)

- Reflections (at least need to be attenuated)--no perfect reflectors
- We must stop the recursion at some point
  - when contributions are too small
    - need to track the cumulative effect
  - typically also limit the depth explicitly

Mechanics

- Primary issue is intersection computations.
  - E.g. sphere, triangle.
- Polygon (see book page 461-2, handout for a better way)
- Sphere (see book page 460, or handout)

Mechanics

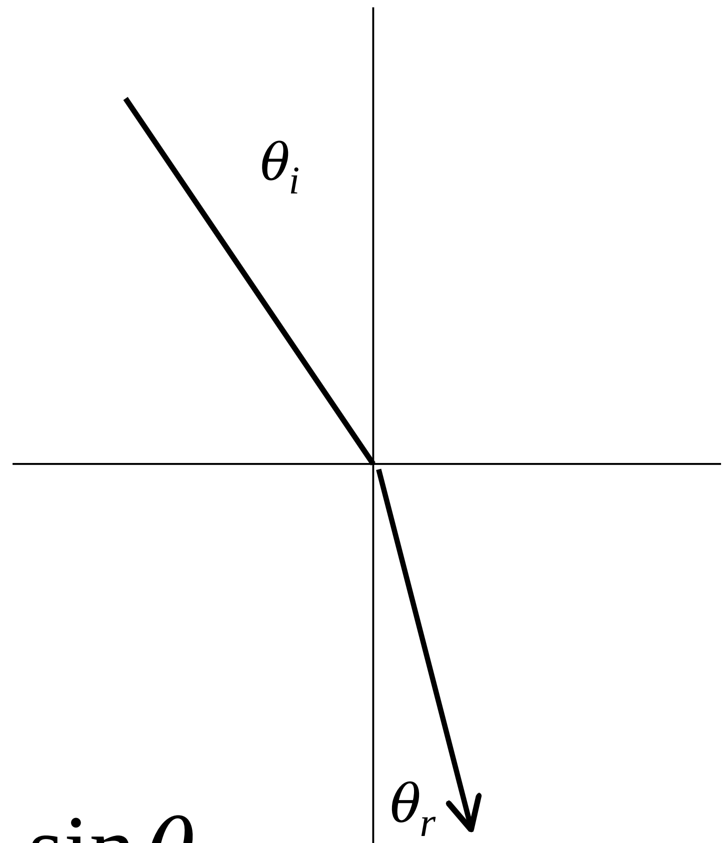
- Other issue is computing ray directions
- We know how to do the reflected direction (text page 486, or handout--important to know this)!
- Transmitted direction from a study of refraction
- Refraction:
  - light changes speed going from medium to medium
  - Snell's law gives relationship between directions and refractive indexes of the media
  - light bends toward the normal going into a medium with higher refractive index

Refraction Details

Index of refraction, n, is the ratio of speed of light in a vacuum, to speed of light in medium.

Typical values:

water: 1.33
glass: 1.45-1.6
diamond: 2.2



$$n_i \sin\theta_i = n_r \sin\theta_r$$

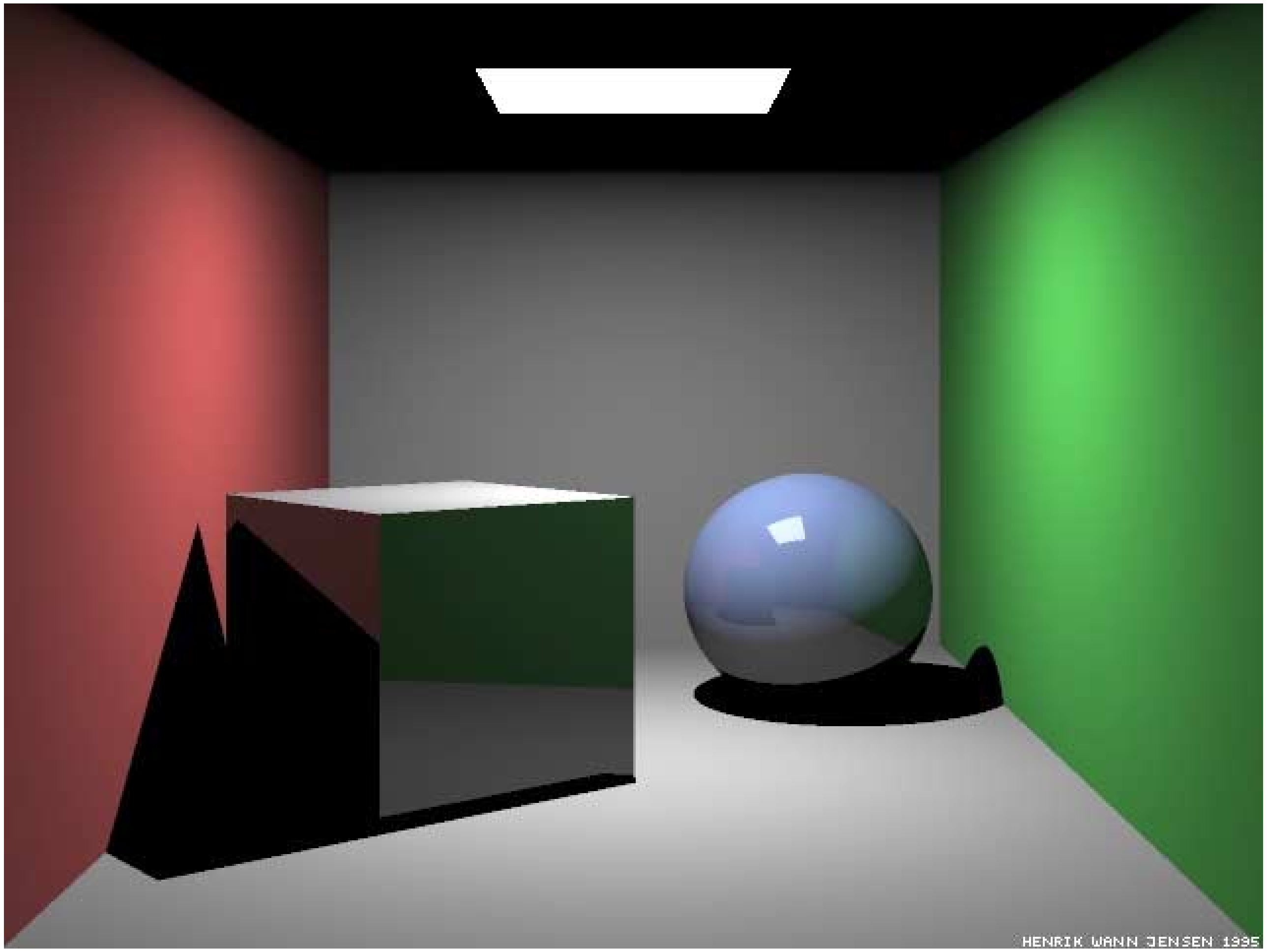
Mechanics

- Another issue is attenuation
- Reflection
- No perfect mirror, typically 95% attenuation at each bounce (or user controlled). If you allow 100% then have to track the depth!
- Absorption
  - Usually can ignore in air, but it depends on the application
  - Translucent absorption is exponential in depth

$$I = I_0 e^{-\alpha d}$$

PCKTWTCH by Kevin Odhner, POVRay

6Z4.JPG - A Philco 6Z4 vacuum tube by Steve



Ray-traced Cornell box, due to Henrik J
http://www.gk.dtu.dk/~hwj

Issues

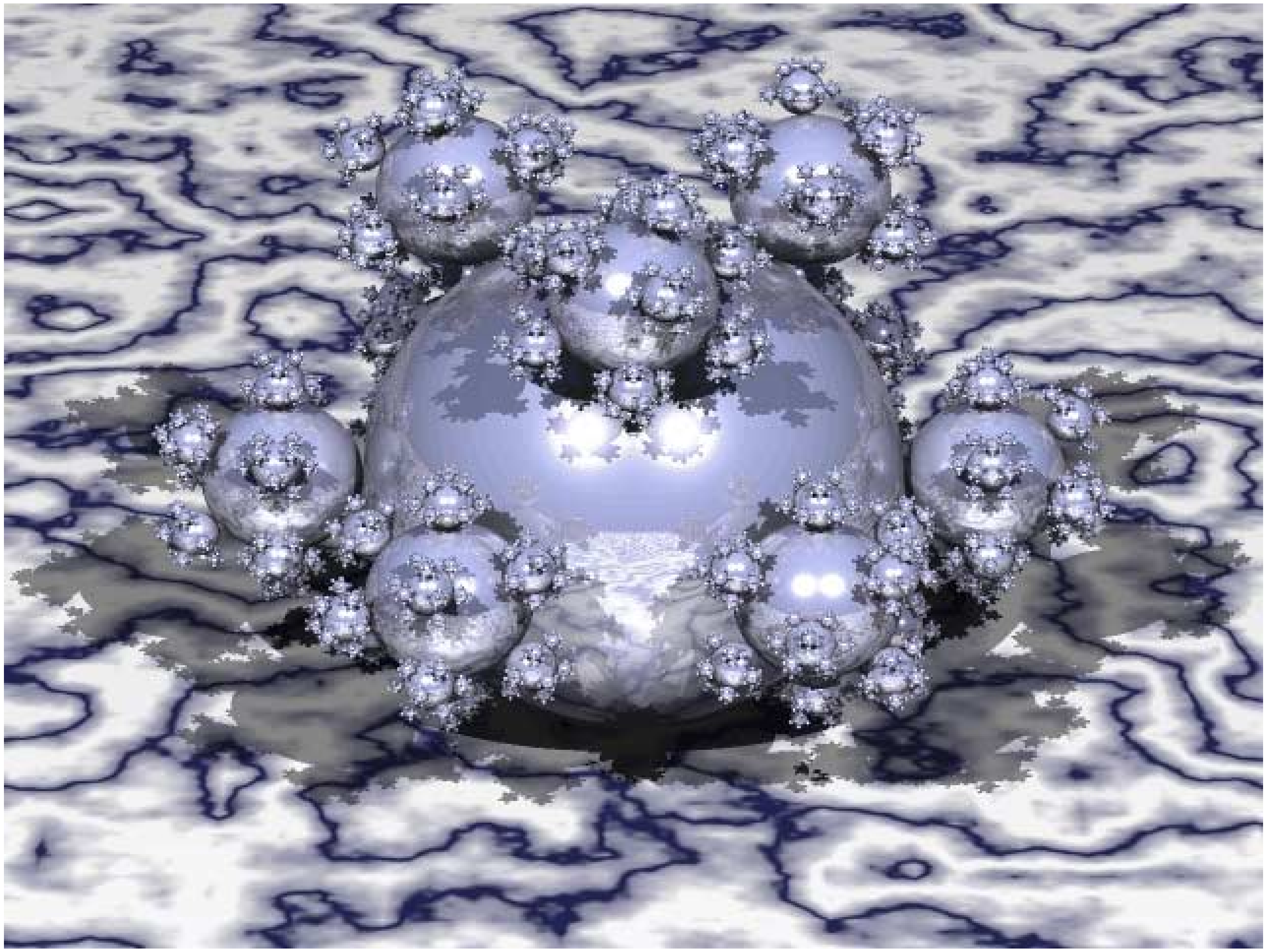
- Sampling
- Very large numbers of objects
  - making intersections efficient
- Surface detail
  - bumps, texture, etc.
- Illumination effects
  - Caustics, specular to diffuse transfer
- Lenses

Sampling

- Simplest ray-tracer is one ray per pixel
  - this aliases (go back to old notes)
- Solutions
  - cast multiple rays per pixel, and use a weighted average
    - go back to old notes for why weighted
  - rays can be on a uniform grid
  - it turns out to be better if they are “quite random” in position
    - “hard-core” Poisson model appears to be very good
    - different patterns of rays at each pixel

Efficiency - large numbers of objects

- Construct a space subdivision hierarchy of some form to make it easier to tell which objects a ray might intersect
- Uniform grid
  - easy, but many cells
- Bounding Spheres
  - easy intersections first
- Octtree
  - rather like a grid, but hierarchical
- BSP tree

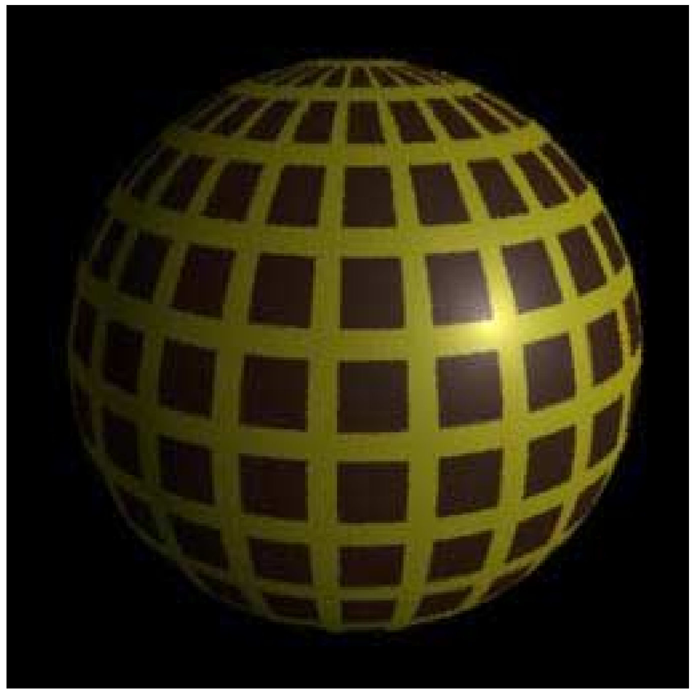


500,000 spheres, Henrik Jensen, http://www.gk.

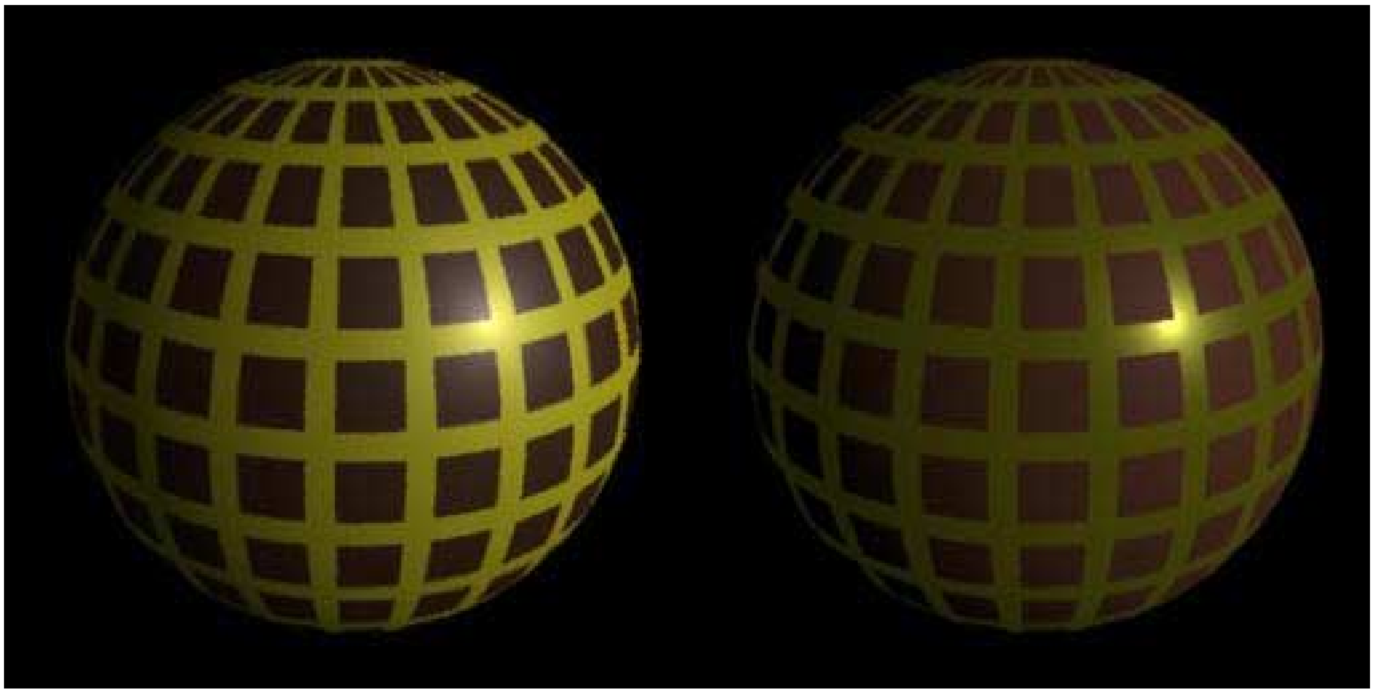
Surface detail

- Knowing the intersection point gives us a position in intrinsic coordinates on the surface
  - e.g. for a triangle, distance from two of three bounding planes
- This is powerful - we could attach some effect at that point

- Texture:
  - make albedo a function of position in these coordinates
  - rendering: at intersection, compute coordinates and get albedo from a map
  - (this is not specific to ray-tracing)



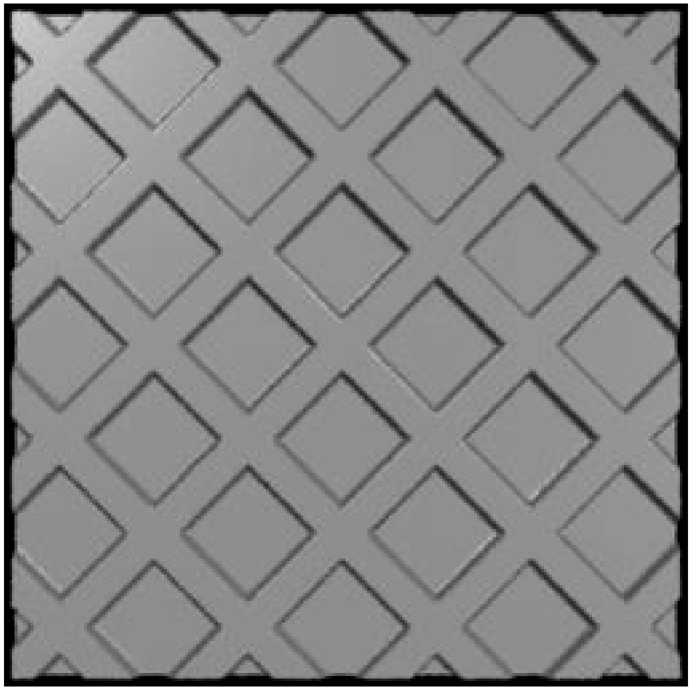
From RmanNotes
http://www.cgrg.ohio-state.edu/~smay/RManNotes/i

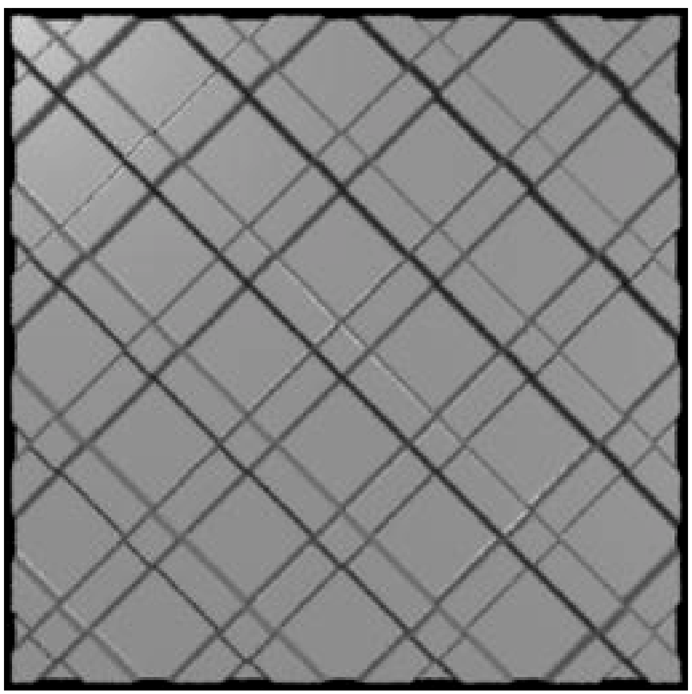


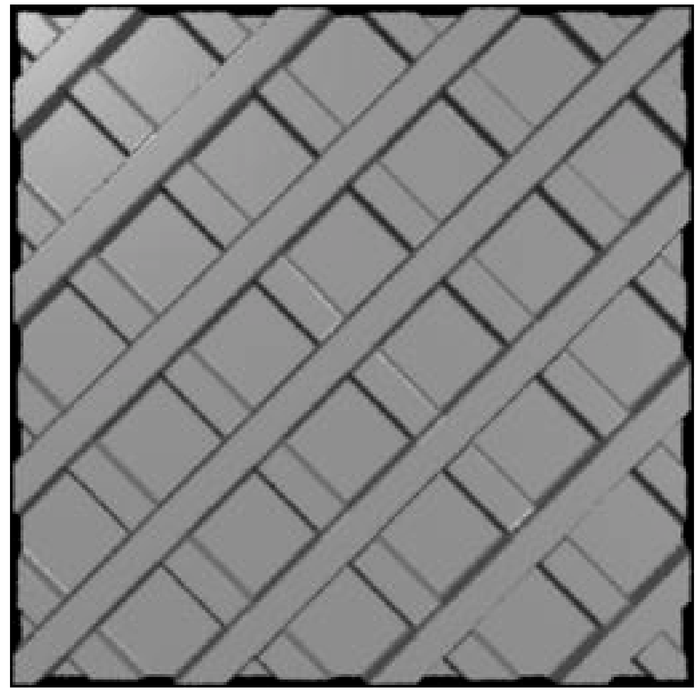
From RmanNotes
http://www.cgrg.ohio-state.edu/~smay/RManNotes/i

Surface detail, II

- Bumps
  - we assume that the surface has a set of bumps on it
    - e.g. the pores on an orange
  - these bumps are at a fine scale, so don't really affect the point of intersection, but do affect the normal
  - strategy:
    - obtain normal from "bump function"
    - shade using this modified normal
    - notice that some points on the surface may be entirely dark
    - bump maps might come from pictures (like texture maps)



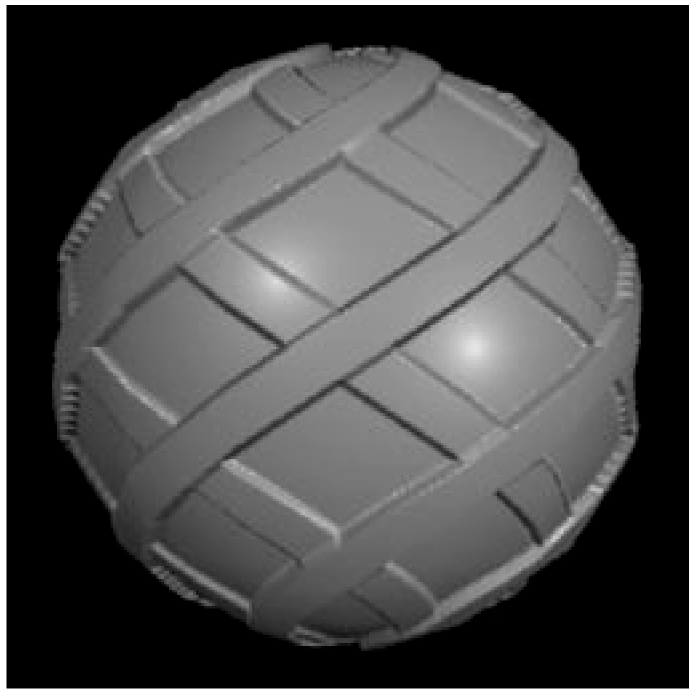




From RmanNotes
http://www.cgrg.ohio-state.edu/~smay/RManNotes/
x.html

Surface detail, III

- A more expensive trick is to have a map which includes **displacements** as well
- Must be done **before** visibility

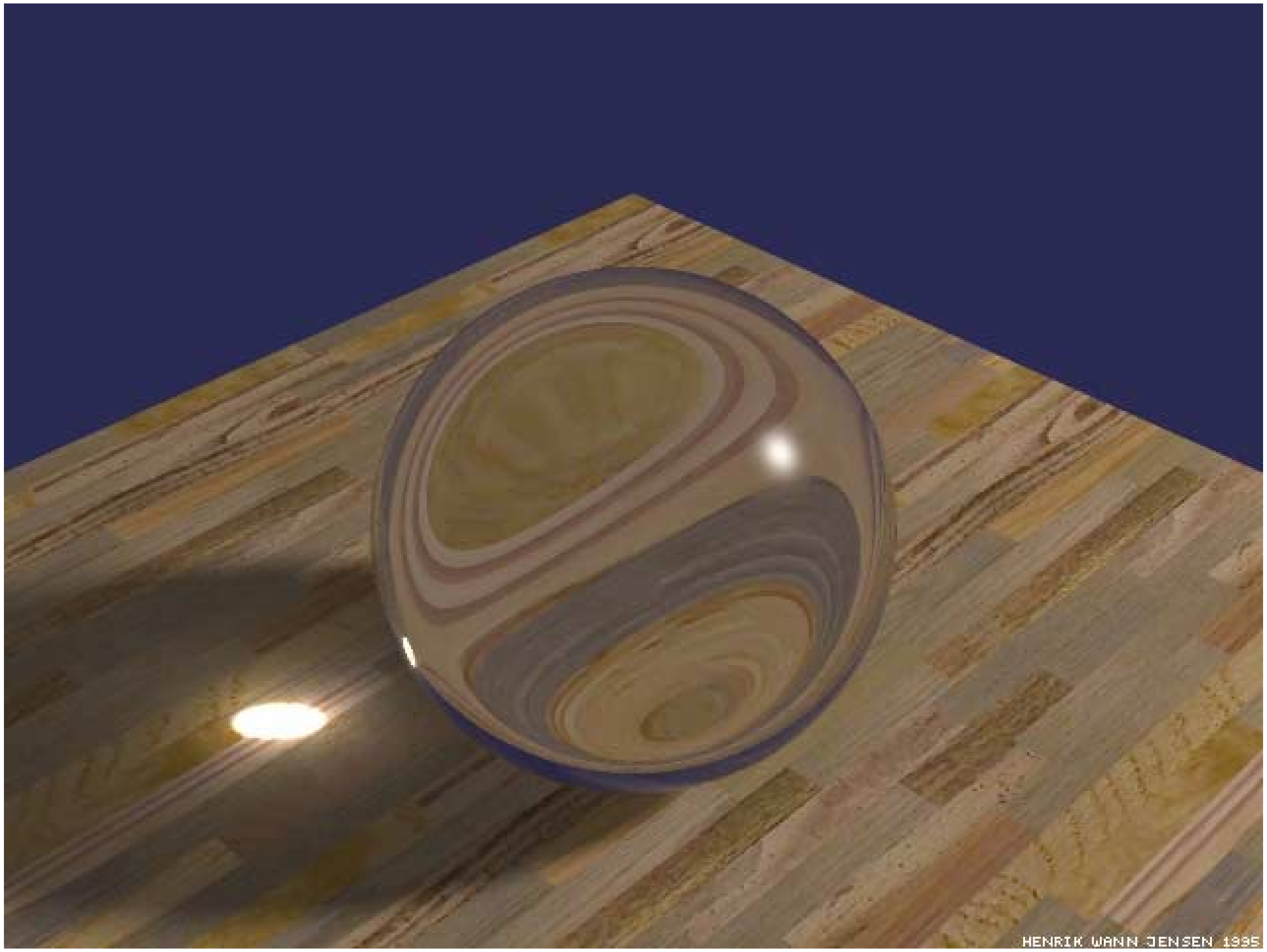


From RmanNotes
http://www.cgrg.ohio-state.edu/~smay/RManNotes/i

Illumination effects

- Caustics:
  - refraction or reflection causes light to be “collected” in some regions.
- Specular-> diffuse transfer
  - source reflected in a mirror
- Can’t render this by tracing rays from the eye - how do they know how to get back to the source?
- Instead, trace rays from the light to the first diffuse surface
  - leave a note that illumination has arrived - an illumination map, or photon map
  - now retrieve this note by tracing eye rays
- Issues
  - efficiency (why trace rays to things that might be invisible?)
  - aliasing (rays are spread out by, say, curved mirrors)

Refraction caustic



Henrik Jensen, http://www.gk.dtu.dk/~hwj