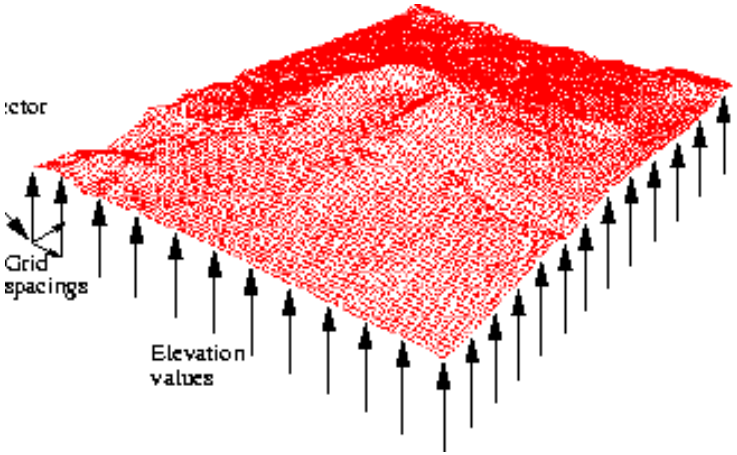
Ray-Tracing Height Fields

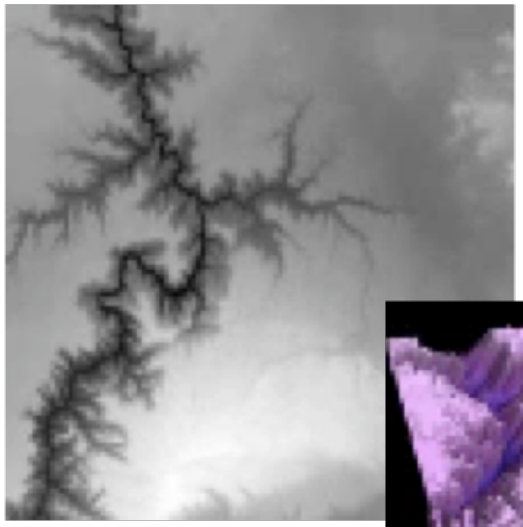
[Henning & Stephenson, 2004]

- Height Field = all kinds of surfaces that can be described by such a function

$$z = f(x, y)$$

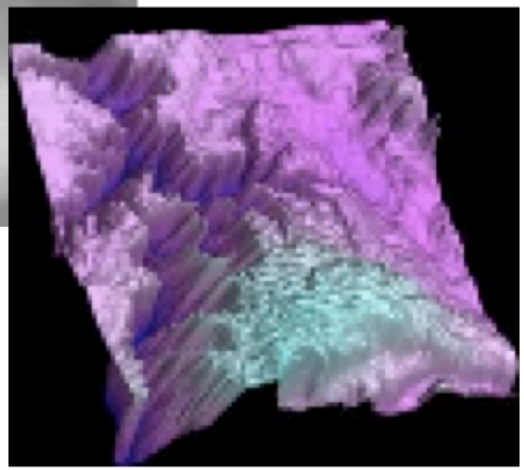
  - Examples: terrain, measurements sampled on a plane, 2D scalar field



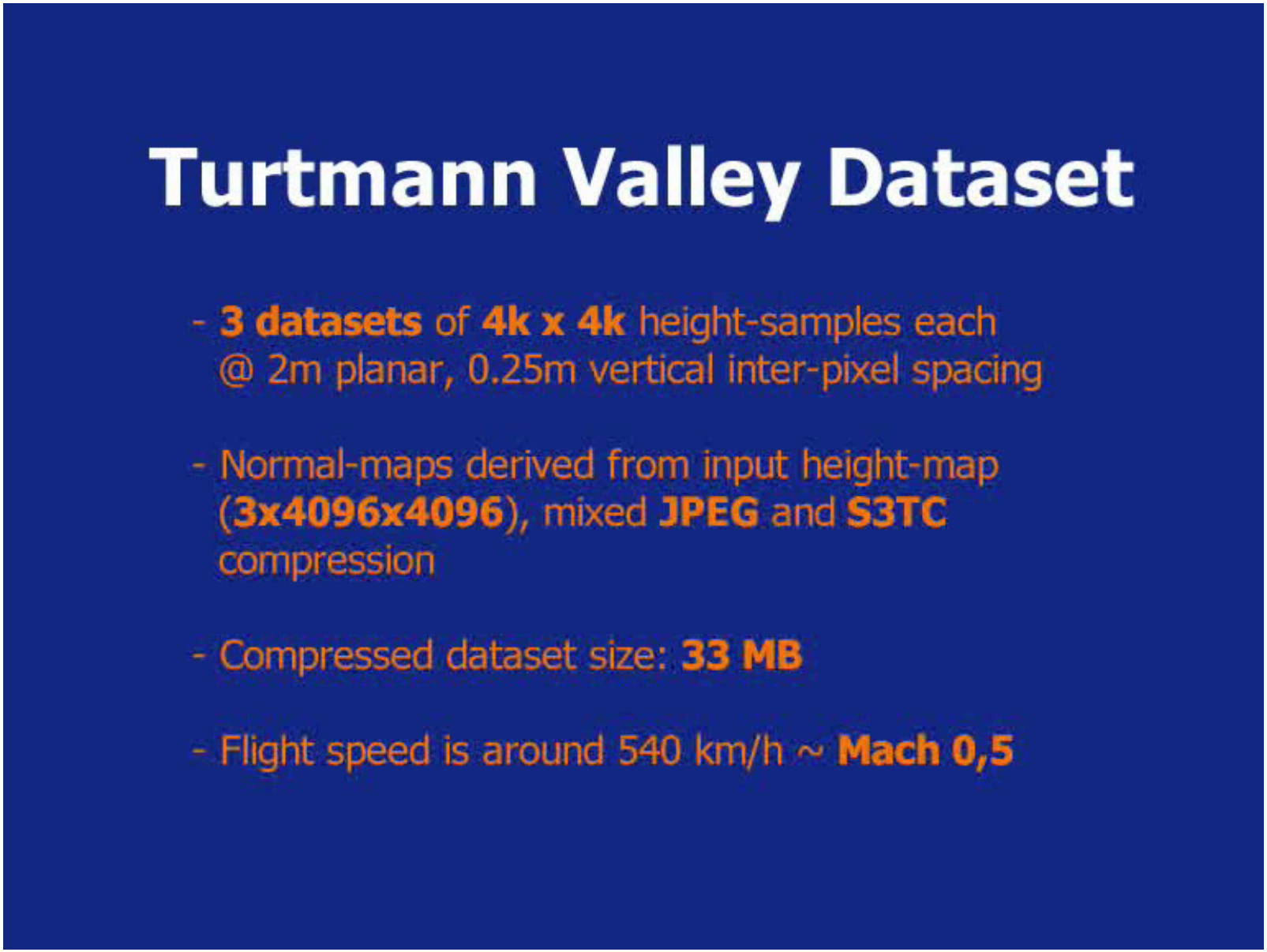


Height field
(= Bitmap)

Rendered



Example for Terrains

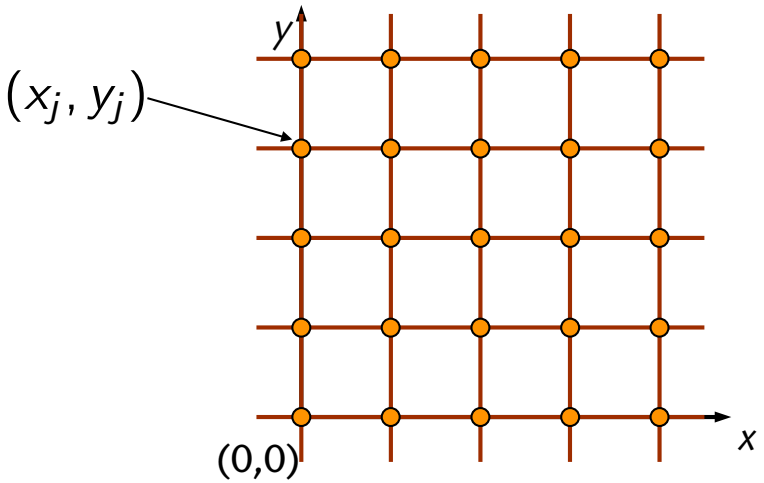


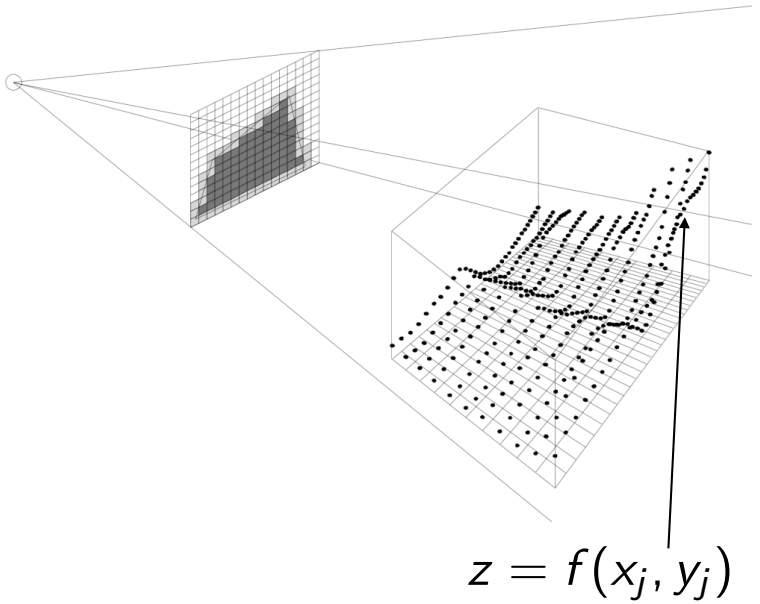
Bonn University

Vallis Marineris, Mars; presented by Phil Christensen, Arizona State University (http://mars.jpl.nasa.gov)

The Situation

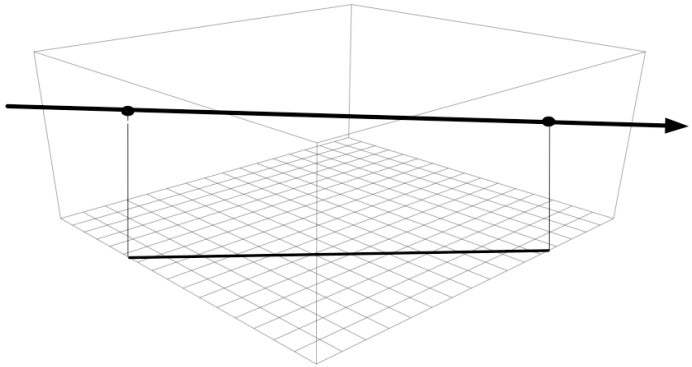
- Given:
  - Ray
  - Array [0...n]x[0...n] with heights
- The naïve method to ray-trace a height field:
  - Convert to $2n^2$ triangles, test ray against each triangle
  - Problems: slow, needs lots of memory
- Goal: direct ray-tracing of a height field represented as 2D array



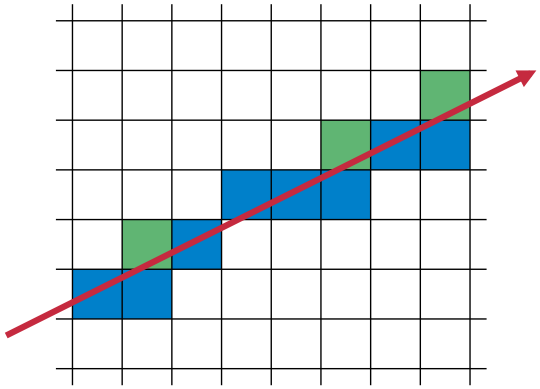


The Method

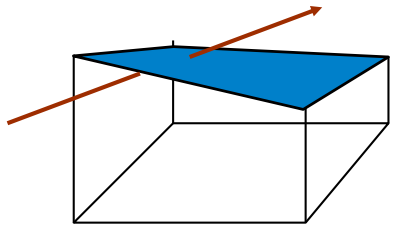
1. Reduce the dimension:
   - Project ray into xy plane



2. Visit all cells that are hit by the ray, starting with the nearest one
   - Notice similarity to scan conversion!
   - Use one of the DDA algorithms from CG1 😀

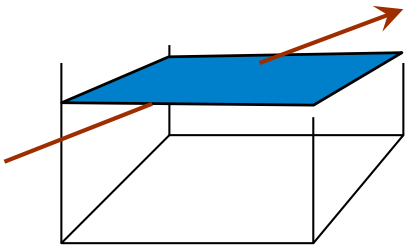


3. Test ray against the surface patch spanned by the 4 corners of the cell

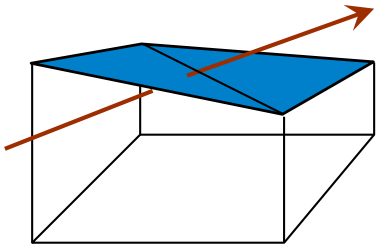


Intersection of Ray with Surface Patch of Cell

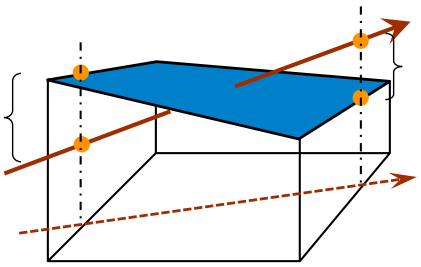
- Naïve methods:
  - "Nearest neighbor":
    - Compute average height of the 4 corner height values
    - Intersect ray with horizontal square of that average height
    - Problem: very imprecise



  - Tessellate by 2 triangles:
    - Construct 2 triangles from the 4 corner points
    - Problem: tessellation is not unique, diagonal edge could produce severe artefact

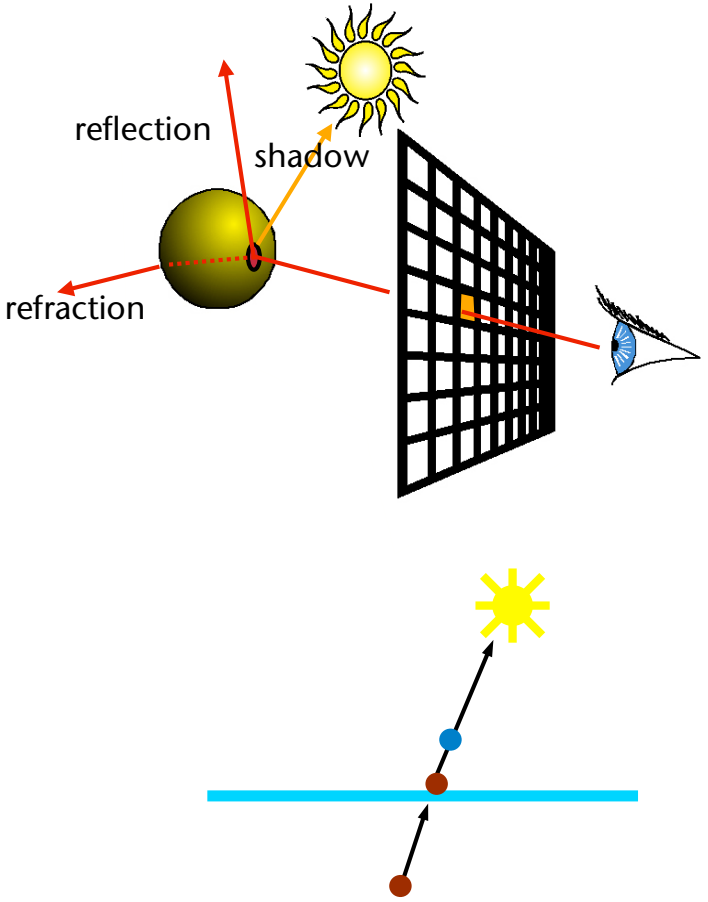


- Better: *bilinear interpolation*
  - Determine height of the surface on the edge of the cell above the point where the ray enters/leaves the cell ⟶ linear interpolation of corner heights
  - Compare signs
  - Insert ray equation into bilinear equation of surface ⟶ quadratic equation for line parameter *t*
  - (The surface is called a *parabolic hyperboloid*)

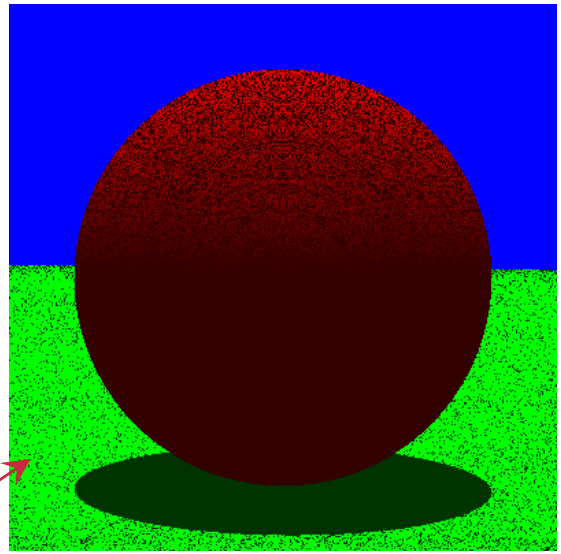


The evil ε

- What happens when the origin of a ray is "exactly" on the surface of an object?
- Remember: floating-point calculations are always imprecise!
  - Consequence: in subsequent ray-scene intersection tests, the ray might appear to be inside the original object!
  - Further consequence: we get wrong intersection points!
- "Solution": move the origin of the ray by a small ε along the direction of the (new) ray

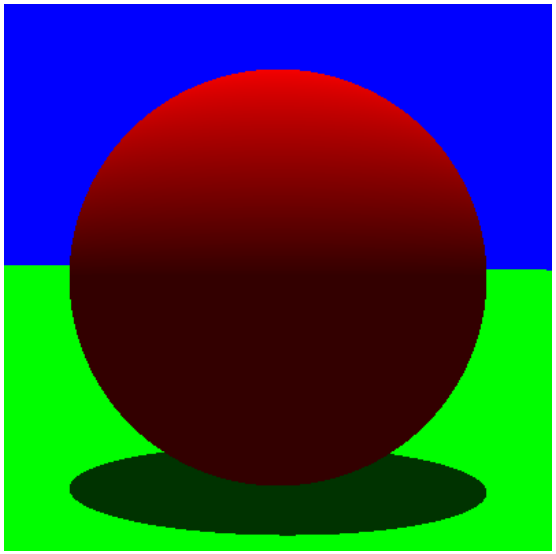


More Glitch Pictures

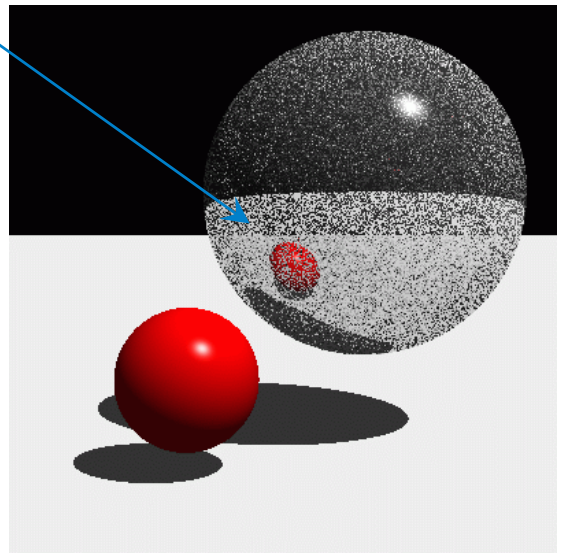


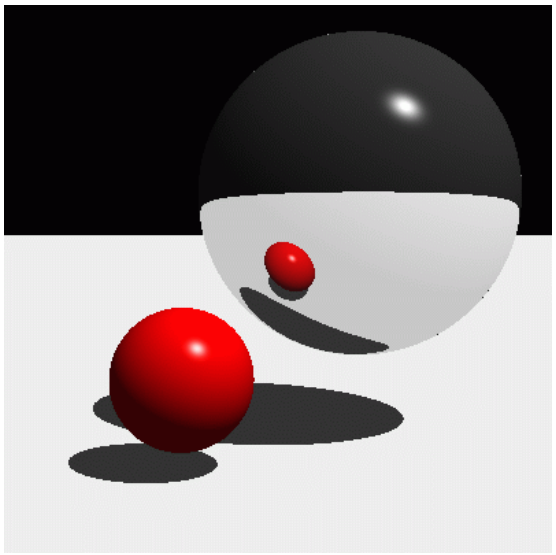
"Speckles"

Without epsilon

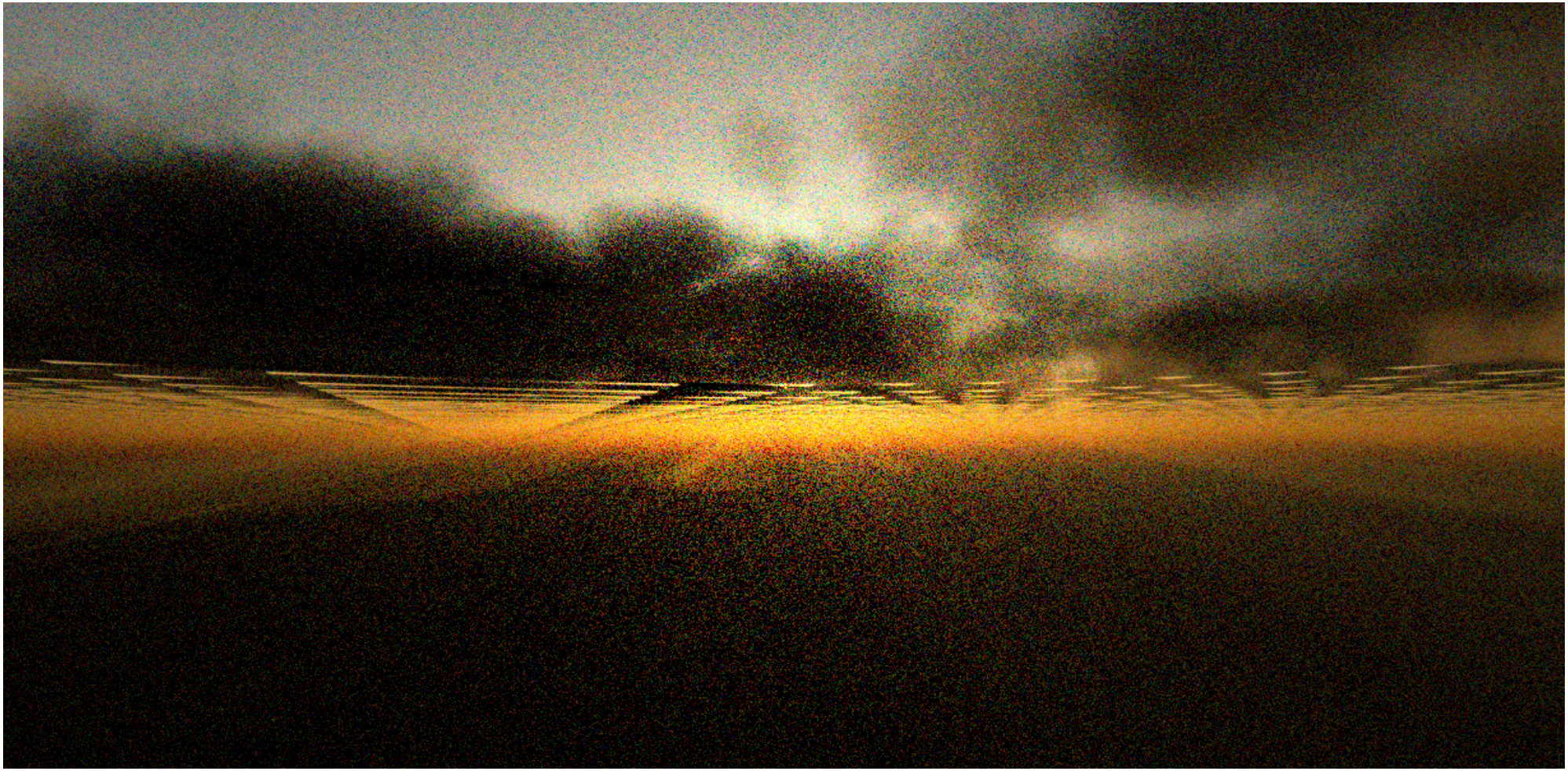


With epsilon





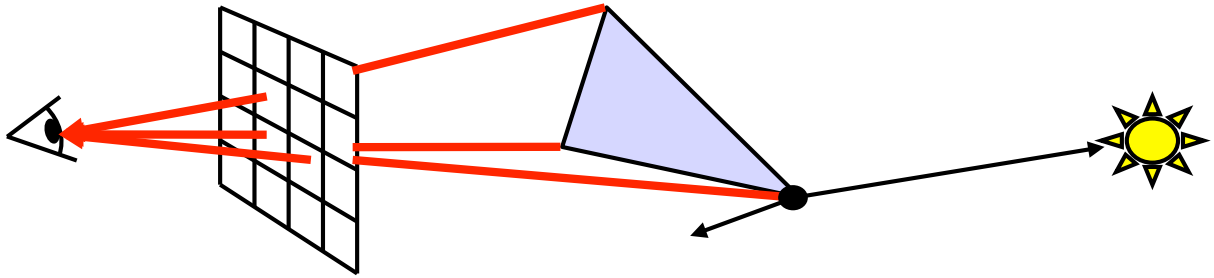
Numerically unstable cloud layer intersection



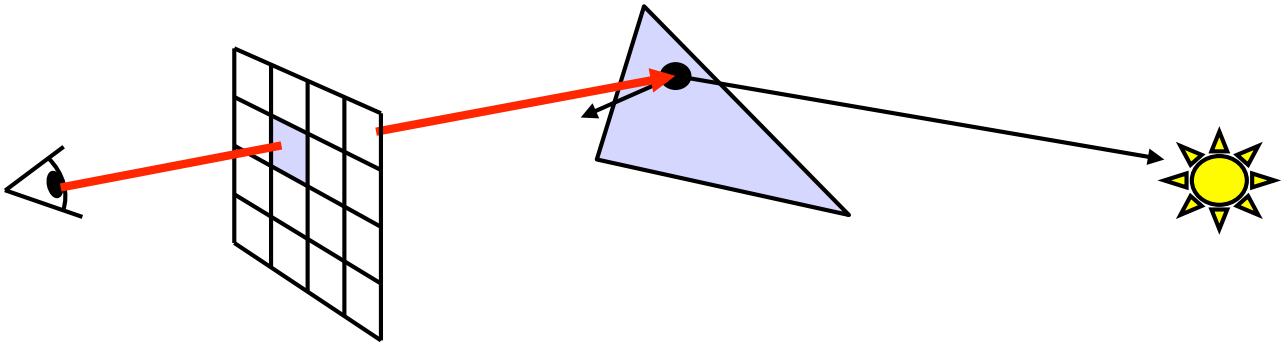
Source: necro (http://igad2.nhtv.nl/ompf2)

Comparison of Scan Conversion and Ray-Tracing

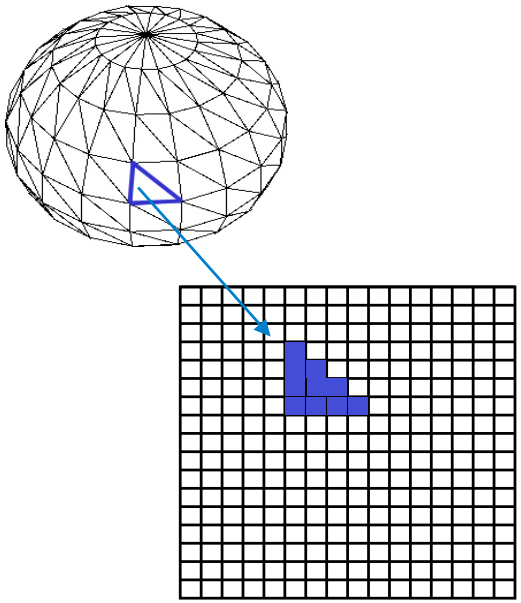
- Scan conversion: start with triangles, project each vertex = send ray through each vertex



- Raytracing: start with pixels, send ray through each pixel

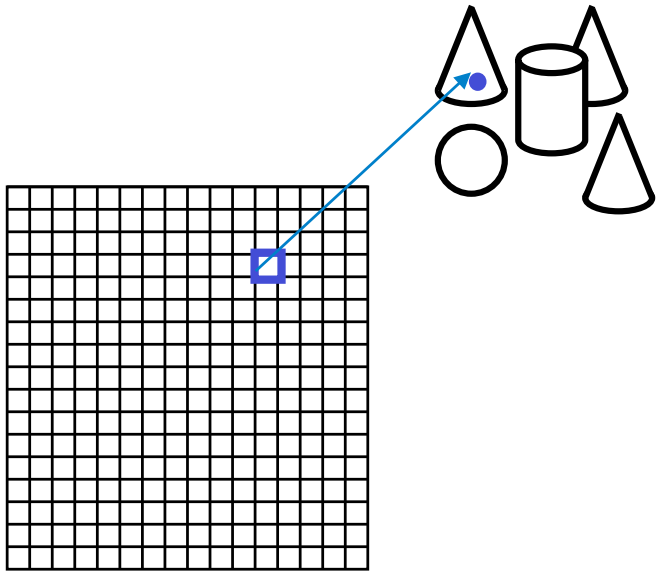


- For rendering a complete scene using scan conversion ...



... scan-convert each triangle

- For rendering a complete scene using raytracing ...



... trace a ray through each pixel

Advantages & Disadvantages

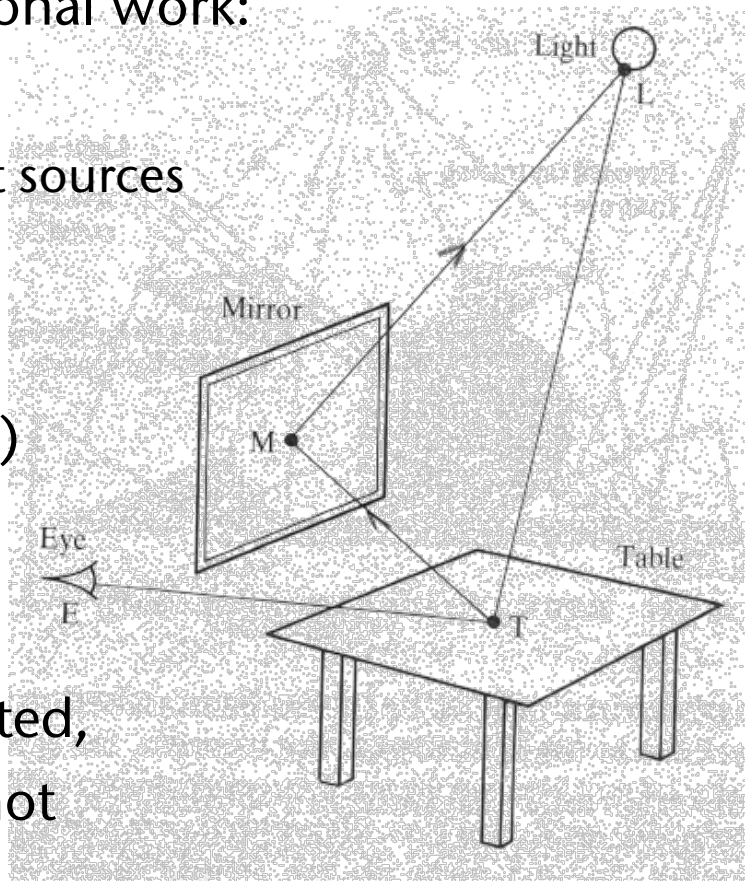
- Scan conversion:
  - Fast (for a number of reasons)
    - Well-suited for real-time graphics
  - Supported by all graphics hardware
  - Only heuristic solutions for shadows and transparent objects
  - No interreflections
- Raytracing:
  - Offers general and simple (in principle) solution for global effects, such as shadows, interreflection, transparent objects, etc.
  - Much slower (unless you cast only primary rays)
  - Not directly supported by most graphics hardware
  - Difficult to achieve real-time rendering

Other Advantages of Ray-Tracing

- Shines with scenes that contain lots of glossy/shiny surfaces and transparent objects
- Fairly easy to incorporate other object representations (e.g., CSG, smoke, fluids, ...)
  - Only prerequisite: must be possible to compute the itersection between ray and object, and to compute the normal at the point of intersection
- No separate clipping step necessary

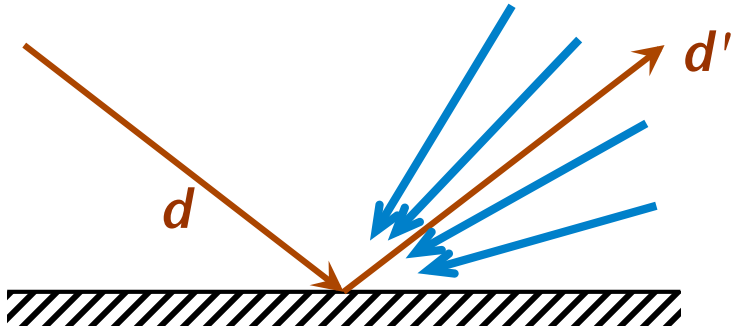
Disadvantages of (Simple) Ray-Tracing

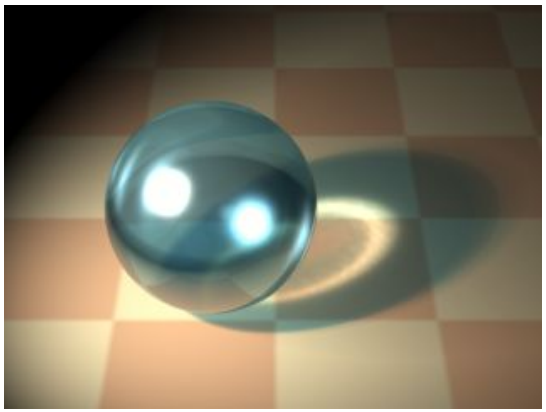
- Needs a huge amount of computational work:
  - Just for primary rays: $O(p \cdot (n+l))$,
    p = # pixels, n = # polygons, l = # light sources
  - Number of rays grows exponentially with number of recursions!
- No indirect lighting (e.g., by mirrors)
- No soft shadows
- With each camera movement, the complete ray tree must be recomputed, although an object's shading does not depend on the camera's position
- For all of these disadvantages, a number of remedies have been proposed ...

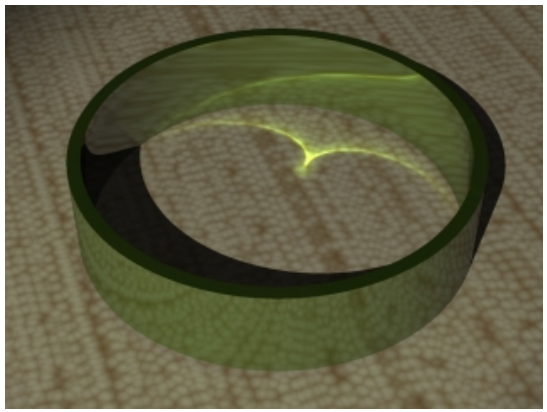


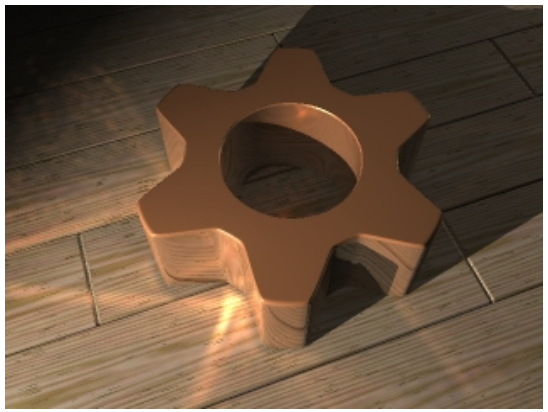
Example for the Problem with (Missing) Indirect Lighting: Caustics

- Caustic = reflected / transmitted light is concentrated in a point or, possibly curved, line on the surface of another object
- Problem:
  - Ray-tracing follows light paths "in reverse"
  - Simple ray-tracing follows only one path







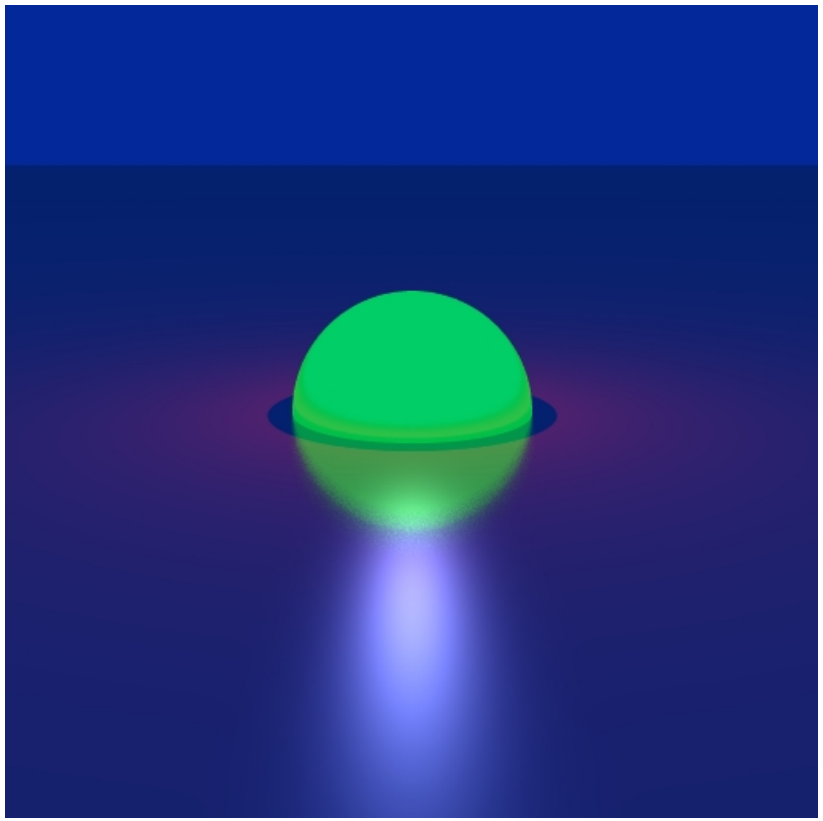


Other Problem: Aliasing

- One ray per pixel → causes typical aliasing artefacts:
  - "Jaggies"
  - Moiré effects

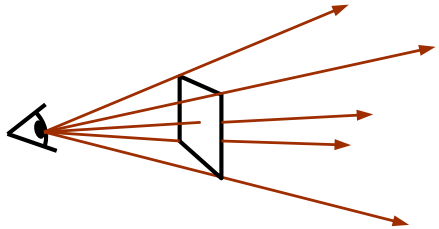
Distribution Ray Tracing

- Simple modification of ray-tracing to achieve
  - Anti-aliasing
  - Soft shadows
  - Depth-of-field
  - Shiny/glossy surfaces
  - Motion blur
- Was proposed under a different name:
  - "Distributed Ray Tracing"
  - Don't use that name ("distributed" = verteilt)

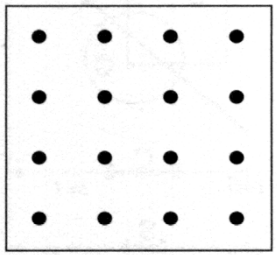


Anti-Aliasing with Ray-Tracing

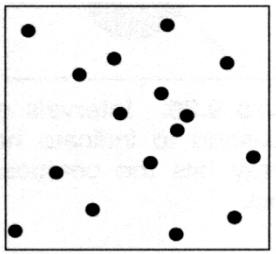
- Shoot many rays per pixel, instead of just one, and average retrieved colors
- Methods for constructing the rays:



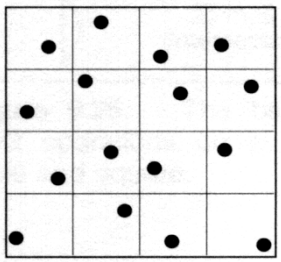
1. Regular sampling
   - Perhaps problems with Moiré patterns



2. Random sampling
   - Might result in noisy images

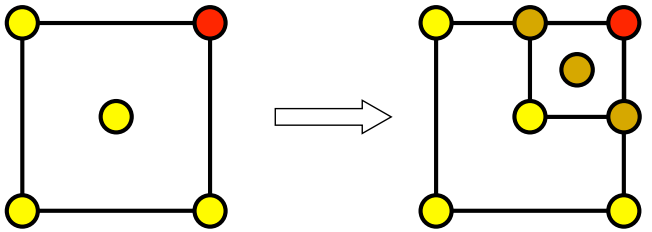


3. Stratification: combination of regular and random sampling, e.g., by placing a grid over the pixel, and picking one random position per cell

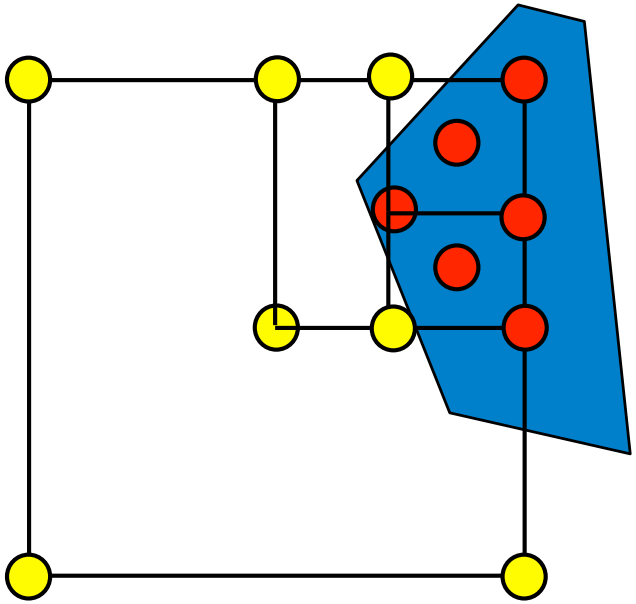


Adaptive Supersampling

- Idea: shoot more rays only in case of large differences in color



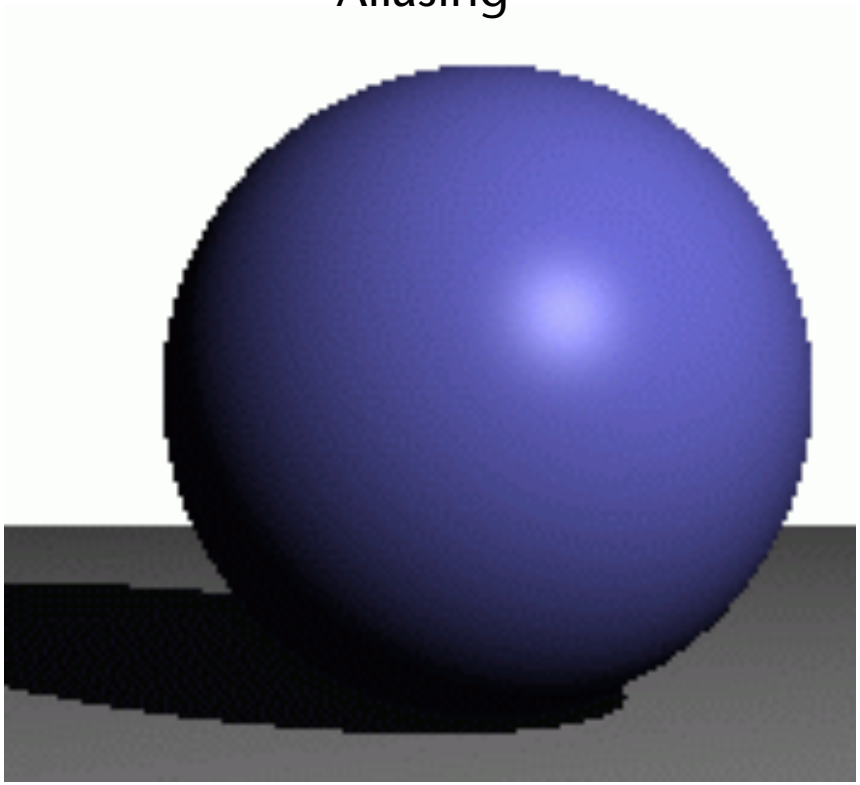
- Example:



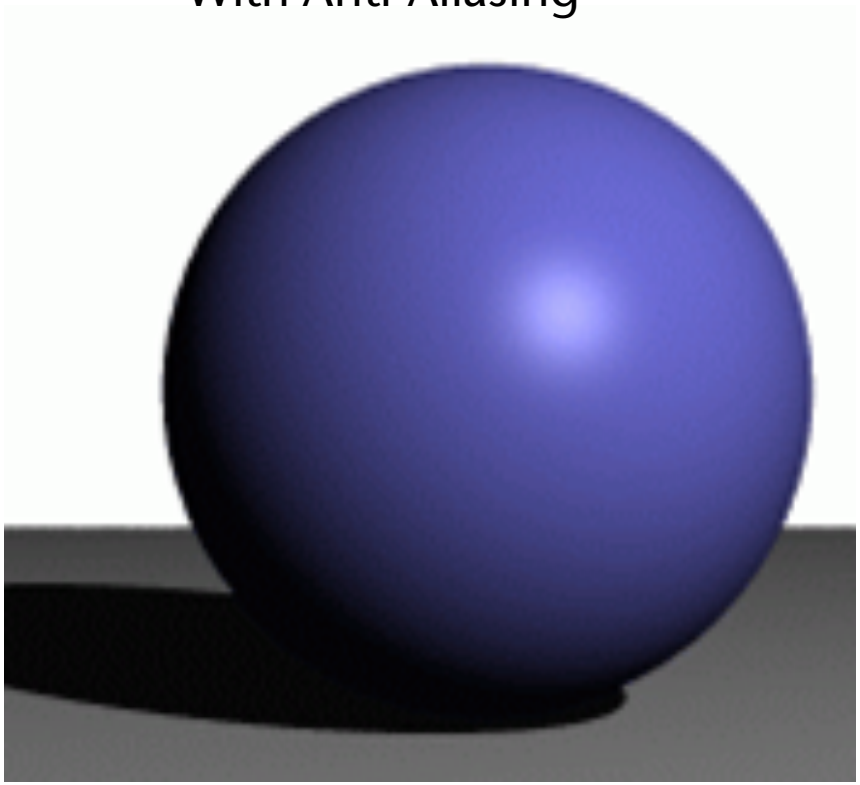
- Resulting color = weighted average of all samples, weighted by the area each sample covers

Result

Aliasing

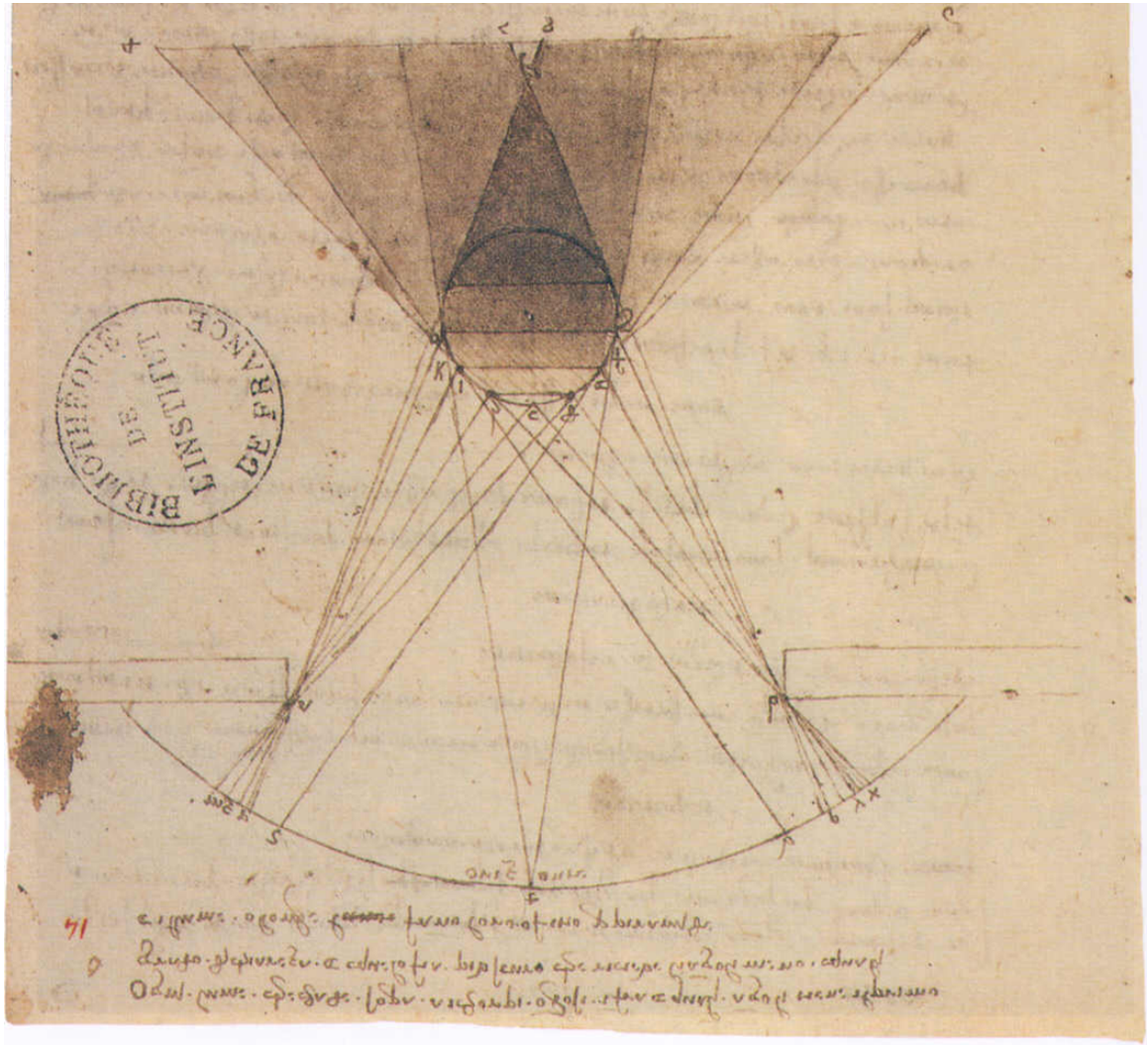


With Anti-Aliasing



Soft Shadows, Penumbra

- Behind a lighted object, there are 3 regions:
  - Completely in shadow (*umbra*)
  - Partially in shadow (*penumbra*)
  - Completely lighted



XVI. Léonard de Vinci (1452-1519). Lumière d'une fenêtre sur une sphère ombreuse avec (en partant du haut) ombre intermédiaire, primitive, dérivée et (sur la surface, en bas) portée. Plume et lavis sur pointe de métal sur papier, 24 x 38 cm. Paris, Bibliothèque de l'Institut de France (ms. 2185 ; B.N. 2038, f° 14 r°).

In Reality ...

http://3media.initialized.org/photos/2000-10-18/index_gall.htm

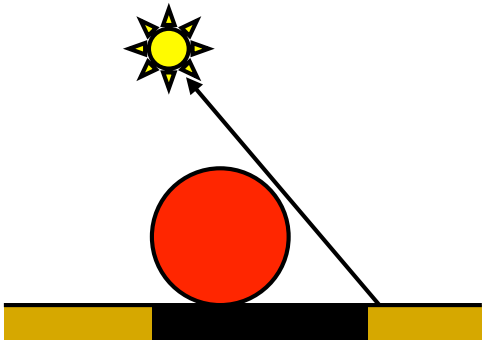
http://www.davidfay.com/index.php

http://www.pa.uky.edu/~sciworks/light/preview/bulb2.htm

... and in Ray-Tracing

- So far, exactly 1 shadow feeler per light:
  - We add a light source's contribution or not, depending on

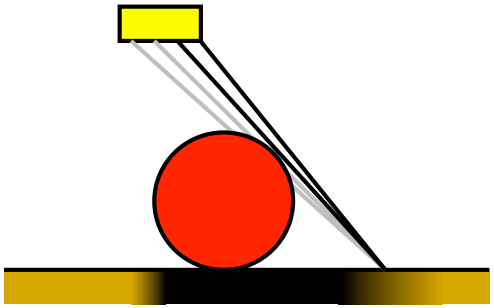
$$s_i = \begin{cases} 1\,, & \text{light source visible} \\ 0\,, & \text{invisible} \end{cases}$$



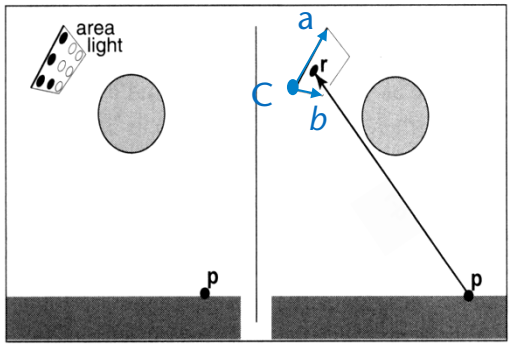
- Improvement: send many shadow feelers

$\longrightarrow$

$$s_i = \frac{\#\ \text{passing shadow rays}}{\#\ \text{shadow rays sent}}$$

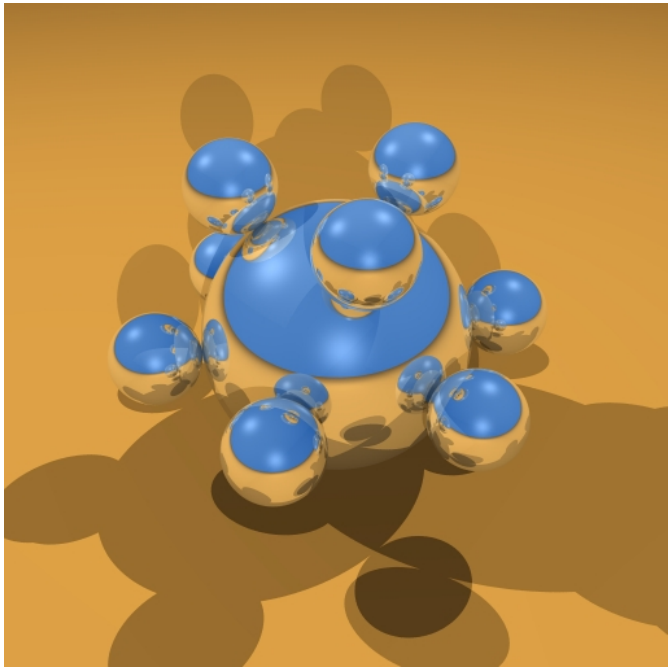


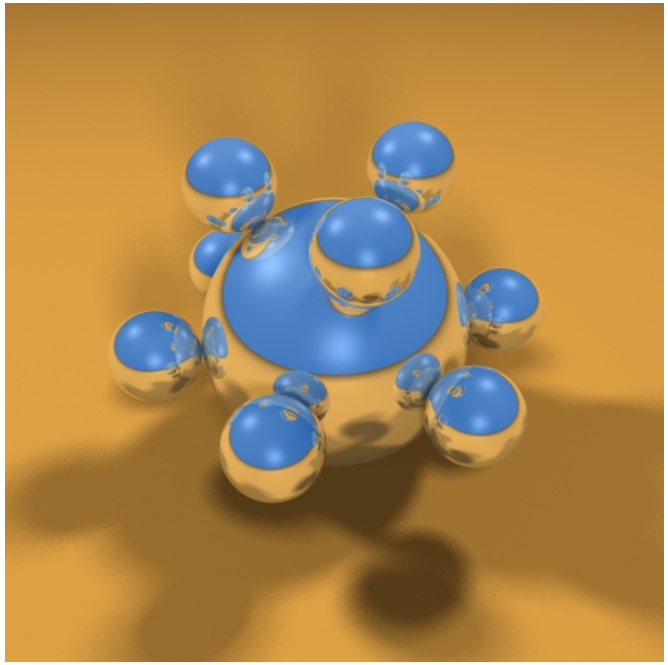
- Three ways of sampling a lightsource:
  1. Regular sampling of the area of the lightsource
  2. Random sampling
  3. Stratified sampling (just like with pixels/AA)

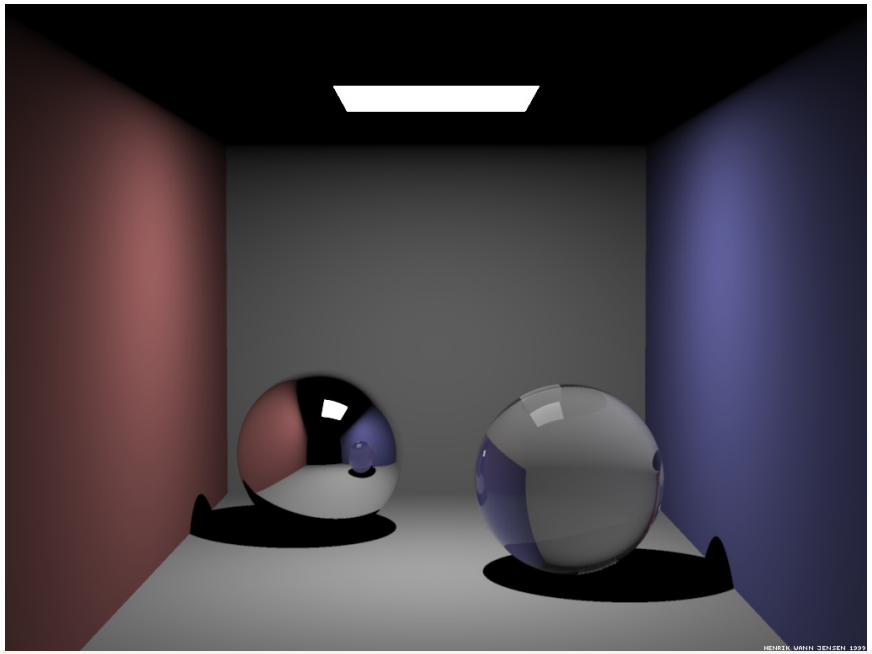


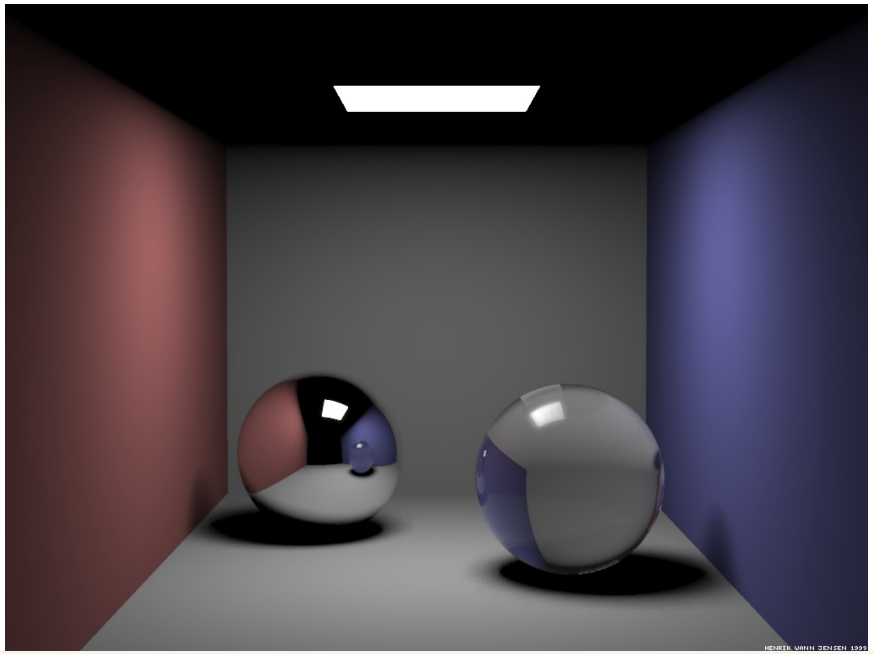
- Modification of the (local) lighting model:

$$L_{\text{Phong}} = \sum_{j=1}^{n} s_j \cdot f(\Phi_j, \Theta_j) \cdot I_j$$









- Geometric construction of the different shadow regions:

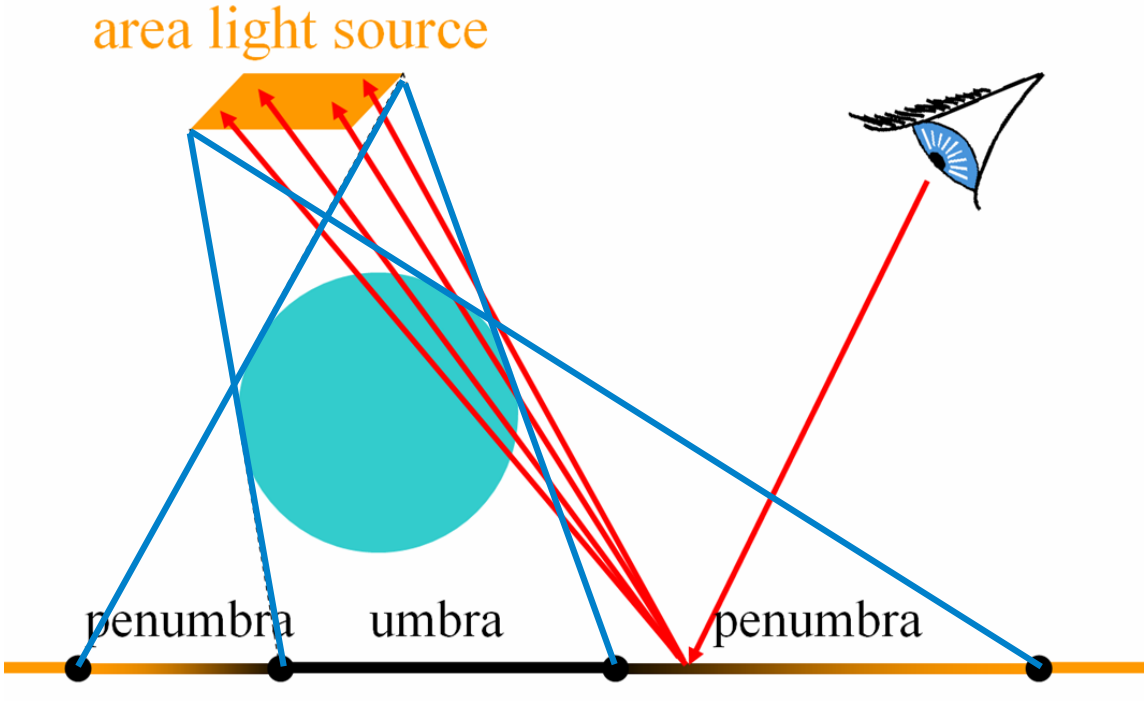
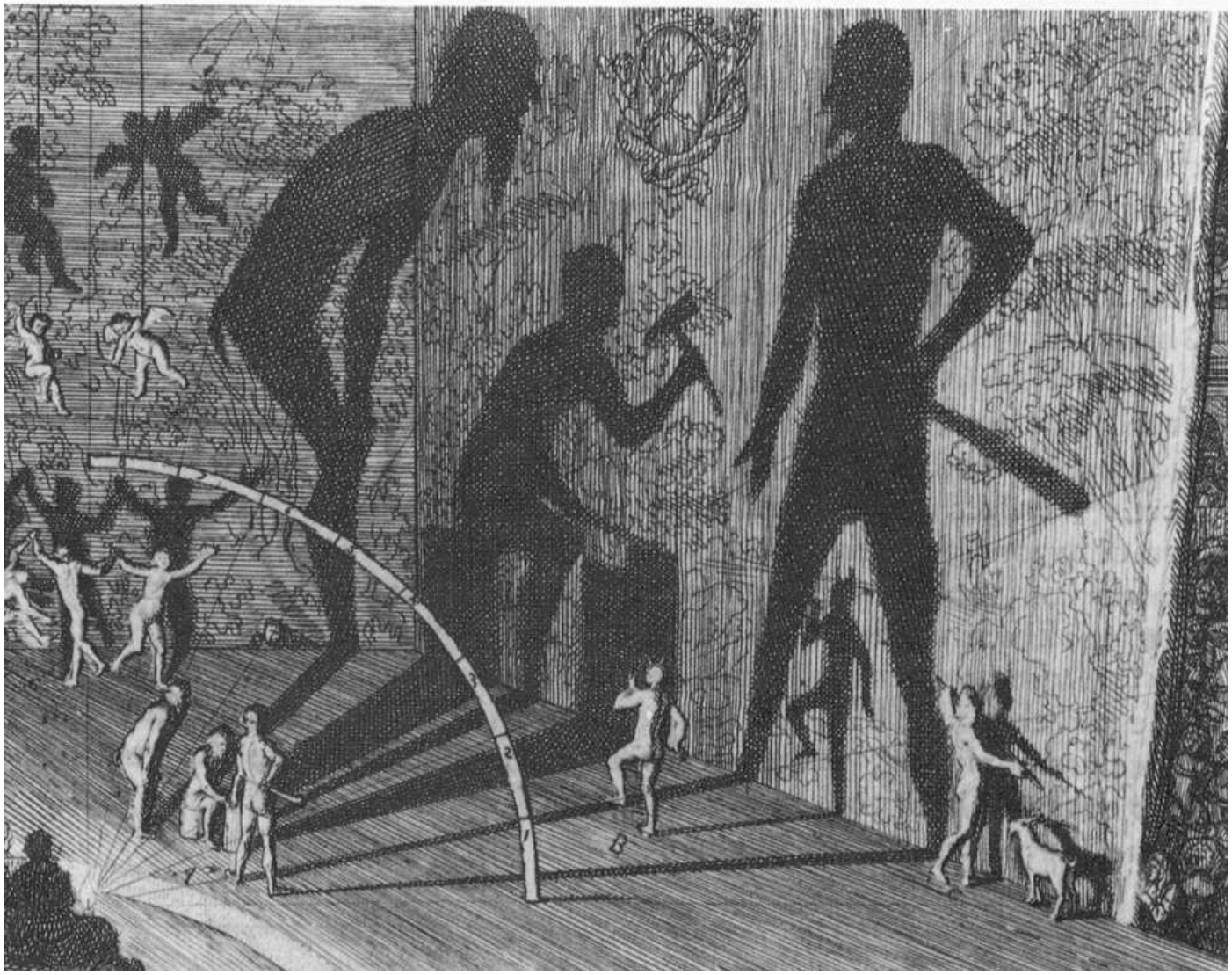
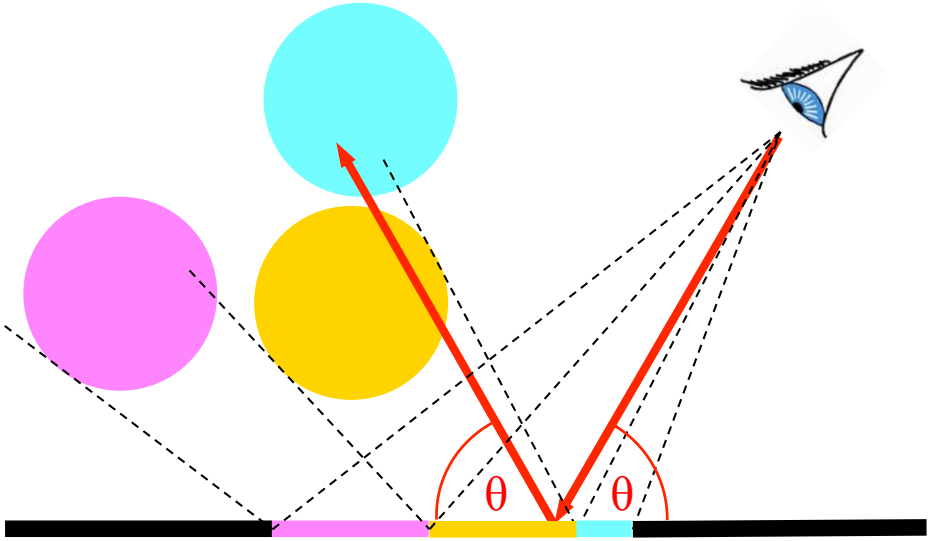


Plate 50 Samuel van Hoogstraten, *Shadow Theatre*. From *Inleyding tot de hooghe schoole der schilderkonst* 1678. (Einführung in die hohe Schule der Gemäldemalerei)



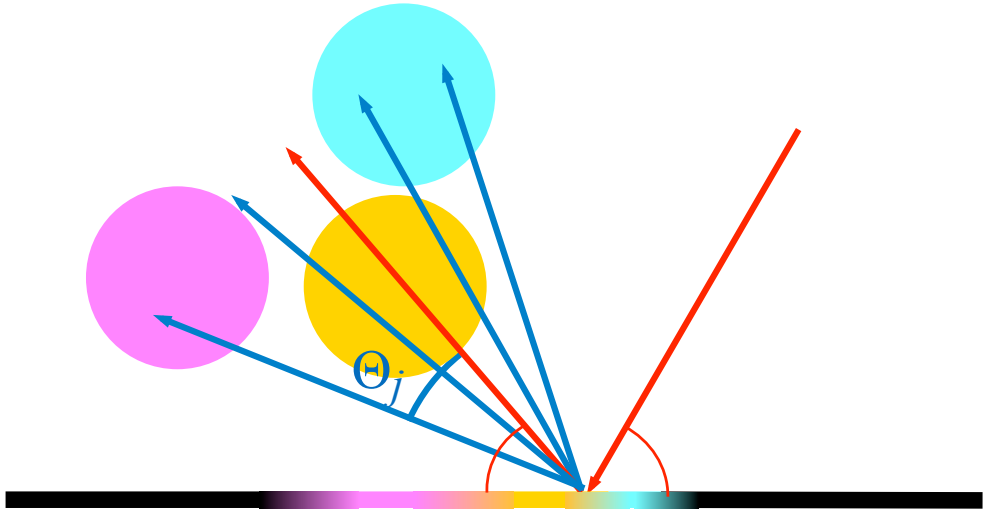
Better Glossy/Specular Reflection

- So far, exactly 1 reflected ray:
  - Problem, if the surface should be matte-glossy ...

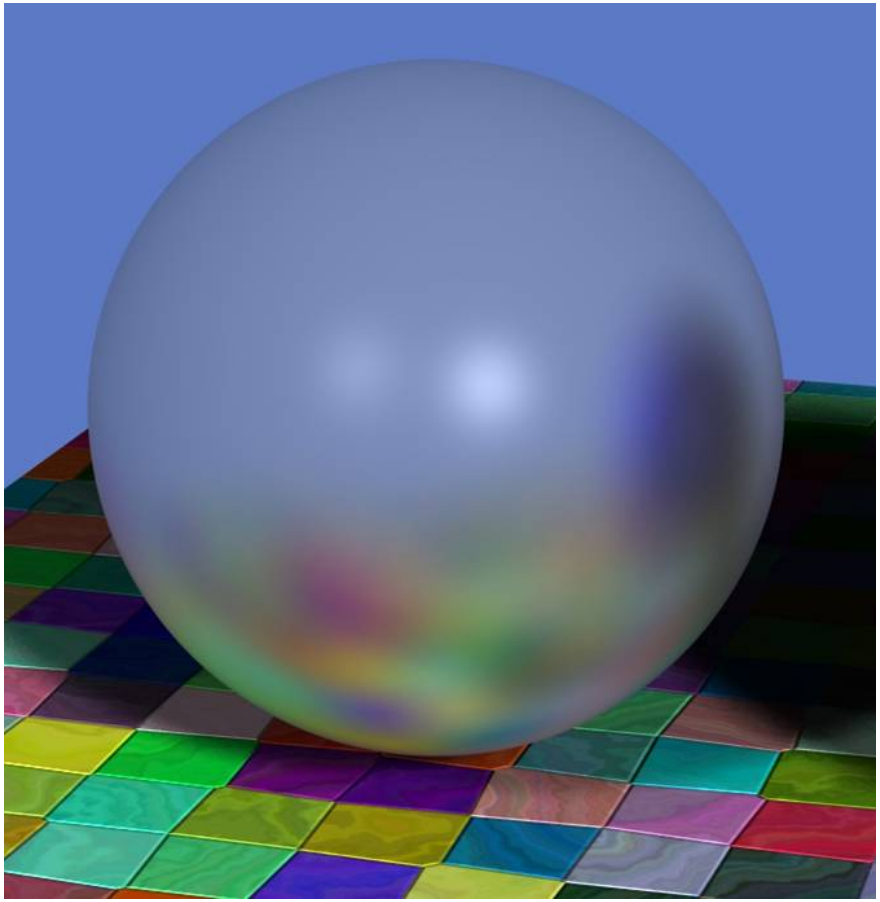


- Solution (somewhat brute-force):
  - Shoot many secondary, "reflected" rays
  - Accumulate weighted by power-cosine law (Phong)

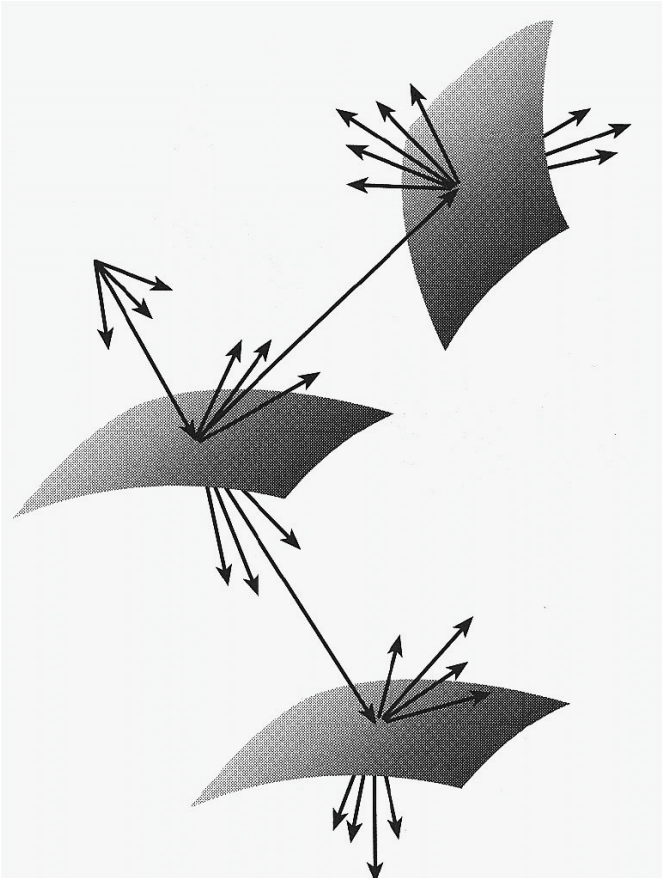
$$\cos^p \Theta_j$$



- Example:

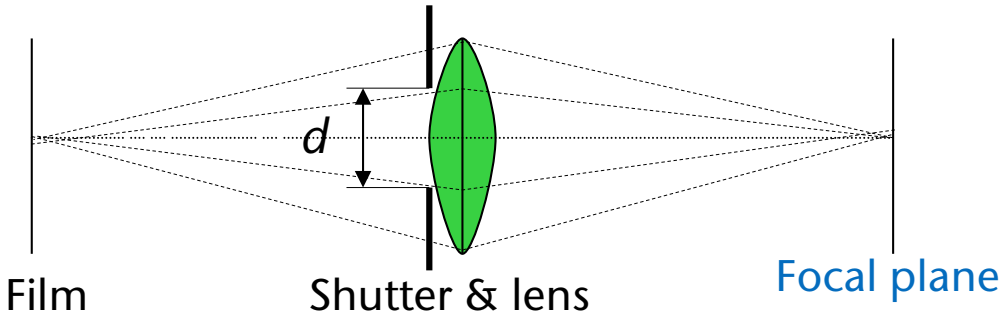


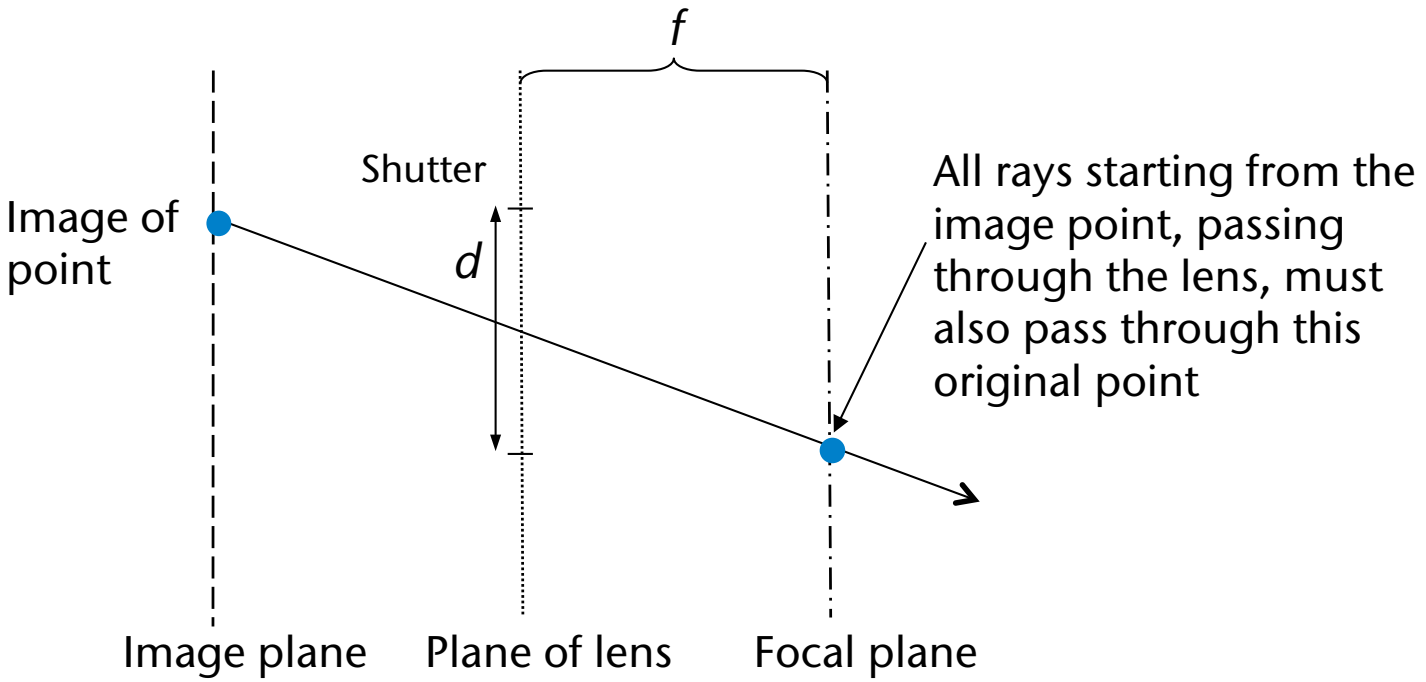
- The ray tree:



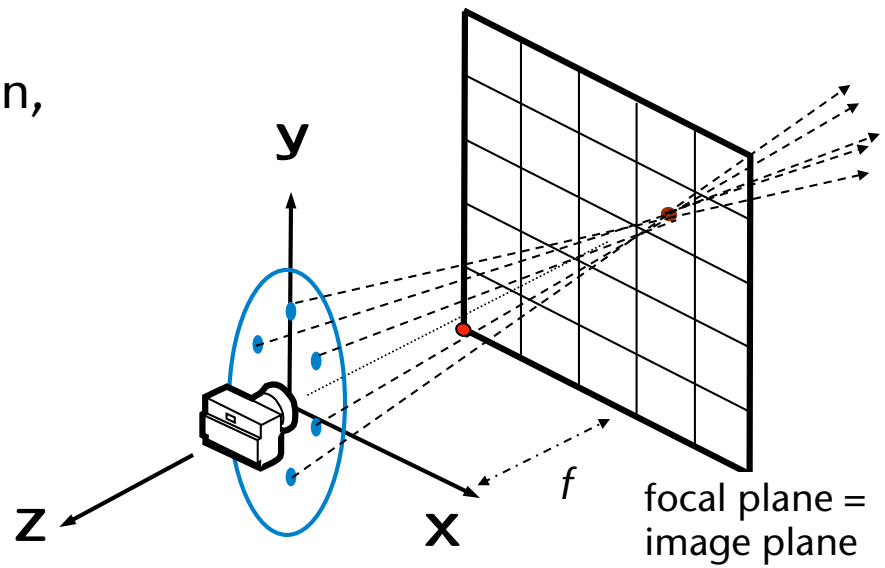
Depth-of-Field (Tiefen(un-)schärfe)

- So far: ideal pin-hole camera model
- For depth-of-field, we need to model real cameras



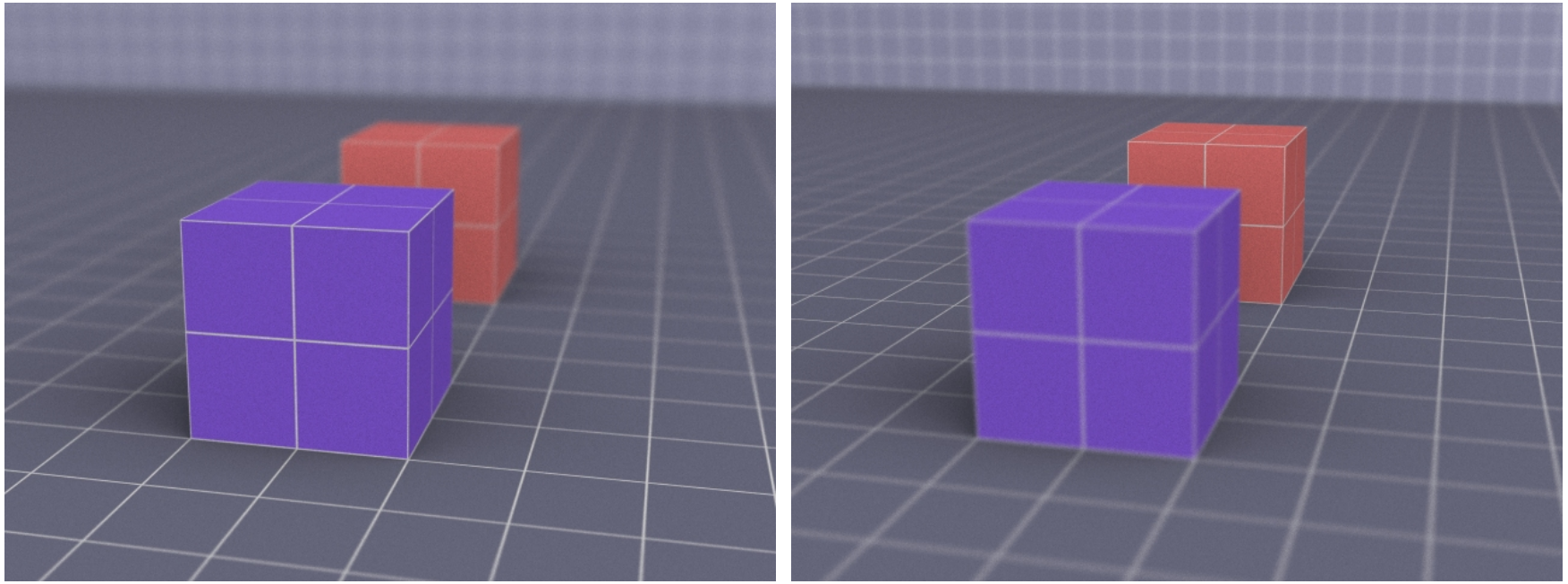


- A class **`LensCamera`** would generate rays like this:
  - Sample the whole shutter opening by some distribution, shoot ray from each *sample* point through focal plane = image plane



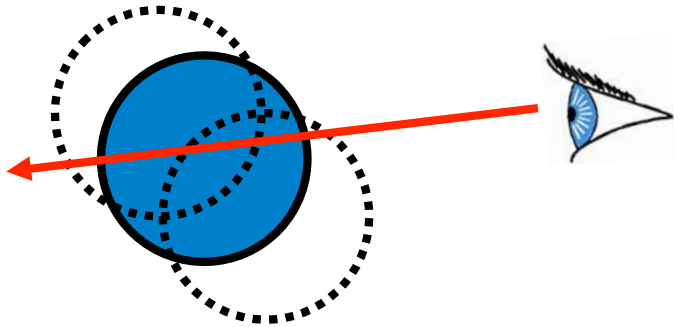
- Remark:
  - Again, use stratified sampling for sampling the disc (= shutter)

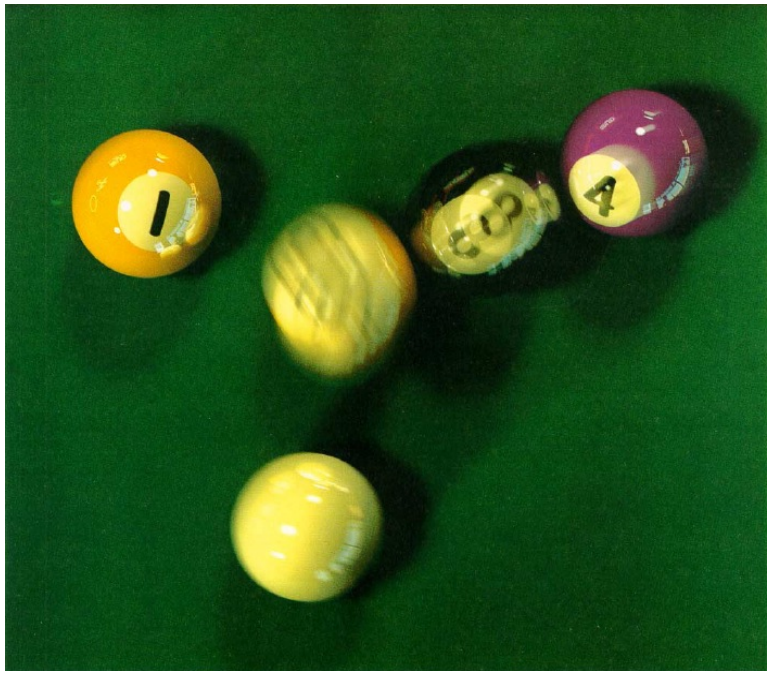
Examples



Motion Blur (Bewegungsunschärfe)

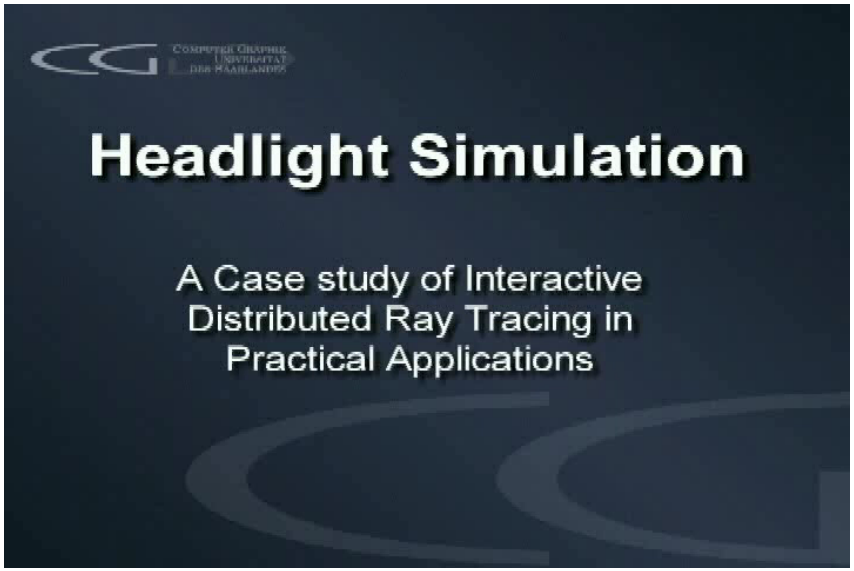
- Goal: compute "average" image for time interval $[t_0, t_1]$, during which objects move
- Sample time interval with $t \in [t_0, t_1]$ and shoot one ray per pixel per time t
- When computing ray-object intersections, use positions $P = P(t)$ for all objects
- Average color of all rays for one pixel



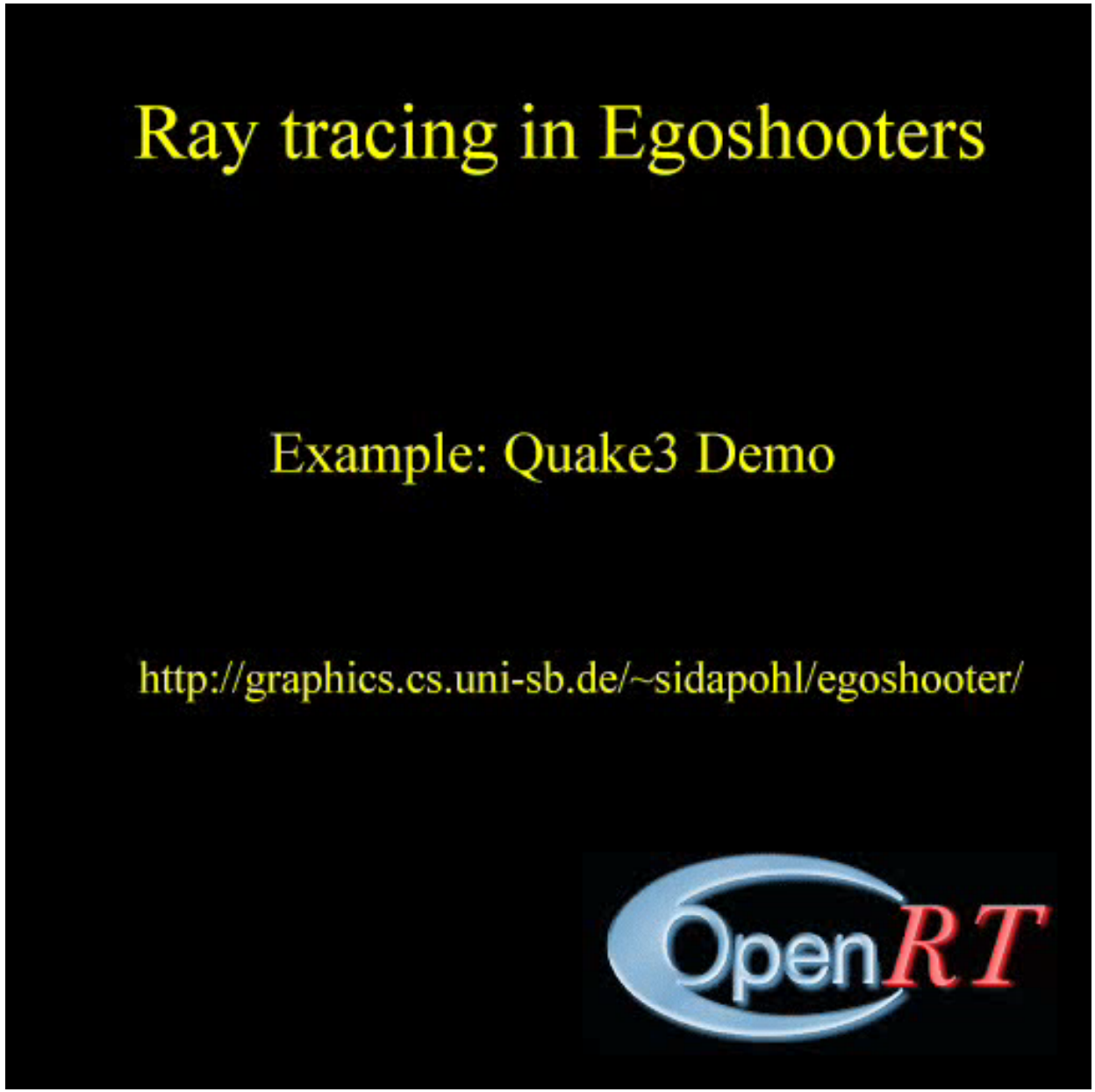


"But is it real-time?"

- Ray Tracing used to be very slow (and still is slower than polygonal)
- For some, simple scene, it is real-time already
  - And low screen resolutions
- Example: the OpenRT project
- Special purpose hardware (Intel tried), PC cluster
- Perhaps, some day, graphics cards will do ray-tracing only ...



Uni Saarbrücken



Quake 3 mit Ray-Tracing. Plattform: Cluster mit 20 AMD XP1800. 2004
http://graphics.cs.uni-sb.de/~sidapohl/egoshooter/ [outdated link]

Eine Anmerkung zu Typos

- Typos passieren auch auf den Folien
  - Keine Angst haben zu fragen!
  - Bitte teilen Sie mir Fehler mit

- Typos passieren sogar in Lehrbüchern
  - Ich selbst habe 2 nicht-triviale Fehler im Shirley-Buch, 2-te Auflage gefunden [WS 05/06]
  - Fazit: mitdenken, nicht einfach direkt kopieren