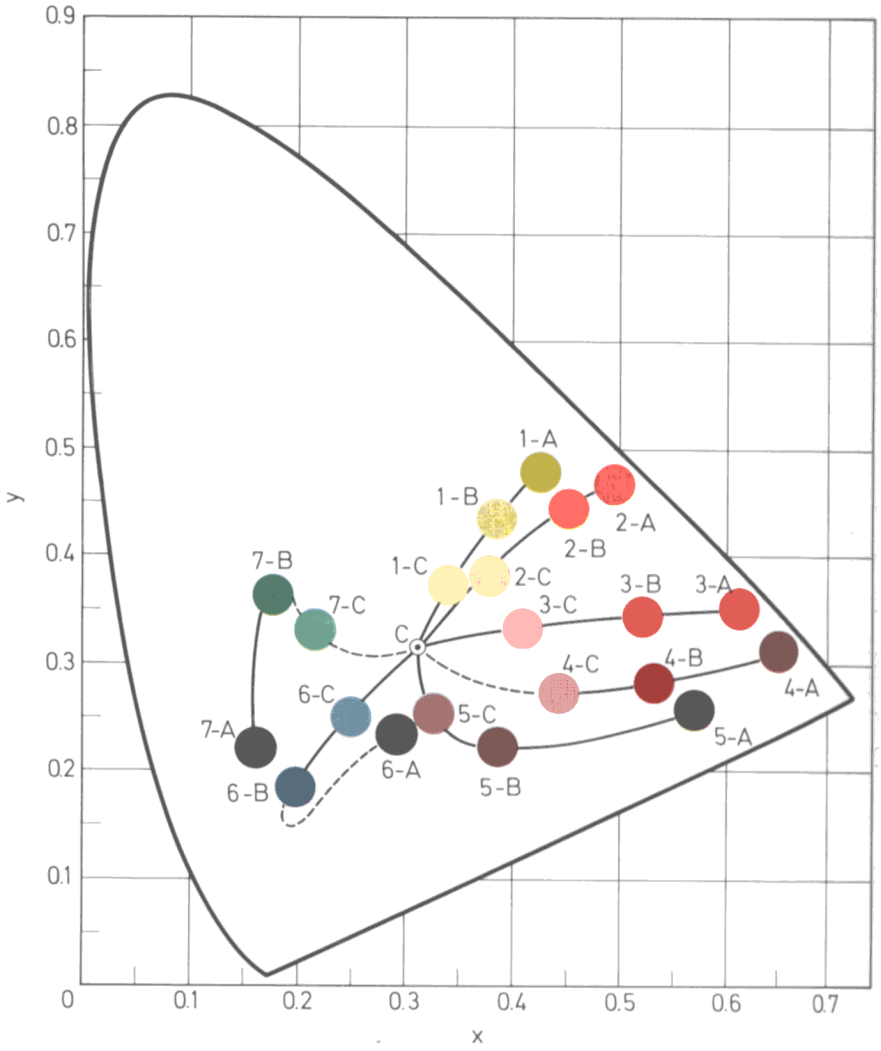
Subtractive mixing

- Inks subtract light from white, whereas phosphors glow.
- Linearity depends on pigment properties - often hugely non-linear.
- Inks: Cyan=White-Red, Magenta=White-Green, Yellow=White-Blue.
- For a good choice of inks, matching is not too from linear
- e.g.. C+M+Y=White-White=Black
  C+M=White-Yellow=Blue
- Usually require CMY and Black, because colored inks are more expensive, and registration is hard
- For good choice of inks, there is a approximate linear transform between XYZ and CMY

Mixing pigments in CIE



Color matching is linear, but combining pigments is not necessarily linear like mixing light .

Device independent colour imaging

- Problem: ensure that colours on a display, printer, etc. give the same experience that a viewer would have seeing relevant light spectra
- Difficulty: limited gamuts of most output devices
- Strategy: exploit a model of human experience
  - Simple model: The CIE XYZ matching paradigm
  - Being implemented in “Color Management Systems”
  - These try to relieve the user of the different color capabilities of devices
  - Complicated because every device needs to register properly with the CMS
- Deficiencies--as we have seen, the CIE systems does not count for spatial effects, illumination environments, etc., and these are important
- Some progress is being made but the models tend to be complicated

Shading values for colored surfaces

- Simplest:
  - Use appropriate shading model in 3 channels, instead of one
  - Implies red albedo, green albedo, blue albedo, etc.
  - Works because the shading model is independent of wavelength.
  - Can lead to somewhat inaccurate colour reproduction in some cases - particularly coloured light on coloured surfaces
- Better
  - Use appropriate shading model at many different wavelength samples - 7 is usually enough
  - Estimate receptor response in eye using sum over wavelength
  - Set up pixel value to generate that receptor response

Recursive ray tracing (chapter 14)

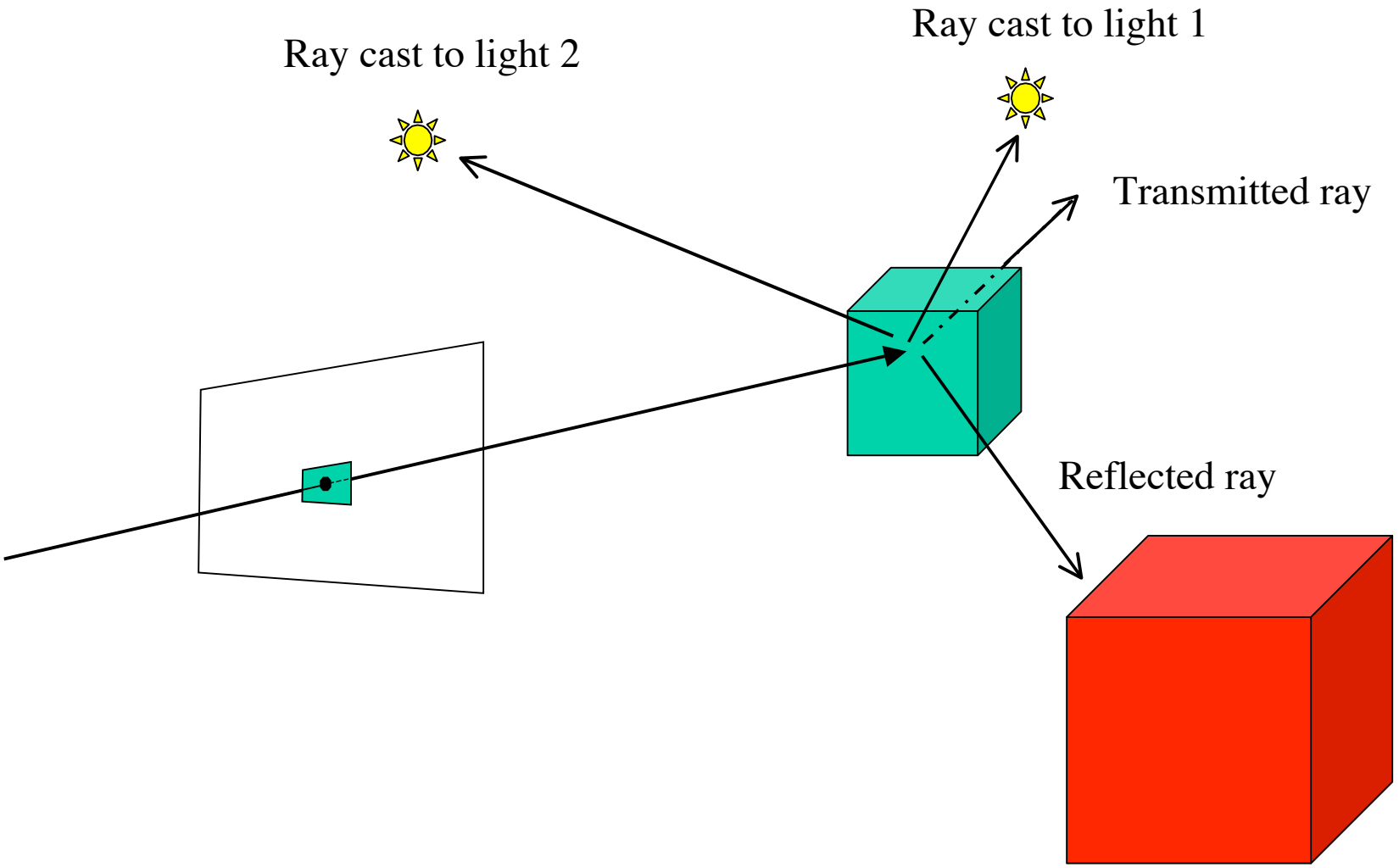
- Pixel brightness =

  radiance along ray to pinhole =

  β_l(diffuse and specular lobe)

  \+ β_m(reflected)

  \+ β_t(transmitted)
- Local component, β_l:
  - from sources alone (local shading model) typically Lambertian with a Phong type specular model
  - typically we add some global illumination (“ambient”)
- We cast on ray to the center of each light (shadow ray): To compute the Penumbra and specular lobe reflection properly, we would have to shoot lots of addtional rays---expensive!
- Reflected component is due to radiance along ray from intersection along mirror direction (often referred to as specular ray, but this is not to be confused with the specular lobe already mentioned)
- Transmitted component is due to radiance along ray from intersection along transmitted direction


Ray cast to light 2
Ray cast to light 1
Transmitted ray
Reflected ray

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 (diffuse and for specular lobe)
  - along reflected direction = reflected ray
  - along transmitted dir = refracted ray
- Determine each component and add them up with contribution from ambient illumination.
- To determine some of the components, the ray tracer must be called **recursively**.

Recursive ray tracing rendering (cont)

- Recursion needs to stop at some point!
- Contributions die down after multiple bounces---there is no such thing as a perfect reflector---so we either set mirror reflections to be less than 100% (even if the user asks for 100%), or simply include an attenuation factor for each new ray.
- Can also model absorbtion due to light traveling in medium
  - Usually ignored in air, but depends on the application
  - Translucent absorption is exponential in depth

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

- Recursion is stopped when contributions are too small
  - need to track the cumulative effect
  - common to also limit the depth explicitly

Mechanics

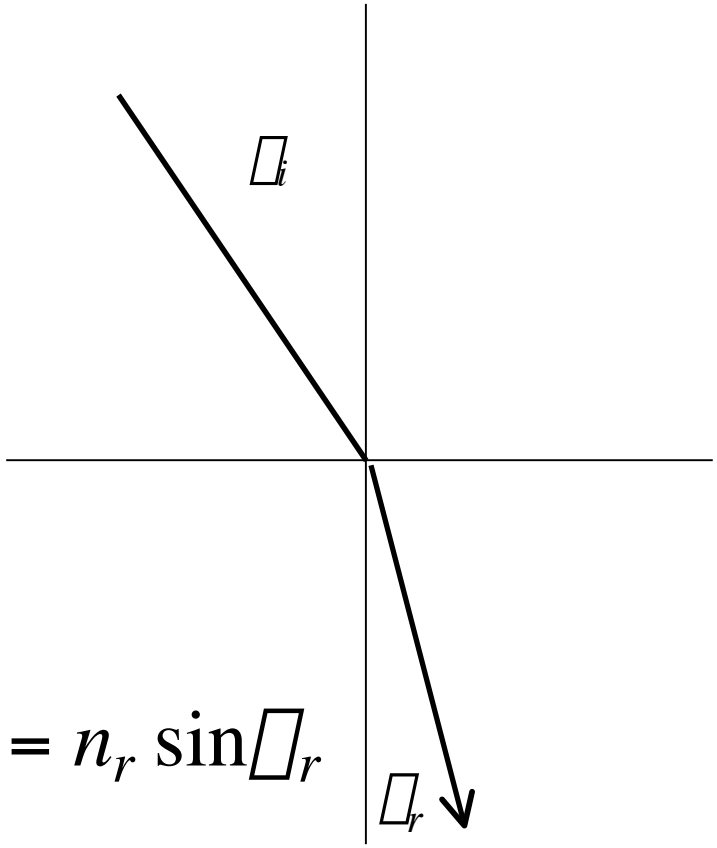
- Primary issue is intersection computations.
  - E.g. sphere, triangle.
- Polygon (should feel familiar!)
- Find point on plane of polygon and then determine if it is inside
  - One way is to make an argument with angles
  - Another way---thinking of the polygon as a surface of a polyhedra---is to check if the point is on the inside side of each of the other planes of the polyhedra.
- Sphere (see book chapter 13.4.1)

Refraction Details

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

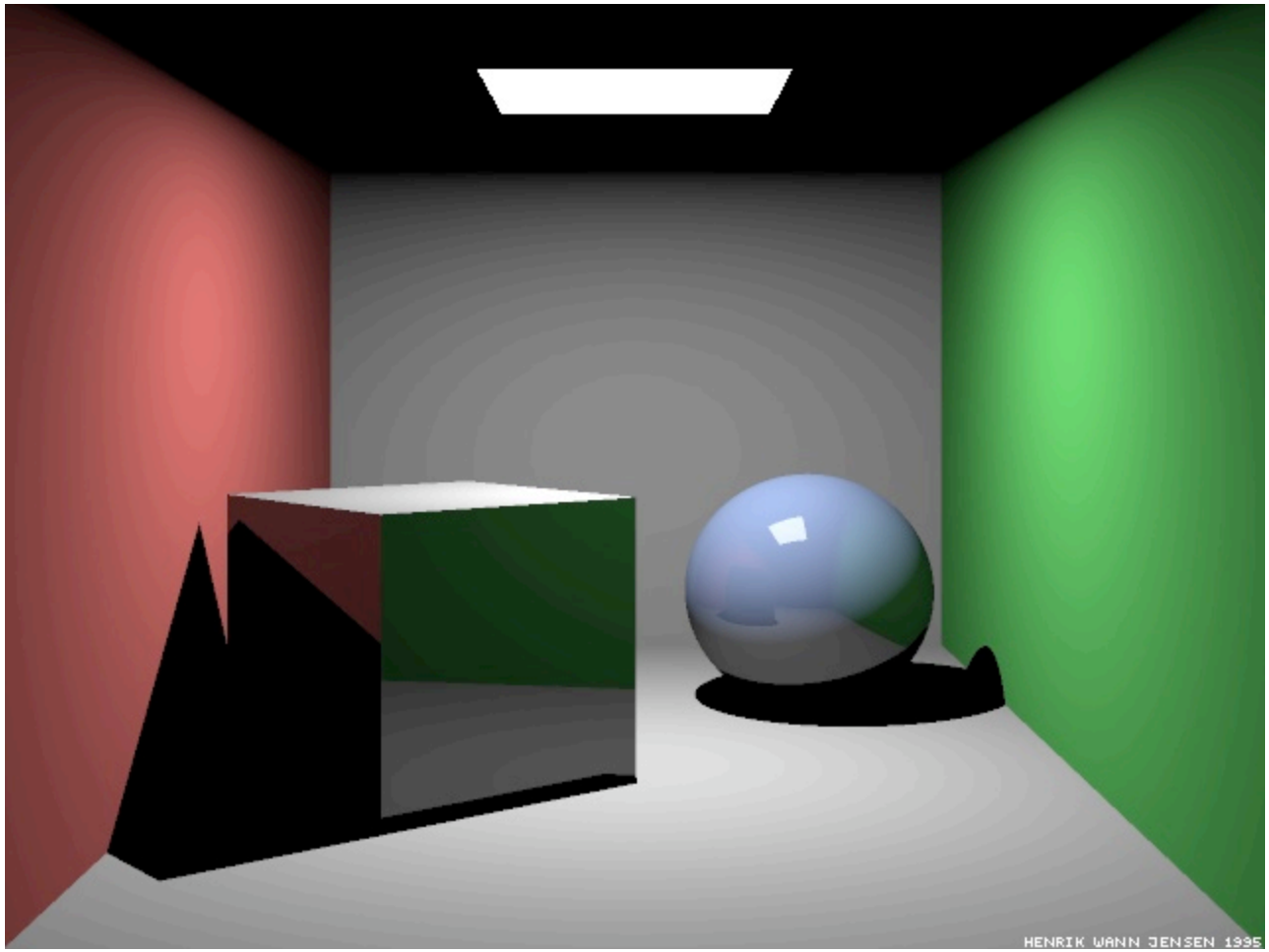
Typical values:

air:	1.00 (nearly)
water:	1.33
glass:	1.45-1.6
diamond:	2.2



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

The indicies of refraction for the two media, and the incident angle, θ_i, yield the refracted angle θ_r.



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

PCKTWTCH by Kevin Odhner, POVRay

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

Issues

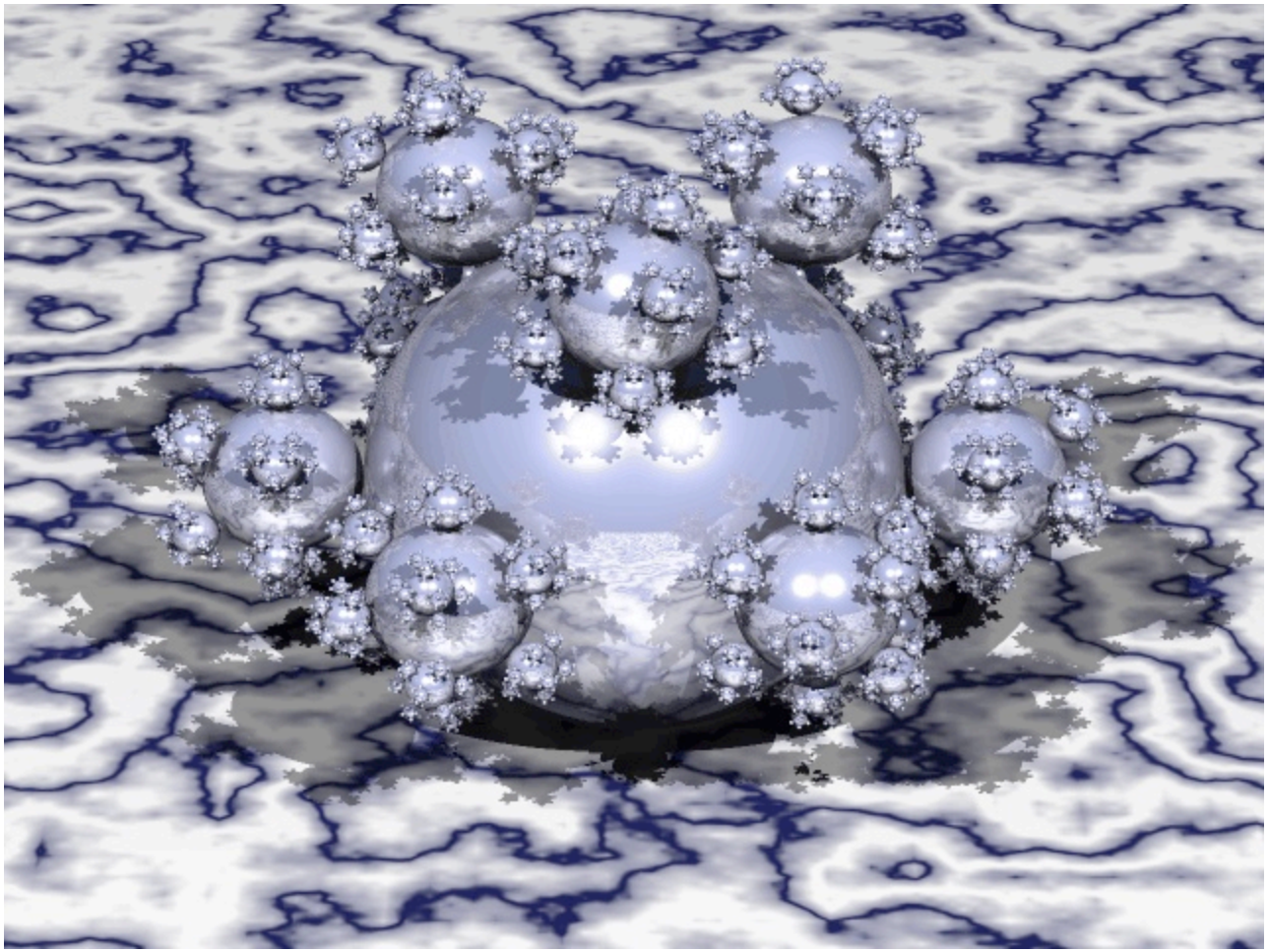
- Sampling (aliasing)
- Very large numbers of objects
  - Need making intersections efficient, exclude as much as possible using clever data structures
- Surface detail
  - bumps, texture, etc.
- Illumination effects
  - Caustics, specular to diffuse transfer
- Camera models

Sampling

- Simplest ray-tracer is one ray per pixel
  - This gives aliasing problems
- Solutions
  - Cast multiple rays per pixel, and use a weighted average
  - 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.dtu.dk/~hwj