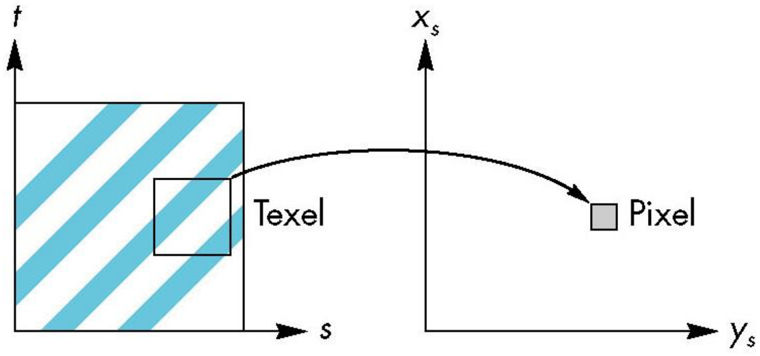
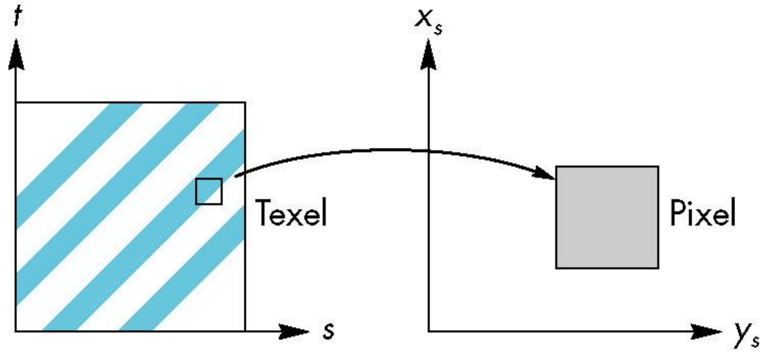
Texture Mapping II

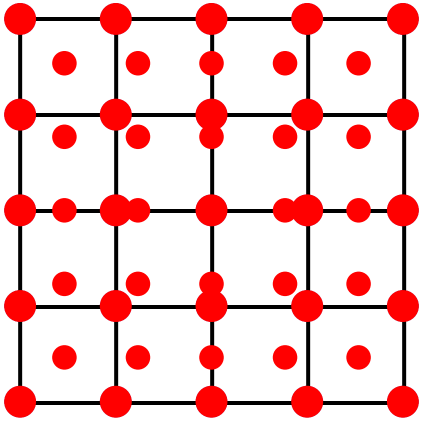
Slides from constructed from various Web sources

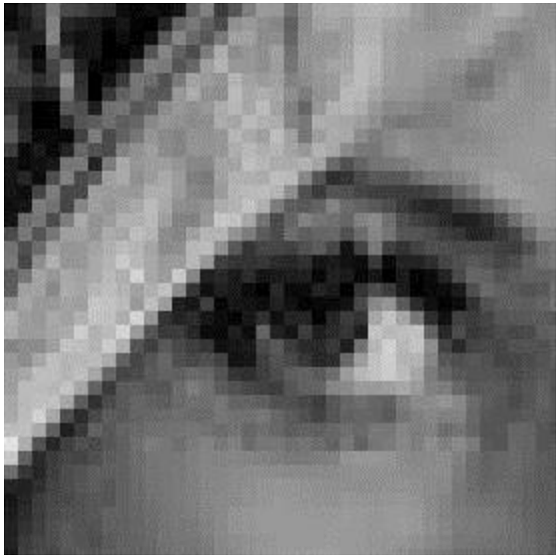
Sampling Issues





Interpolation



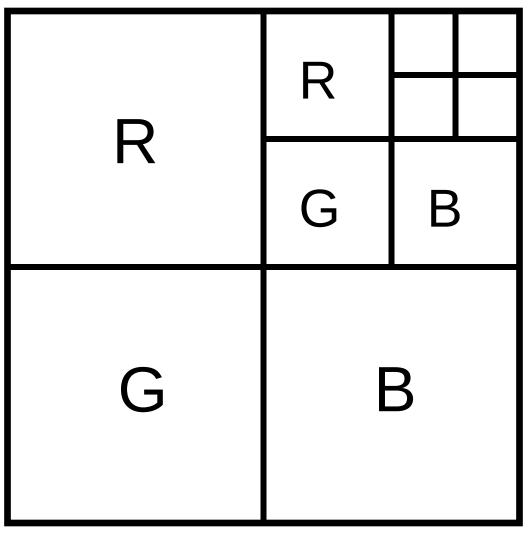


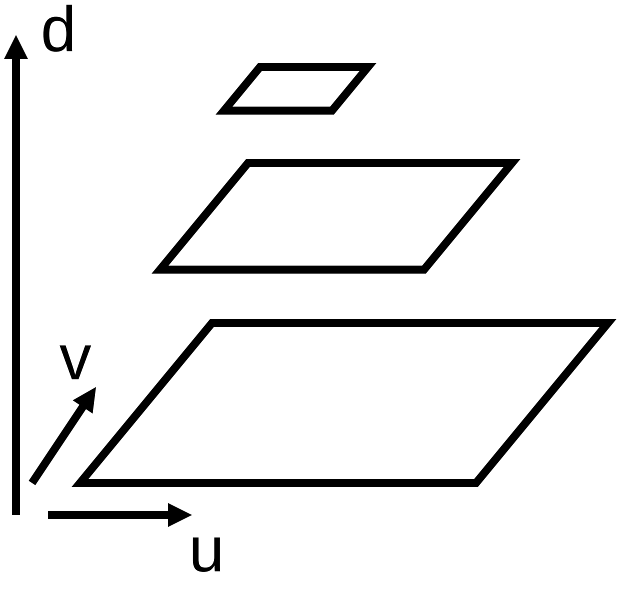
Nearest neighbor

Linear Interpolation

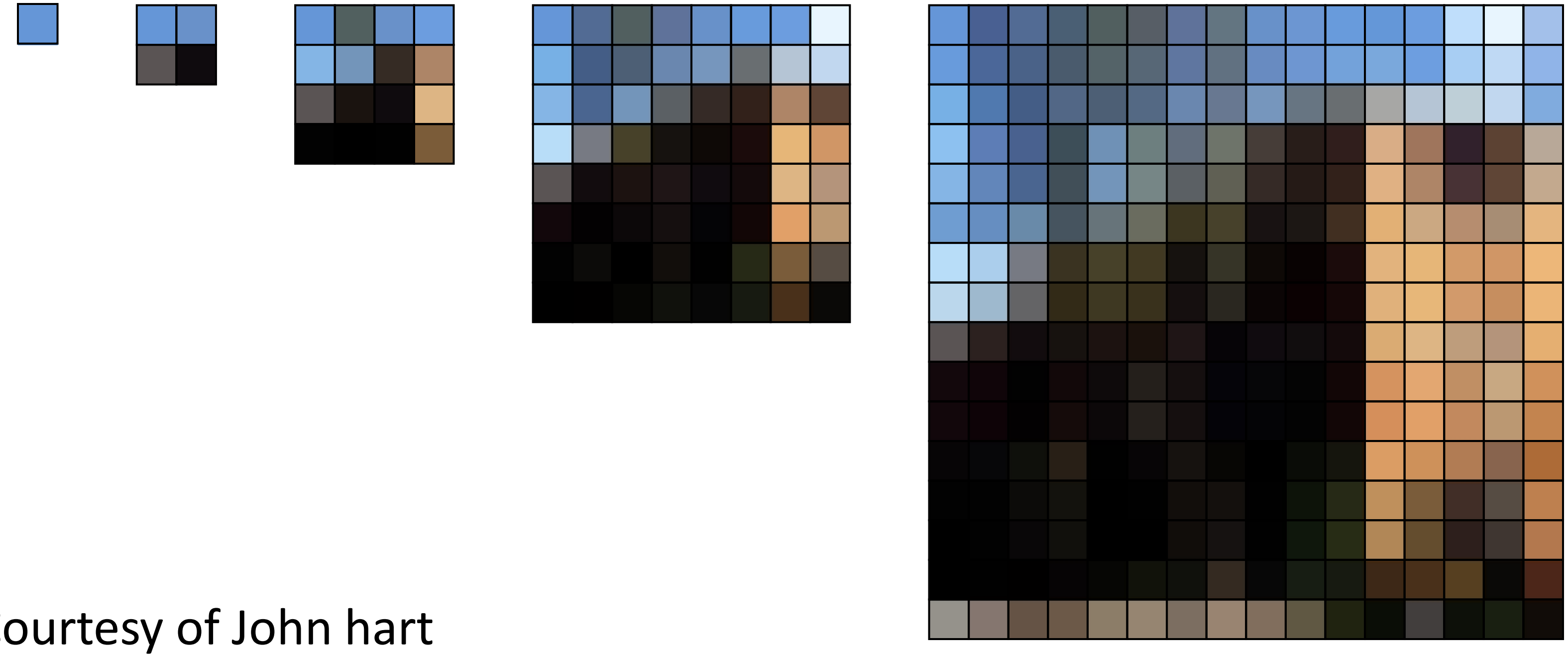
Mip Mapping [Williams]

MIP = Multim In Parvo = Many things in a small place





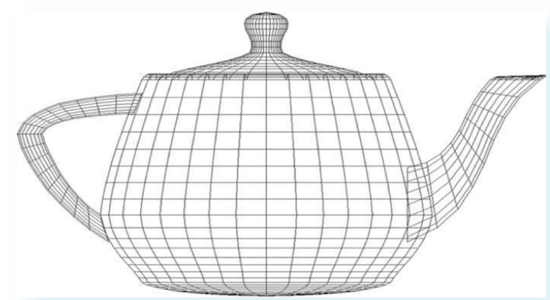
Mip Mapping - Example

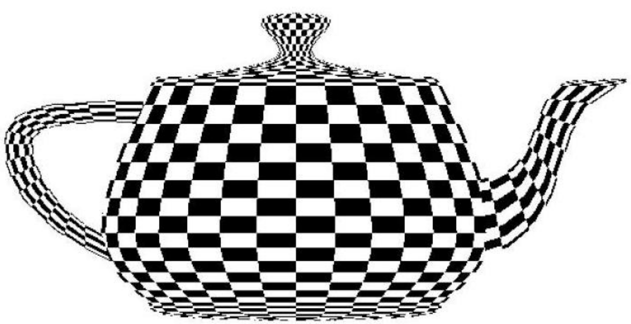


Courtesy of John hart

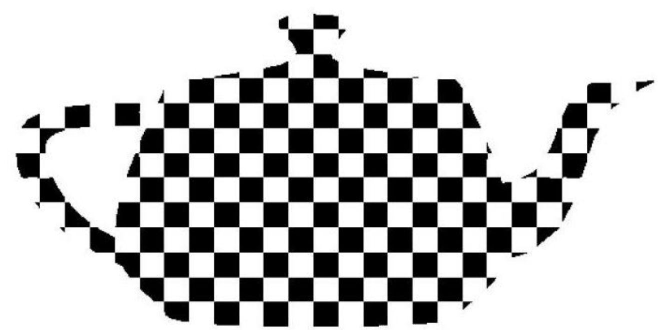
Assigning Texture Coordinates

- We generally want an even sharing of texels (pixels in the texture) across all triangles
- But what about this case?
  – Want to texture the teapot:





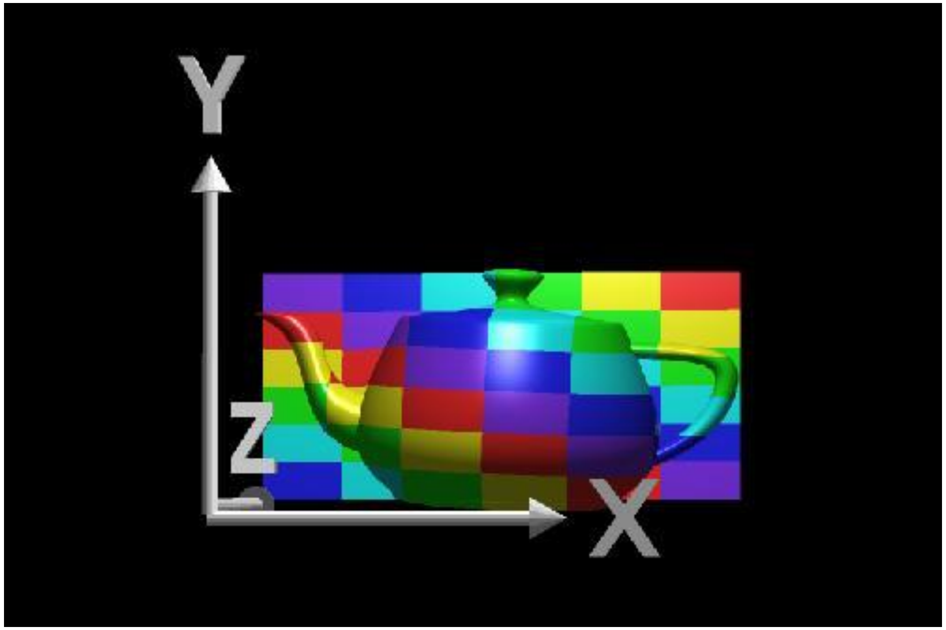
Do we want this?

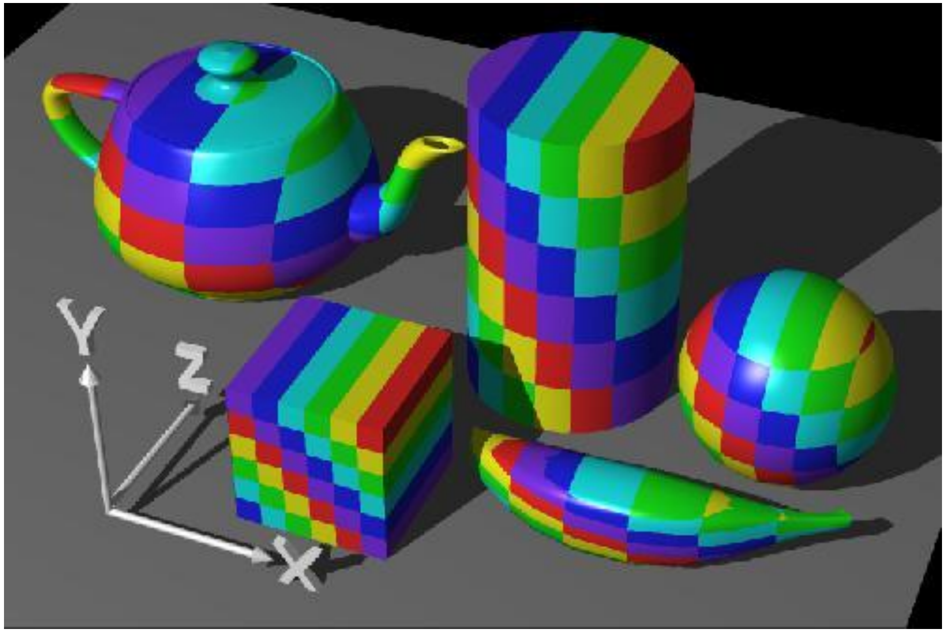


Or this?

Planar Mapping

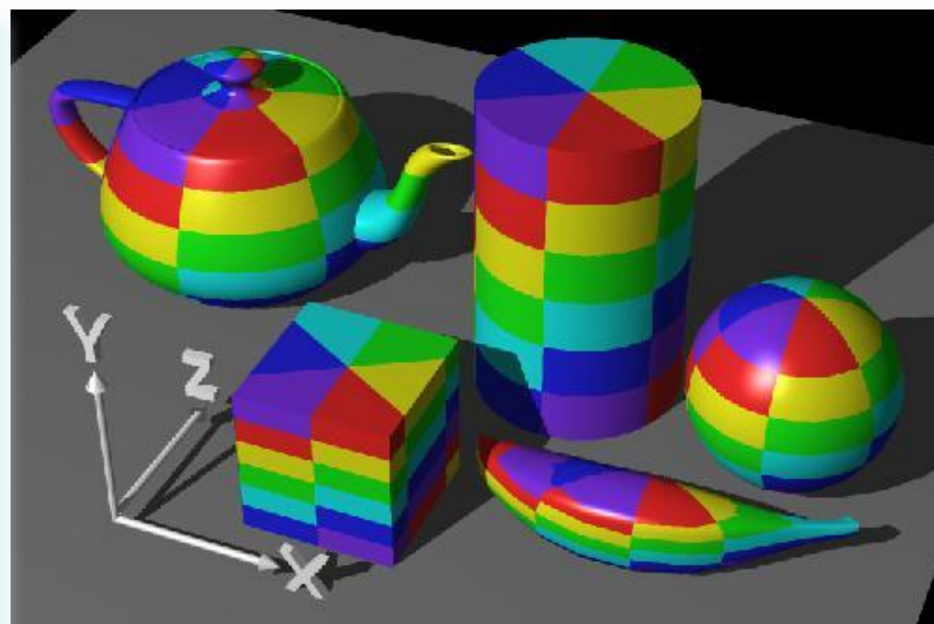
- Just use the texture to fill all of space
  - Same color for all z-values
  - $(u, v) = (x, y)$





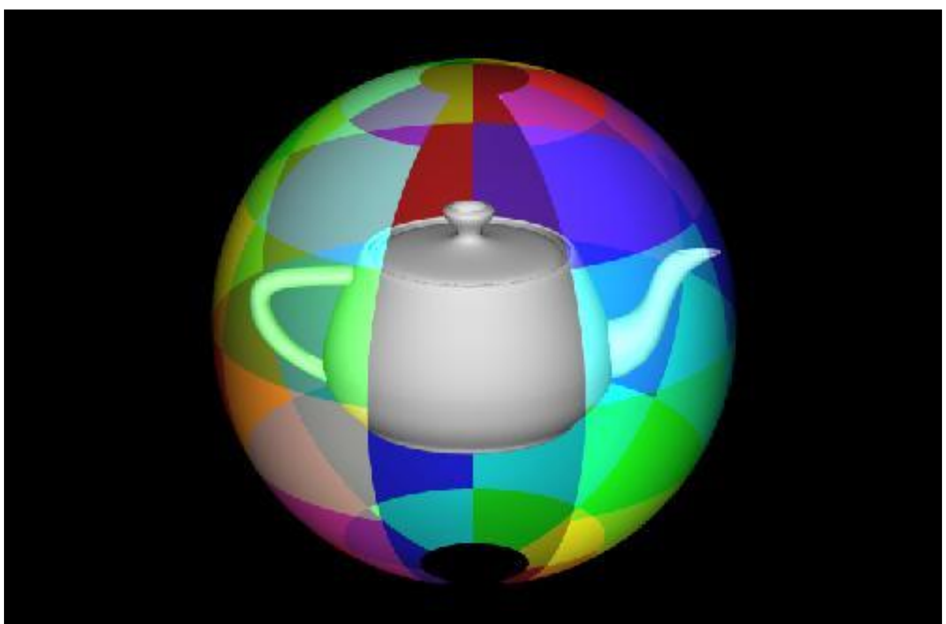
Cylindrical Mapping

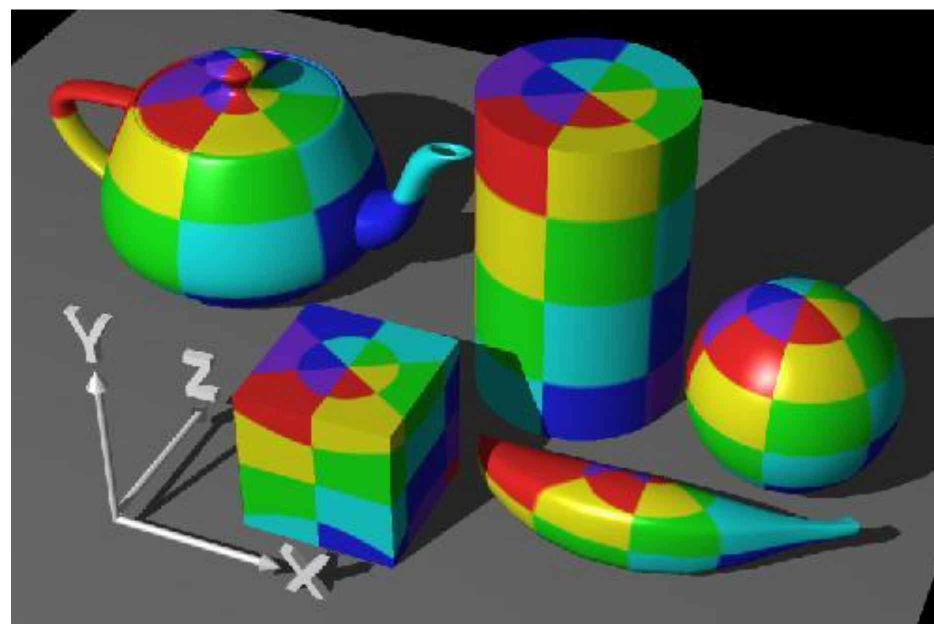
- "Wrap" the texture around your object
  - Like a coffee can
  - Same color for all pixels with the same angle
    - $u = \theta / 2\pi$
      $v = y$



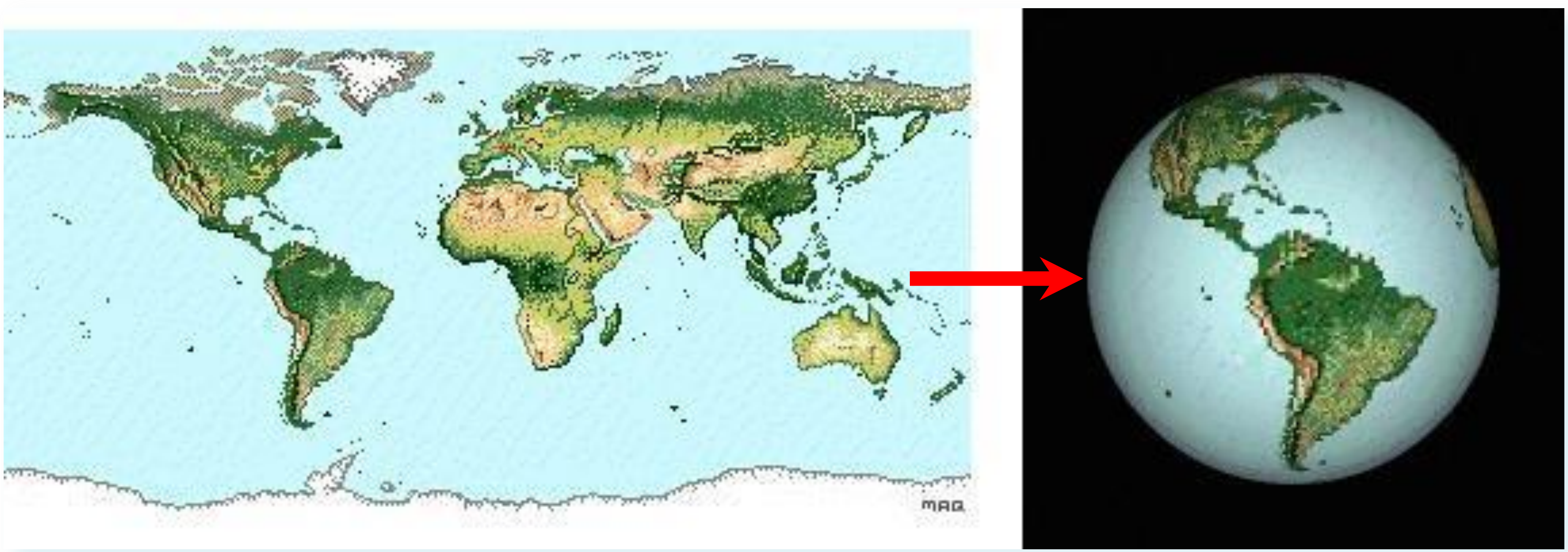
Spherical Mapping

- "Wrap" the texture around your object
  - Like a globe
  - Same color for all pixels with the same angle
    - $u = \varphi / 2\pi$
      $v = (\pi - \theta) / \pi$



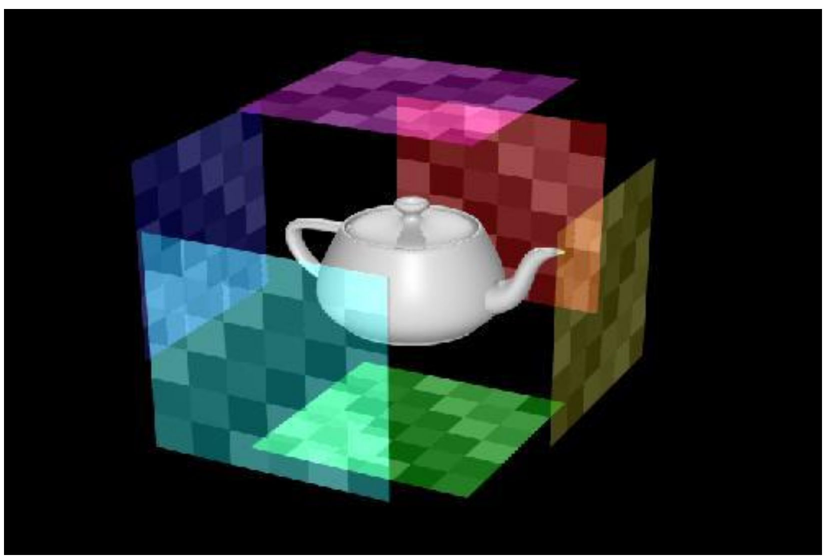


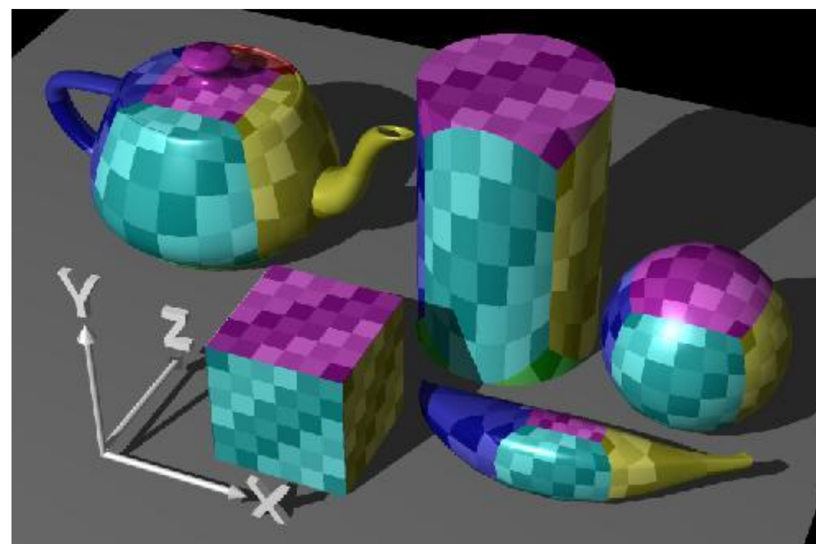
Spherical Mapping Example



Cube Mapping

- Not quite the same as the others
  - Uses multiple textures (6, to be specific)
- Maps each texture to one face of a cube surrounding the object to be textured
  - Then applies a planar mapping to each face

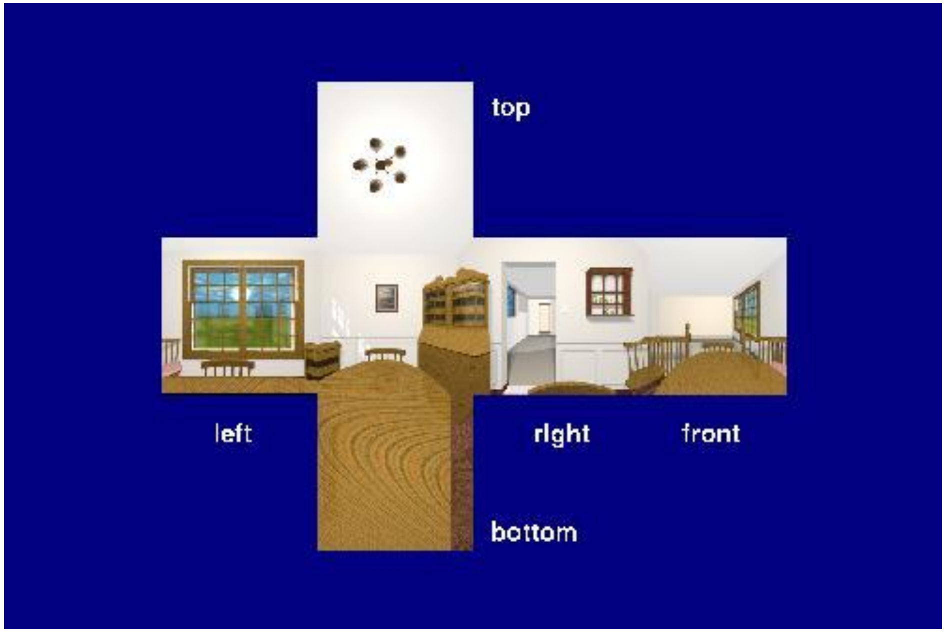




Environment Maps

- Cube mapping is commonly used to implement <u>environment maps</u>
- This allows us to “hack” reflection
  - Render the scene from the center of the cube in each face direction
  - Store each of these results into a texture
  - Then render the scene from the actual viewpoint, applying the environment textures

Cube Environment Map



Sphere Environment Map

Spatially variant resolution

Bump Mapping

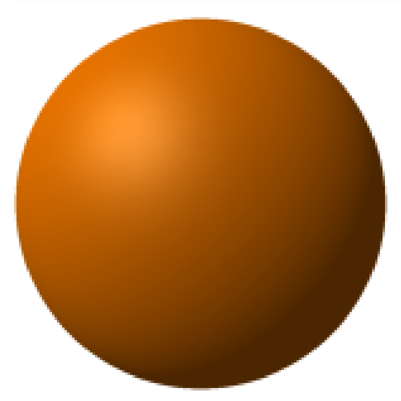
- How do we get this?
  – The underlying model is just a sphere

Bump Mapping

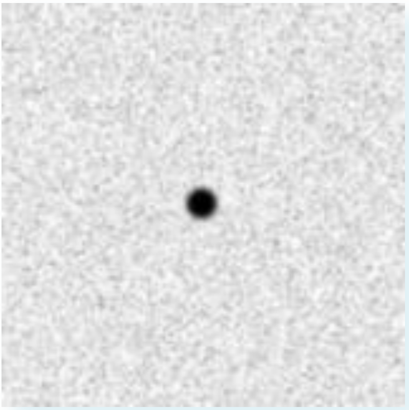
- Requires per-pixel (Phong) shading
- Just interpolating from the vertex normals gives a smooth-looking surface
- Bump mapping uses a “texture” to define how much to perturb the normal at that point
  - Results in a “bumpy” surface

Bump Mapping

Note: Silhouette doesn't change



Rendered Sphere

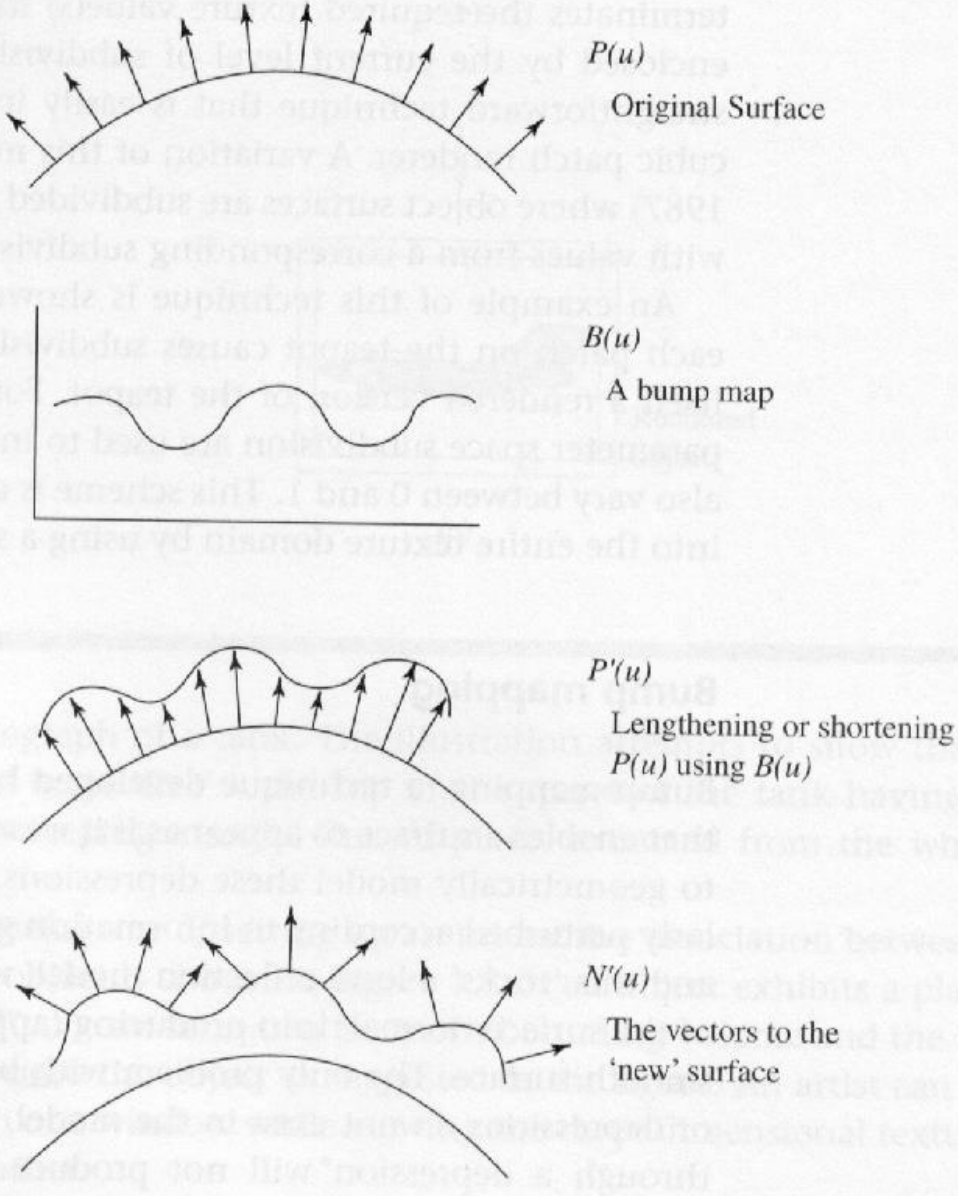


Bump Map

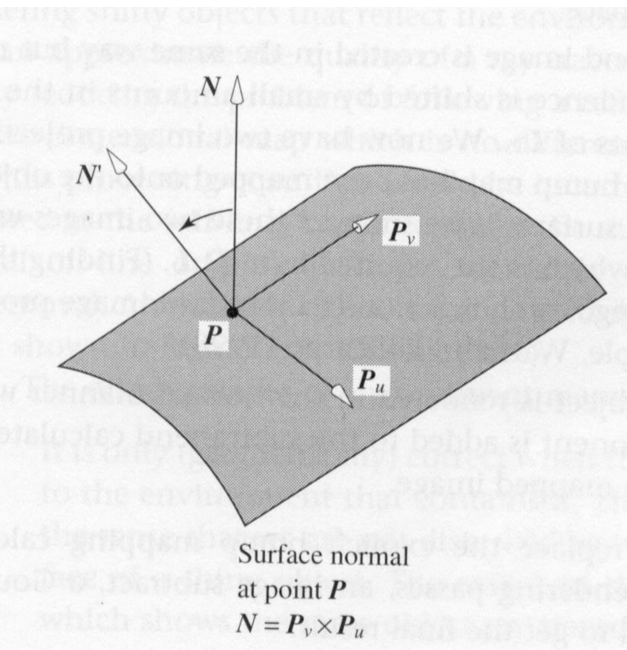
Bump Mapped Sphere

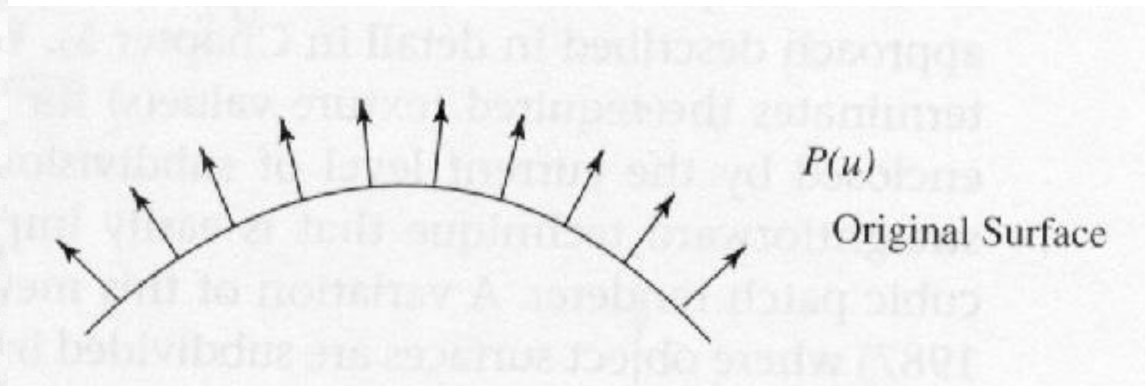
- At each point on the surface:
  - Do a look-up into the bump map "texture"
  - Perturb the normal slightly based on the "color"
    - Note that "colors" are actually just 3- or 4-vectors

Images from Wikipedia


P(u)
Original Surface
B(u)
A bump map
P'(u)
Lengthening or shortening
P(u) using B(u)
N'(u)
The vectors to the
'new' surface

Bump Mapping

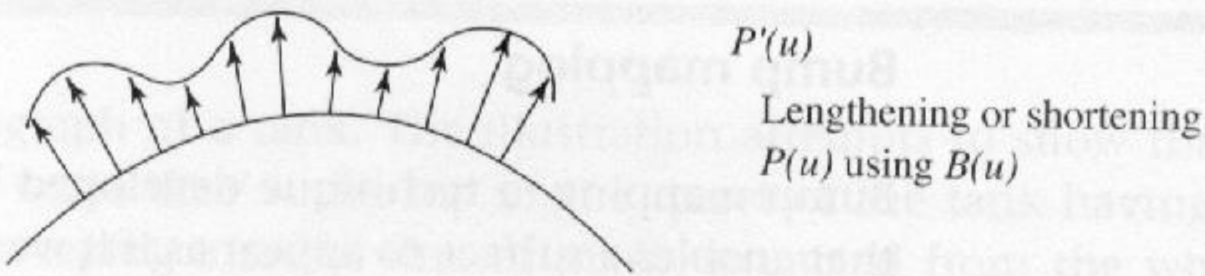




$$\boldsymbol{N} = \frac{\partial \boldsymbol{P}}{\partial u} \times \frac{\partial \boldsymbol{P}}{\partial v}$$

$$= \boldsymbol{P}_{\mathrm{u}} \times \boldsymbol{P}_{\mathrm{v}}$$

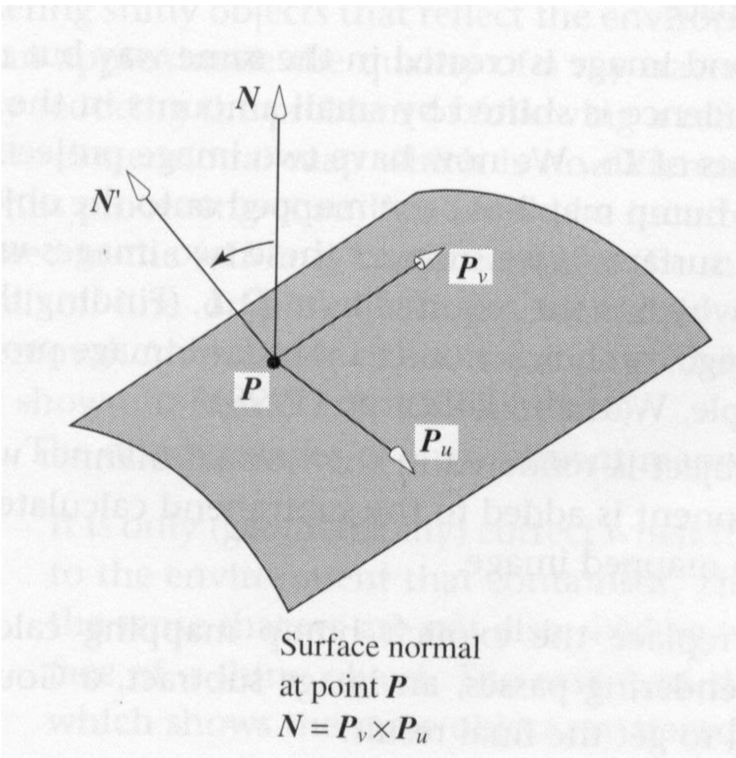
Bump Mapping



$$\mathbf{P}'(u,v) = \mathbf{P}(u,v) + B(u,v)\mathbf{N}$$

New Imaginary Surface P’

Bump Mapping



$$\mathbf{N}' = \mathbf{P}'_u \times \mathbf{P}'_v$$

$$\mathbf{P}'_u = \mathbf{P}_u + B_u\mathbf{N} + B(u,v)\mathbf{N}_u$$

$$\mathbf{P}'_v = \mathbf{P}_v + B_v\mathbf{N} + B(u,v)\mathbf{N}_v$$

Bump Mapping

$$\boldsymbol{N}' = \mathbf{N} + B_u\mathbf{N} \times \boldsymbol{P}_v + B_v\boldsymbol{P}_u \times \mathbf{N}$$

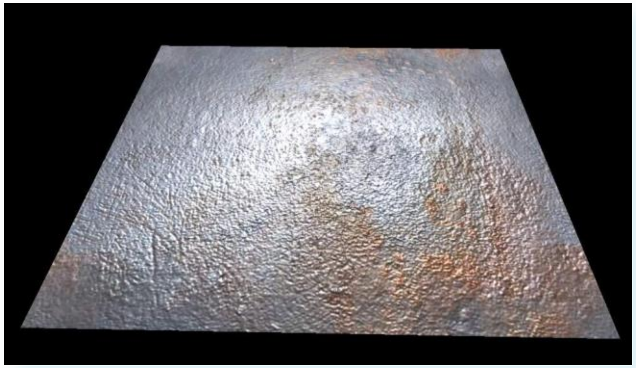
or

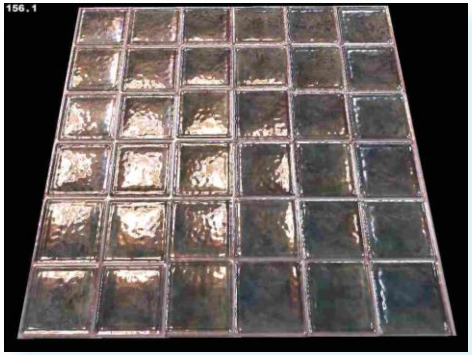
$$\begin{aligned}\boldsymbol{N}' &= \mathbf{N} + B_u\mathbf{N} \times \boldsymbol{P}_v - B_v\mathbf{N} \times \boldsymbol{P}_u \\ &= \mathbf{N} + (B_u\mathbf{A} - B_v\boldsymbol{B}) \\ &= \mathbf{N} + \boldsymbol{D}\end{aligned}$$

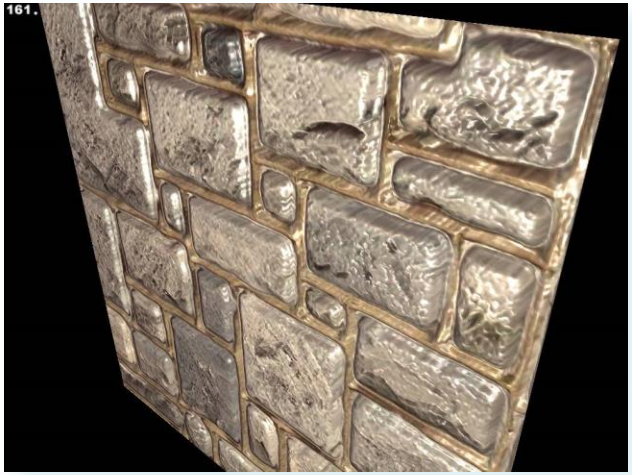
$\boldsymbol{D}$ is given by

$$\boldsymbol{D} = B_u\,\boldsymbol{A} - B_v\,\boldsymbol{B}$$

More Bump Mapping Examples







Displacement Mapping

- Bump mapping: use texture map to perturb surface normal
- Displacement mapping: use texture map to displace surface (perturb 3D shape)

Displacement Mapping

Bump Mapping

Displacement Mapping

Displacement Mapping

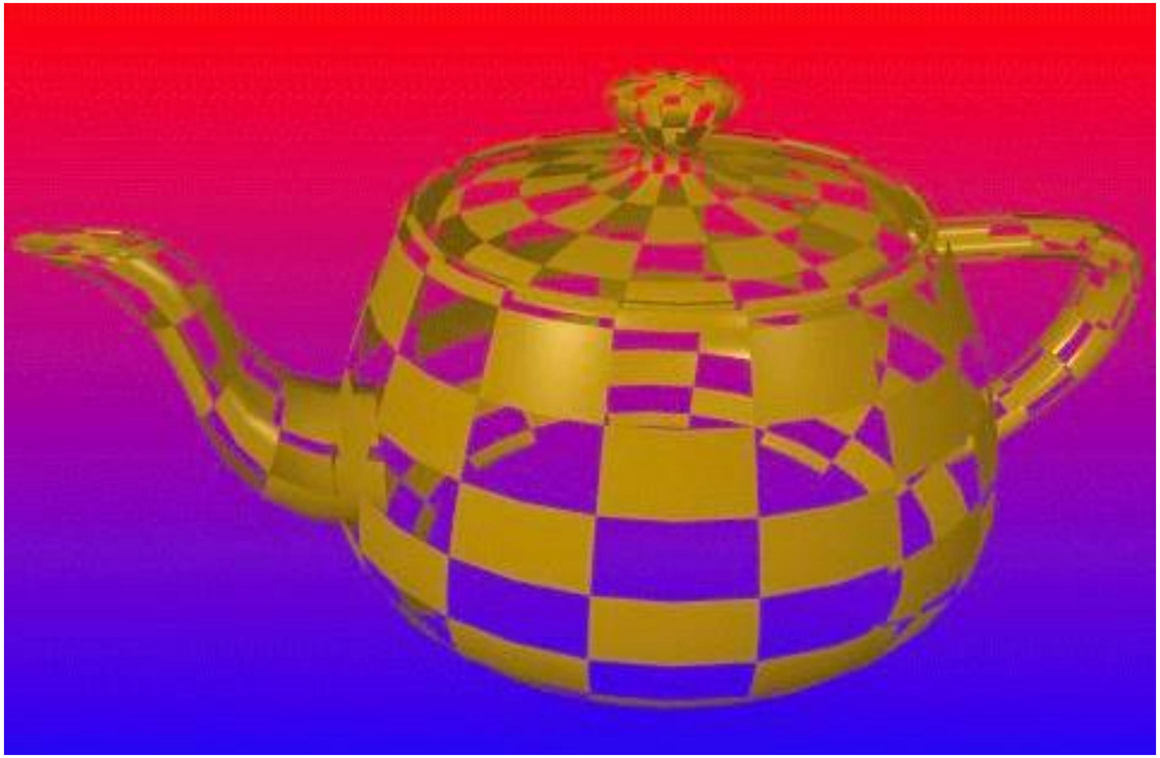
- Displacement mapping shifts all points on the surface in or out along their normal vectors
  - Assuming a displacement texture *d*,

    $$\mathbf{p}' = \mathbf{p} + d(\mathbf{p}) * \mathbf{n}$$

- Note that this actually changes the vertices, so it needs to happen in geometry processing

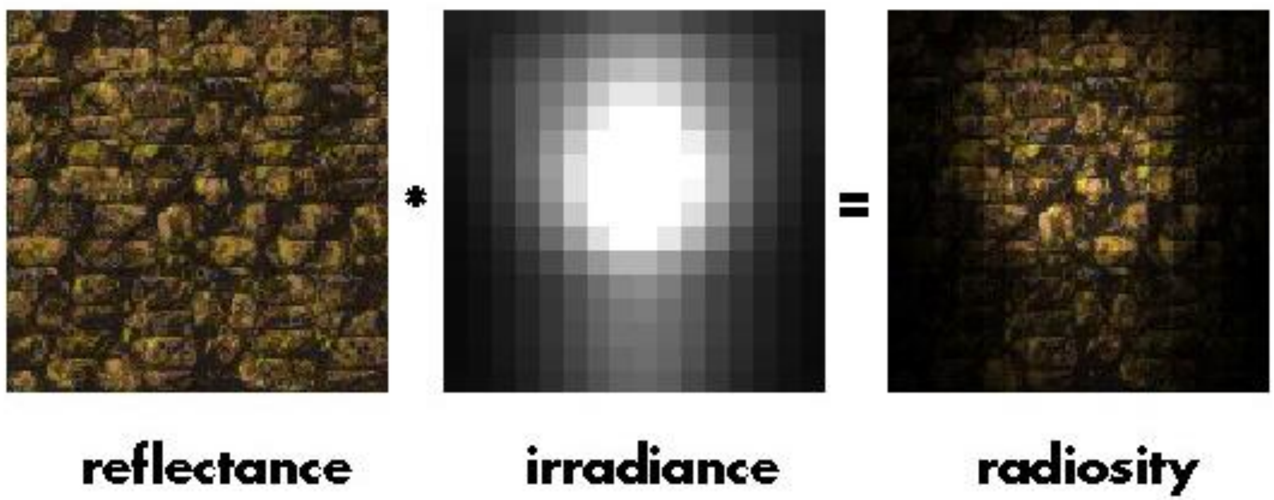
Opacity Maps

Use texture to represent opacity



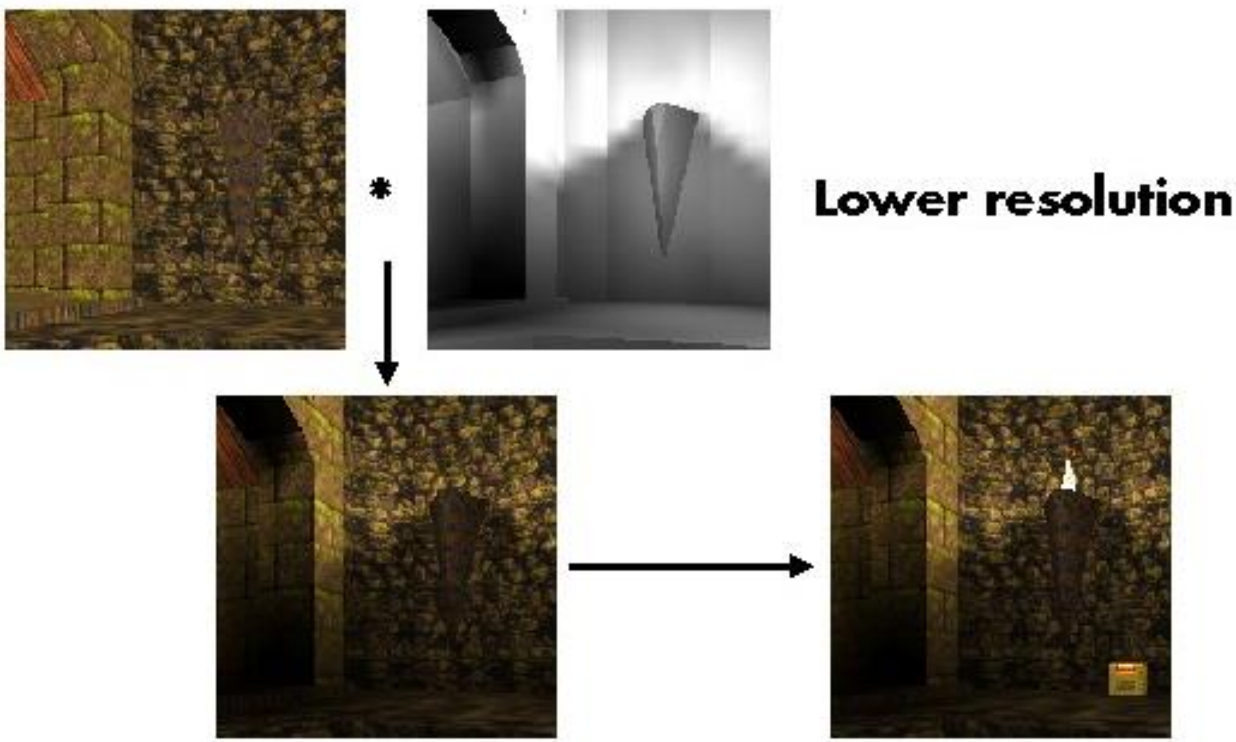
Illumination Maps

Use texture to represent illumination footprint



Illumination Maps

Quake light maps



Ray Tracing

Slides from constructed from various Web sources

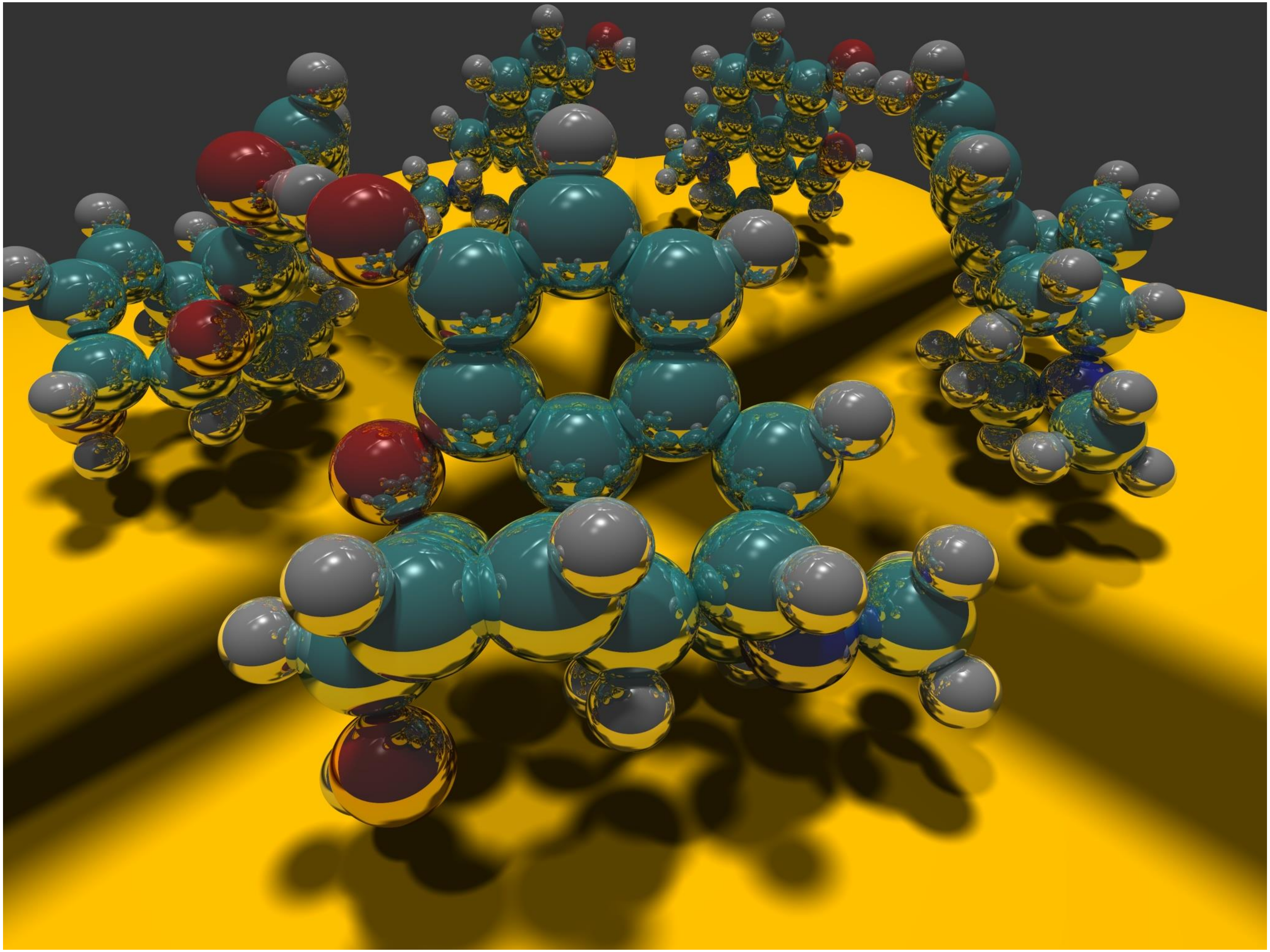
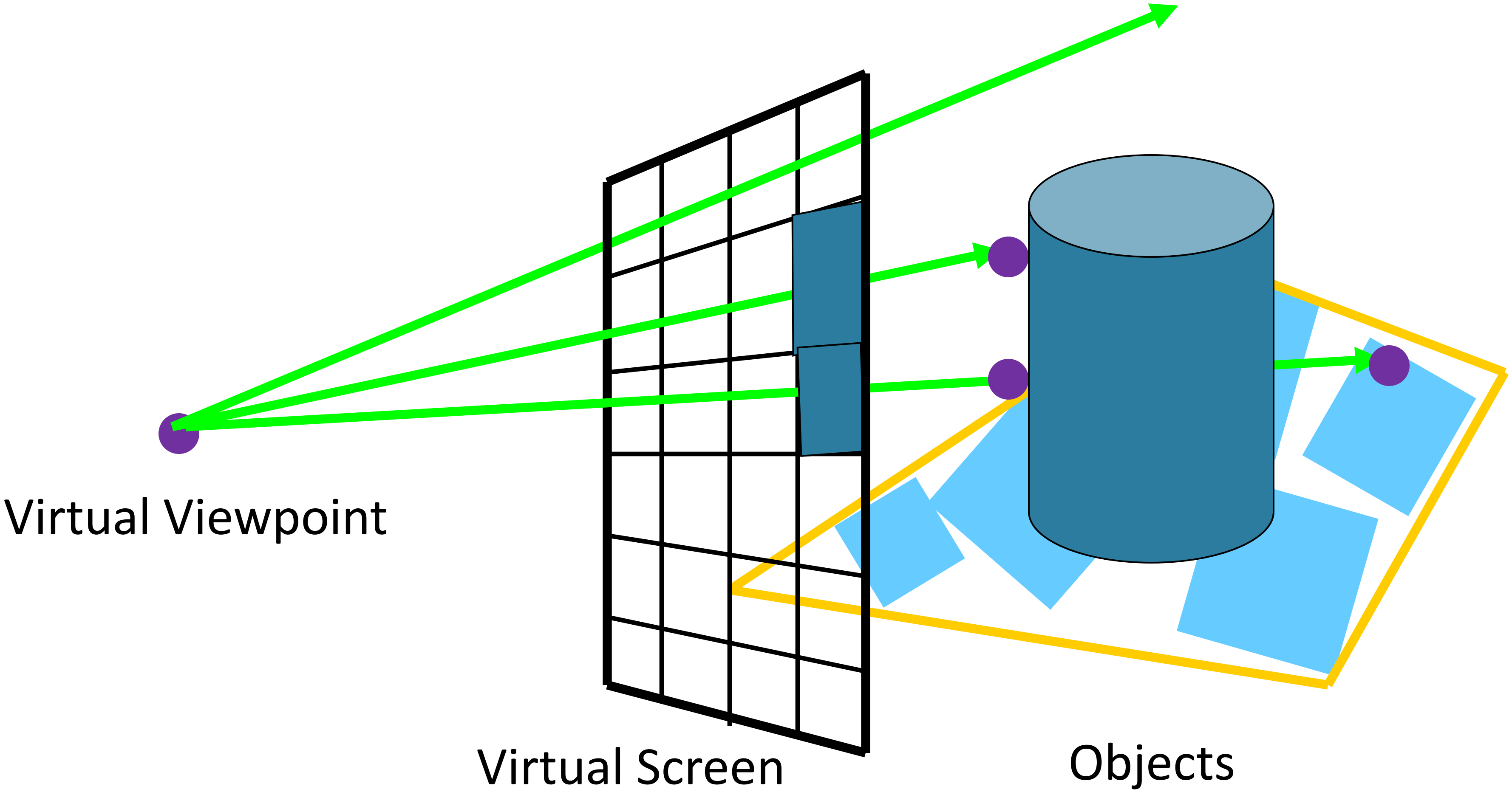


Image courtesy Paul Heckbert 1983

Publicly available Ray Tracer

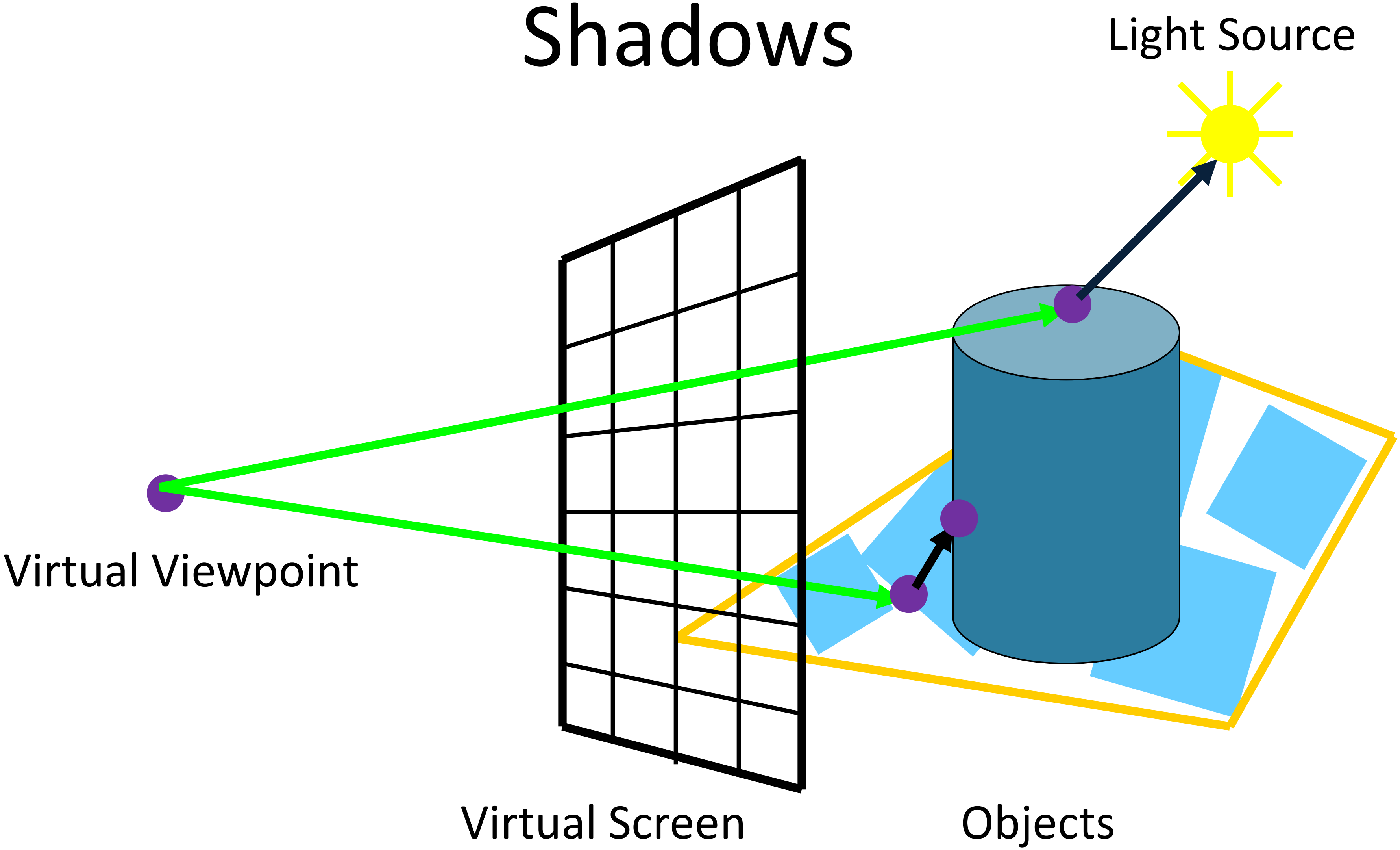
http://www.povray.org/

Ray Casting



Ray misses all objects: Pixel colored black
Ray intersects object: shade using color, lights, materials
Multiple intersections: Use closest one

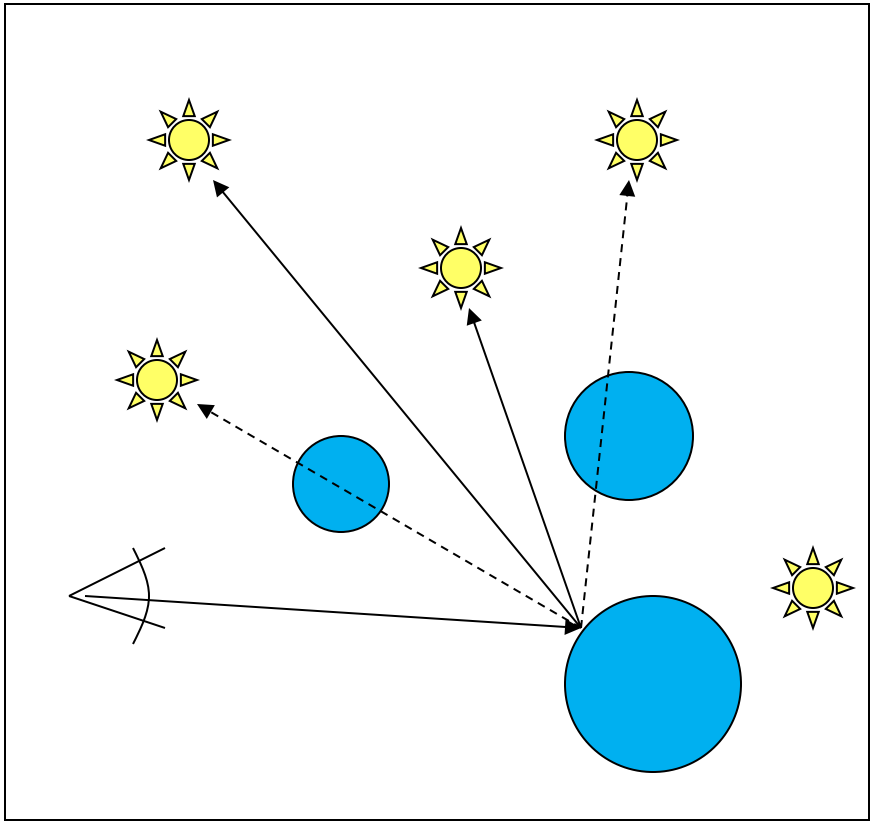
Shadows



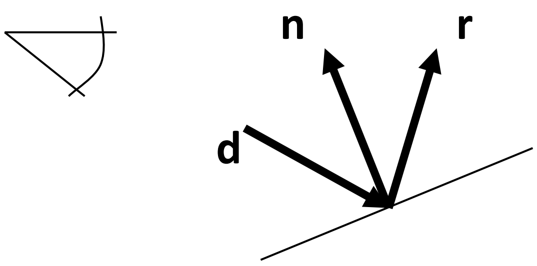
Shadow ray to light is unblocked: object visible

Shadow ray to light is blocked: object in shadow

Shadow Rays

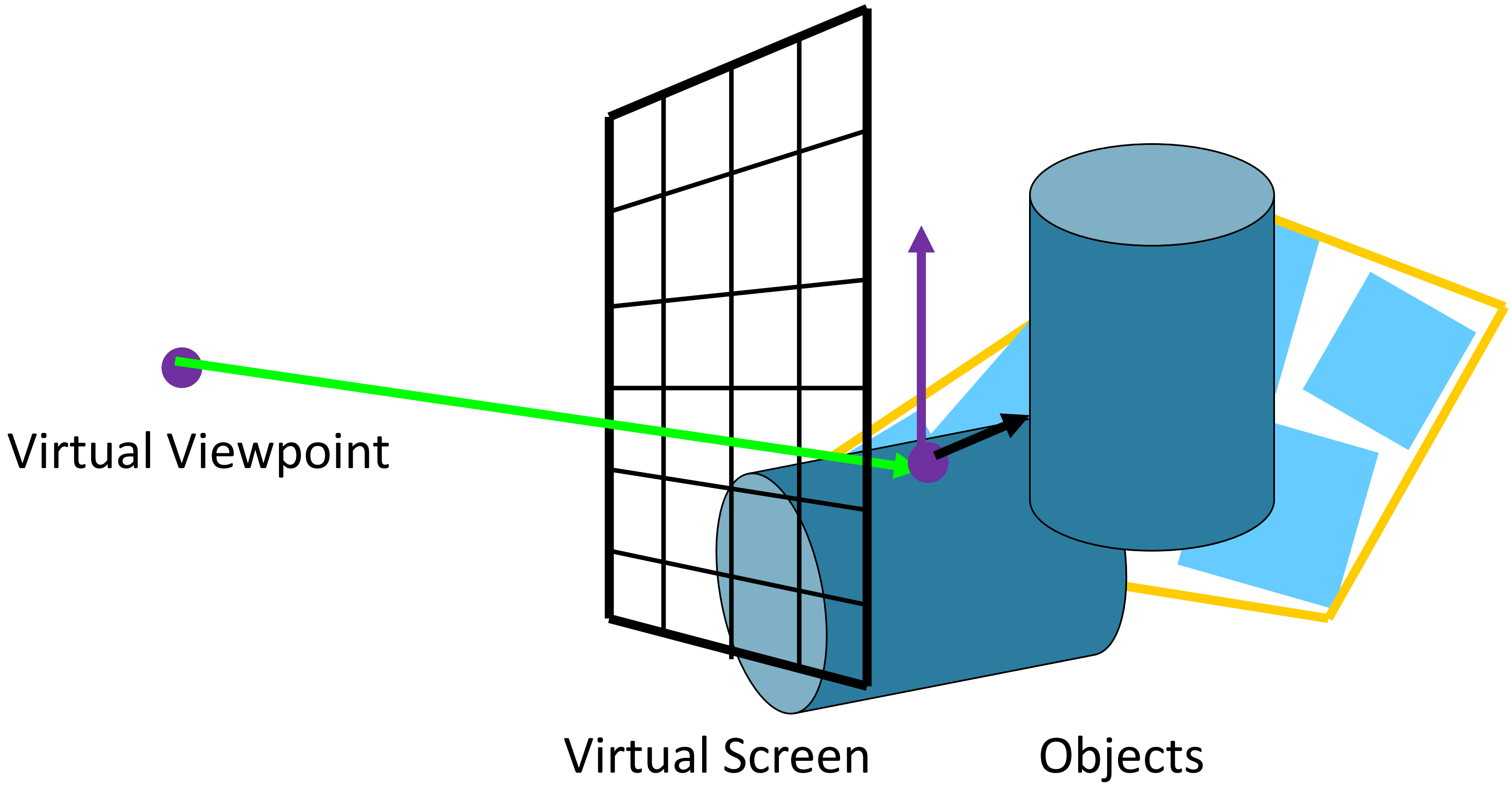


Computing Reflection Direction



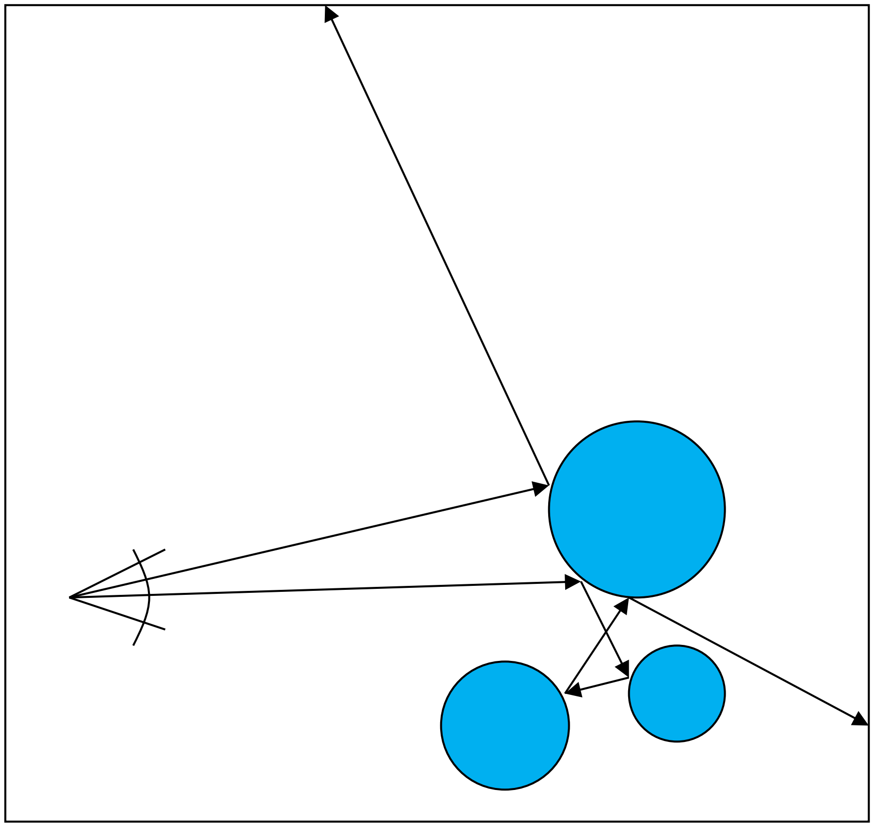
$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}$$

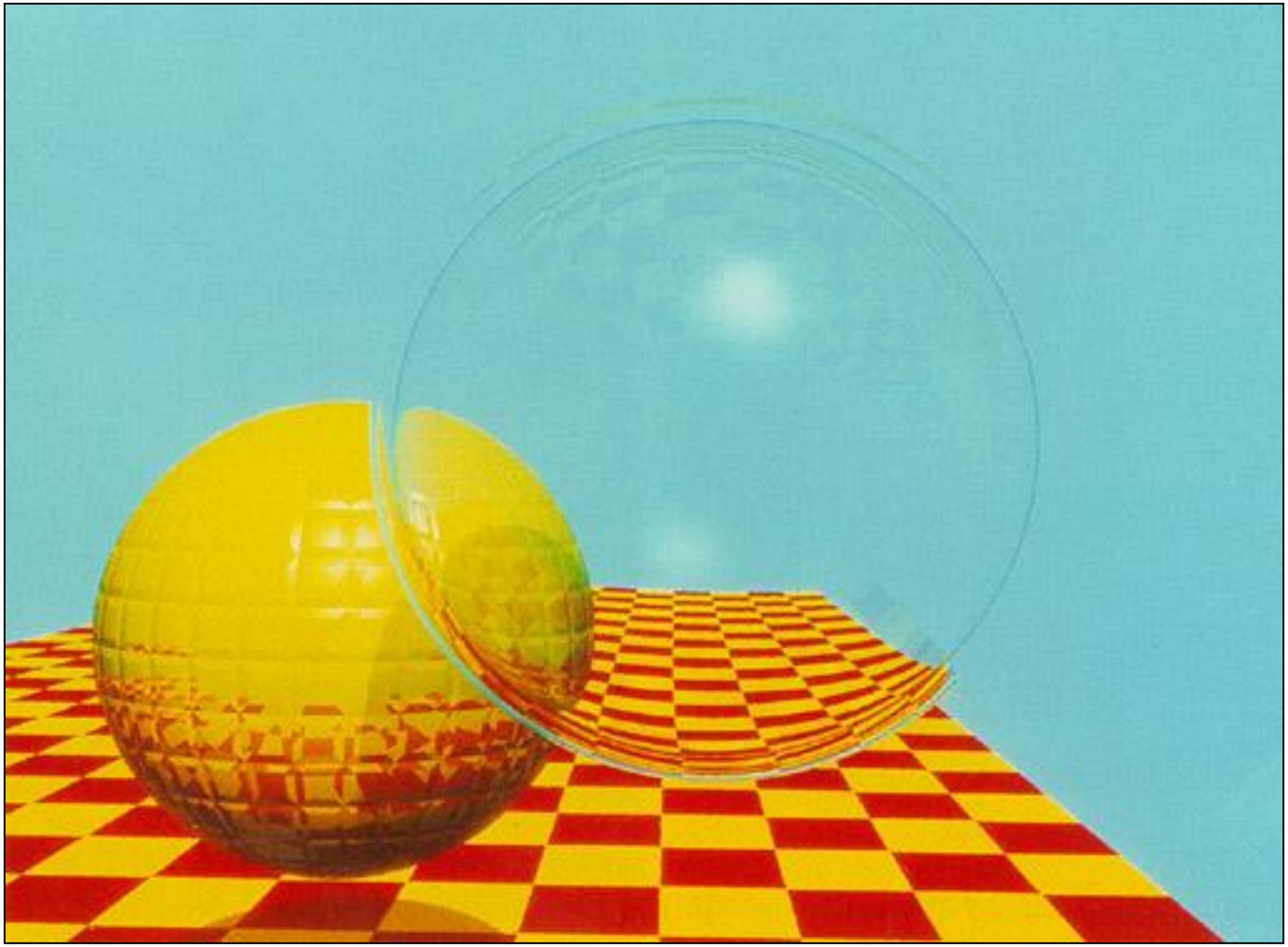
Mirror Reflections/Refractions



Generate reflected ray in mirror direction,
Get reflections and refractions of objects

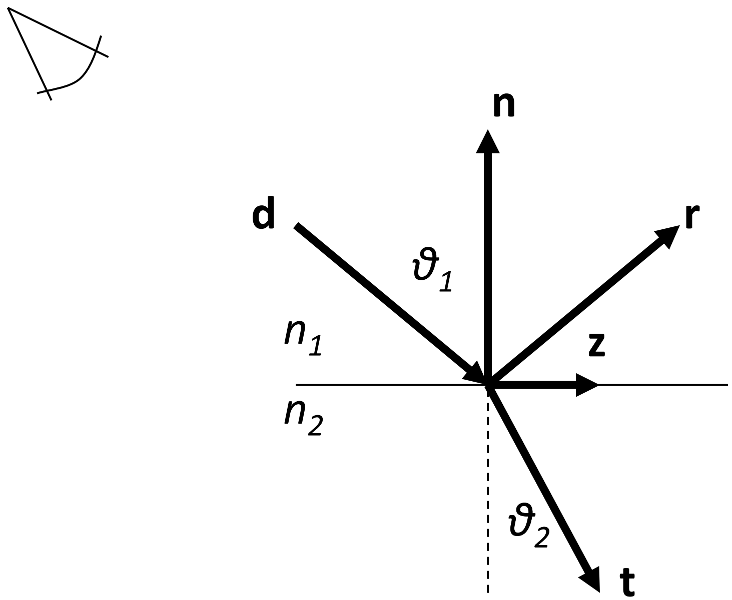
Reflections





Turner Whitted 1980

Computing Transmission (Refraction) Direction



$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}$$

$$\mathbf{z} = \frac{n_1}{n_2}\left(\mathbf{d} - (\mathbf{d} \cdot \mathbf{n})\mathbf{n}\right)$$

$$\mathbf{t} = \mathbf{z} - \left(\sqrt{1 - |\mathbf{z}|^2}\right)\mathbf{n}$$

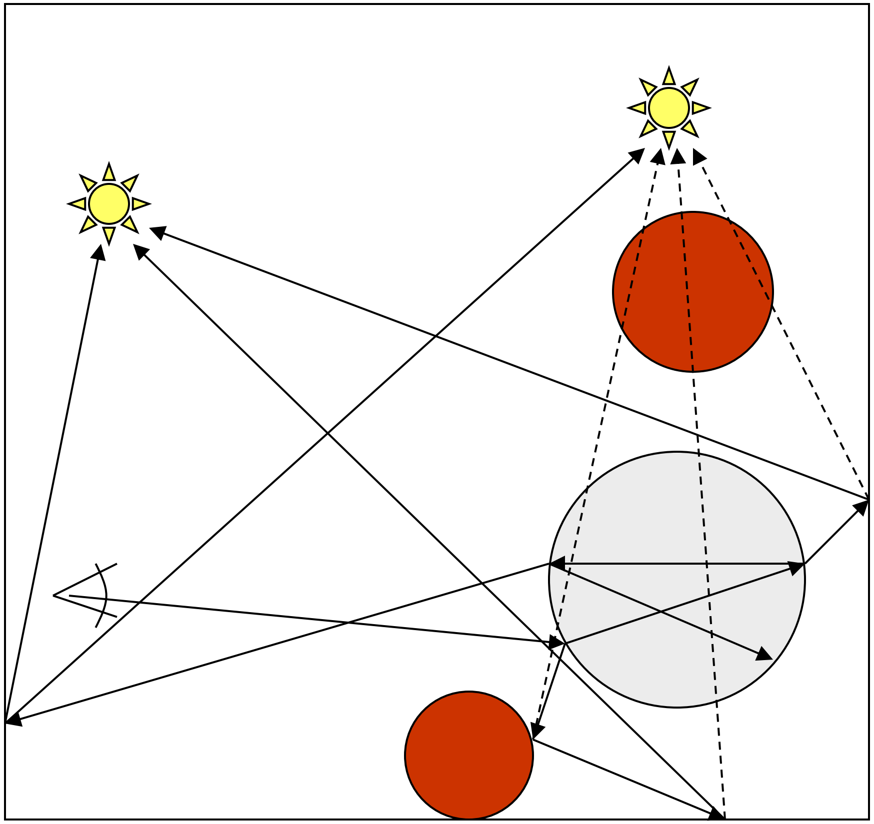
Spawning Multiple Rays

- When light hits a transparent surface, we not only see refraction, but we get a reflection off of the surface as well
- Therefore, we will actually generate two new rays and trace both of them into the scene and combine the results
- The results of an individual traced ray is a color, which is the color of the light that the ray 'sees'
- This color is used as the pixel color for primary rays, but for secondary rays, the color is combined somehow into the final pixel color

Recursive Ray Tracing

- The classic ray tracing algorithm includes features like shadows, reflection, refraction.
- A single primary ray may end up spawning many secondary and shadow rays, depending on the number of lights and the arrangement and type of materials
- These rays can be thought of as forming a tree like structure

Recursive Ray Tracing



Ray Tree

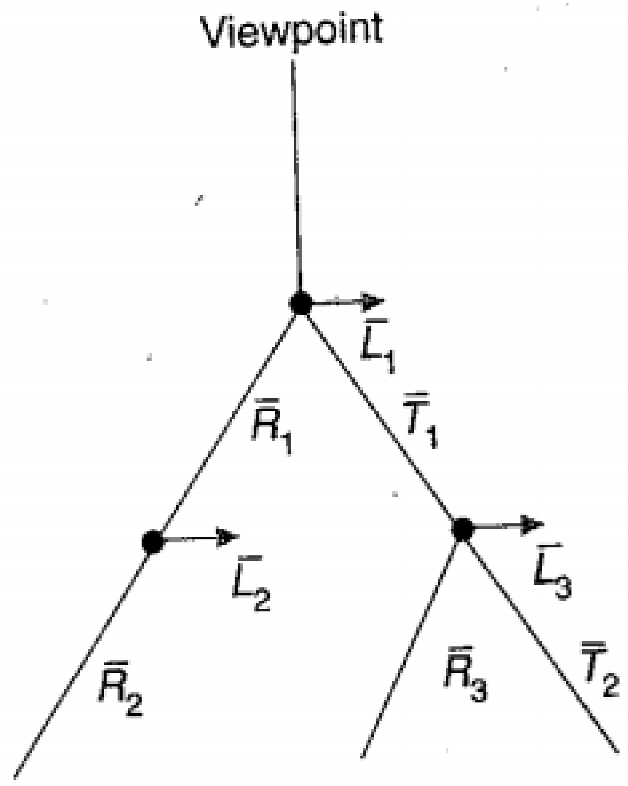


Fig. 16.55 The ray tree for Fig. 16.54.

Recursive Ray Tracing

For each pixel

- Trace Primary Ray, from eye to find intersection (if any) with the scene
- Trace Secondary Rays to all light(s)
  - Color += Visible ? apply illumination, otherwise 0
- Trace Secondary Ray for reflected ray
  - Color += k_r* color of reflected ray
- Trace Secondary Ray for refracted ray
  - Color += k_t * Color of transmitted ray

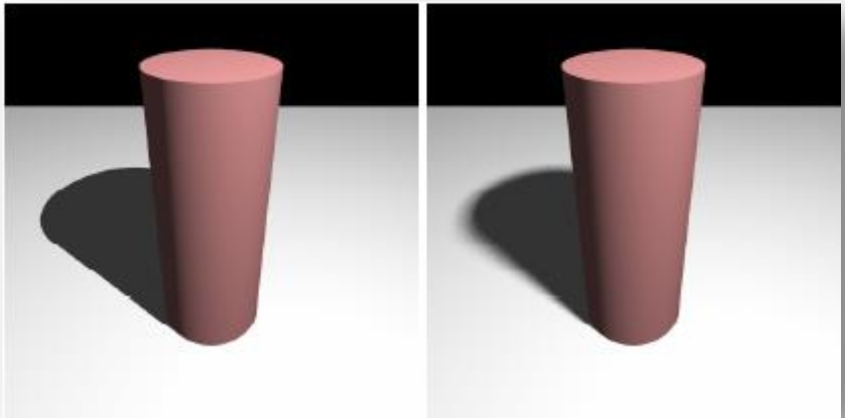
Effects needed for Realism

- (Soft) Shadows
- Reflections (Mirrors and Glossy)
- Transparency (Water, Glass)
- Inter-reflections (Color Bleeding)
- Complex Illumination (Natural, Area Light)
- Realistic Materials (Velvet, Paints, Glass)

Discussed in this lecture
Not discussed but possible with distribution ray tracing
Hard (but not impossible) with ray tracing; radiosity methods

Distributed Ray Tracing

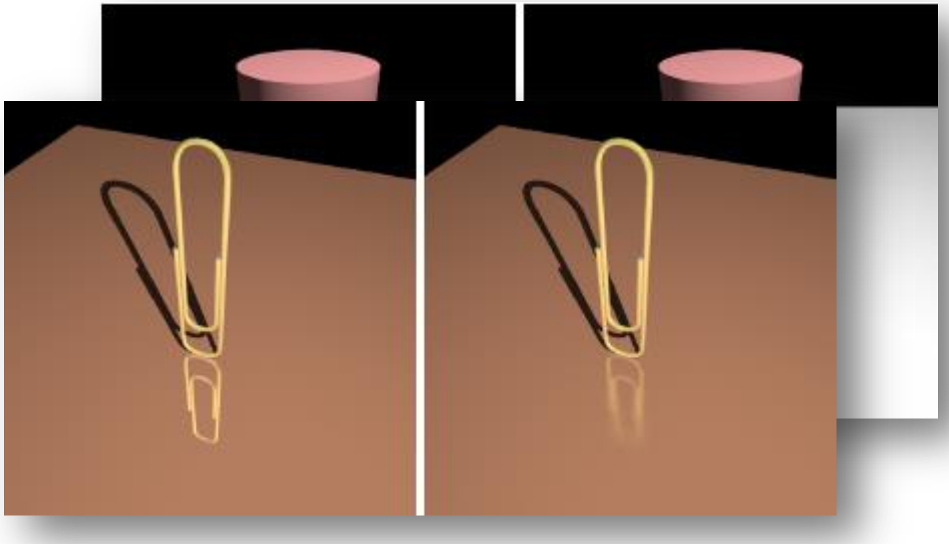
- Allows many physically correct effects:
  - Soft area shadows



(from [Boulos07])

Distributed Ray Tracing

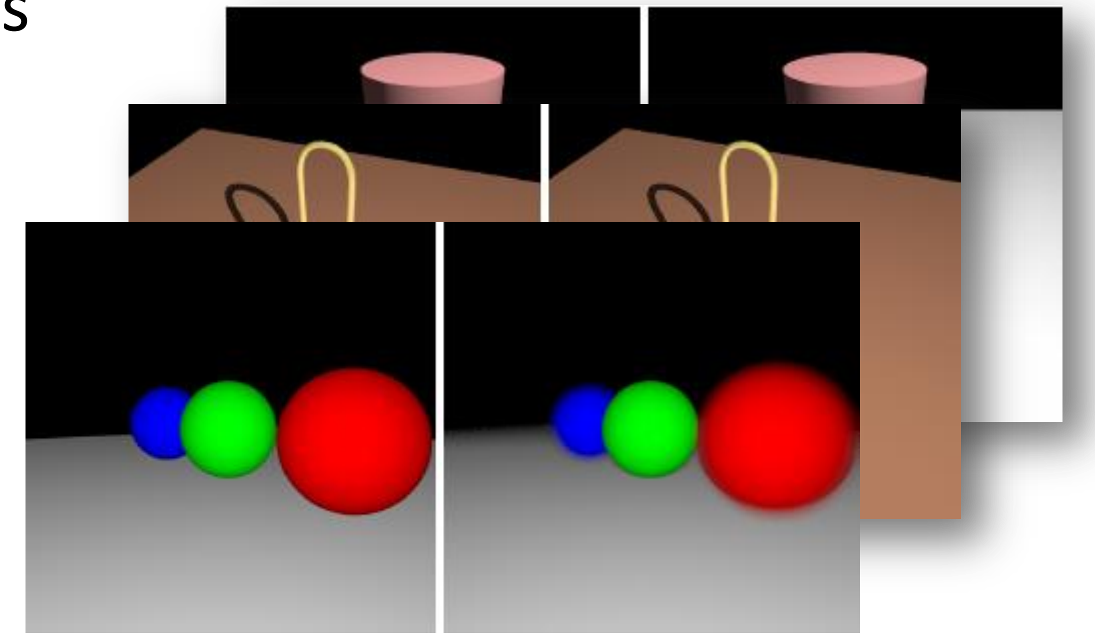
- Allows many physically correct effects:
  - Soft area shadows
  - Glossy surfaces



(from [Boulos07])

Distributed Ray Tracing

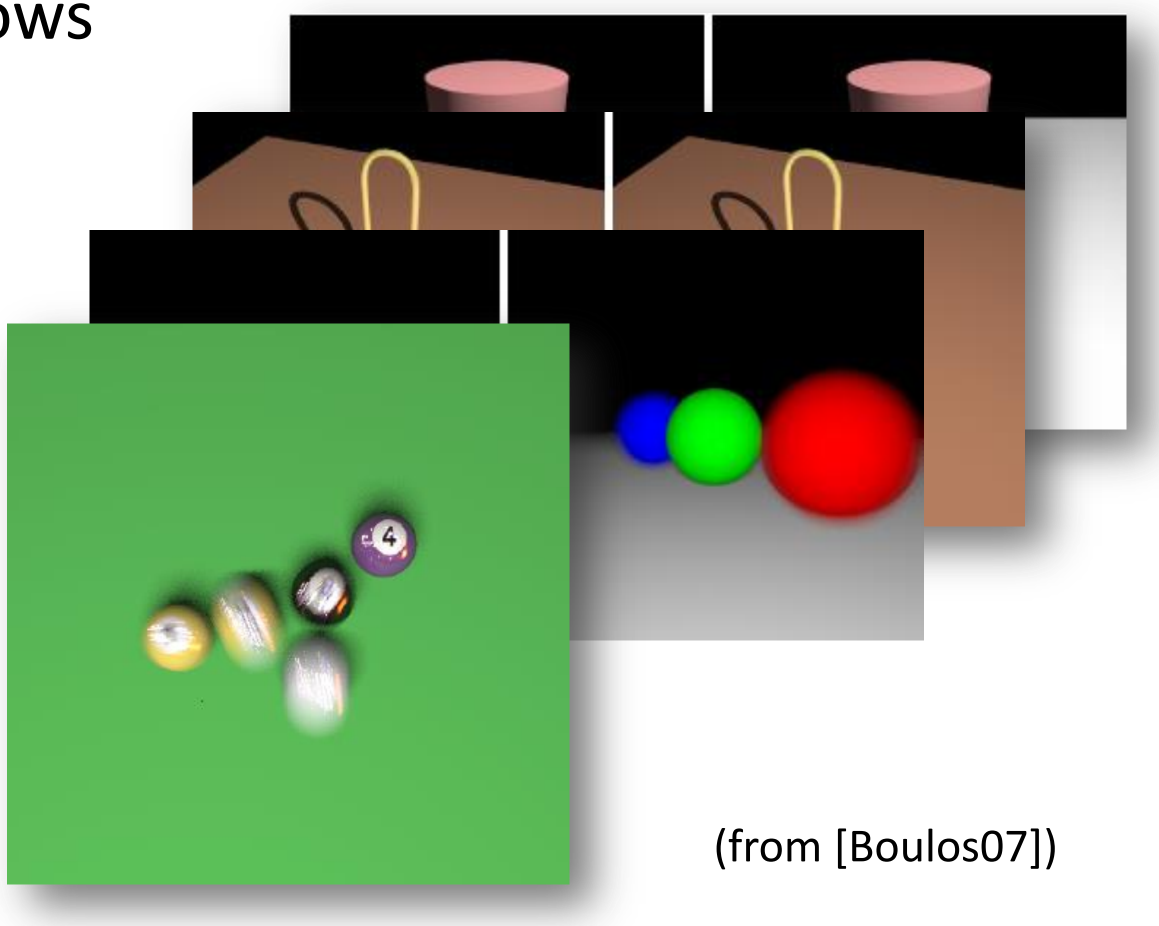
- Allows many physically correct effects:
  - Soft area shadows
  - Glossy surfaces
  - Depth of Field



(from [Boulos07])

Distributed Ray Tracing

- Allows many physically correct effects:
  - Soft area shadows
  - Glossy surfaces
  - Depth of Field
  - Motion blur

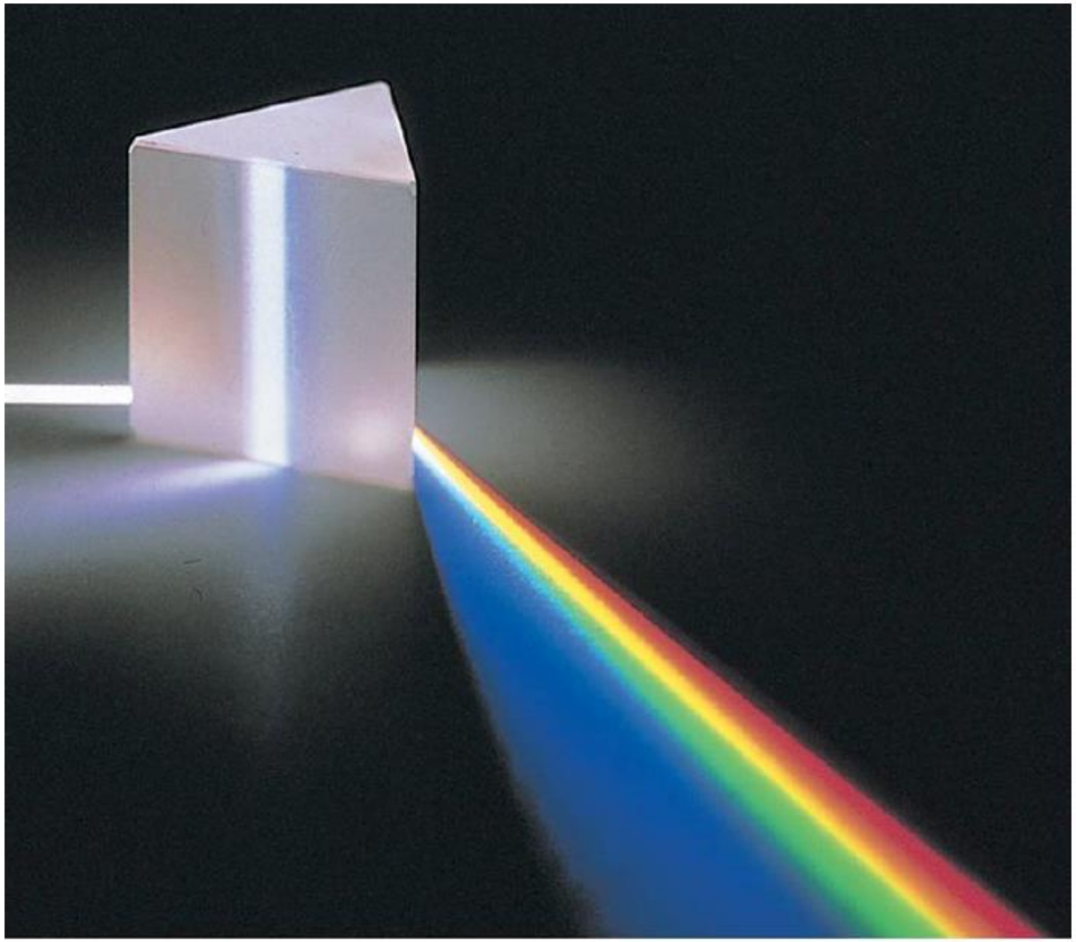


(from [Boulos07])

Advanced Topics

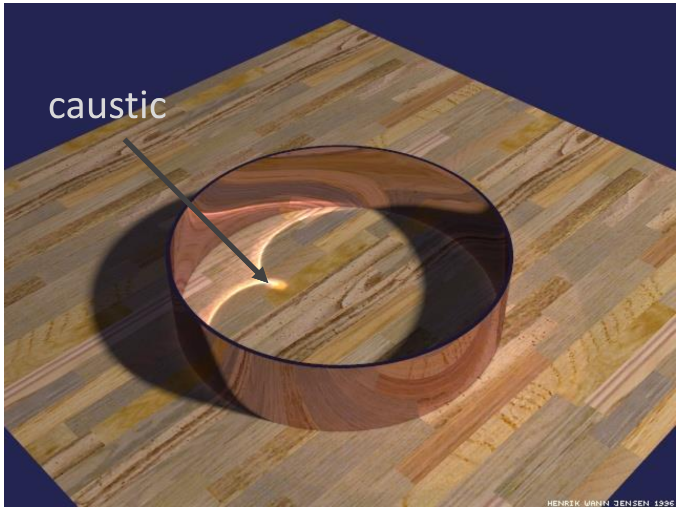
Slides from constructed from various Web sources

Forward ray tracing



Bidirectional Ray Tracing:

- Caustic – (concentrated) specular reflection/refraction onto diffuse surface
- Standard ray tracing cannot handle caustics



http://graphics.ucsd.edu/~henrik/

UCSD

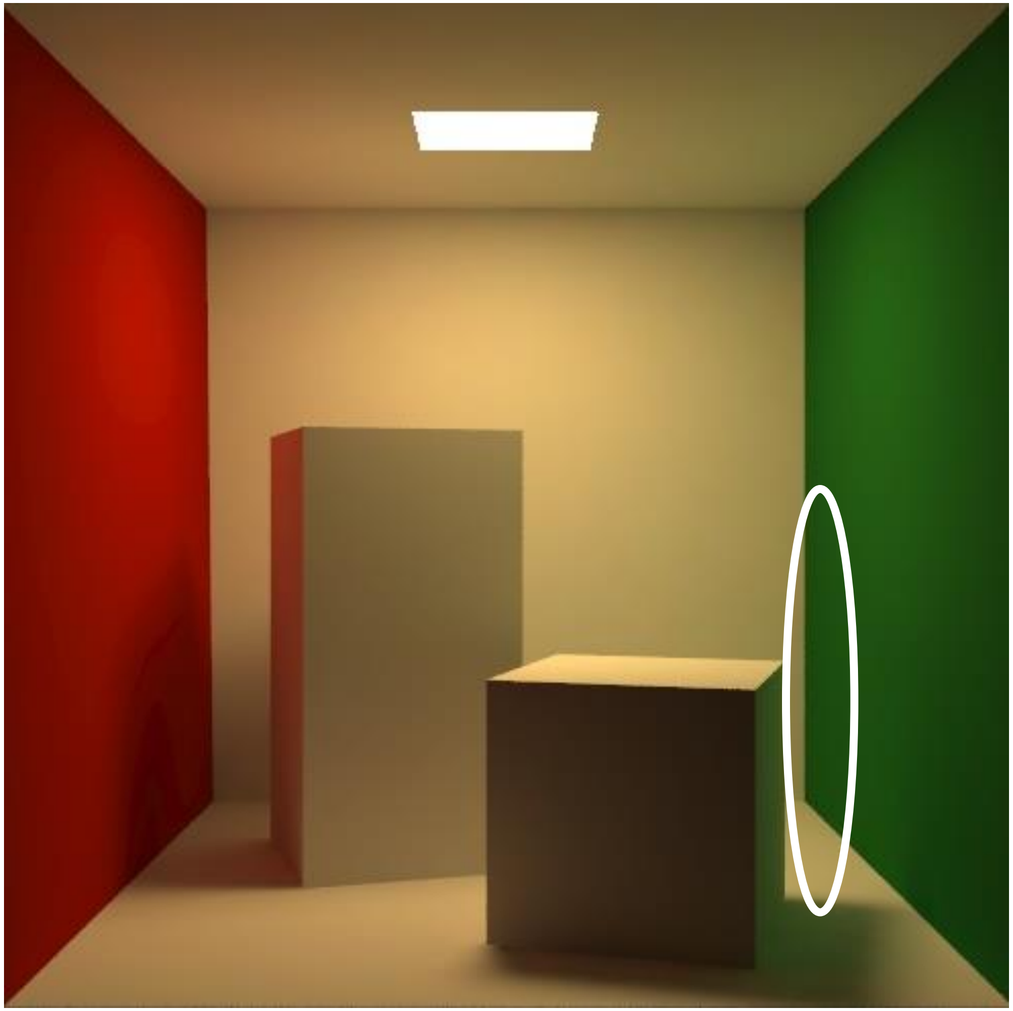
Adapted from slides © 2005 M. Thomas & C. Khambamettu, U. Del.
CISC 440/640: Computer Graphics, Spring 2005 – http://bit.ly/hz1kfU

University of Delaware

CAUSTICS

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

COLOR BLEEDING



Adapted from slides © 2005 M. Thomas & C. Khambamettu, U. Del.
CISC 440/640: Computer Graphics, Spring 2005 – http://bit.ly/hz1kfU

UNIVERSITY OF DELAWARE

RADIOSITY

Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg

RADIOSITY

RADIOSITY

ADVANCED TOPICS

ADVANCED TOPICS