Computer Graphics 1

Ludwig-Maximilians-Universität München
Summer semester 2020

Prof. Dr.-Ing. Andreas Butz
lecture additions by Dr. Michael Krone, Univ. Stuttgart

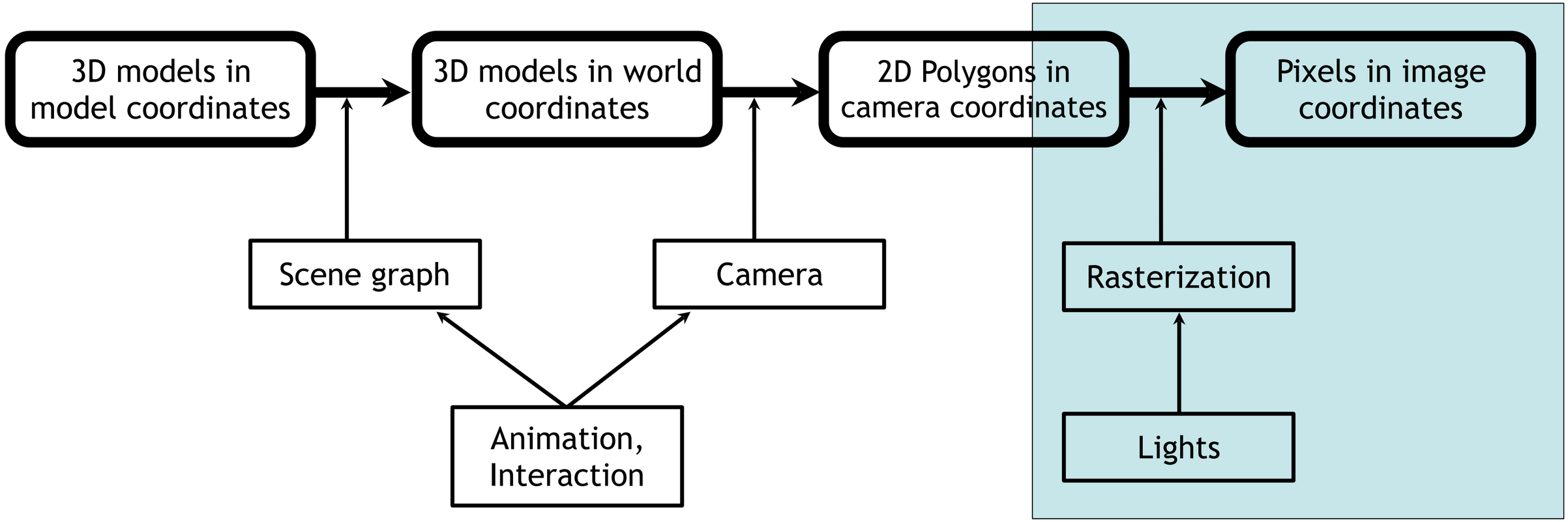
https://commons.wikimedia.org/wiki/File:Stanford_bunny_qem.png

Chapter 7 – Shading and Rendering

- Local Illumination Models: Shading
- Global Illumination: Ray Tracing
- Global Illumination: Radiosity
- Monte-Carlo Methods
- Non-Photorealistic Rendering

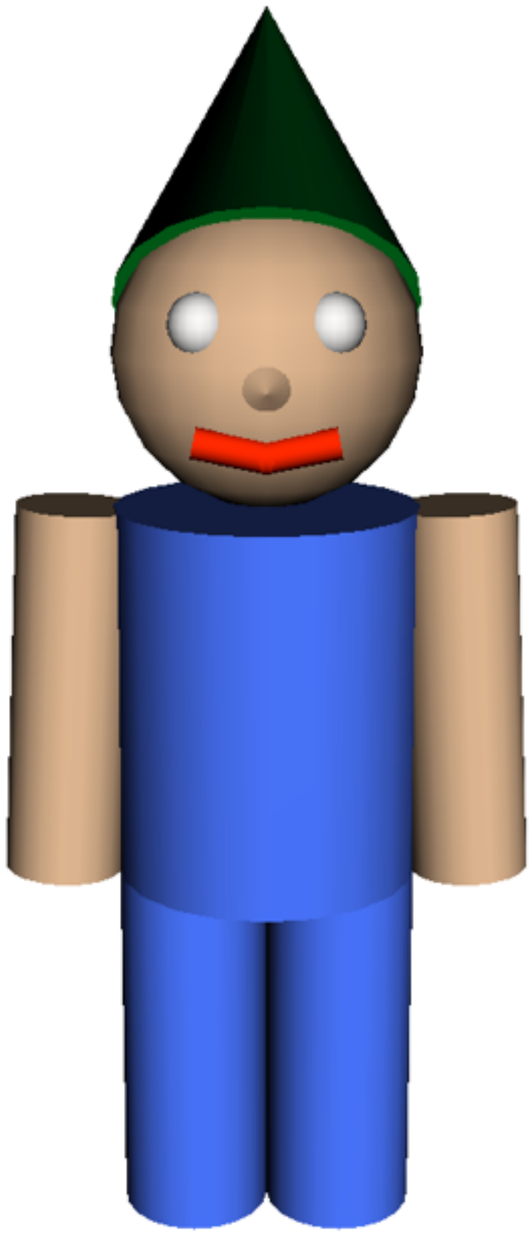
Literature: H.-J. Bungartz, M. Griebel, C. Zenger: Einführung in die Computergraphik, 2. Auflage, Vieweg 2002.
This lecture is partially based on slides by Prof. Thomas Ertl and Dr. Guido Reina, University of Stuttgart.

The 3D rendering pipeline (our version for this class)



Local Illumination: Shading

- Local illumination:
  - Light calculations are done **locally** without the global scene
  - No cast shadows
    (since those would be from other objects, hence **global**)
    - Shadow rendering via “tricks”, e.g., Shadow Maps
  - Object shadows are OK, only depend on the surface normal



- Simple idea: Loop over all polygons („object order“)
  - For each polygon:
    - Determine the pixels it occupies on the screen and their color
    - Draw using, e.g., Z-buffer algorithm to get occlusion right
- Each polygon only considered once
- Some pixels considered multiple times
- More efficient: Scan-line algorithms („image order“)

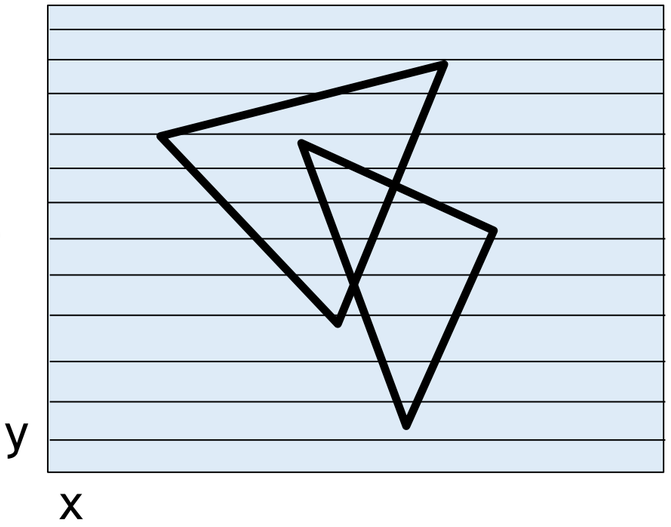
Object Order or Image Order?

- Image order algorithms iterate over the pixels in the image.
- Object order algorithms iterate over the elements in the scene.

- Hi-res image, few polygons: object order
- Many objects or large data set: image order

Scan-Line Algorithms in More Detail

- Polygon Table (PT):
  - List of all polygons with plane equation parameters, color information and inside/outside flag (see rasterization)
- Edge Table (ET):
  - List of all non-horizontal edges, sorted by y value of top end point
  - Including a reference back to polygons to which the edge belongs
- Active Edge Table (AET):
  - List of all edges crossing the current scan line, sorted by x value



→ Each ***polygon*** considered only once

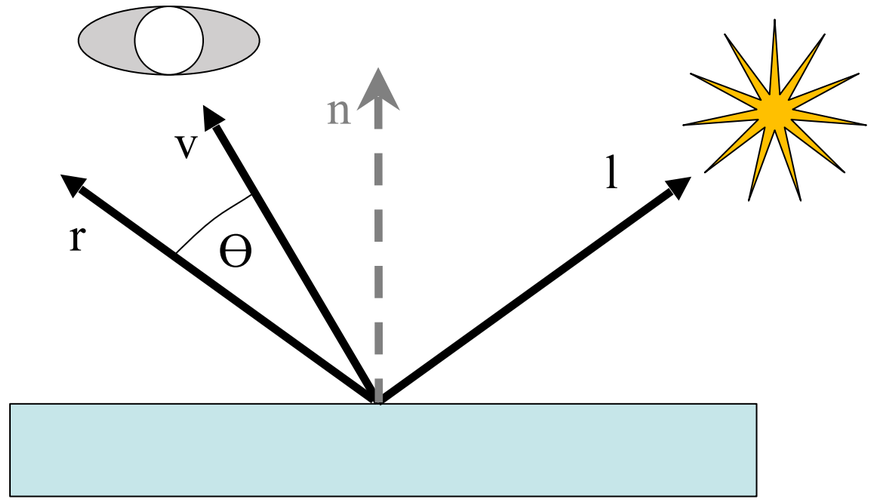
→ Each ***pixel*** considered only once

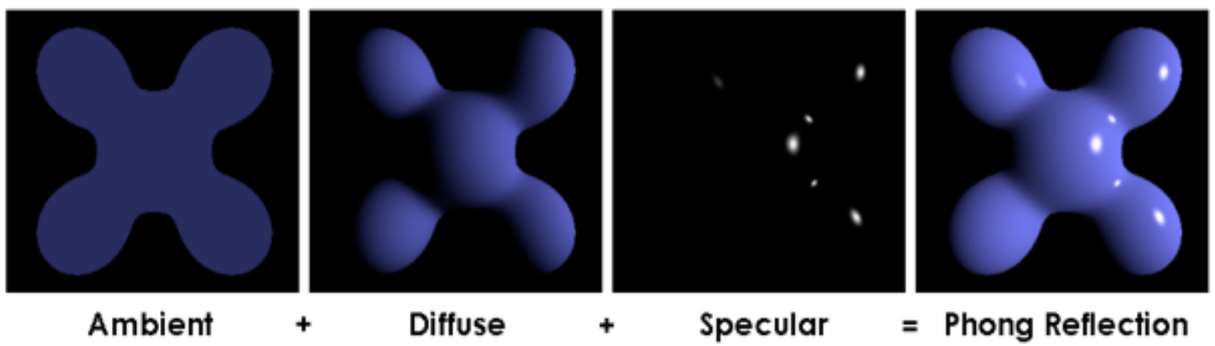
```
for v = 0..V: // (all scan lines)
   Compute AET, reset flags in PT;
   for all crossings in AET:
      Update flags;
      Determine currently visible polygon P (Z-buffer);
      Set pixel color according to info for P in PT;
   end
end
```

Reminder: Phong's Illumination Model

$$I_o = I_{amb} + I_{diff} + I_{spec} = I_a k_a + I_i k_d (\vec{l} \cdot \vec{n}) + I_i k_s (\vec{r} \cdot \vec{v})^n$$

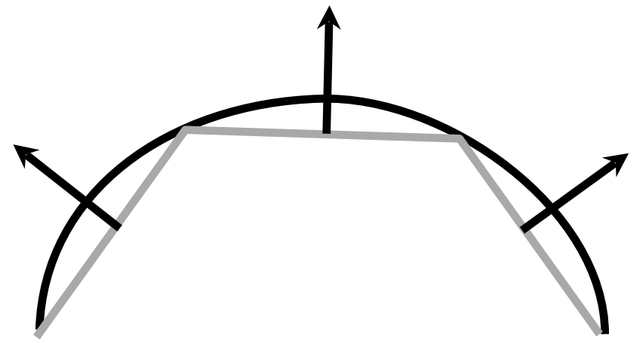
- Prerequisites for using the model:
  - Exact location on surface known
  - Light source(s) known
- Generalization to many light sources:
  - Summation of all diffuse and specular components created by all light sources
- Light colors easily covered by the model
- Do we really have to compute the equation for each pixel?



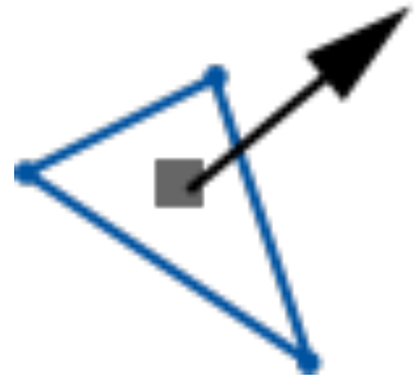


http://de.wikipedia.org/w/index.php?title=Datei:Phong_components_version_4.png

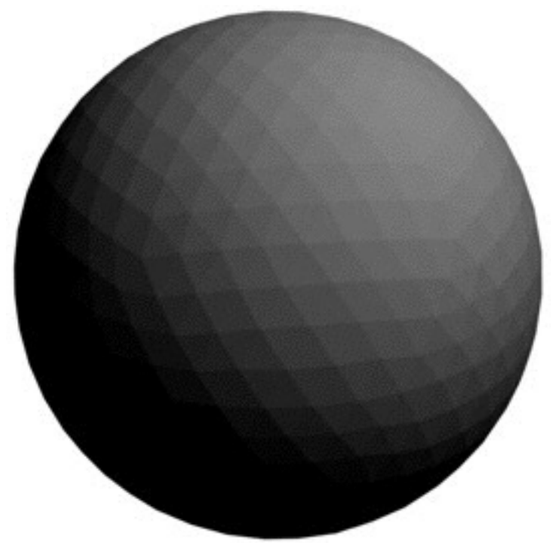
Flat Shading



- Determine one surface normal for each triangle
- Compute the color for this triangle
  - Using e.g., the Phong illumination model
  - Usually for the center point of the triangle
  - Using the normal, camera and light positions
- Draw the entire triangle in this color



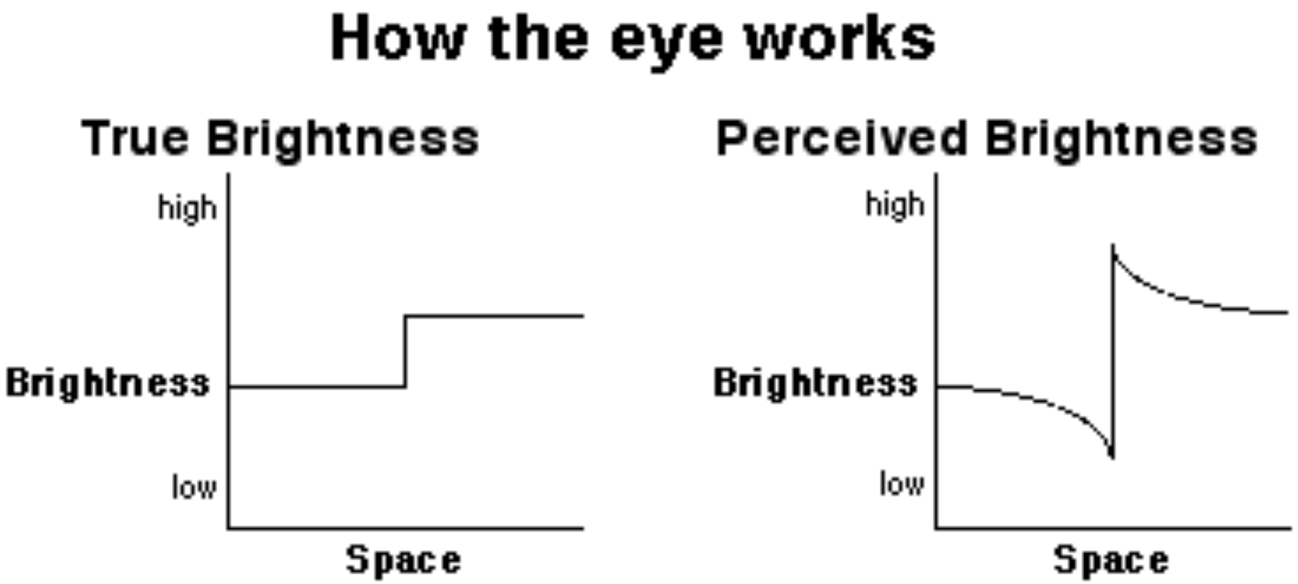
- Neighboring triangles will have different shades
- Visible „crease“ between triangles



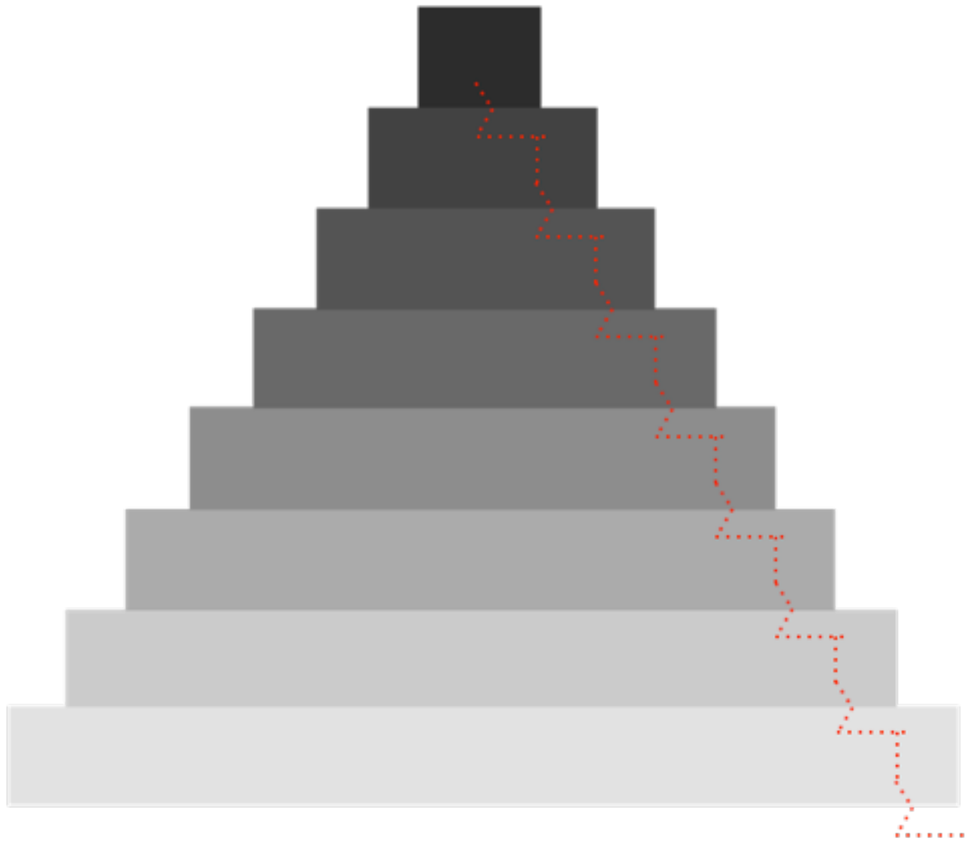
- Cheapest and fastest form of shading
- Can be a desired effect, e.g., with primitives
  - Deliberate, artistic choice
  - Hard edges for cubes etc.

Mach Band Effect

- Flat Shading suffers from an optical illusion
  - Human visual system accentuates discontinuity at brightness boundary
  - Darker stripes appear to exist at dark side, and vice versa

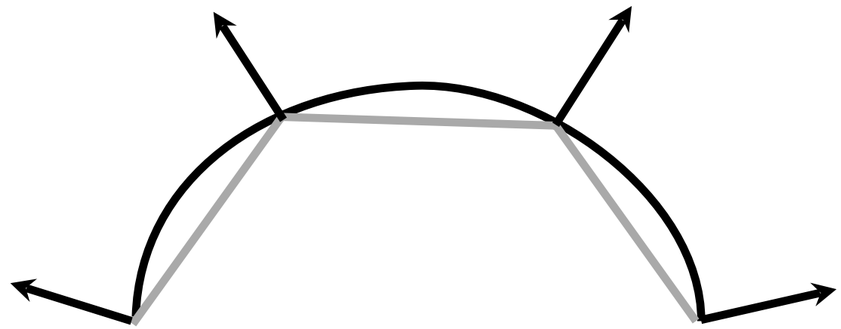


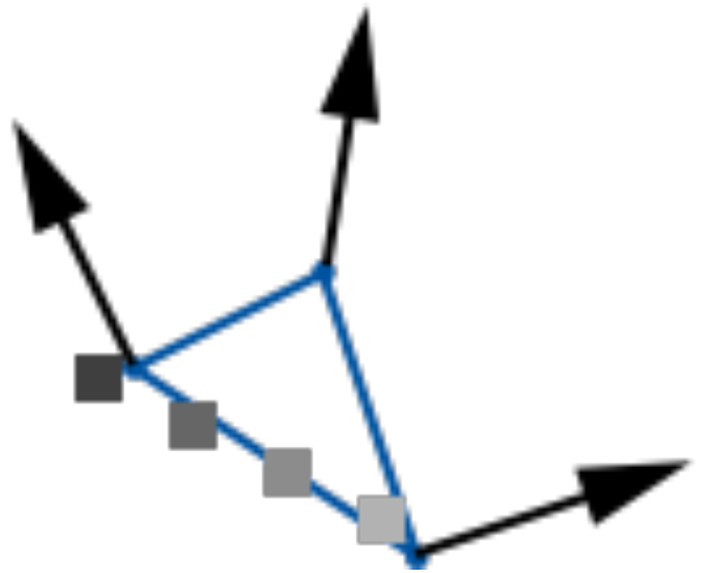
Source: `keithwiley.com`



Source: Wikipedia

Gouraud Shading

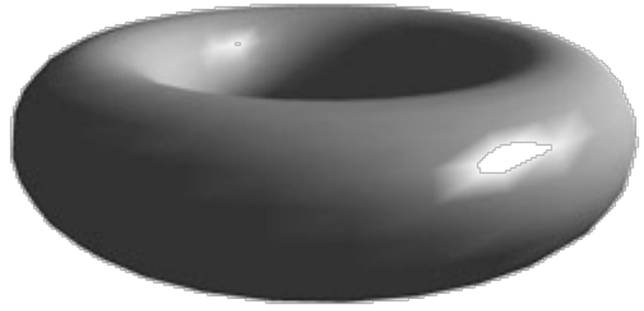


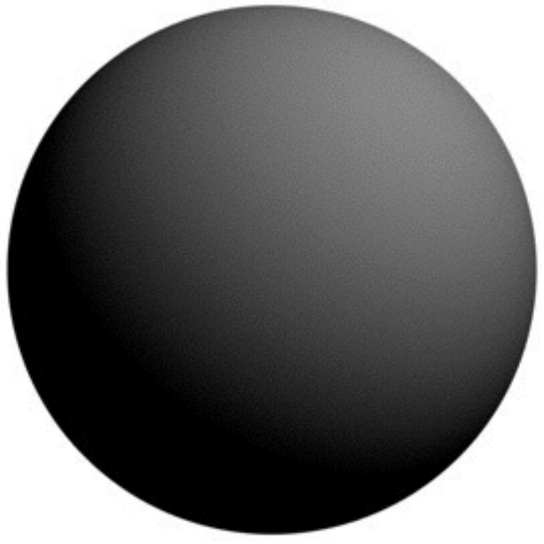


- Determine normals for all mesh vertices
  - i.e., each triangle now has 3 normals
- Compute colors at all vertices
  - Using e.g., the Phong illumination model
  - Using the 3 normals, camera and light positions
- Interpolate between these colors along the edges
- Interpolate also for the inner pixels of the triangle

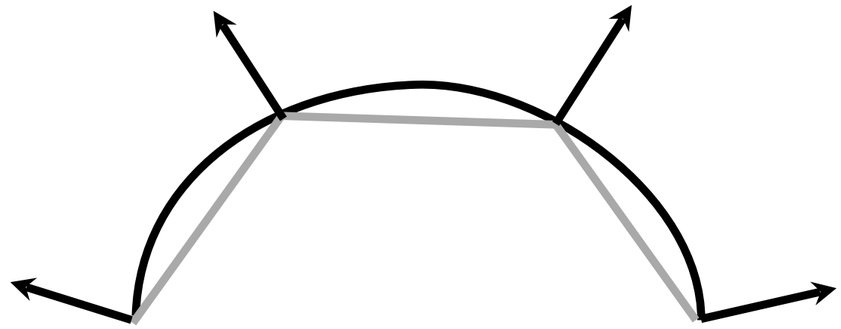
- Neighboring triangles will have smooth transitions…
  - …if normals at a vertex are the same for all triangles using it

- Simplest form of smooth shading
  - Specular highlights only if they fall on a vertex by chance

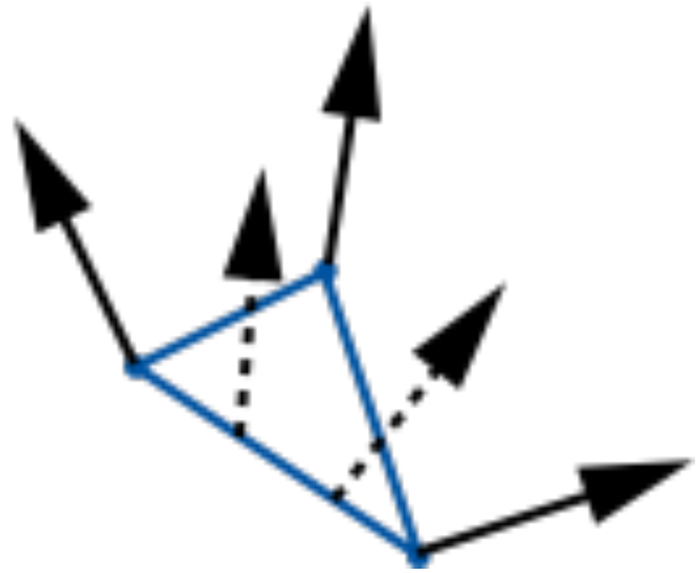




Phong Shading

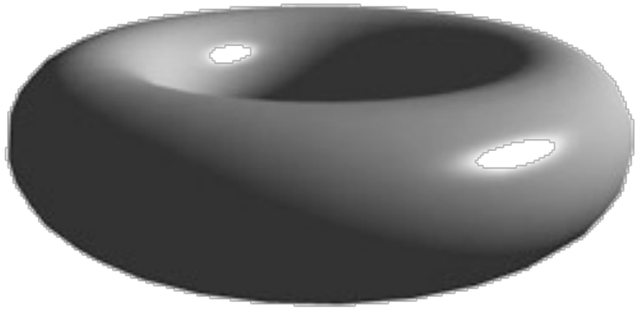


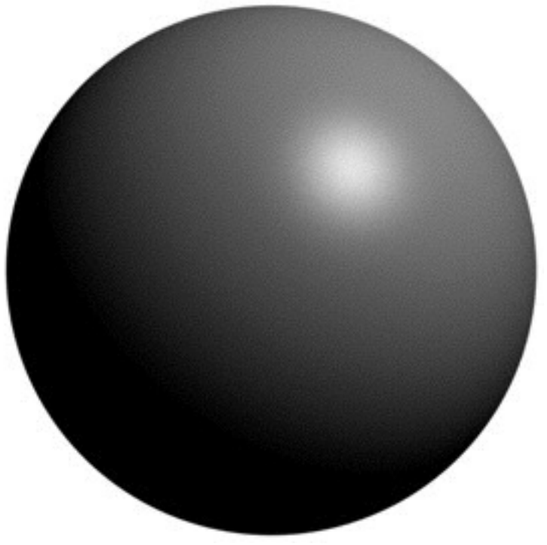
- Determine normals for all mesh vertices
- Interpolate between these normals along the edges
- Compute colors at all vertices
  - Using e.g., the Phong illumination model
  - Using the interpolated normal, camera and light positions



- Neighboring triangles will have smooth transitions…
  - …if normals at a vertex are the same for all triangles using it

- Has widely substituted Gouraud shading
  - Specular highlights in arbitrary positions
  - Have to compute Phong illumination model for every pixel



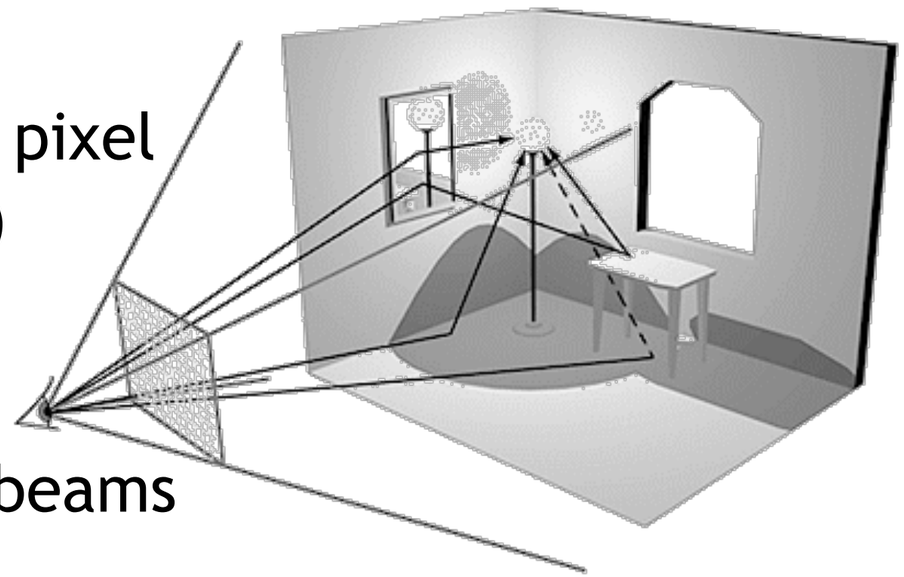


Chapter 7 – Shading and Rendering

- Local Illumination Models: Shading
- Global Illumination: Ray Tracing
- Global Illumination: Radiosity
- Monte-Carlo Methods
- Non-Photorealistic Rendering

Global Illumination: Ray Tracing (Basic idea)

- Global illumination:
  - Light calculations are done globally considering the entire scene
  - i.e. cast shadows are OK if properly calculated
  - Object shadows are OK anyway
- Ray casting:
  - From the eye, cast a ray through every screen pixel
  - Find the first polygon it intersects with
  - Determine its color at intersection and use for the pixel
  - Also solves occlusion (makes Z-Buffer unnecessary)
- Ray tracing:
  - Recursive ray casting
  - From intersection, follow reflected and refracted beams
  - Up to a maximum recursion depth
  - Works with arbitrary geometric primitives



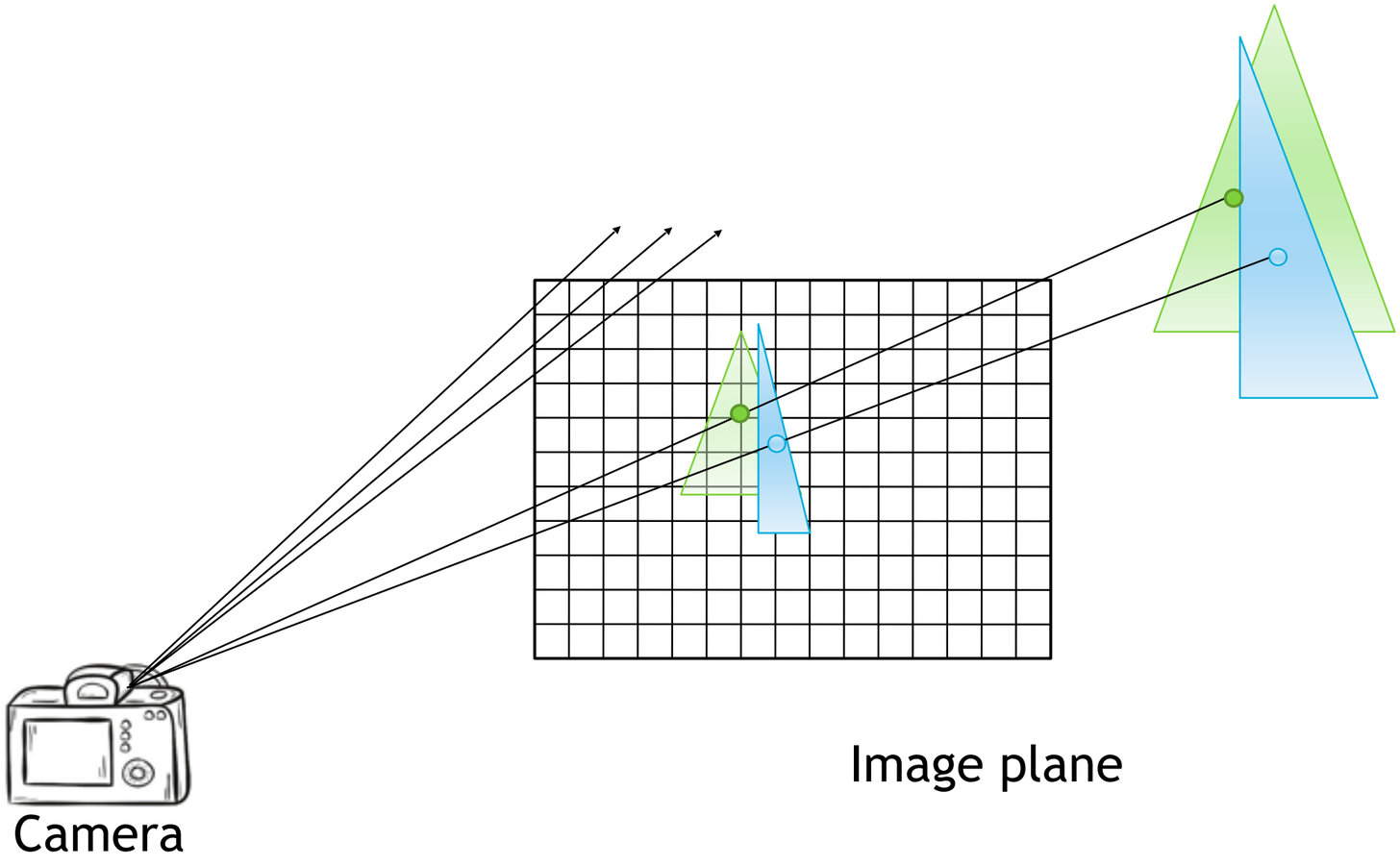
http://pclab.arch.ntua.gr/03postgra/mladenstamenico/ (probably not original)

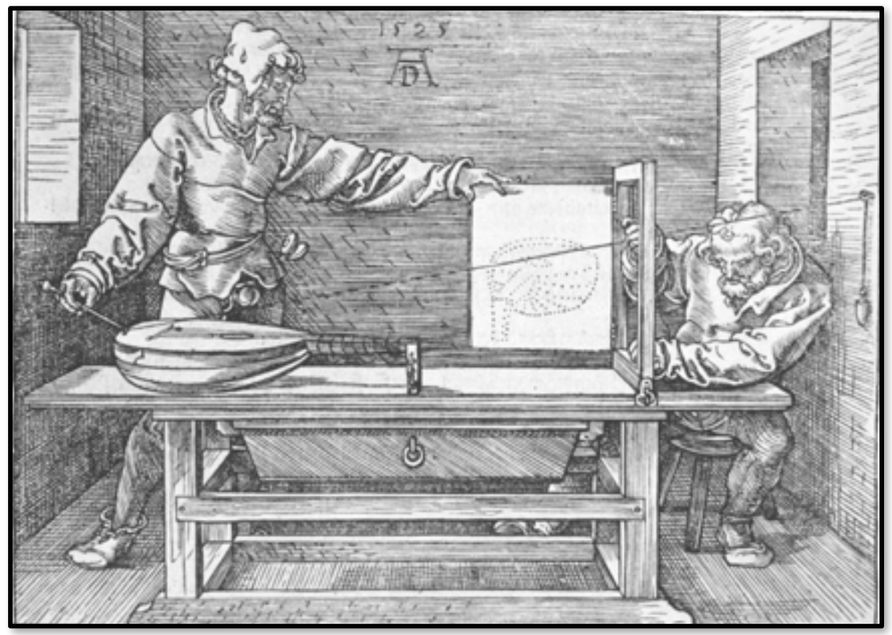
source: Blender Gallery

source: Blender Gallery

Ray Tracing (Image Order)

- Basic idea: geometric considerations about light rays (“light transport”)
  - Trace light rays that reach the opening of our pinhole camera
  - For each pixel:
    - Find all objects that influence this pixel
    - Compute the pixel color based on the materials of the objects

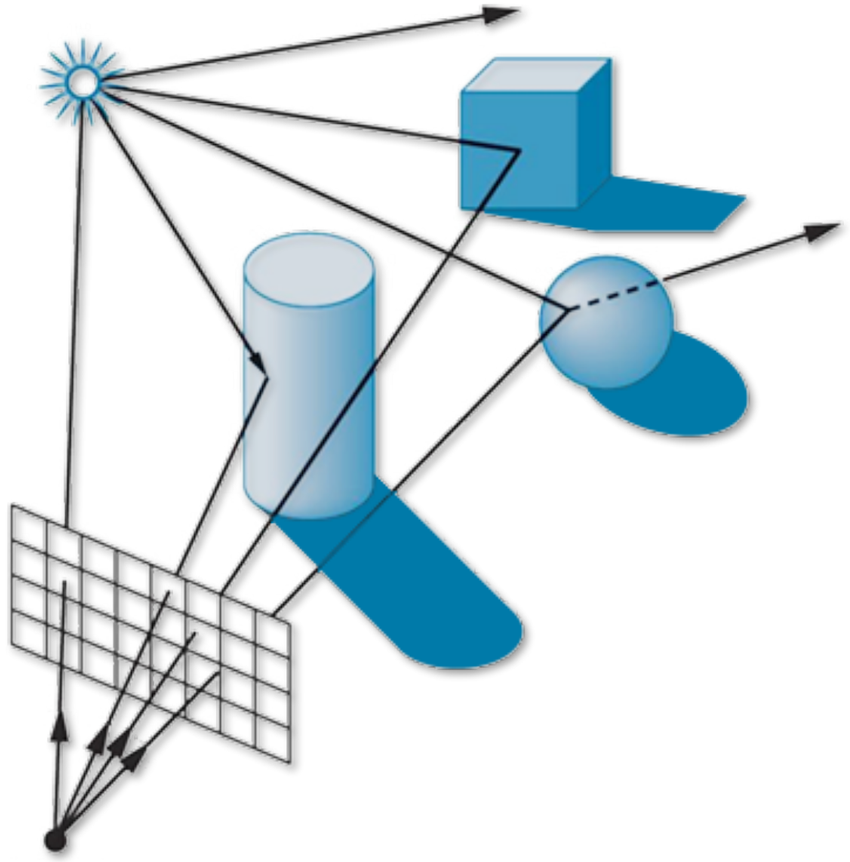


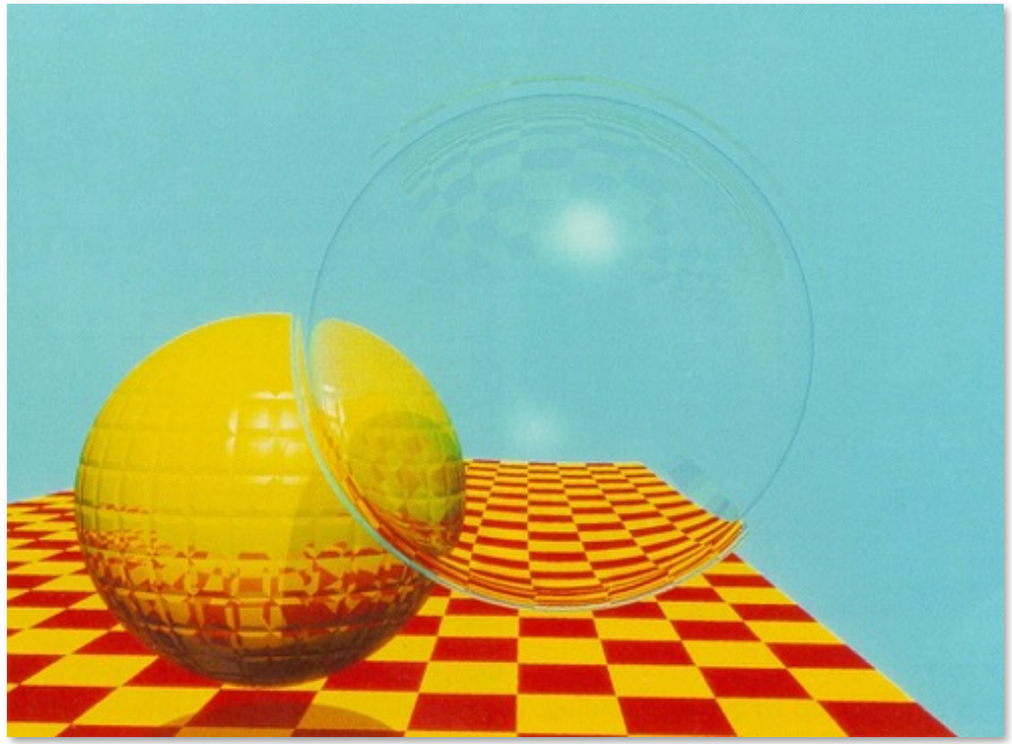


Albrecht Dürer: „Underweysung der messung mit dem Zirkel und richtscheyt, in Linien Ebnen un gantzen Corporen“, 1525

Ray Tracing

- Inverse “light transport”
- Start at the camera
- Search paths from which light can reach the camera
  - Assumption: light transport follows the laws of geometric optics





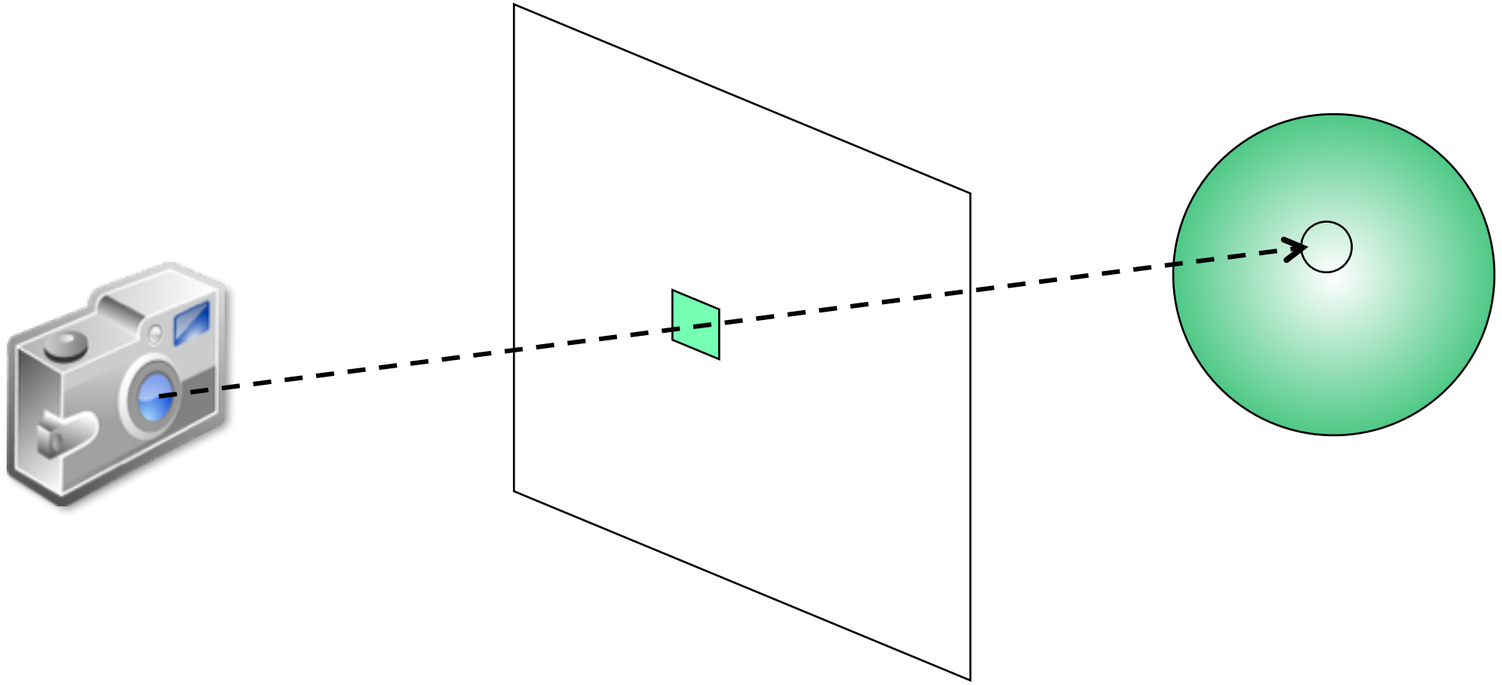
[T. Whitted: An Improved Illumination Model for Shaded Display, 1980]

Excursion: Geometric Optics

- Neglects wave nature of light
- Light rays don’t interfere with each other
- Fermat’s Principle: Light always takes the shortest path
- Law of Reflection: Incoming angle = Outgoing angle
- Law of Refraction: Transmission + Reflection at material boundaries
- Specifically, there is ***NO***
  - Absorption
  - Scattering
- …and ***NO*** effects that require quantum mechanics
  - Polarization
  - Diffraction
  - Interference

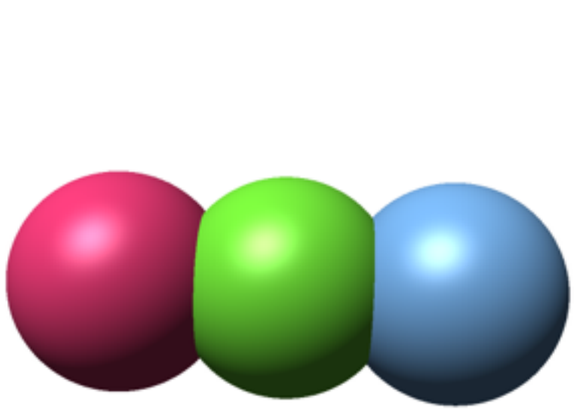
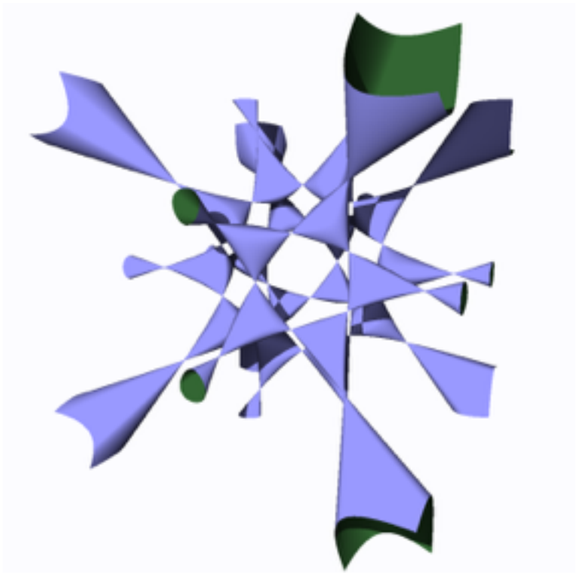
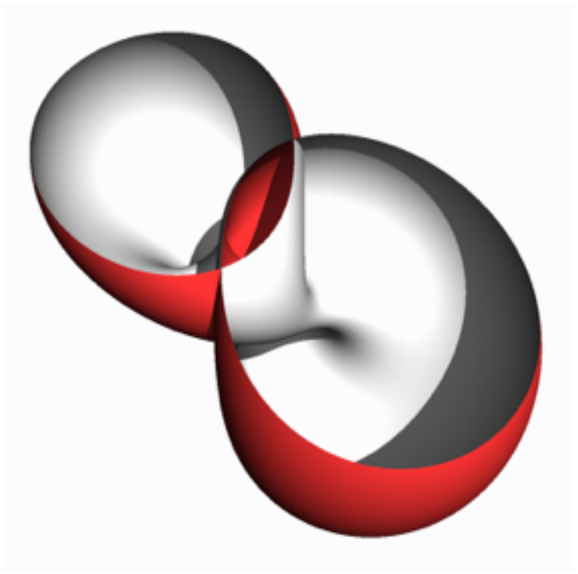
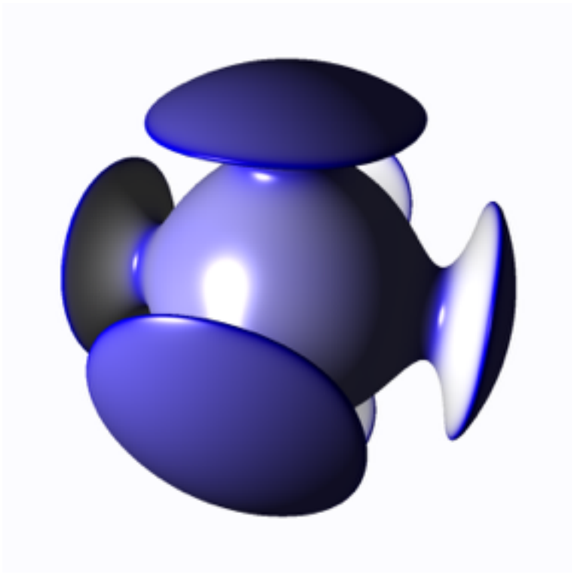
Ray Tracing: Principle

- Compute color values of pixels one by one
- Find the object that is visible for the camera at this pixel
  - View ray: a straight line from the camera through the center of the pixel
  - Find the object that is intersected by the ray and closest to the camera
- Compute the color and shading for the object at the intersection point



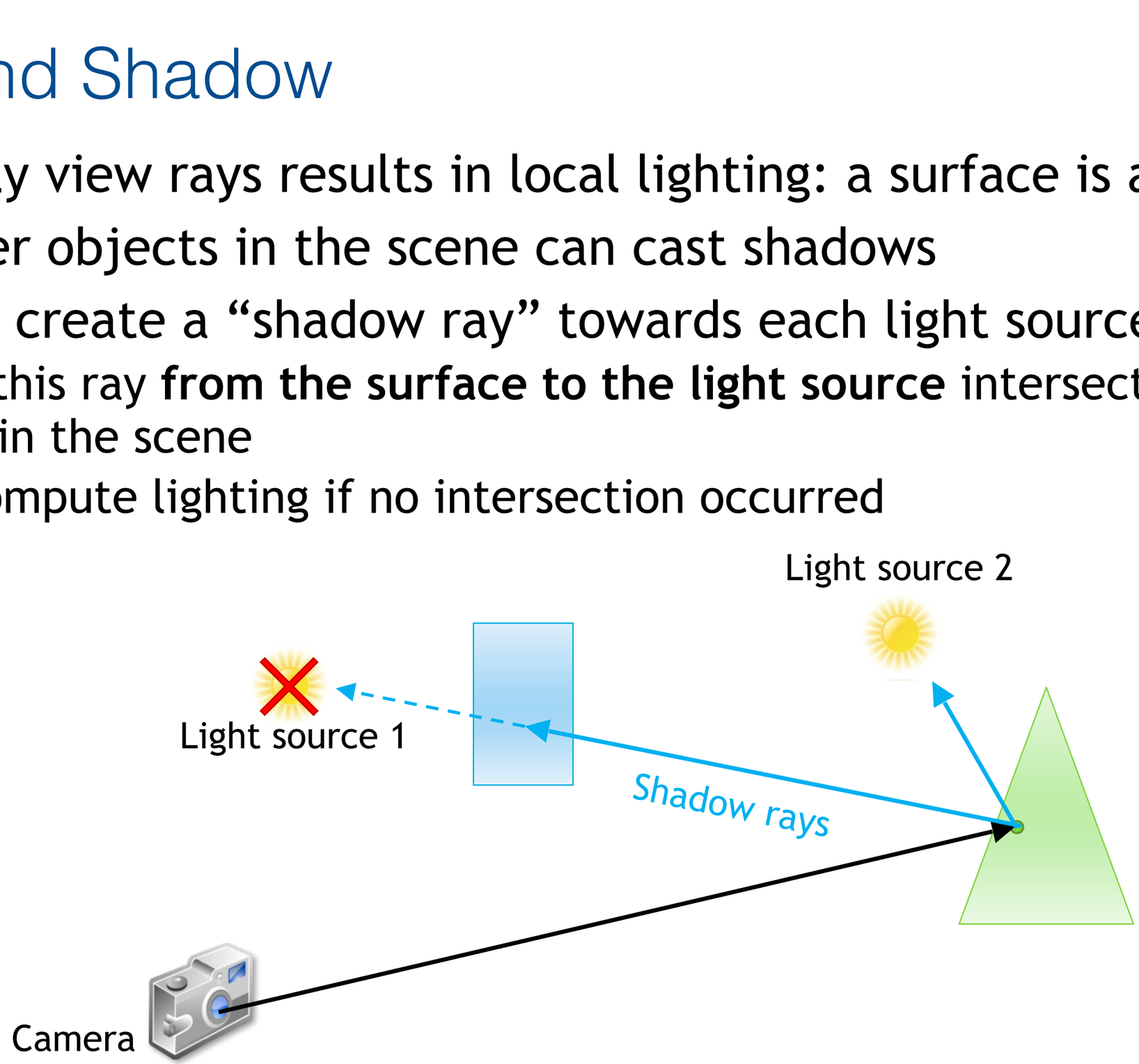
Ray Tracing: Ray-Object Intersections

- Ray tracing can use arbitrary objects
  - Not limited to flat polygons/triangles
- Objects can be described implicitly (mathematical description)
  - No subdivision
  - Sphere, ellipsoid, cylinder, cone, torus…
  - Intersection test requires root finding for polynomial functions
  - Pro: perfectly smooth surfaces
  - Con: potentially costly root finding for higher-order surfaces

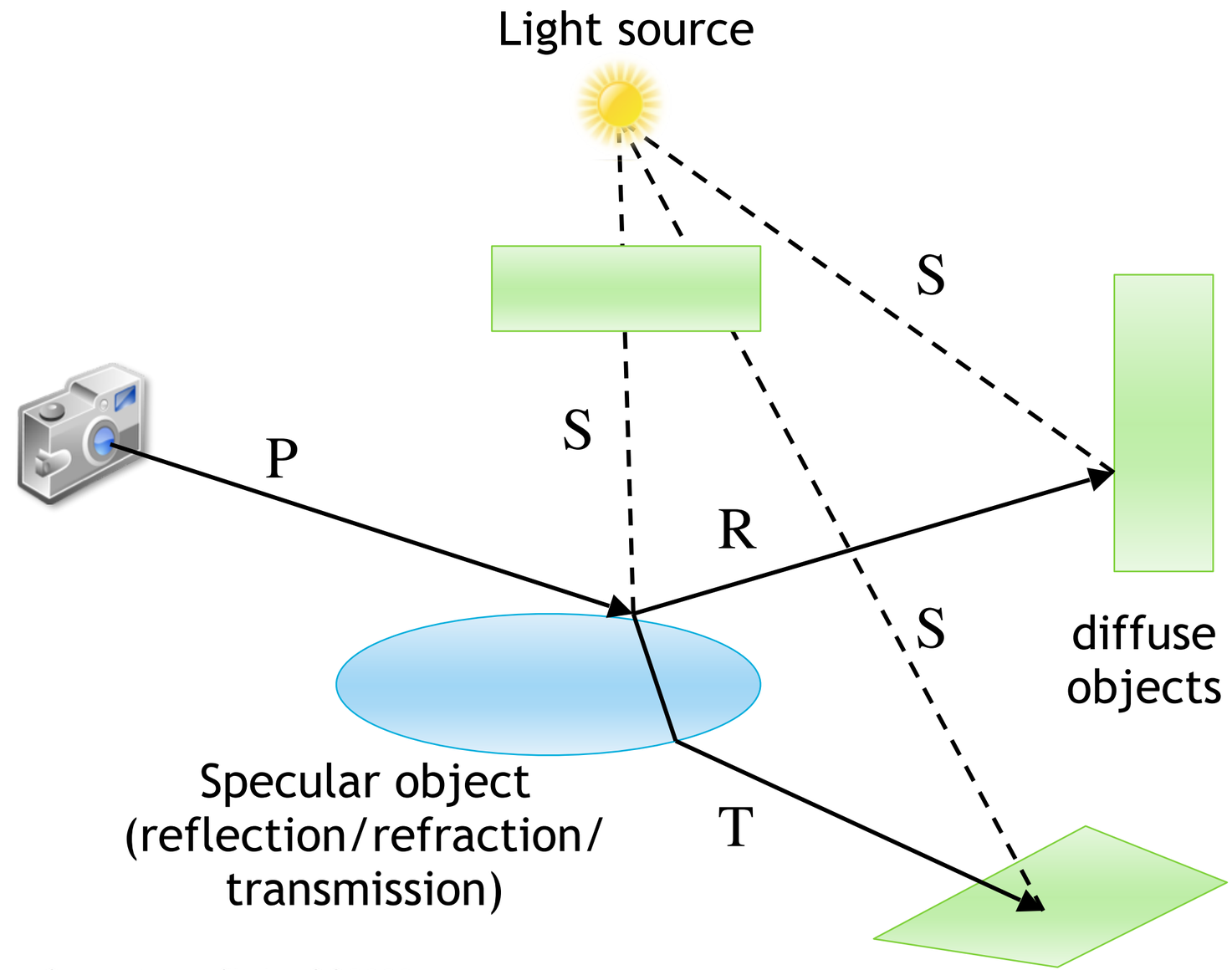
Light and Shadow

- Using only view rays results in local lighting: a surface is always shaded
- But: other objects in the scene can cast shadows
- Solution: create a “shadow ray” towards each light source
  - Test if this ray **from the surface to the light source** intersects another object in the scene
  - Only compute lighting if no intersection occurred



Primary and Secondary Rays

- Geometric optics
- Basic idea: Trace rays starting at the camera/eye position
- Recursion
- Secondary rays (S, R, T)



P: Primary ray

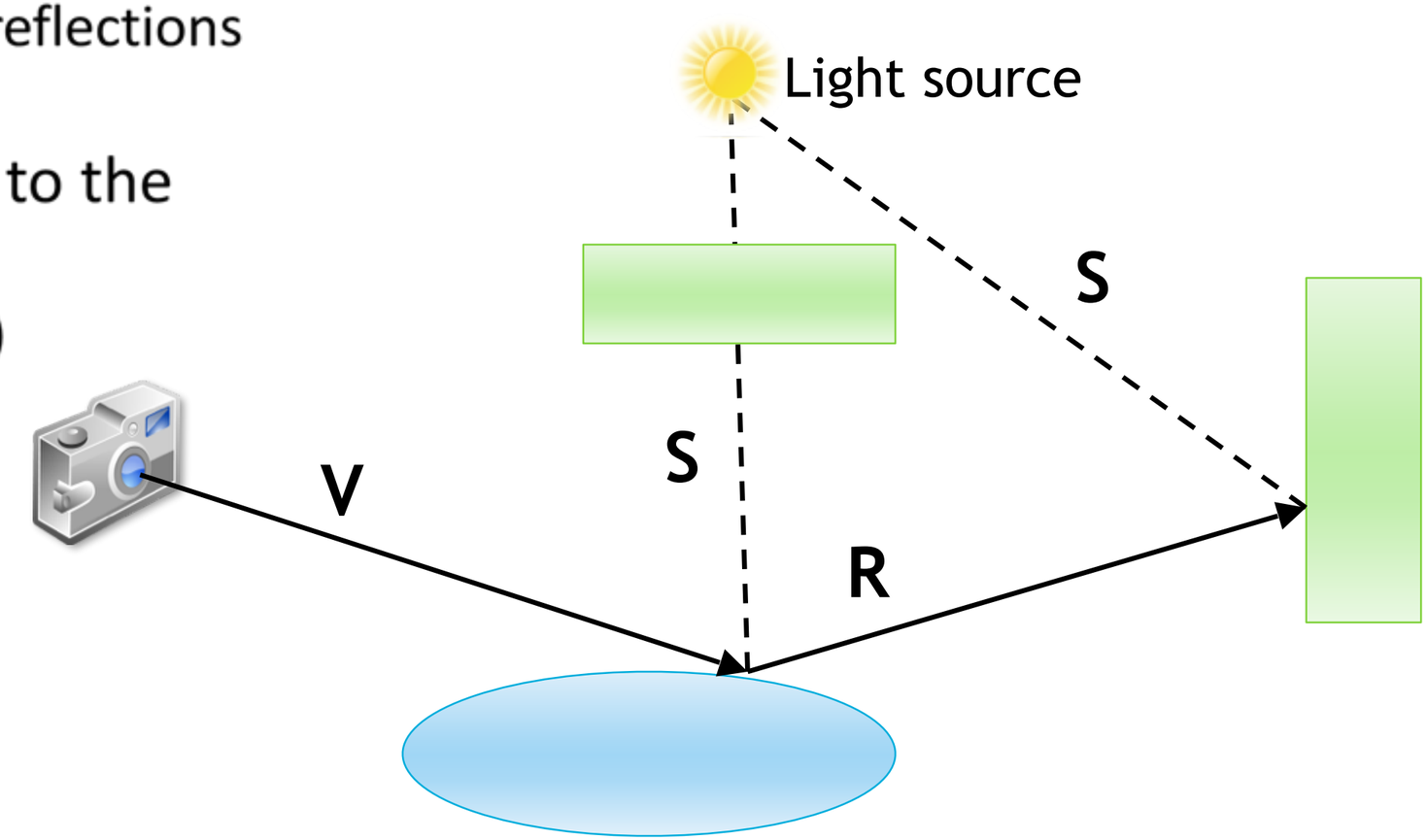
S: Shadow ray

R: Reflection

T: Transmission

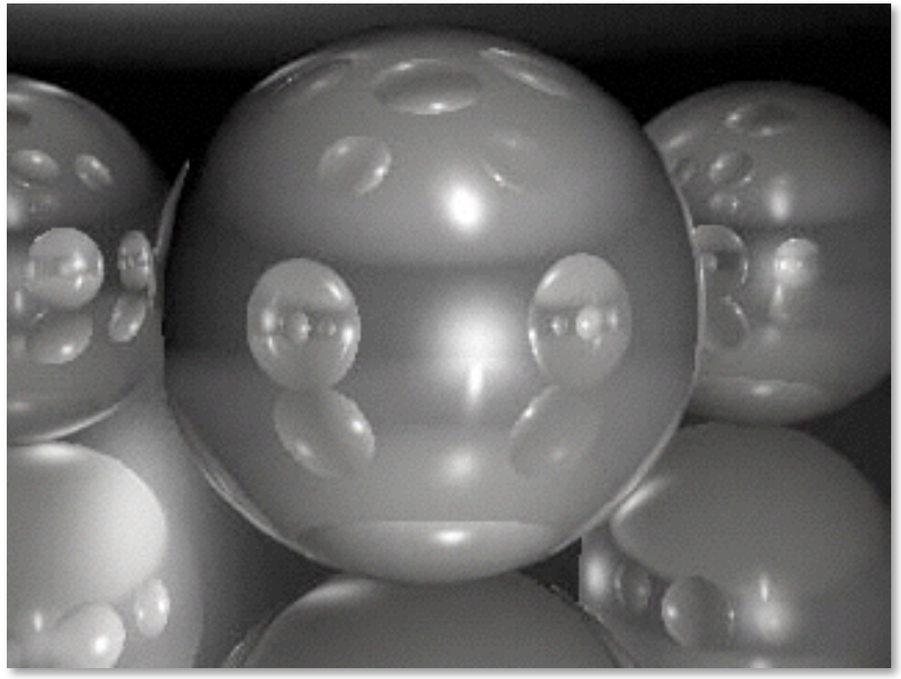
Mirroring, Reflection Rays

- Mirror images of the scene at reflecting surfaces
  - View ray $\mathbf{V}$ is reflected at the surface → Reflection ray $\mathbf{R}$
  - Reflection rays are handled (almost) like view rays: compute the intersection point and return the color
    - Lighting calculations for "view direction" $\mathbf{R}$
    - Recursive tracing of (multiple) reflections
    - Additional shadow rays
  - Add color of the reflection ray to the color at the origin
    - Additional coefficient k_r (or k_t)



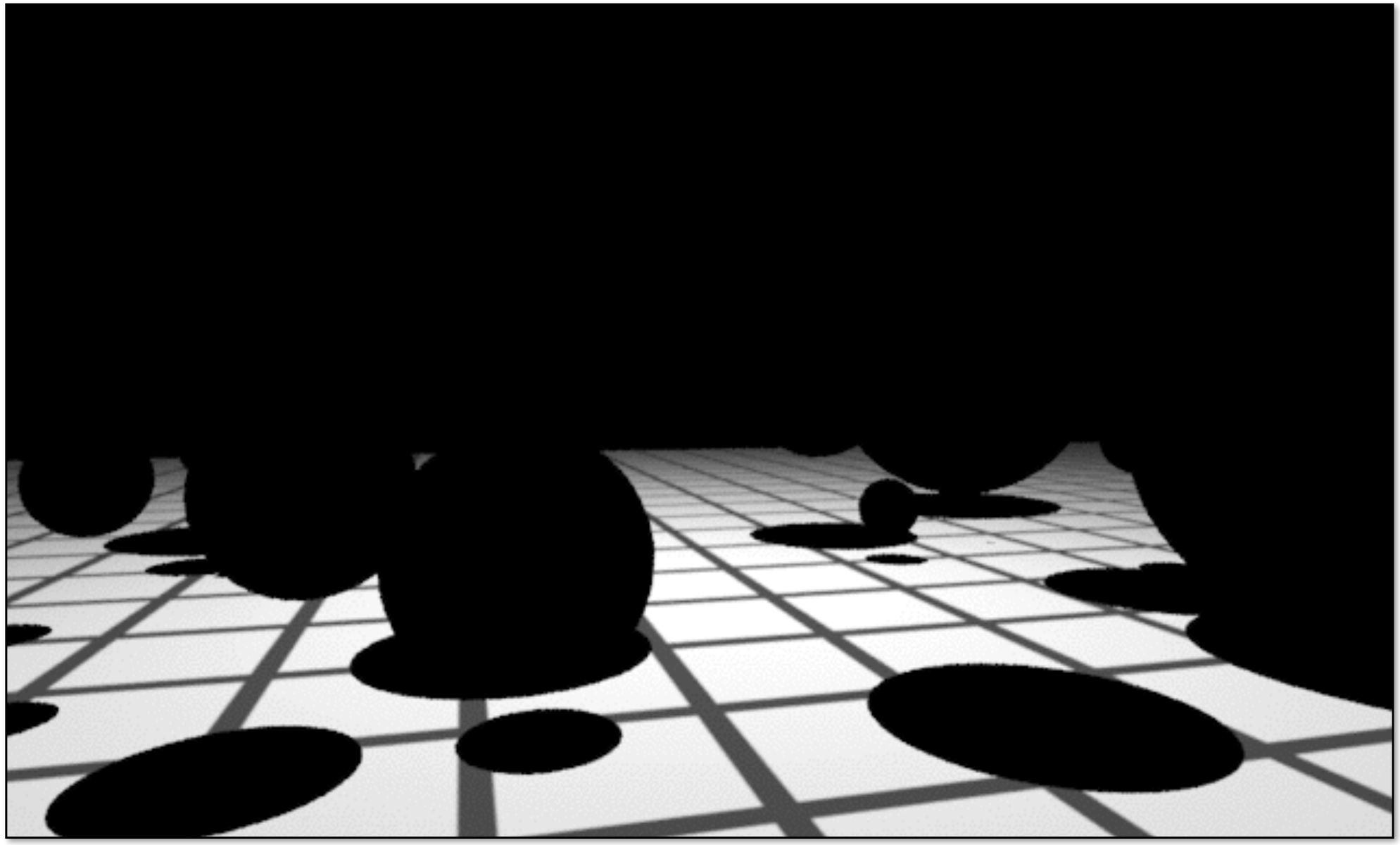
Mirroring, Reflection Rays

- Recursion depth
  - (Predefined) maximum number of reflections
  - Recursion until the contribution to the color is negligible (convergence)
    - Contribution of 1. reflection: $k_{r,1}$
    - Contribution of 2. reflection : $k_{r,1} \cdot k_{r,2}$
    - Trace until $\prod_i k_{r,i} < \epsilon_r$



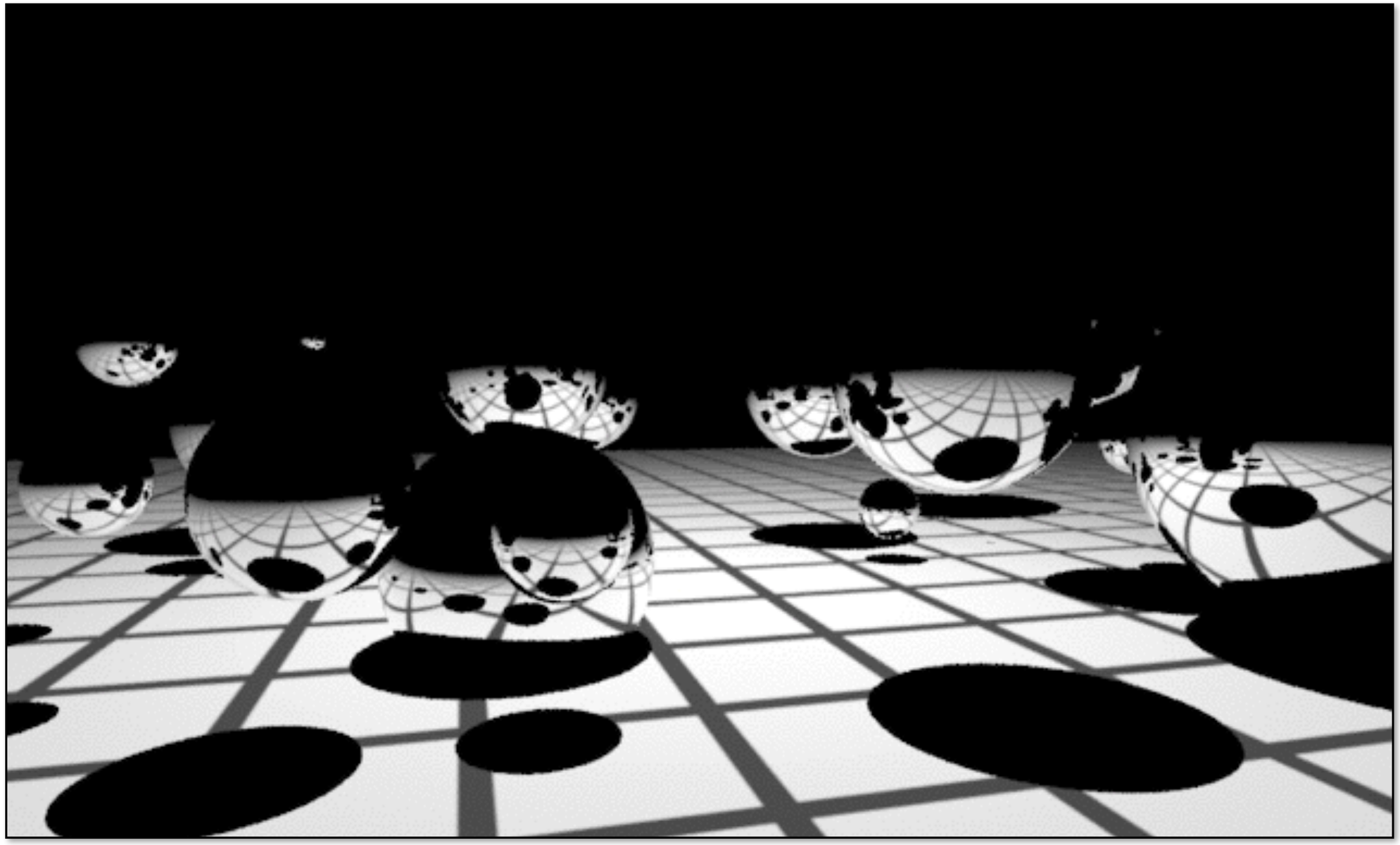
Metal Spheres – Recursion depth 0

- Images courtesy of Pat Hanrahan



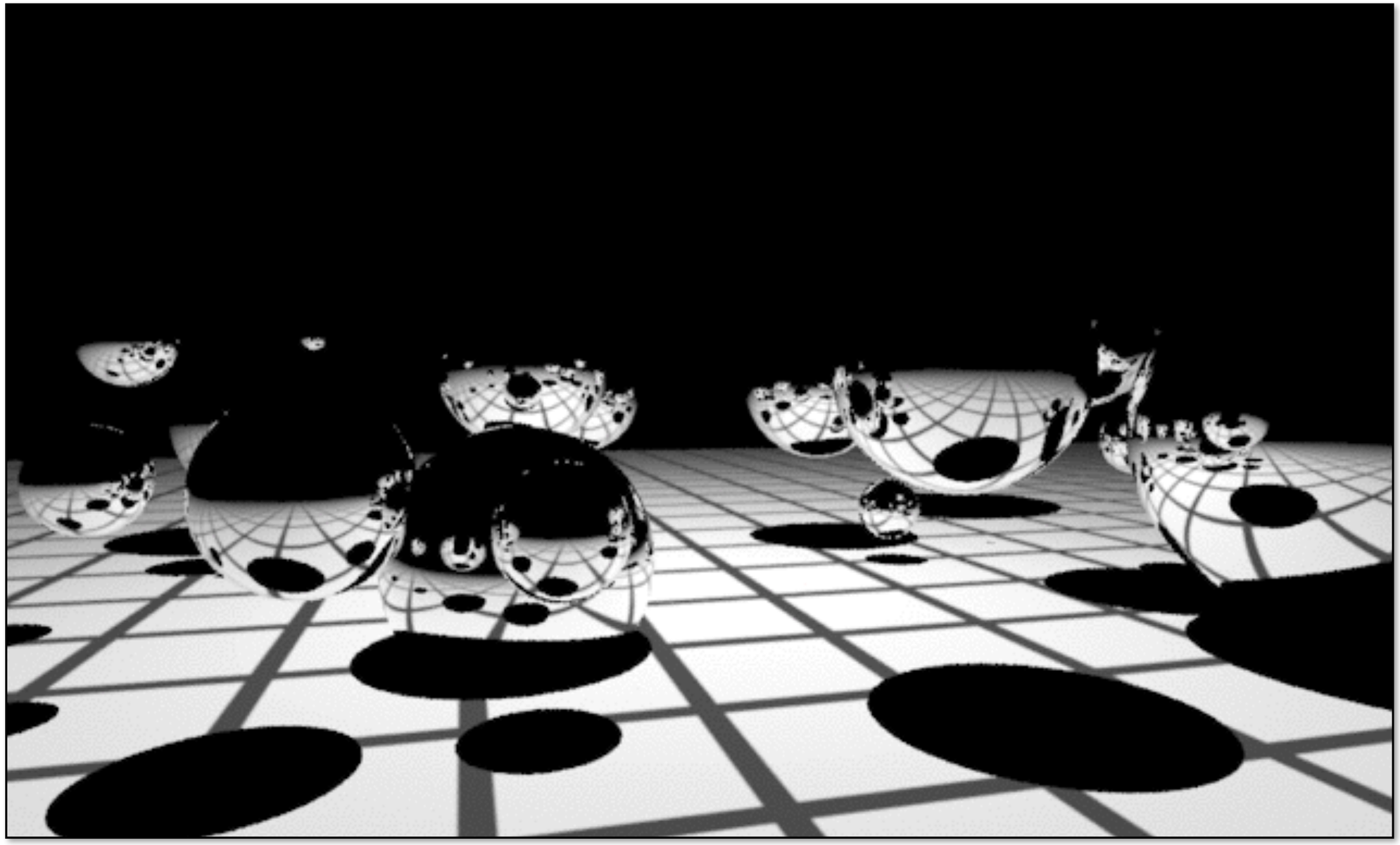
Metal Spheres – Recursion depth 1

- Images courtesy of Pat Hanrahan



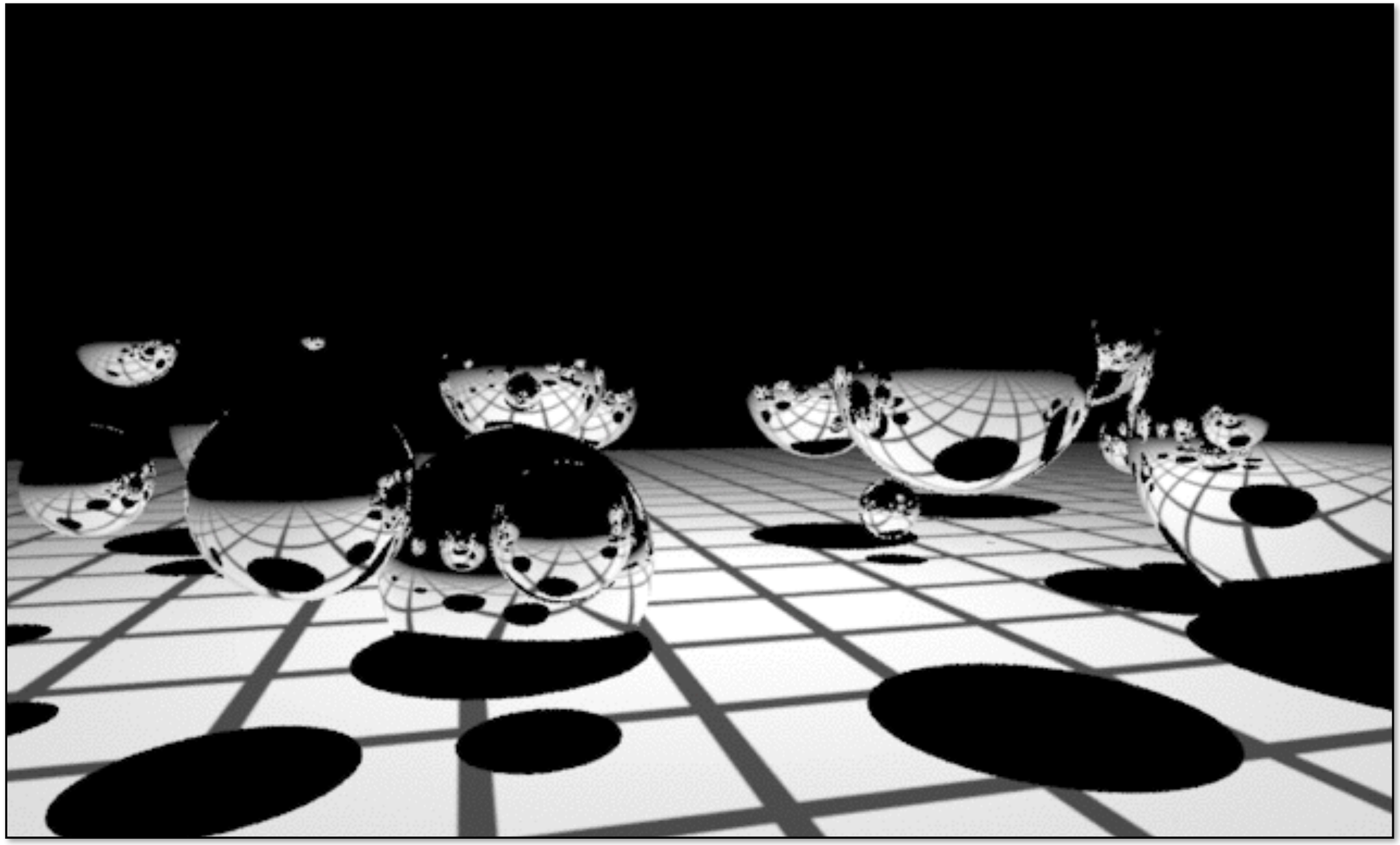
Metal Spheres – Recursion depth 2

- Images courtesy of Pat Hanrahan



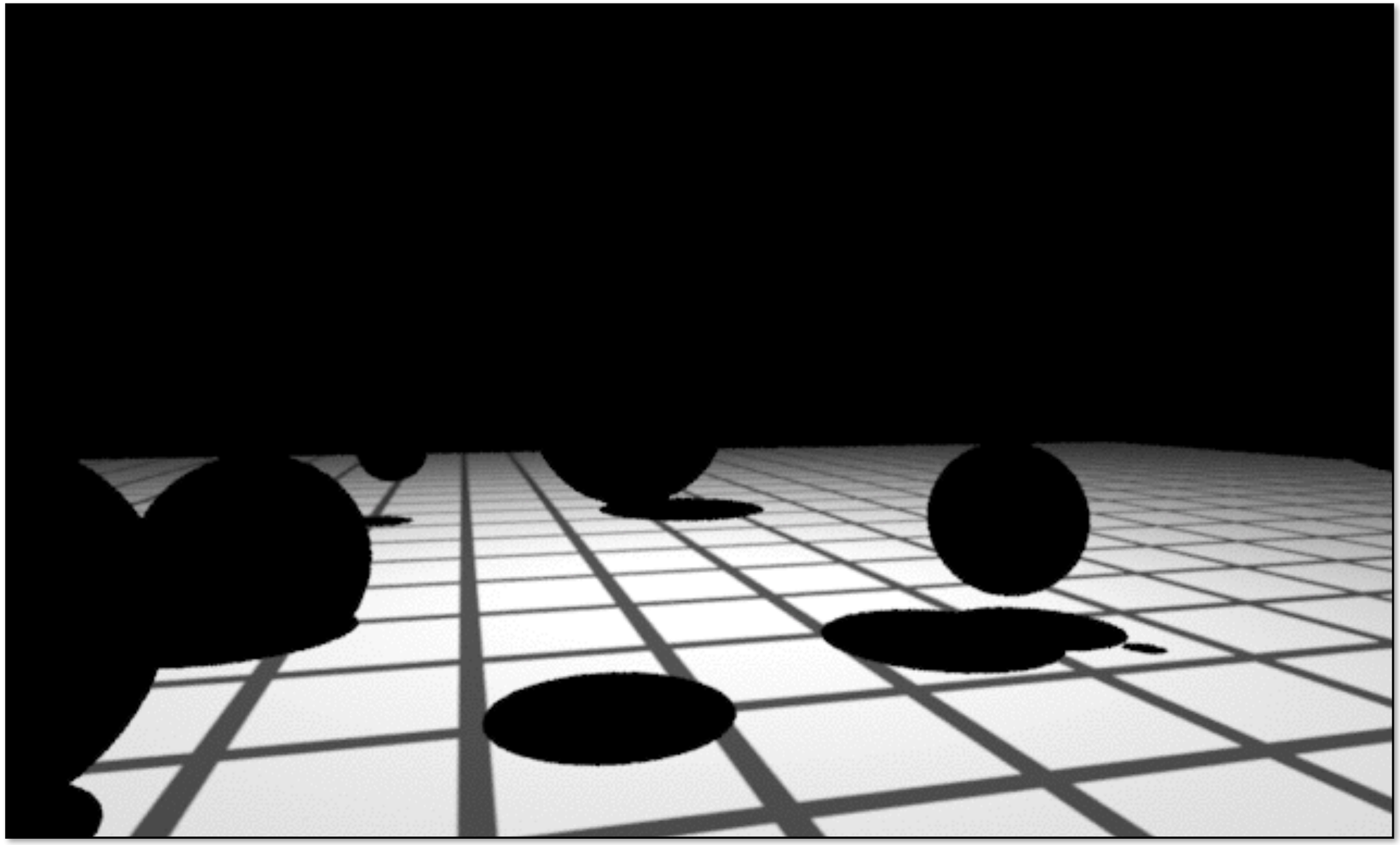
Metal Spheres – Recursion depth 5

- Images courtesy of Pat Hanrahan



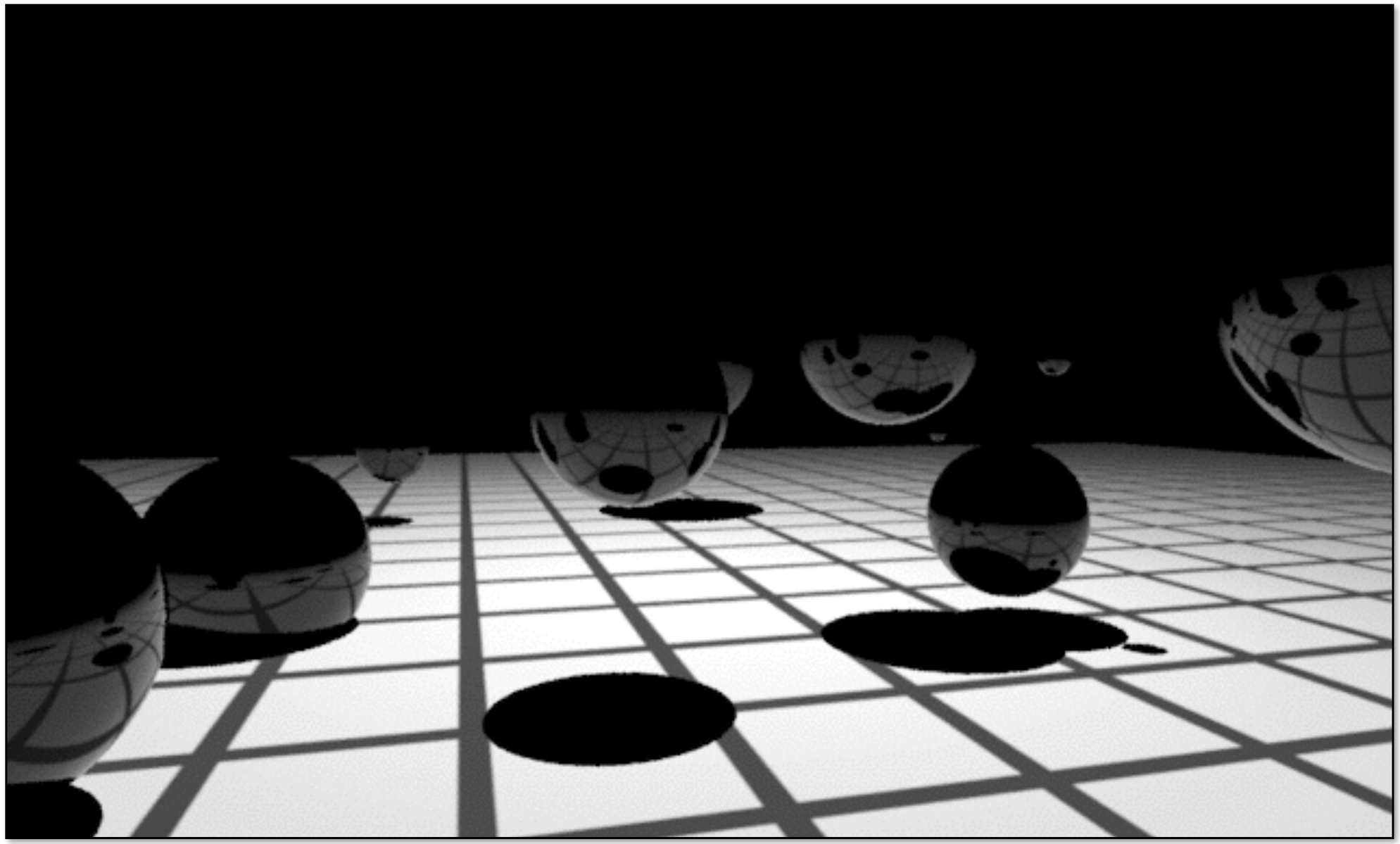
Glass Spheres – Recursion depth 0

- Images courtesy of Pat Hanrahan



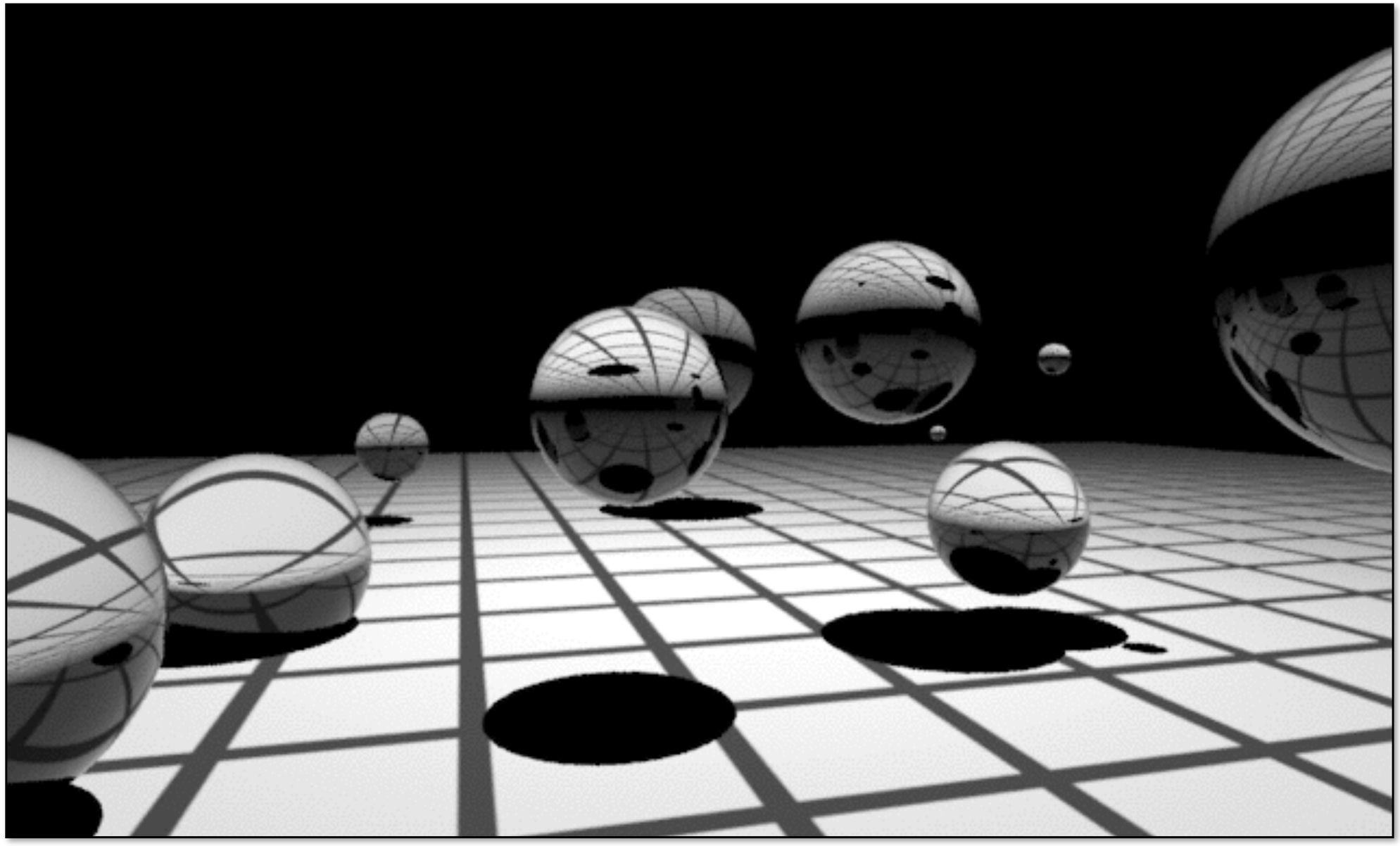
Glass Spheres – Recursion depth 1

- Images courtesy of Pat Hanrahan



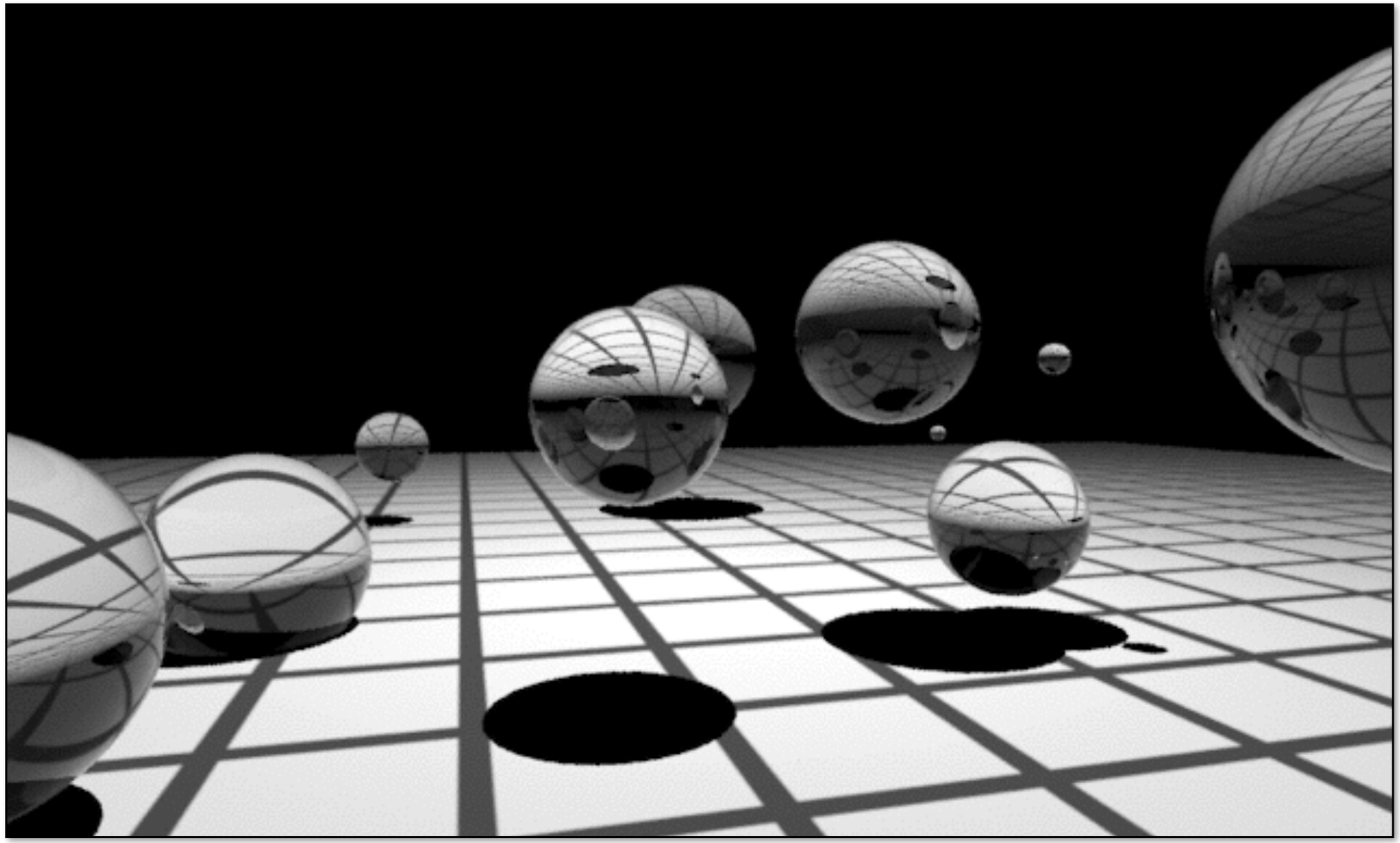
Glass Spheres – Recursion depth 2

- Images courtesy of Pat Hanrahan



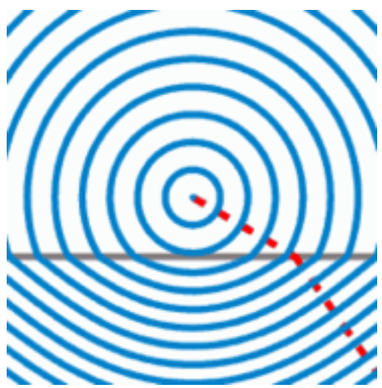
Glass Spheres – Recursion depth 5

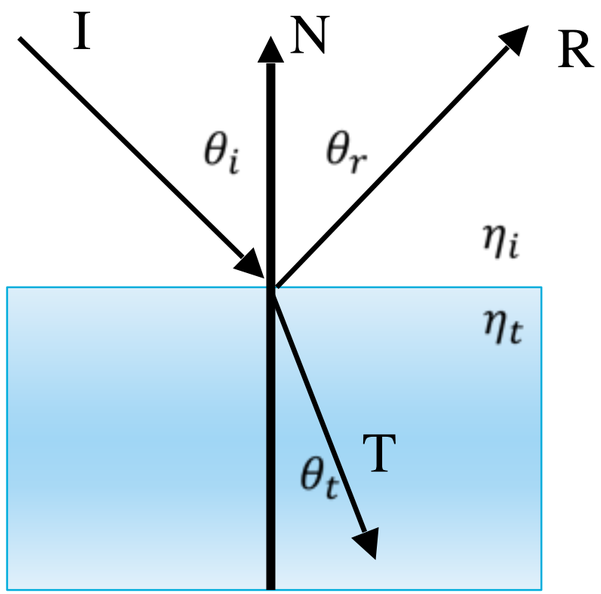
- Images courtesy of Pat Hanrahan

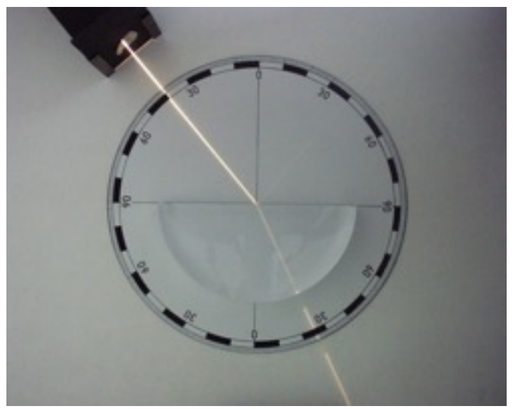


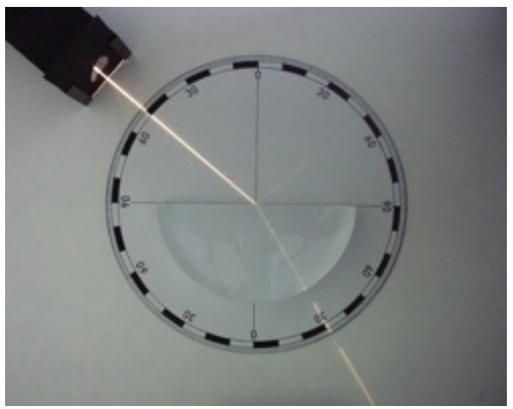
Refraction – Snell's Law

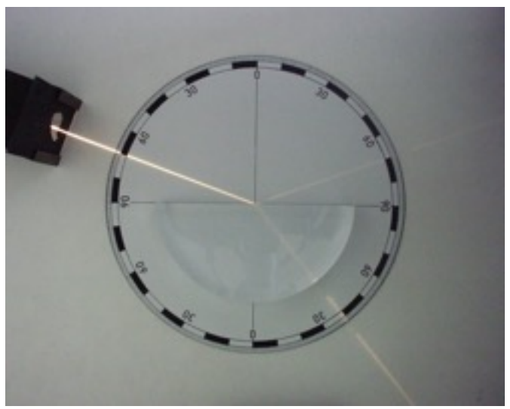
- Snell's law described the change of direction for a wave (light) if it crosses the boundary between media with different refraction coefficients
- Licht moves at different speed in different media
  - (real) refraction coefficient $\eta = {}^{c_0}/_{c_\eta}$
  - c_0 phase speed in vacuum
  - c_η phase speed in the medium
- Refraction coefficient usually depends on wave length









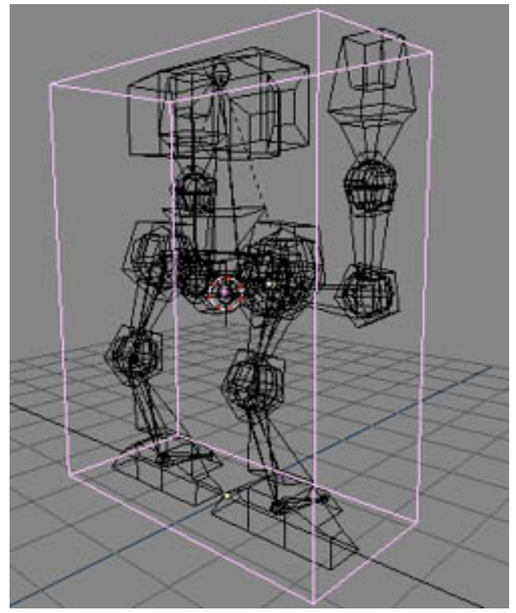


Images: Wikipedia

Brainstorming: What Makes Ray Tracing Hard?

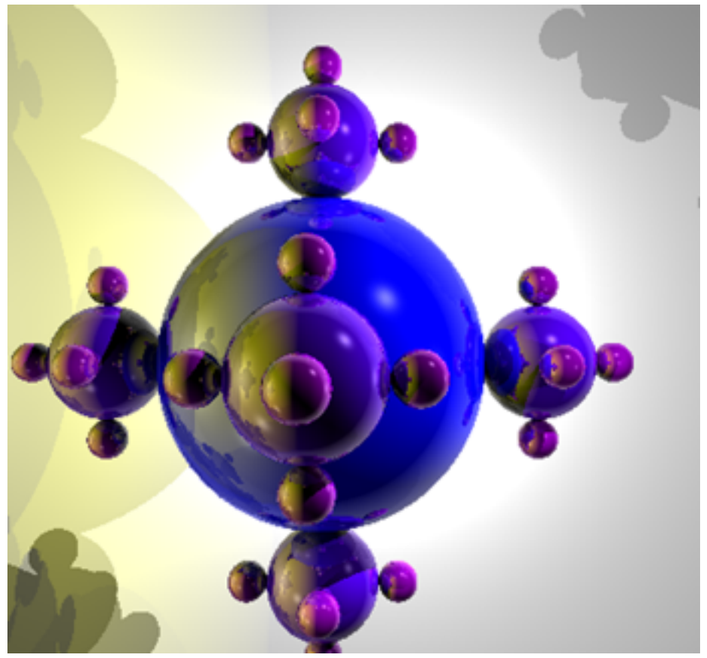
Optimizations for Ray Tracing

- Bounding volumes:
  - Instead of calculating intersection with individual objects, first calculate intersection with a volume containing several objects
  - Can decrease computation time to less than linear complexity (in number of existing objects)



https://sites.google.com/site/axeltp28/bounding-box

- Adaptive recursion depth control
  - Maximum recursion limit is not always necessary
  - Recursion should be stopped as soon as possible
  - E.g., stop if intensity change goes below a threshold value



http://www.emeyex.com/index.php?page=raytracer

- Monte Carlo Methods
  - Improve complexity (cascading recursion = exponential)
  - Use ***one*** random ray for recursive tracing (instead of refracted/reflected rays)
  - Carry out multiple experiments (e.g. 100) and compute average values

Real Time Ray Tracing

- Various optimizations presented over the last few years
  - Libraries for CPU-/GPU-accelerated ray tracing (e.g., Intel OSPRay, Nvidia OptiX)
- Real time ray tracing has become feasible

`http://openrt.de/` (images from there, now offline)

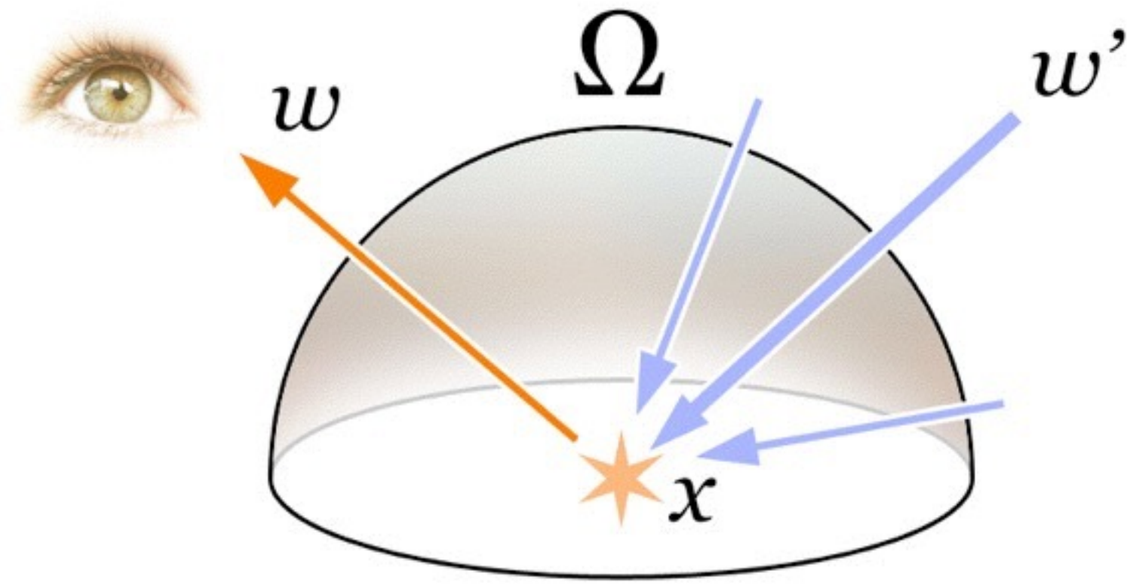
Chapter 7 – Shading and Rendering

- Local Illumination Models: Shading
- Global Illumination: Ray Tracing
- Global Illumination: Radiosity
- Monte-Carlo Methods
- Non-Photorealistic Rendering

Reminder: The Rendering Equation [Kajiya '86]

$$I_o(x,\vec{\omega}) = I_e(x,\vec{\omega}) + \int_\Omega f_r(x,\vec{\omega}',\vec{\omega}) I_i(x,\vec{\omega}')(\vec{\omega}' \cdot \vec{n}) d\vec{\omega}'$$

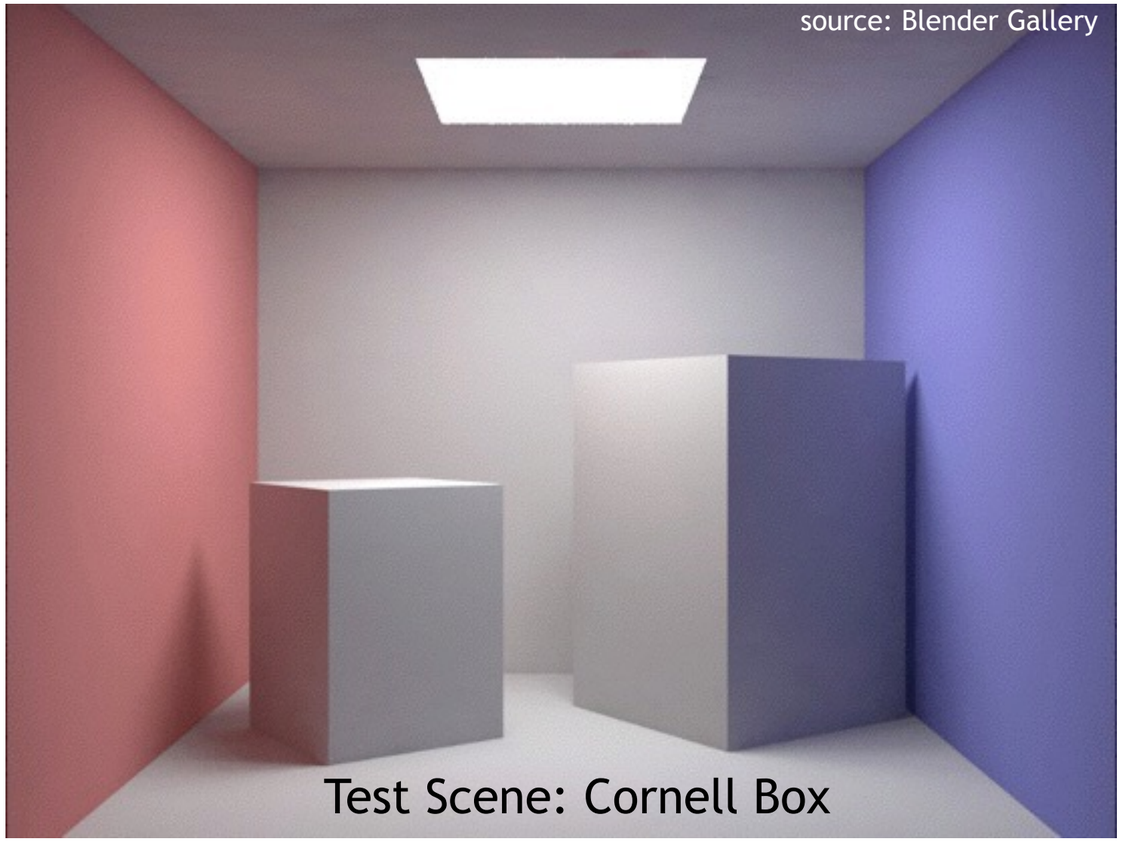
- I_o = outgoing light
- I_e = emitted light
- Reflectance Function
- I_i = incoming light
- Angle of incoming light
- Describes all flow of light in a scene in an abstract way



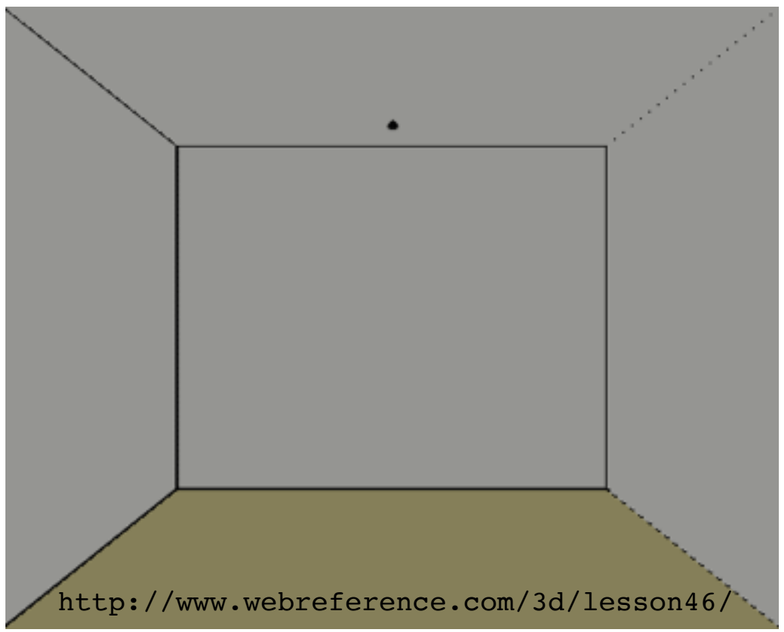
http://en.wikipedia.org/wiki/File:Rendering_eq.png

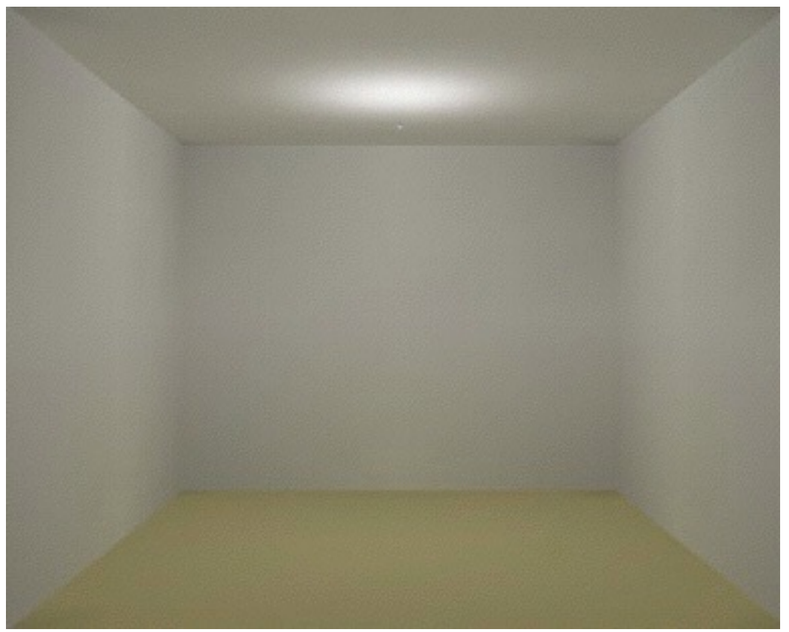
Global Illumination: Radiosity

- Simulation of energy flow in scene
- Can show „color bleeding“
  - Example: blueish and reddish sides of boxes
- Naturally deals with area light sources
- Creates soft shadows
- Only uses diffuse reflection
  - does not produce specular highlights



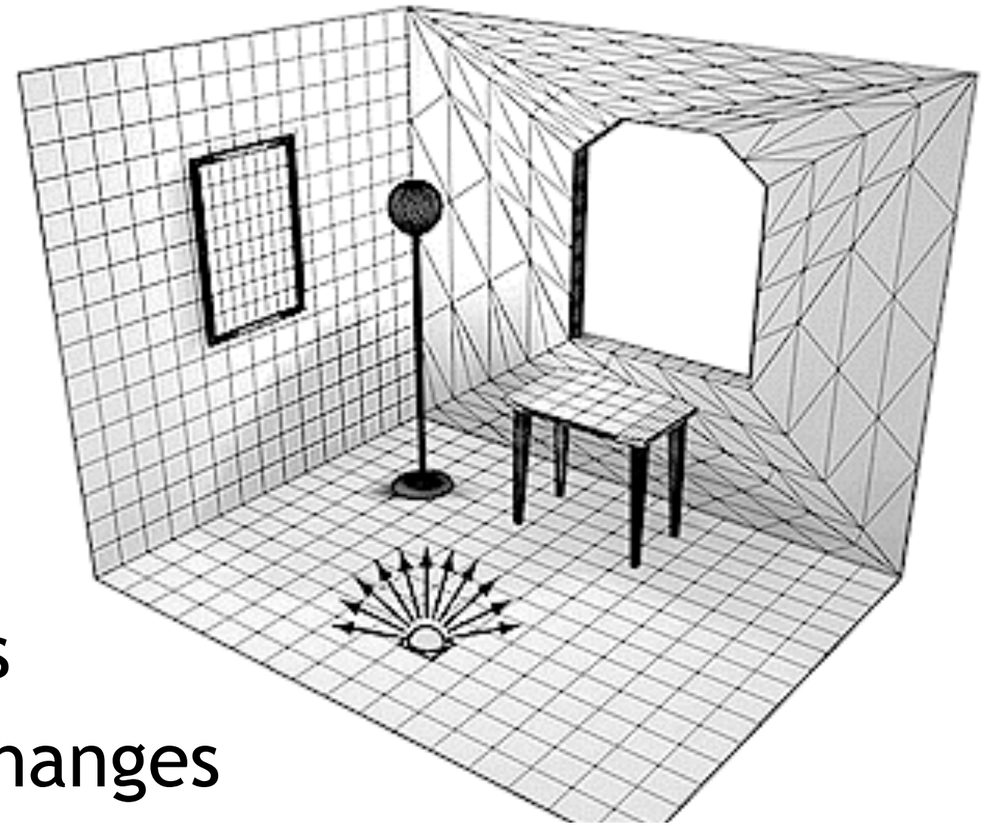
Test Scene: Cornell Box





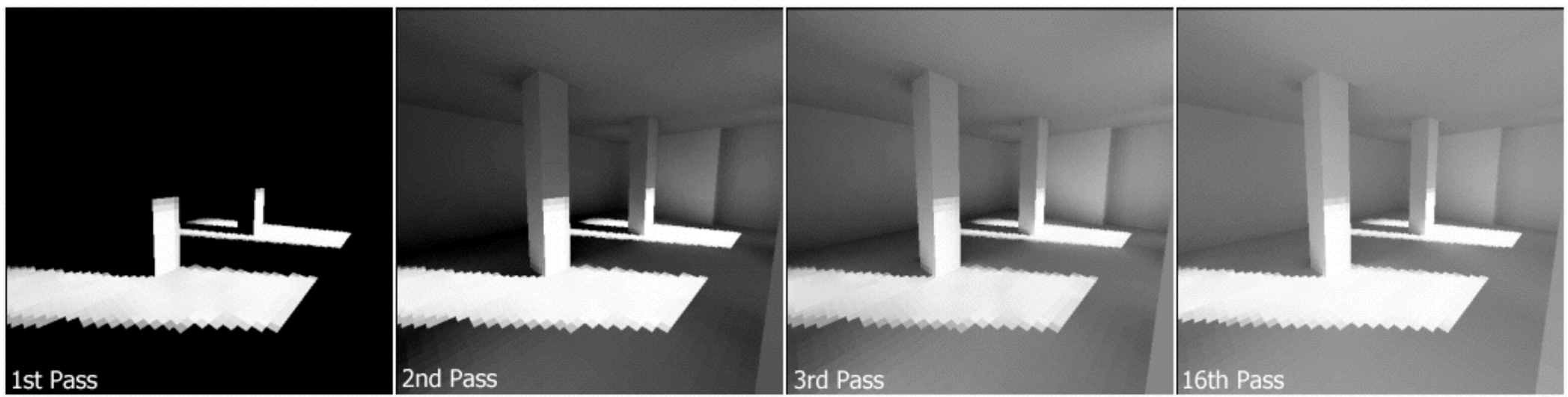
Radiosity Algorithm

- Divide all surfaces into small patches
- For each patch determine its initial energy
- Loop until close to energy equilibrium
  - Loop over all patches
    - determine energy exchange with every other patch
- „Radiosity solution“: energy for all patches
- Recompute if _____ changes



http://pclab.arch.ntua.gr/03postgra/mladenstamenico/ (probably not original)

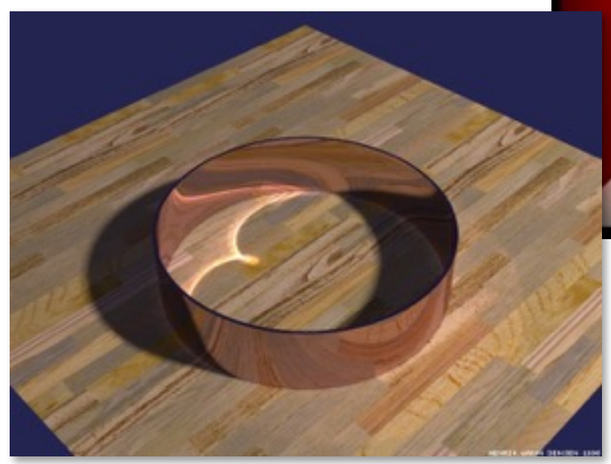
http://en.wikipedia.org/wiki/File:Radiosity_Progress.png



source: Blender Gallery

Combinations

- Ray Tracing is adequate for reflecting and transparent surfaces
- Radiosity is adequate for the interaction between diffuse light sources
- What we want is a combination of the two!
  - This is non-trivial, a simple sequence of algorithms is not sufficient
- State-of-the-art “combination”(more like another innovative approach): *Photon Mapping* [Jensen 1996]
  - First step:
    - Inverse ray tracing with accumulation of light energy
    - Photons are sent from light sources into scene, using Monte Carlo approach
    - Surfaces accumulate energy from various sources
  - Second step:
    - “Path tracing” (i.e. Monte Carlo based ray tracing) in optimized version (e.g. only small recursion depth)



www.cs.cmu.edu/

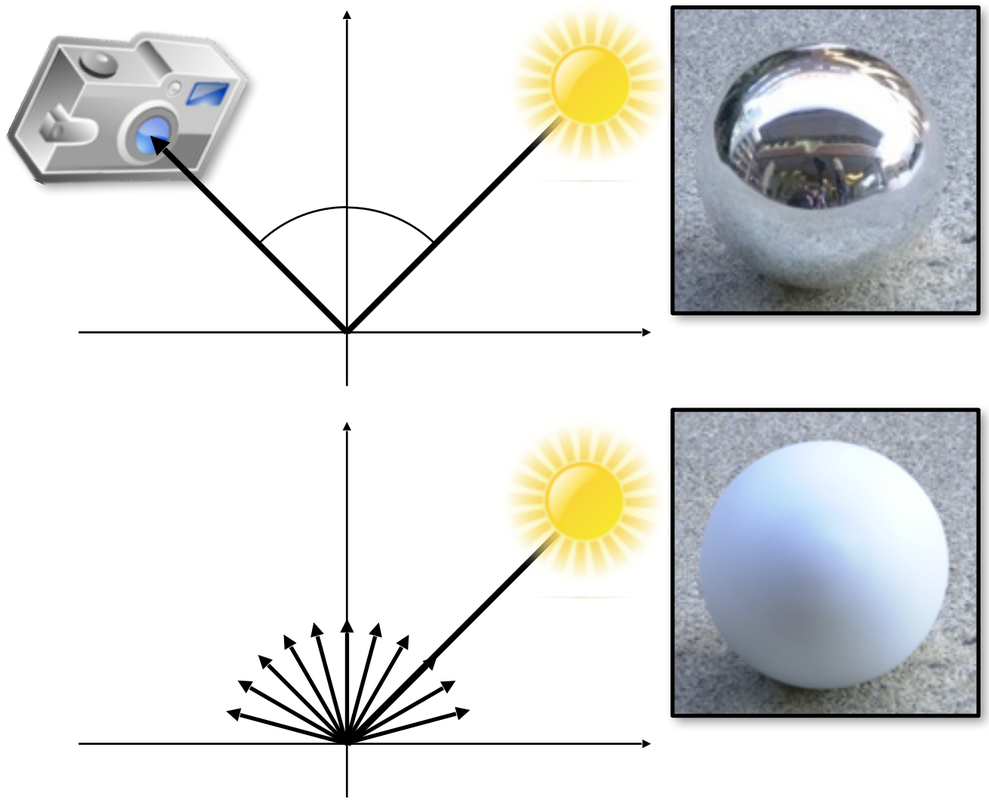
Chapter 7 – Shading and Rendering

- Local Illumination Models: Shading
- Global Illumination: Ray Tracing
- Global Illumination: Radiosity
- Monte-Carlo Methods
- Non-Photorealistic Rendering

Reflection

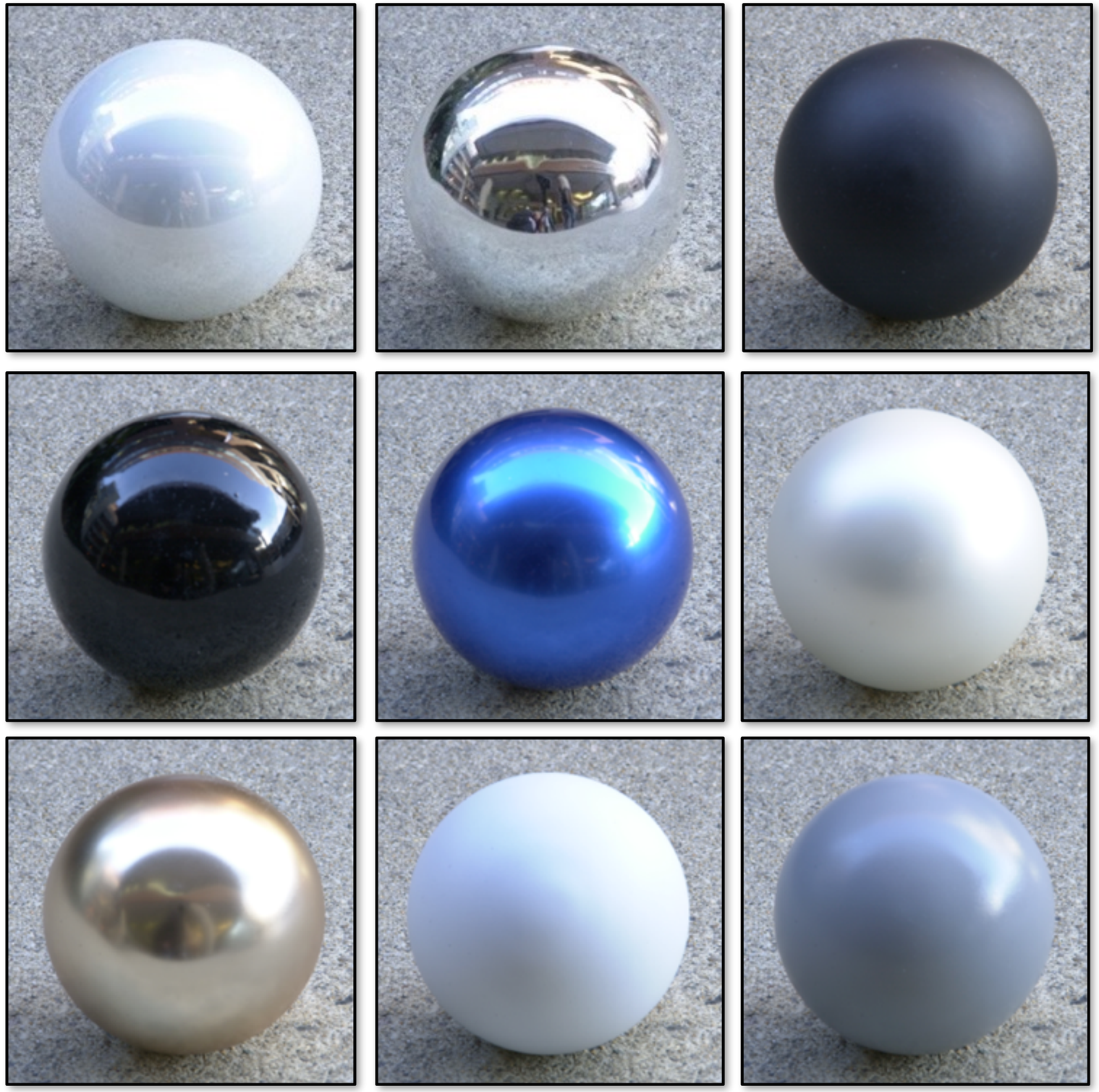
- Shading and lighting are essential for 3D appearance of objects
- Interaction between
  - Light sources (intensity, color, position, …) and
  - Surfaces (Material properties, orientation relative to light spoiurces)

- Two basic types of reflection
  - Specular reflection
    - incident angle = emergent angle
  - Diffuse reflection
    - same reflection in all directions



Reflection

- These spheres just look different because they have different **reflection properties**

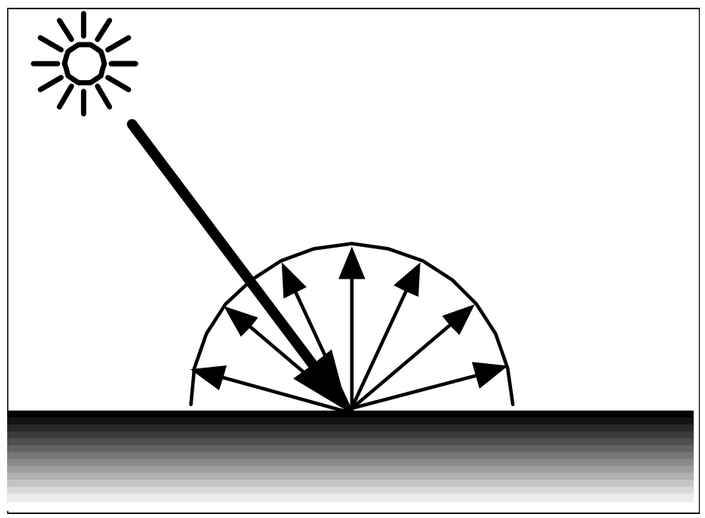


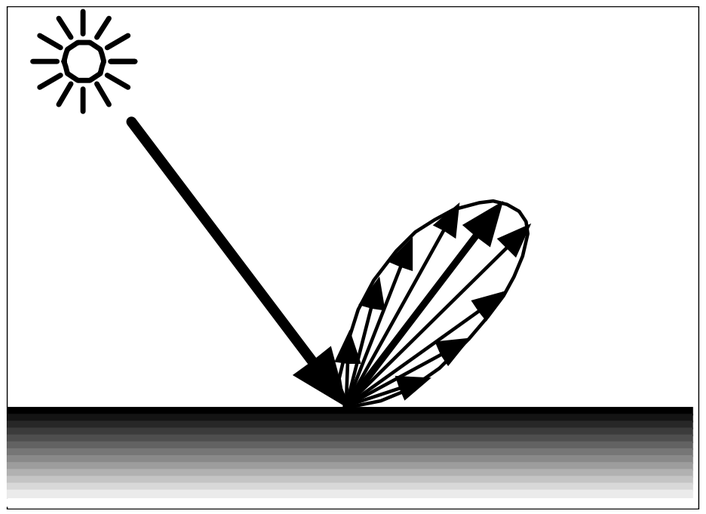
Reflection: Terminology

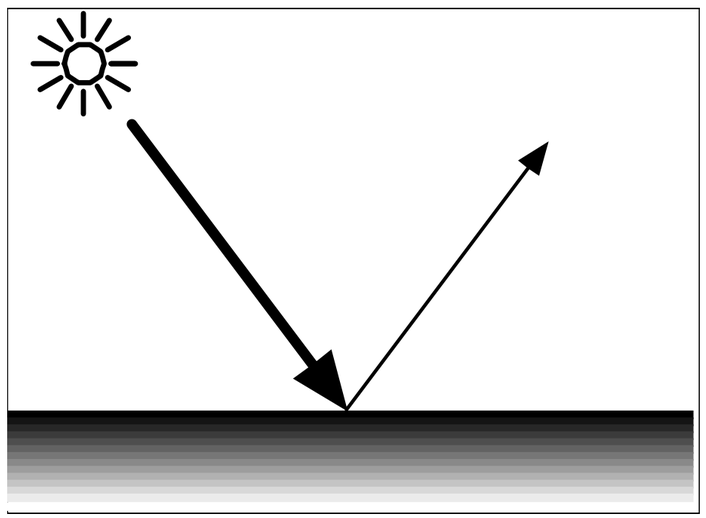
diffuse

glossy

mirroring

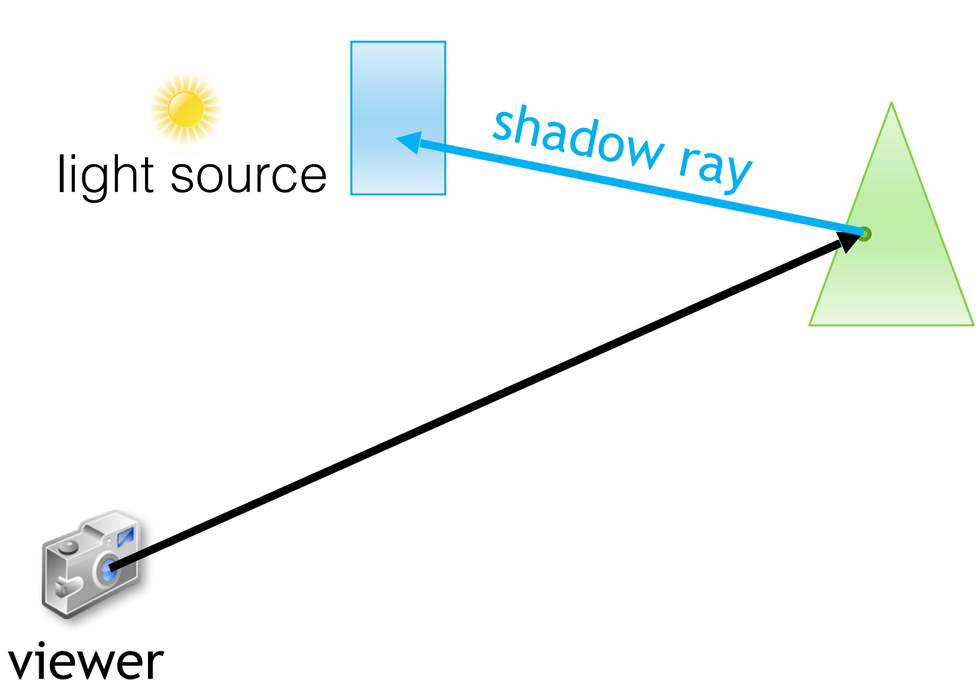




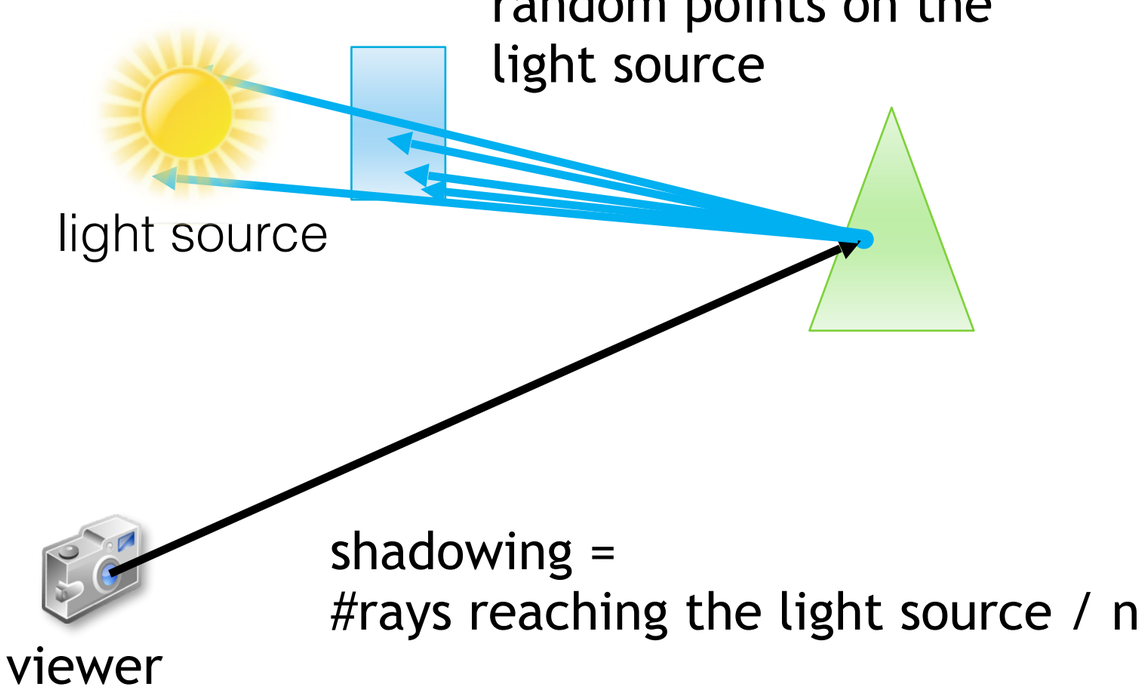


Distributed Ray Tracing

- „Problems“ of the original Ray Tracing
  - images look too perfect:
    - perfect mirroring and transmission
    - hard shadow borders
    - infinite depth of field etc.
- Example: shadow rays



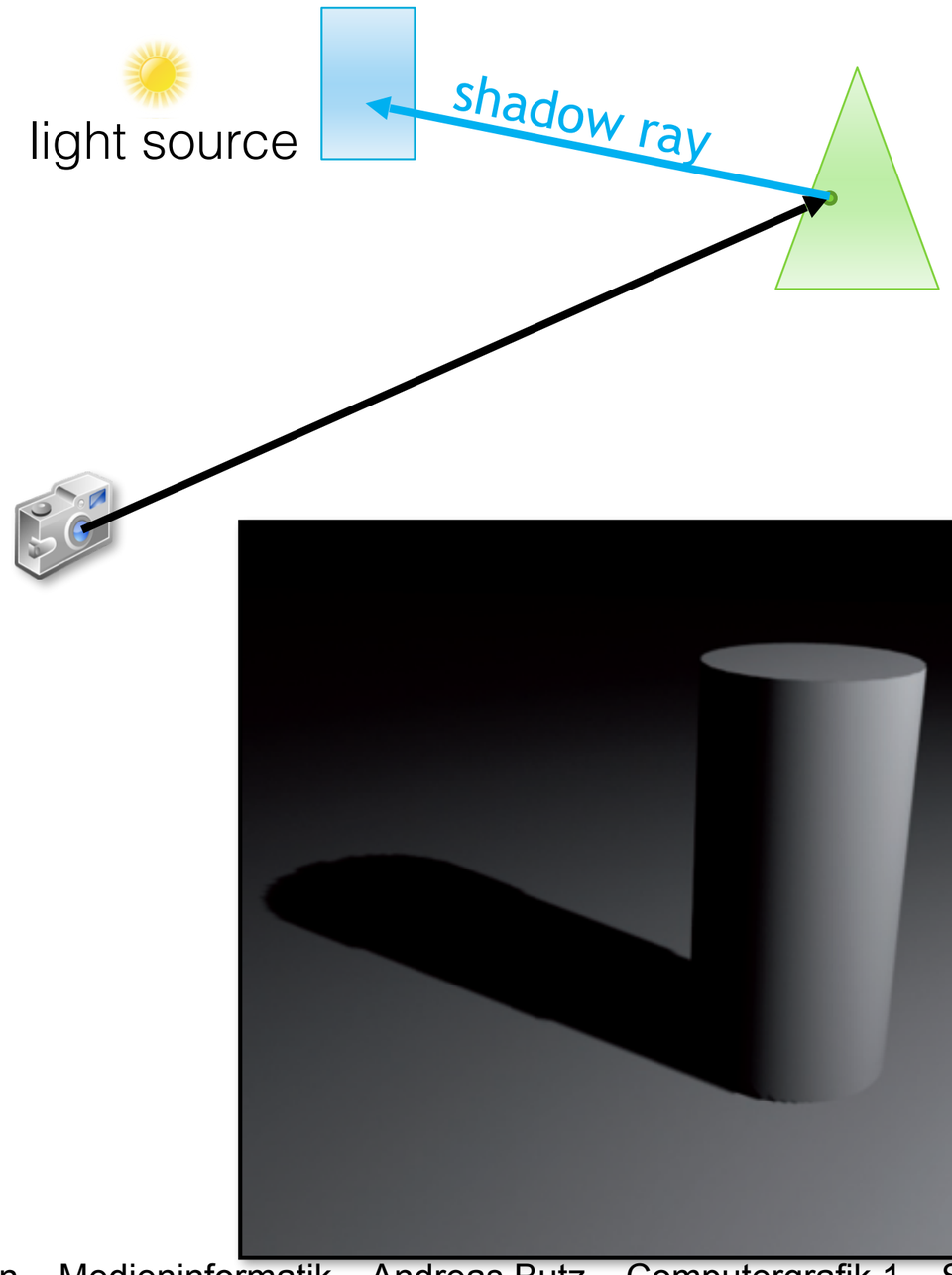
n shadow rays towards random points on the light source

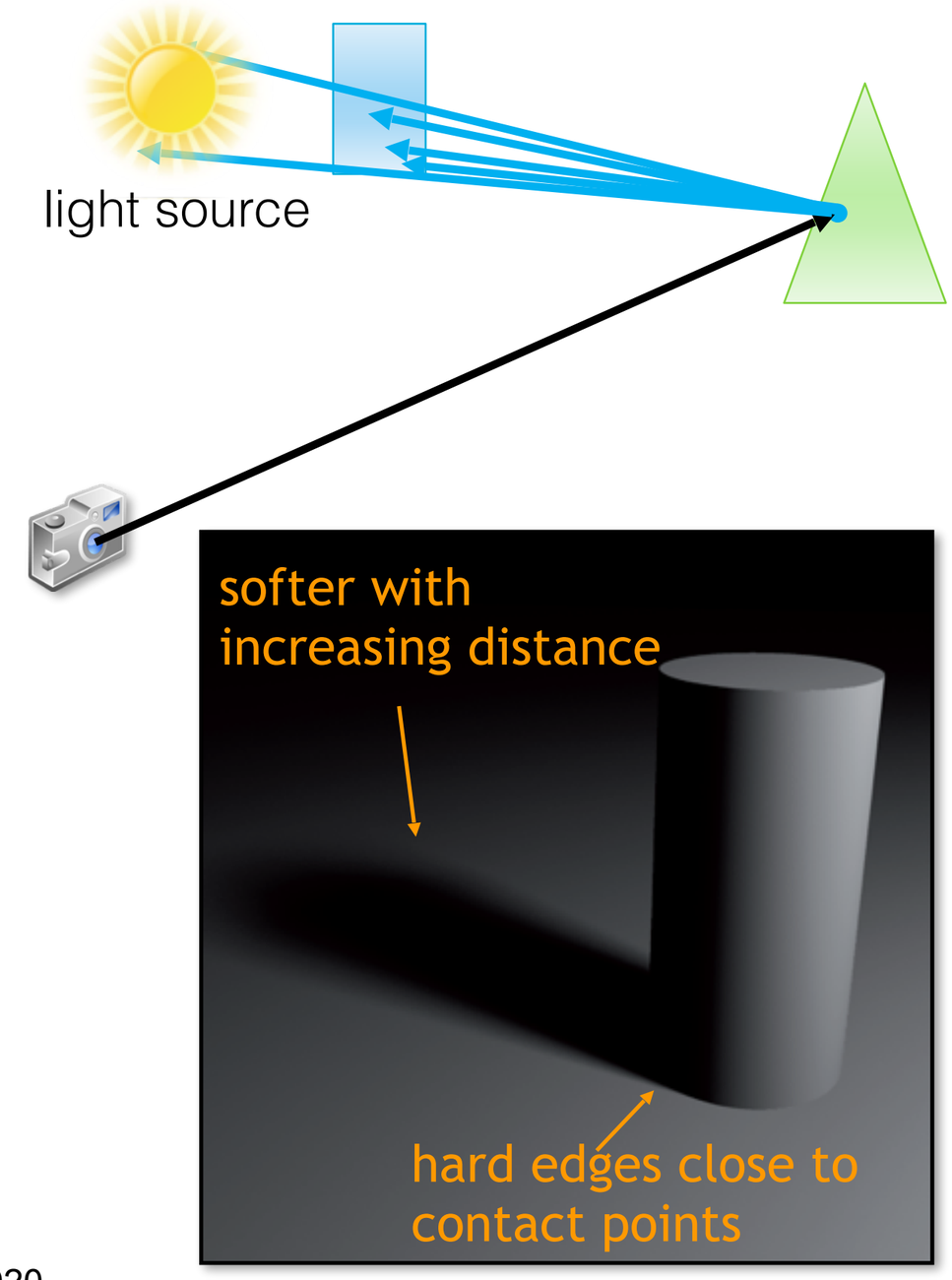


shadowing =
#rays reaching the light source / n

Soft Shadows

- real world light sources have a finite area

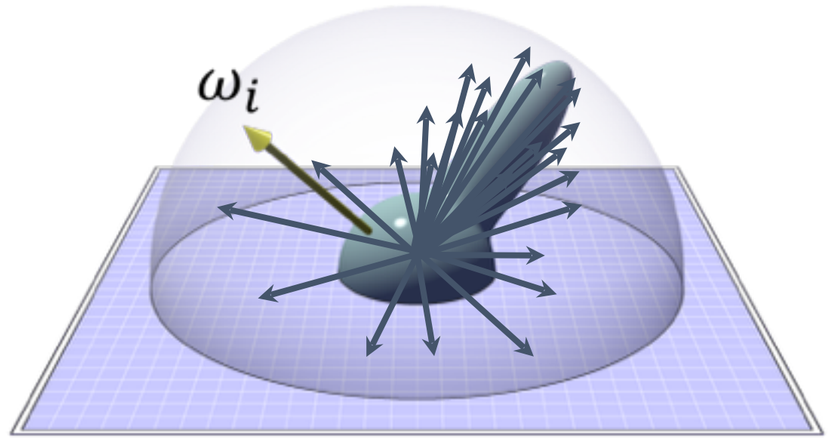




Imperfect Mirroring and Transmission

Distributed Ray Tracing

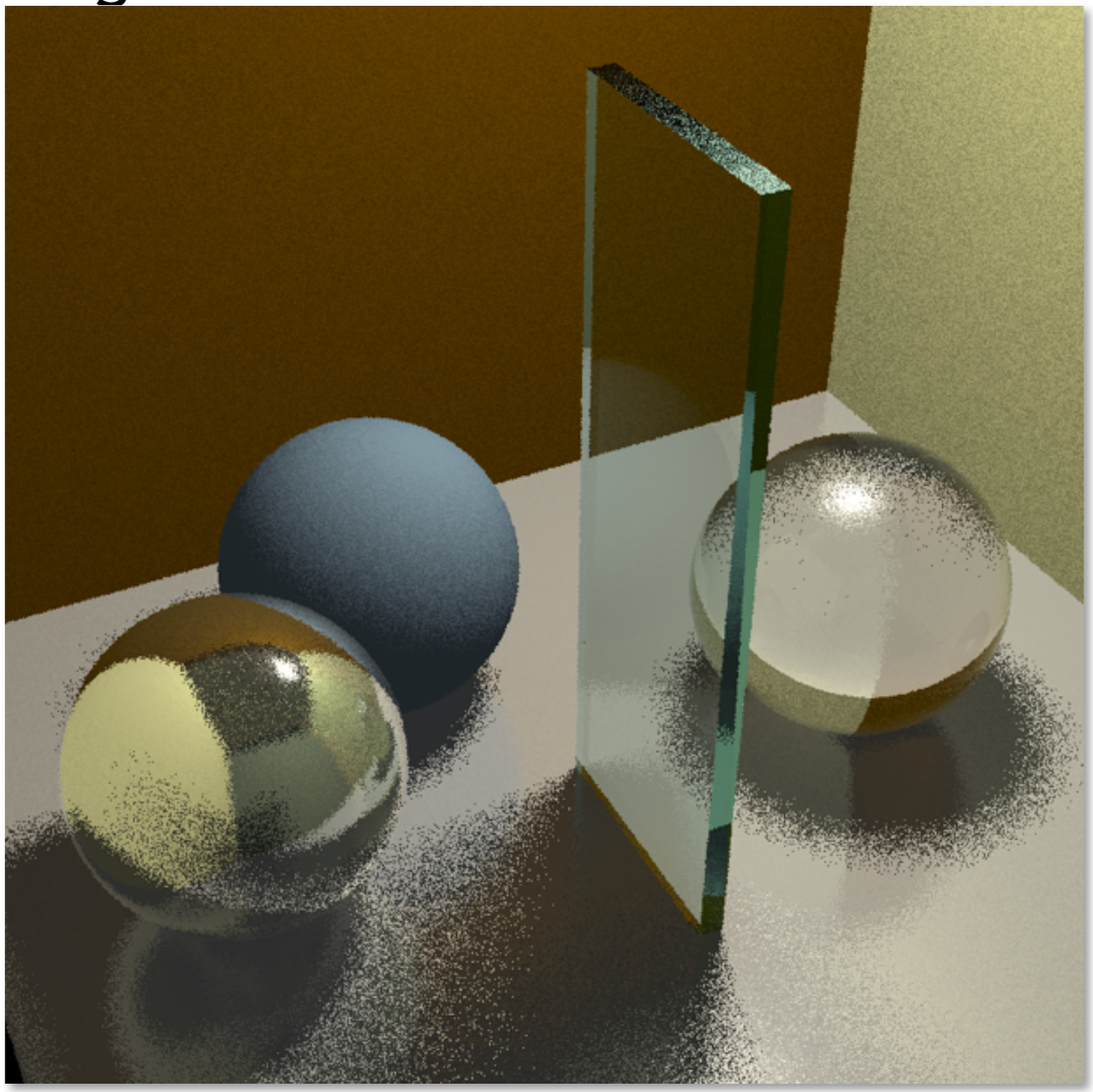
- Several Rays for mirroring and transmission
  - at each intersections, many secondary rays are traced
  - (quasi-)random direction
  - weighting according to material properties



- How many rays are needed?
  - think about recursion and explosion of the tree of rays…

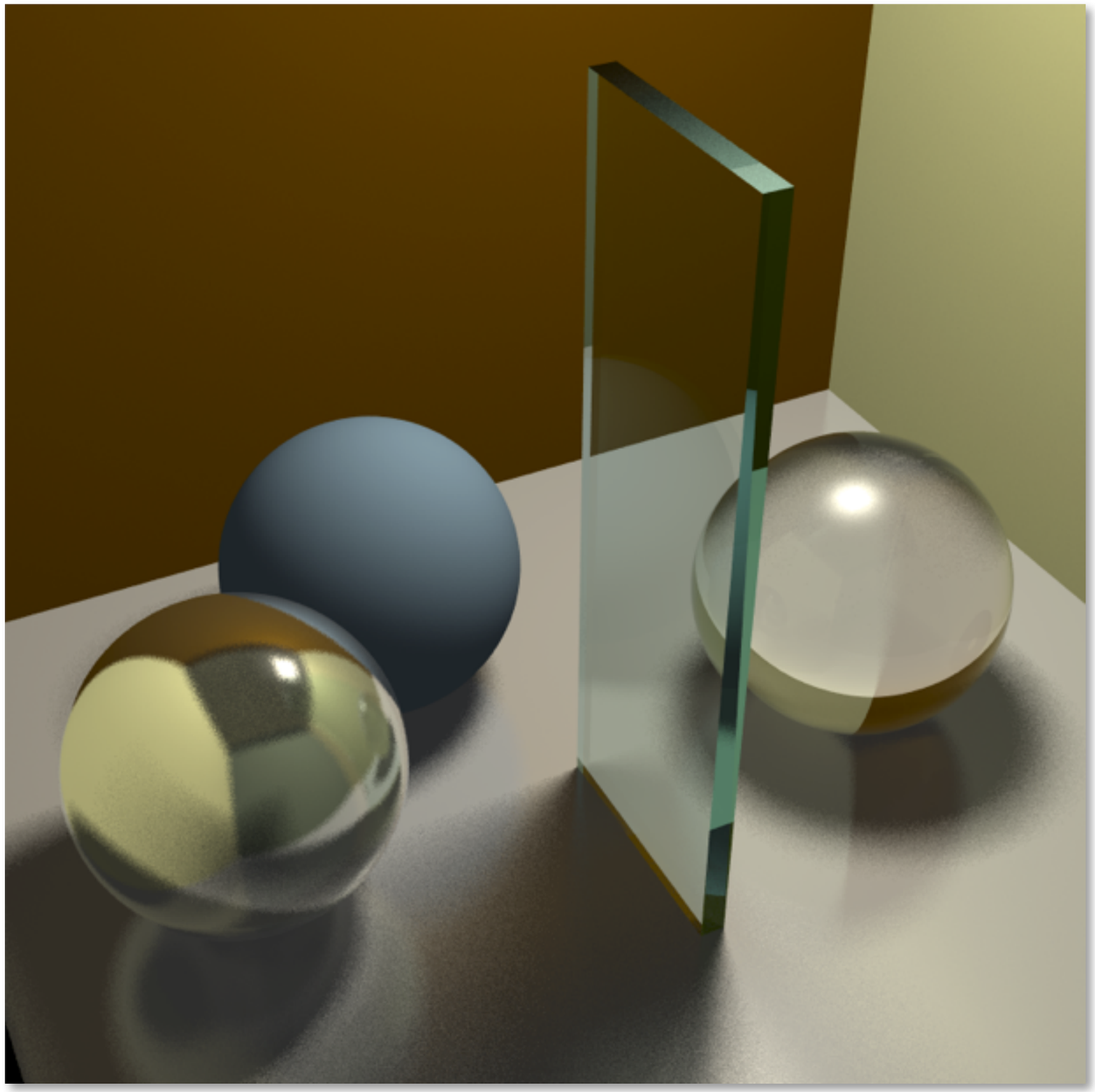
Imperfect Mirroring and Transmission

- 1 random ray: image noise



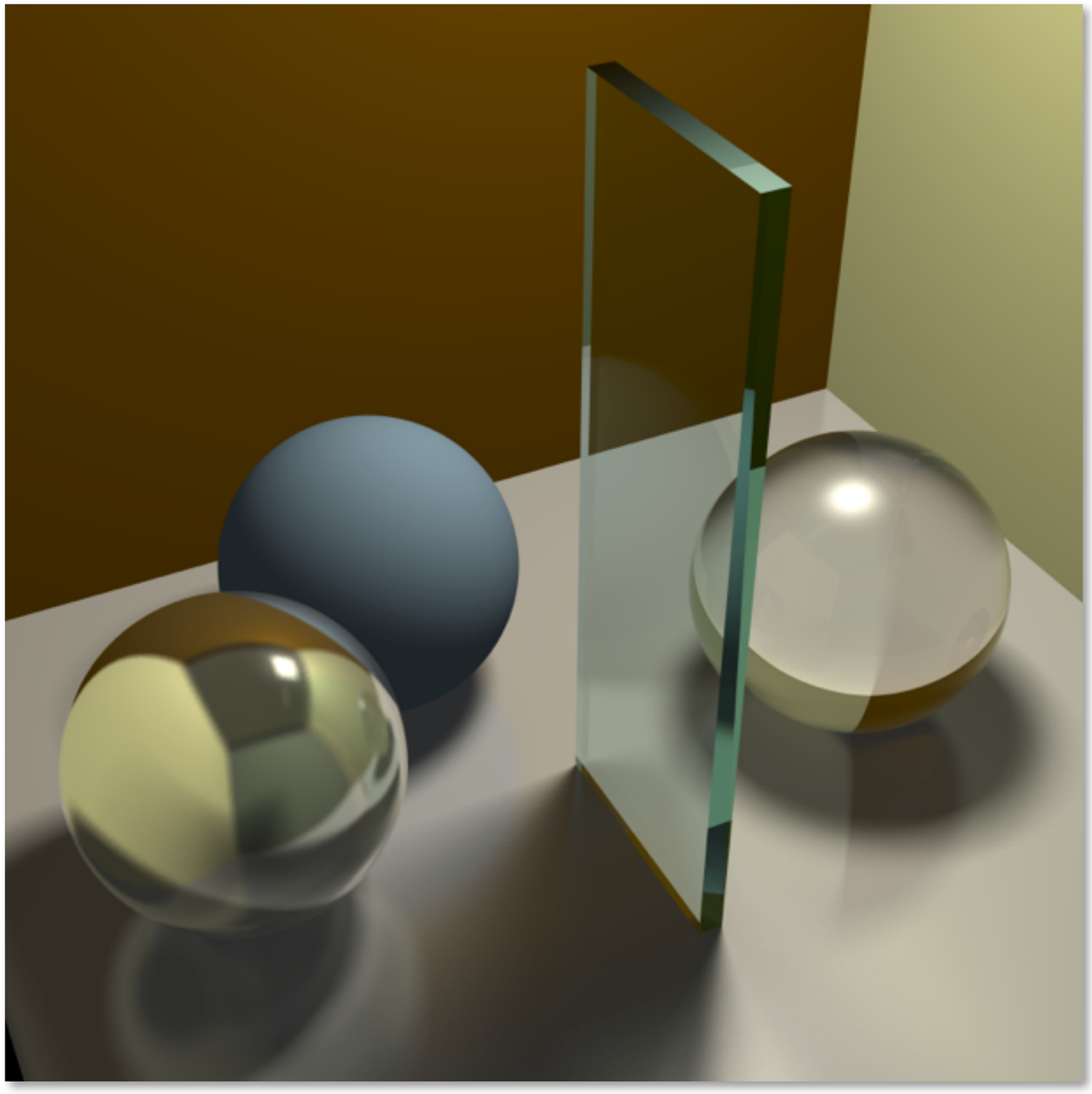
Imperfect Mirroring and Transmission

- 16 random rays



Imperfect Mirroring and Transmission

- 256 random rays



Light Transport, Lighting Simulation

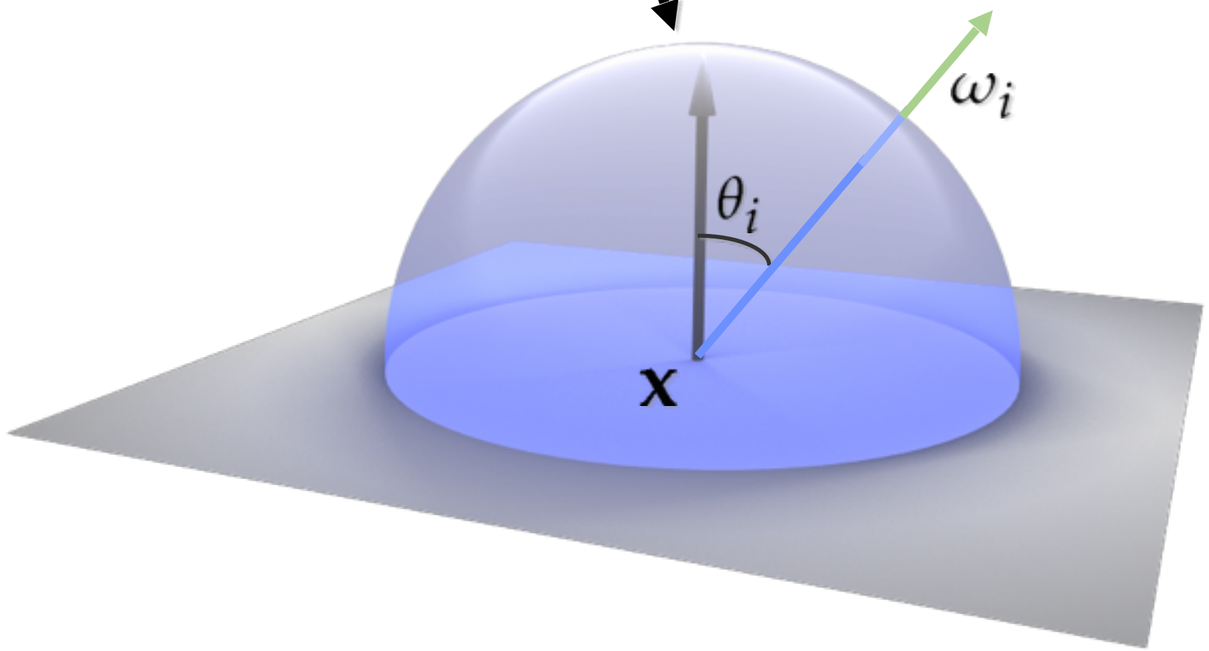
$$L(\mathbf{x},\omega) = E(\mathbf{x},\omega) + \int_{\Omega^+} f(\mathbf{x},\omega_i \rightarrow \omega) L_{in}(\mathbf{x},\omega_i) cos\theta_i d\omega_i$$

Rendering Equation

incident angle

$$L(\mathbf{x},\omega) = E(\mathbf{x},\omega) + \int_{\Omega^+} f(\mathbf{x},\omega_i \rightarrow \omega) L_{in}(\mathbf{x},\omega_i) cos\theta_i d\omega_i$$

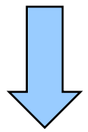
Integral over positive hemisphere:
all directions from which light can come



Rendering Equation and Monte Carlo Integration

- Light distribution in a scene [Kajiya86]
  - light potentially flows from and to all surfaces
  - …not accounting for volume effects
  - Fredholm integral equation 2. kind

$$L(\mathbf{x},\omega) = E(\mathbf{x},\omega) + \int_{\Omega^+} f(\mathbf{x},\omega_i \rightarrow \omega) L_{in}(\mathbf{x},\omega_i) cos\theta_i d\omega_i$$



$$L(\mathbf{x},\omega) = E(\mathbf{x},\omega) + \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x},\omega_i \rightarrow \omega) L_{in}(\mathbf{x},\omega_i) cos\theta_i$$

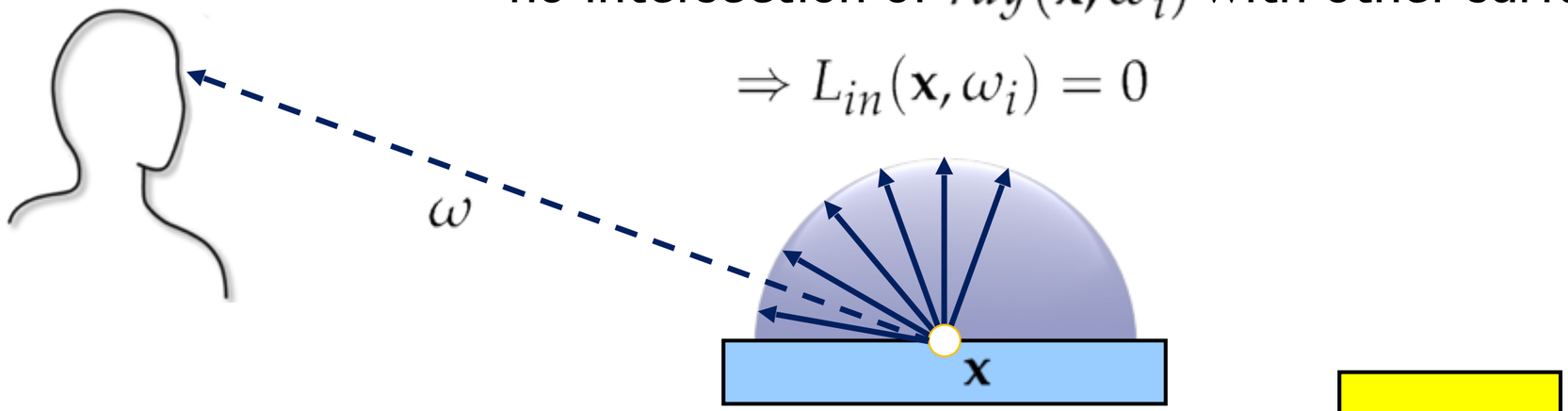
- Monte Carlo Integration
  - well suited for approximating high-dimensional integrals

Monte Carlo Integration

$$L(\mathbf{x},\omega) \;=\; E(\mathbf{x},\omega) \;+\; \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x},\omega_i \rightarrow \omega) L_{in}(\mathbf{x},\omega_i) cos\theta_i$$

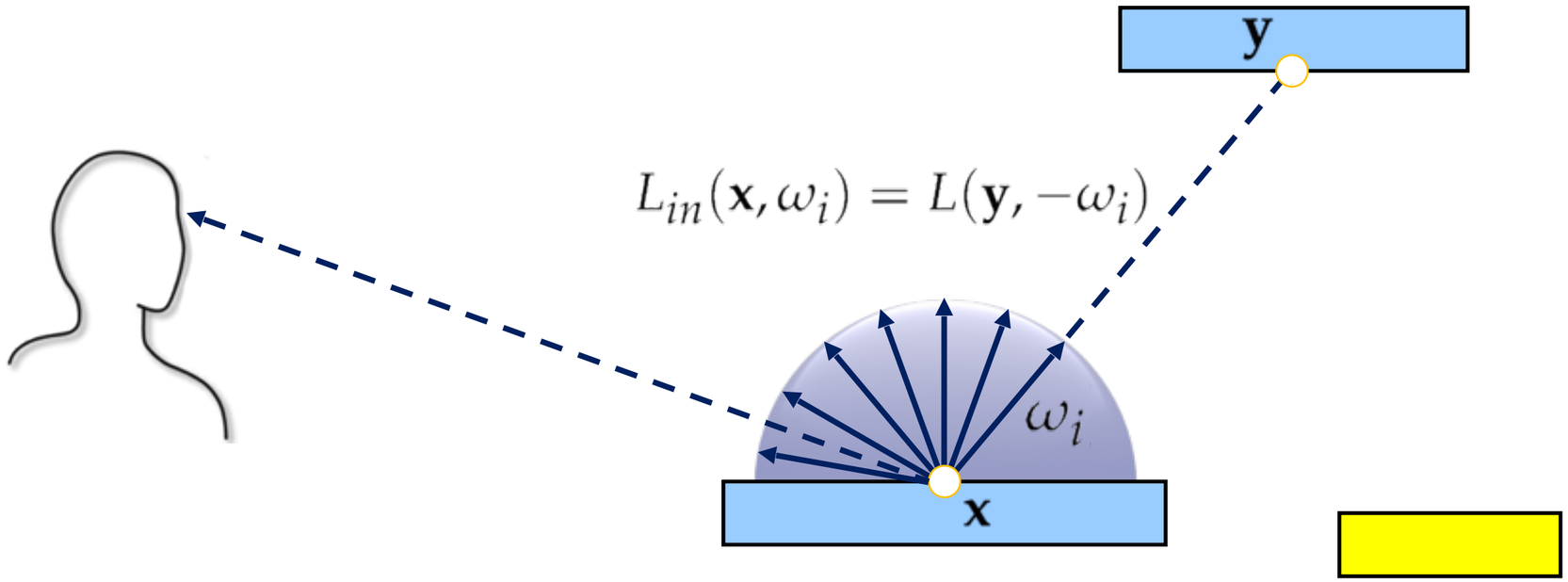
no intersection of $ray(\mathbf{x},\omega_i)$ with other surfaces

$$\Rightarrow L_{in}(\mathbf{x},\omega_i) = 0$$



Monte Carlo Integration

$$L(\mathbf{x},\omega) \;=\; E(\mathbf{x},\omega) \;+\; \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x},\omega_i \rightarrow \omega) L_{in}(\mathbf{x},\omega_i) cos\theta_i$$



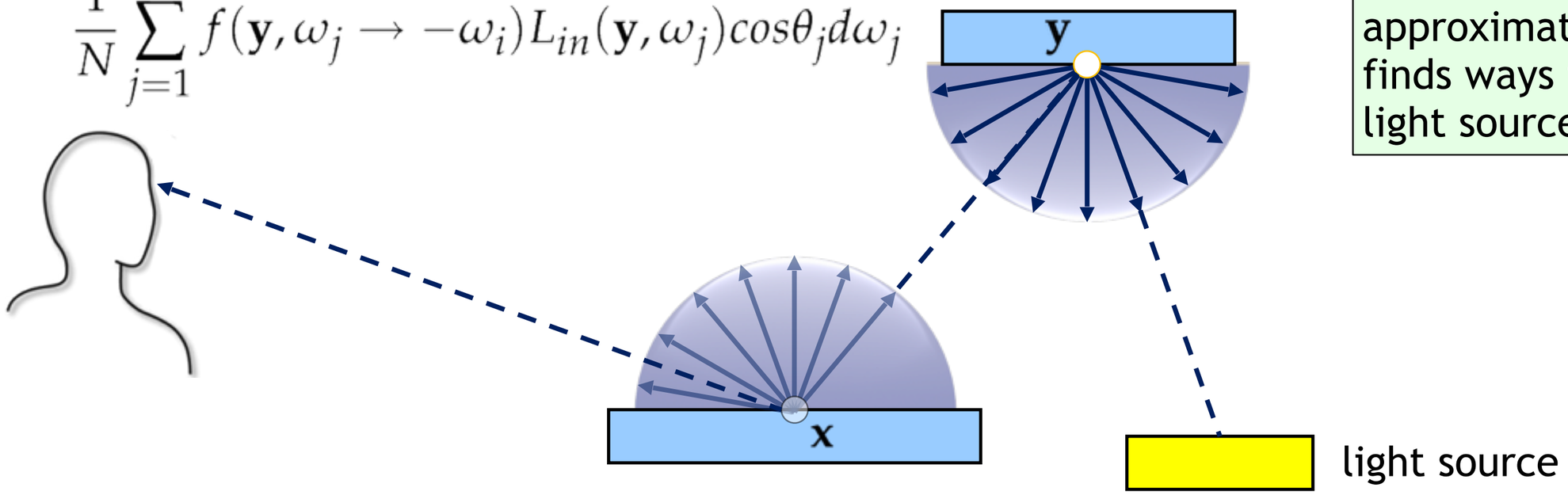
Monte Carlo Integration

- well suited for approximating high-dimensional integrals
- recursive approximation of integrals

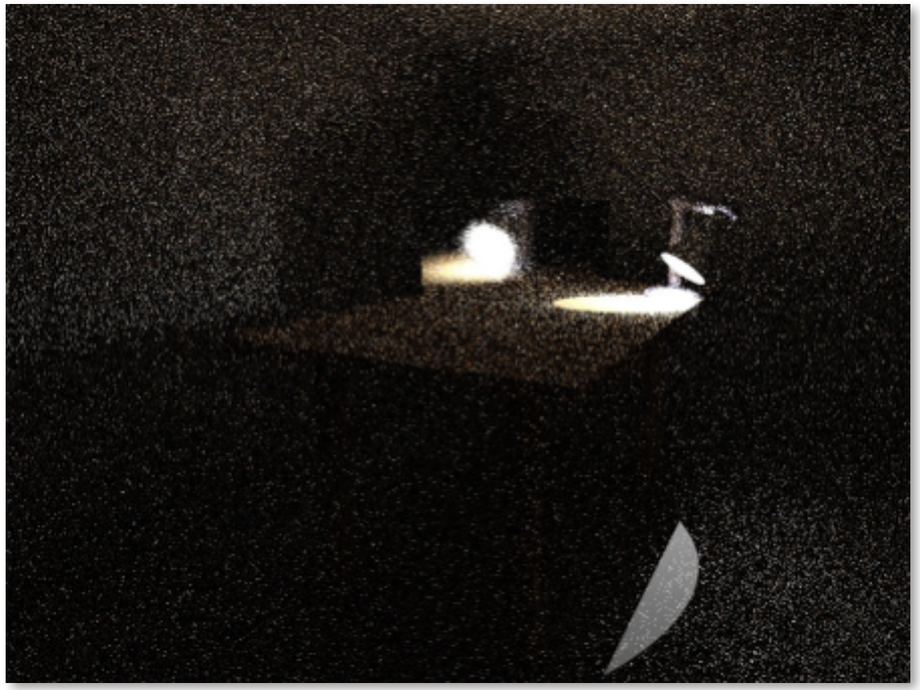
$$L(\mathbf{x},\omega) = E(\mathbf{x},\omega) + \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x},\omega_i \rightarrow \omega) L_{in}(\mathbf{x},\omega_i) cos\theta_i$$

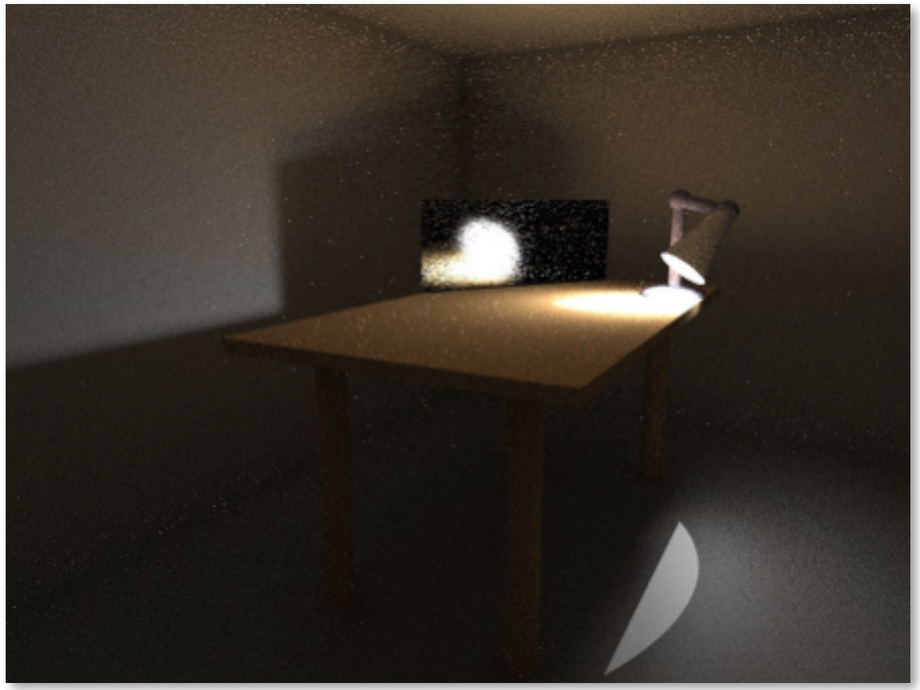
$$L(\mathbf{y},-\omega_i) = E(\mathbf{y},-\omega_i) + \frac{1}{N}\sum_{j=1}^{N} f(\mathbf{y},\omega_j \rightarrow -\omega_i) L_{in}(\mathbf{y},\omega_j) cos\theta_j d\omega_j$$

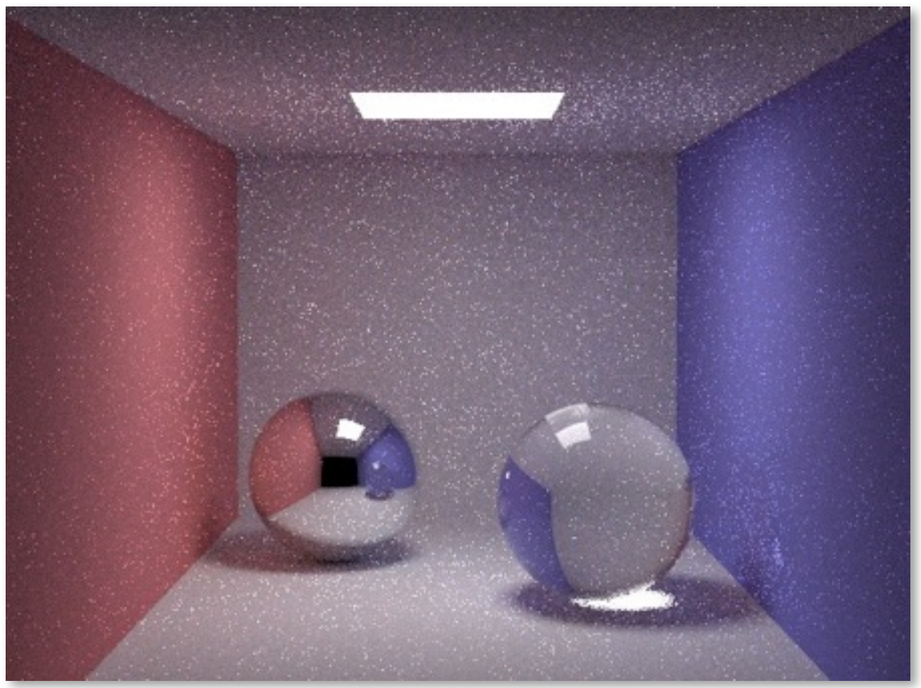
recursive approximation finds ways to the light sources

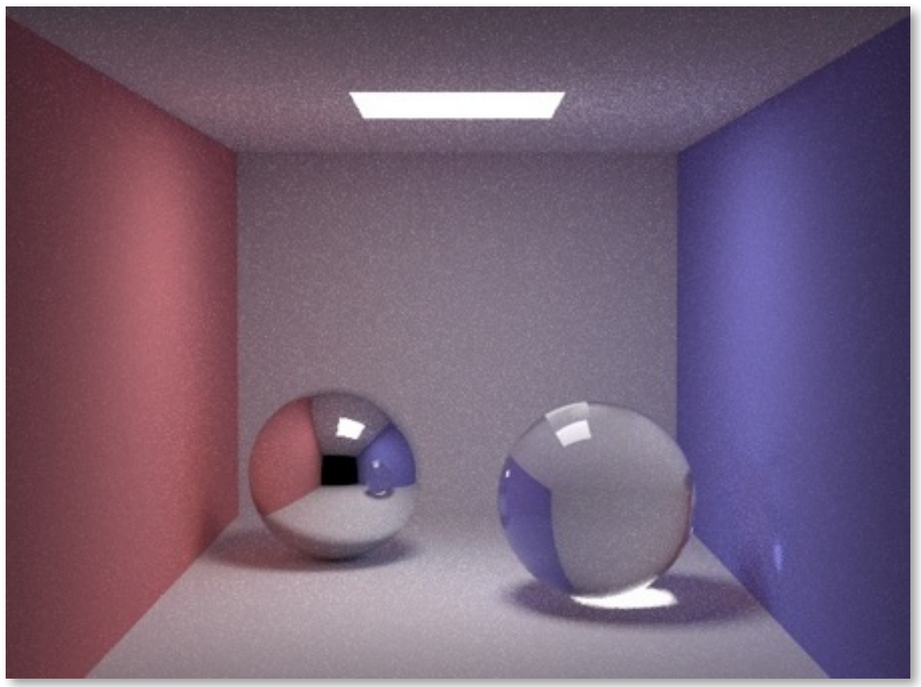


Distributed Ray Tracing, Path Tracing

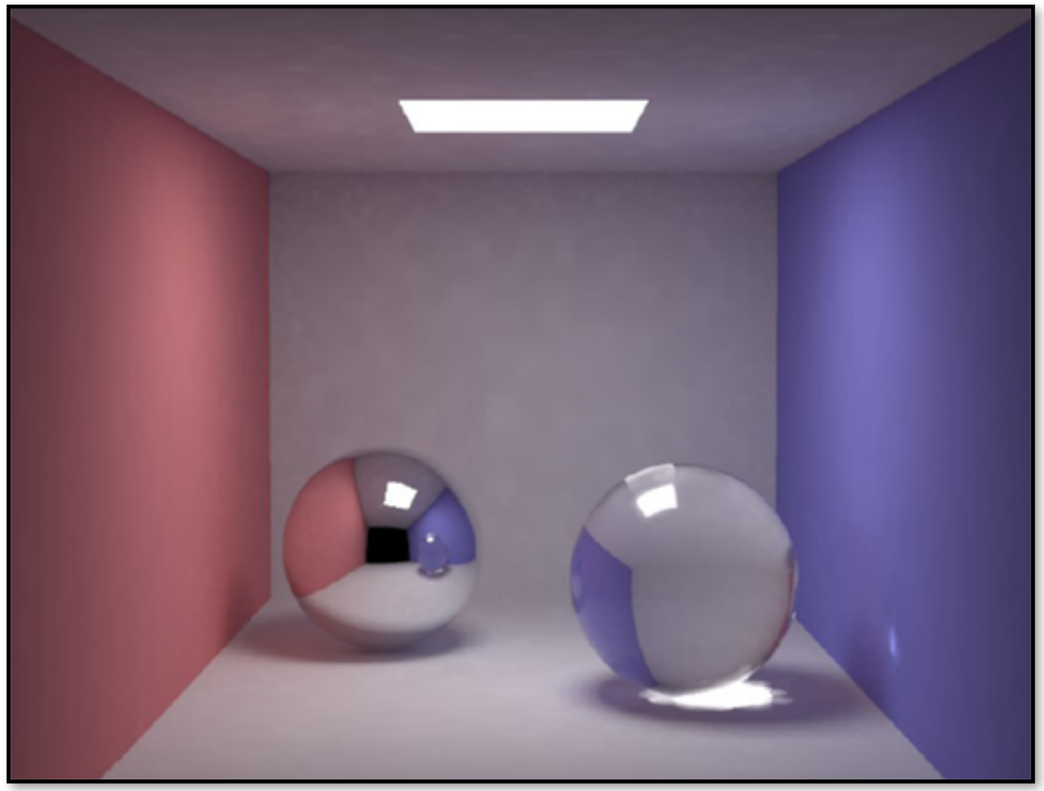


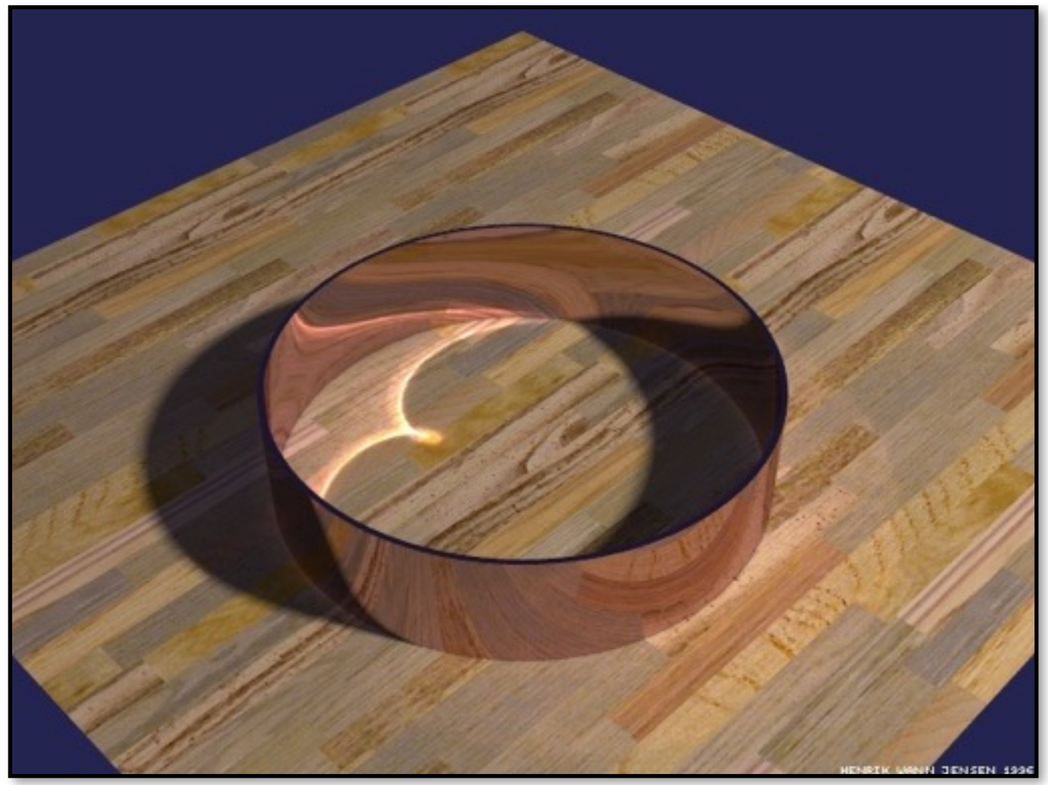






Indirect Illumination and Caustics





Chapter 7 – Shading and Rendering

- Local Illumination Models: Shading
- Global Illumination: Ray Tracing
- Global Illumination: Radiosity
- Monte-Carlo Methods
- Non-Photorealistic Rendering

Non-Photorealistic Rendering (NPR)

Telltale's The Wolf Among Us

- Create graphics that look like drawings or paintings

- Popular example: Stroke-based NPR methods
  - Instead of grey shades, determine a stroke density and pattern
  - Imitates pencil drawings or etchings
  - e.g., contour lines or hatching

http://www.cs.ucdavis.edu/~ma/SIGGRAPH02/course23/

- Image-based methods
  - Using image manipulation on rendered images
  - Can in principle often be done in Photoshop

- Active field of research
  - http://www.cs.ucdavis.edu/~ma/SIGGRAPH02/course23/
  - http://graphics.uni-konstanz.de/forschung/npr/watercolor/
  - …many others

http://www.katrinlang.de/npr/