Tracé de Rayons,Ombres et Éclairage Global

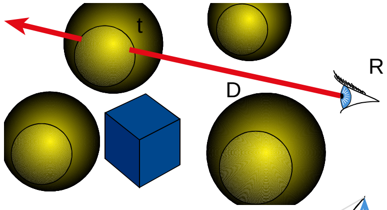
Séance 4

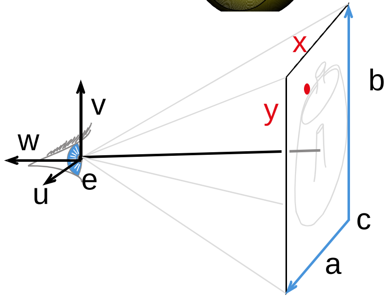
Images et Sons de Synthèse

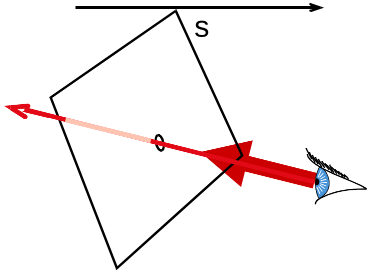
Tracé de Rayons

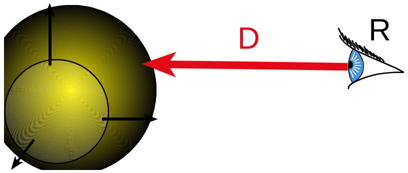
Tracé de Rayons

- Introduction
- Camera and ray generation
- Ray-plane intersection
- Ray-sphere intersection



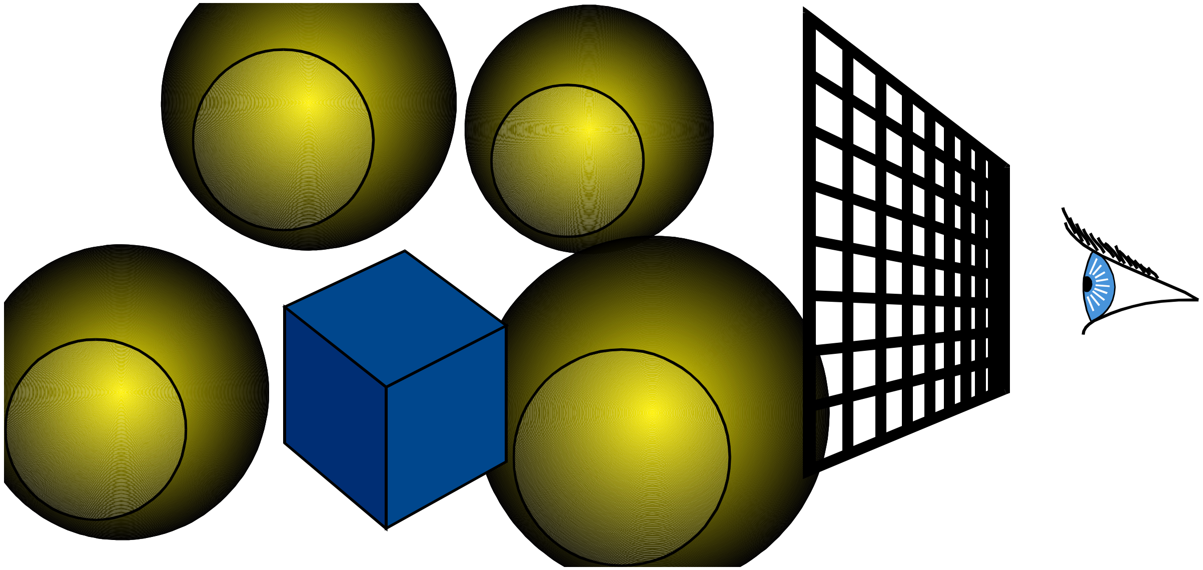






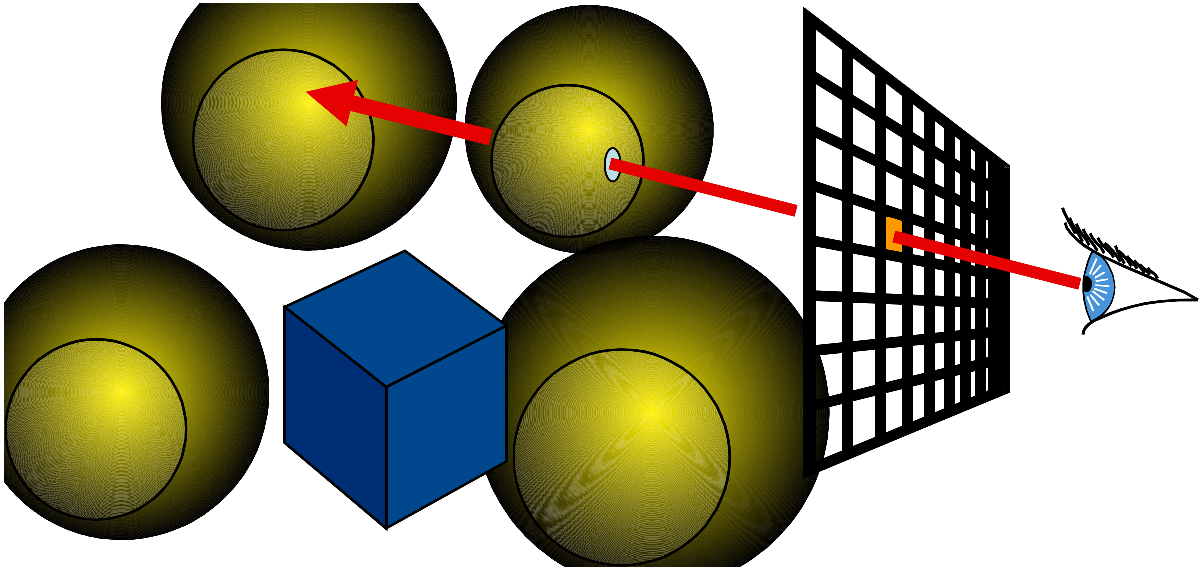
Ray Casting

```
For every pixel
   Construct a ray from the eye
   For every object in the scene
        Find intersection with the ray
        Keep if closest
```



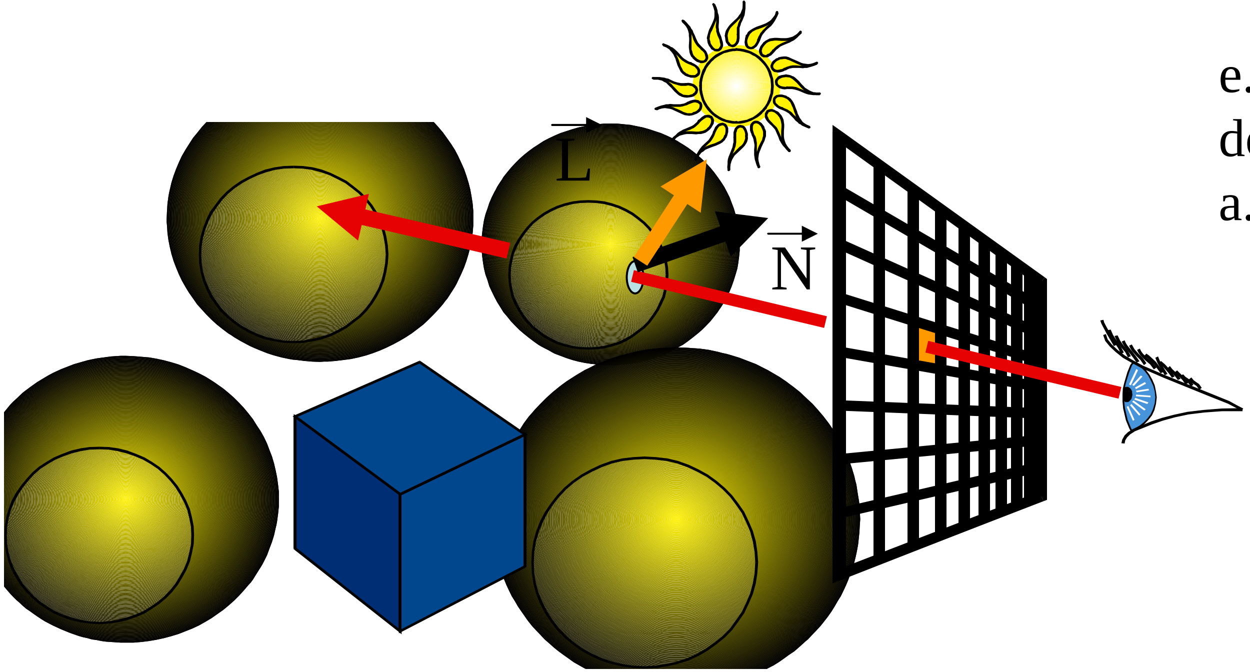
Ray Casting

```
For every pixel
  Construct a ray from the eye
  For every object in the scene
      Find intersection with the ray
      Keep if closest
```



Shading

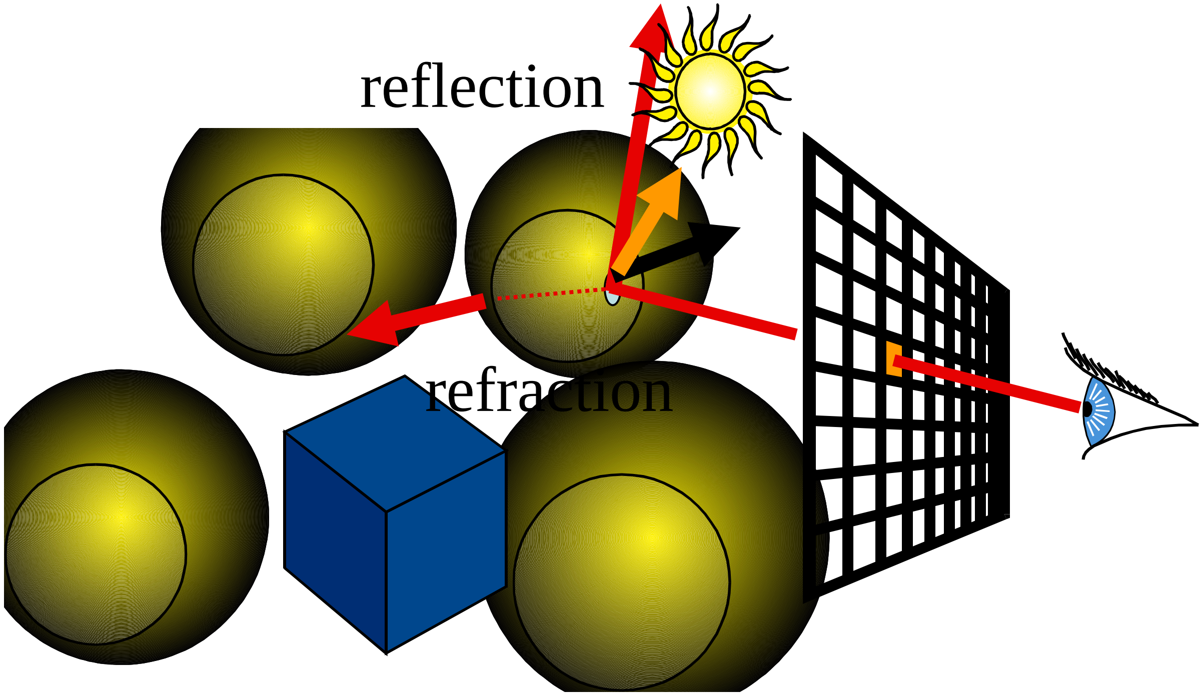
```
For every pixel
  Construct a ray from the eye
  For every object in the scene
      Find intersection with the ray
      Keep if closest
   Shade depending on light and normal vector
```



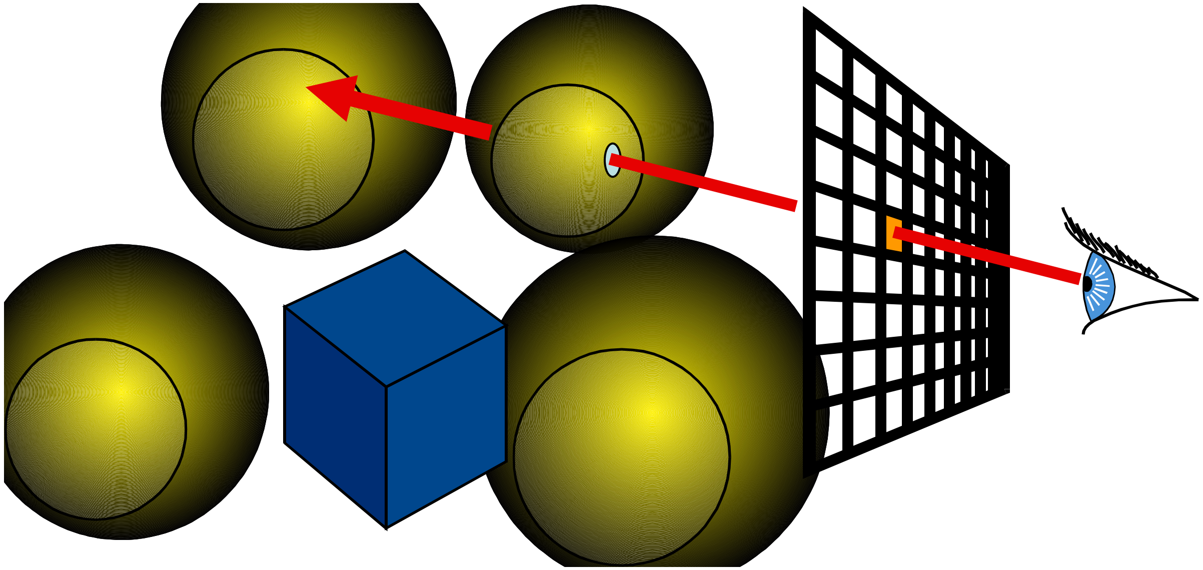
e.g. diffuse shading:
dot product N.L
a.k.a. Lambertian

Ray Tracing

- Secondary rays (shadows, reflection, refraction)

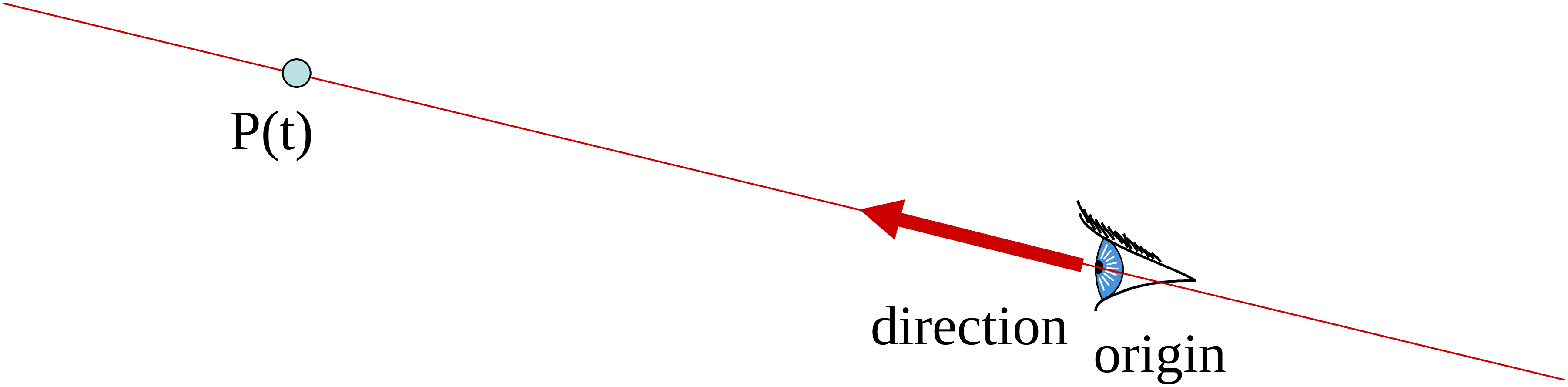


Ray representation?



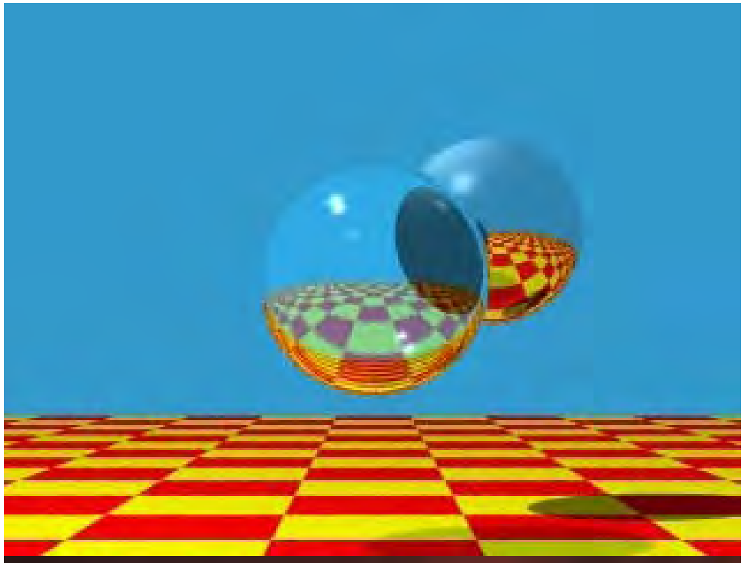
Ray representation

- Two vectors:
  - Origin
  - Direction (normalized is better)
- Parametric line
  - P(t) = origin + t * direction

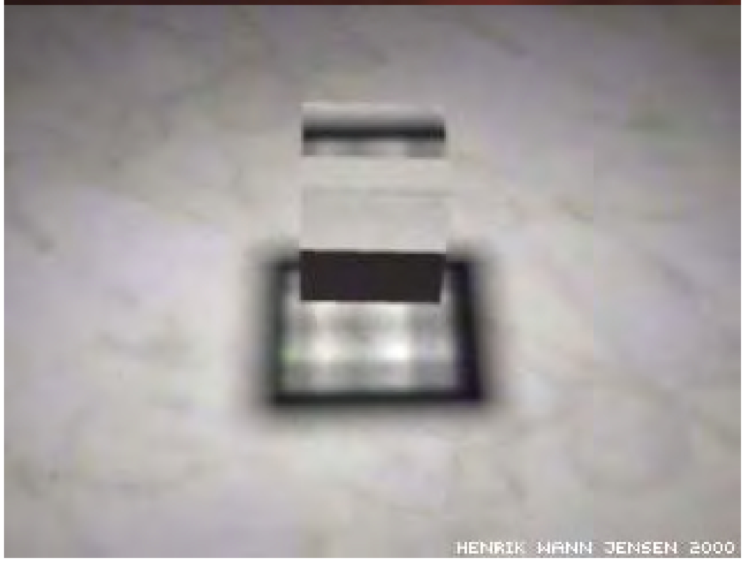


Ray Tracing

- Original Ray-traced image by Whitted (1981)

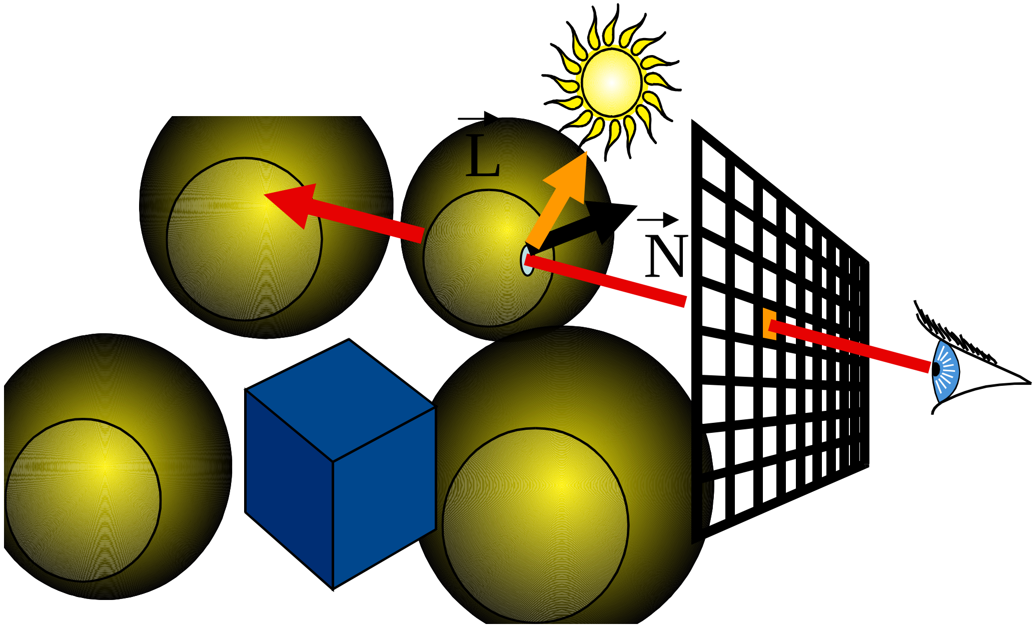


- Image computed using the Dali ray tracer by Henrik Wann Jensen
- Environment map by Paul Debevec



Ray casting

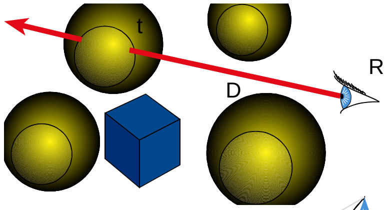
```
For every pixel
  Construct a ray from the eye
  For every object in the scene
      Find intersection with the ray
      Keep if closest
   Shade depending on light and normal vector
```



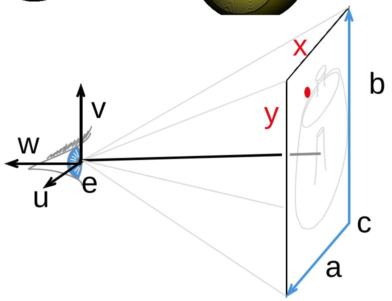
Finding the intersection and normal is the central part of ray casting

Overview of today

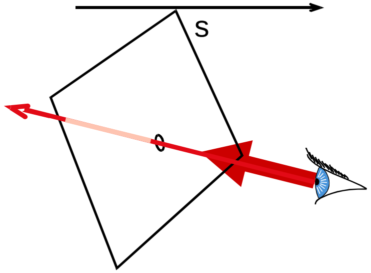
- Introduction



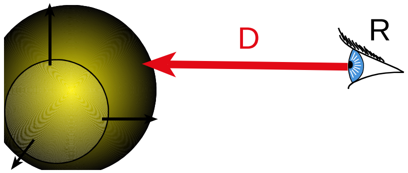
- Camera and ray generation



- Ray-plane intersection

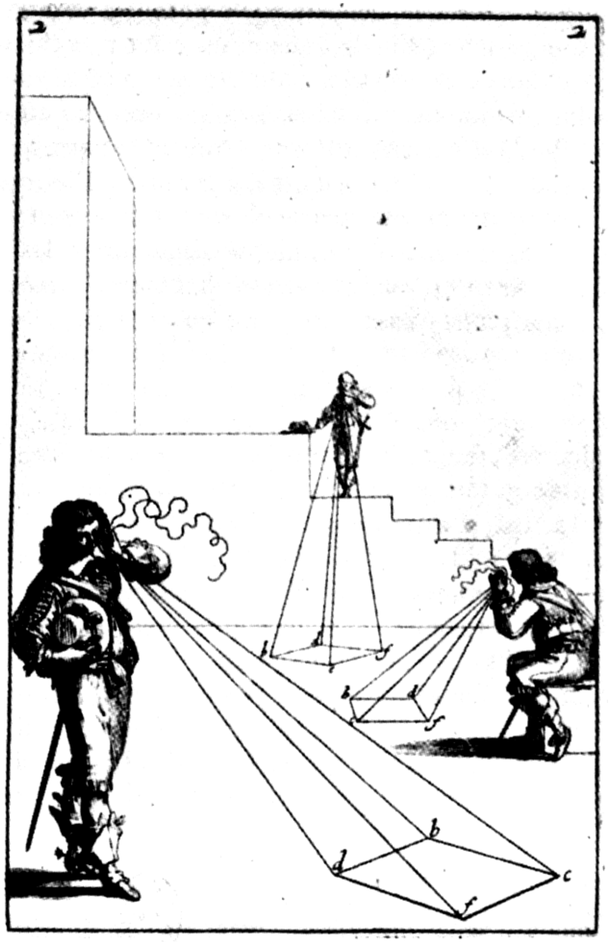


- Ray-sphere intersection



Cameras

```
For every pixel
   Construct a ray from the eye
   For every object in the scene
        Find intersection with the
          ray
        Keep if closest
```

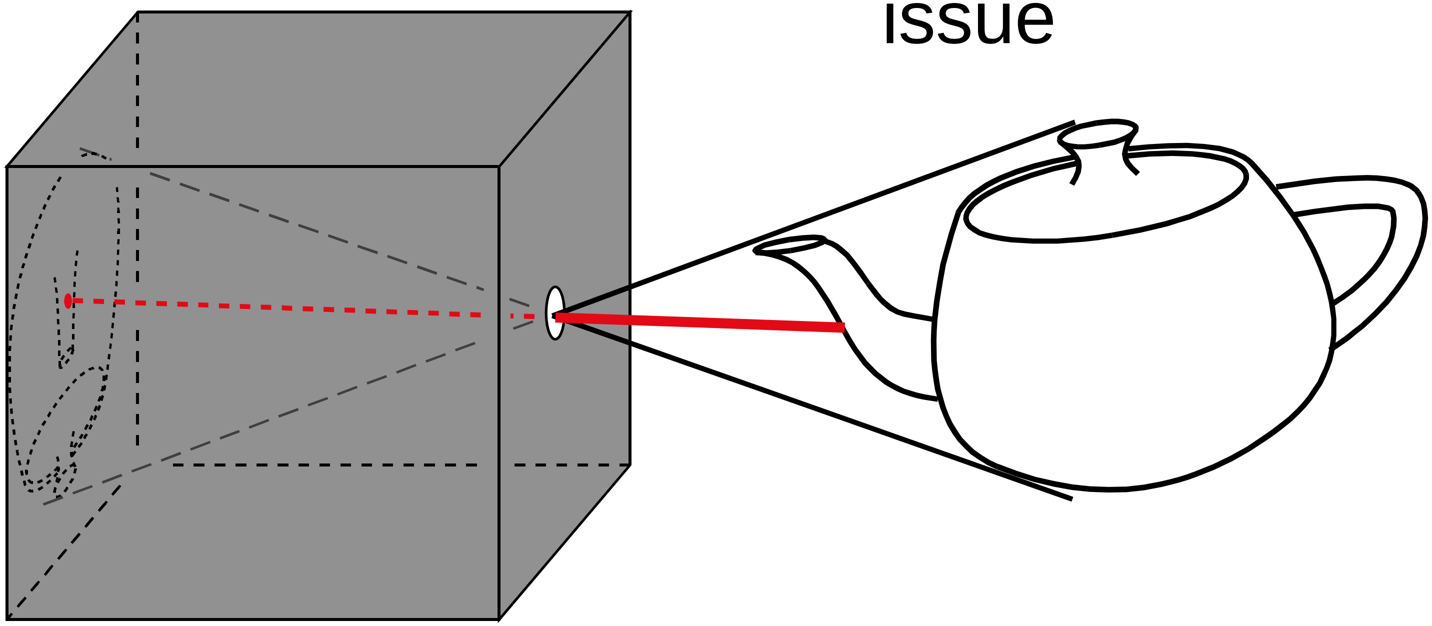


braham Bosse, *Les Perspecteurs*. Gravure extraite de la *M*

Pinhole camera

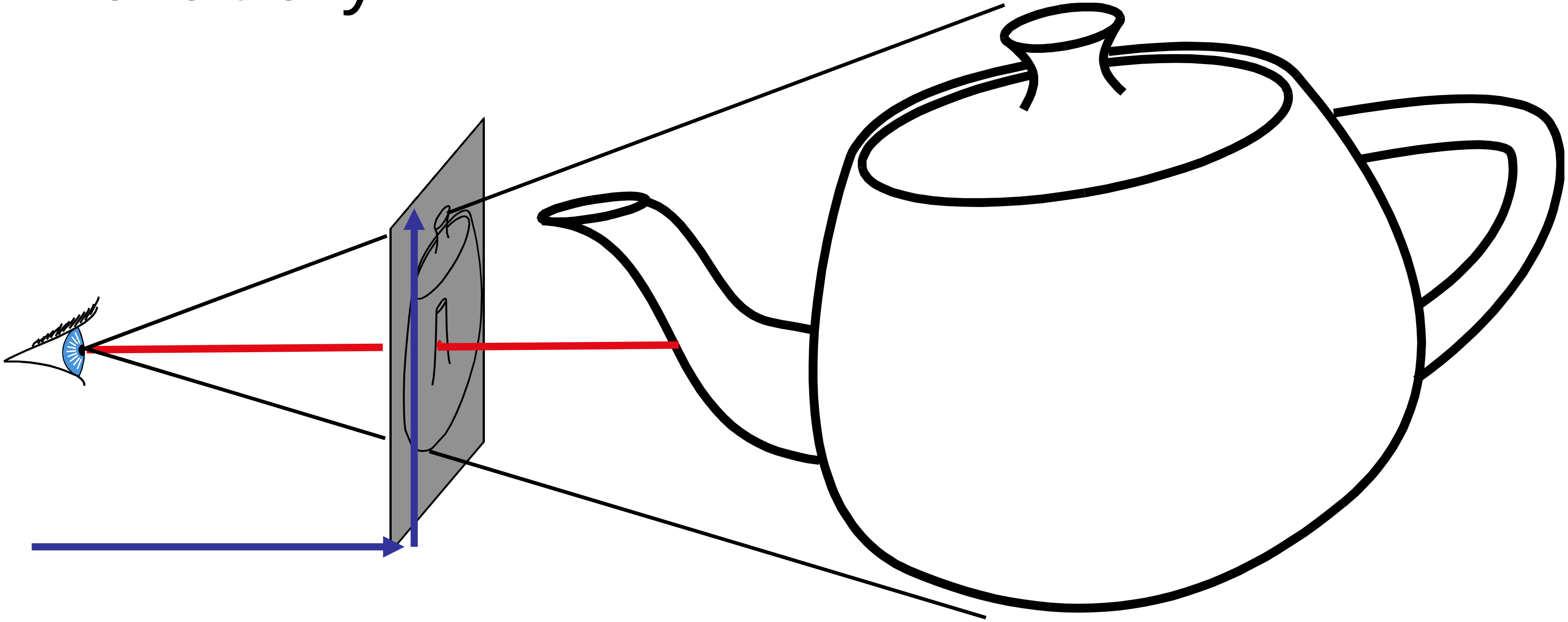
- Box with a tiny hole
- Inverted image
- Similar triangles

- Perfect image if hole infinitely small
- Pure geometric optics
- No depth of field issue



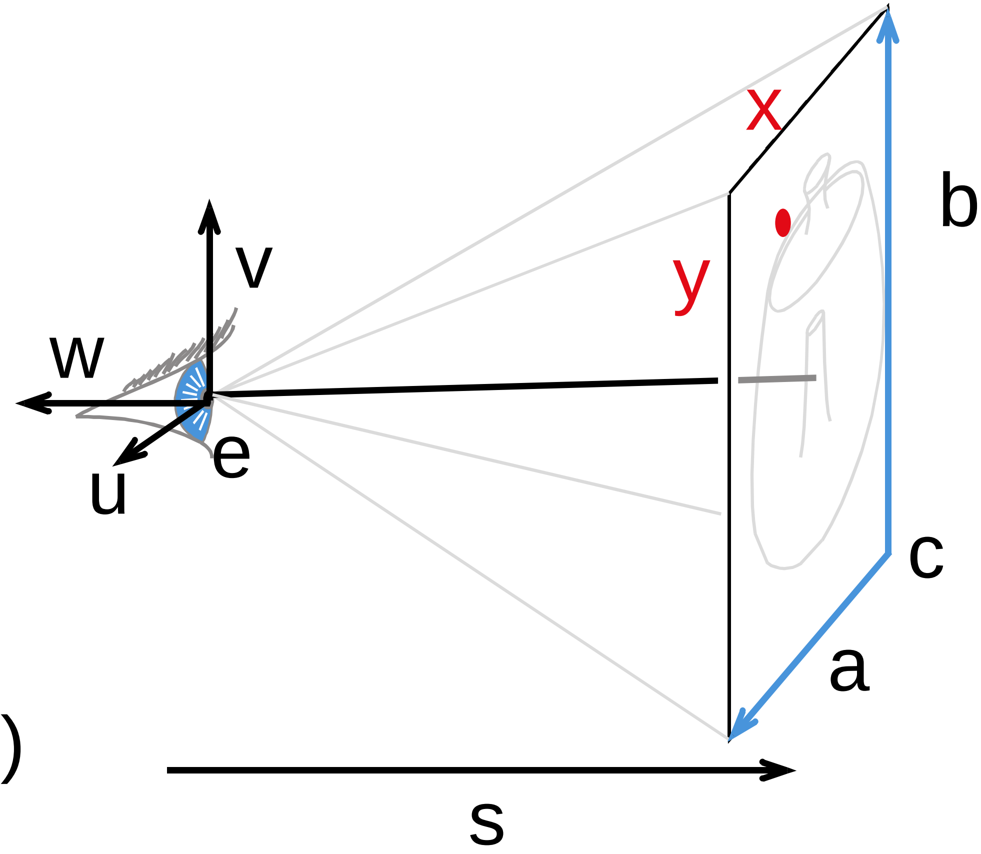
Simplified pinhole camera

- Eye-image pyramid (frustum)
- Note that the distance/size of image are arbitrary



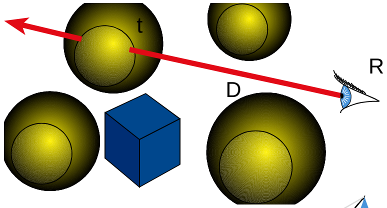
Camera description

- – Eye point e
- – Orthobasis u, v, w
- – Image distance s
- – Image rectangle (u0, v0, u1, v1)

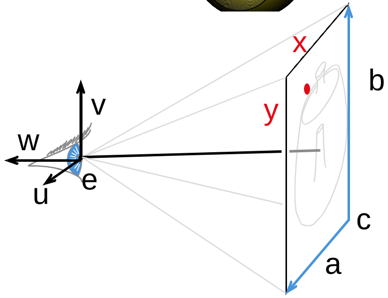


- – Deduce c (lower left)
- – Deduce a and b
- – Screen coordinates in [0,1]*[0,1]
- – A point is then c + x a +y b

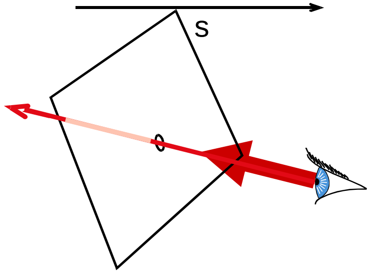
Ray Casting



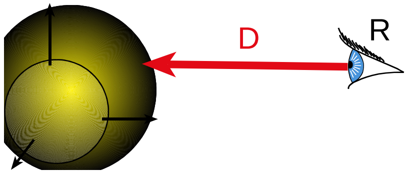
- Introduction



- Camera and ray generation



- Ray-plane intersection

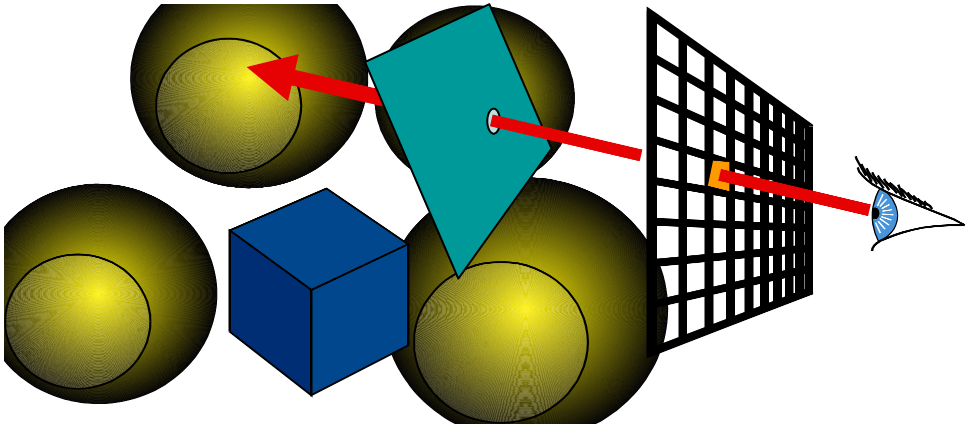


- Ray-sphere intersection

Ray Casting

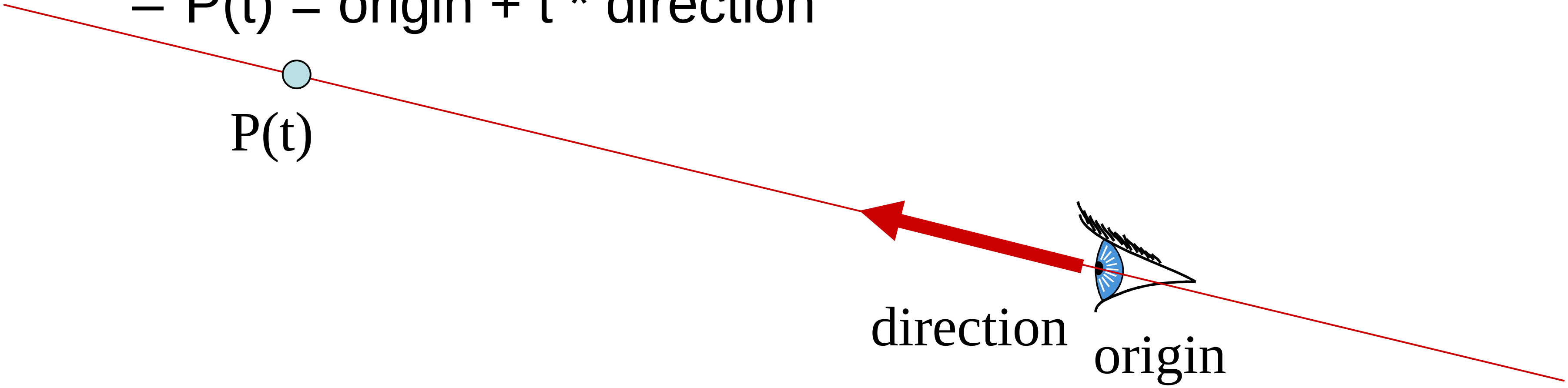
```
For every pixel
   Construct a ray from the eye
   For every object in the scene
        Find intersection with the ray
        Keep if closest
```

First we will study ray-plane intersection



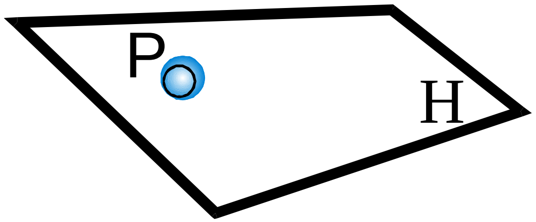
Recall: Ray representation

- Two vectors:
  - – Origin
  - – Direction (normalized)
- Parametric line
  - – P(t) = origin + t * direction



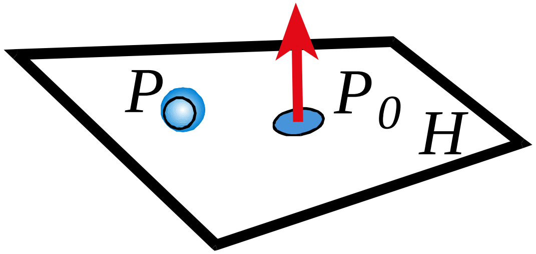
3D plane equation

- Implicit plane equation
  $H(p) = Ax+By+Cz+D = 0$
- Gradient of H?



3D plane equation

- Implicit plane equation
  $H(p) = Ax+By+Cz+D = 0$
- Gradient of H?
- Plane defined by
  - $P_0(x,y,z,1)$
  - $n(A, B, C, 1)$

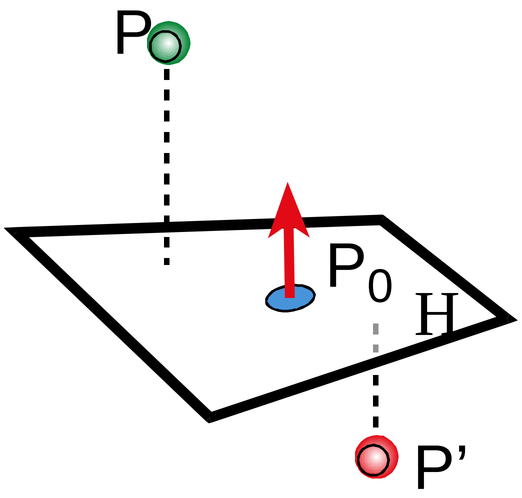


Explicit vs. implicit?

- Plane equation is implicit
  - Solution of an equation
  - Does not tell us how to generate a point on the plane
  - Tells us how to check that a point is on the plane
- Ray equation is explicit
  - Parametric
  - How to generate points
  - Harder to verify that a point is on the ray

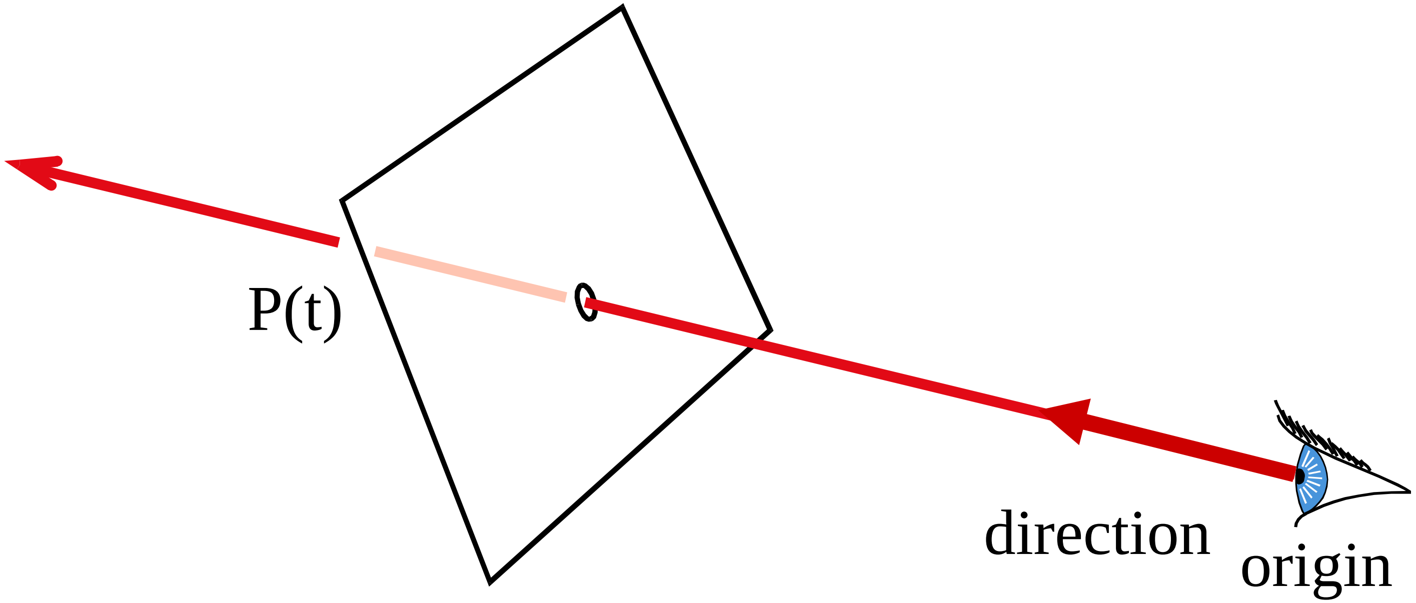
Plane-point distance

- Plane $Hp=0$
- If n is normalized
  $d=HP$
- Signed distance!



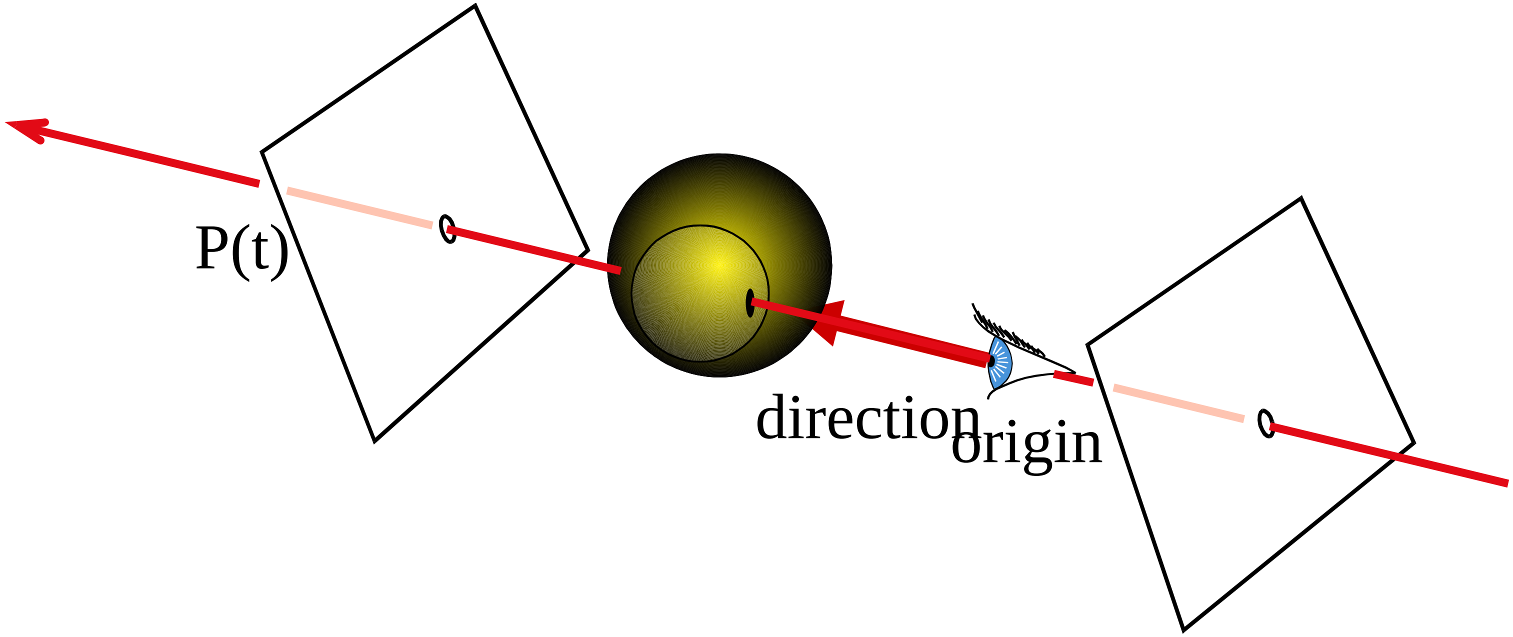
Line-plane intersection

- Insert explicit equation of line into implicit equation of plane



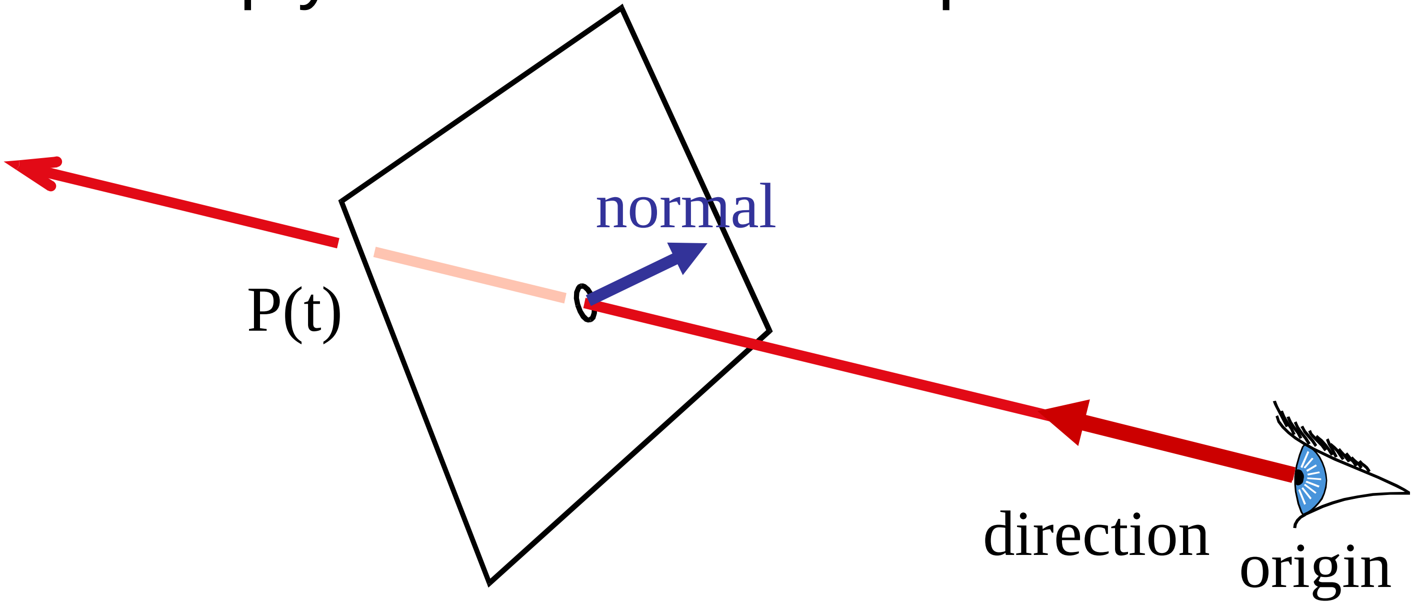
Additional house keeping

- Verify that intersection is closer than previous
- Verify that it is in the allowed range
  (in particular not behind the camera, t<0)

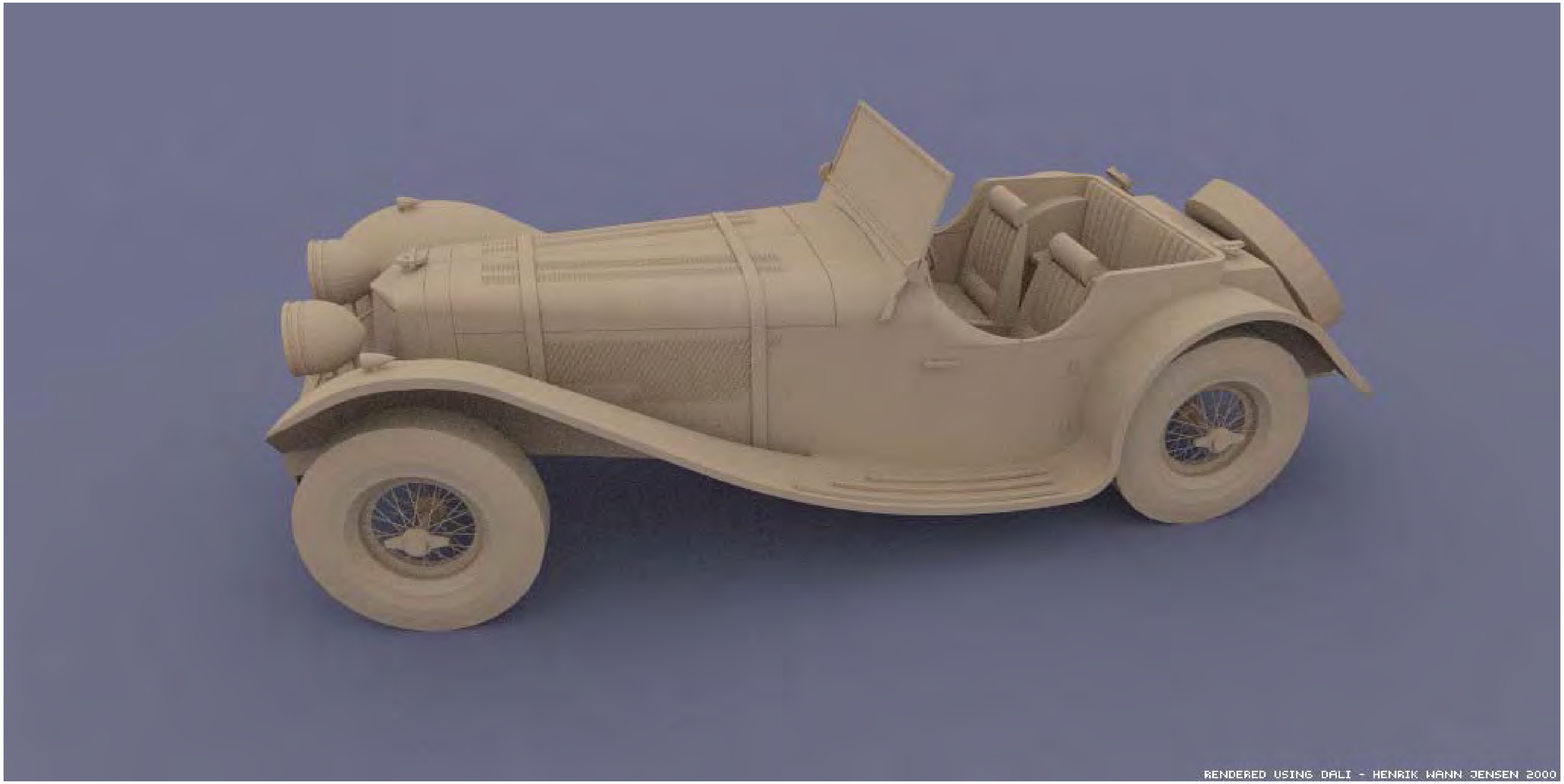


Normal

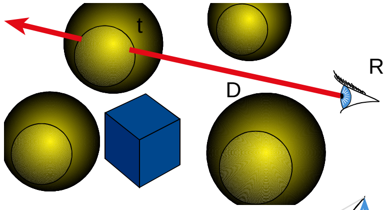
- For shading (recall, diffuse: dot product between light and normal)
- Simply the normal to the plane

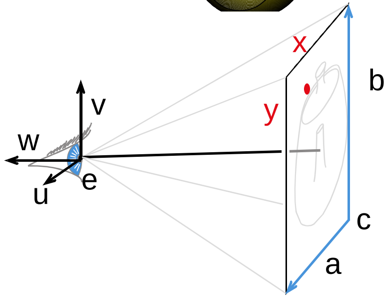


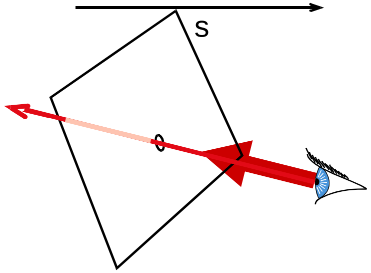
- Image by Henrik Wann Jensen using Ray Casting

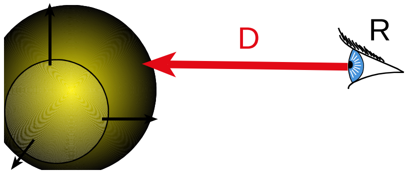


- Introduction
- Camera and ray generation
- Ray-plane intersection
- Ray-sphere intersection



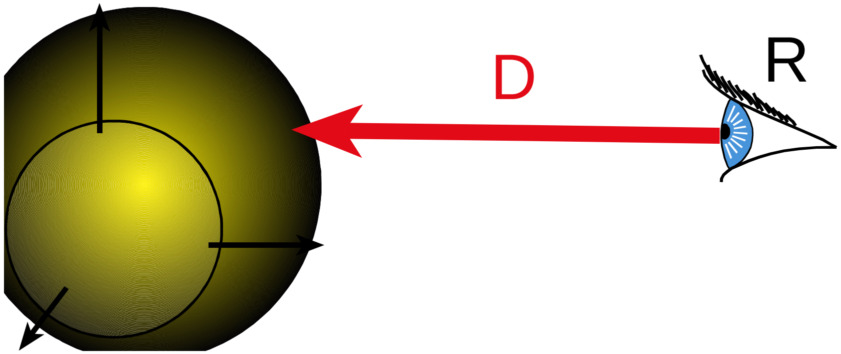






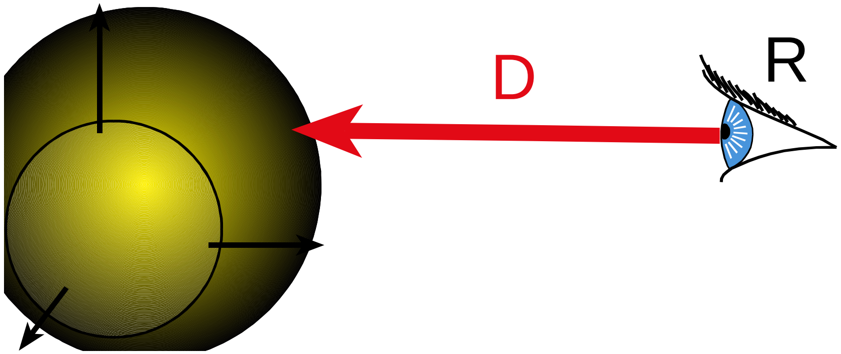
Sphere equation

- Sphere equation (implicit): $||P||^2 = r^2$
- (assume centered at origin,
  easy to translate)



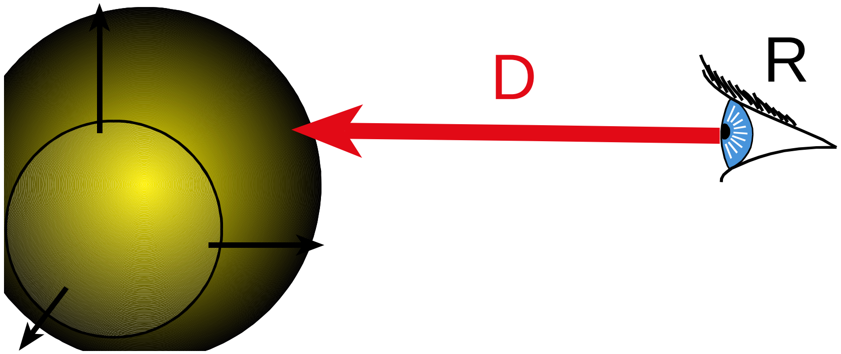
Ray-Sphere Intersection

- Sphere equation (implicit): $||P||^2 = r^2$
- Ray equation (explicit): $P(t) = R+tD$ with $||D|| = 1$
- Intersection means both are satisfied



Ray-Sphere Intersection

$$
\begin{aligned}
0 &= \mathbf{P} \cdot \mathbf{P} - r^2 \\
&= (\mathbf{R} + t\mathbf{D}) \cdot (\mathbf{R} + t\mathbf{D}) - r^2 \\
&= \mathbf{R} \cdot \mathbf{R} + 2t\mathbf{D} \cdot \mathbf{R} + t^2\mathbf{D}^2 - r^2 \\
&= t^2 + 2t\mathbf{D} \cdot \mathbf{R} + \mathbf{R} \cdot \mathbf{R} - r^2
\end{aligned}
$$

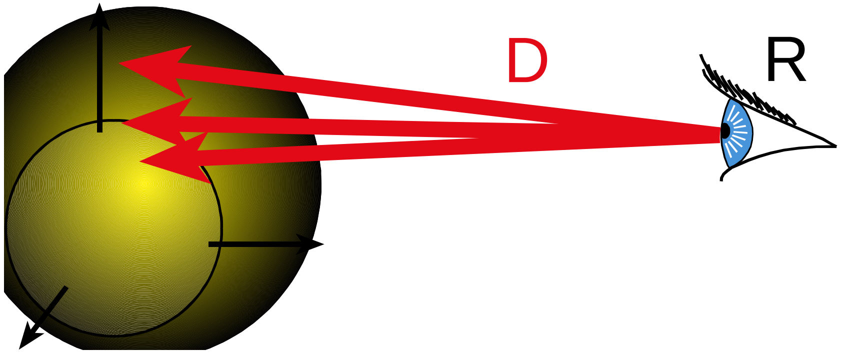


Ray-Sphere Intersection

- This is just a quadratic at^2 + bt + c = 0, where
  - a = 1
  - b = 2D.R
  - c = R.R – r^2
- With discriminant $$d = \sqrt{b^2 - 4ac}$$
- and solutions $$t_\pm = \frac{-b \pm d}{2a}$$

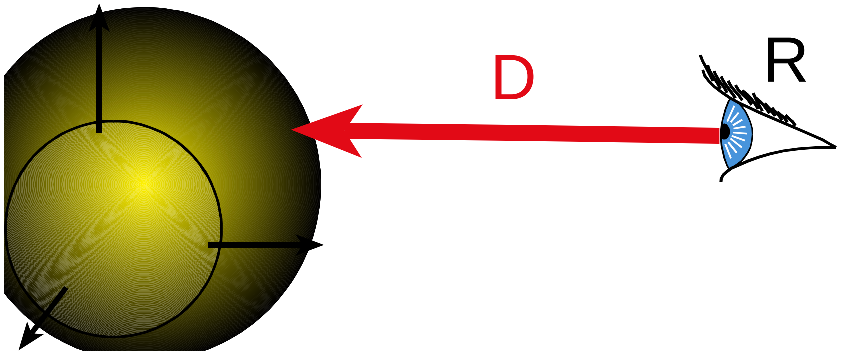
Ray-Sphere Intersection

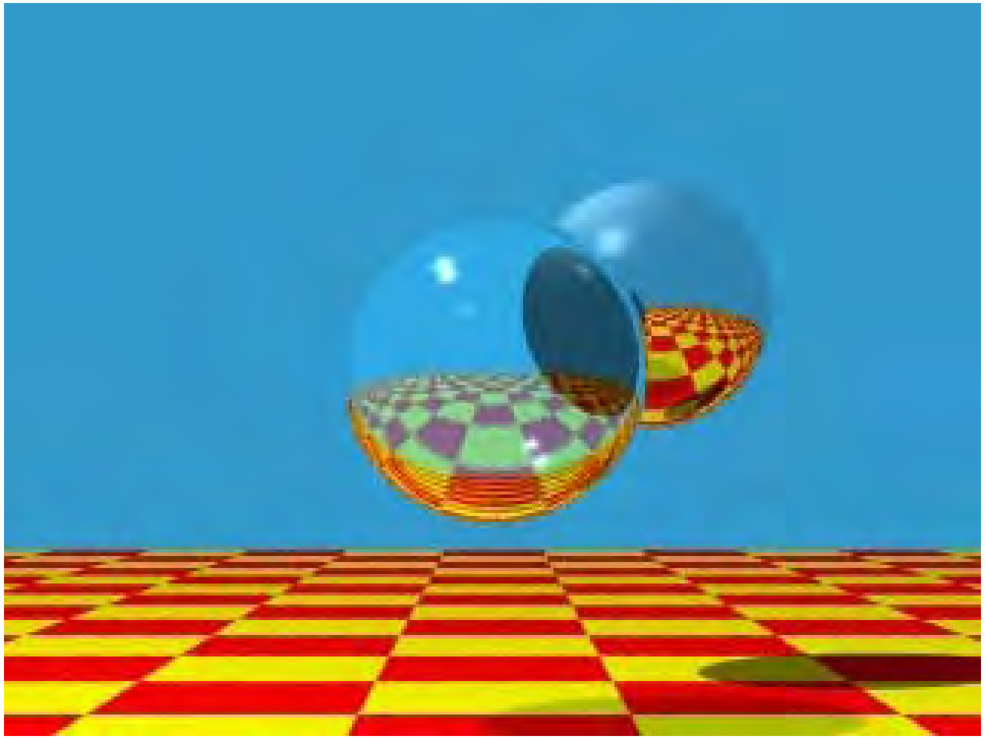
- Discriminant $d = \sqrt{b^2 - 4ac}$
- Solutions $t_{\pm} = \dfrac{-b \pm d}{2a}$
- Three cases, depending on sign of b^2 –4ac
- Which root (t+ or t-) should you choose?
  - Closest positive! (usually t-)



Ray-Sphere Intersection

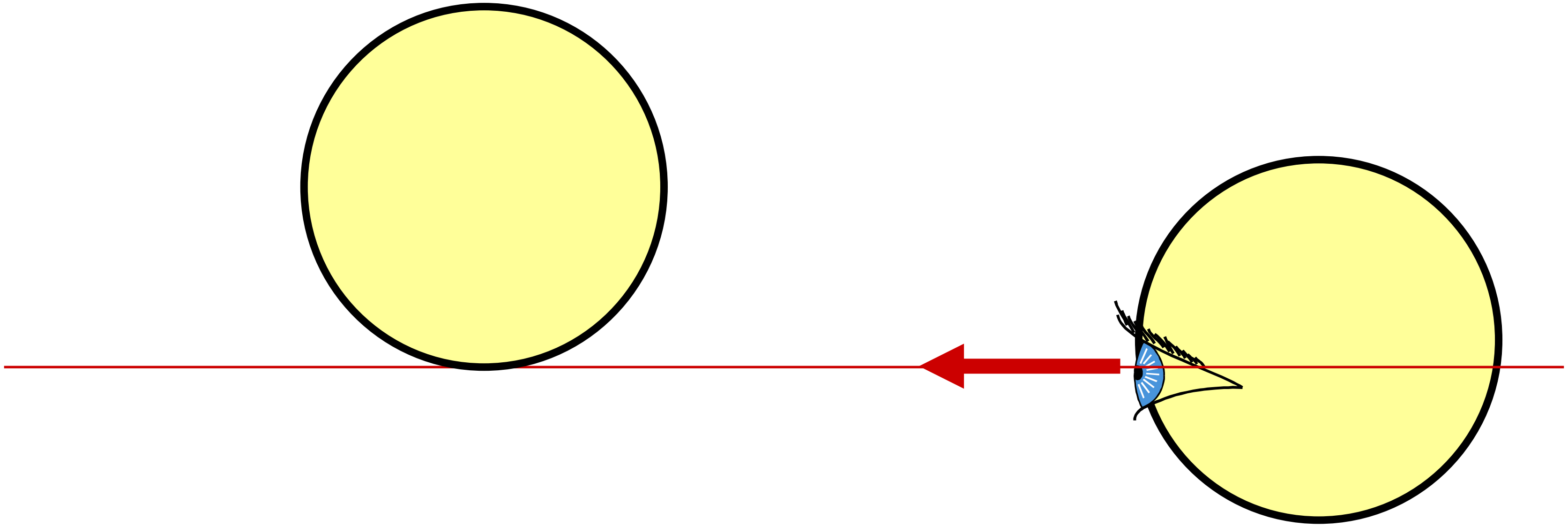
- So easy that all ray-tracing images have spheres!





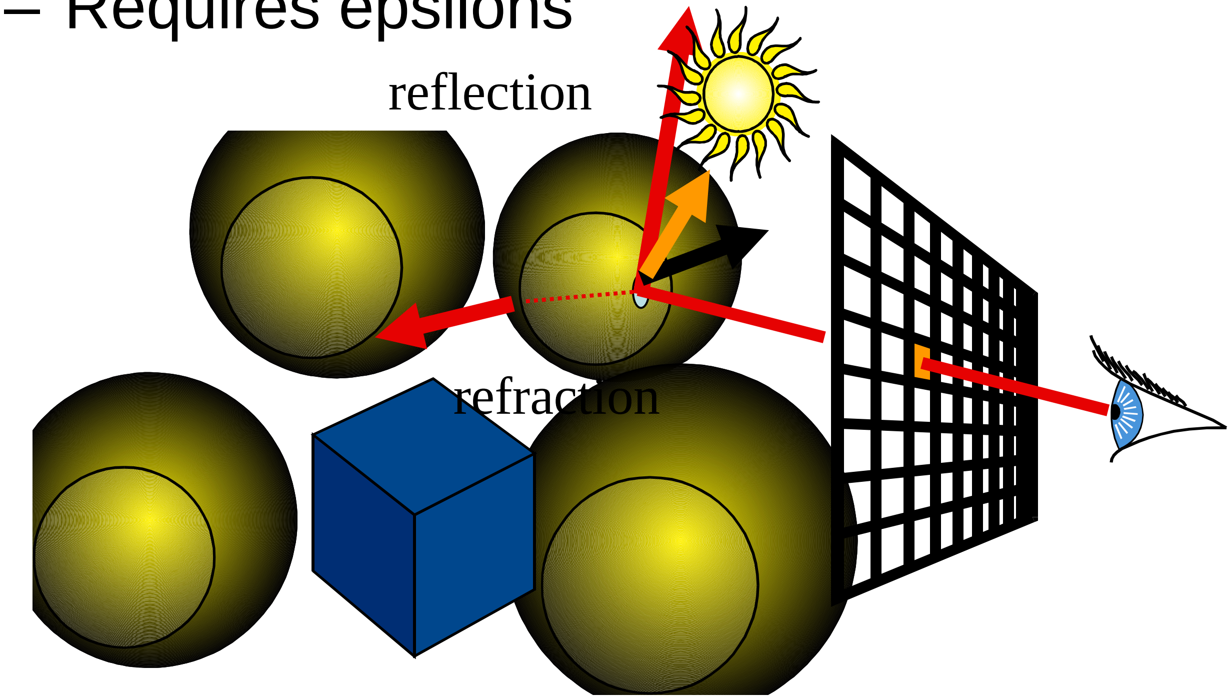
Precision

- What happens when
  - Origin is on an object?
  - Grazing rays?
- Problem with floating-point approximation



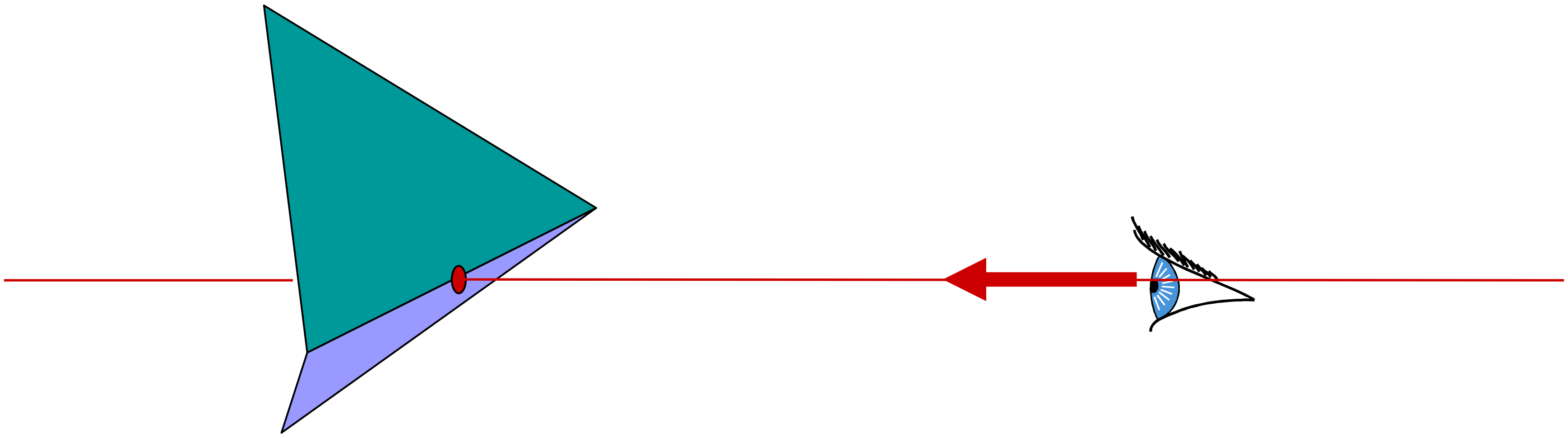
The evil ε

- In ray tracing, do NOT report intersection for rays starting at the surface (no false positive)
  - Because secondary rays
  - Requires epsilons



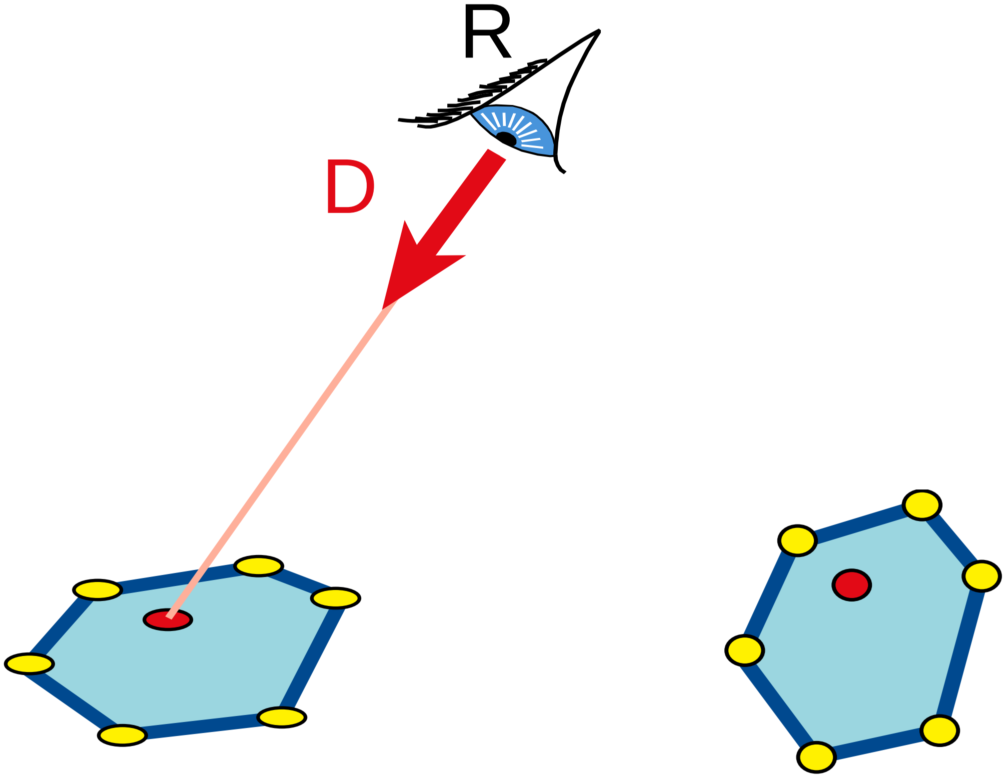
The evil ε: a hint of nightmare

- Edges in triangle meshes
  - Must report intersection (otherwise not watertight)
  - No false negative



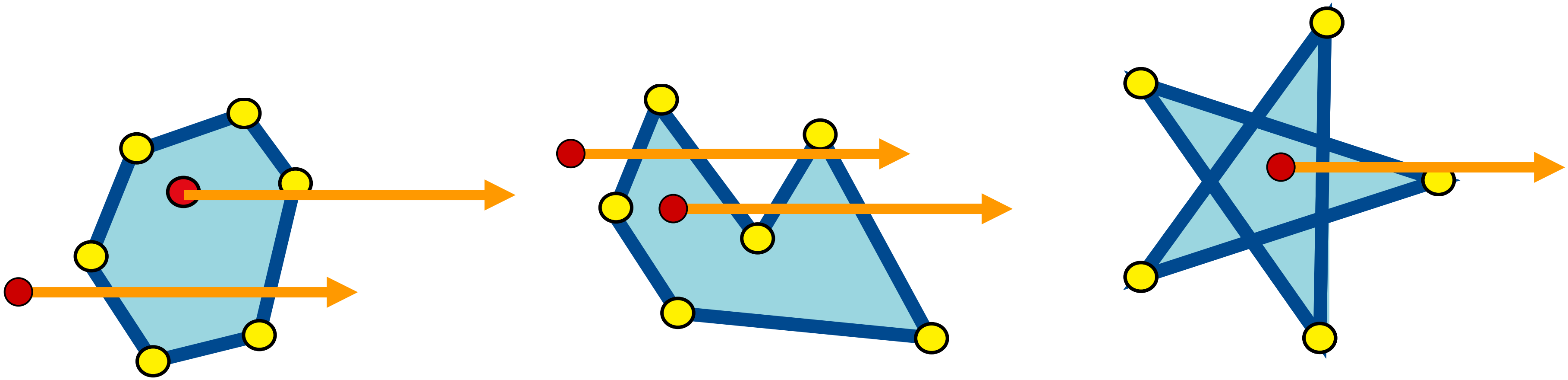
Ray-polygon intersection

- Ray-plane intersection
- Test if intersection is in the polygon
  - Solve in the 2D plane



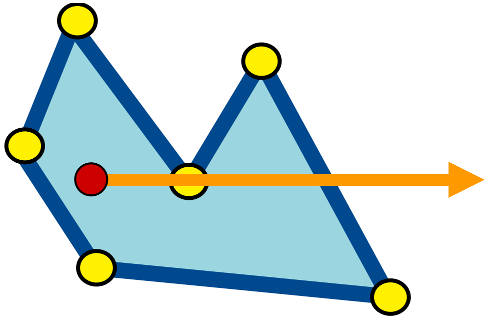
Point inside/outside polygon

- Ray intersection definition:
  - Cast a ray in any direction
    - (axis-aligned is smarter)
  - Count intersection
  - If odd number, point is inside
- Works for concave and star-shaped



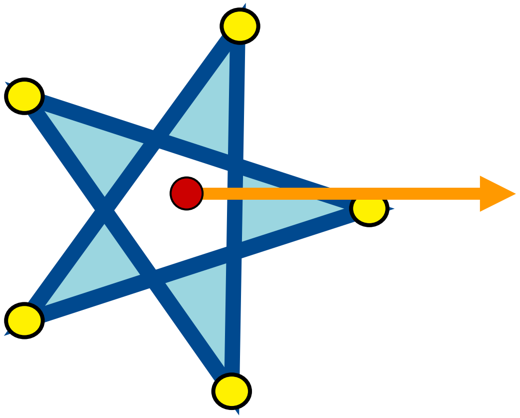
Precision issue

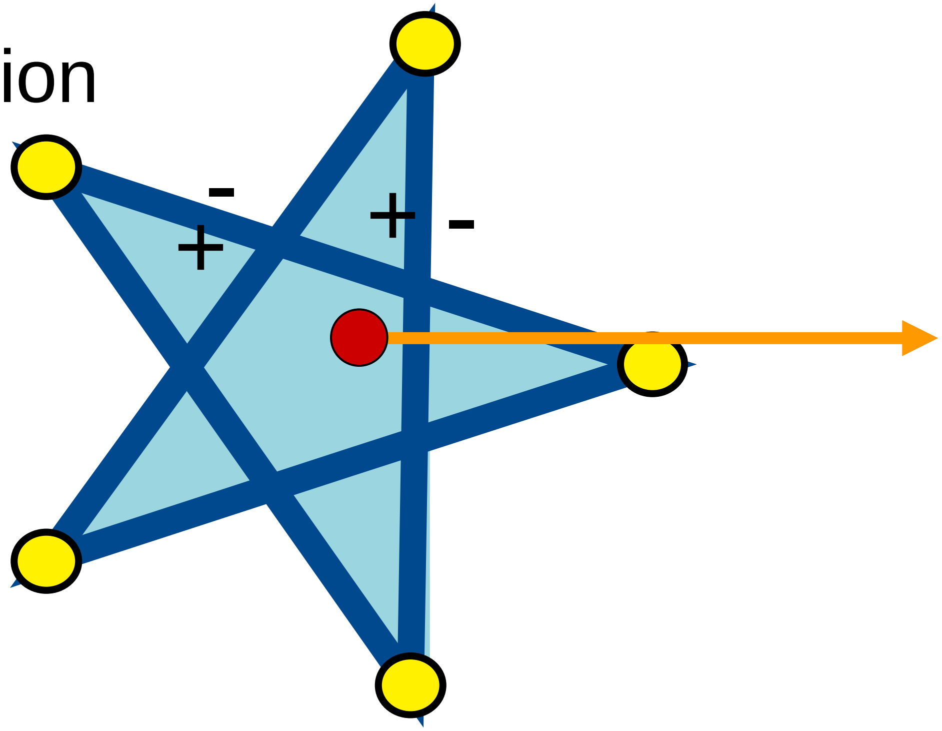
- What if we intersect a vertex?
  - We might wrongly count an intersection for each adjacent edge
- Decide that the vertex is always above the ray



Winding number

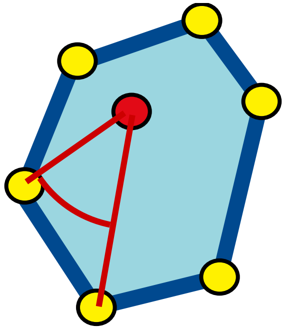
- To solve problem with star pentagon
- Oriented edges
- Signed number of intersection
- Outside if 0 intersection

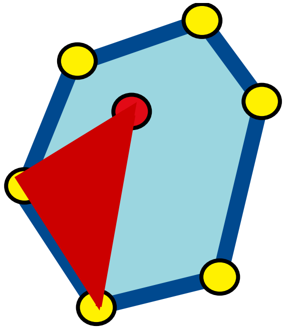




Alternative definitions

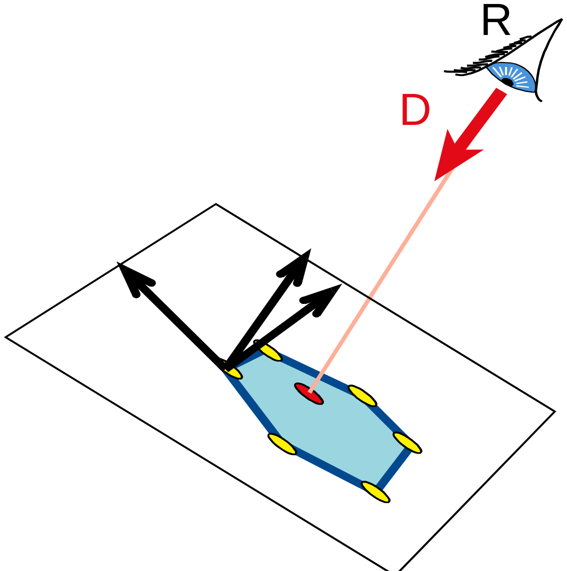
- Sum of the signed angles from point to vertices
  - 360 if inside, 0 if outside
- Sum of the signed areas of point-edge triangles
  - Area of polygon if inside, 0 if outside

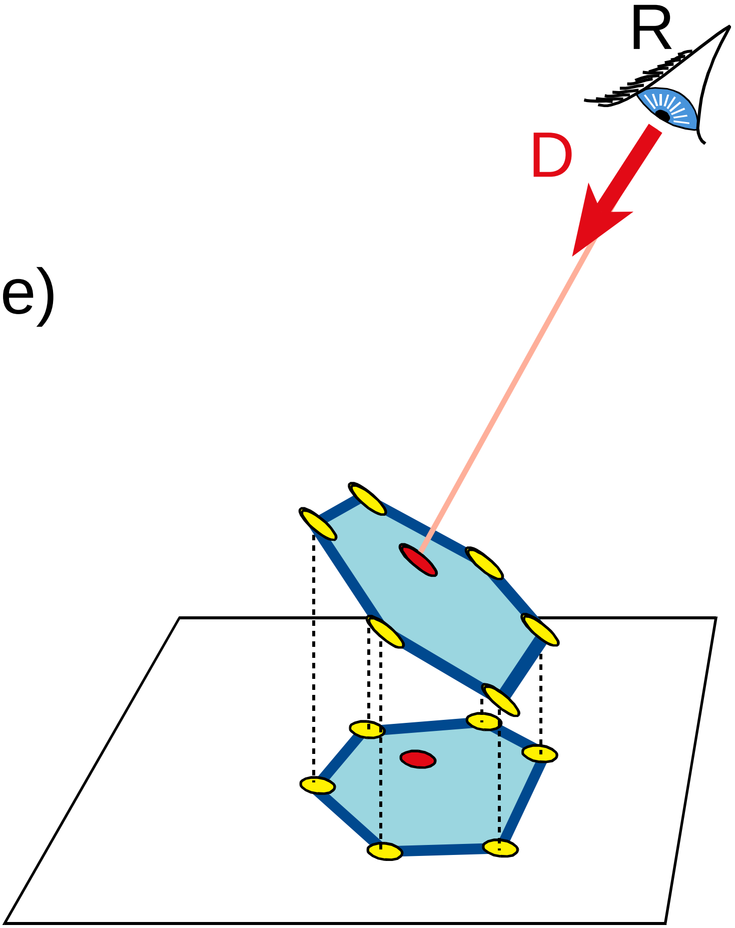




How do we project into 2D?

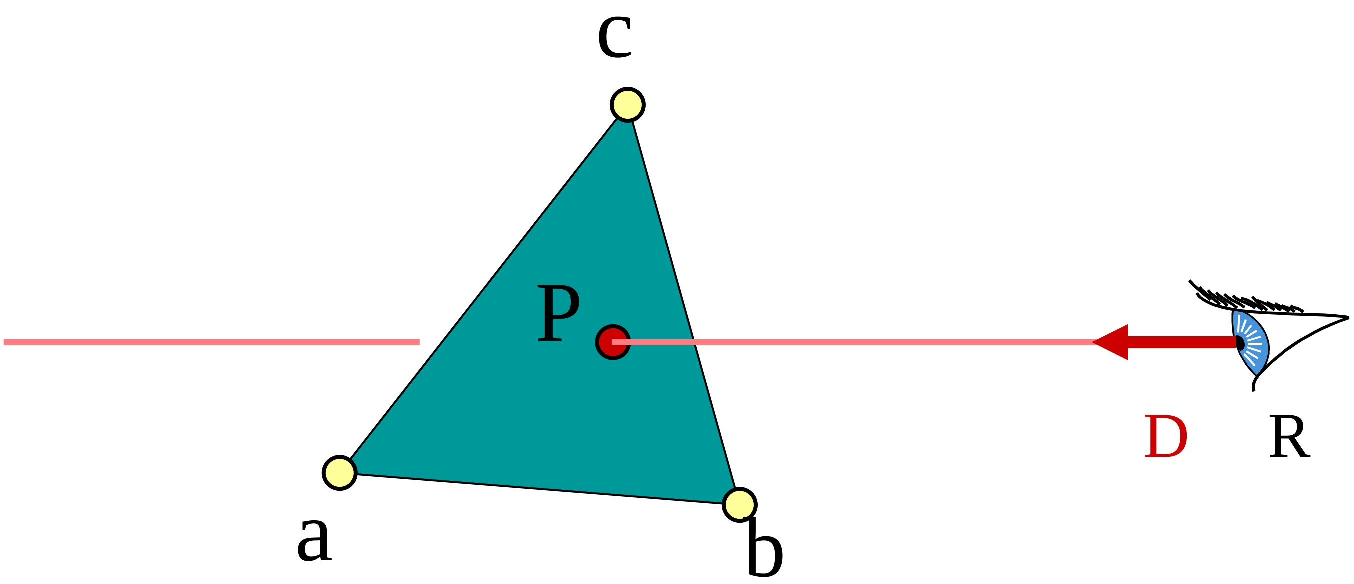
- Along normal
  - Costly
- Along axis
  - Smarter (just drop 1 coordinate)
  - Beware of parallel plane





Ray triangle intersection

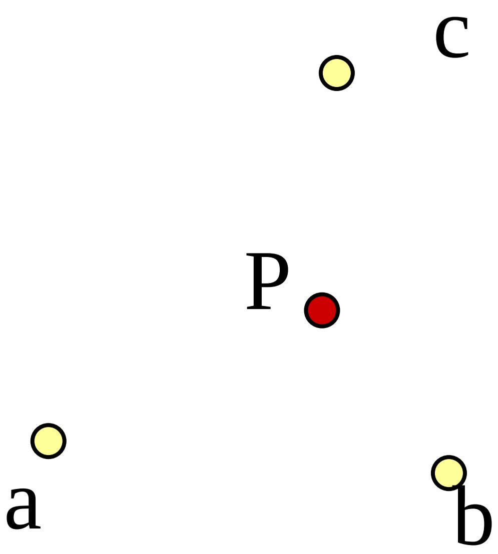
- Use ray-polygon
- Or try to be smarter
  - – Use barycentric coordinates



Barycentric definition of a plane

[Möbius, 1827]

- $P(\alpha, \beta, \gamma) = \alpha a + \beta b + \gamma c$
  with $\alpha + \beta + \gamma = 1$



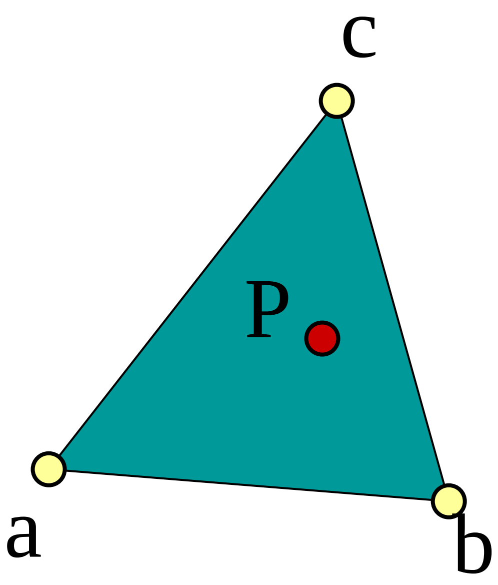
Barycentric definition of a triangle

- $P(\alpha, \beta, \gamma)=\alpha a+\beta b+\gamma c$ with $\alpha + \beta + \gamma = 1$

$0 < \alpha < 1$

$0 < \beta < 1$

$0 < \gamma < 1$

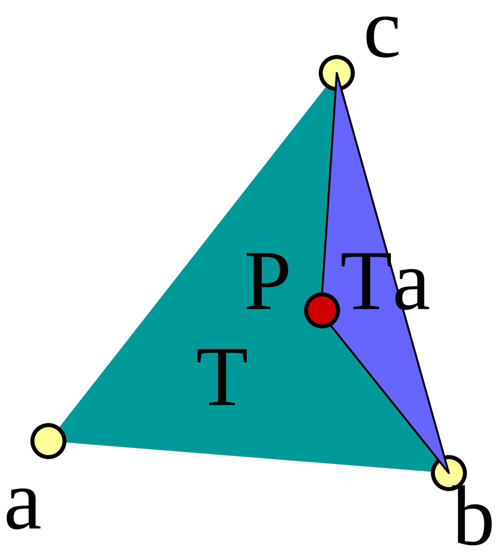


Given P, how can we compute α, β, γ ?

- Compute the areas of the opposite subtriangle
  - Ratio with complete area

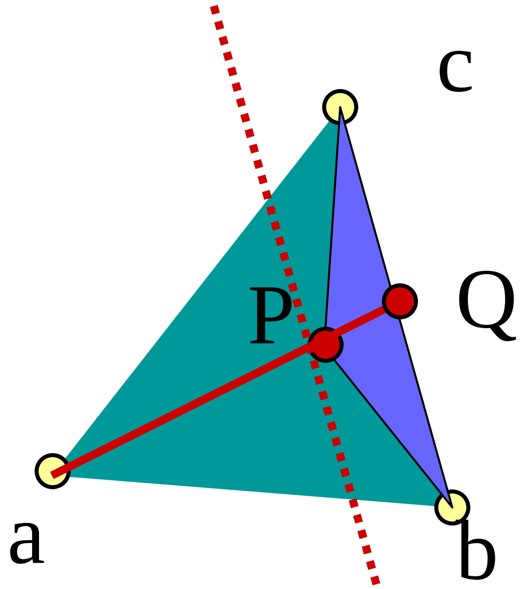
$\alpha = A_a/A$, $\beta = A_b/A$ $\gamma = A_c/A$

Use signed areas for points outside the triangle



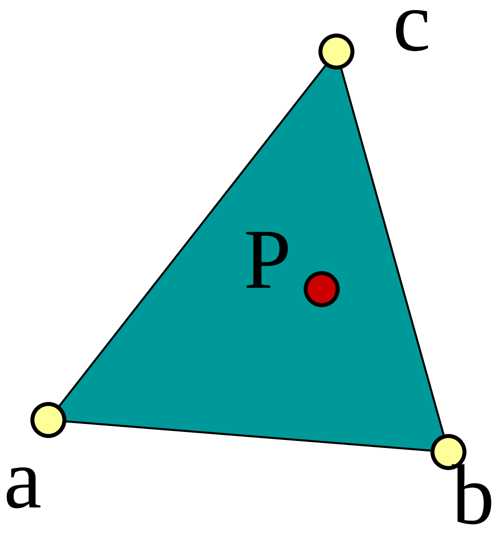
Intuition behind area formula

- P is barycenter of a and Q
- A is the interpolation coefficient on aQ
- All points on line parallel to bc have the same α
- All such Ta triangles have same height/area



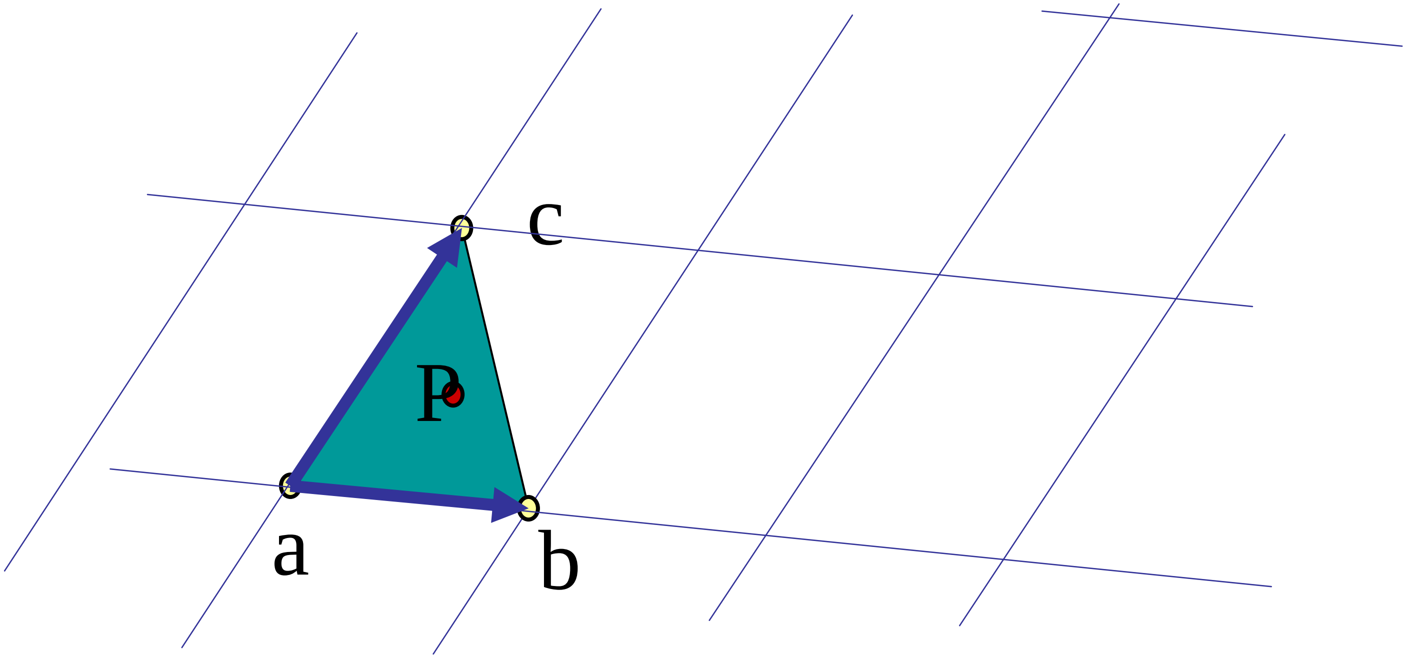
Simplify

- Since $\alpha + \beta + \gamma = 1$
  we can write $\alpha = 1 - \beta - \gamma$
- $P(\beta, \gamma) = (1 - \beta - \gamma)\, a + \beta b + \gamma c$



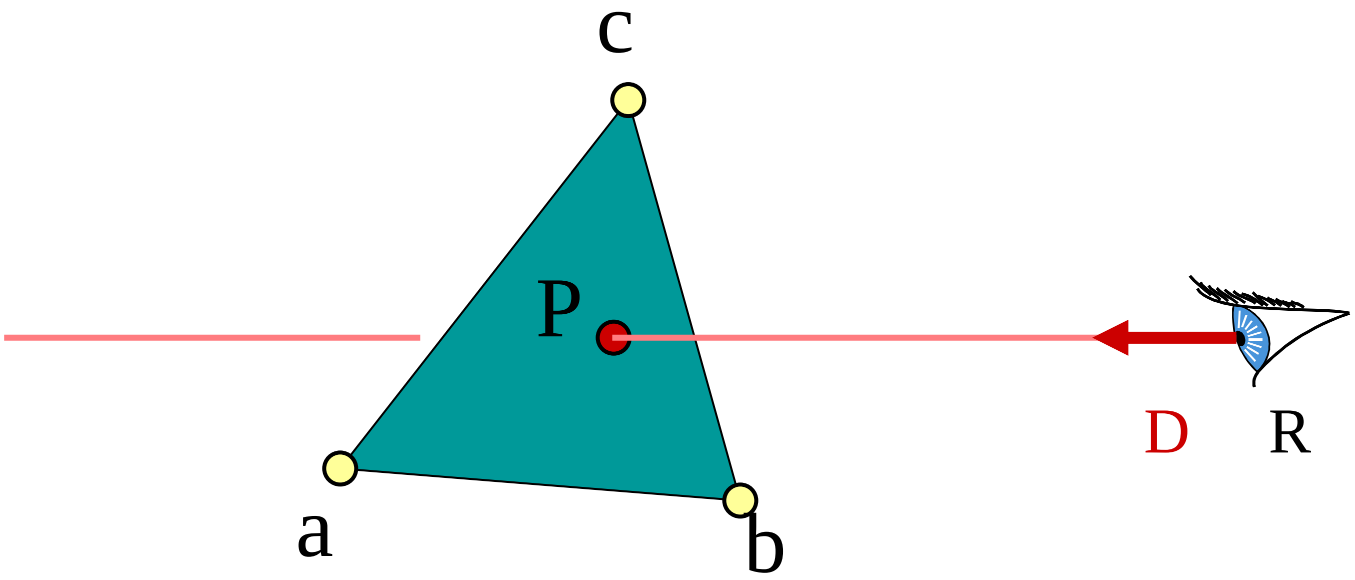
Simplify

- $P(\beta, \gamma)=(1-\beta-\gamma)\ a + \beta b + \gamma c$
- $P(\beta, \gamma)=a + \beta(b-a) + \gamma(c-a)$
- Non-orthogonal coordinate system of the plane



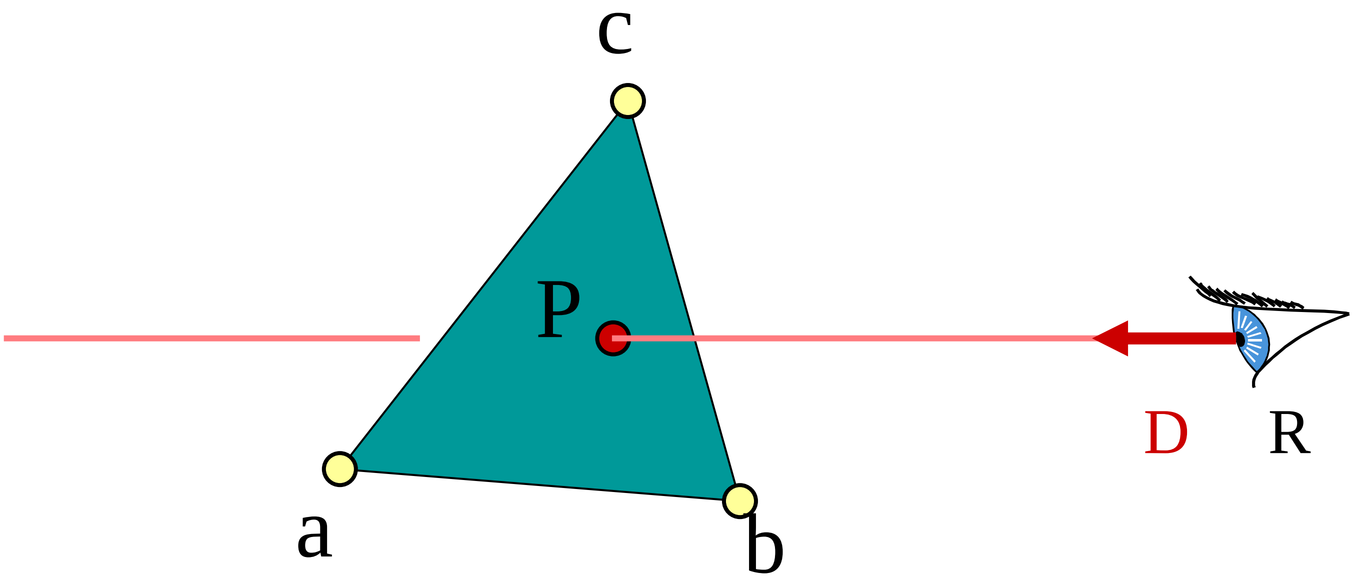
How do we use it for intersection?

- Insert ray equation into barycentric expression of triangle
- $P(t) = a + \beta\ (b-a) + \gamma\ (c-a)$
- Intersection if $\beta+\gamma<1$; $0<\beta$ and $0<\gamma$



Intersection

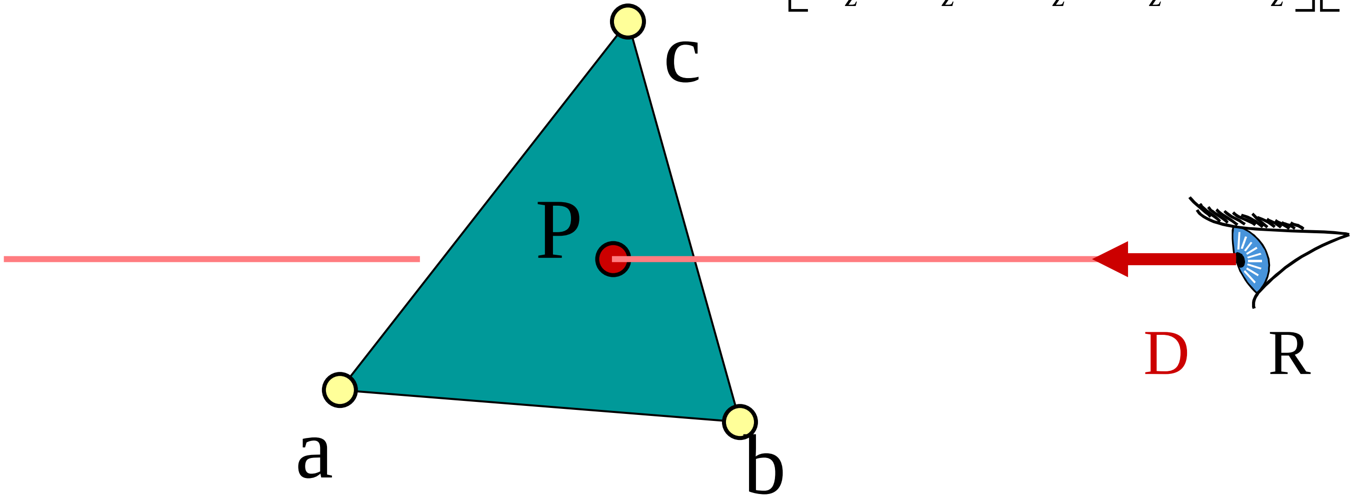
- $R_x + tD_x = a_x + \beta\ (b_x - a_x) + \gamma\ (c_x - a_x)$
- $R_y + tD_y = a_y + \beta\ (b_y - a_y) + \gamma\ (c_y - a_y)$
- $R_z + tD_z = a_z + \beta\ (b_z - a_z) + \gamma\ (c_z - a_z)$



Matrix form

- $R_x+tD_x= a_x+\beta\ (b_x-a_x)+\gamma\ (c_x-a_x)$
- $R_y+tD_y= a_y+\beta\ (b_y-a_y)+\gamma\ (c_y-a_y)$
- $R_z+tD_z= a_z+\beta\ (b_z-a_z)+\gamma\ (c_z-a_z)$

$$\begin{bmatrix} a_x - b_x & a_x - c_x & D_x \\ a_y - b_y & a_y - c_y & D_y \\ a_z - b_z & a_z - c_z & D_z \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x - R_x \\ a_y - R_y \\ a_z - R_z \end{bmatrix}$$

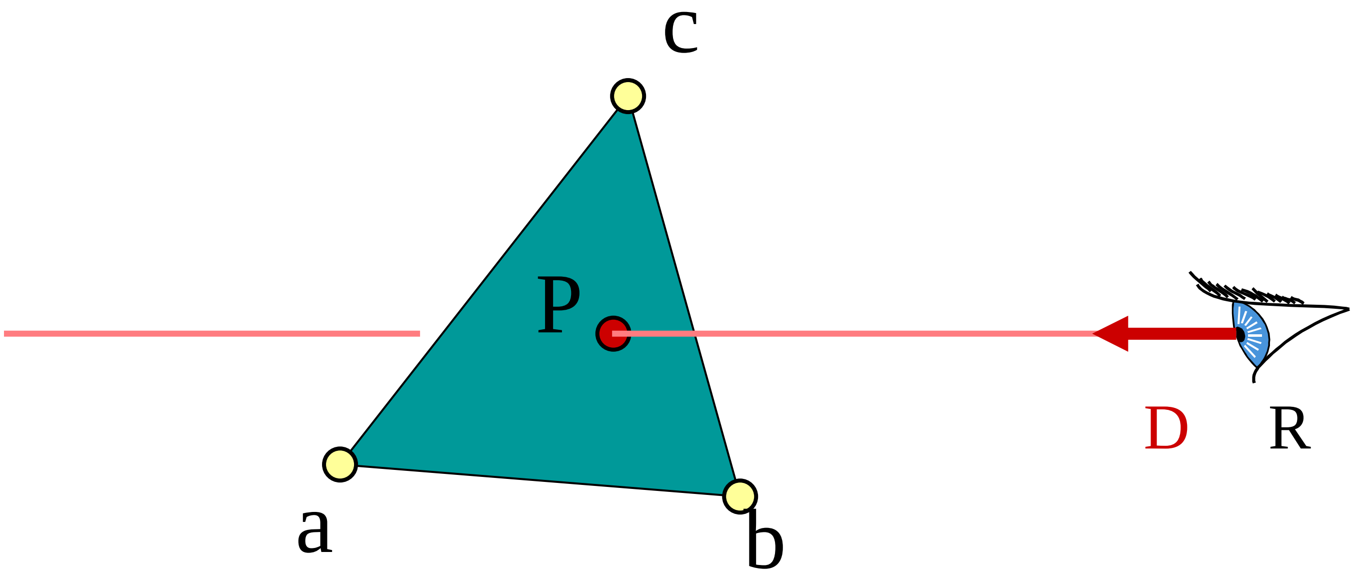


Cramer's rule

- | | denotes the determinant

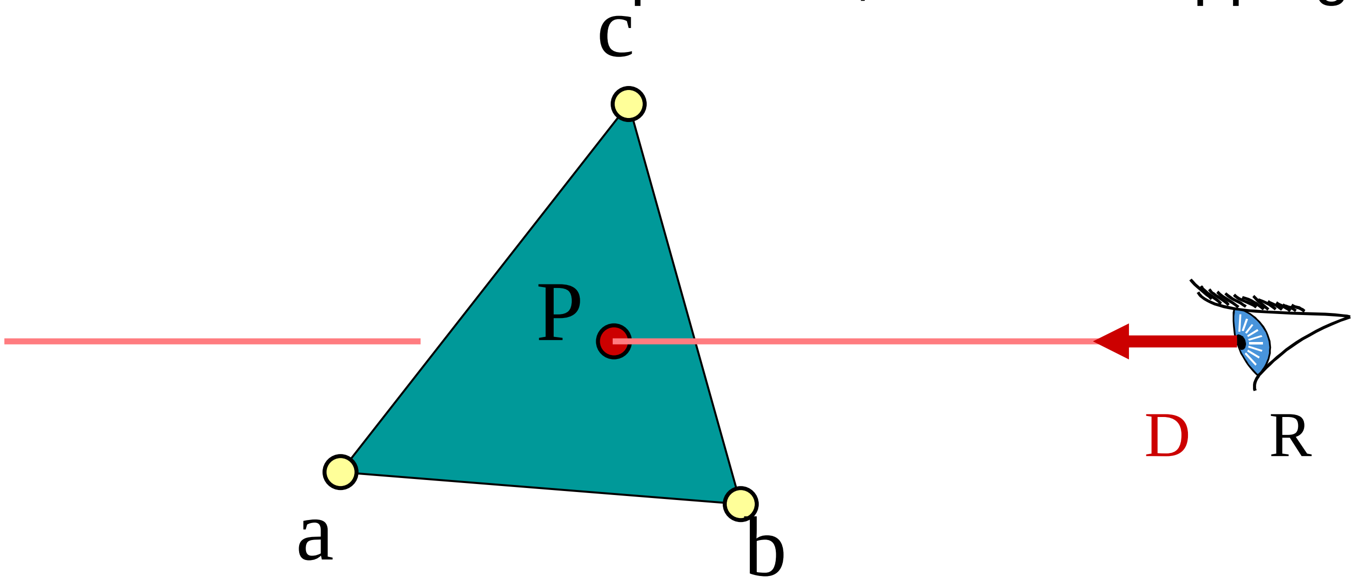
$$\beta = \frac{\begin{vmatrix} a_x - R_x & a_x - c_x & D_x \\ a_y - R_y & a_y - c_y & D_y \\ a_z - R_z & a_z - c_z & D_z \end{vmatrix}}{|A|} \qquad \gamma = \frac{\begin{vmatrix} a_x - b_x & a_x - R_x & D_x \\ a_y - b_y & a_y - R_y & D_y \\ a_z - b_z & a_z - R_z & D_z \end{vmatrix}}{|A|} \qquad t = \frac{\begin{vmatrix} a_x - b_x & a_x - c_x & a_x - R_x \\ a_y - b_y & a_y - c_y & a_y - R_y \\ a_z - b_z & a_z - c_z & a_z - R_z \end{vmatrix}}{|A|}$$

- Can be copied mechanically in the code



Advantage

- Efficient
- Store no plane equation
- Get the barycentric coordinates for free
  - Useful for interpolation, texture mapping



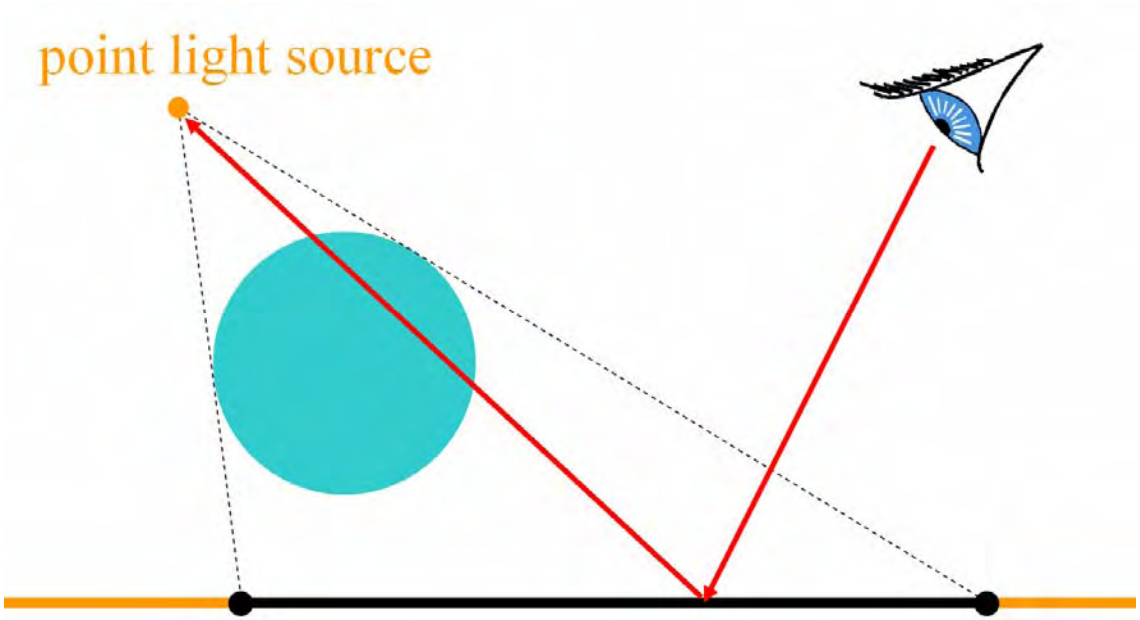
More Effects

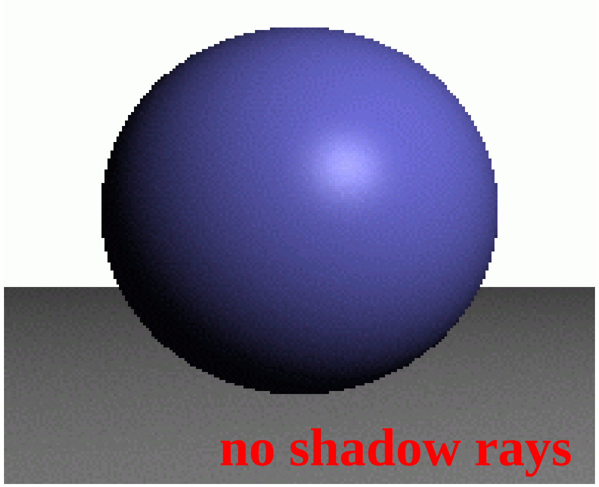
Extra rays needed for these effects:

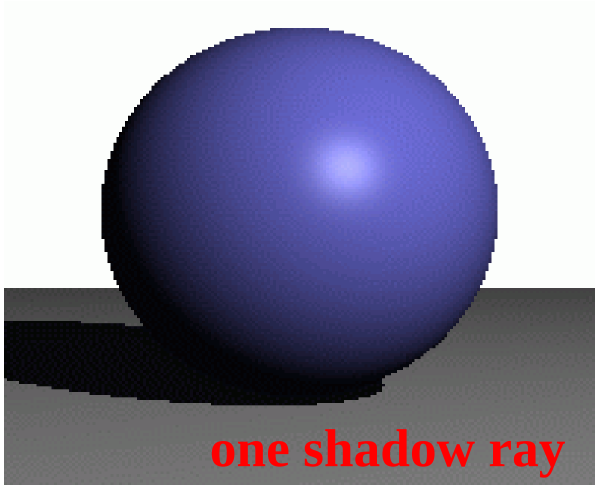
- Distribution Ray Tracing
  - – Soft shadows
  - – Anti-aliasing (getting rid of jaggies)
  - – Glossy reflection
  - – Motion blur
  - – Depth of field (focus)

Shadows

- one shadow ray per intersection per point light source

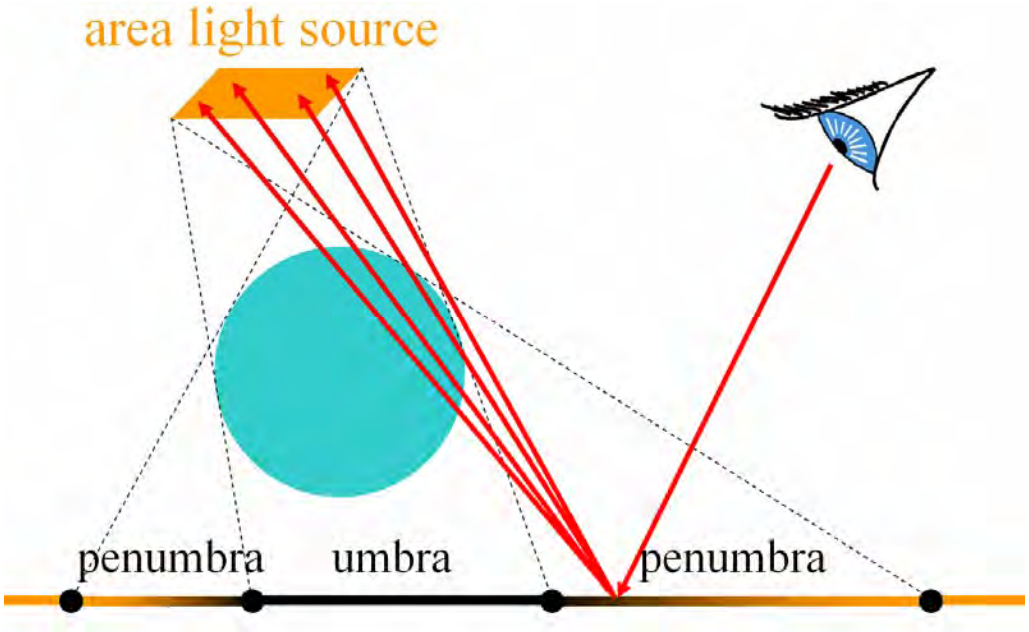


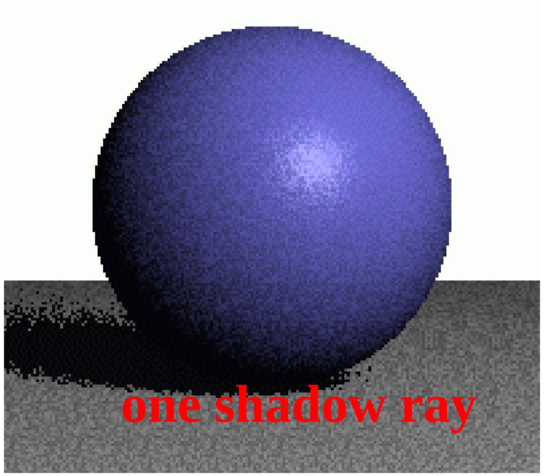


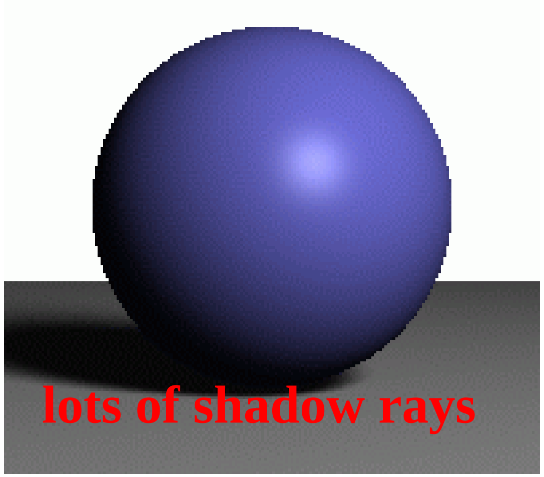


Soft Shadows

- multiple shadow rays to sample area light source





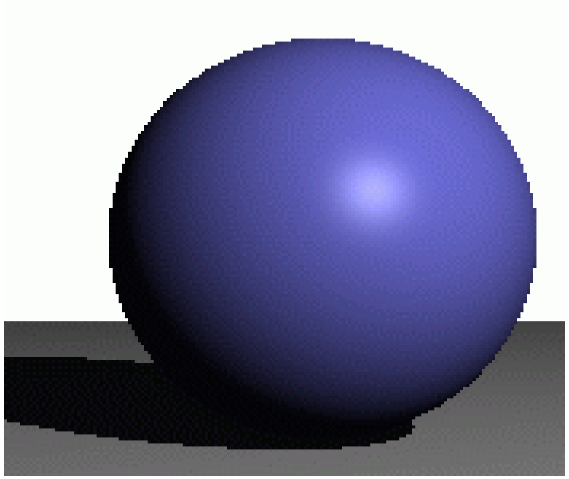


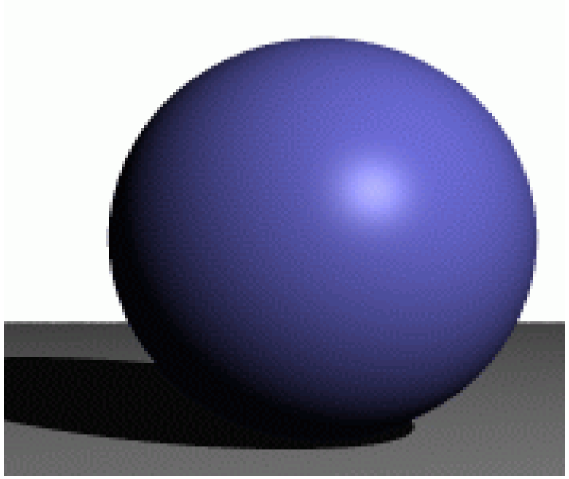
Antialiasing – Supersampling

jaggies **w/ antialiasing**

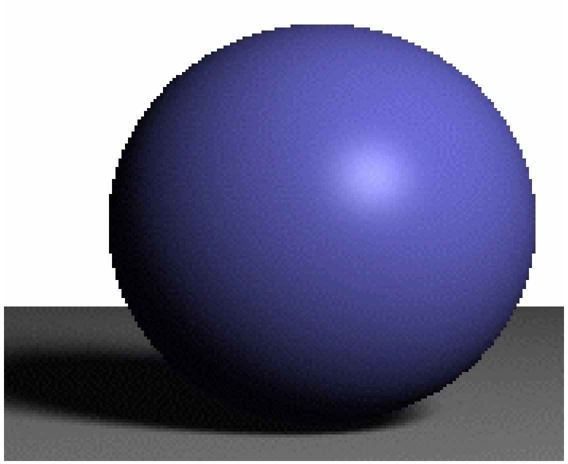
- multiple rays per pixel

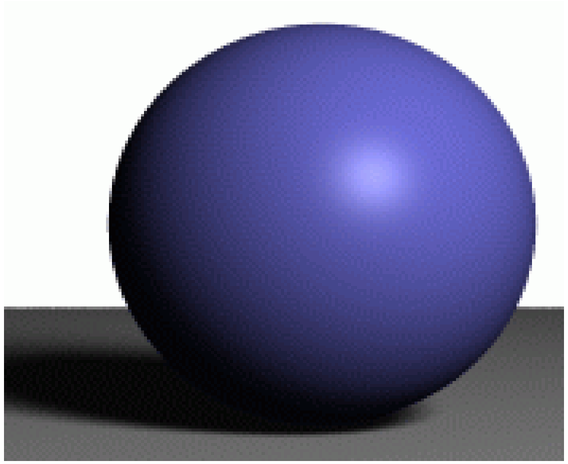
point light





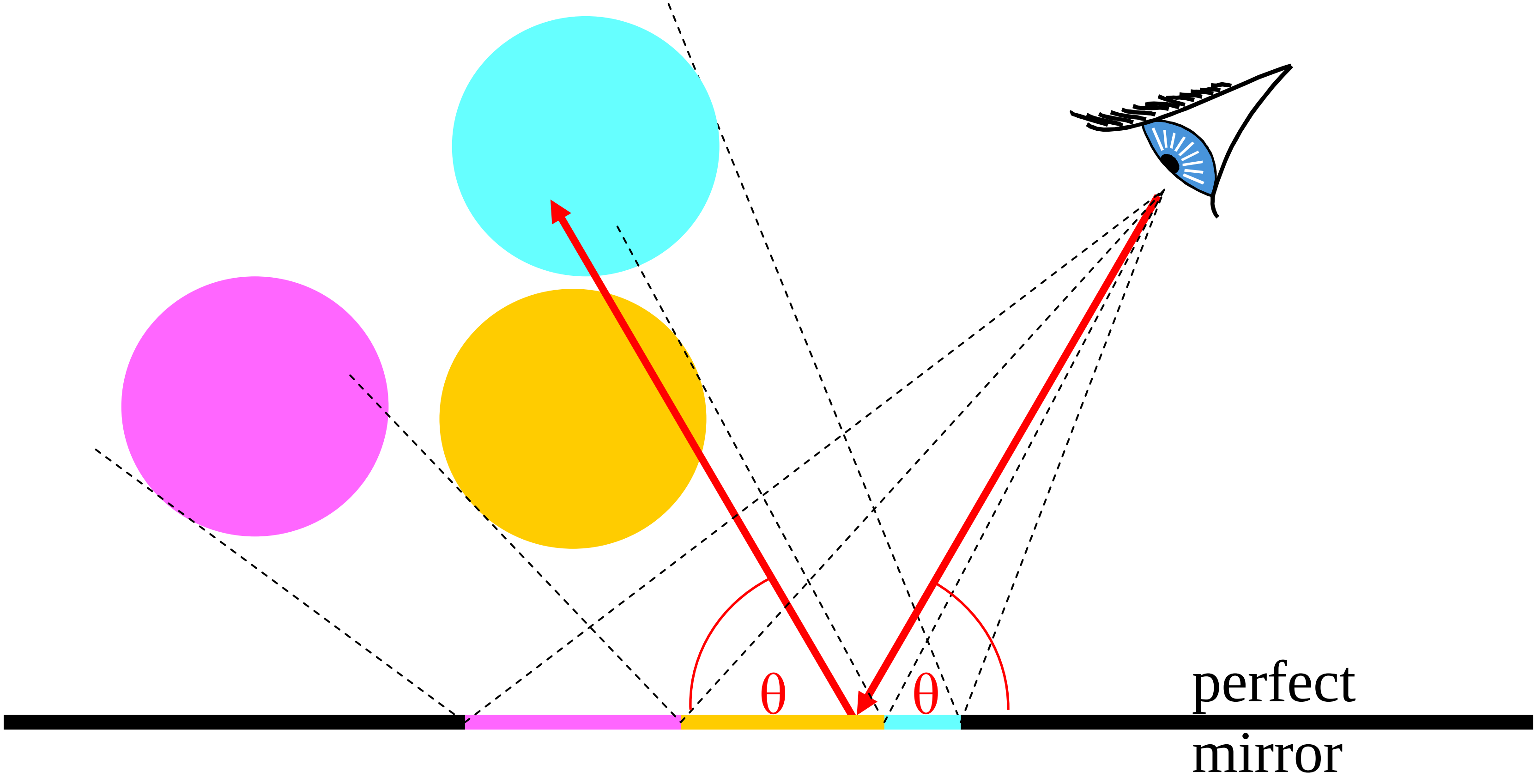
area light





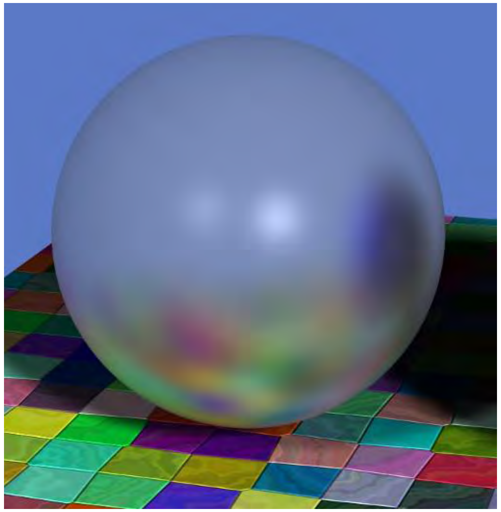
Reflection

- one reflection ray per intersection

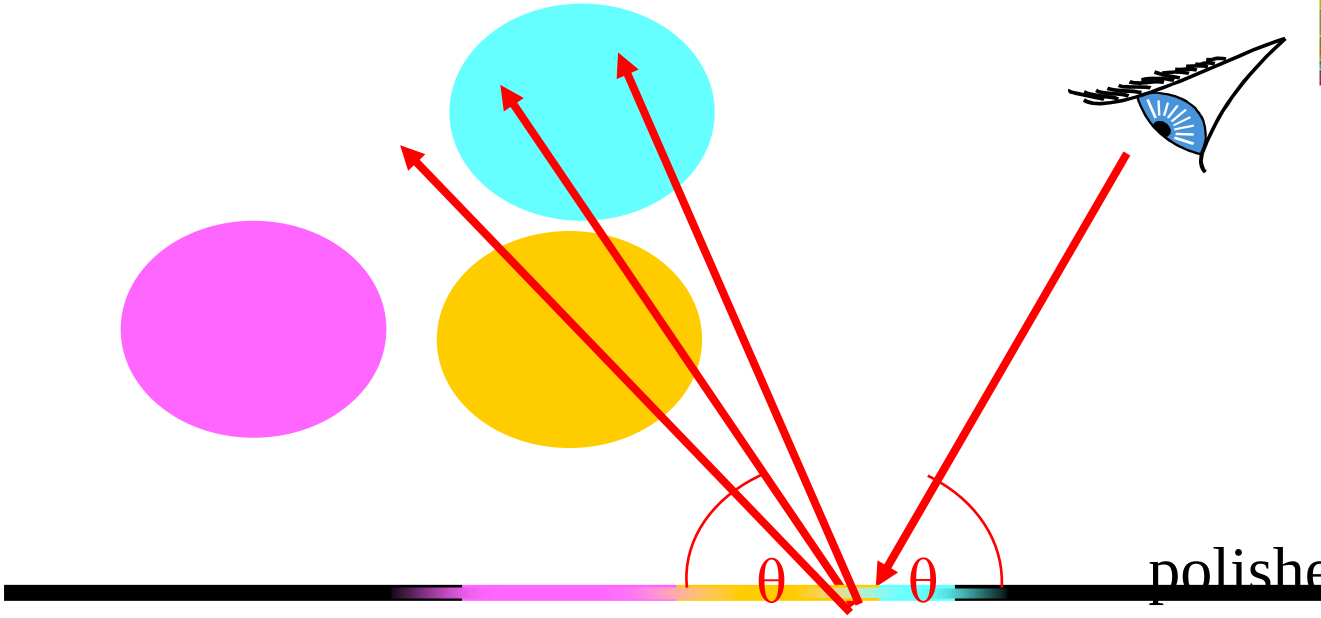


Glossy Reflection

- multiple reflection rays



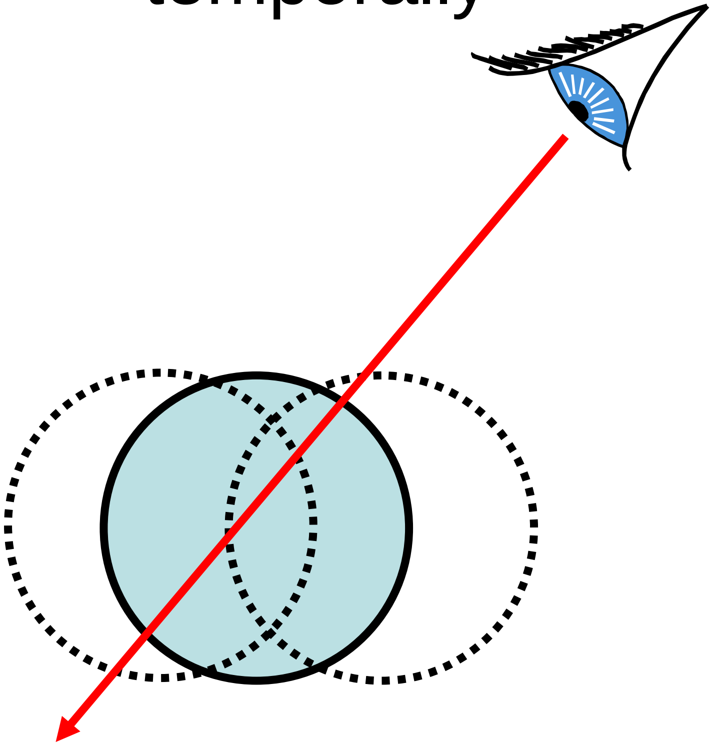
Justin Legakis

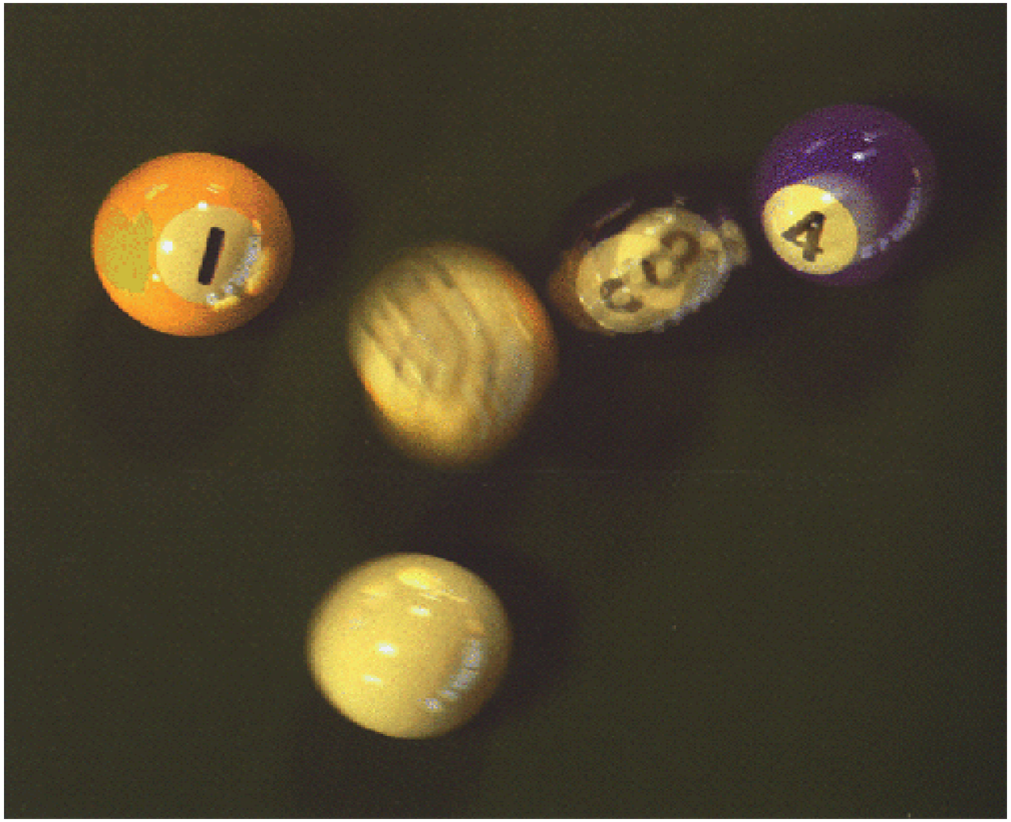


polished surface

Motion Blur

- Sample objects temporally

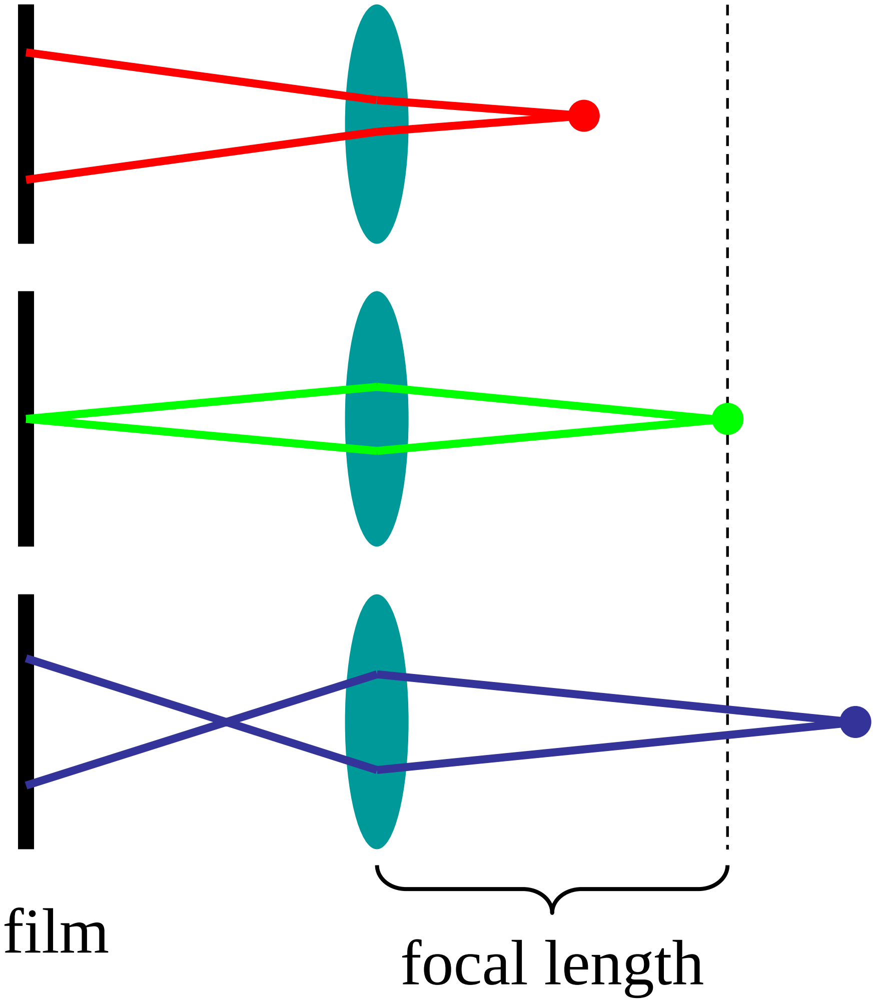


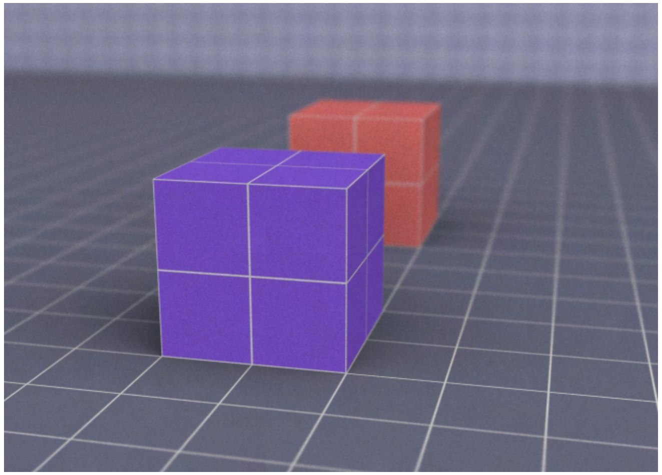


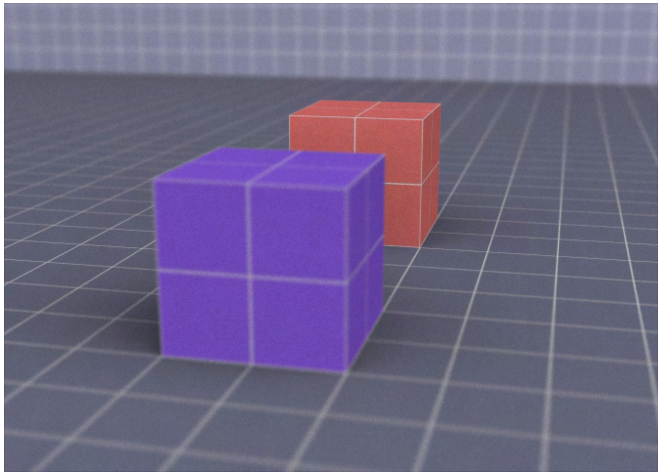
Rob Cook

Depth of Field

- multiple rays per pixel







Justin Legakis

Algorithm Analysis

- Ray casting
- Lots of primitives
- Recursive
- Distributed Ray Tracing Effects
  - Soft shadows
  - Anti-aliasing
  - Glossy reflection
  - Motion blur
  - Depth of field

cost ≤ height * width *
num primitives *
intersection cost *
num shadow rays *
supersampling *
num glossy rays *
num temporal samples *
max recursion depth *
. . .

can we reduce this?

Accelerating RT

- Reduce the number of pixels traced
  - Render cache
- Reduce the number of intersections
  - Spatial subdivision data structures

Reduce the number of pixels traced

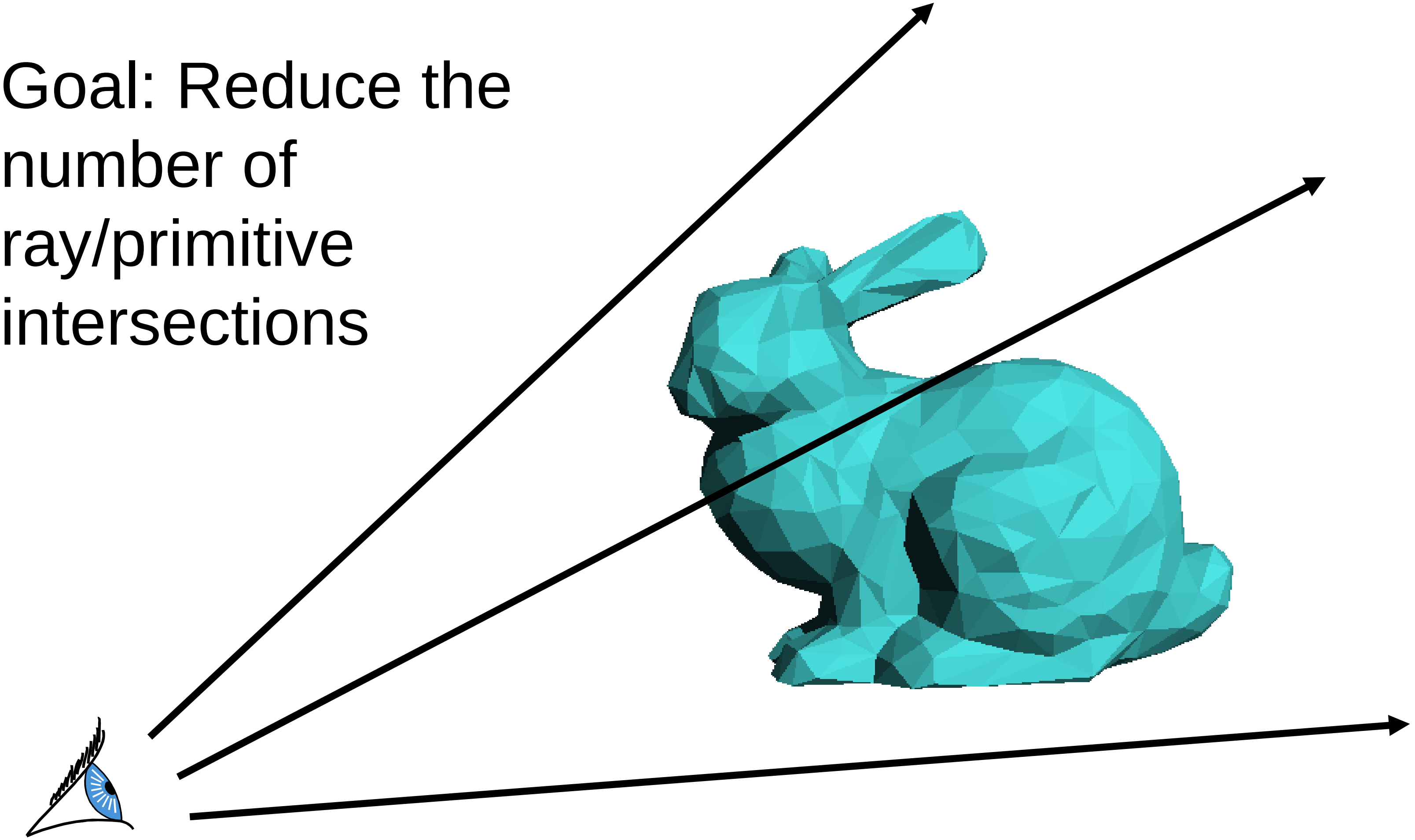
- Render Cache
  - EG Workshop on Rendering 99 (Walter et al.)
  - Only trace a small number of rays
  - Separate display loop and render loop
  - Interpolate a cache of reprojected points
- More details at publication page
- demo

Reduce the number of intersections

- Bounding Boxes
  - of each primitive
  - of groups
  - of transformed primitives
- Spatial Acceleration Data Structures
- Flattening the transformation hierarchy

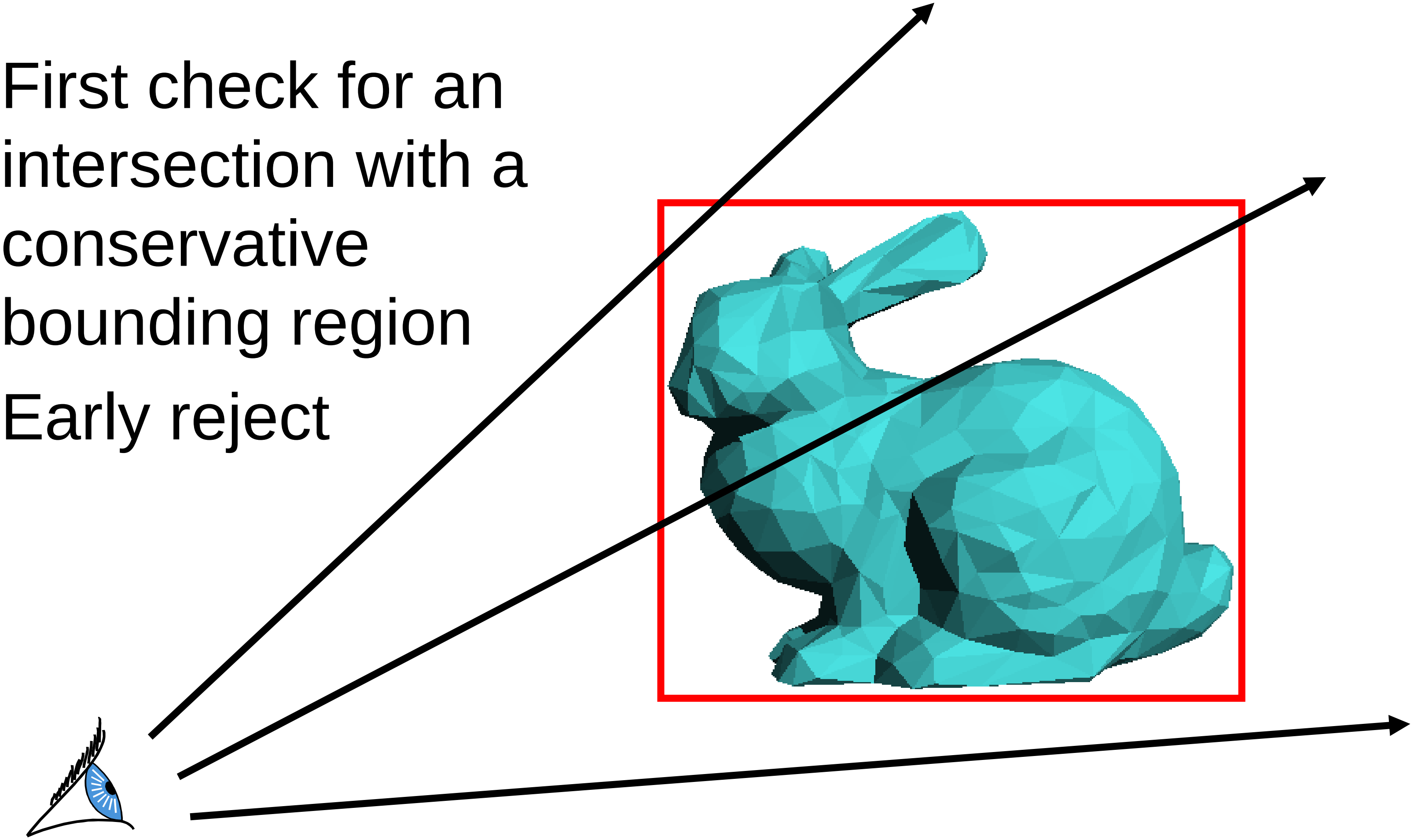
Acceleration of Ray Casting

- Goal: Reduce the number of ray/primitive intersections



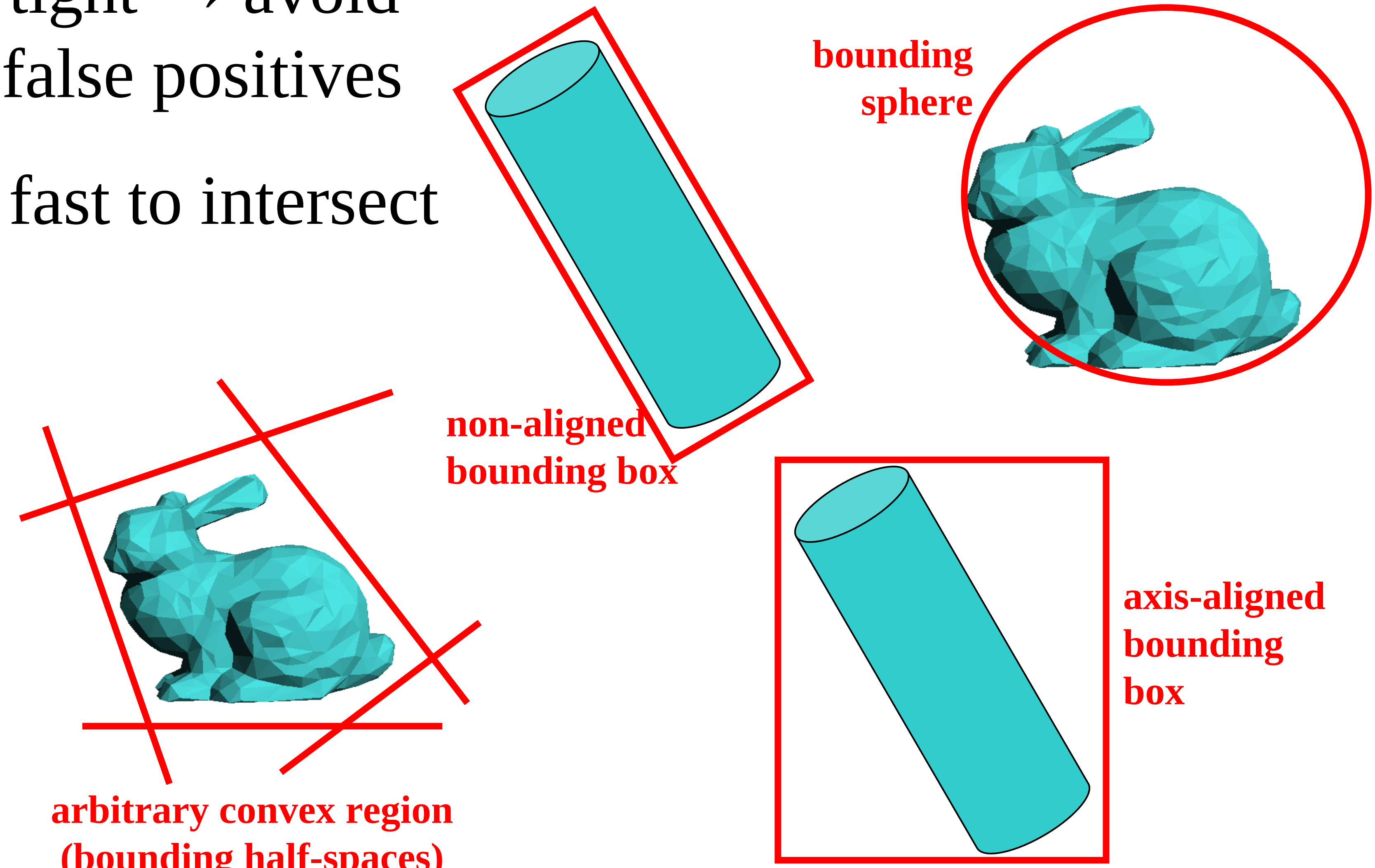
Conservative Bounding Region

- First check for an intersection with a conservative bounding region
- Early reject



Conservative Bounding Regions

- tight → avoid false positives
- fast to intersect



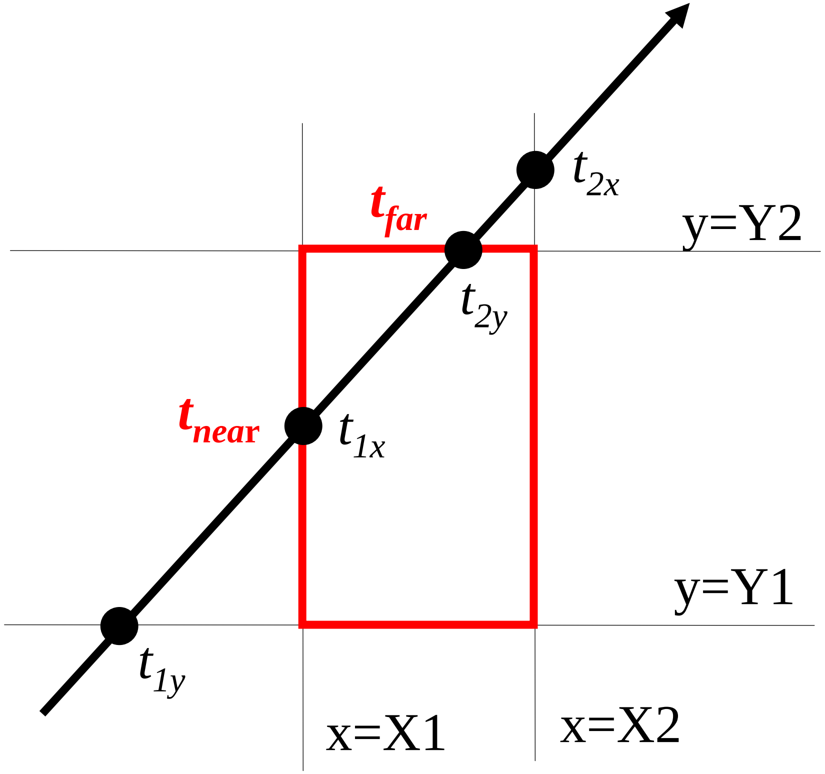
bounding sphere

non-aligned bounding box

axis-aligned bounding box

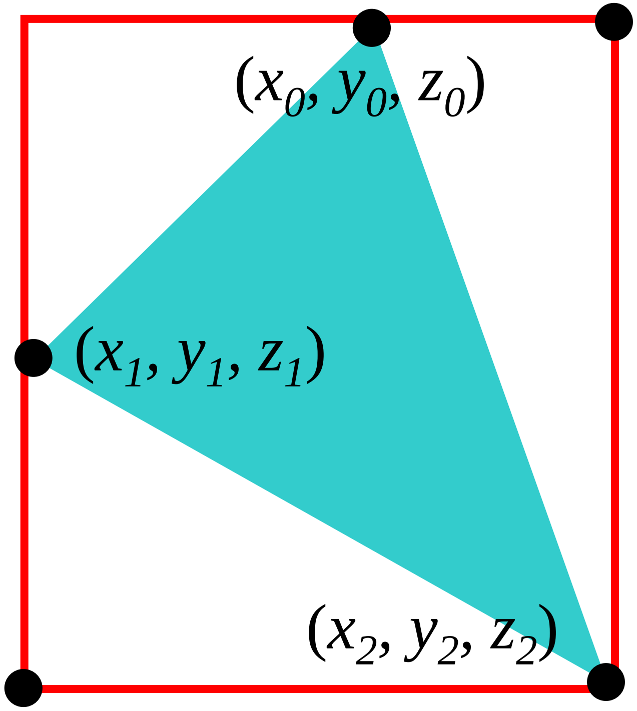
arbitrary convex region (bounding half-spaces)

Intersection with Axis-Aligned Box



- For all 3 axes, calculate the intersection distances t_1 and t_2
- $t_{near} = \max(t_{1x}, t_{1y}, t_{1z})$
  $t_{far} = \min(t_{2x}, t_{2y}, t_{2z})$
- If $t_{near} > t_{far}$, box is missed
- If $t_{far} < t_{min}$, box is behind
- If box survived tests, report intersection at t_{near}

Bounding Box of a Triangle



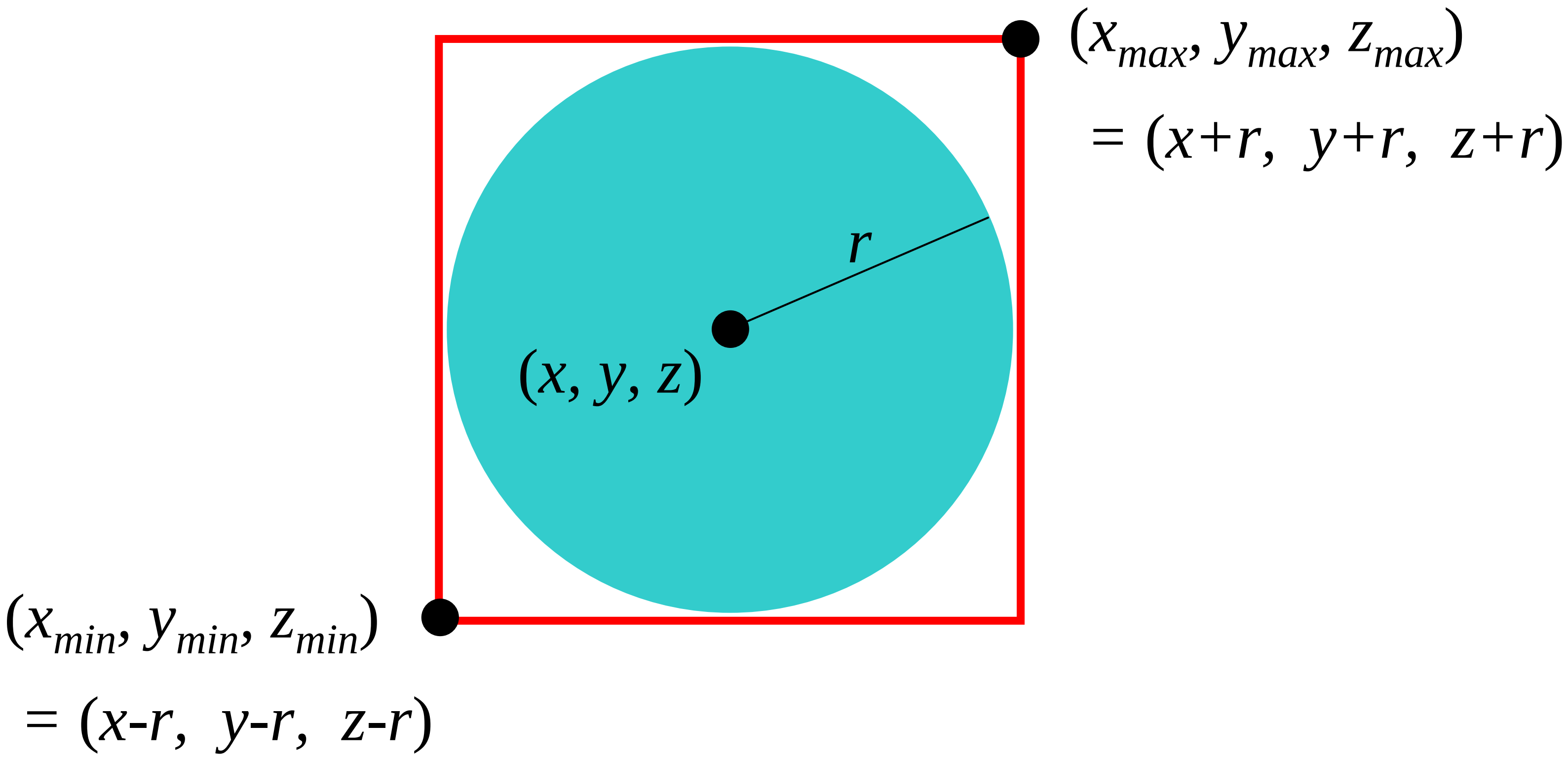
$(x_{max}, y_{max}, z_{max})$

$= (\max(x_0,x_1,x_2),$
$\max(y_0,y_1,y_2),$
$\max(z_0,z_1,z_2))$

$(x_{min}, y_{min}, z_{min})$

$= (\min(x_0,x_1,x_2),$
$\min(y_0,y_1,y_2),$
$\min(z_0,z_1,z_2))$

Bounding Box of a Sphere



$(x_{max}, y_{max}, z_{max})$

$= (x+r,\ \ y+r,\ \ z+r)$

r

(x, y, z)

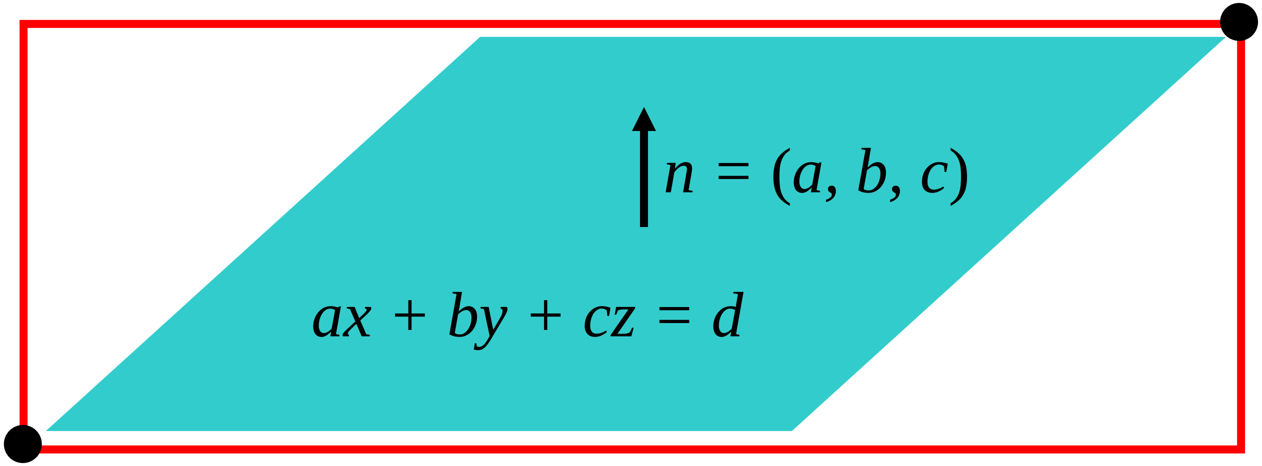
$(x_{min}, y_{min}, z_{min})$

$= (x\text{-}r,\ \ y\text{-}r,\ \ z\text{-}r)$

Bounding Box of a Plane

$(x_{max}, y_{max}, z_{max})$

$= (+\infty, +\infty, +\infty)$*



$n = (a, b, c)$

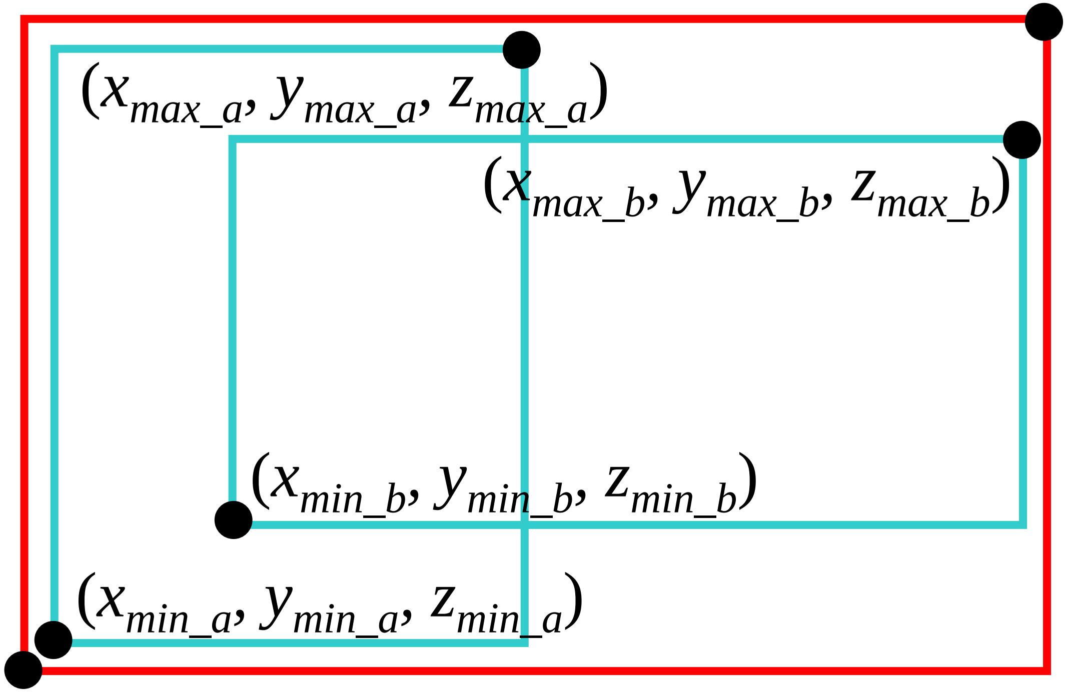
$ax + by + cz = d$

$(x_{min}, y_{min}, z_{min})$

$= (-\infty, -\infty, -\infty)$*

** unless n is exactly perpendicular to an axis*

Bounding Box of a Group



$(x_{max\_a}, y_{max\_a}, z_{max\_a})$

$(x_{max\_b}, y_{max\_b}, z_{max\_b})$

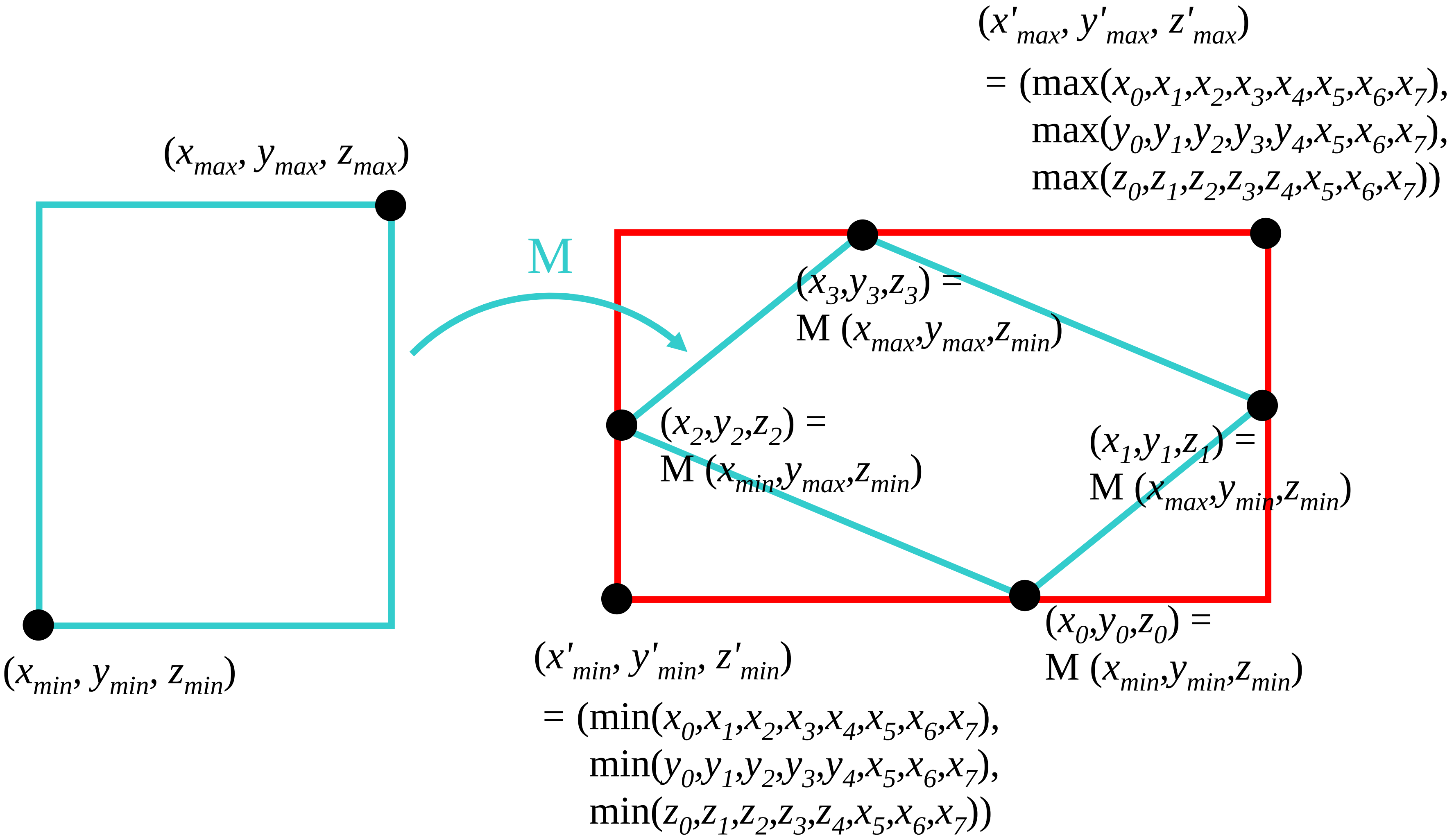
$(x_{min\_b}, y_{min\_b}, z_{min\_b})$

$(x_{min\_a}, y_{min\_a}, z_{min\_a})$

$$(x_{max}, y_{max}, z_{max}) = (\max(x_{max\_a},x_{max\_b}), \max(y_{max\_a},y_{max\_b}), \max(z_{max\_a},z_{max\_b}))$$

$$(x_{min}, y_{min}, z_{min}) = (\min(x_{min\_a},x_{min\_b}), \min(y_{min\_a},y_{min\_b}), \min(z_{min\_a},z_{min\_b}))$$

Bounding Box of a Transform



$(x_{max}, y_{max}, z_{max})$

$(x_{min}, y_{min}, z_{min})$

M

$(x'_{max}, y'_{max}, z'_{max})$

$= (\max(x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7),$
$\max(y_0,y_1,y_2,y_3,y_4,x_5,x_6,x_7),$
$\max(z_0,z_1,z_2,z_3,z_4,x_5,x_6,x_7))$

$(x_3,y_3,z_3) =$
$M\ (x_{max},y_{max},z_{min})$

$(x_2,y_2,z_2) =$
$M\ (x_{min},y_{max},z_{min})$

$(x_1,y_1,z_1) =$
$M\ (x_{max},y_{min},z_{min})$

$(x_0,y_0,z_0) =$
$M\ (x_{min},y_{min},z_{min})$

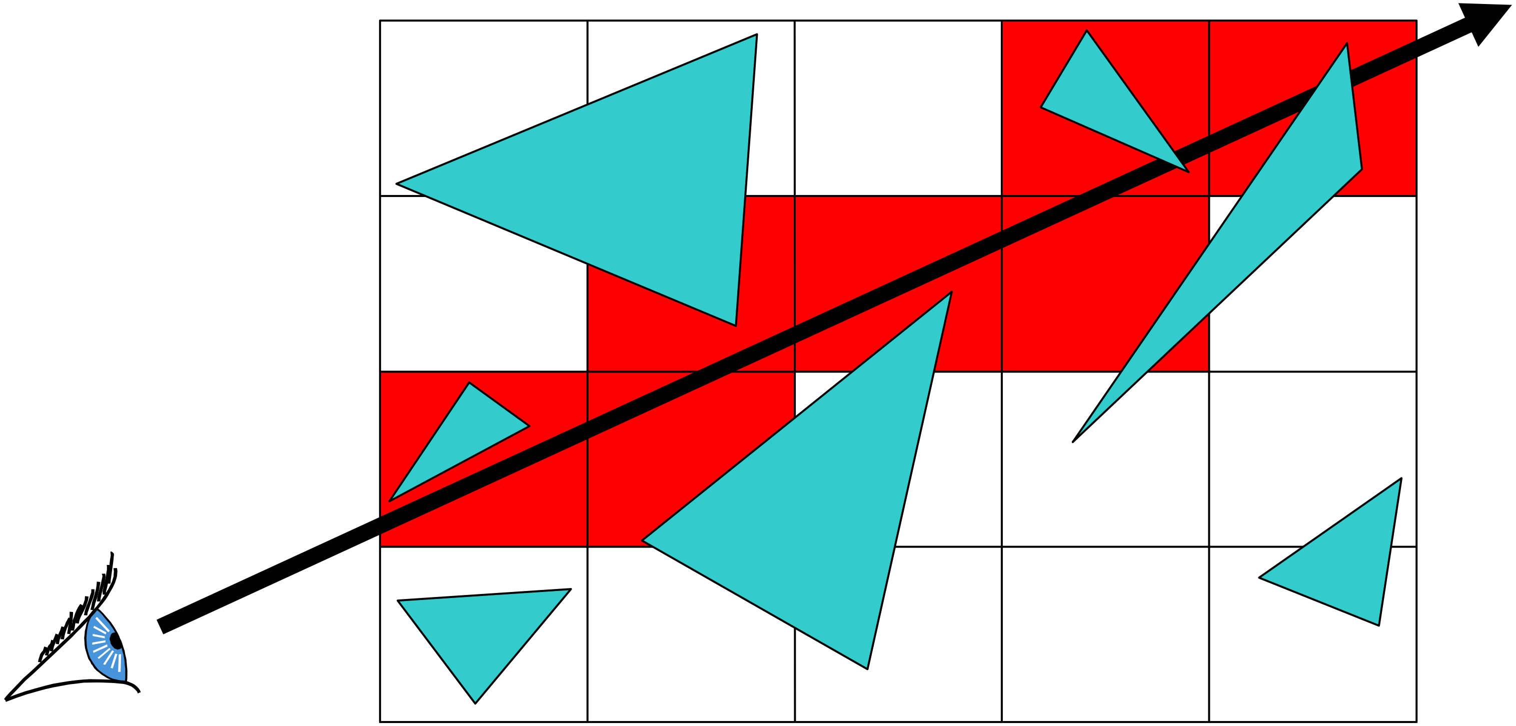
$(x'_{min}, y'_{min}, z'_{min})$

$= (\min(x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7),$
$\min(y_0,y_1,y_2,y_3,y_4,x_5,x_6,x_7),$
$\min(z_0,z_1,z_2,z_3,z_4,x_5,x_6,x_7))$

Reduce the number of intersections

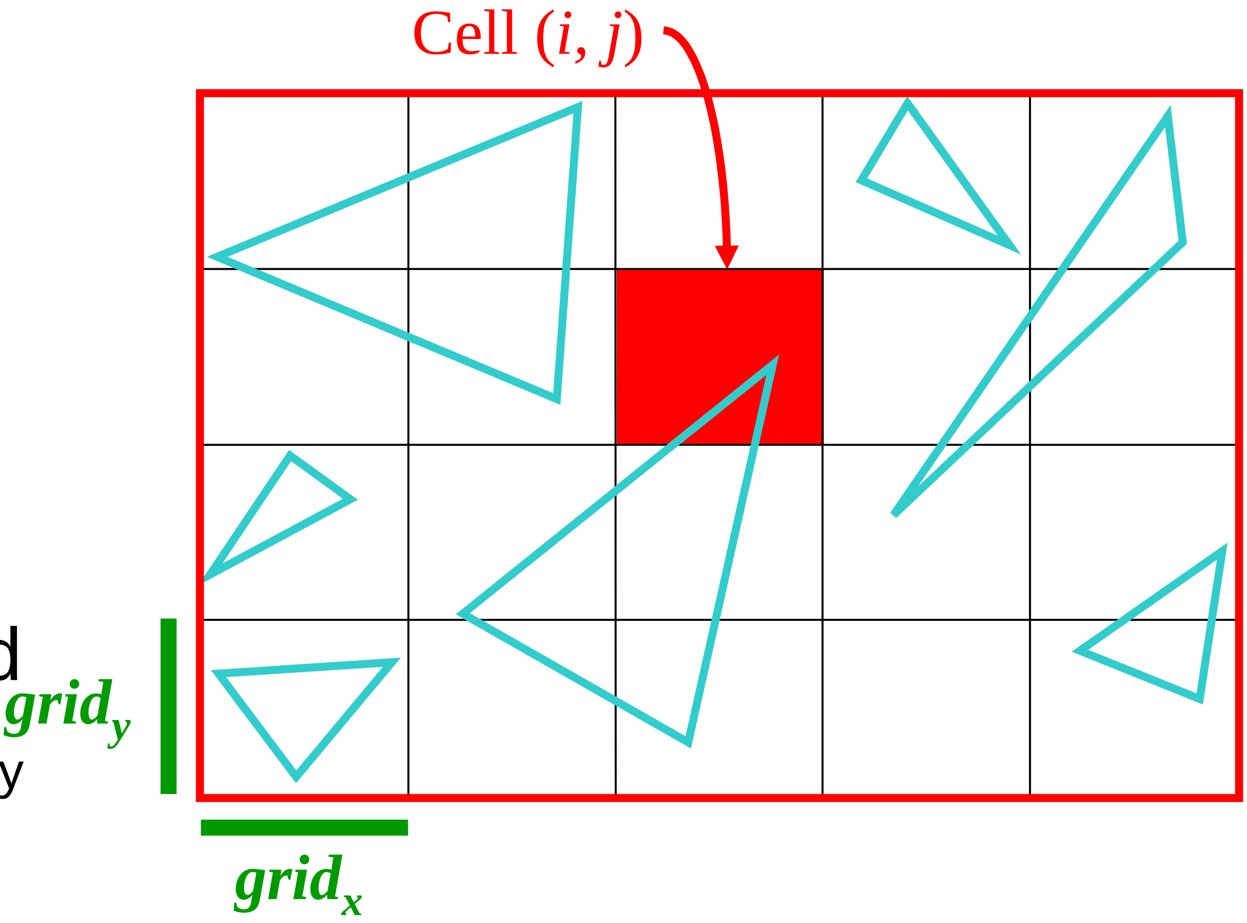
- Bounding Boxes
- Spatial Acceleration Data Structures
  - Regular Grid
  - Adaptive Grids
  - Hierarchical Bounding Volumes
- Flattening the transformation hierarchy

Regular Grid



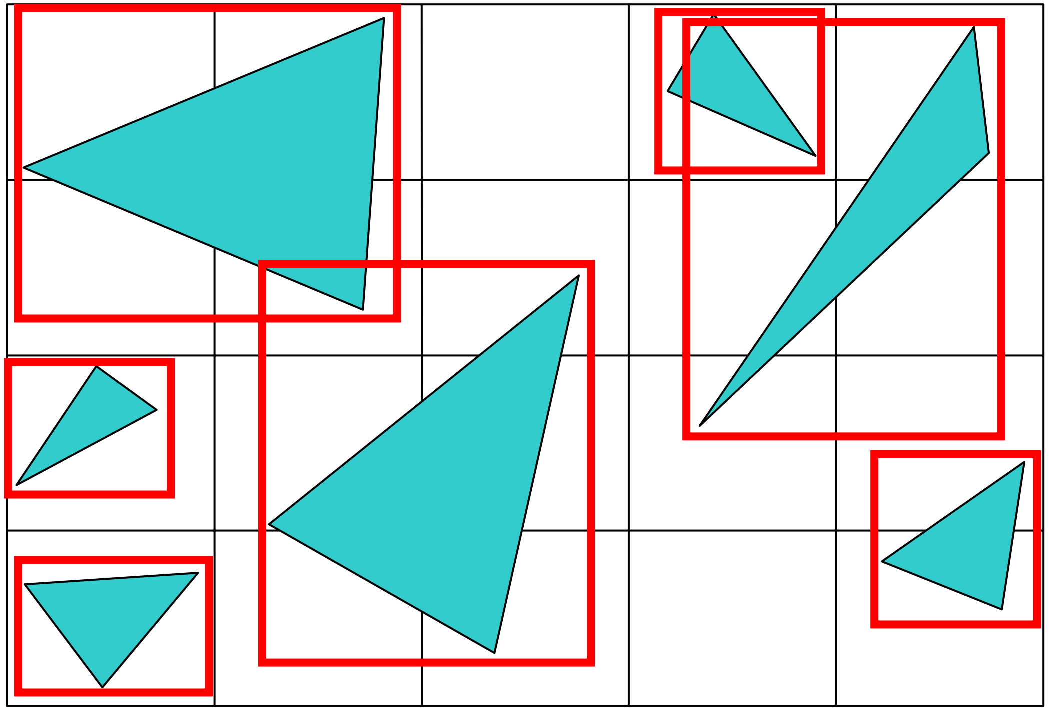
Create grid

- Find bounding box of scene
- Choose grid spacing
- grid_x need not = grid_y



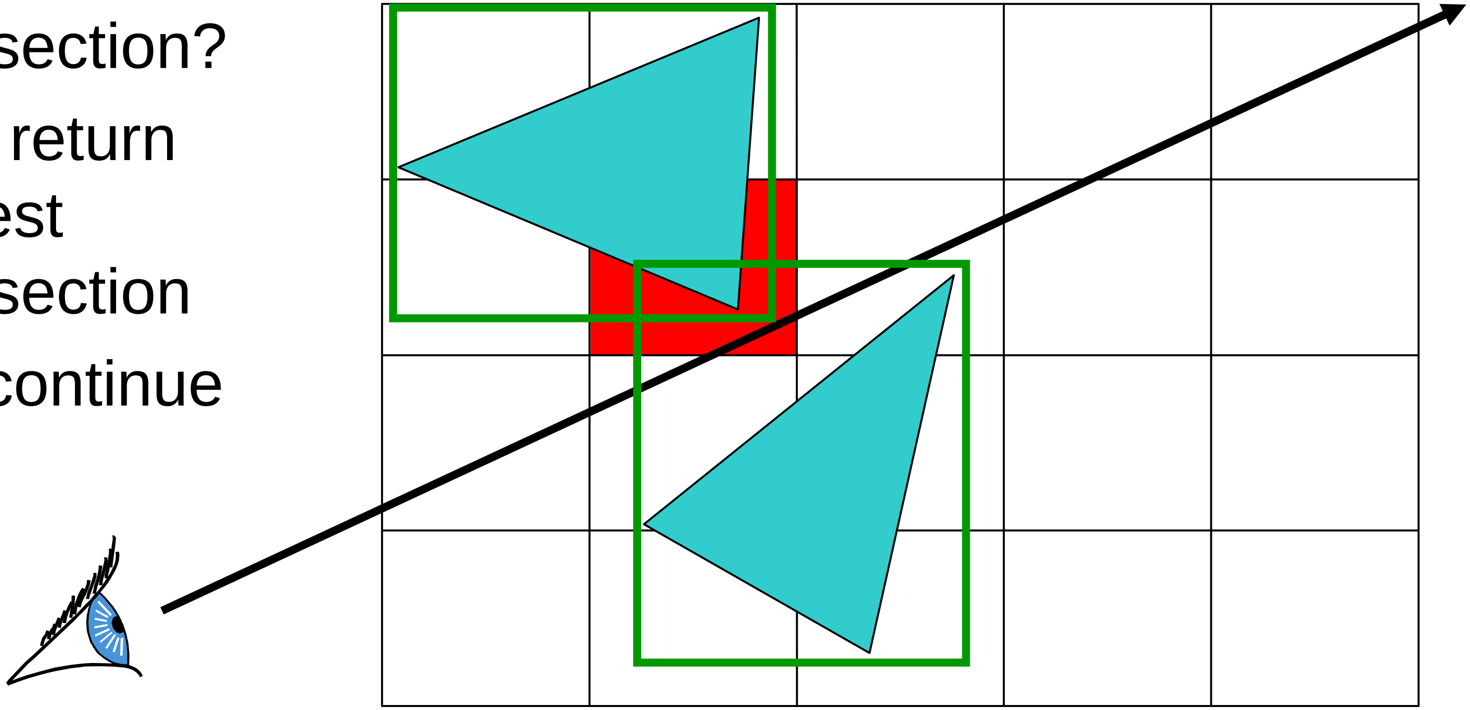
Insert primitives into grid

- Primitives that overlap multiple cells?
- Insert into multiple cells (use pointers)



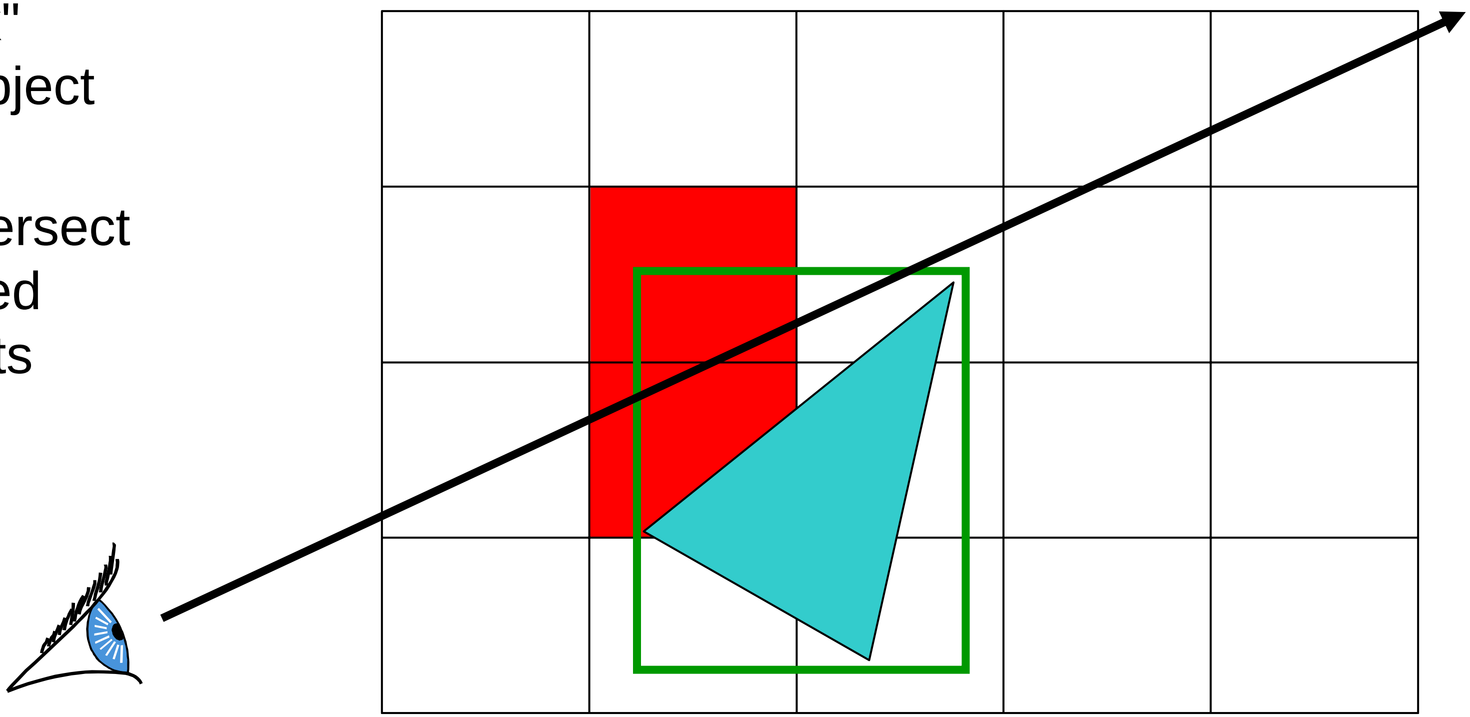
For each cell along a ray

- Does the cell contain an intersection?
- Yes: return closest intersection
- No: continue



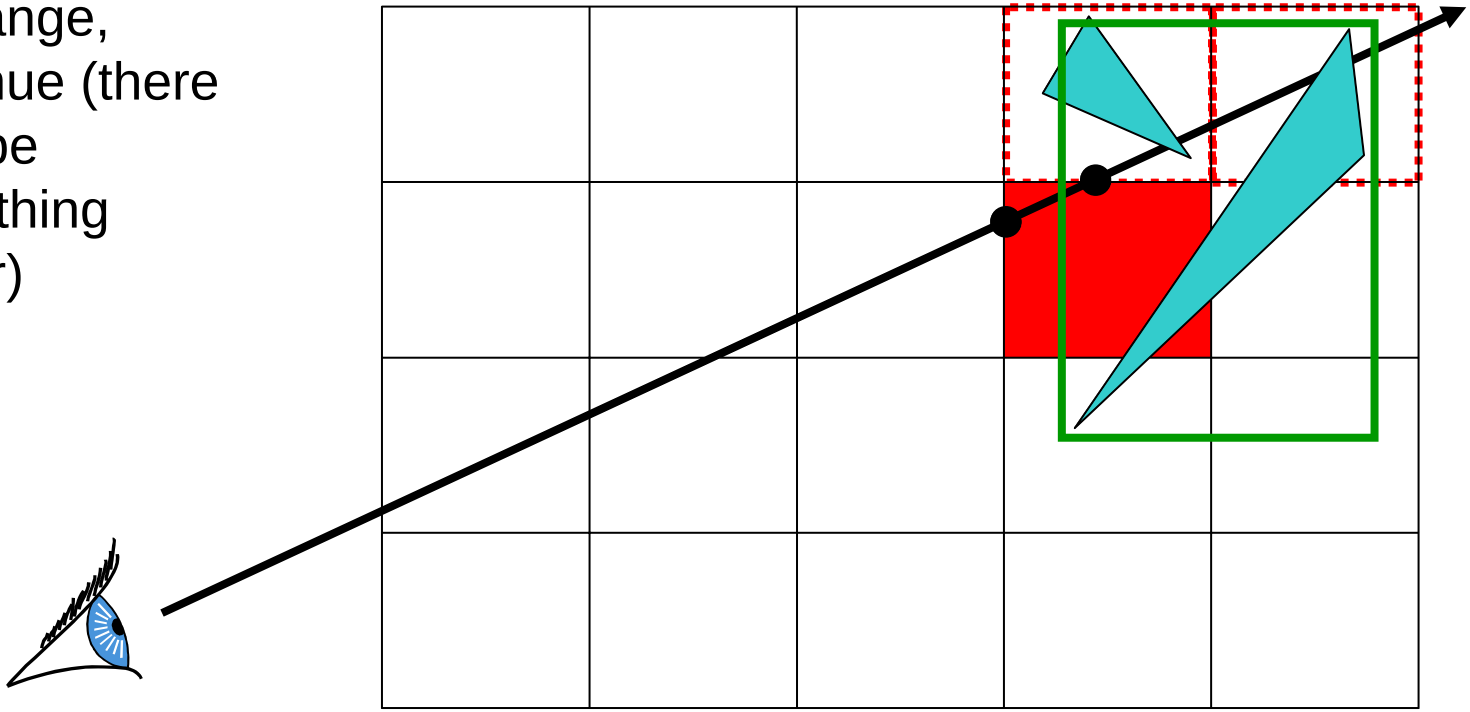
Preventing repeated computation

- Perform the computation once, "mark" the object
- Don't re-intersect marked objects



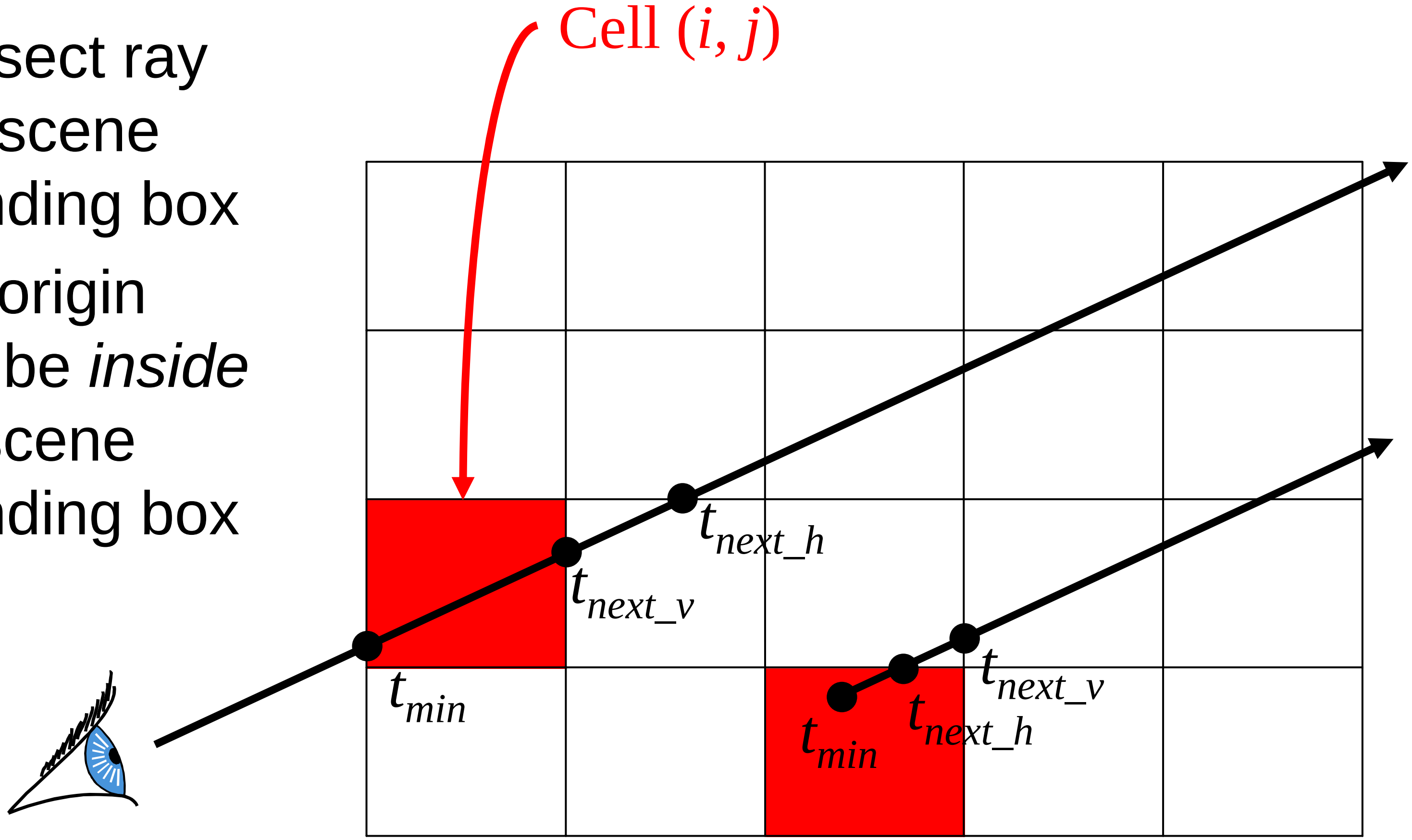
Don't return distant intersections

- If intersection t is not within the cell range, continue (there may be something closer)



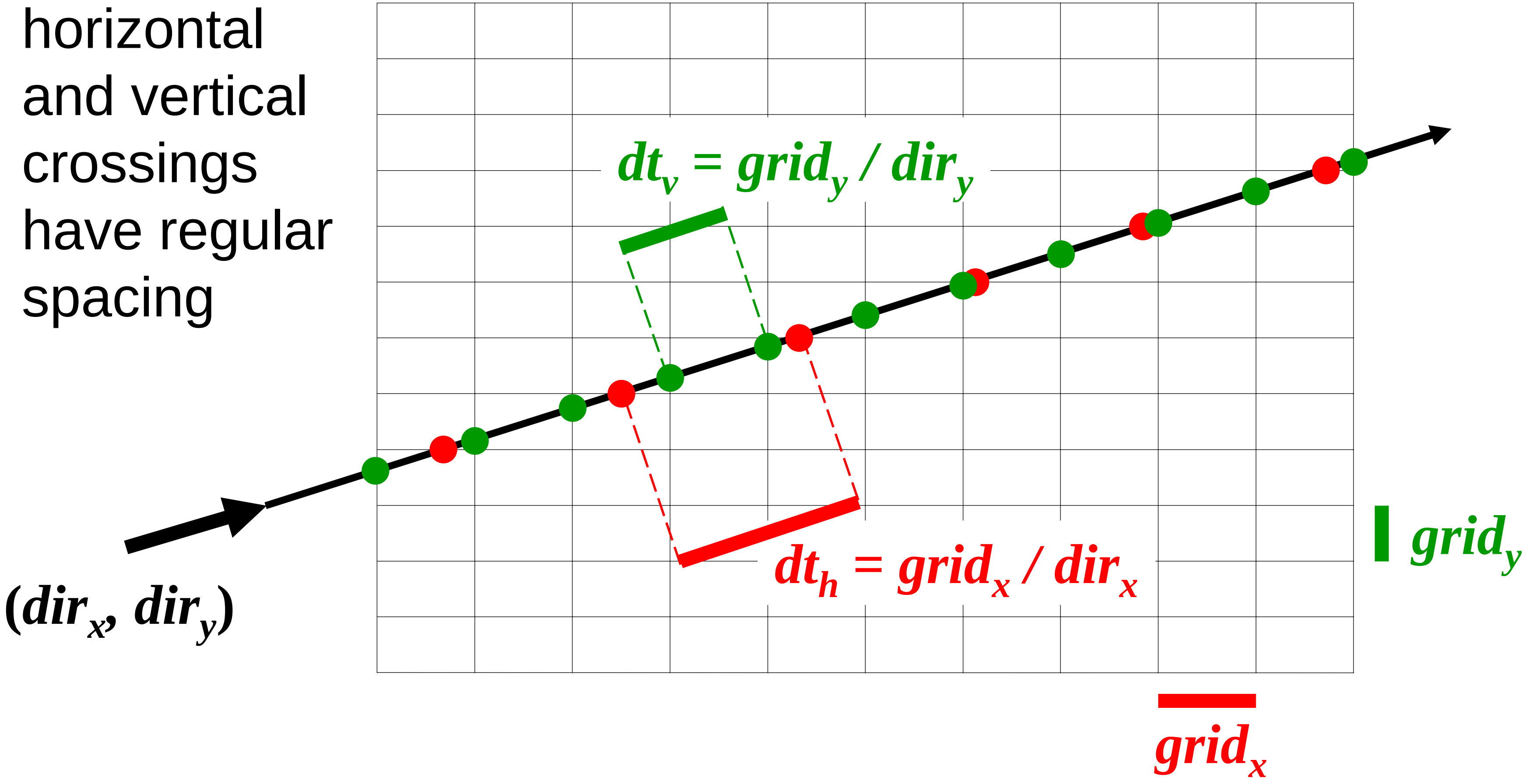
Where do we start?

- Intersect ray with scene bounding box
- Ray origin may be *inside* the scene bounding box



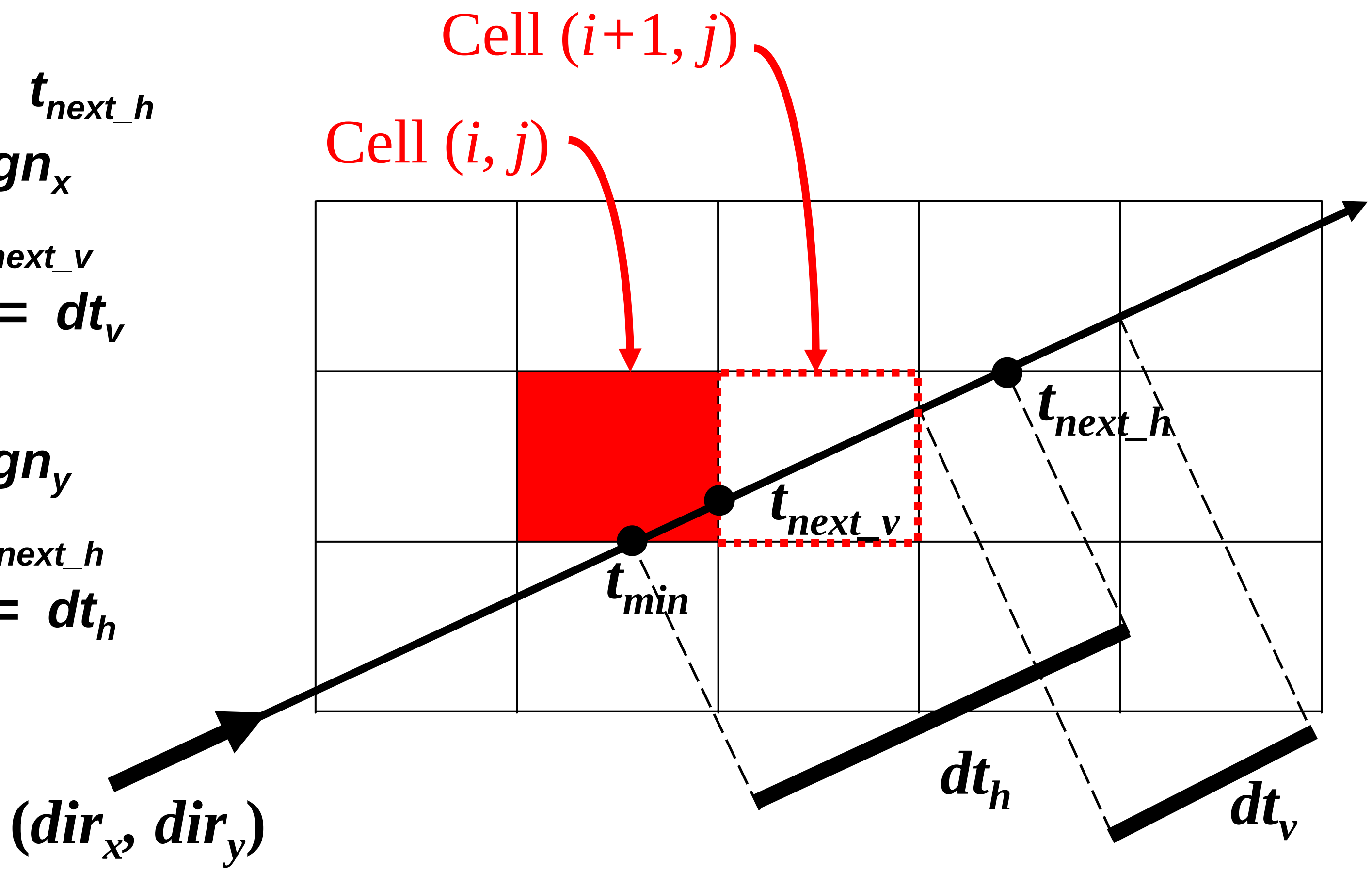
Is there a pattern to cell crossings?

- Yes, the horizontal and vertical crossings have regular spacing



What's the next cell?

```
if  $t_{next\_v} < t_{next\_h}$
    $i$ += $sign_x$
    $t_{min} = t_{next\_v}$
    $t_{next\_v}$ += $dt_v$
else
    $j$ += $sign_y$
    $t_{min} = t_{next\_h}$
    $t_{next\_h}$ += $dt_h$
```

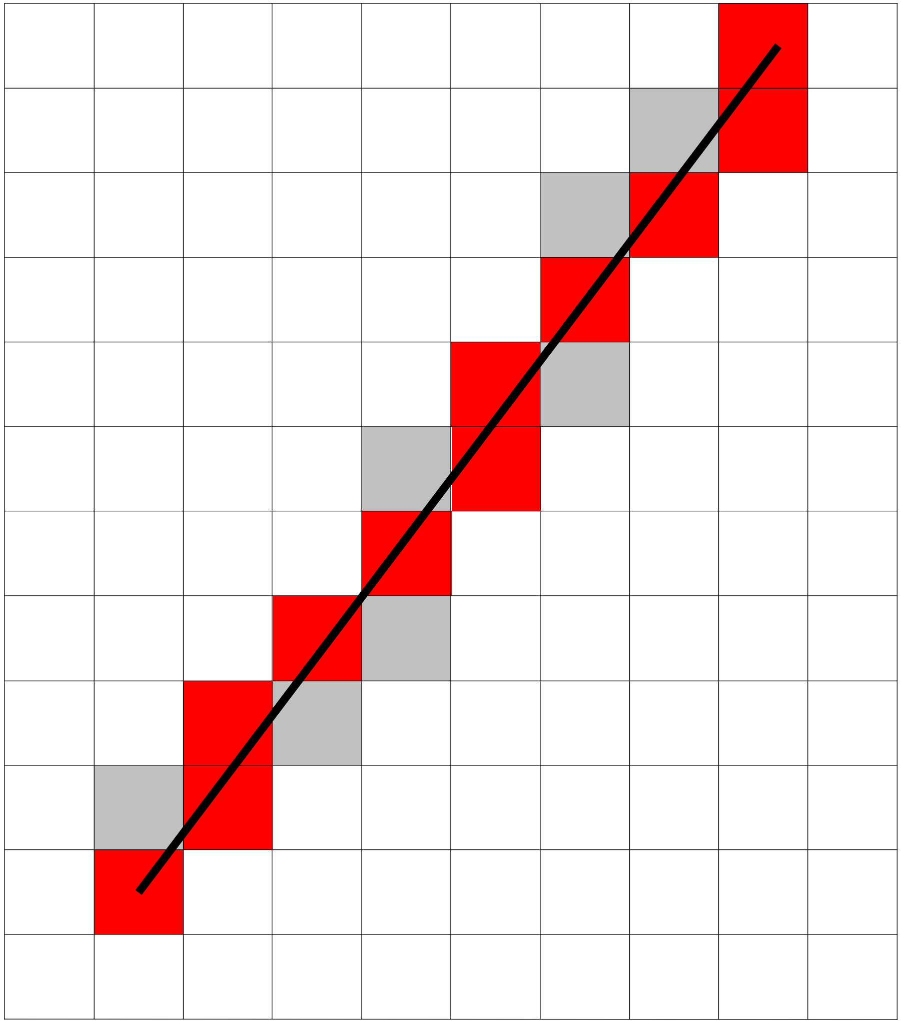


$\mathit{if}\ (dir_x > 0)\ sign_x = 1\ \mathit{else}\ sign_x = -1$

$\mathit{if}\ (dir_y > 0)\ sign_y = 1\ \mathit{else}\ sign_y = -1$

What's the next cell?

- 3DDDA – Three Dimensional Digital Difference Analyzer



Pseudo-code

```
create grid
insert primitives into grid
for each ray r
  find initial cell c(i,j), $t_{min}$, $t_{next\_v}$ & $t_{next\_h}$
  compute $dt_v$, $dt_h$, $sign_x$ and $sign_y$
  while c != NULL
    for each primitive p in c
      intersect r with p
      if intersection in range found
        return
    c = find next cell
```

Regular Grid Discussion

- Advantages?
  - easy to construct
  - easy to traverse

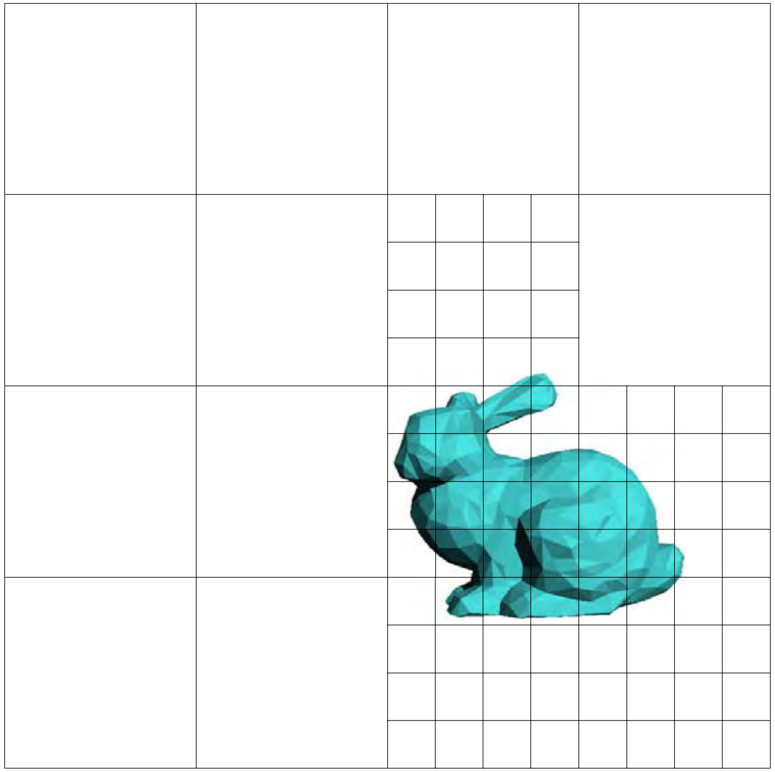
- Disadvantages?
  - may be only sparsely filled
  - geometry may still be clumped

Reduce the number of intersections

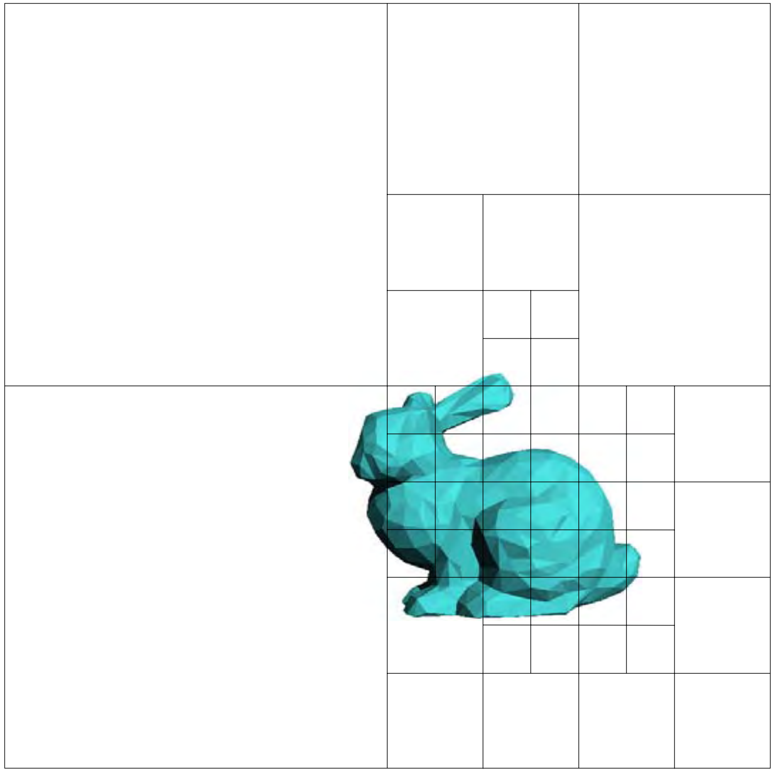
- Bounding Boxes
- Spatial Acceleration Data Structures
  - Regular Grid
  - Adaptive Grids
  - Hierarchical Bounding Volumes
- Flattening the transformation hierarchy

Adaptive Grids

- Subdivide until each cell contains no more than *n* elements, or maximum depth *d* is reached



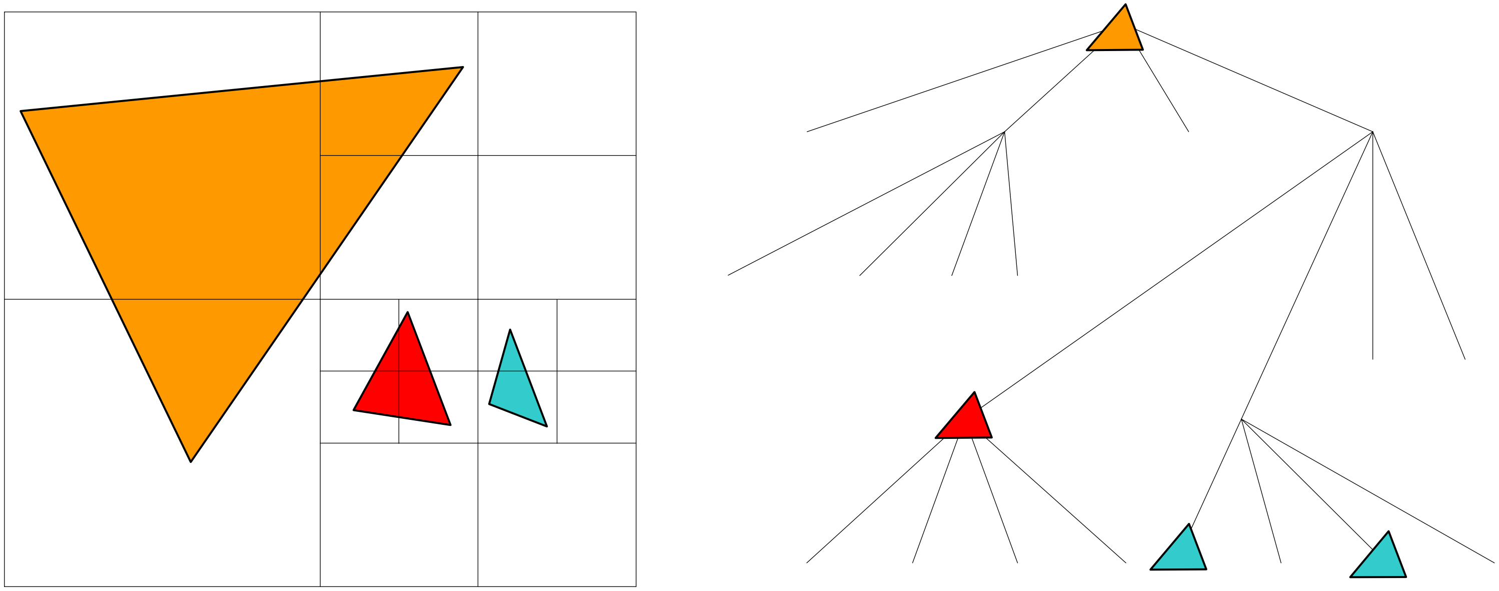
Nested Grids



Octree/(Quadtree)

Primitives in an Adaptive Grid

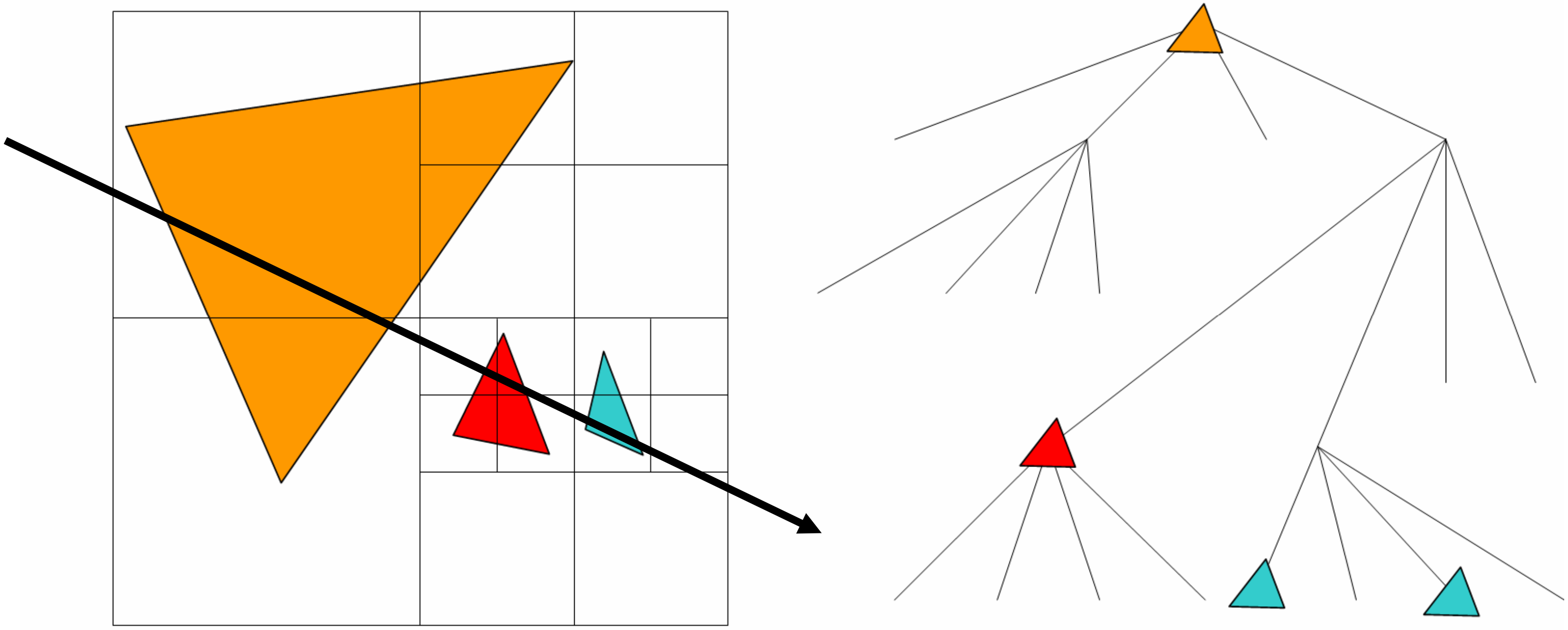
- Can live at intermediate levels, or be pushed to lowest level of grid



Octree/(Quadtree)

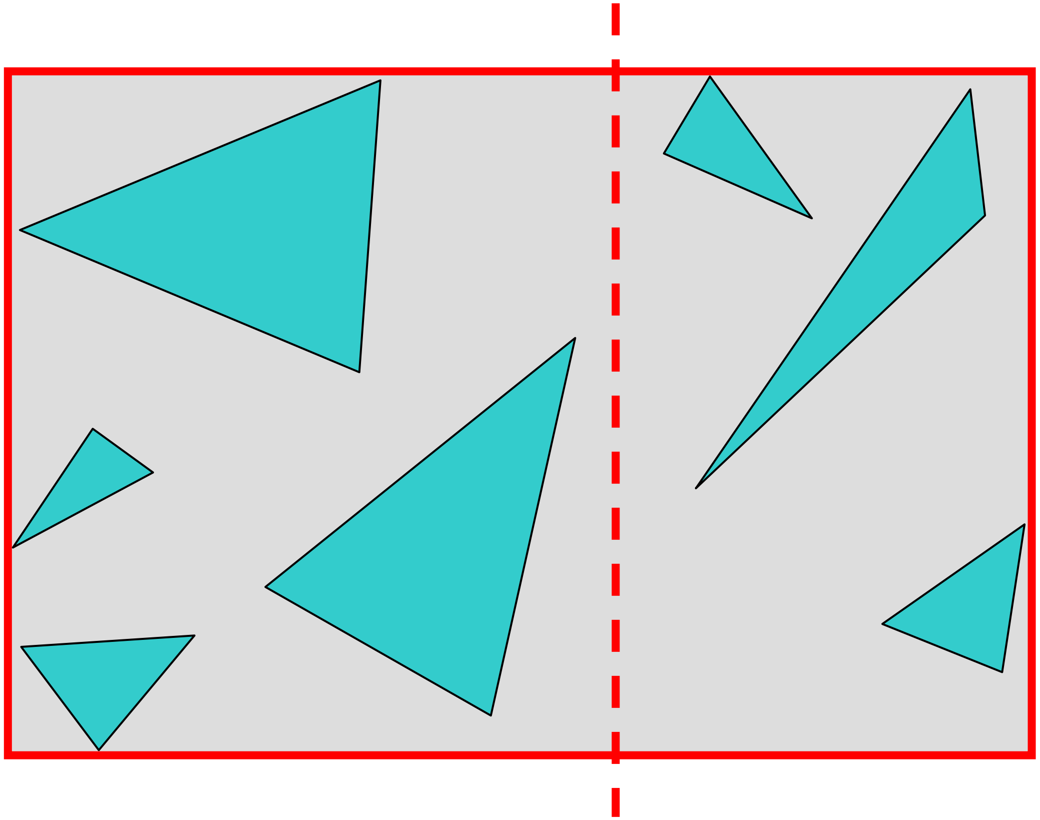
Adaptive Grid Discussion

- Advantages?
  - grid complexity matches geometric density
- Disadvantages?
  - more expensive to traverse (especially octree)



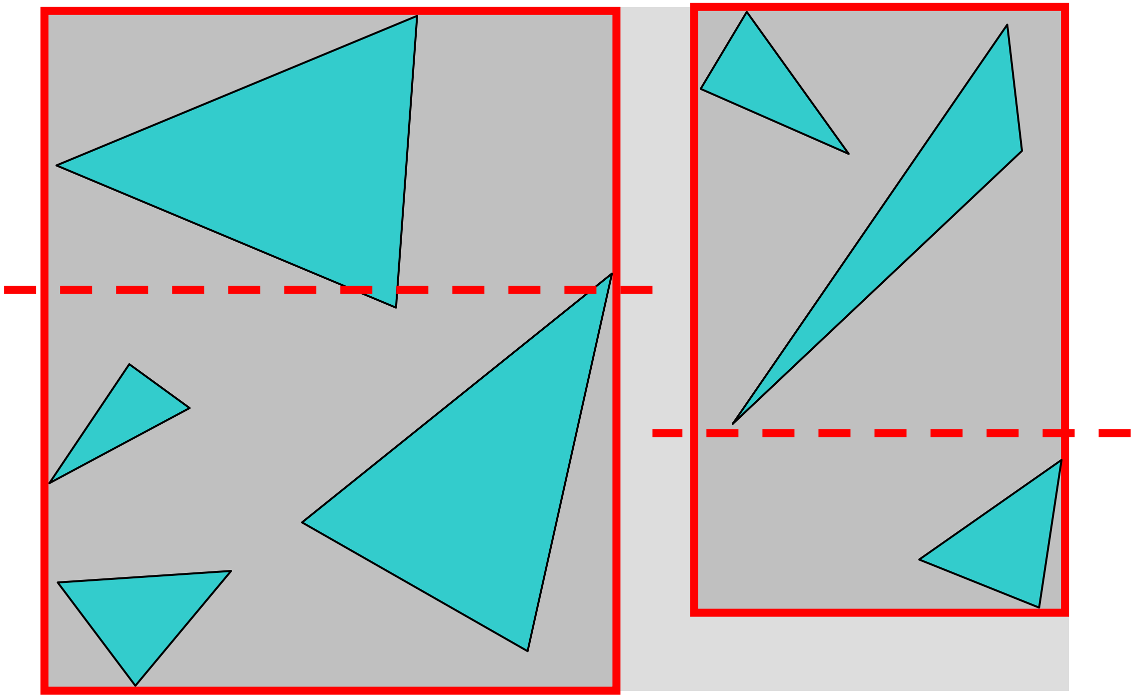
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



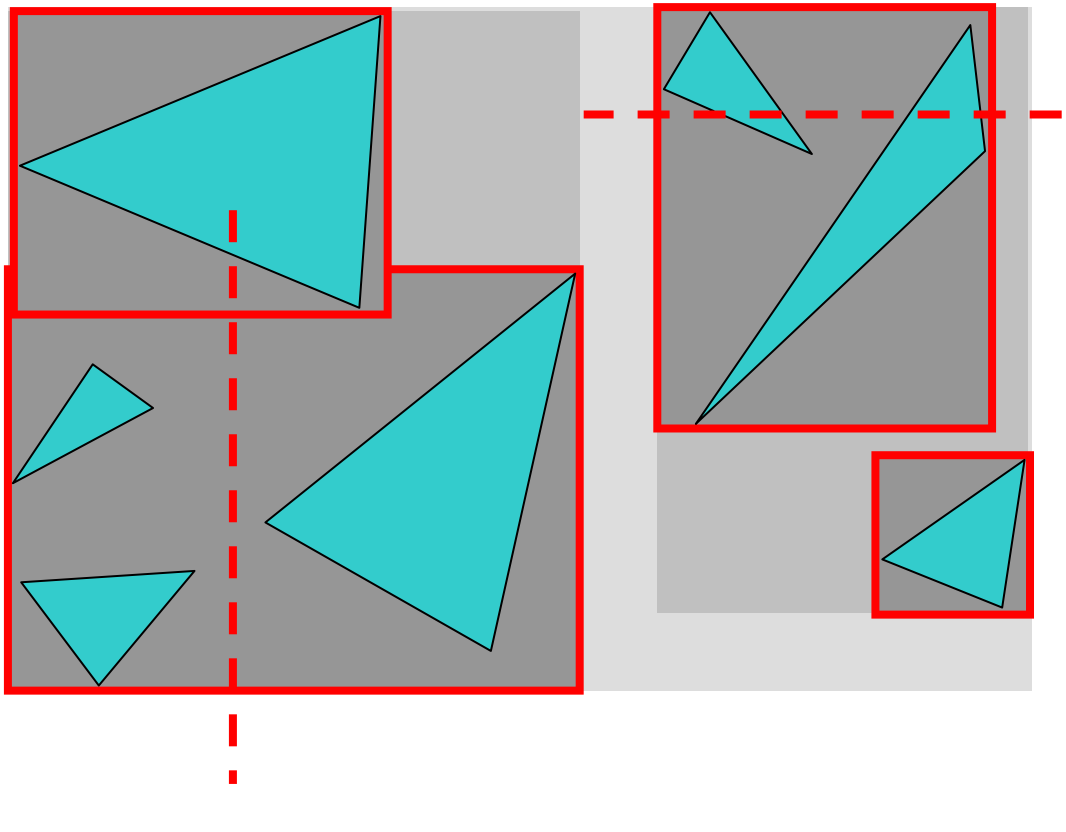
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



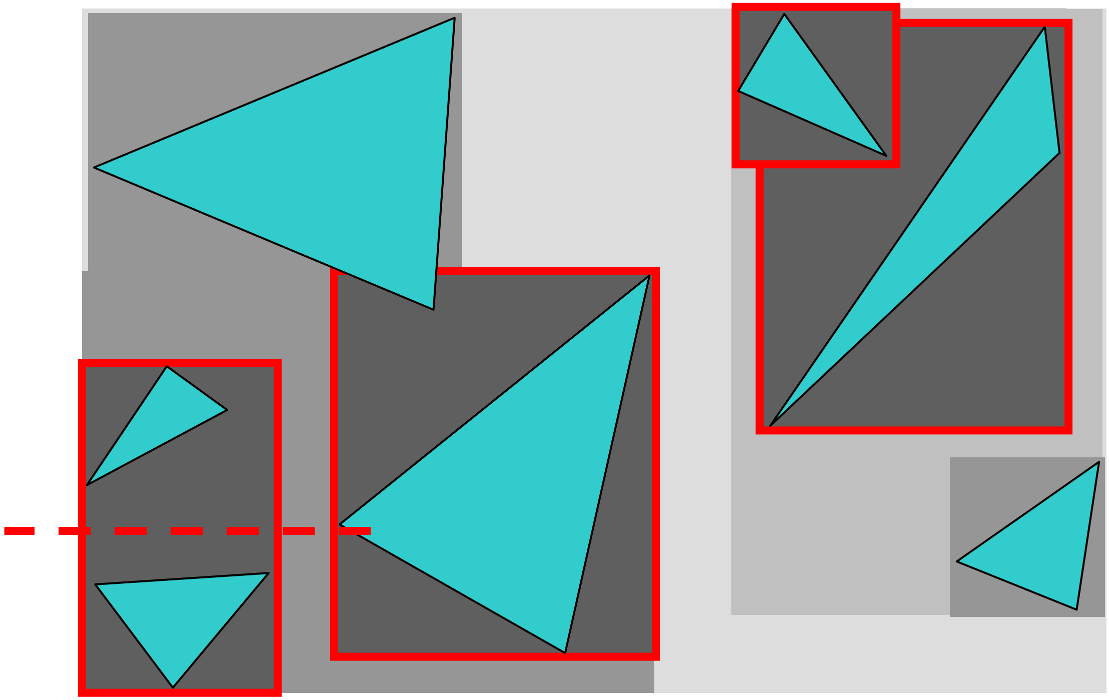
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



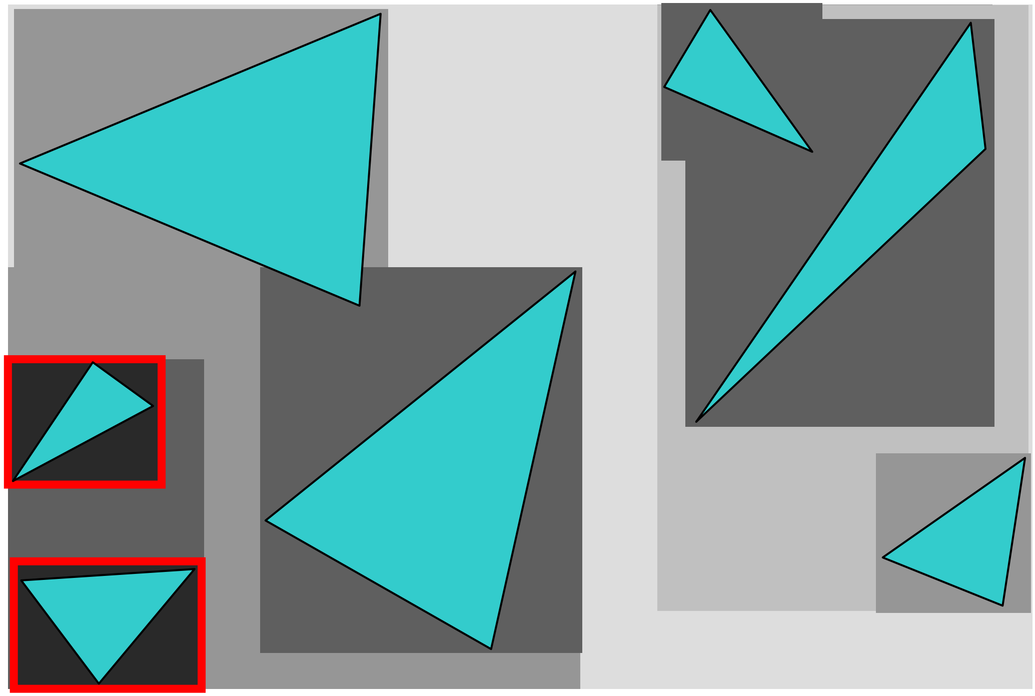
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



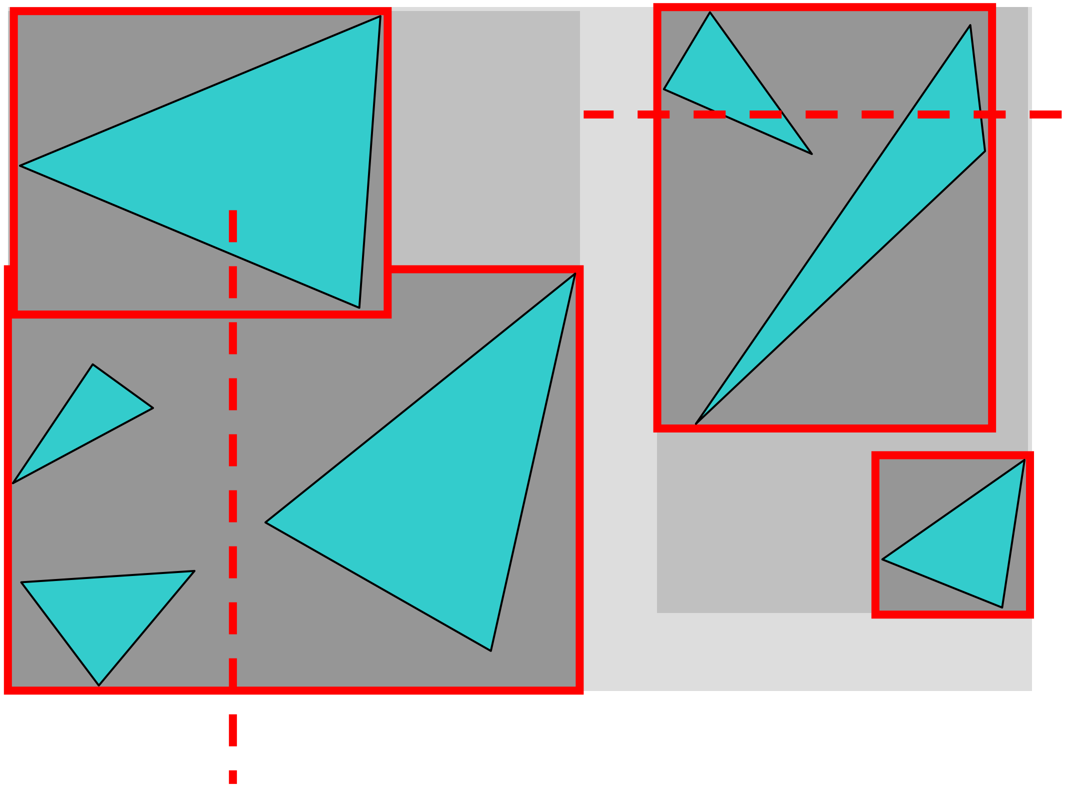
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse



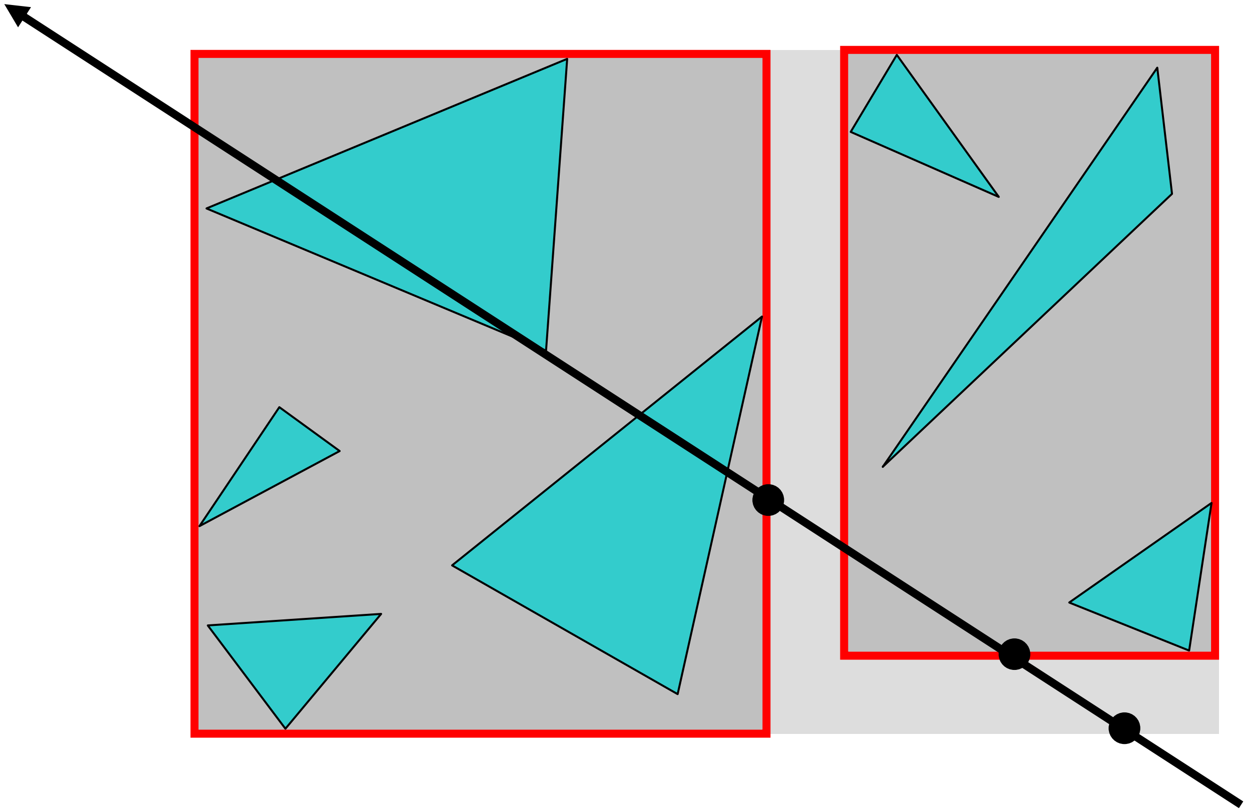
Where to split objects?

- At midpoint *OR*
- Sort, and put half of the objects on each side *OR*
- Use modeling hierarchy



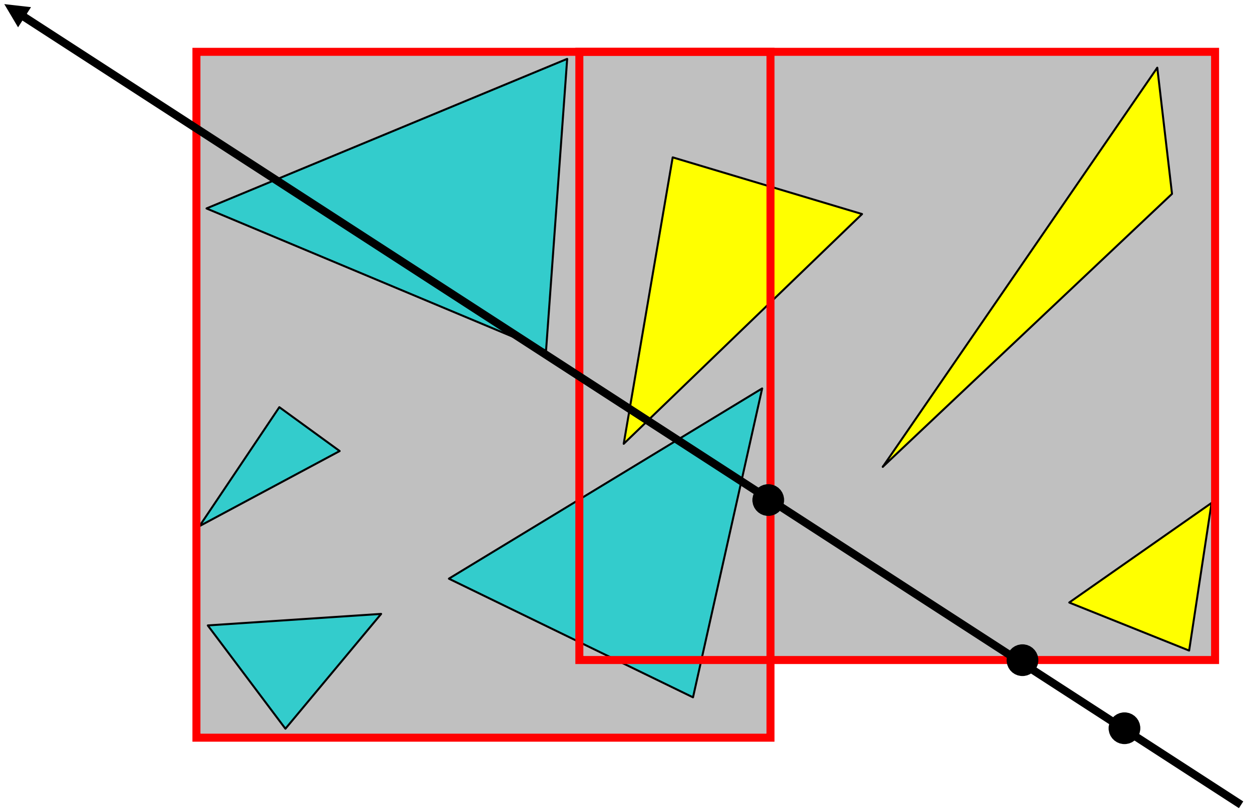
Intersection with BVH

- Check subvolume with closer intersection first



Intersection with BVH

- Don't return intersection immediately if the other subvolume may have a closer intersection



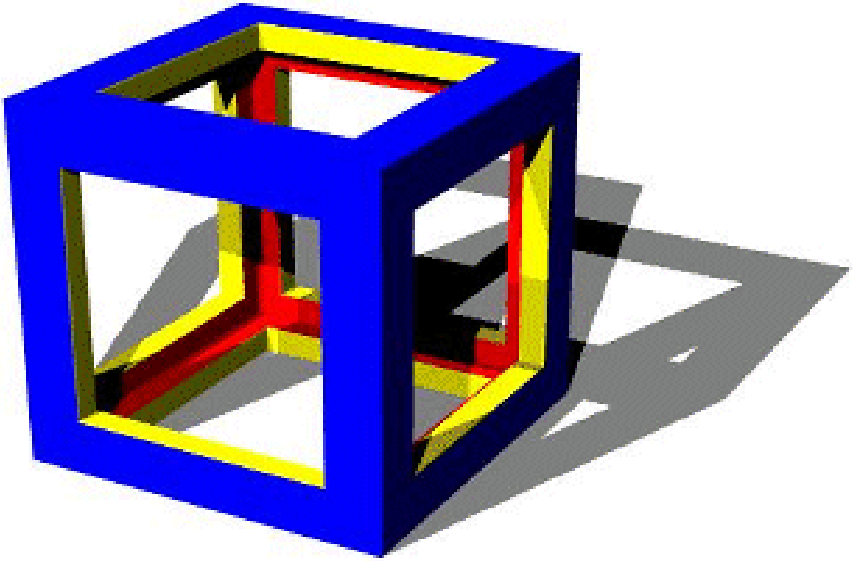
Bounding Volume Hierarchy Discussion

- Advantages
  - easy to construct
  - easy to traverse
  - binary

- Disadvantages
  - may be difficult to choose a good split for a node
  - poor split may result in minimal spatial pruning

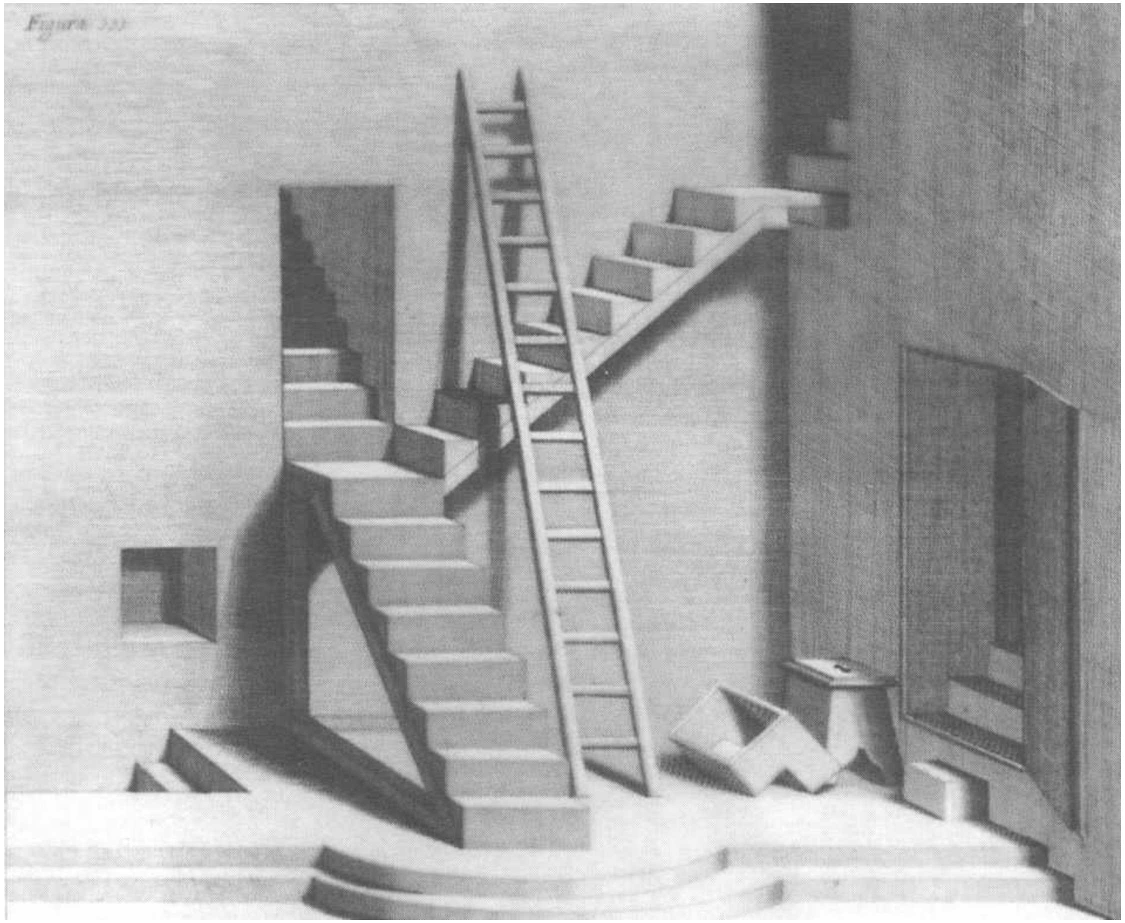
Ombres

- Why are Shadows Important?
- Shadows & Soft Shadows in Ray Tracing
- Planar Shadows
- Shadow Maps
- Shadow Volumes

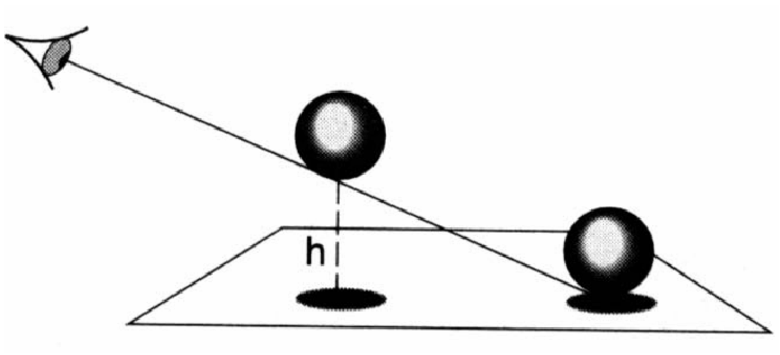


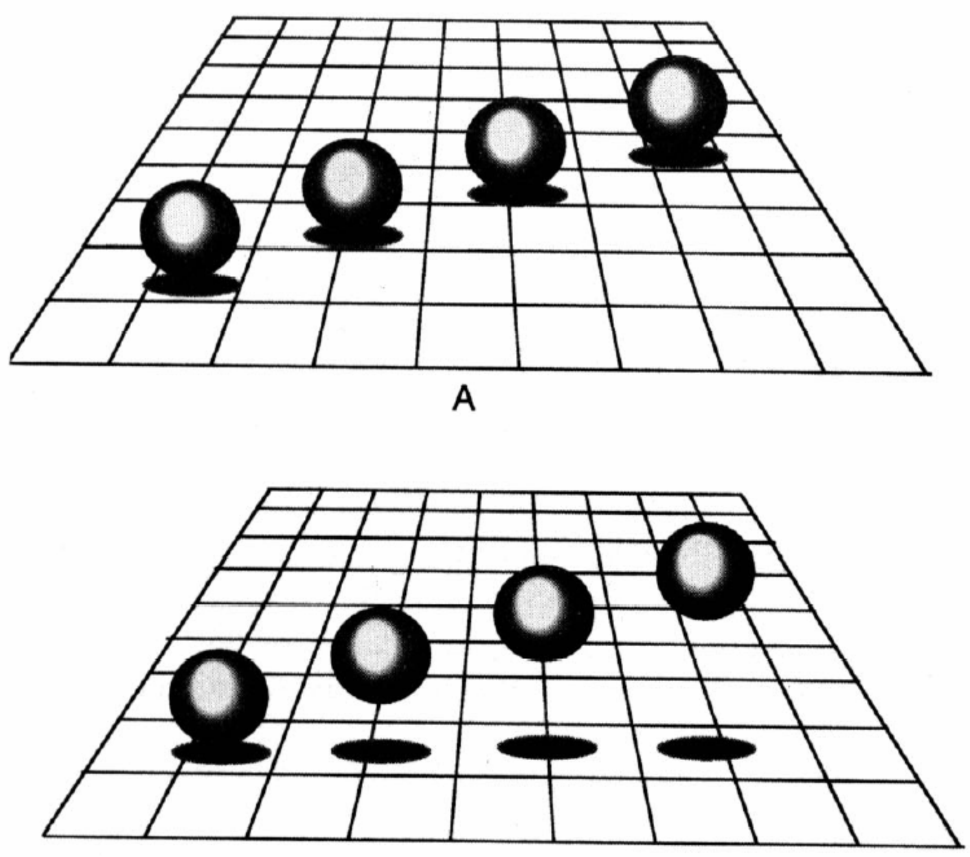
Why are Shadows Important?

- Depth cue
- Scene Lighting
- Realism
- Contact points



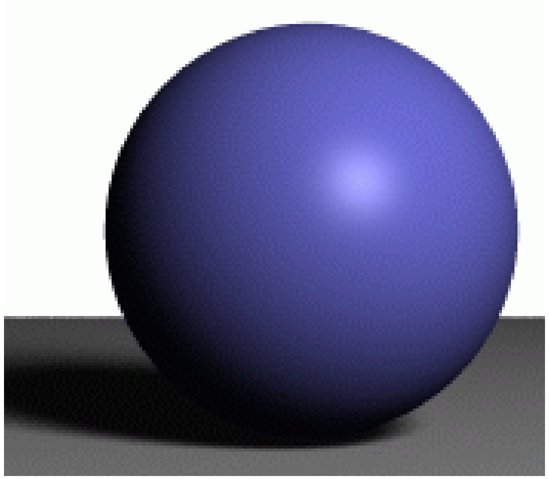
Shadows as a Depth Cue

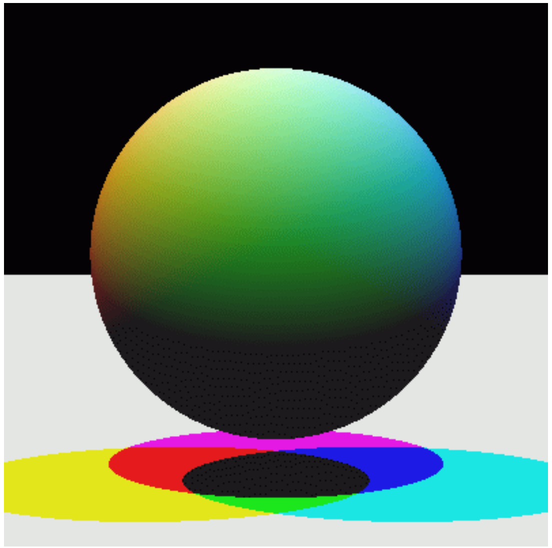


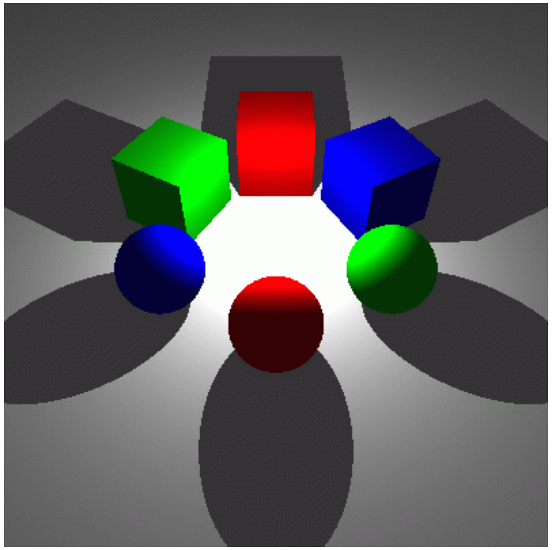


For Intuition about Scene Lighting

- Position of the light (e.g. sundial)
- Hard shadows vs. soft shadows
- Colored lights
- Directional light vs. point light



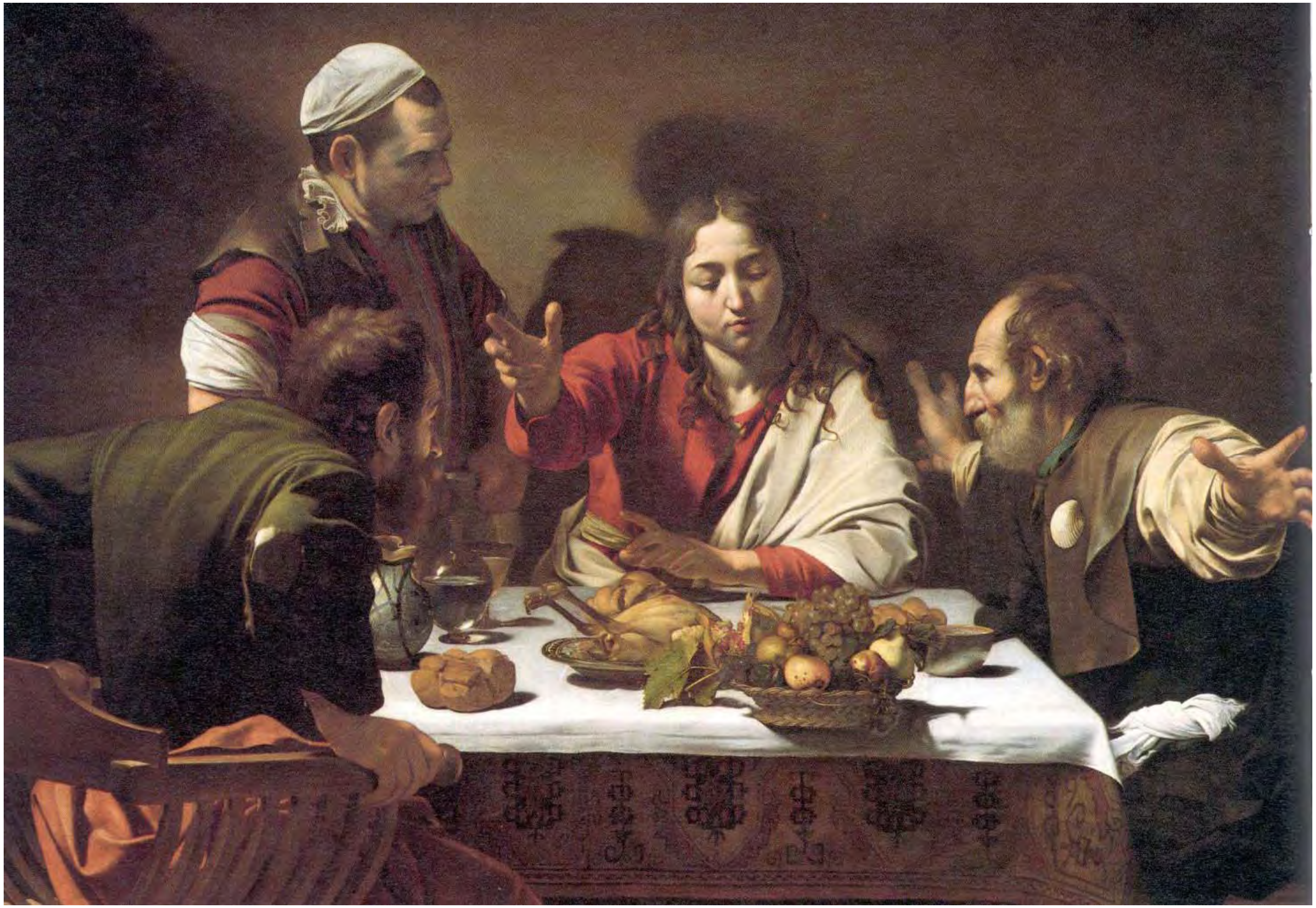




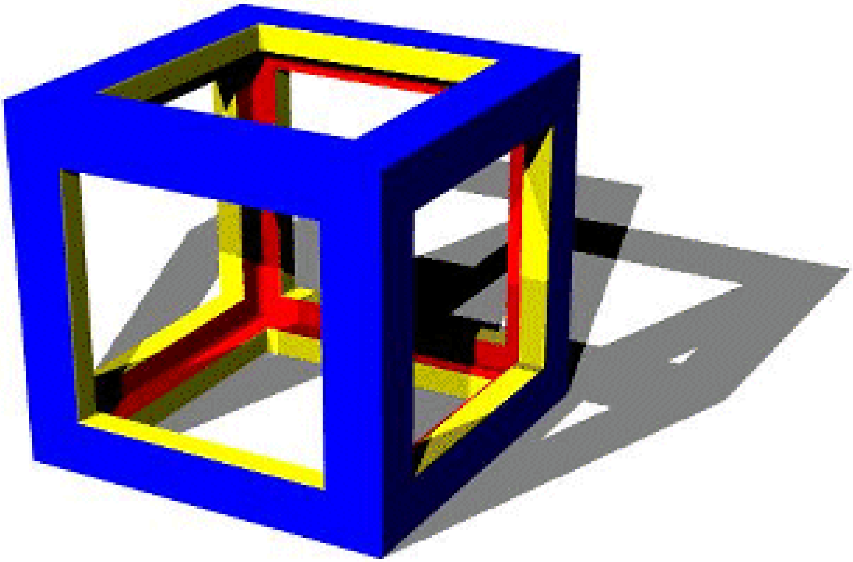
Shadows as the Origin of Painting

Shadows and Art

- Only in Western pictures (here Caravaggio)

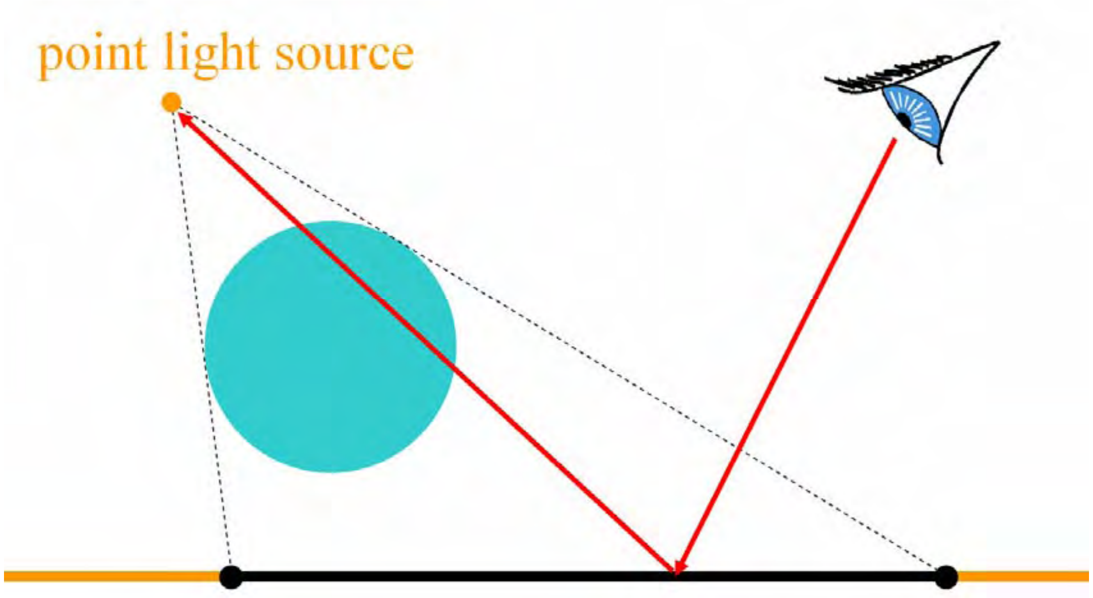


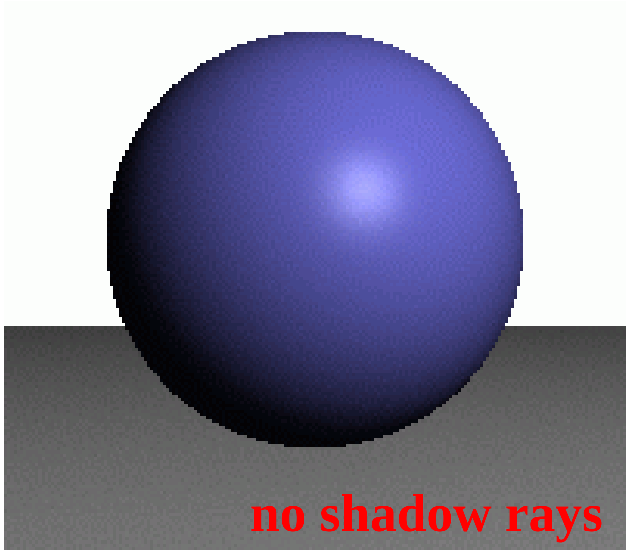
- Why are Shadows Important?
- Shadows & Soft Shadows in Ray Tracing
- Planar Shadows
- Shadow Maps
- Shadow Volumes

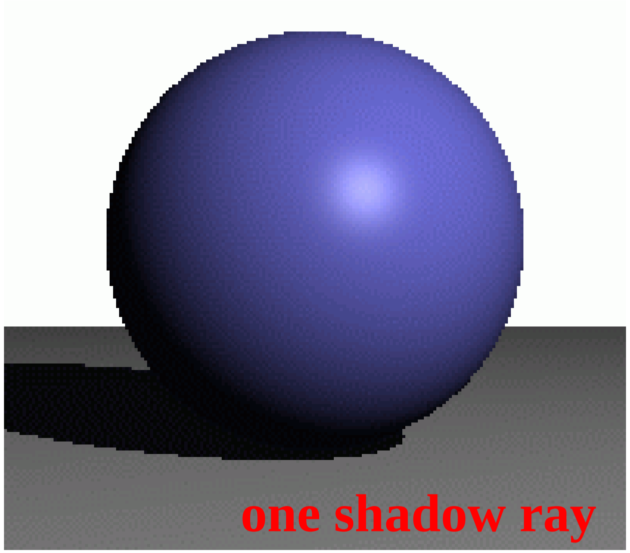


Shadows

- One shadow ray per intersection per point light source

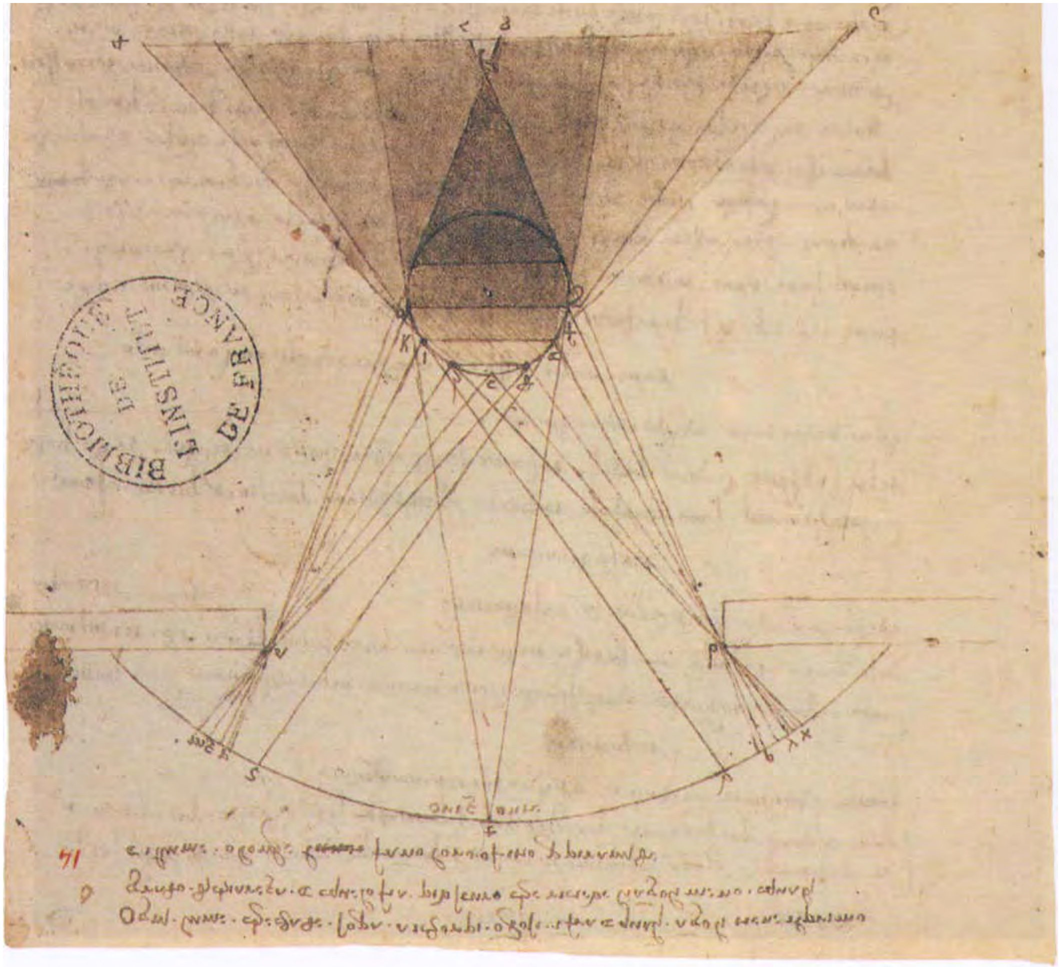






Soft Shadows

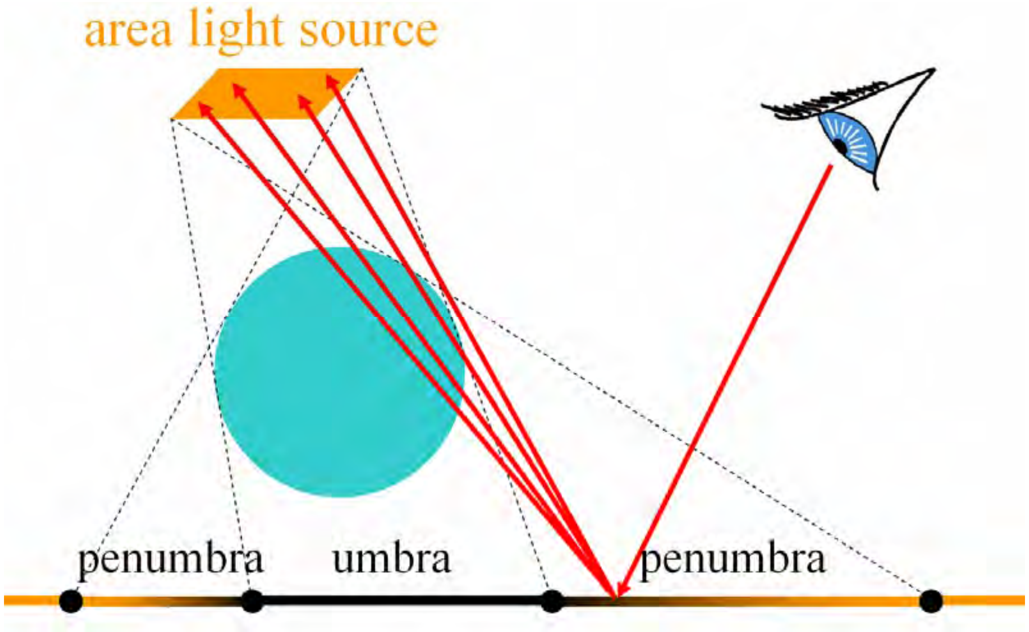
- Caused by extended light sources
- Umbra
  - source completely occluded
- Penumbra
  - Source partially occluded
- Fully lit

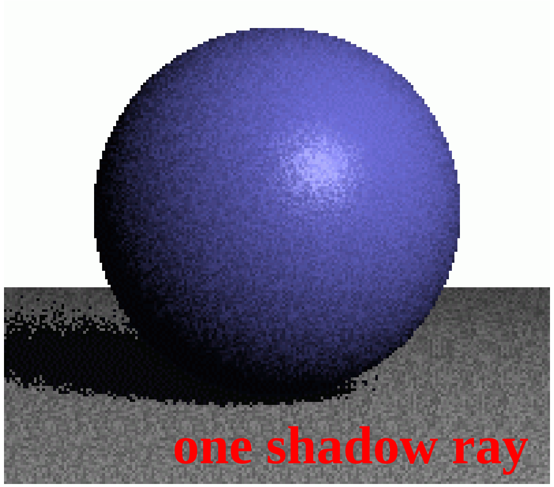


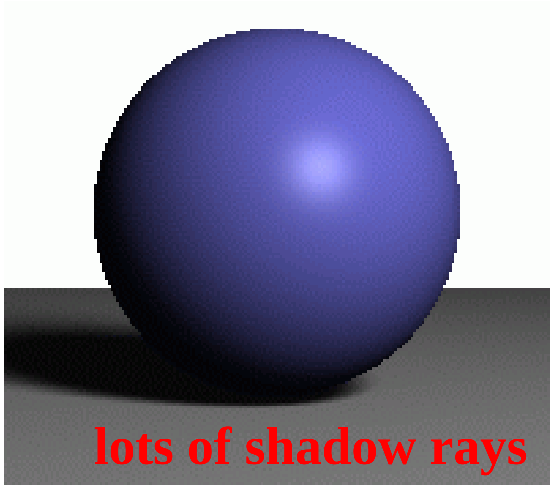
XVI. Léonard de Vinci (1452-1519). Lumière d'une fenêtre sur une sphère ombreuse avec (en partant du haut) ombre intermédiaire, primitive, dérivée et (sur la surface, en bas) portée. Plume et lavis sur pointe de métal sur papier. 24 x 38 cm. Paris, Bibliothèque de l'Institut de France (ms. 2185 ; B.N.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

Soft Shadows

- Multiple shadow rays to sample area light source

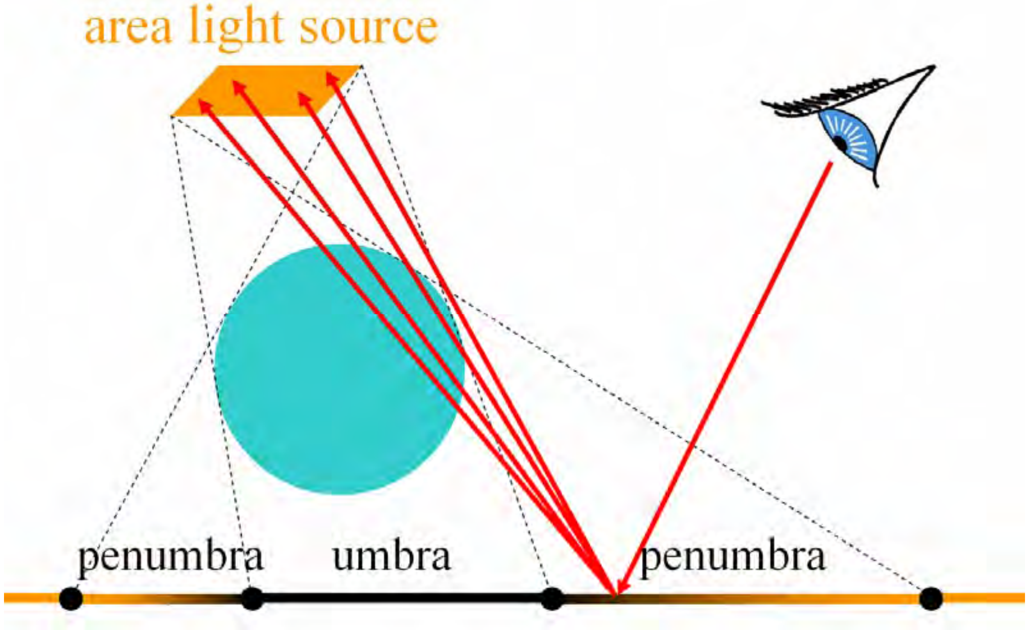




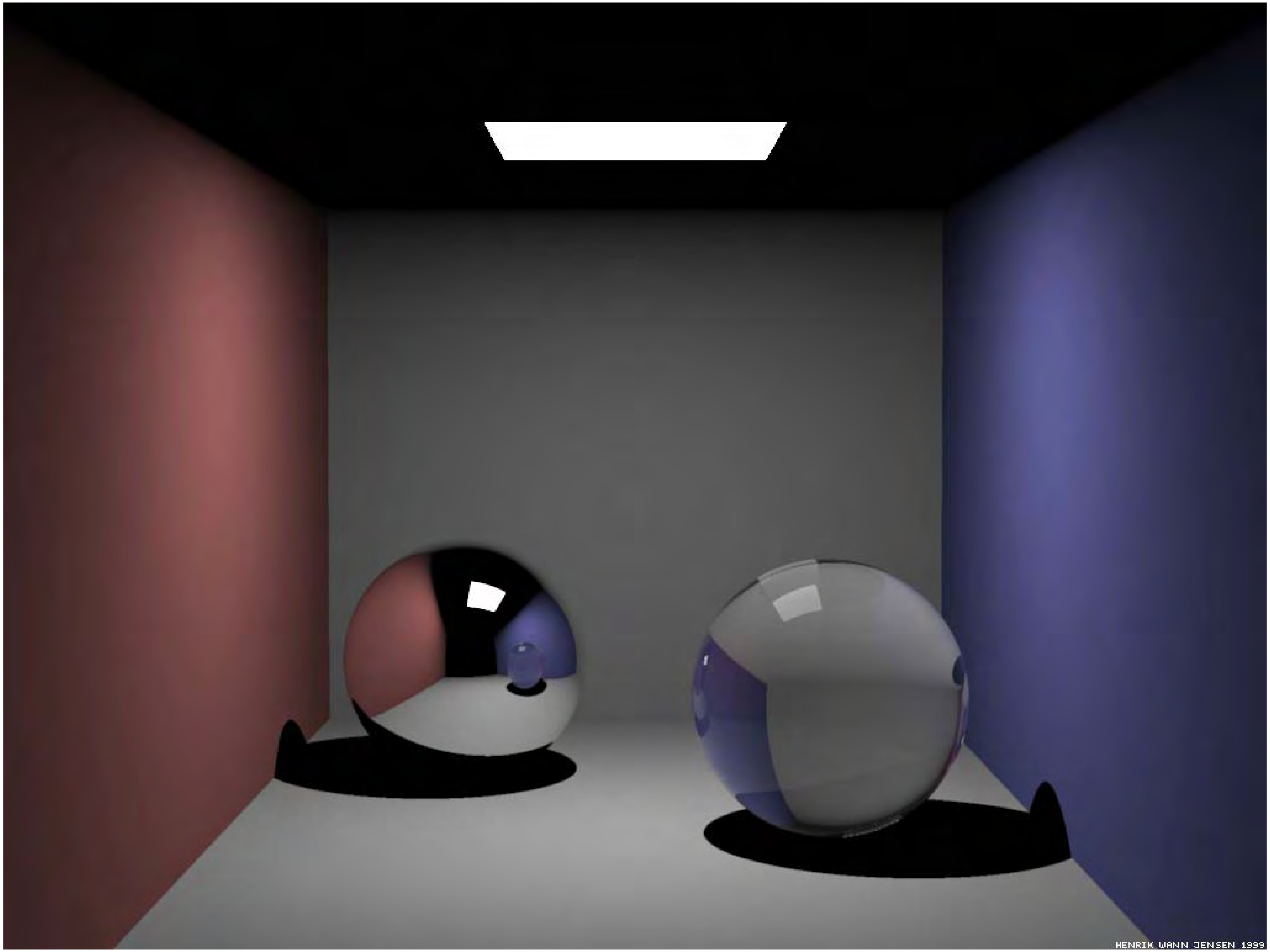


Shadows in Ray Tracing

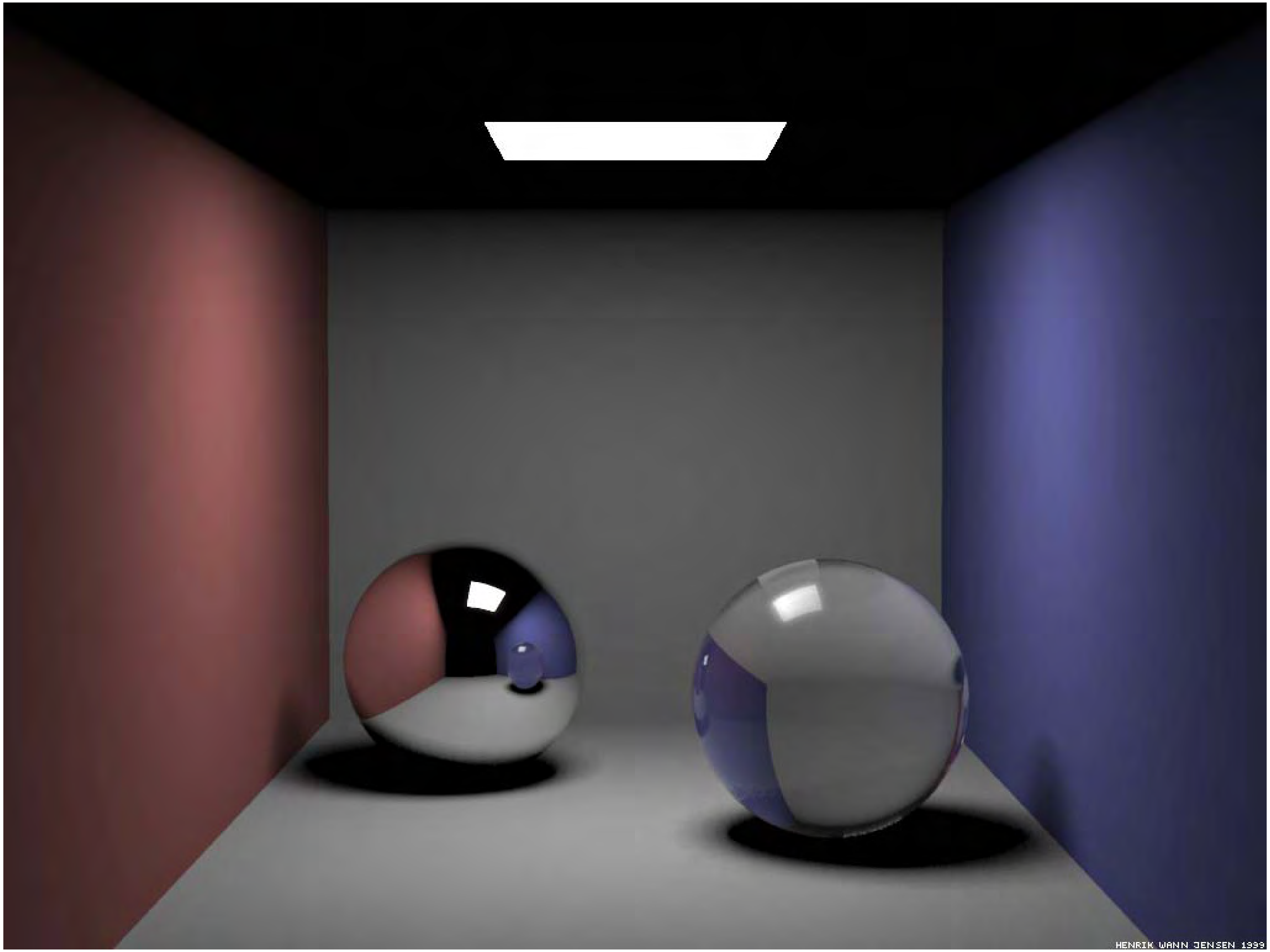
- Shoot ray from visible point to light source
- If blocked, discard light contribution
- Optimization?
  - Stop after first intersection (don’t worry about tmin)
  - Coherence: remember the previous occluder, and test that object first



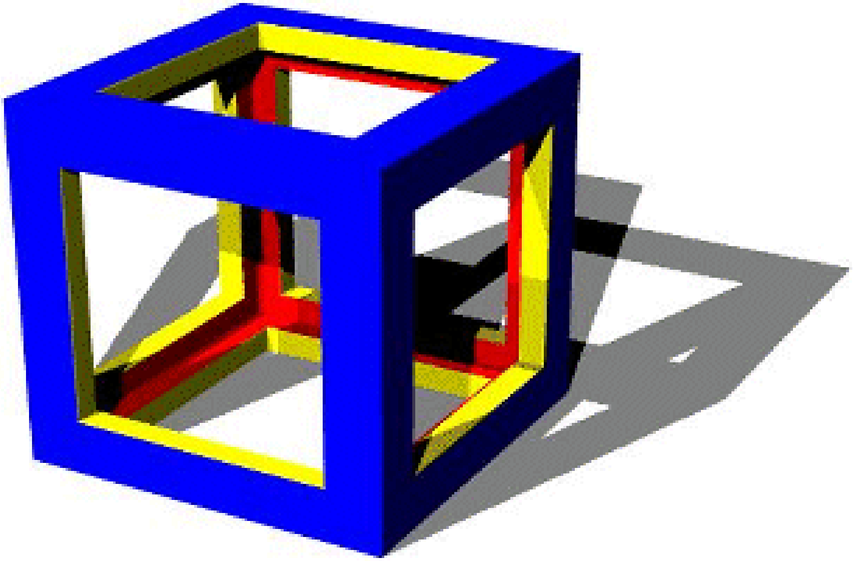
Traditional Ray Tracing



Ray Tracing + Soft Shadows

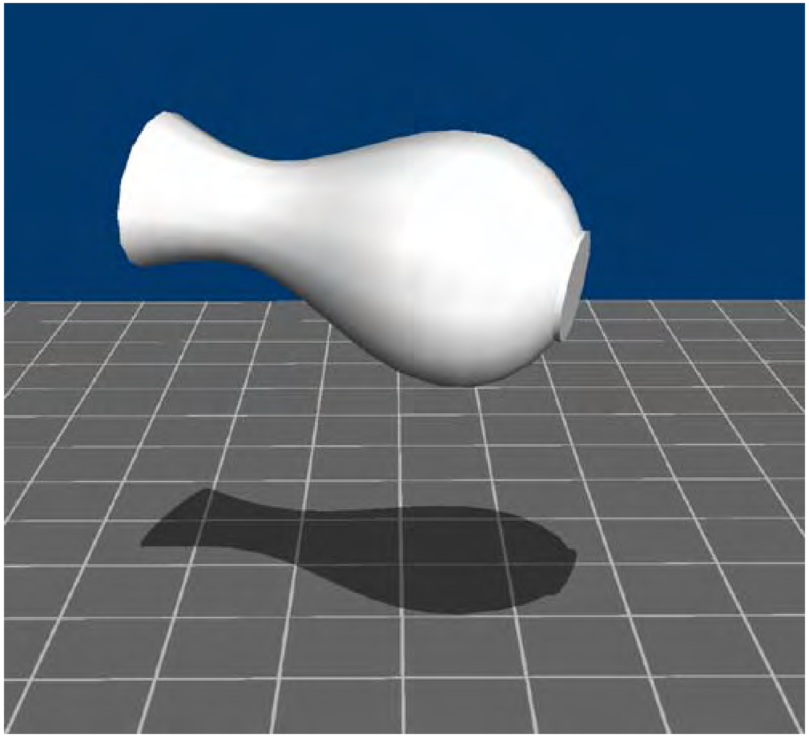


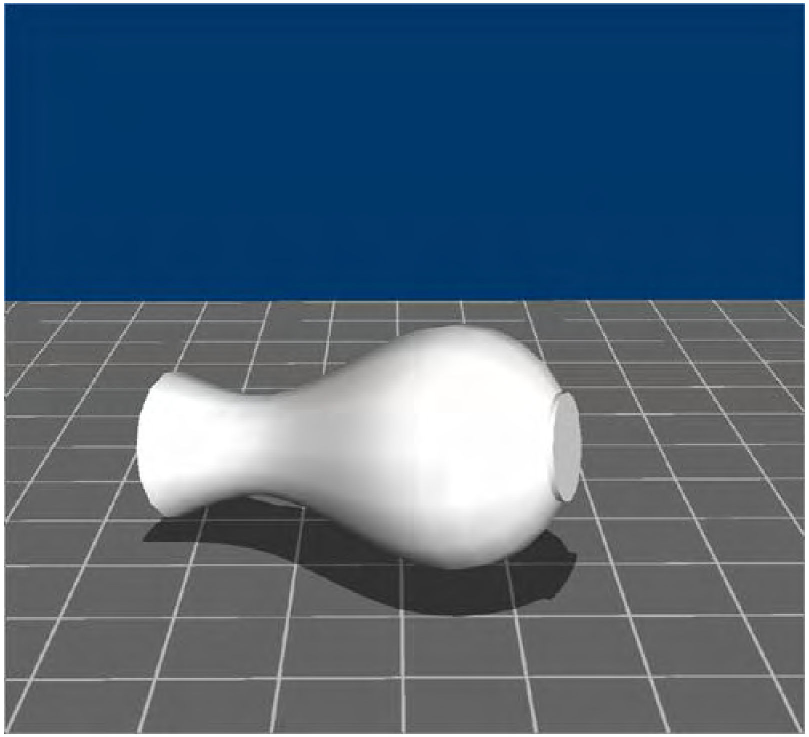
- Why are Shadows Important?
- Shadows & Soft Shadows in Ray Tracing
- Planar Shadows
- Shadow Maps
- Shadow Volumes



Cast Shadows on Planar Surfaces

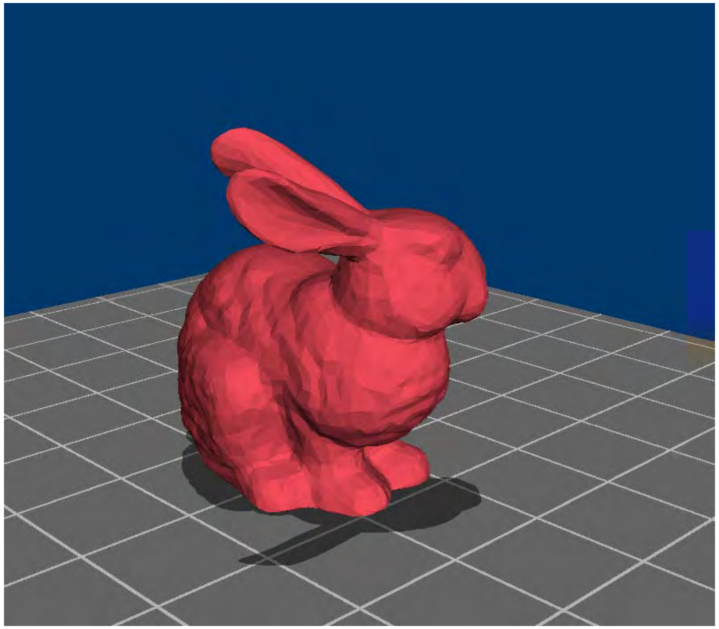
- Draw the object primitives a second time, projected to the ground plane

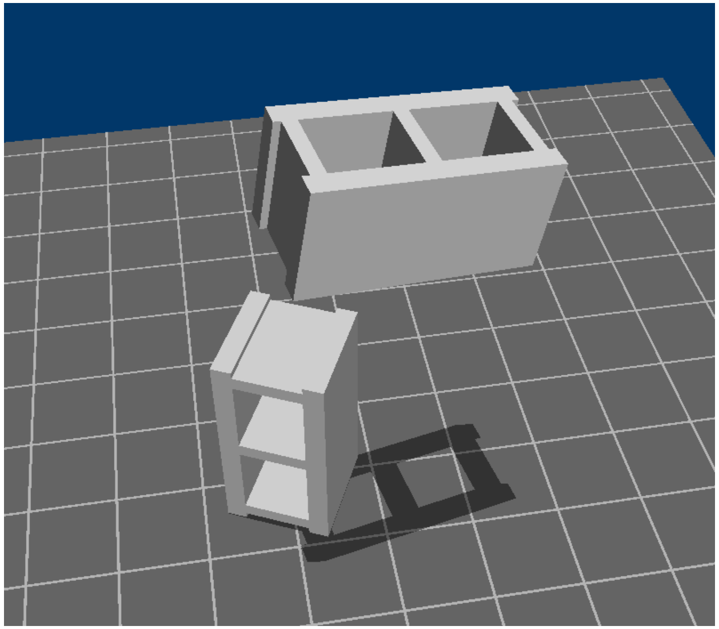




Limitations of Planar Shadows

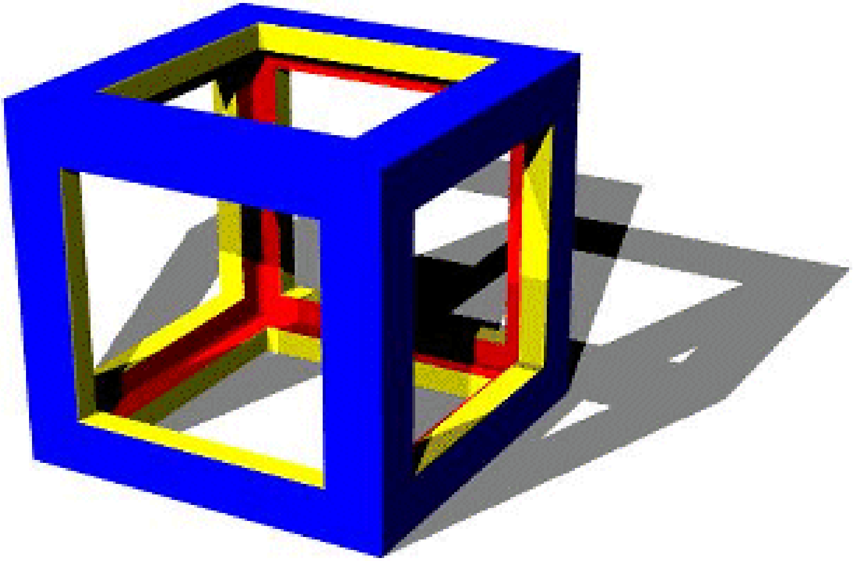
- Does not produce self-shadows, shadows cast on other objects, shadows on curved surfaces, etc.





Today

- Why are Shadows Important?
- Shadows & Soft Shadows in Ray Tracing
- Planar Shadows
- Shadow Maps
  - Texture Mapping
  - Shadow View Duality
- Shadow Volumes



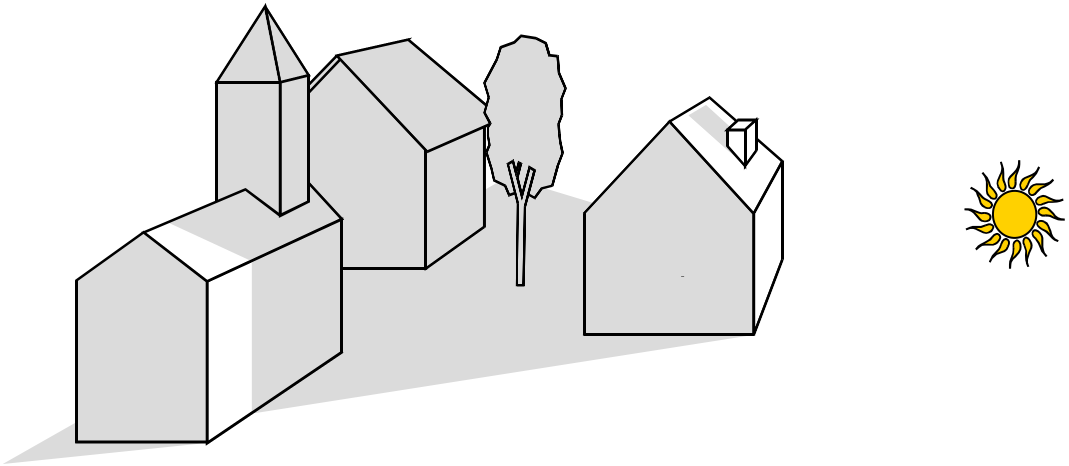
Texture Mapping

- Don't have to represent everything with geometry

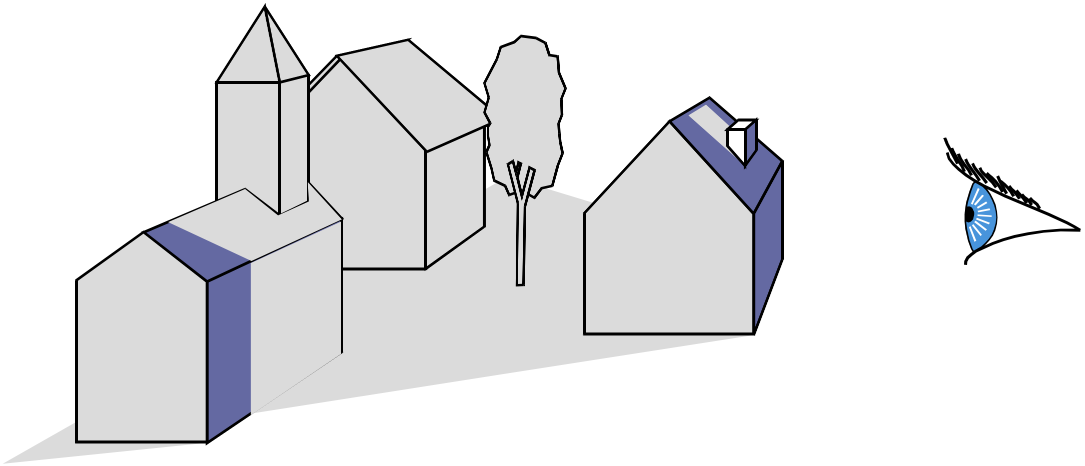
SHREK
© 2001 PDI/DreamWorks

Shadow/View Duality

- A point is lit if it is visible from the light source



- Shadow computation similar to view computation



Fake Shadows using Projective Textures

- Separate obstacle and receiver
- Compute b/w image of obstacle from light
- Use image as projective texture for each receiver

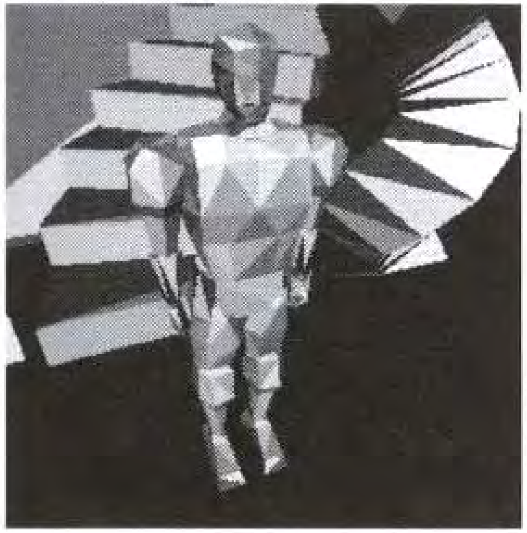
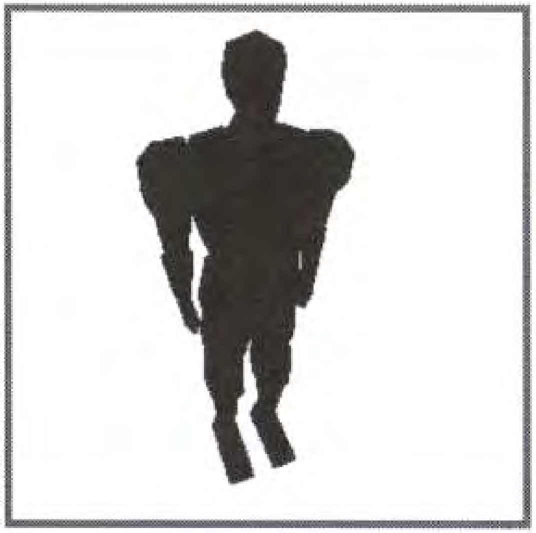
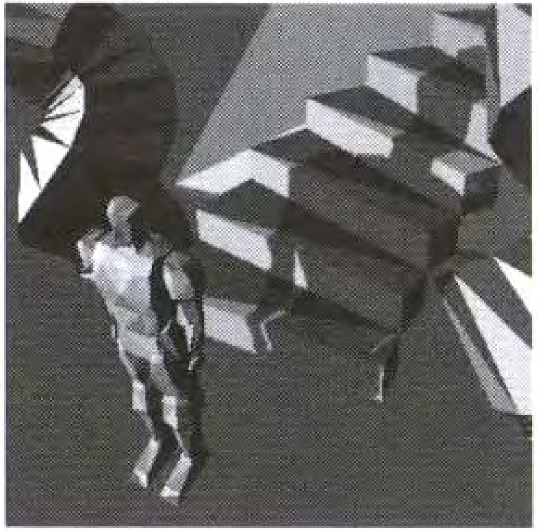


Image from light source



BW image of obstacle

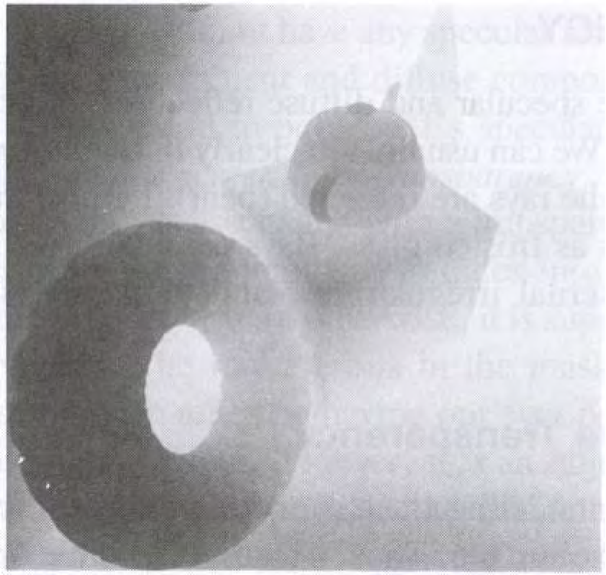


Final image

Figure from Moller & Haines “Real Time Rendering”

Shadow maps

- In Renderman
  - (High-end production software)



Shadow Mapping

- Texture mapping with depth information
- ≥ 2 passes through the pipeline
  - Compute shadow map (depth from light source)
  - Render final image (check shadow map to see if points are in shadow)

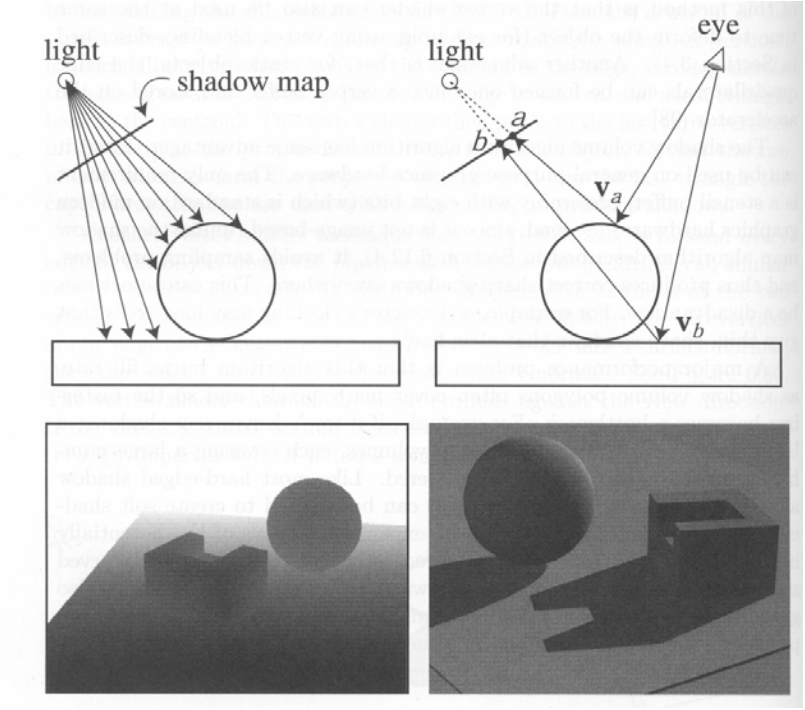
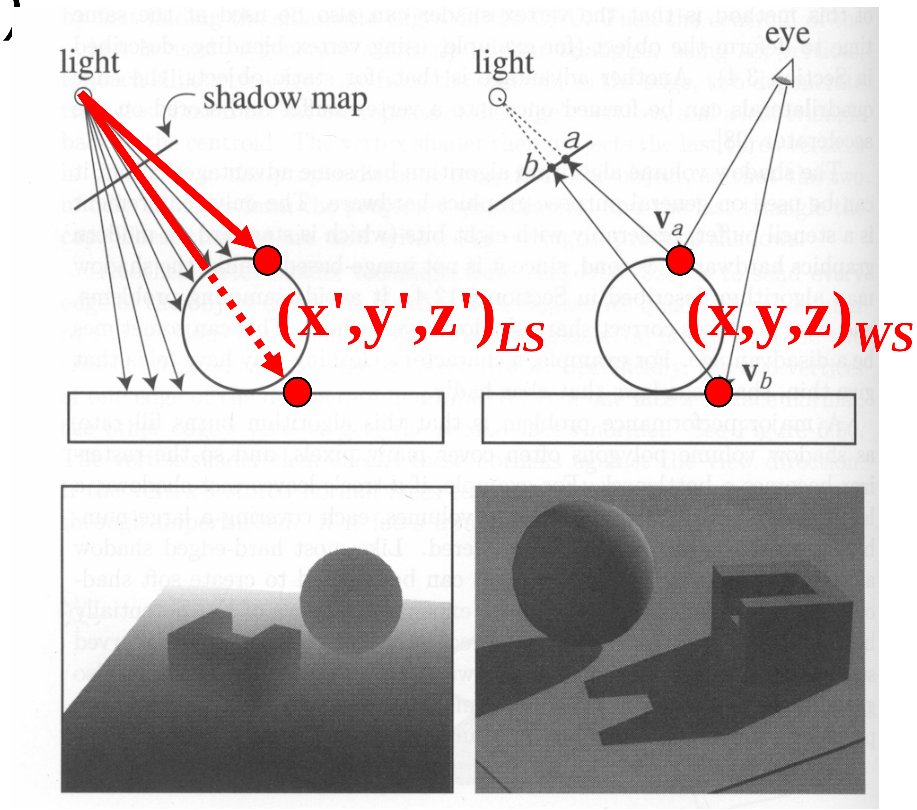


Figure from Foley et al. “Computer Graphics Principles and Practice”

Shadow Map Look Up

- We have a 3D point (x,y,z)
- How do we look up the depth from the shadow map?

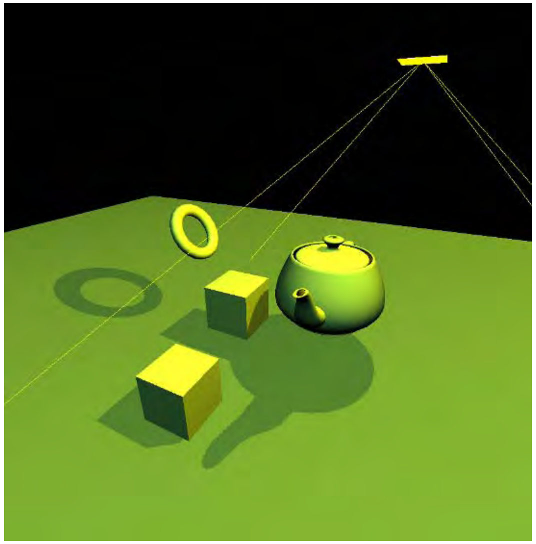
- Use the 4x4 perspective projection matrix from the light source to get $(x',y',z')_{LS}$
- ShadowMap(x',y') < z'?

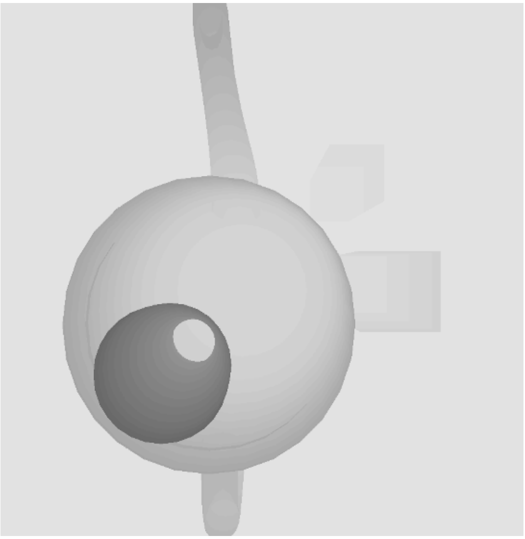


Foley et al. “Computer Graphics Principles and Practice”

Shadow Maps

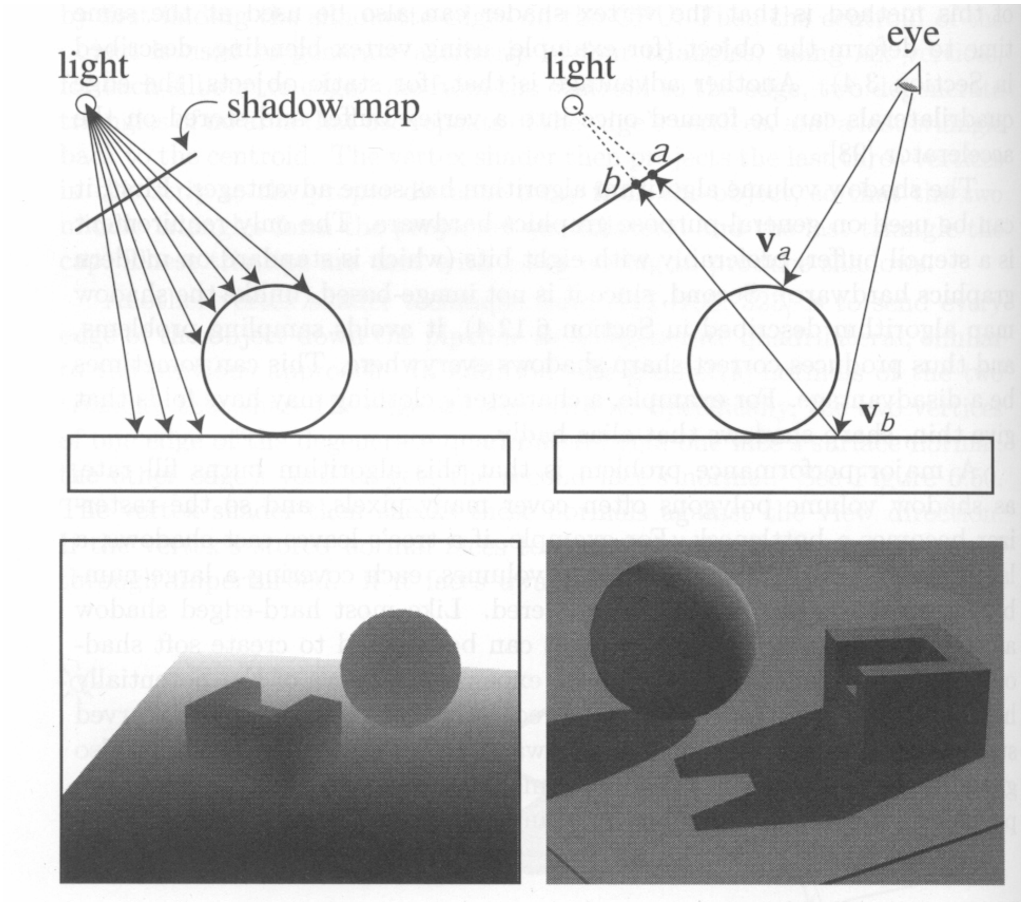
- Can be done in hardware
- Using hardware texture mapping
  - Texture coordinates u,v,w generated using 4x4 matrix
  - Modern hardware permits tests on texture values





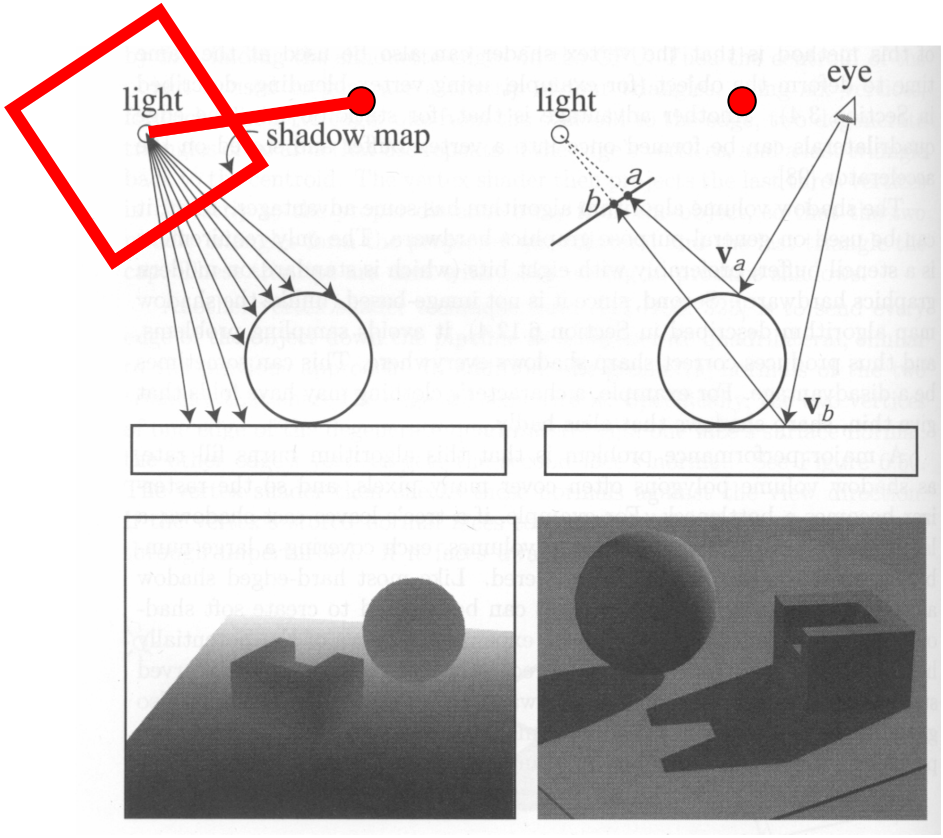
Limitations of Shadow Maps

1. Field of View
2. Bias (Epsilon)
3. Aliasing



1. Field of View Problem

- What if point to shadow is outside field of view of shadow map?
  - Use cubical shadow map
  - Use only spot lights!

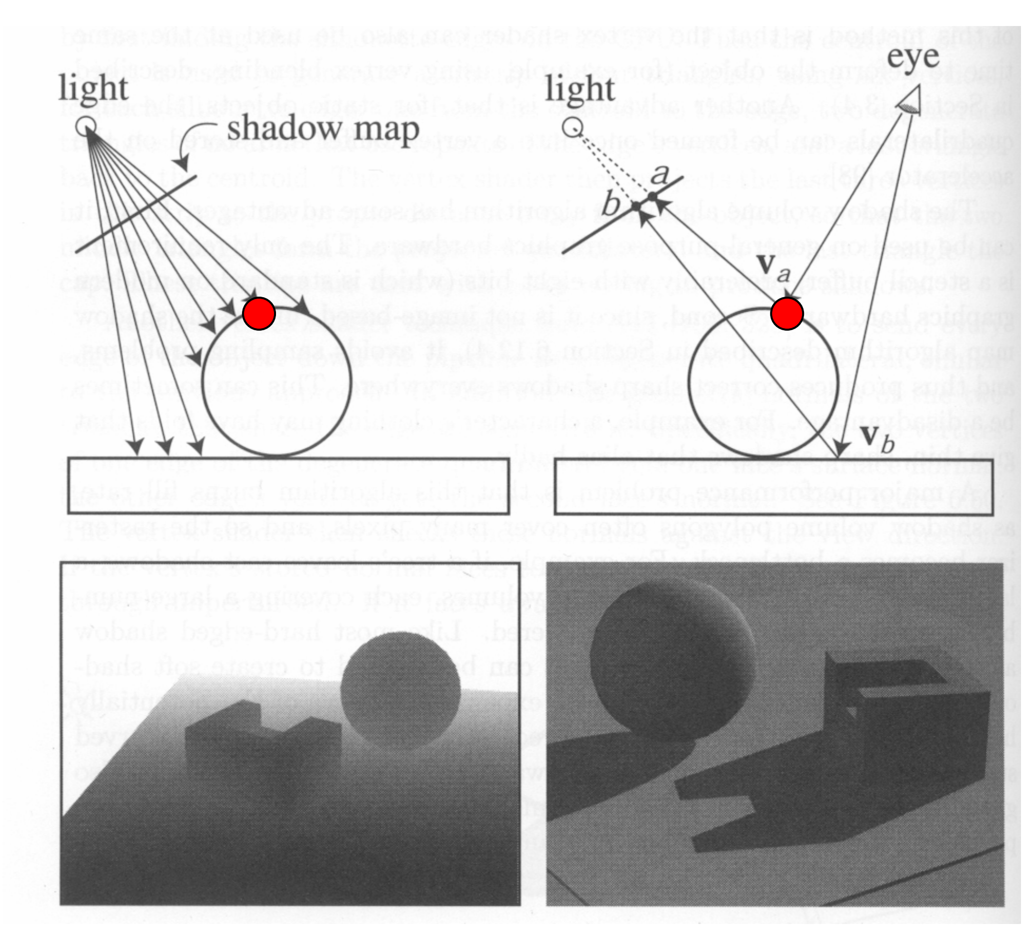


2. The Bias (Epsilon) Nightmare

- For a point visible from the light source

  ShadowMap(x’,y’) ≈ z’

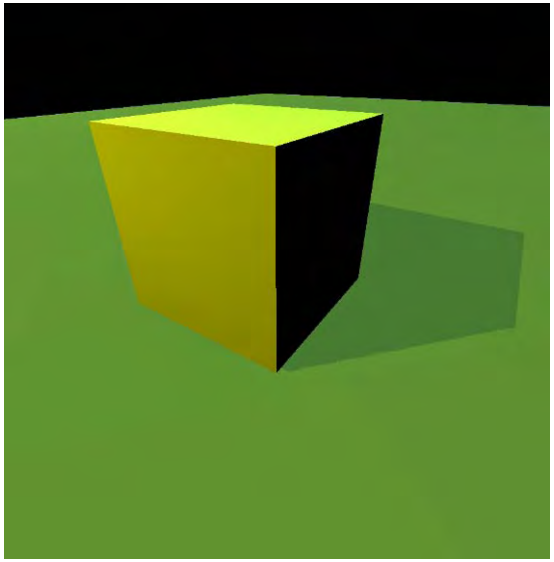
- How can we avoid erroneous self-shadowing?
  – Add bias (epsilon)



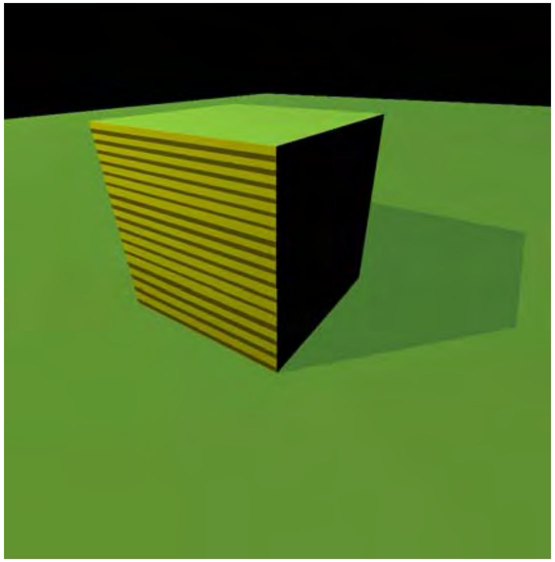
2. Bias (Epsilon) for Shadow Maps

ShadowMap(x’,y’) + bias < z’

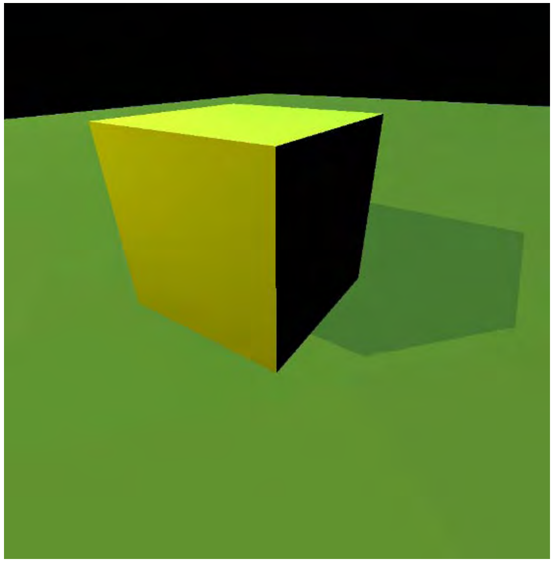
Choosing a good bias value can be very tricky



Correct image



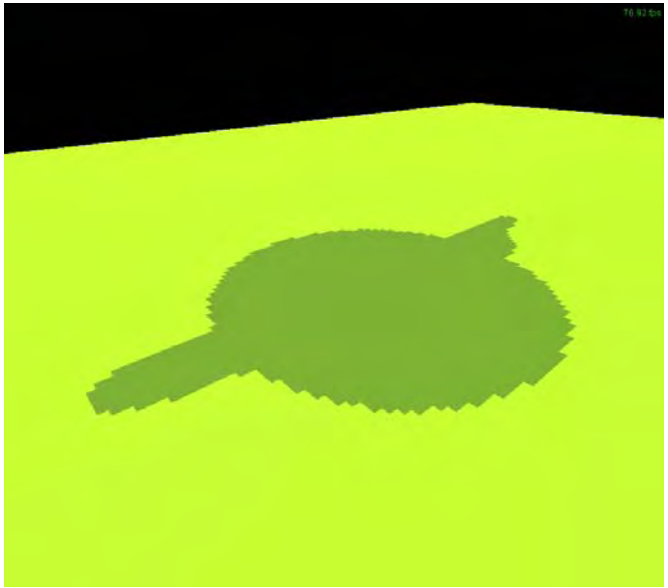
Not enough bias

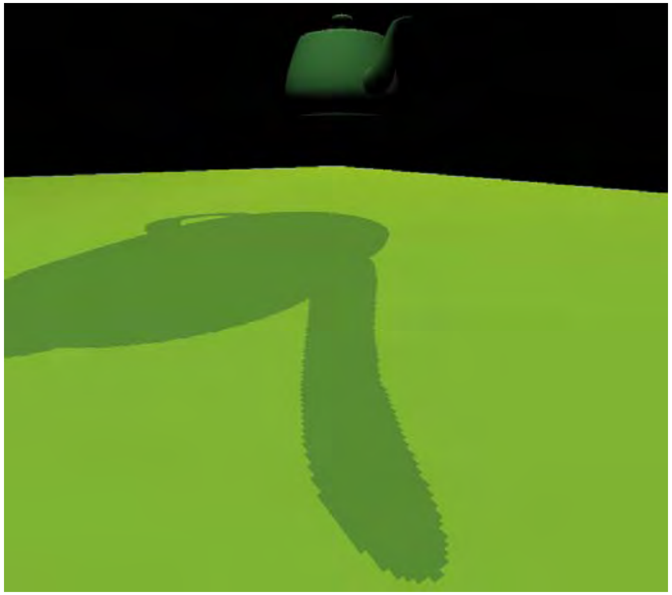


Way too much bias

3. Shadow Map Aliasing

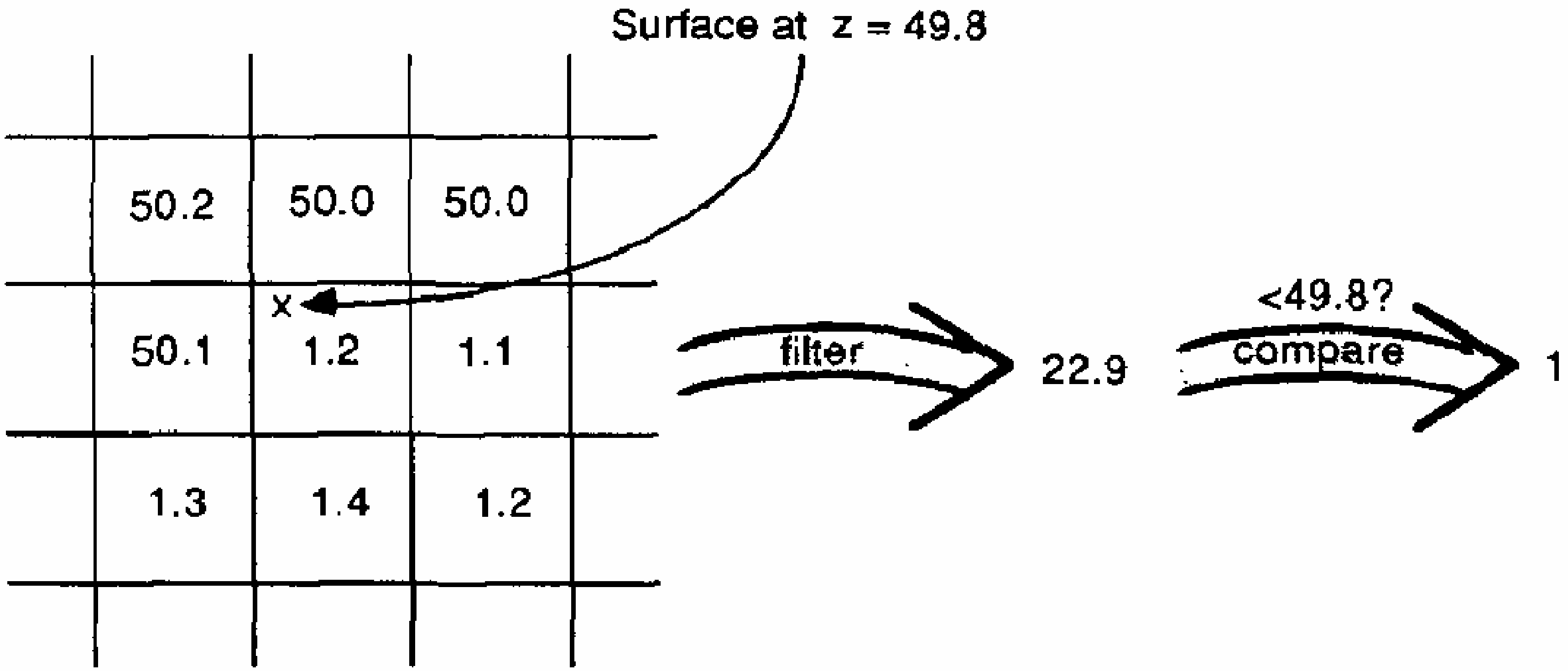
- Under-sampling of the shadow map
- Reprojection aliasing – especially bad when the camera & light are pointing towards each other





Shadow Map Filtering

- Should we filter the depth?
  (weighted average of neighboring depth

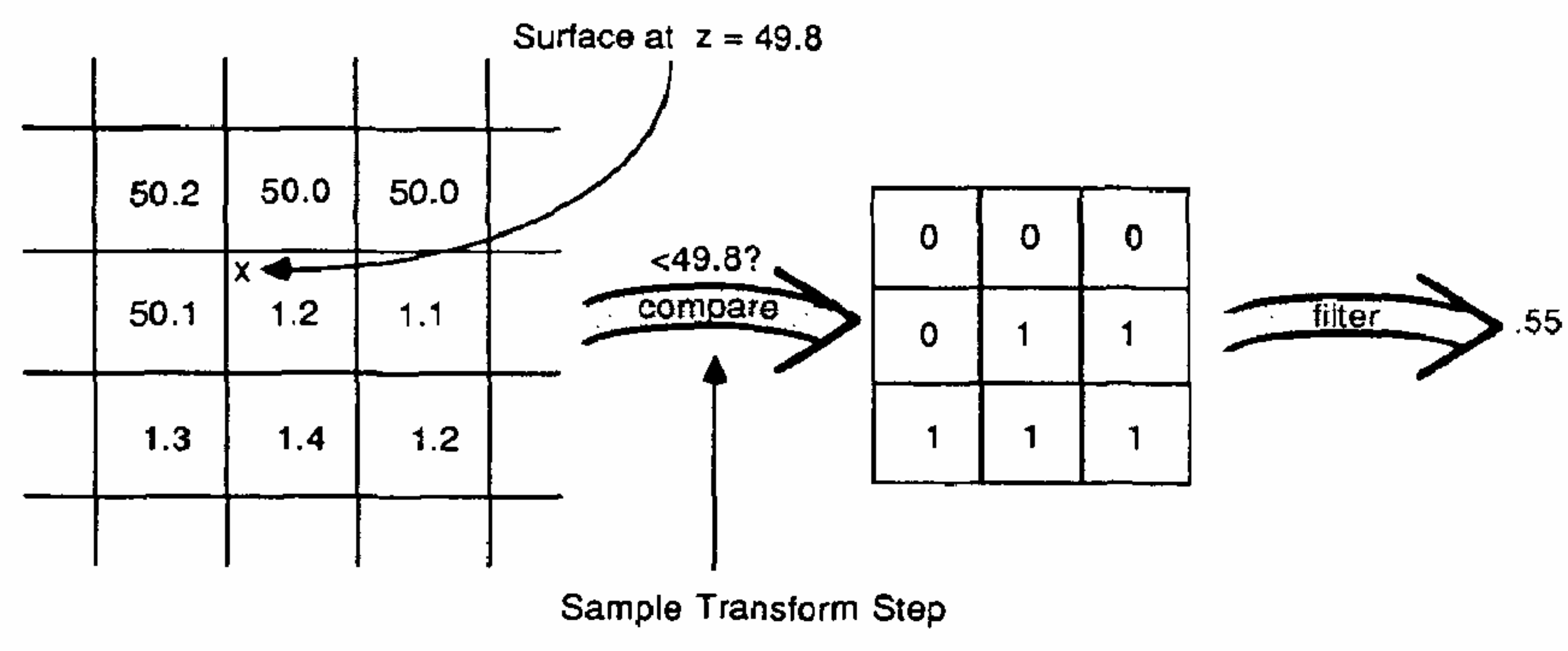


50.2	50.0	50.0
50.1	1.2	1.1
1.3	1.4	1.2

a) Ordinary texture map filtering. Does not work for depth maps.

Percentage Closer Filtering

- Instead filter the result of the test (weighted average of comparison results)
- But makes the bias issue more tricky

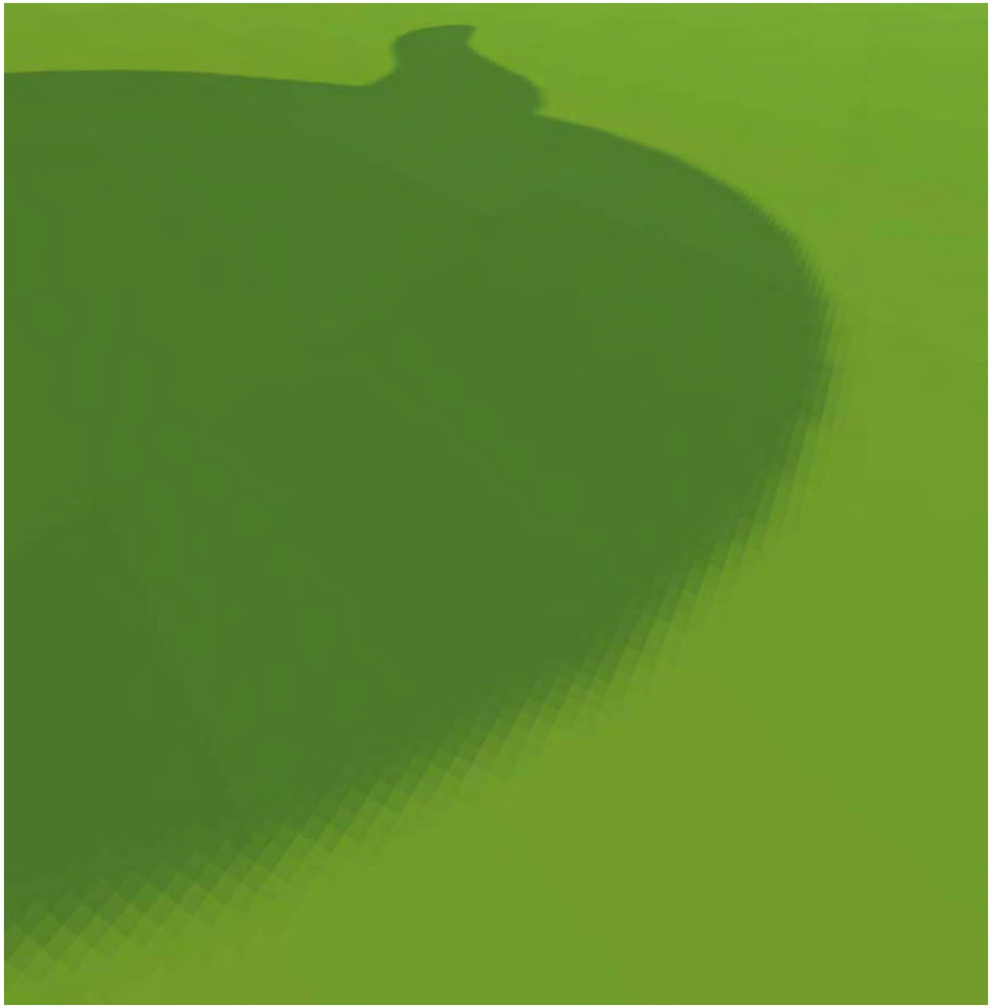


50.2	50.0	50.0
50.1	1.2	1.1
1.3	1.4	1.2

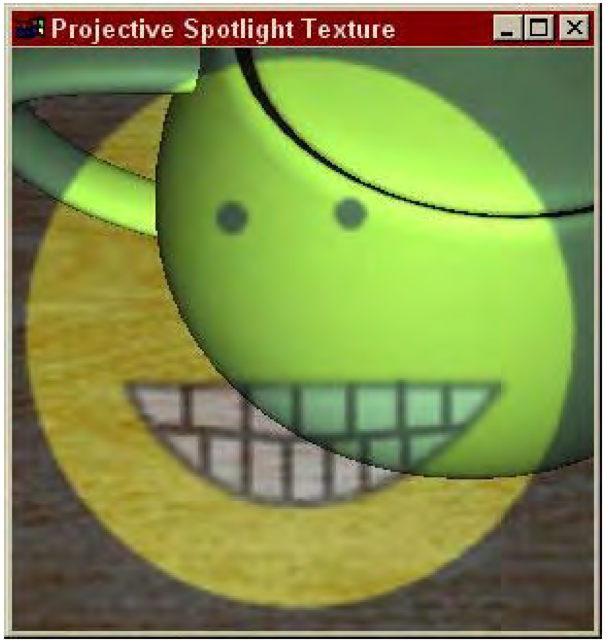
0	0	0
0	1	1
1	1	1

Percentage Closer Filtering

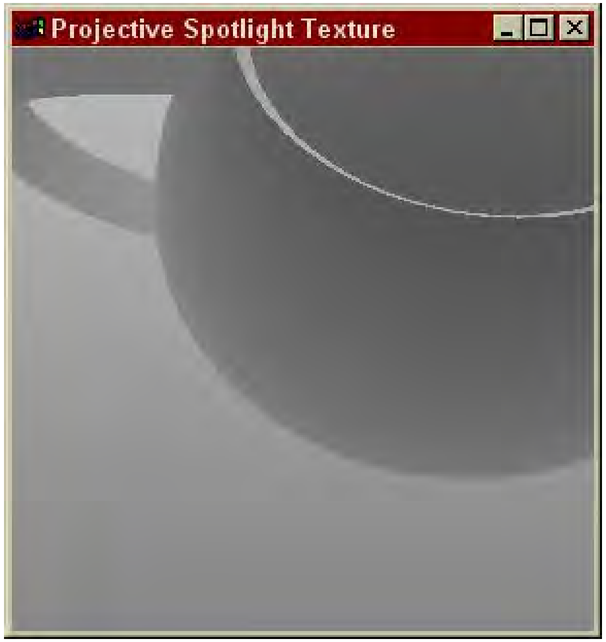
- 5x5 samples
- Nice antialiased shadow
- Using a bigger filter produces fake soft shadows
- Setting bias is tricky



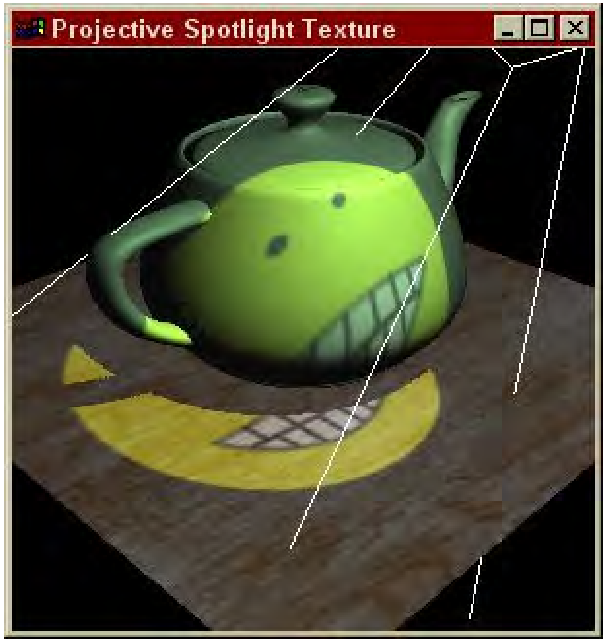
Projective Texturing + Shadow Map



Light’s View



Depth/Shadow Map



Eye’s View

Images from Cass Everitt et al.,
“Hardware Shadow Mapping”
NVIDIA SDK White Paper

Shadow Map Demo

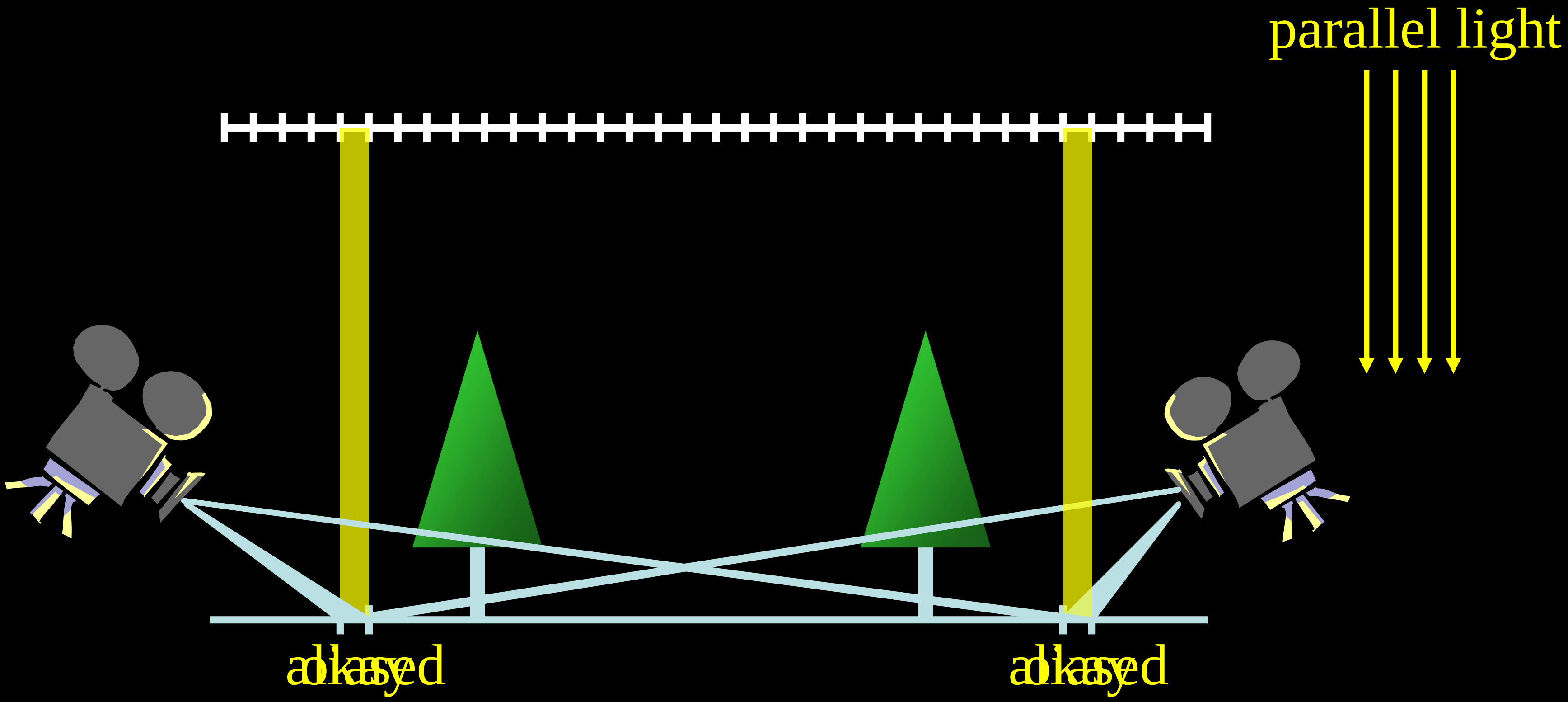
- Demo1 hardware shadow map
- Demo2 hardware shadow map

Perspective Shadow Maps

- Change the projection for the light source
  - Adapt the resolution of the shadow map according to the view
  - SIGGRAPH 2002 (Stamminger & Drettakis)
  - More details at the publication page

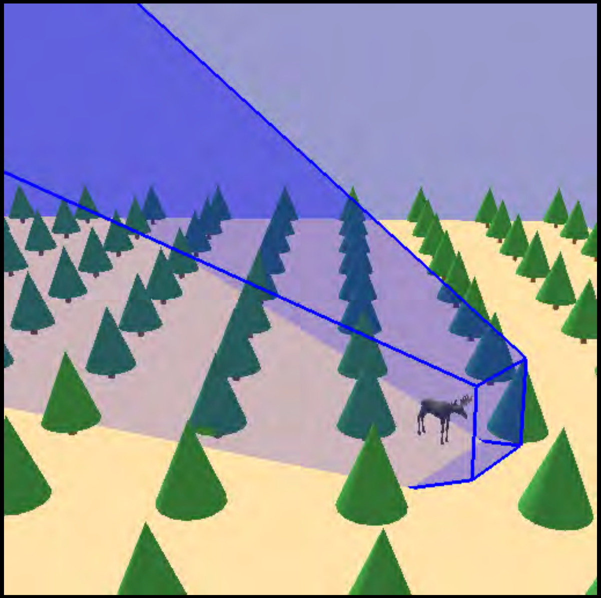
shadow map aliasing

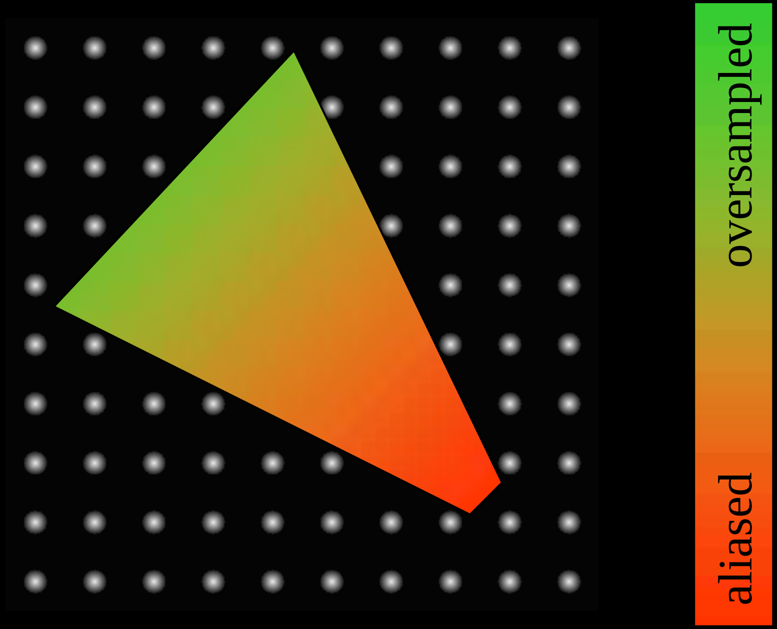
- perspective aliasing



shadow map aliasing

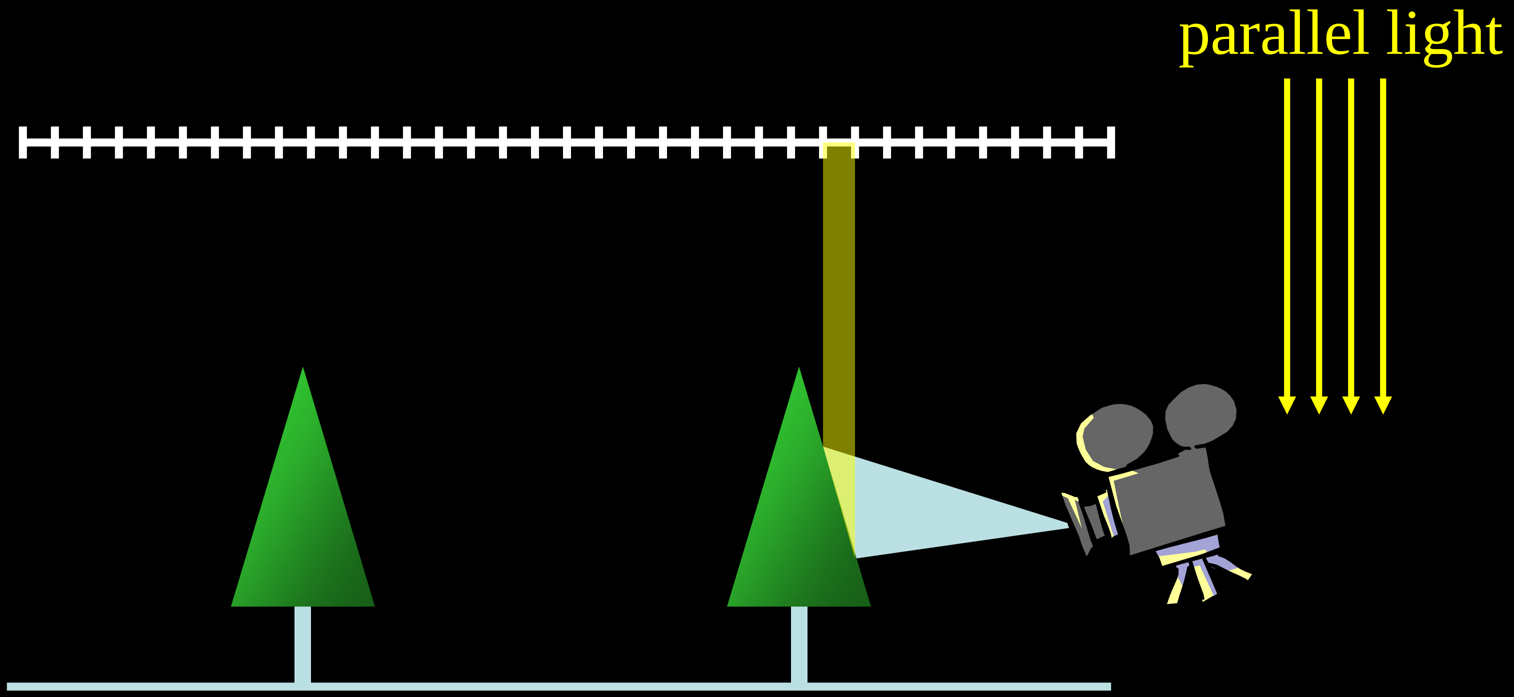
- perspective aliasing
  - smooth transition





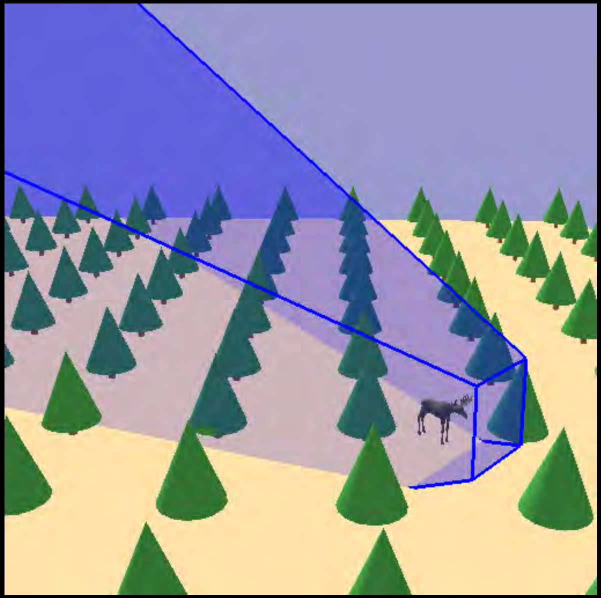
shadow map aliasing

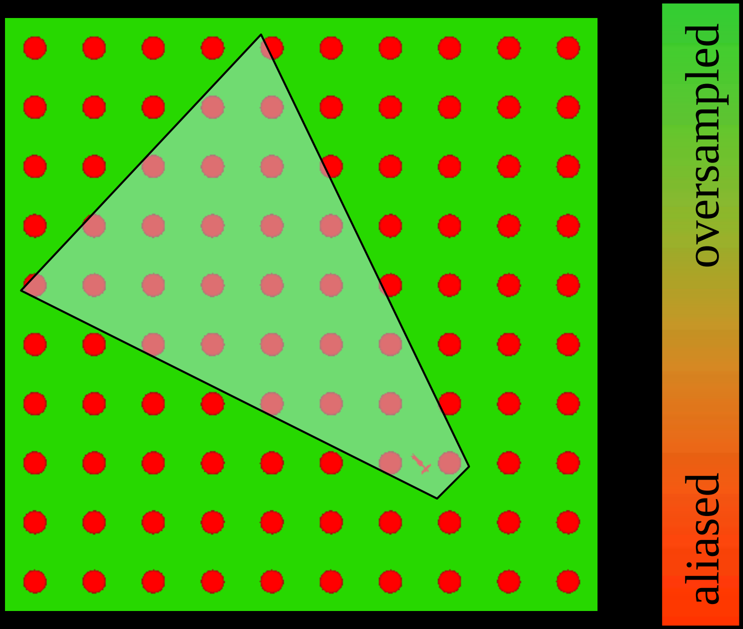
- projection aliasing



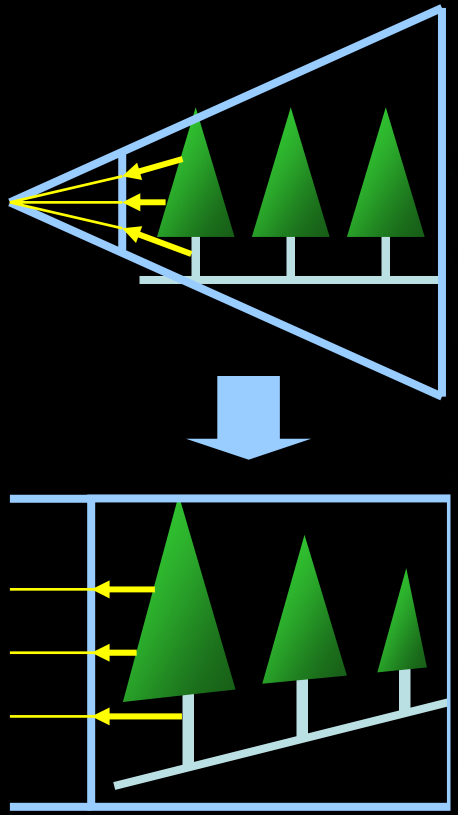
shadow map aliasing

- projection aliasing
  - very local



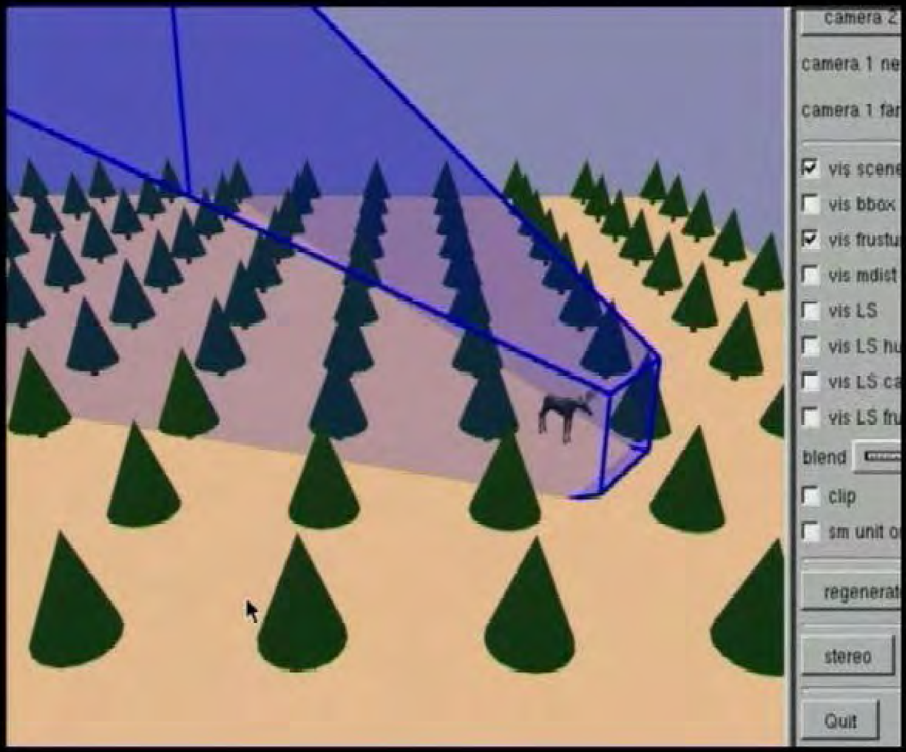


perspective transformation



world space

post-perspective



☑ vis scene

□ vis bbox

☑ vis frustu

□ vis mdist

□ vis LS

□ vis LS hu

□ vis LS ca

□ vis LS fru

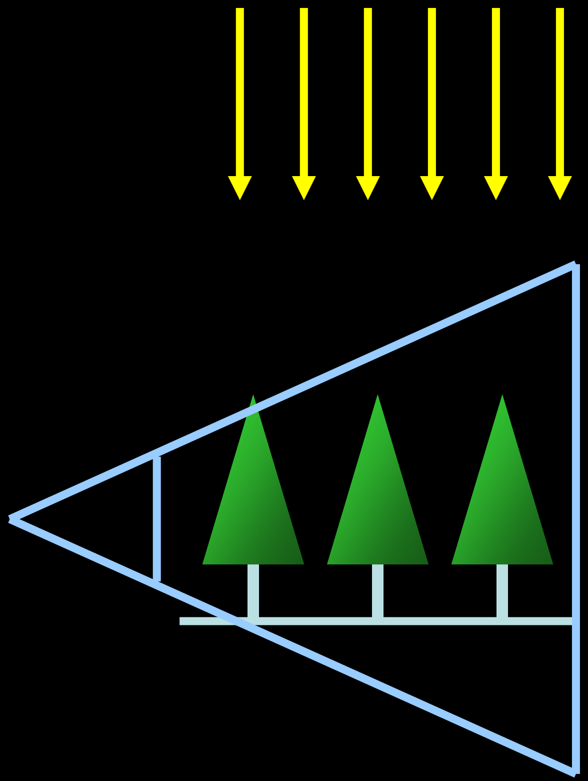
blend

□ clip

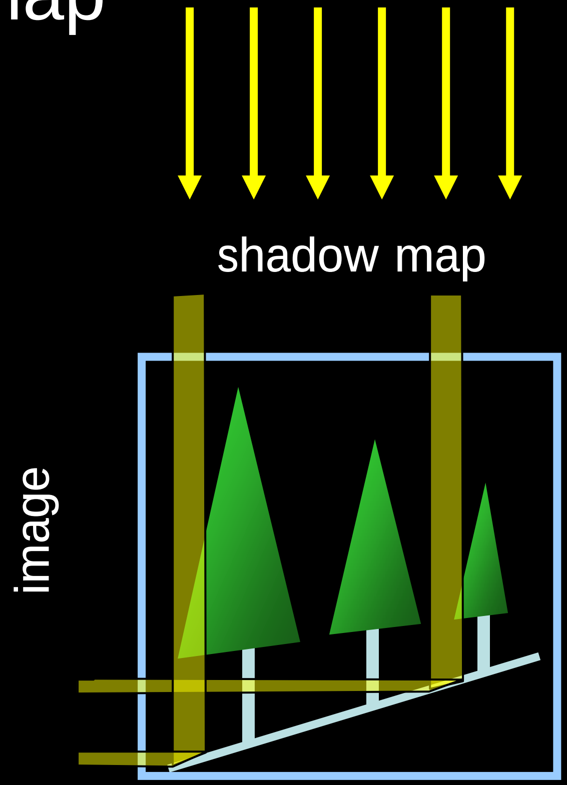
□ sm unit o

perspective shadow map

- standard shadow map

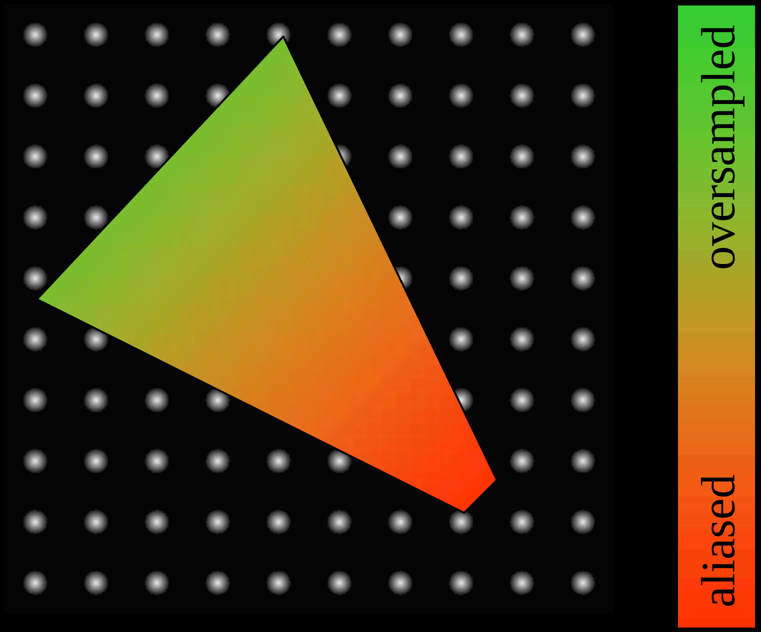


- perspective shadow map



perspective shadow map

- standard shadow map
- perspective shadow map

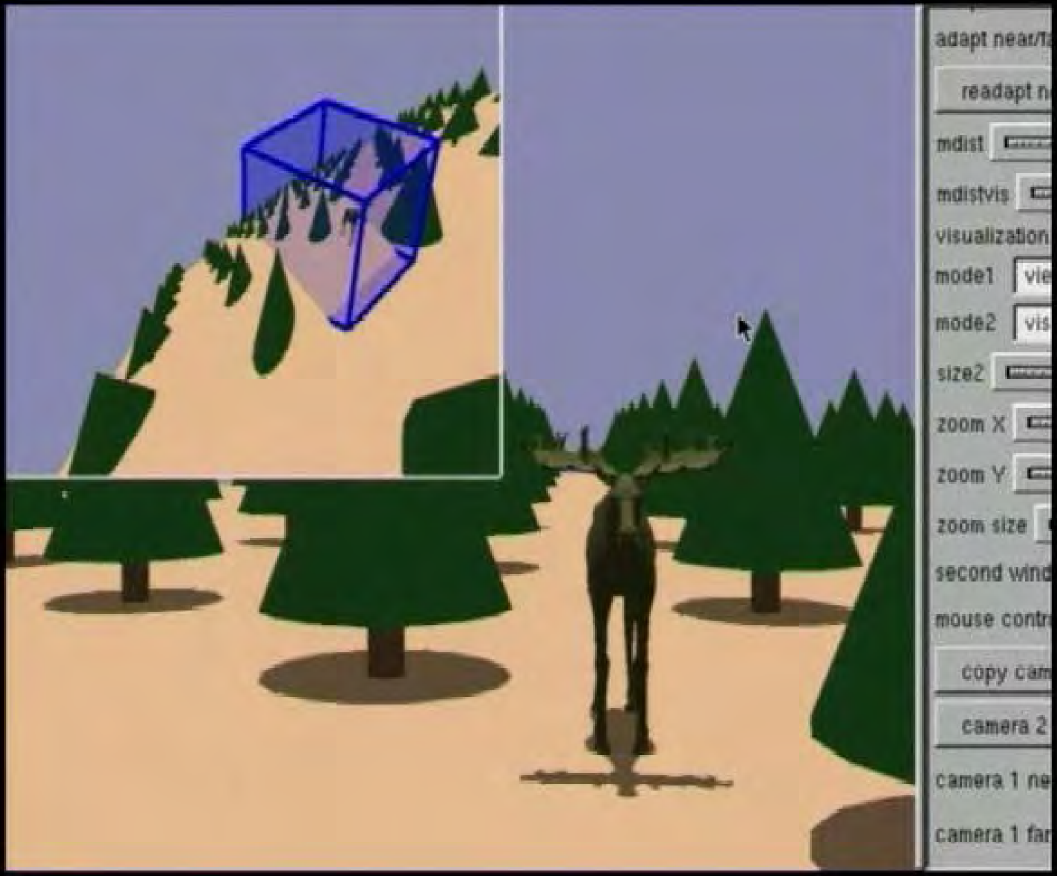


perspective shadow map

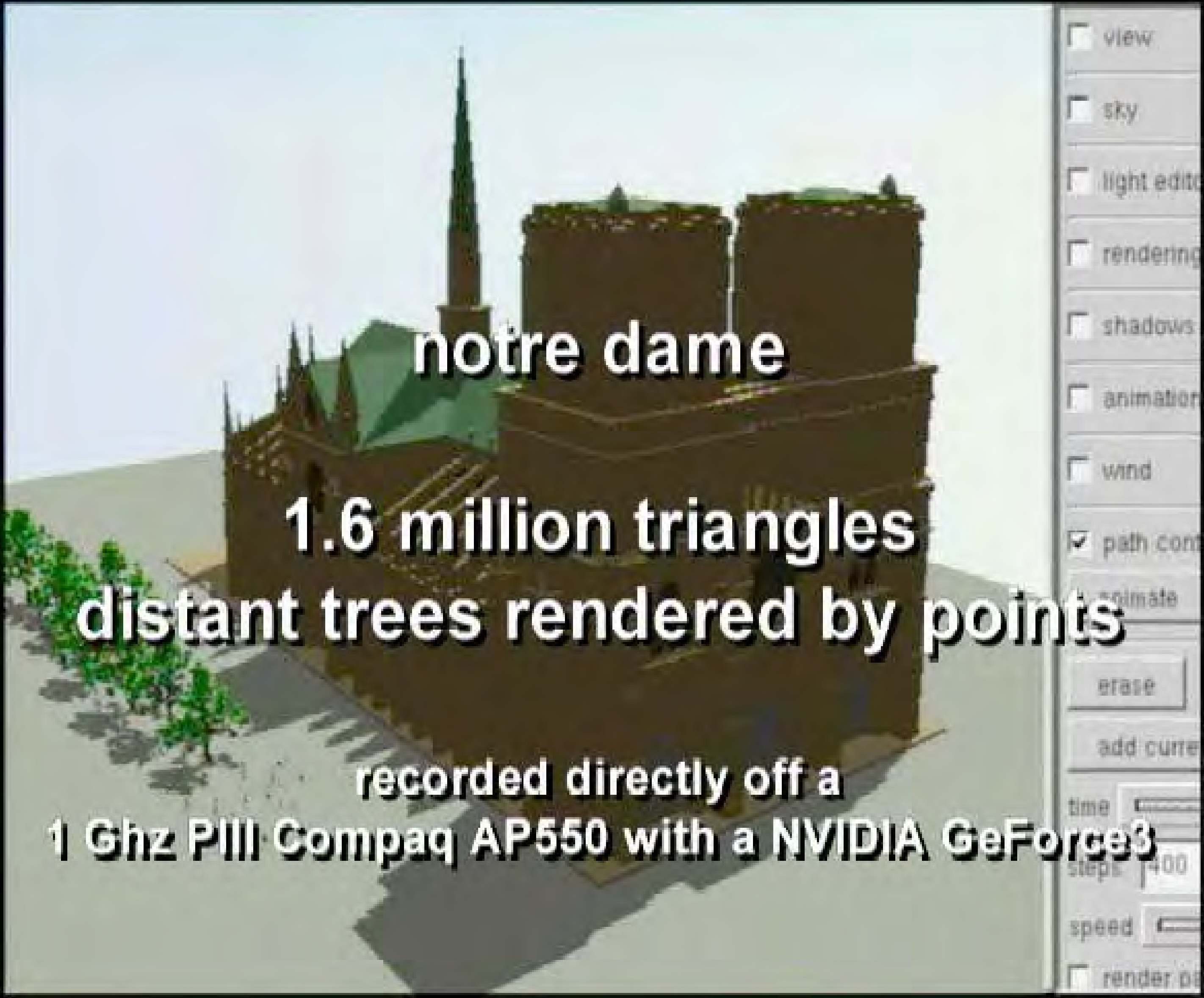
- shadow map in post-perspective space
- just another shadow map projection
- reduces perspective aliasing
- regeneration per frame necessary

light source transformation

- parallel light becomes point light



Results



notre dame

1.6 million triangles
distant trees rendered by points

recorded directly off a
1 Ghz PIII Compaq AP550 with a NVIDIA GeForce3

Perspective Shadow Map Demo

- Demo

Shadows in Production

- Often use shadow maps
- Ray casting as fallback in case of robustness issues

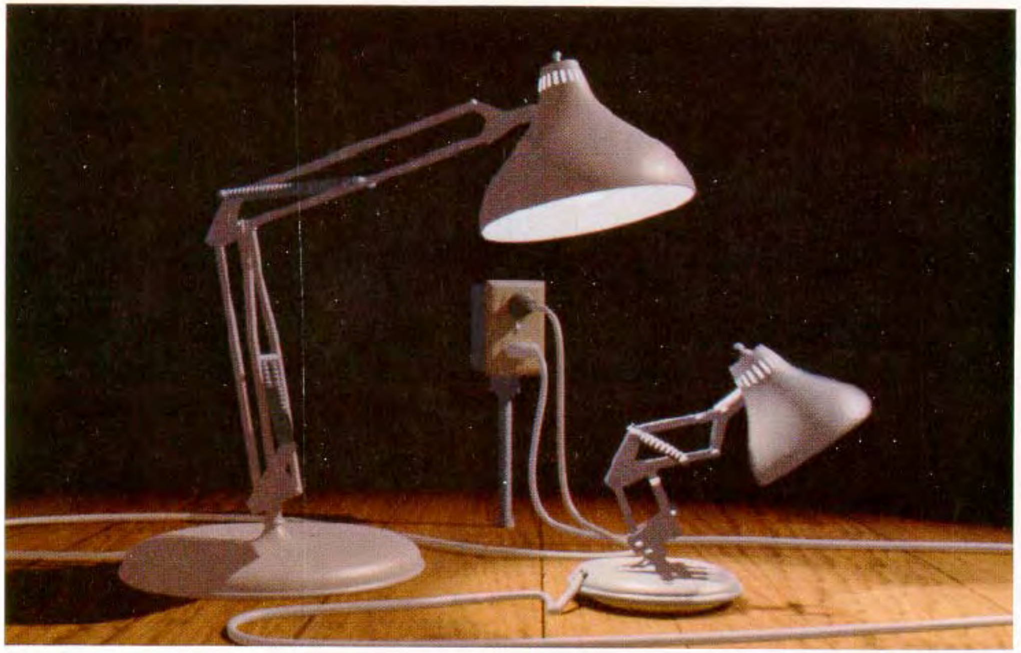


Figure 12. Frame from *Luxo Jr.*

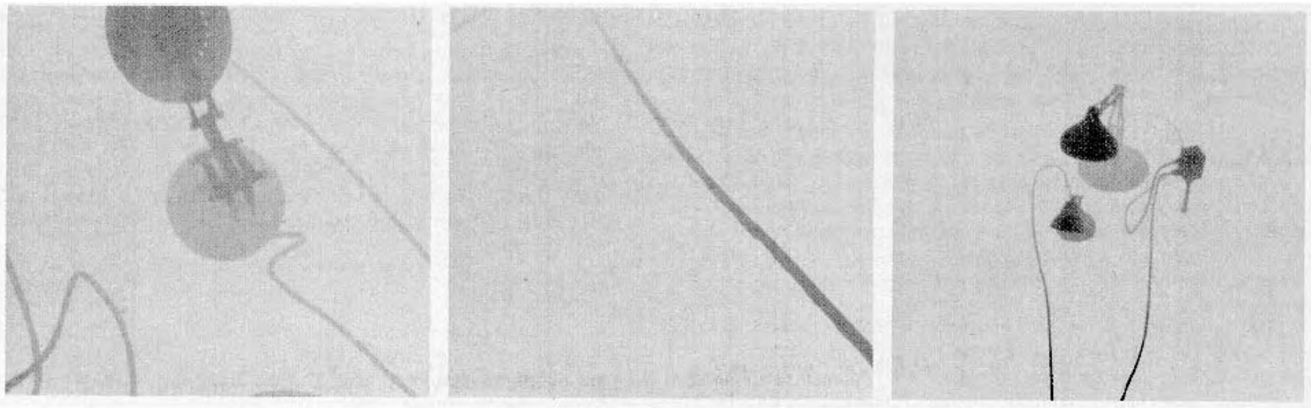
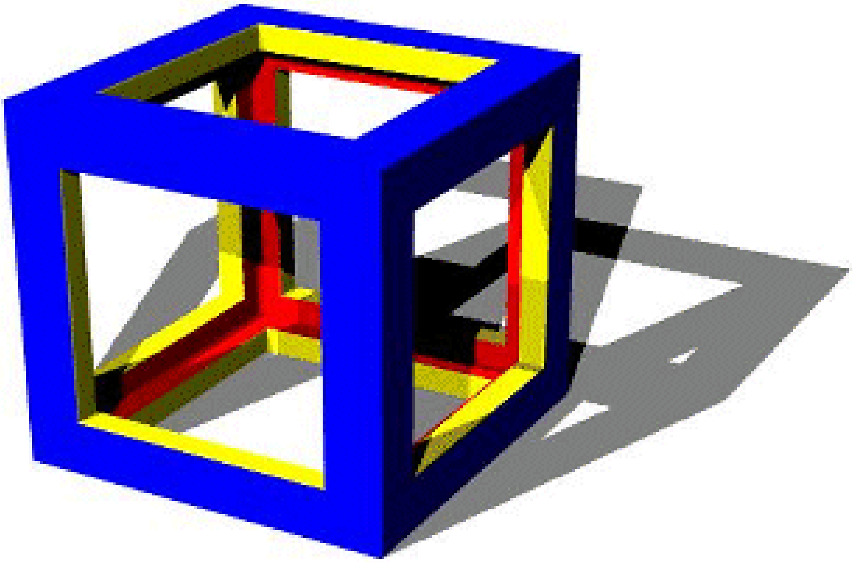


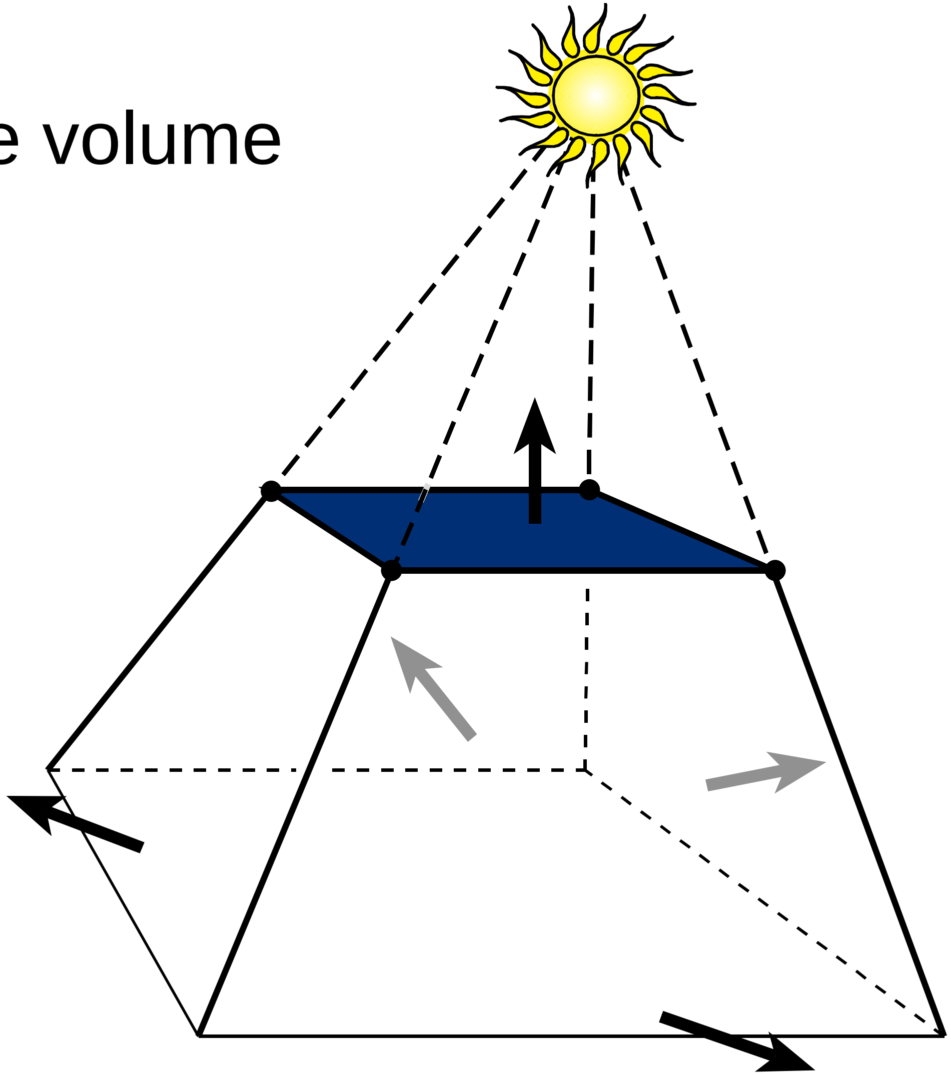
Figure 13. Shadow maps from *Luxo Jr.*

- Why are Shadows Important?
- Shadows & Soft Shadows in Ray Tracing
- Planar Shadows
- Shadow Maps
- Shadow Volumes
  - – The Stencil Buffer



Shadow Volumes

- Explicitly represent the volume of space in shadow
- For each polygon
  - Pyramid with point light as apex
  - Include polygon to cap
- Shadow test similar to clipping

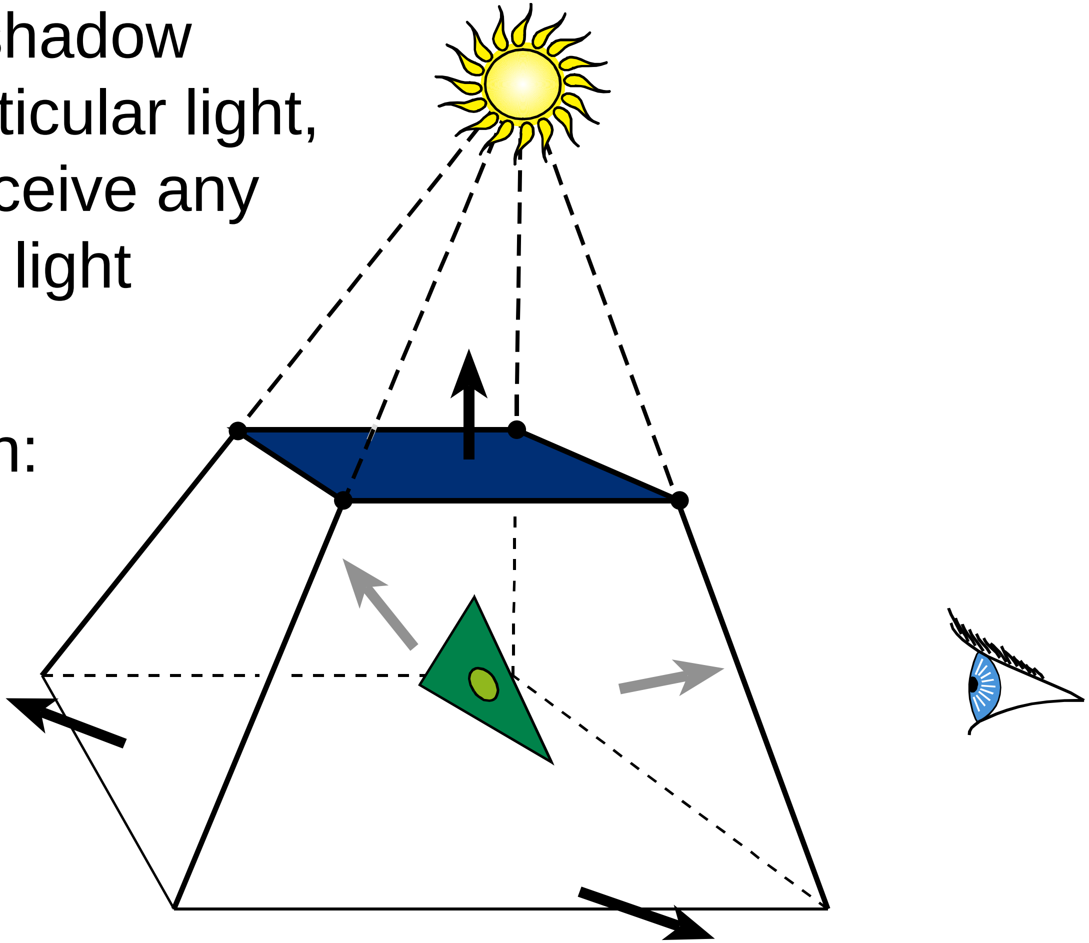


Shadow Volumes

- If a point is inside a shadow volume cast by a particular light, the point does not receive any illumination from that light

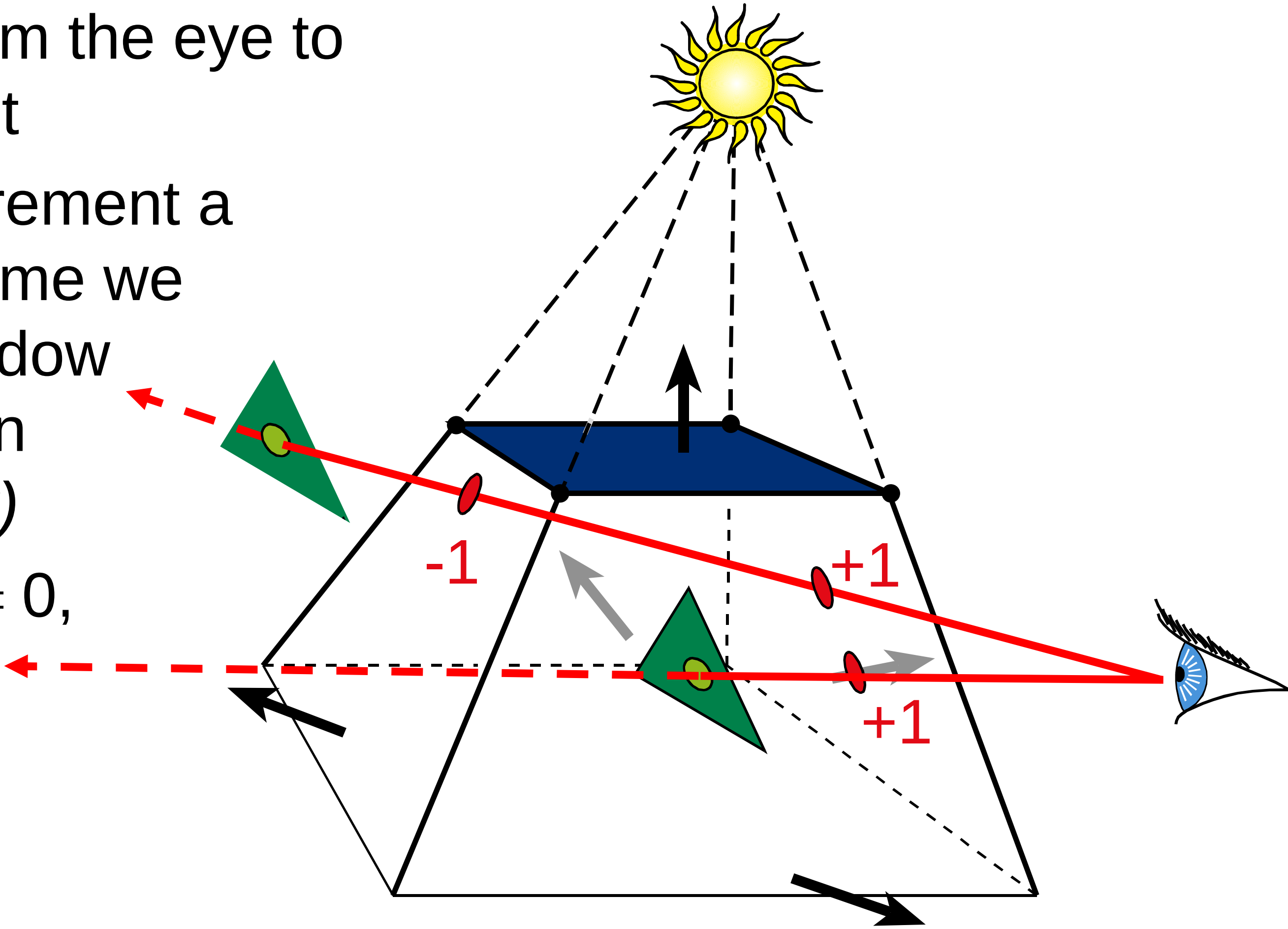
- Naive implementation:

  #polygons * #lights



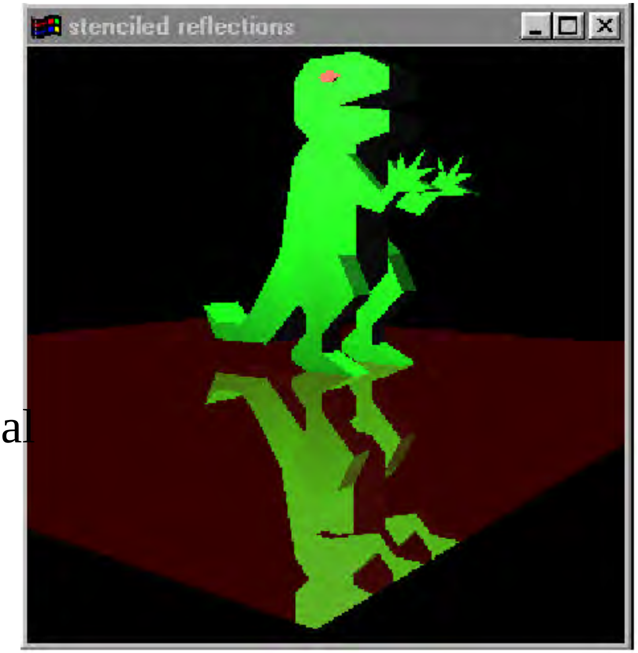
Shadow Volumes

- Shoot a ray from the eye to the visible point
- Increment/decrement a counter each time we intersect a shadow volume polygon *(check z buffer)*
- If the counter ≠ 0, the point is in shadow



Stencil Buffer

- Tag pixels in one rendering pass to control their update in subsequent rendering passes
- "For all pixels in the frame buffer" →
  "For all *tagged* pixels in the frame buffer"
- Used for real-time mirrors
  (& other reflective surfaces),
  shadows & more!



from NVIDIA's stencil buffer tutorial
(http://developer.nvidia.com)

Stencil Buffer

- Can specify different rendering operations for each of the following stencil tests:
  - stencil test fails
  - stencil test passes & depth test fails
  - stencil test passes & depth test passes

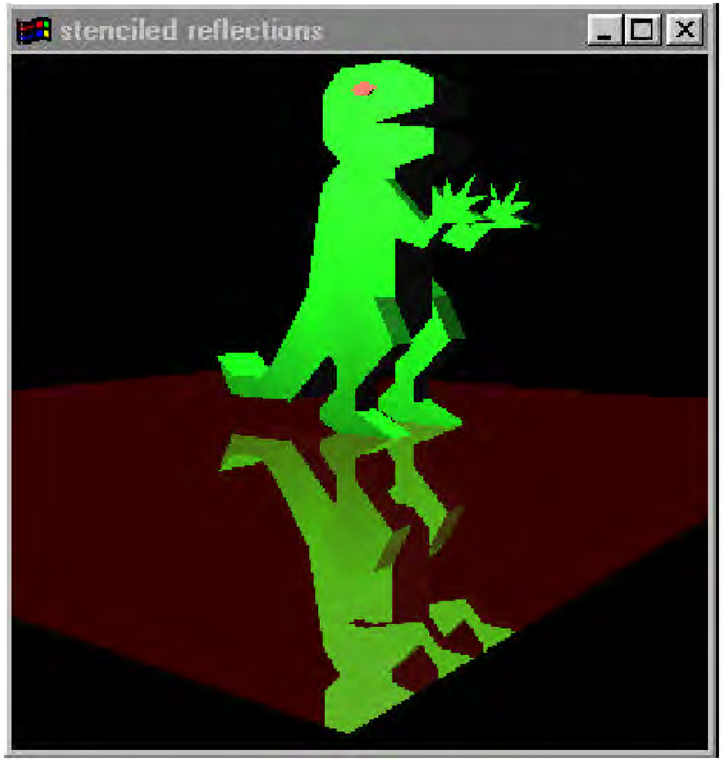


image from NVIDIA's stencil buffer tutorial (http://developer.nvidia.com)

Shadow Volumes w/ the Stencil Buffer

Initialize stencil buffer to 0

Draw scene with ambient light only

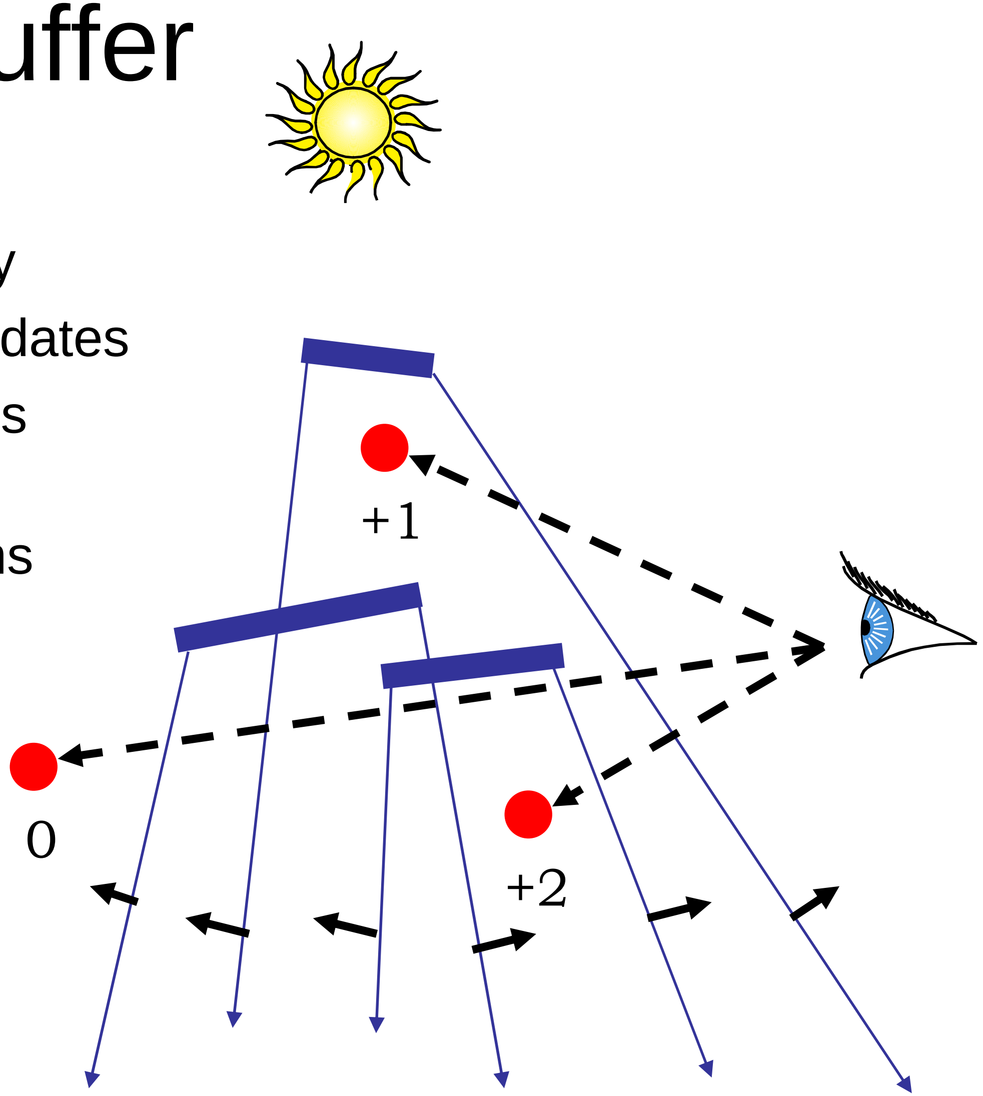
Turn off frame buffer & z-buffer updates

Draw front-facing shadow polygons
    If z-pass → increment counter

Draw back-facing shadow polygons
    If z-pass → decrement counter

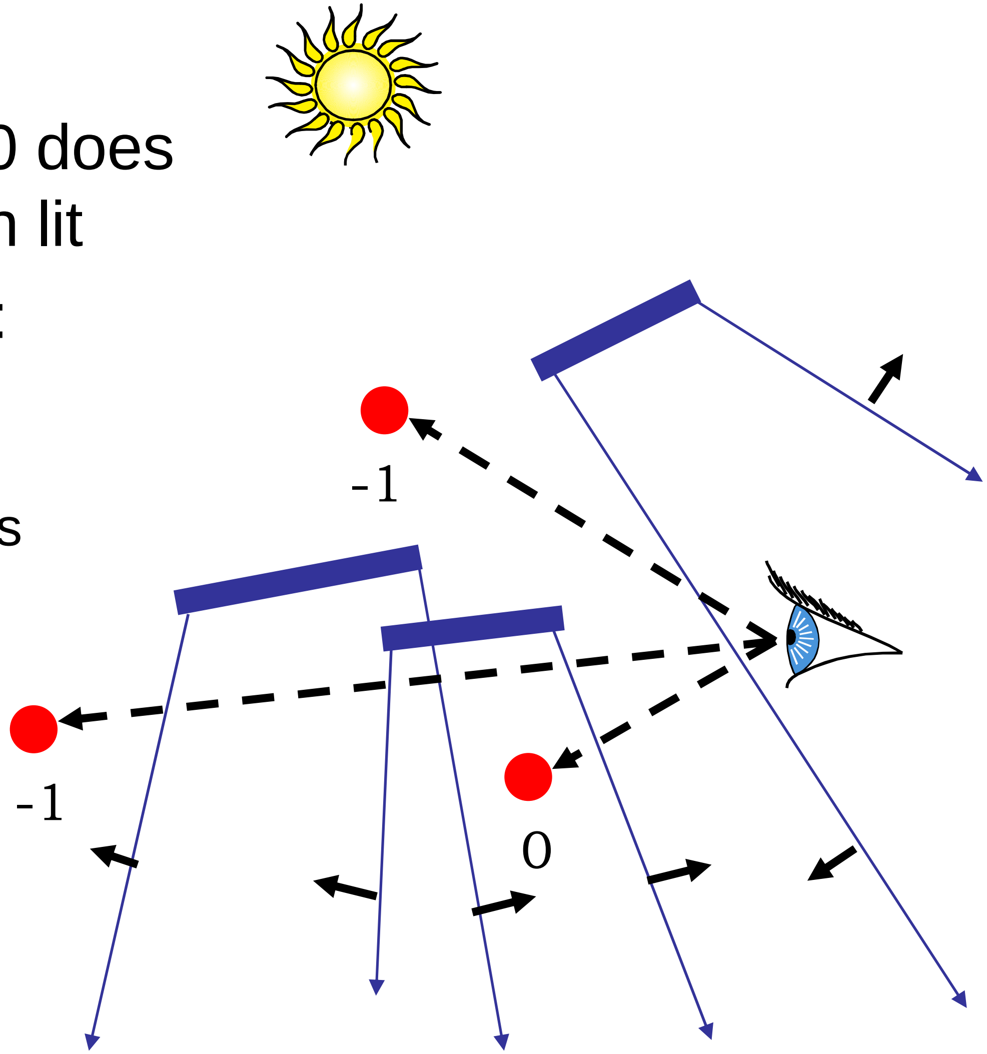
Turn on frame buffer updates

Turn on lighting and
    redraw pixels with
    counter = 0



If the Eye is in Shadow...

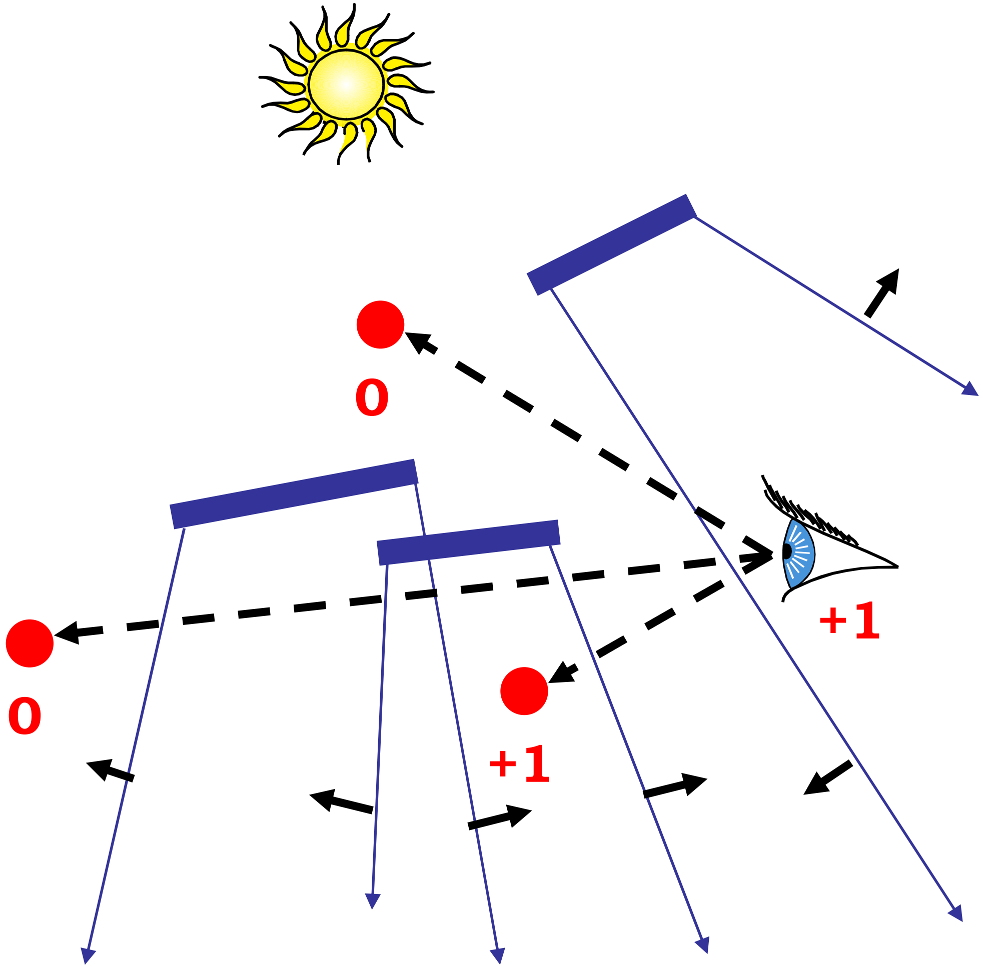
- ... then a counter of 0 does not necessarily mean lit
- 3 Possible Solutions:
  1. Explicitly test eye point with respect to all shadow volumes
  2. Clip the shadow volumes to the view frustum
  3. "Z-Fail" shadow volumes



1. Test Eye with Respect to Volumes

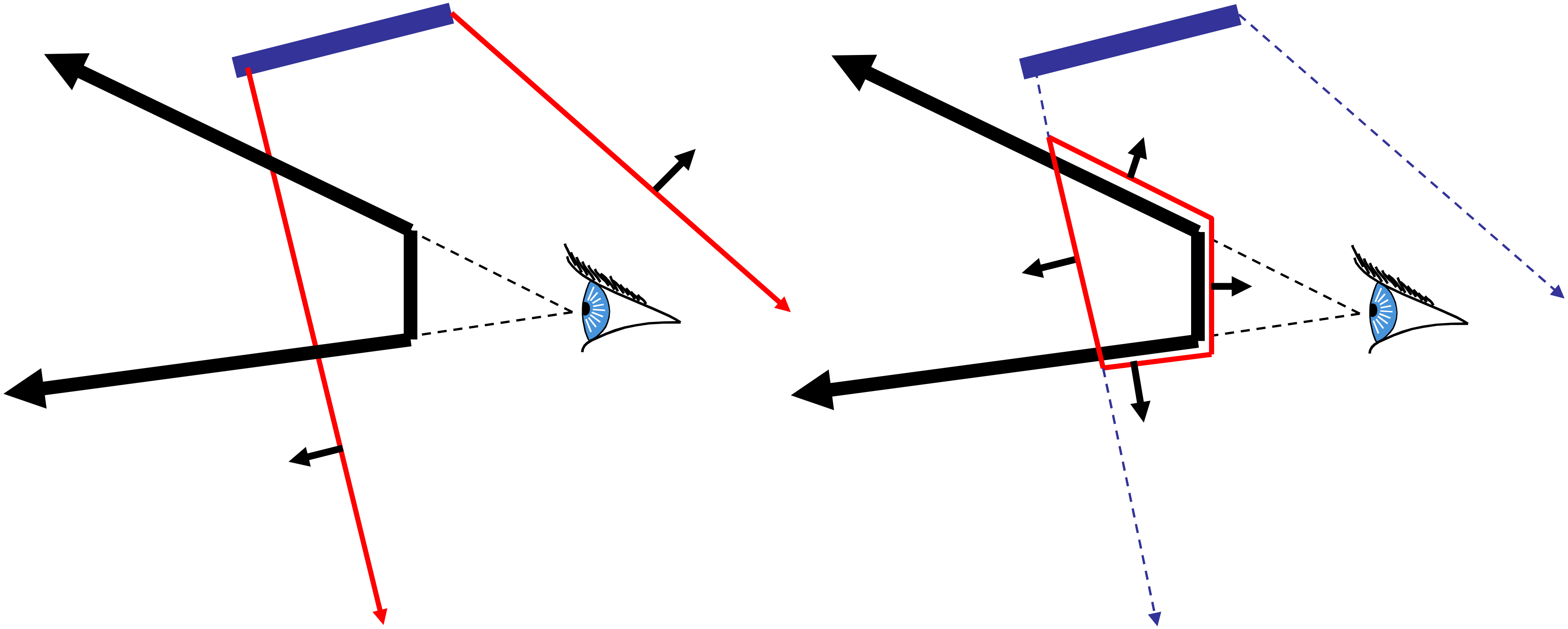
- Adjust initial counter value

Expensive



2. Clip the Shadow Volumes

- Clip the shadow volumes to the view frustum and include these new polygons
- *Messy CSG*



3. "Z-Fail" Shadow Volumes

Start at infinity

...

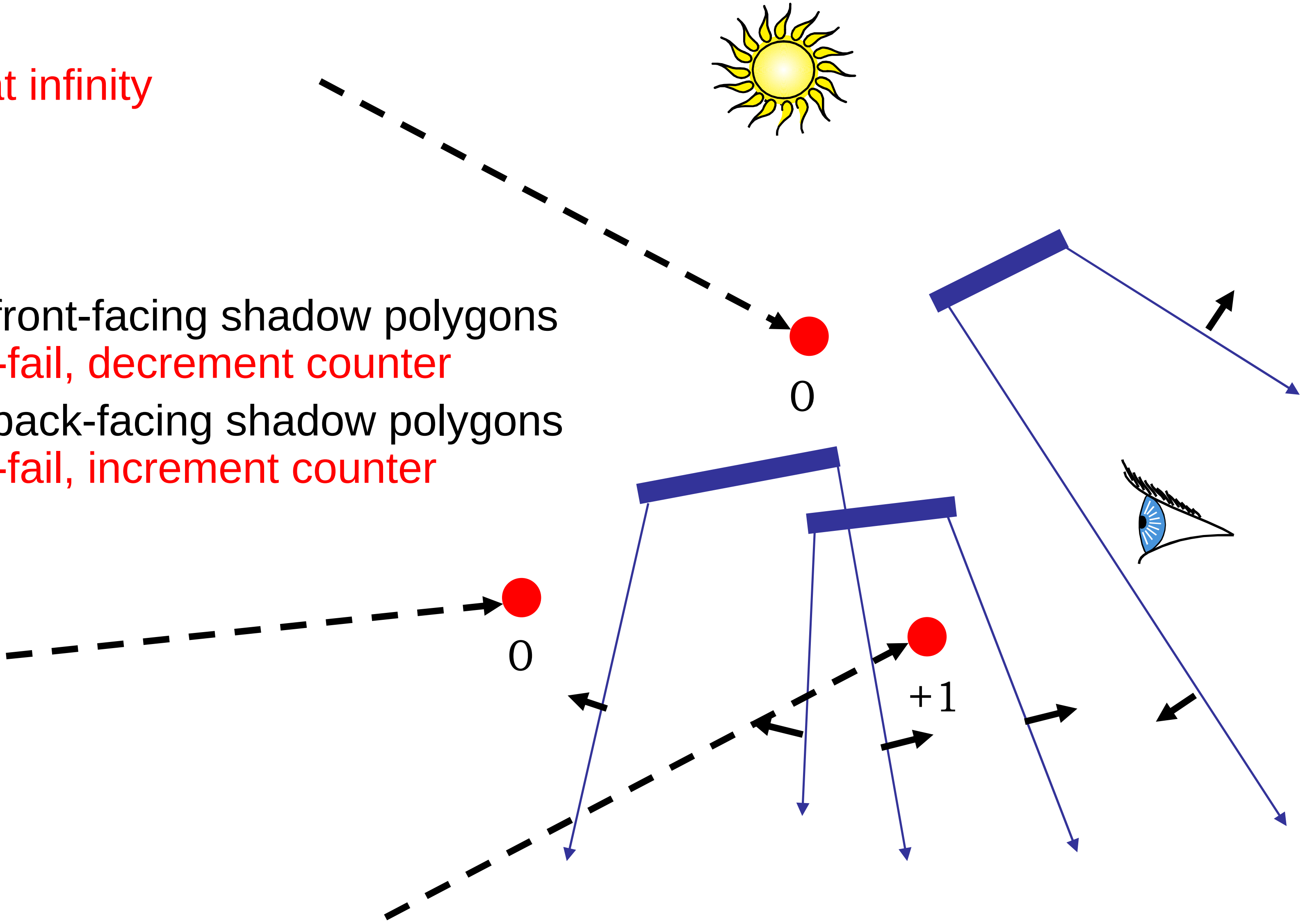
Draw front-facing shadow polygons

- If z-fail, decrement counter

Draw back-facing shadow polygons

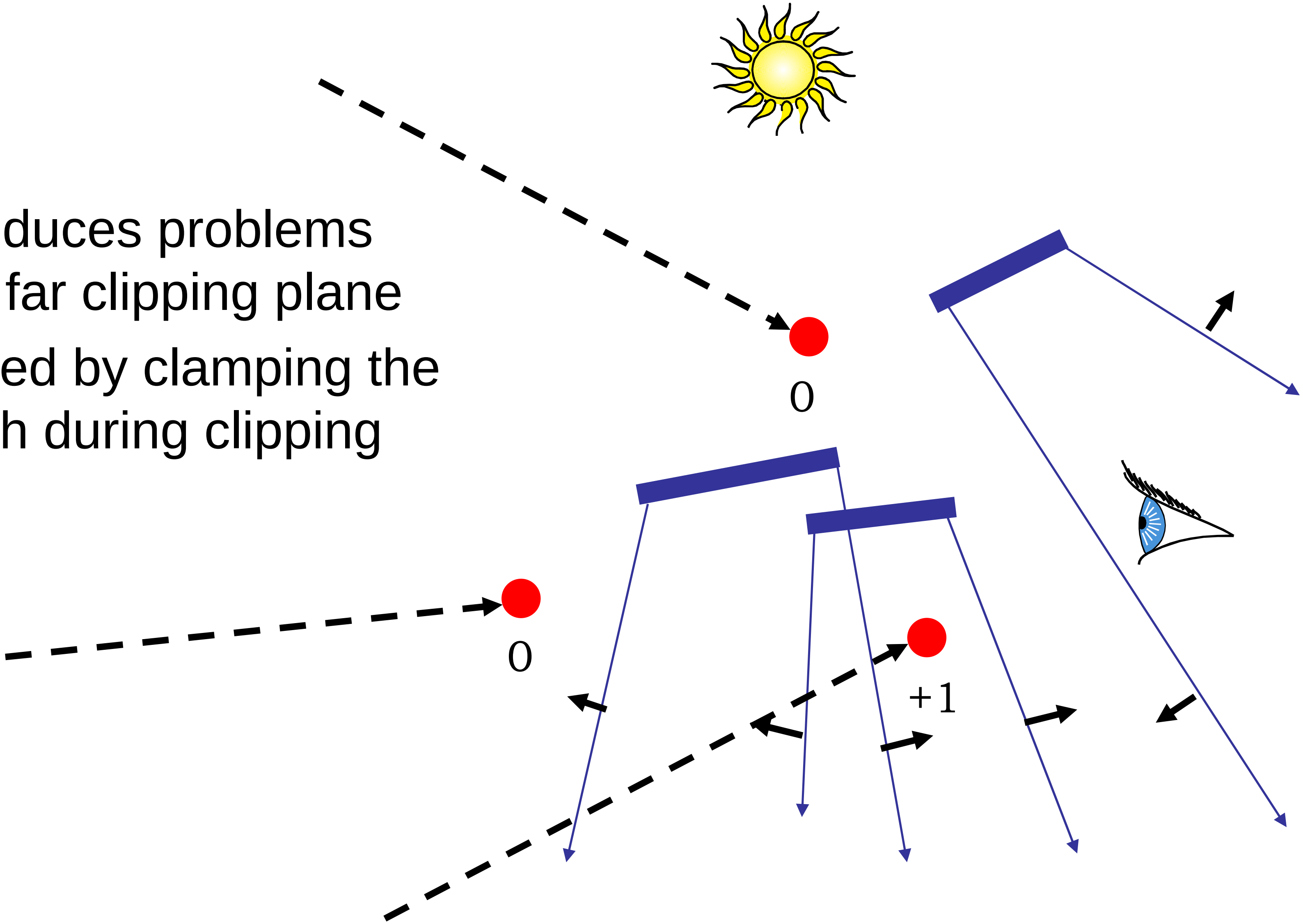
- If z-fail, increment counter

...



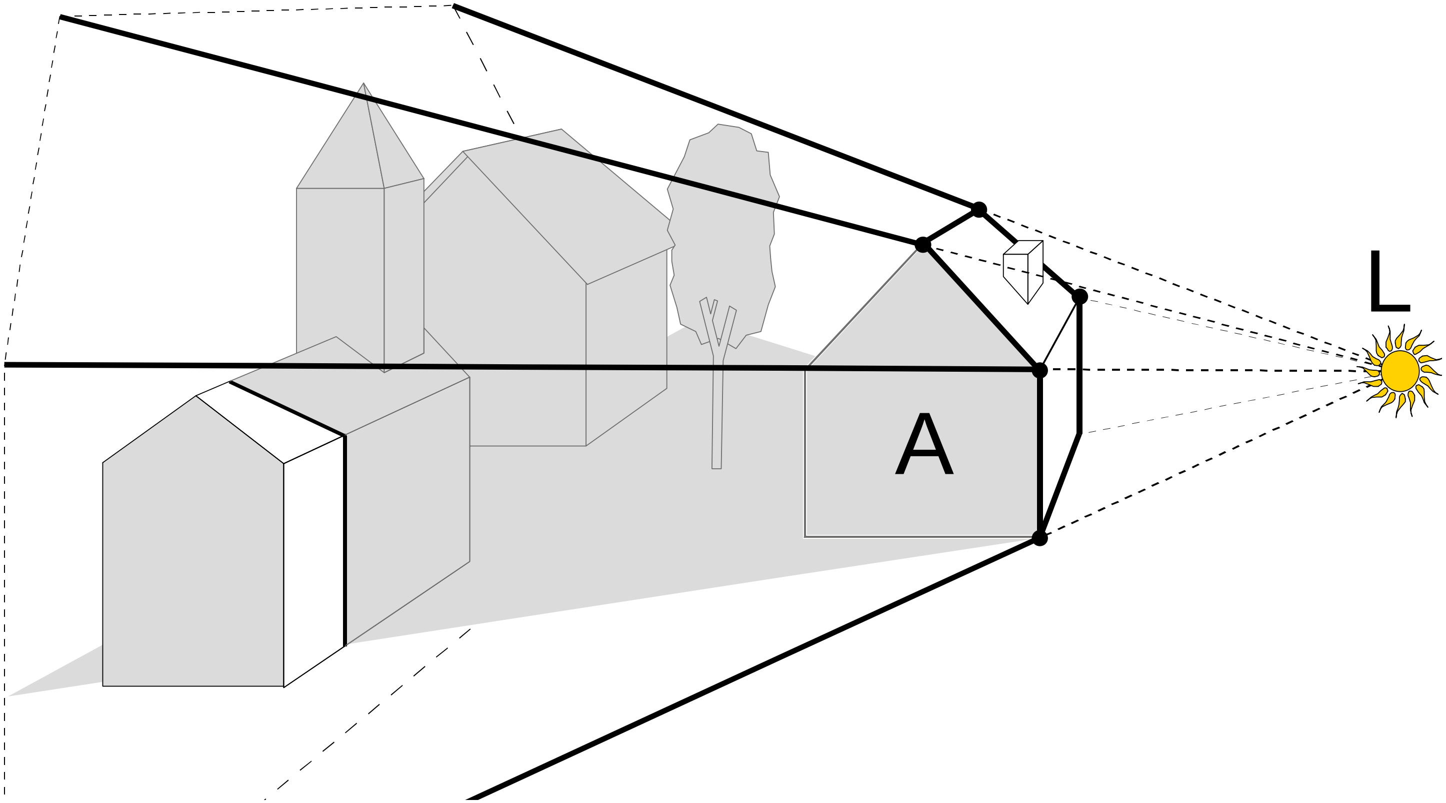
3. "Z-Fail" Shadow Volumes

- Introduces problems with far clipping plane
- Solved by clamping the depth during clipping



Optimizing Shadow Volumes

- Use silhouette edges only (edge where a back-facing & front-facing polygon meet)



Limitations of Shadow Volumes

- Introduces a lot of new geometry
- Expensive to rasterize long skinny triangles
- Limited precision of stencil buffer (counters)
  - for a really complex scene/object,
    the counter can overflow
- Objects must be watertight to use silhouette trick
- Rasterization of polygons sharing an edge
  must not overlap & must not have gap

Shadow Volume Demo

- Stencil buffer shadow volume demo

Global Illumination:
Radiosity and
Monte Carlo Methods

Today

- Radiosity methods
  - Why Radiosity
  - Global Illumination: The Rendering Equation
  - Radiosity Equation/Matrix
  - Calculating the Form Factors
  - Progressive Radiosity
- Monte Carlo methods
  - Expected value and variance
  - Analysis of Monte-Carlo integration
  - Monte-Carlo in graphics
  - Importance sampling
  - Stratified sampling
  - Global illumination
  - Advanced Monte-Carlo rendering

Radiosity

- Why Radiosity
  - The Cornell Box
  - Radiosity vs. Ray Tracing
- Global Illumination: The Rendering Equation
- Radiosity Equation/Matrix
- Calculating the Form Factors
- Progressive Radiosity

Rendering Recap

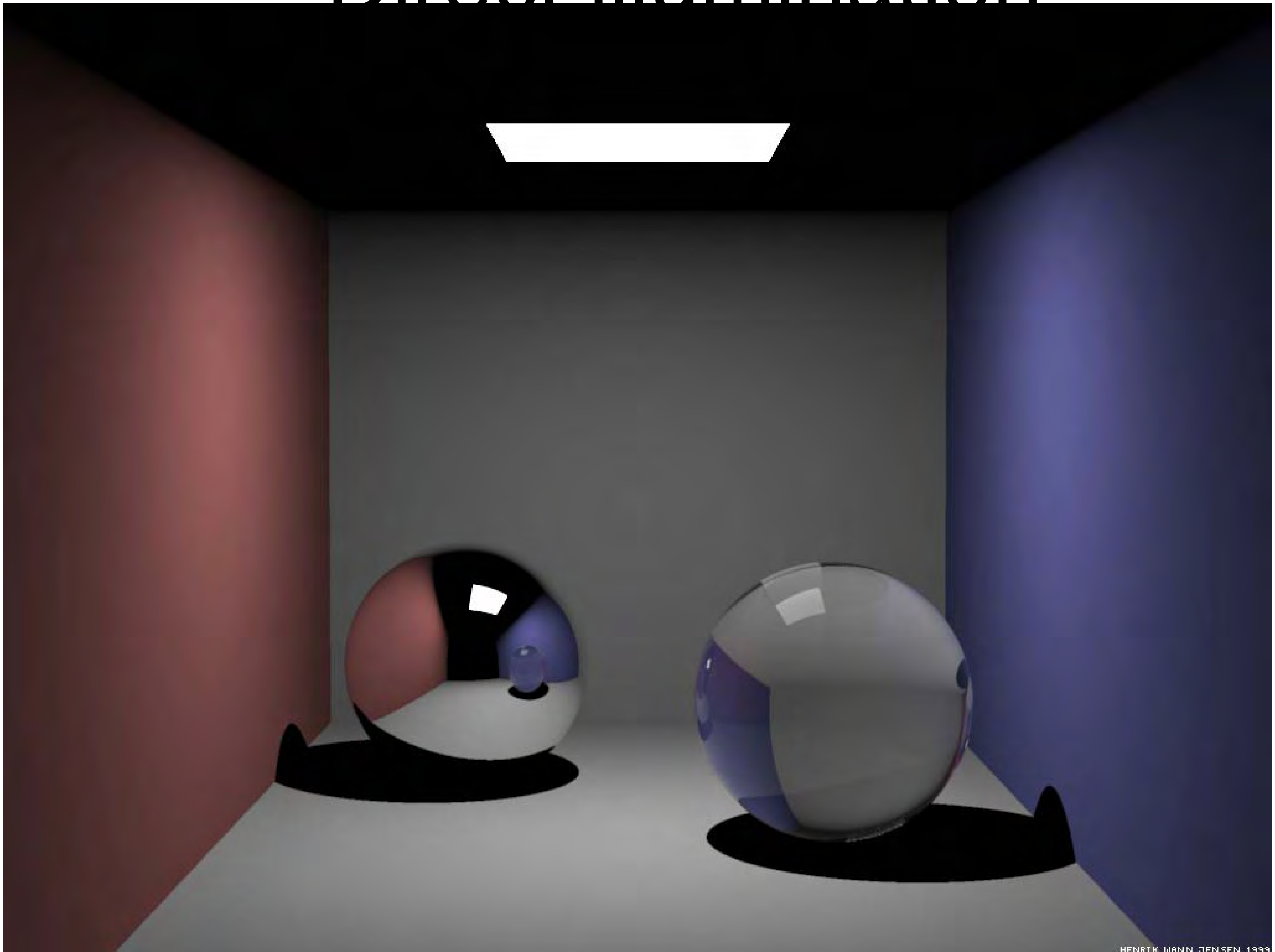
- Ray-tracing
  - For each pixel, for each object
- Graphics pipeline, scan conversion
  - For each object, for each pixel
- Local lighting models
  - Diffuse, Phong
- Shadows
  - Ray casting, shadow maps, shadow volumes
- Reflection, refraction

Why global illumination?

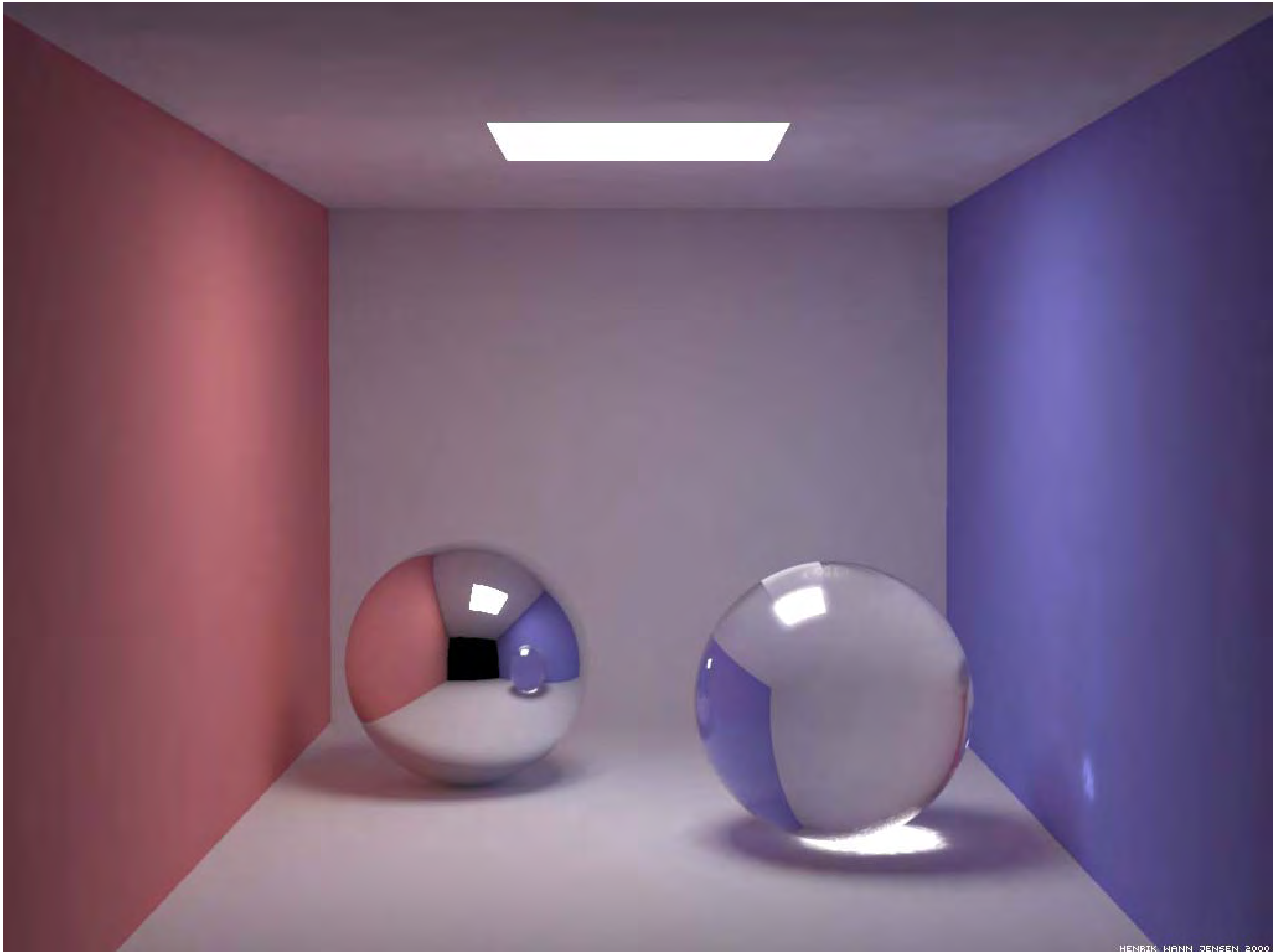
- Simulate all light inter-reflections (indirect lighting)
  - e.g. in a room, a lot of the light is indirect: it is reflected by walls.

- How have we dealt with this so far?
  - Ambient term to fake some uniform indirect light

Direct illumination



Global Illumination



Why Radiosity?

- Sculpture by John Ferren
- *Diffuse* panels

photograph:

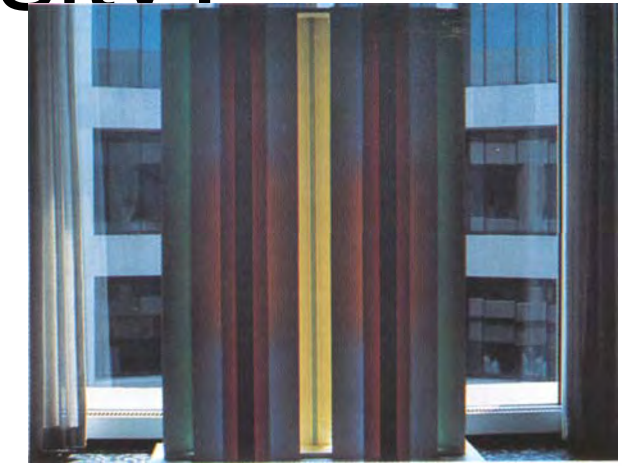
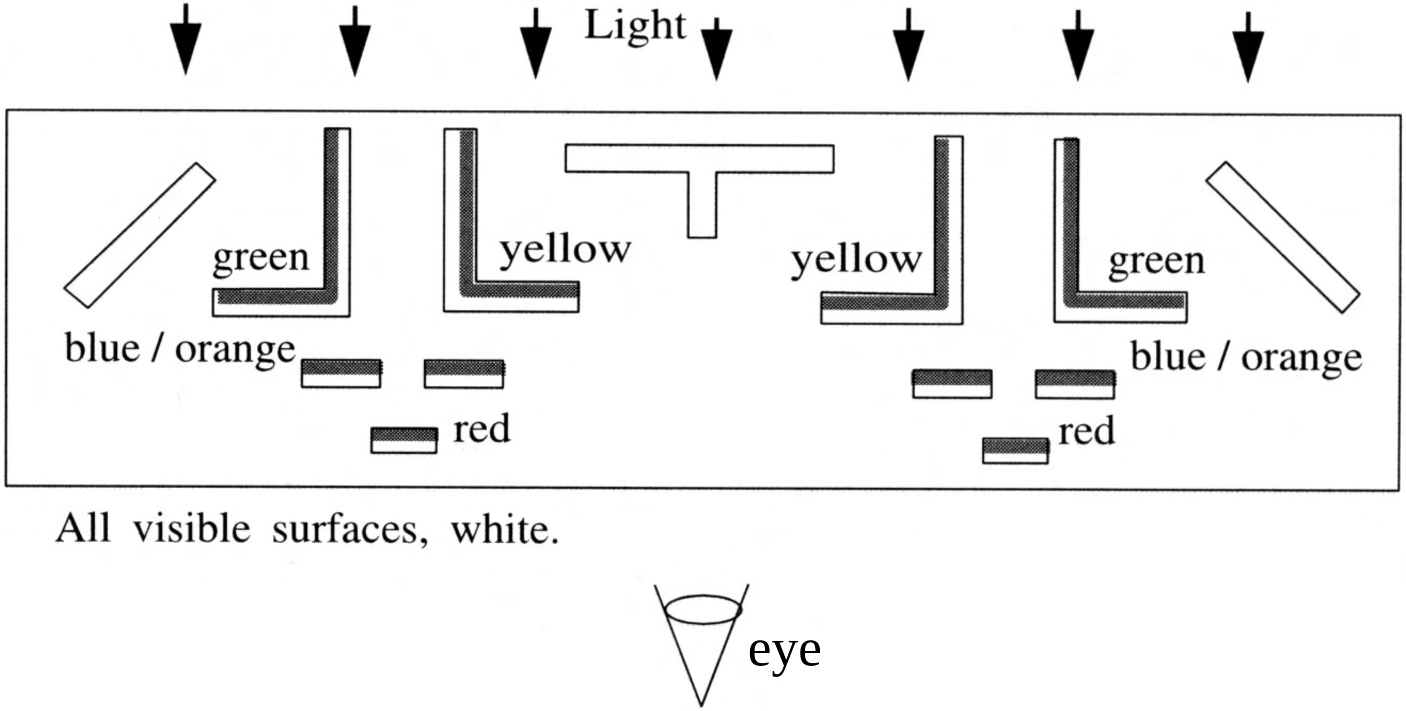
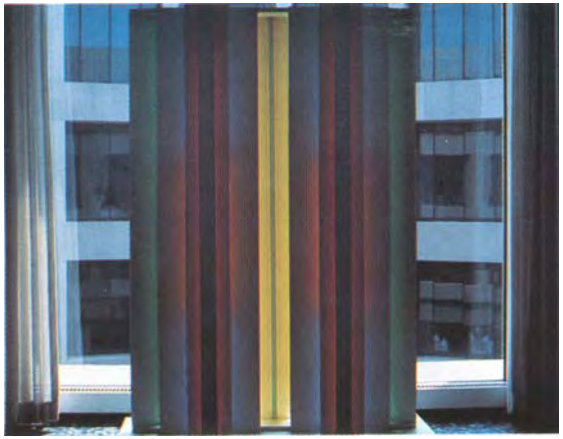


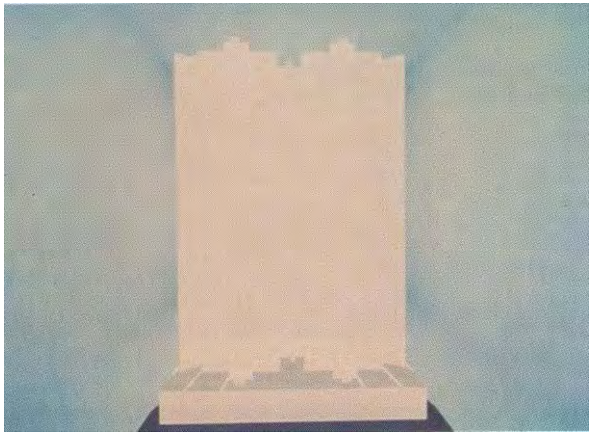
diagram from above:



Radiosity vs. Ray Tracing



Original sculpture by John Ferren lit by daylight from behind.



Ray traced image. A standard ray tracer cannot simulate the interreflection of light between diffuse surfaces.

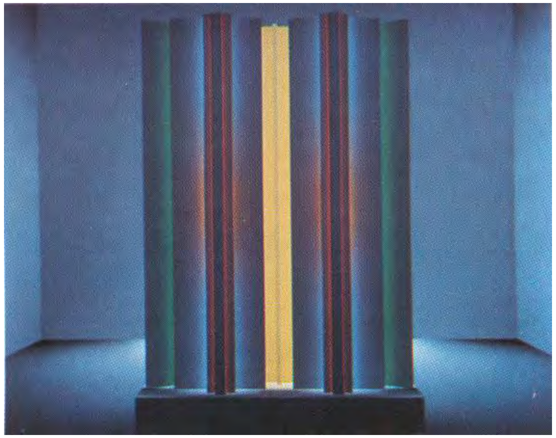


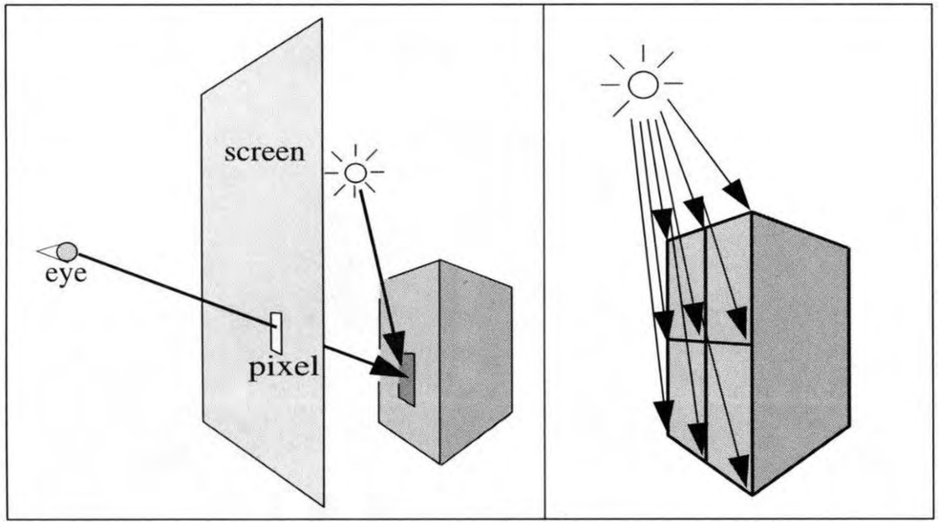
Image rendered with radiosity. note color bleeding effects.

Two approaches for global illumination

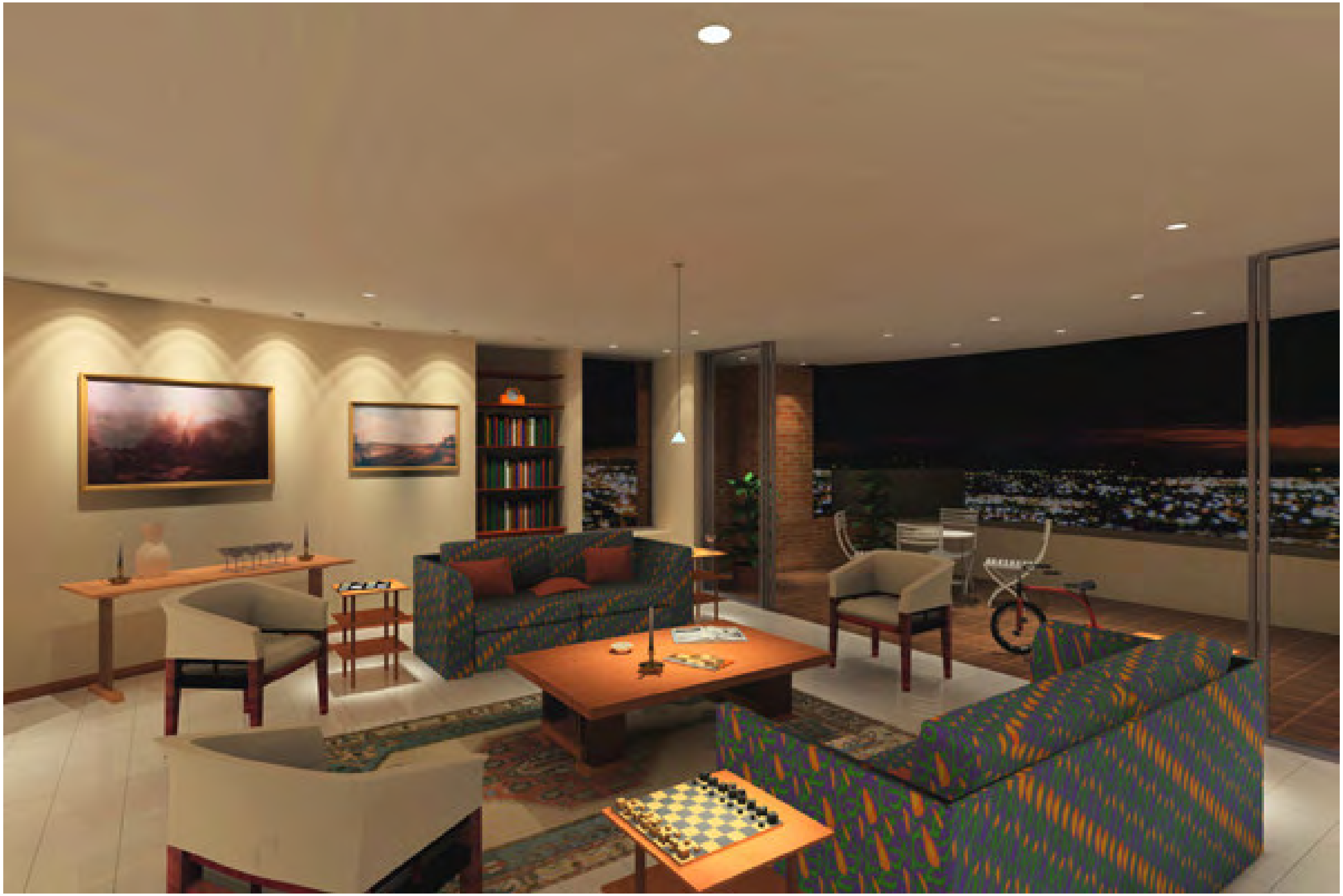
- Radiosity
  - View-independent
  - Diffuse only
- Monte-Carlo Ray-tracing
  - Send tons of indirect rays

Radiosity vs. Ray Tracing

- Ray tracing is an *image-space* algorithm
  - If the camera is moved, we have to start over
- Radiosity is computed in *object-space*
  - View-independent (just don't move the light)
  - Can pre-compute complex lighting to allow interactive walkthroughs



Radiosity

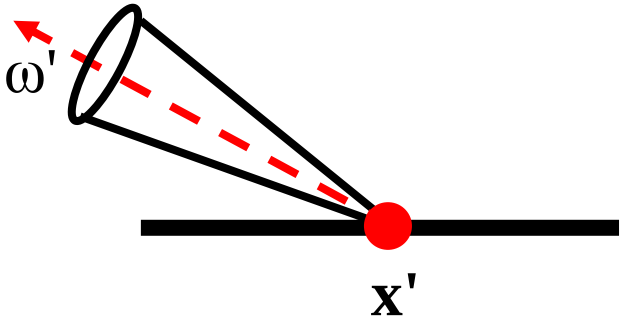


Lightscape `http://www.lightscape.com`

Today

- Why Radiosity
  - The Cornell Box
  - Radiosity vs. Ray Tracing
- Global Illumination: The Rendering Equation
- Radiosity Equation/Matrix
- Calculating the Form Factors
- Progressive Radiosity
- Advanced Radiosity

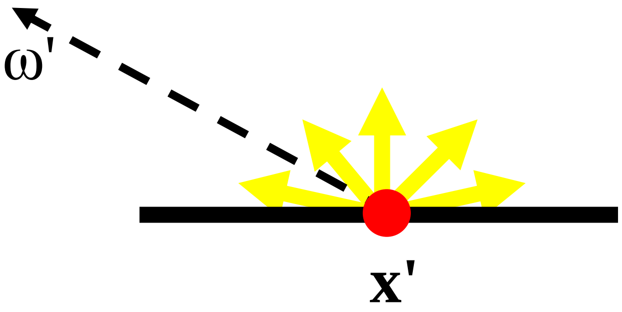
The Rendering Equation



$$L(x',\omega') = E(x',\omega') + \int \rho_{x'}(\omega,\omega')L(x,\omega)G(x,x')V(x,x')\,dA$$

$L(x',\omega')$ is the radiance from a point on a surface in a given direction ω'

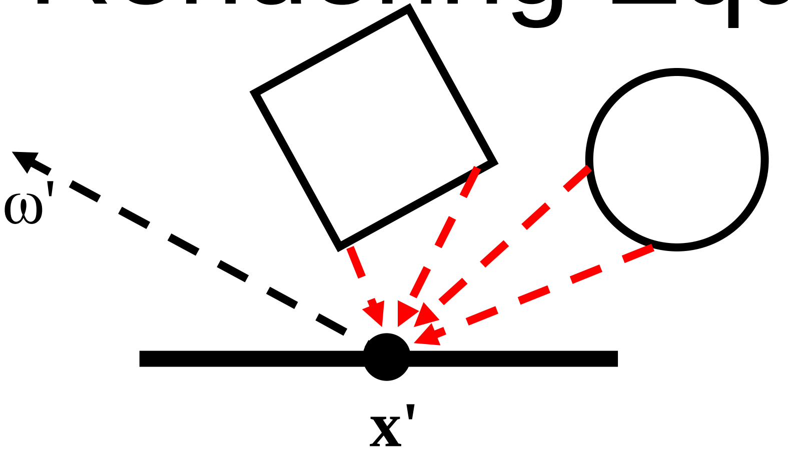
The Rendering Equation



$$L(x',\omega') = E(x',\omega') + \int \rho_{x'}(\omega,\omega')L(x,\omega)G(x,x')V(x,x')\, dA$$

E(x',ω') is the emitted radiance from a point: *E* is non-zero only if x' is emissive (a light *source*)

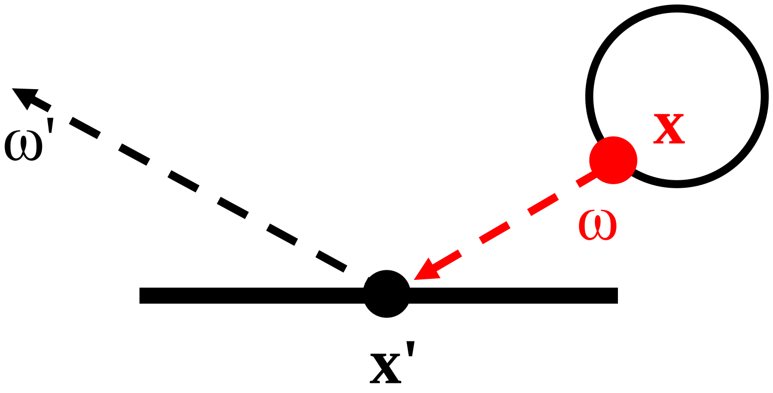
The Rendering Equation



$$L(x',\omega') = E(x',\omega') + \underbrace{\int \rho_{x'}(\omega,\omega')L(x,\omega)G(x,x')V(x,x')\,dA}$$

Sum the contribution from all of the other surfaces in the scene

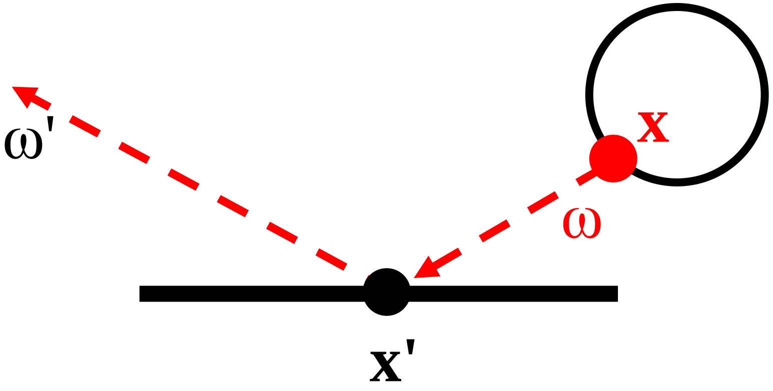
The Rendering Equation



$$L(x',\omega') = E(x',\omega') + \int \rho_{x'}(\omega,\omega')L(x,\omega)G(x,x')V(x,x')\,dA$$

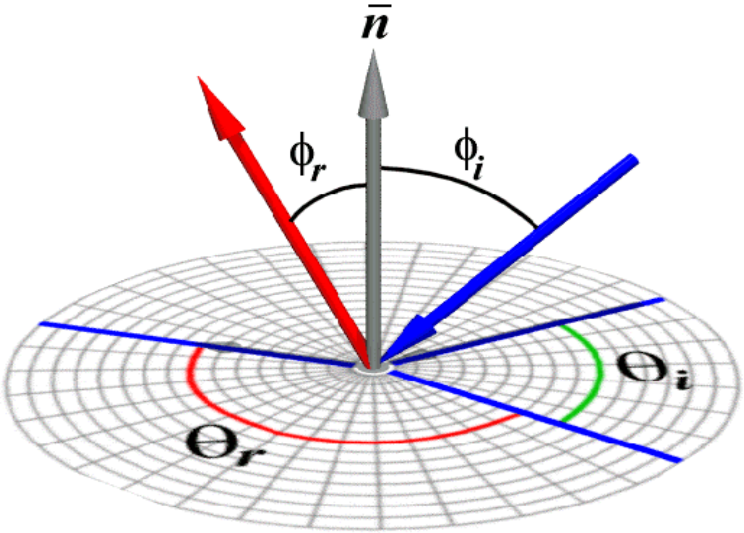
For each *x*, compute *L(x, ω)*, the radiance at point *x* in the direction ω (from *x* to *x'*)

The Rendering Equation

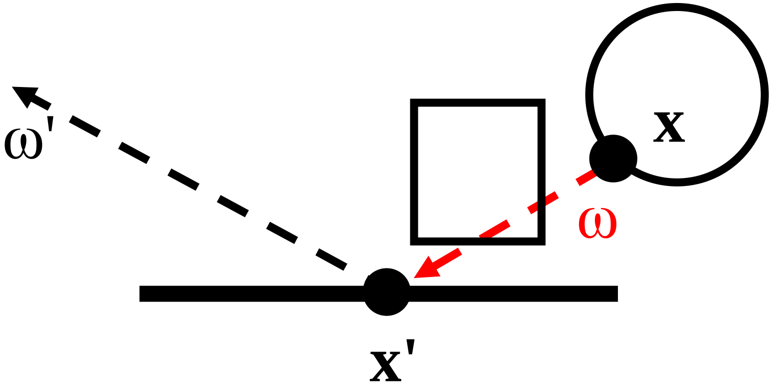


$$L(x',\omega') = E(x',\omega') + \int \rho_{x'}(\omega,\omega') L(x,\omega) G(x,x') V(x,x')\, dA$$

scale the contribution by $\rho_{x'}(\omega,\omega')$, the reflectivity (BRDF) of the surface at x'



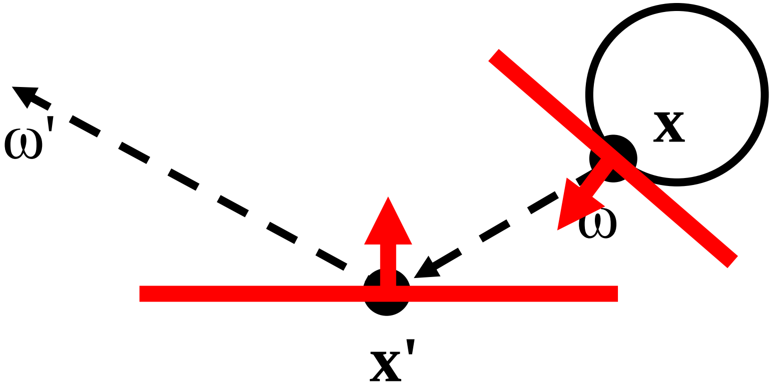
The Rendering Equation



$$L(x',\omega') = E(x',\omega') + \int \rho_{x'}(\omega,\omega')L(x,\omega)G(x,x')V(x,x')\, dA$$

For each *x*, compute *V(x,x')*,
the visibility between *x* and *x'*:
1 when the surfaces are unobstructed
along the direction ω, 0 otherwise

The Rendering Equation

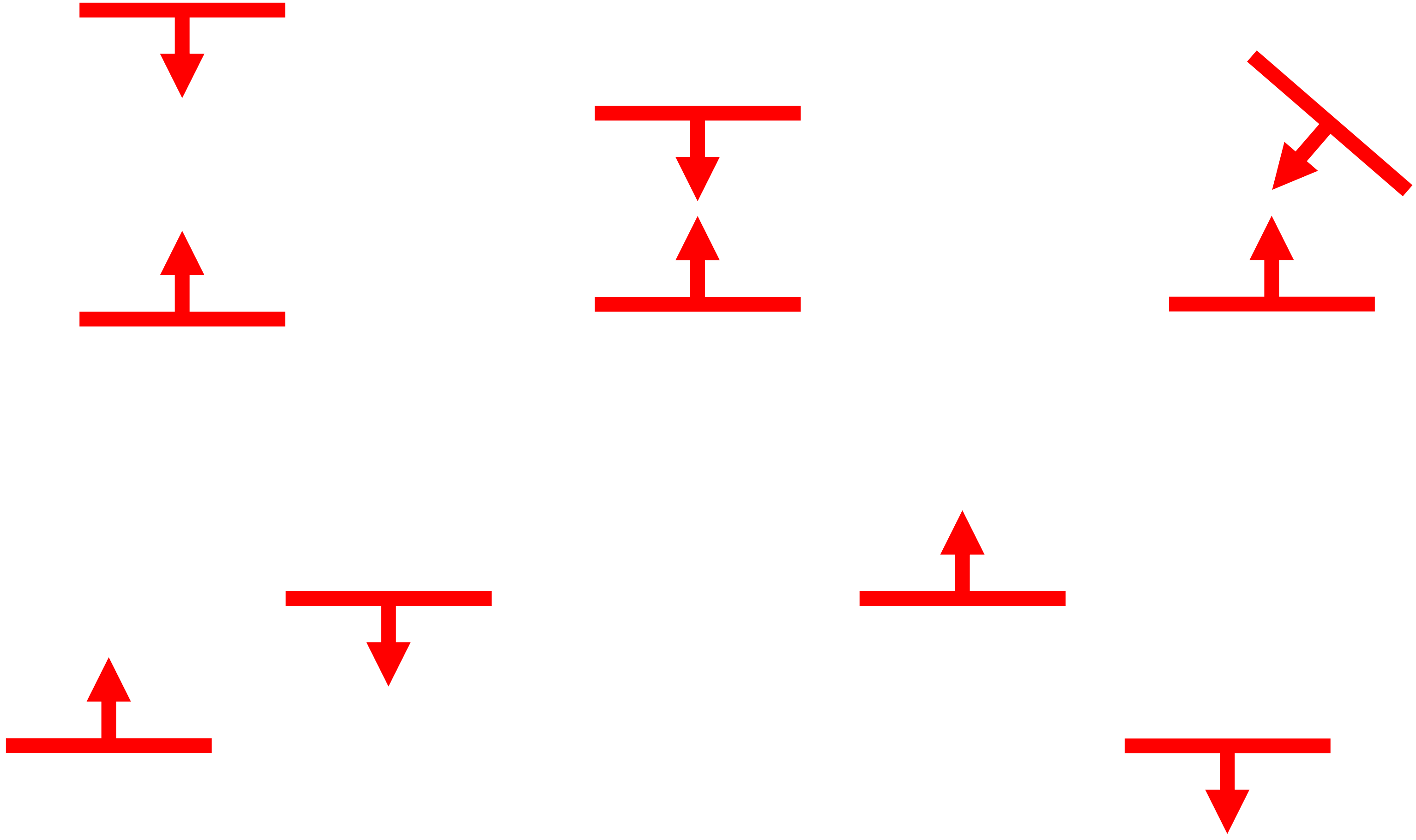


$$L(x',\omega') = E(x',\omega') + \int \rho_{x'}(\omega,\omega')L(x,\omega)G(x,x')V(x,x')\,dA$$

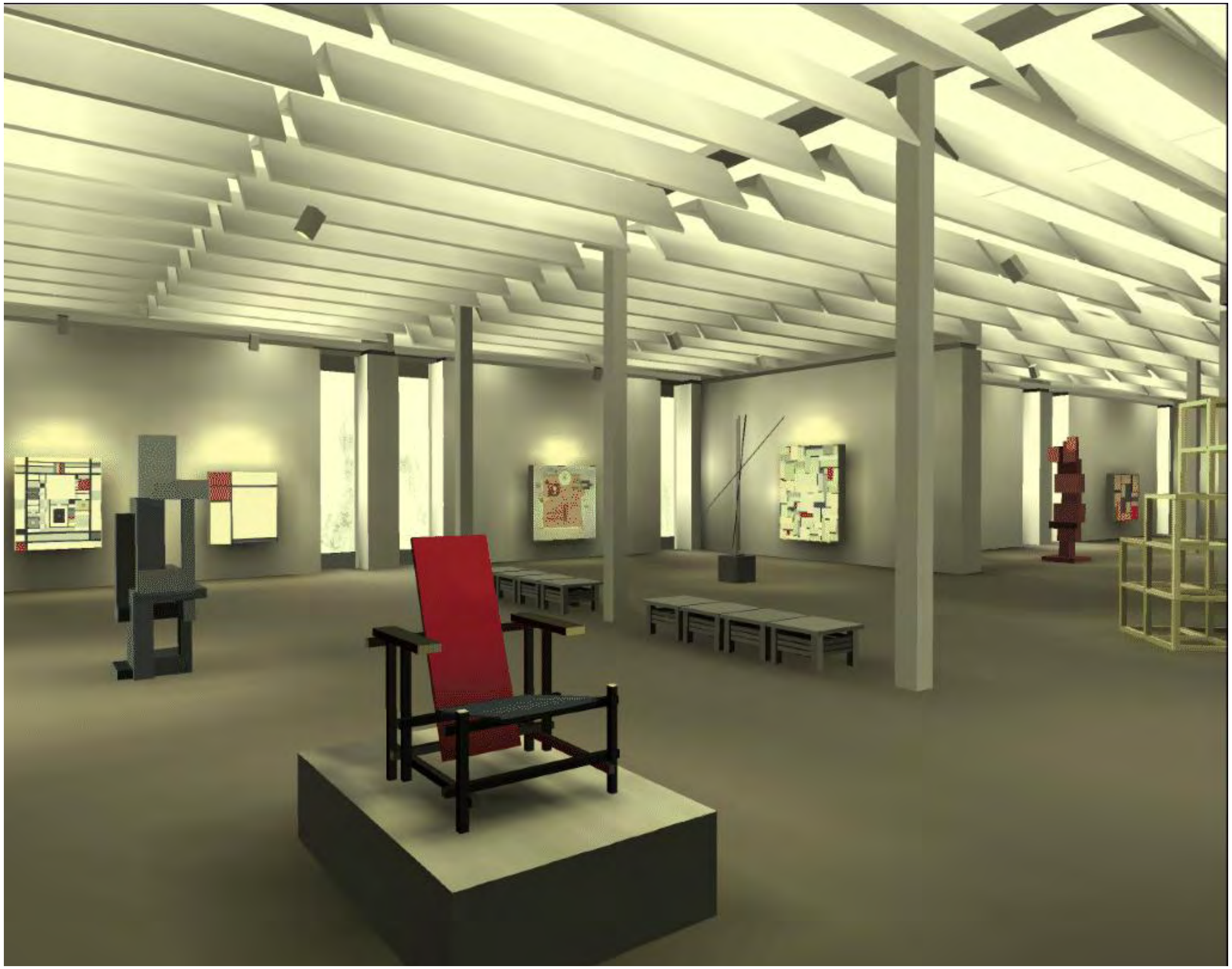
For each x, compute *G(x, x')*, which describes the on the geometric relationship between the two surfaces at x and x’

Intuition about G(x,x')?

- Which arrangement of two surfaces will yield the greatest transfer of light energy? Why?



Older Radiosity Images (1989)



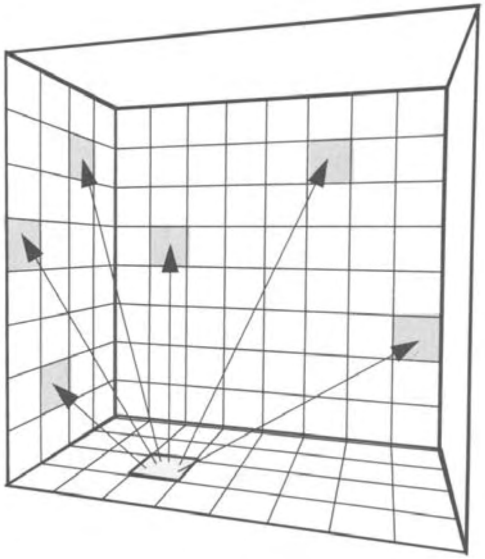
Museum simulation. Program of Computer Graphics, Cornell University.
50,000 patches. Note indirect lighting from ceiling.

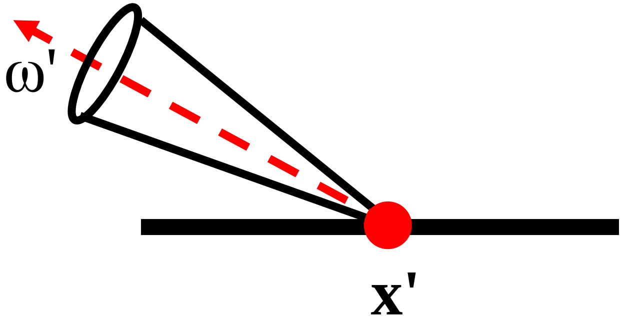
Radiosity

- Why Radiosity
  - The Cornell Box
  - Radiosity vs. Ray Tracing
- Global Illumination: The Rendering Equation
- Radiosity Equation/Matrix
- Calculating the Form Factors
- Progressive Radiosity

Radiosity Overview

- Surfaces are assumed to be perfectly Lambertian (diffuse)
  - reflect incident light in all directions with equal intensity
- The scene is divided into a set of small areas, or patches.
- The radiosity, B_i, of patch *i* is the total rate of energy leaving a surface. The radiosity over a patch is constant.
- Units for radiosity: Watts / steradian * meter^2



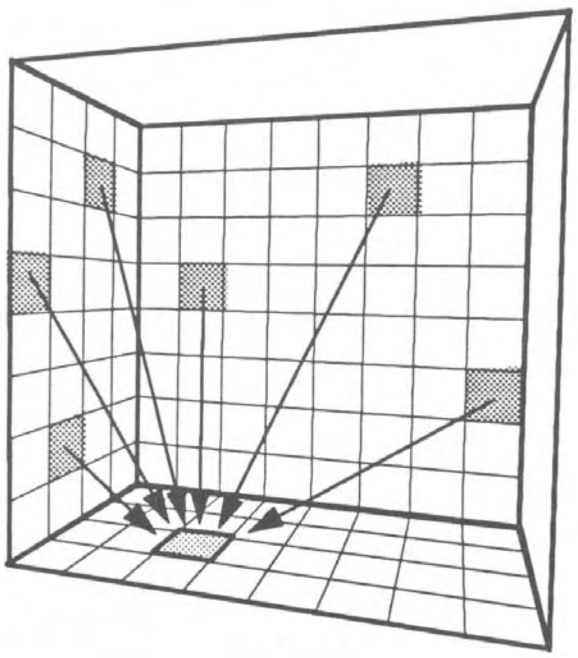


Radiosity Equation

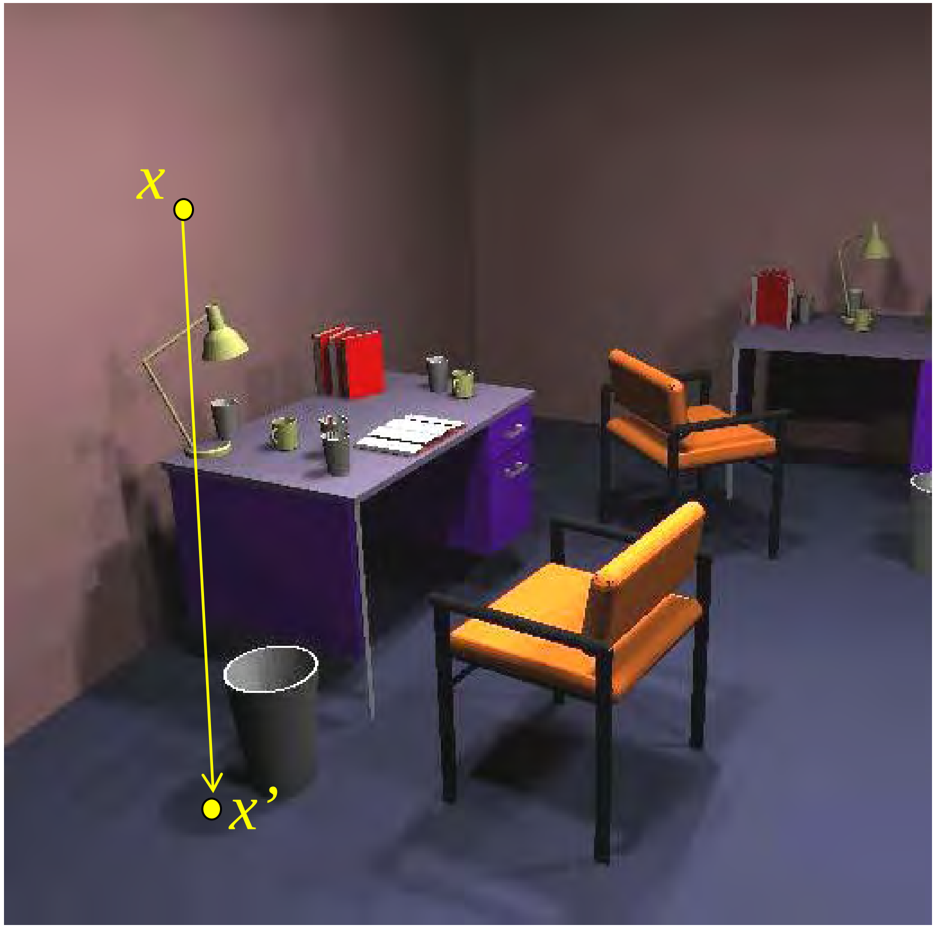
$$L(x',\omega') = E(x',\omega') + \int \rho_{x'}(\omega,\omega') L(x,\omega) G(x,x') V(x,x')\, dA$$

Radiosity assumption:
perfectly diffuse surfaces (not directional)

$$B_{x'} = E_{x'} + \rho_{x'} \int B_x \; G(x,x') V(x,x')$$



Continuous Radiosity Equation



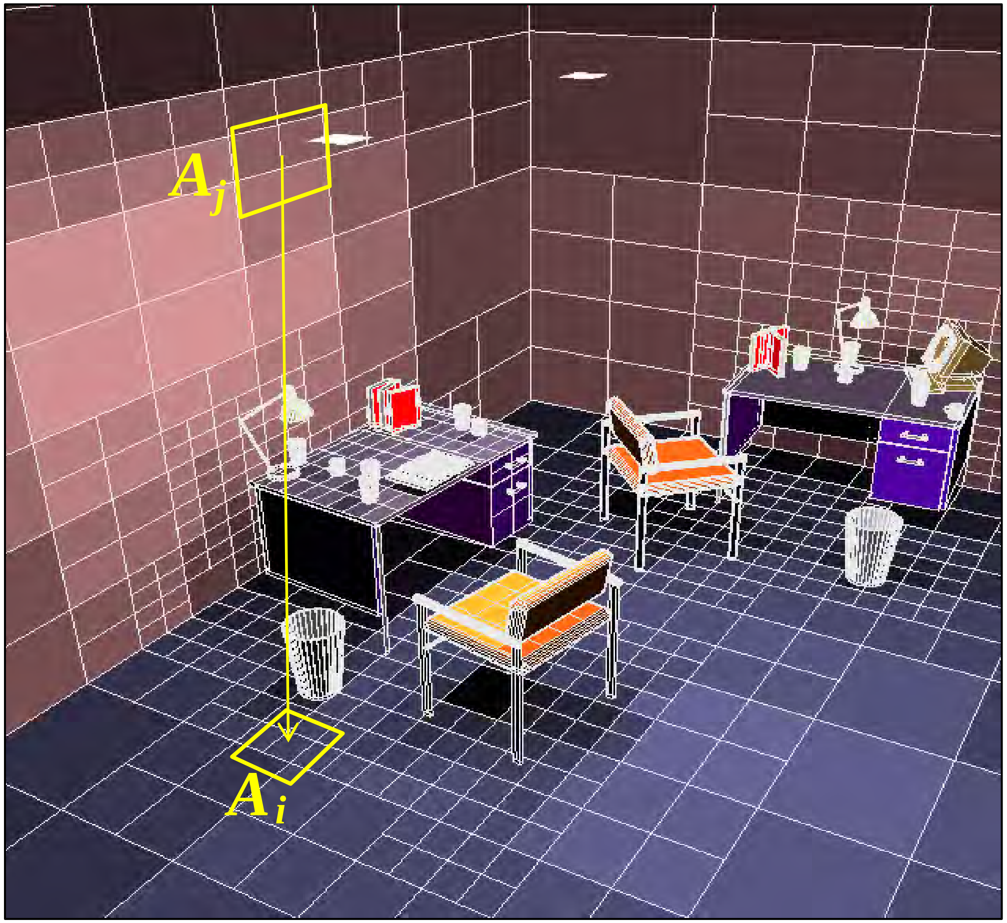
$$B_{x'} = E_{x'} + \overset{\substack{\text{reflectivity}\\ \downarrow}}{\rho_{x'}} \int \underbrace{G(x,x')\, V(x,x')}_{\text{form factor}}\, B_x$$

G: geometry term
V: visibility term

No analytical solution,
even for simple configurations

Discrete Radiosity Equation

Discretize the scene into *n* patches, over which the radiosity is constant



reflectivity

$$B_i = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j$$

form factor

- discrete representation
- iterative solution
- costly geometric/visibility calculations

The Radiosity Matrix

$$B_i = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j$$

n simultaneous equations with *n* unknown B_i values can be written in matrix form:

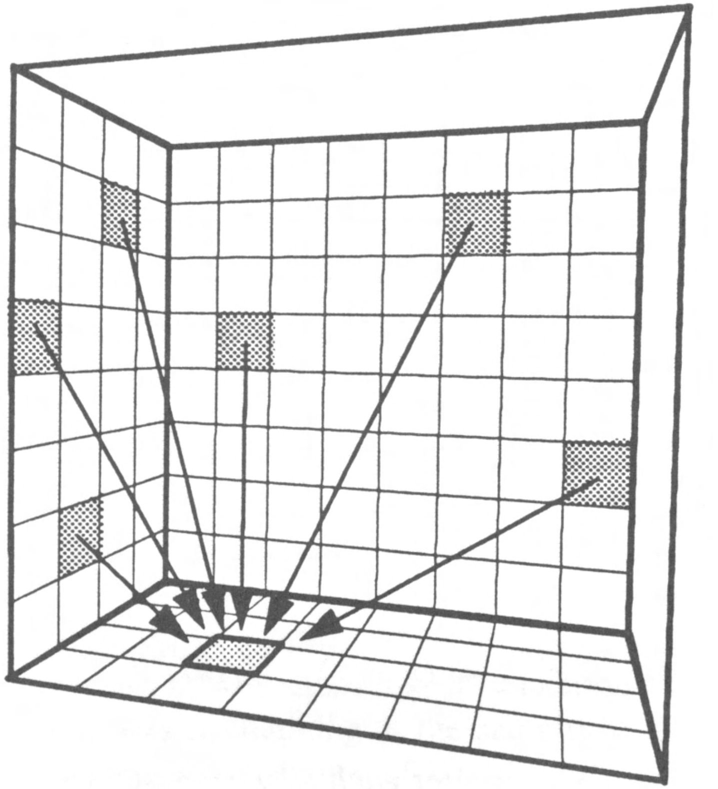
$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & & \\ \vdots & & \ddots & \\ -\rho_n F_{n1} & \cdots & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

A solution yields a single radiosity value B_i for each patch in the environment, a view-independent solution.

Solving the Radiosity Matrix

The radiosity of a single patch *i* is updated for each iteration by *gathering* radiosities from all other patches:

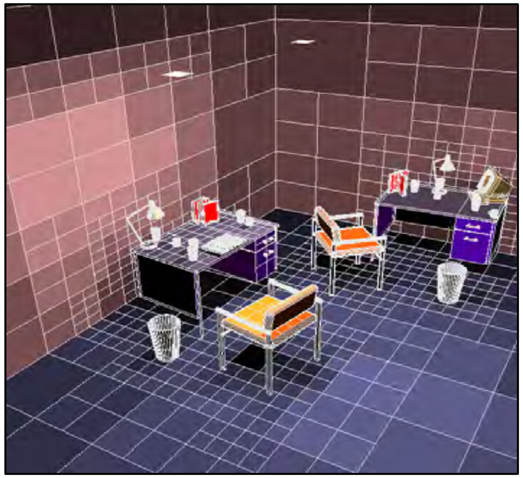
$$\begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_i \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_i \\ \vdots \\ E_n \end{bmatrix} + \begin{bmatrix} & & & \\ & & & \\ & & & \\ \rho_i F_{i1} & \rho_i F_{i2} & \cdots & \rho_i F_{in} \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_i \\ \vdots \\ B_n \end{bmatrix}$$

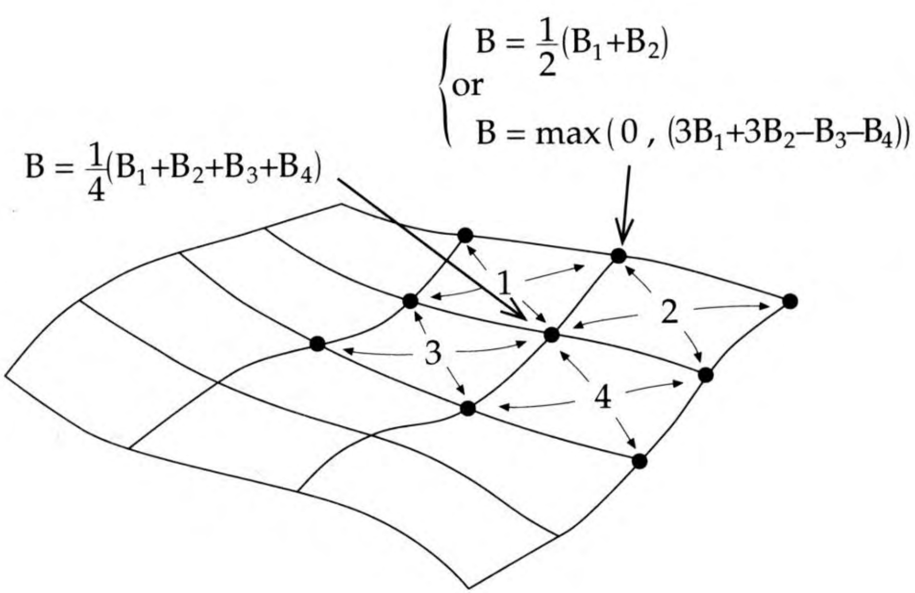


This method is fundamentally a Gauss-Seidel relaxation

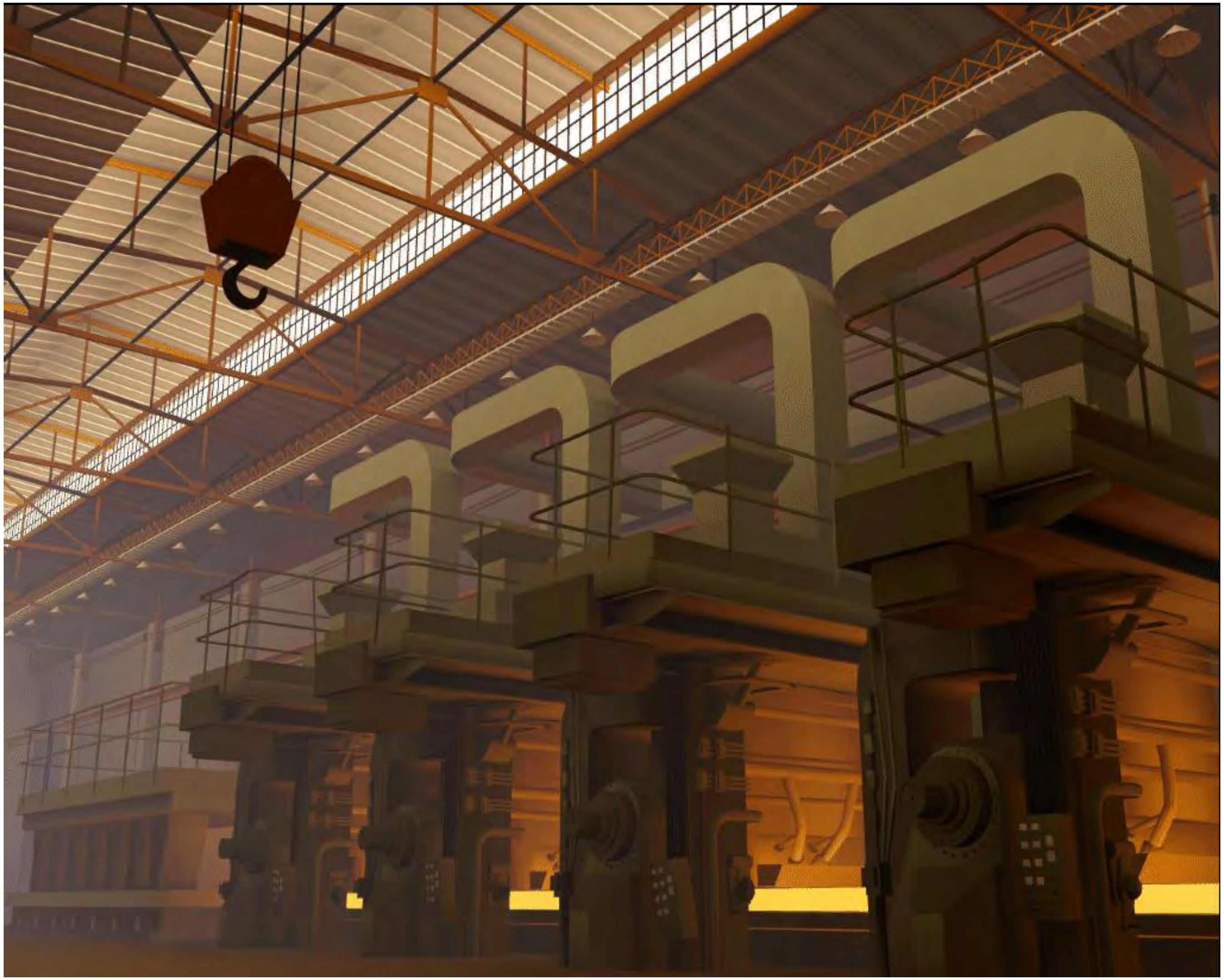
Computing Vertex Radiosities

- B_i radiosity values are constant over the extent of a patch.
- How are they mapped to the vertex radiosities (intensities) needed by the renderer?
  - Average the radiosities of patches that contribute to the vertex
  - Vertices on the edge of a surface are assigned values extrapolation




$B = \frac{1}{2}(B_1+B_2)$
or
$B = \max(0, (3B_1+3B_2-B_3-B_4))$
$B = \frac{1}{4}(B_1+B_2+B_3+B_4)$
1
2
3
4

Radiosity 1988



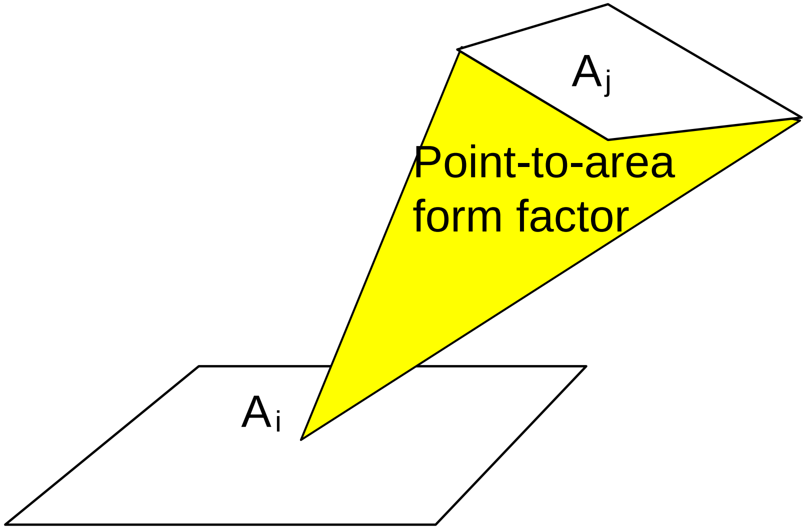
Factory simulation. Program of Computer Graphics, Cornell University.
30,000 patches.

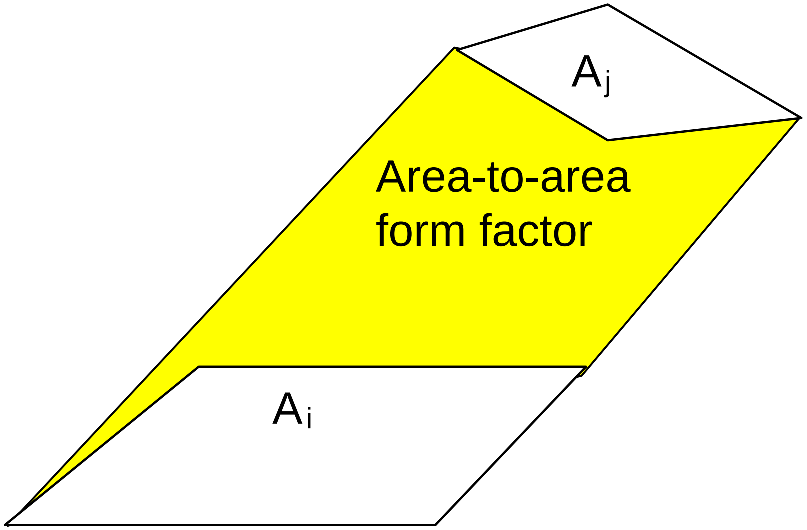
Today

- Why Radiosity
  - The Cornell Box
  - Radiosity vs. Ray Tracing
- Global Illumination: The Rendering Equation
- Radiosity Equation/Matrix
- Calculating the Form Factors
- Progressive Radiosity

Radiosity Patches are Finite Elements

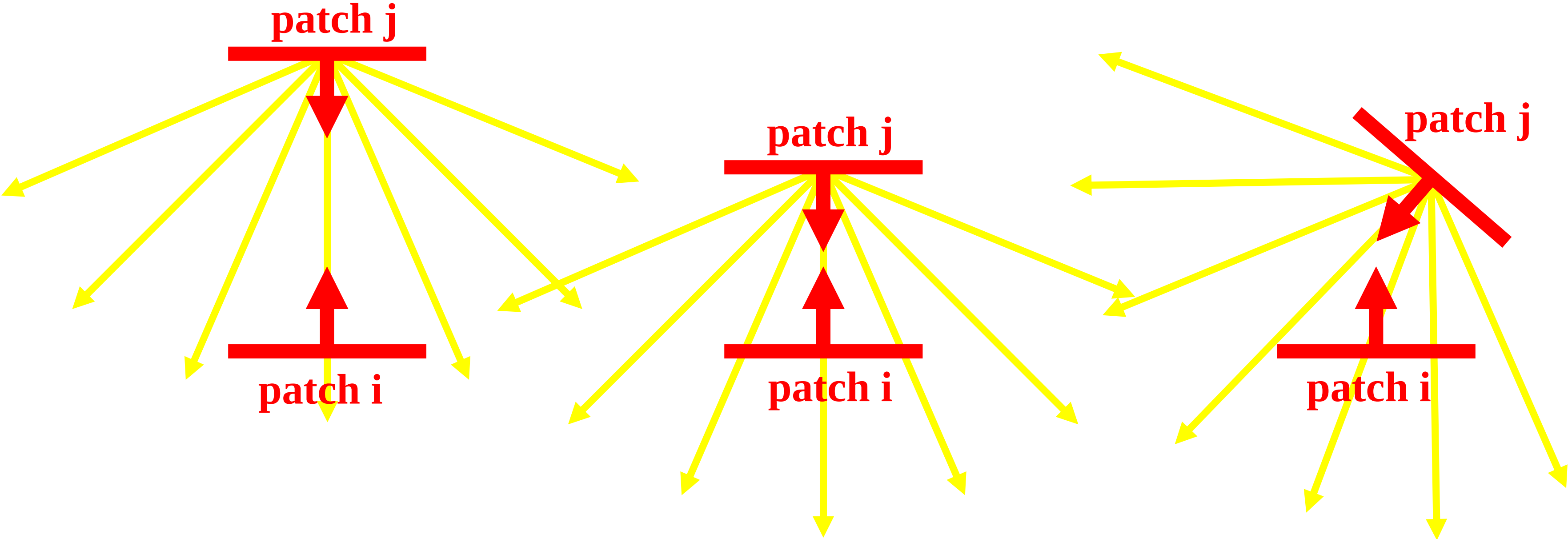
- We are trying to solve an the rendering equation over the *infinite-dimensional* space of radiosity functions over the scene.
- We project the problem onto a *finite basis* of functions: piecewise constant over patches





Calculating the Form Factor F_{ij}

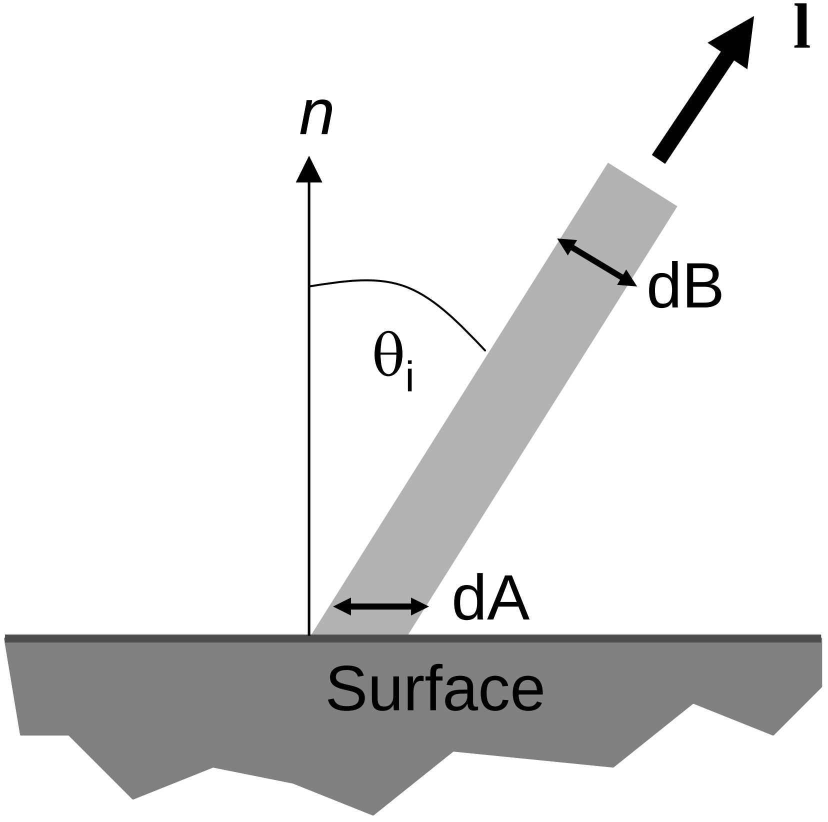
- F_{ij} = fraction of light energy leaving patch j that arrives at patch i
- Takes account of both:
  - geometry (size, orientation & position)
  - visibility (are there any occluders?)



Remember Diffuse Lighting?

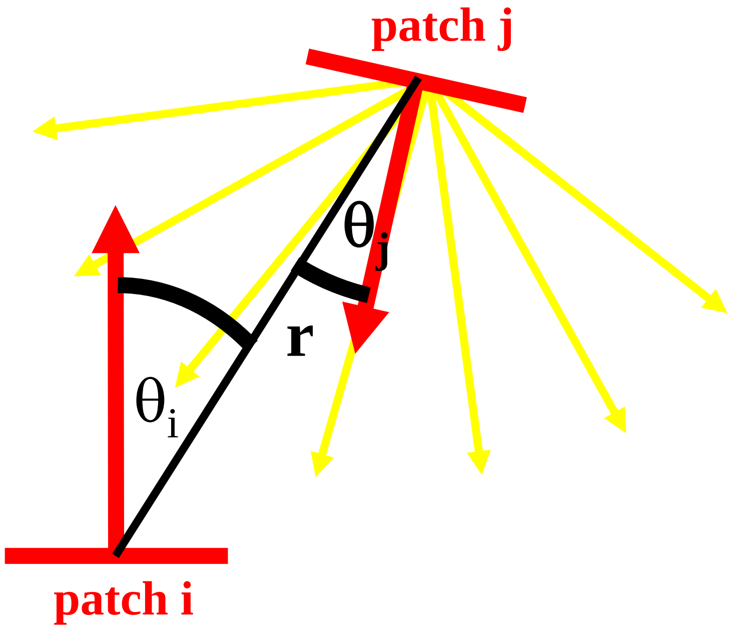
$$L_o = k_d (\mathbf{n} \cdot \mathbf{l}) \frac{L_i}{r^2}$$

$$dA = dB \cos\theta_i$$



Calculating the Form Factor F_{ij}

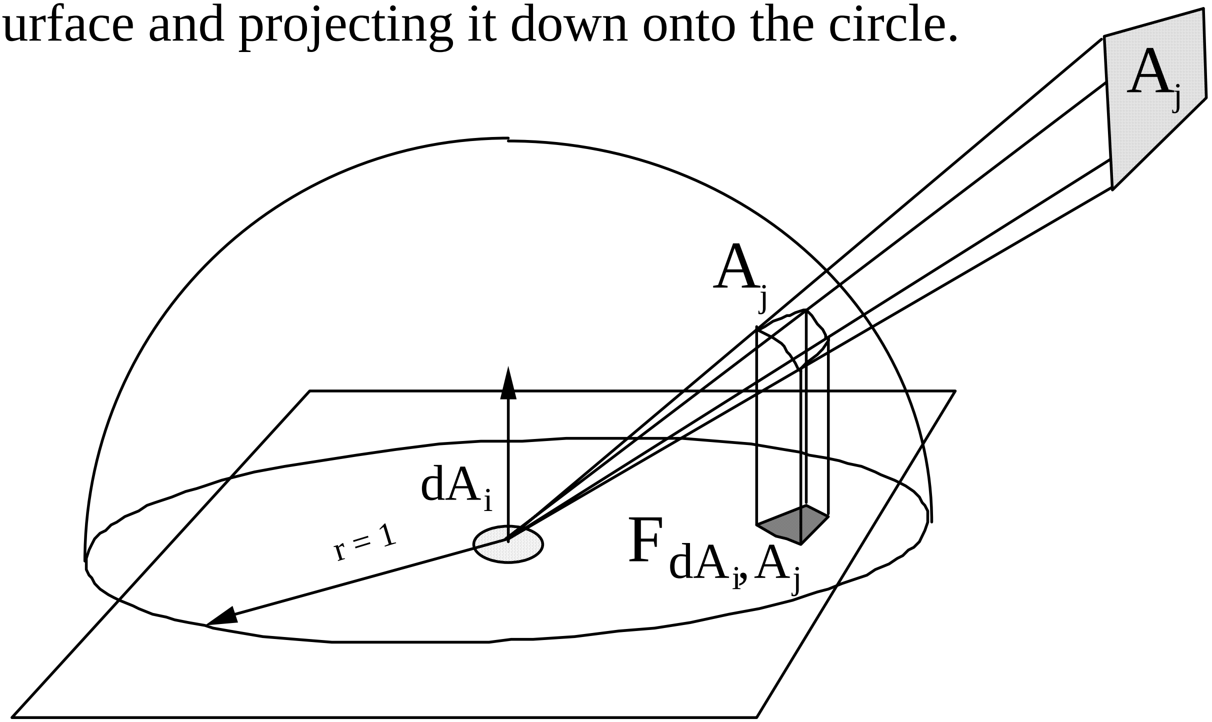
- F_{ij} = fraction of light energy leaving patch j that arrives at patch I



$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} V_{ij}\, dA_j\, dA_i$$

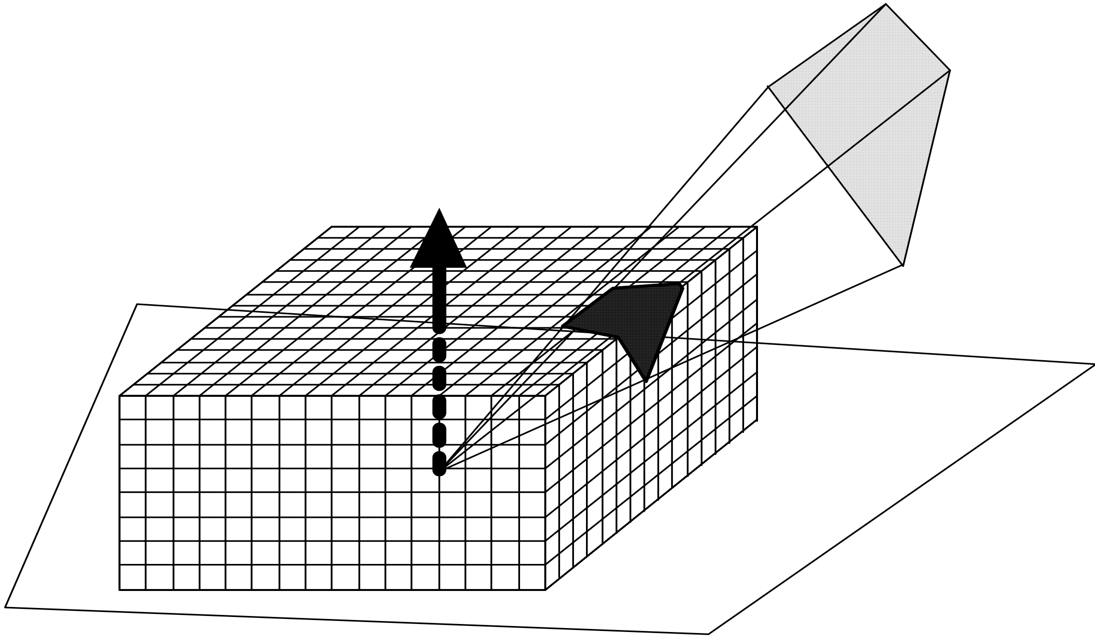
Form Factor Determination

The Nusselt analog: the form factor of a patch is equivalent to the fraction of the the unit circle that is formed by taking the projection of the patch onto the hemisphere surface and projecting it down onto the circle.



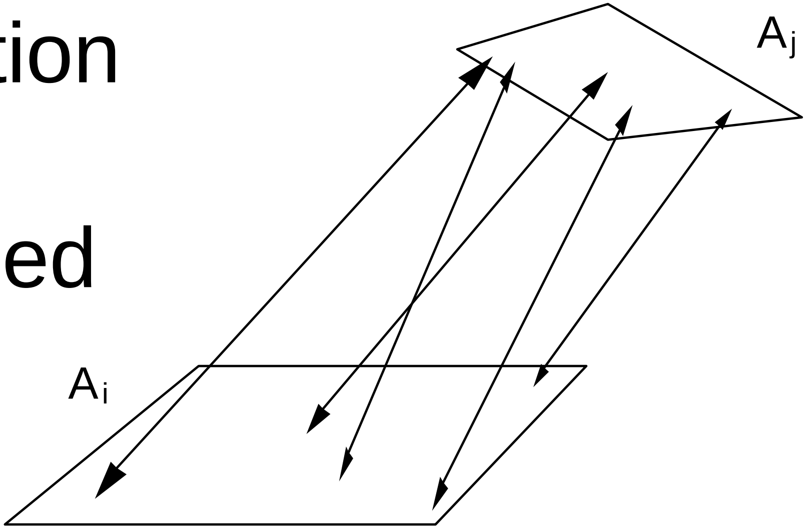
Hemicube Algorithm

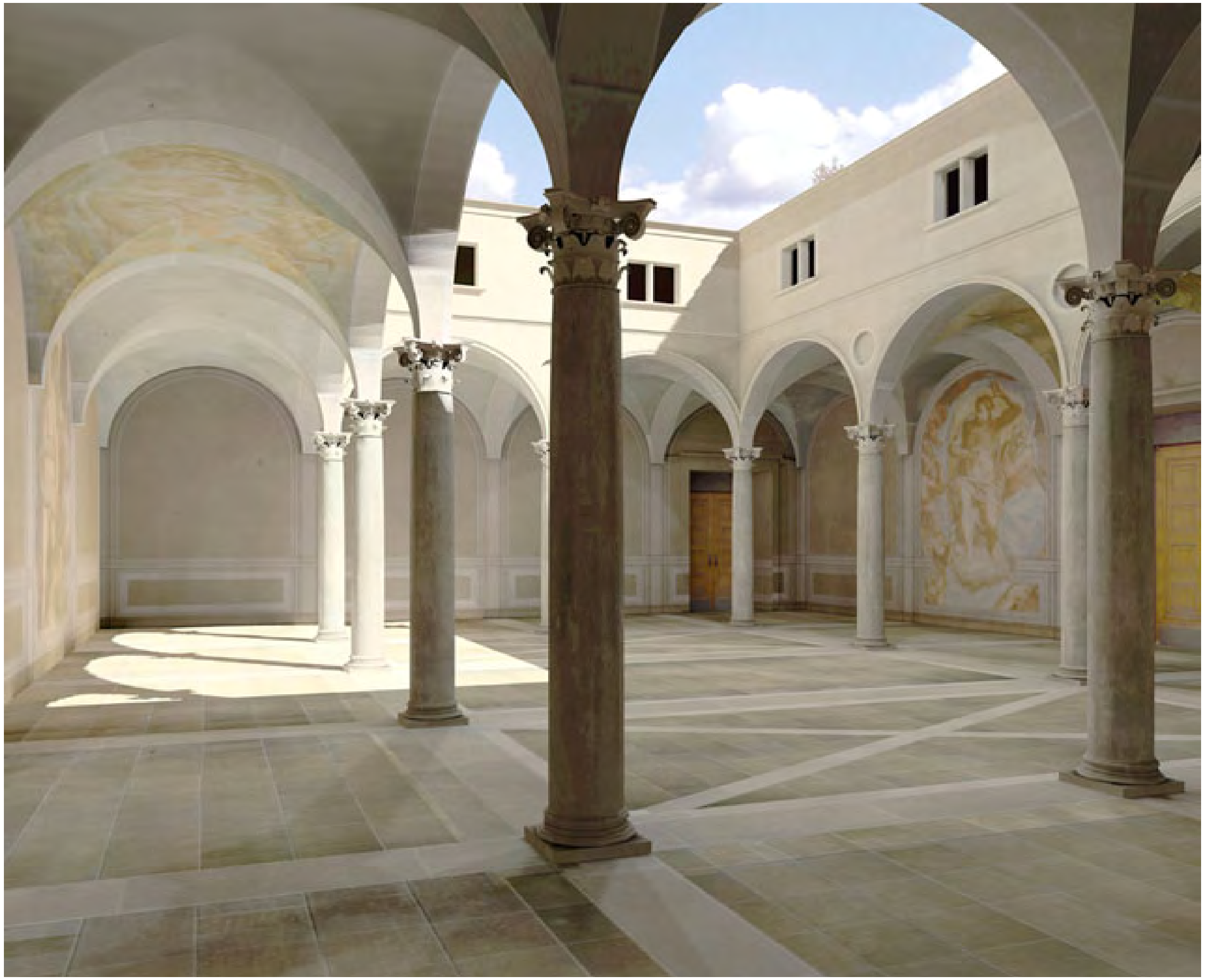
- A hemicube is constructed around the center of each patch
- Faces of the hemicube are divided into "pixels"
- Each patch is projected (rasterized) onto the faces of the hemicube
- Each pixel stores its pre-computed form factor The form factor for a particular patch is just the sum of the pixels it overlaps
- Patch occlusions are handled similar to z-buffer rasterization



Form Factor from Ray Casting

- Cast *n* rays between the two patches
  - *n* is typically between 4 and 32
  - Compute visibility
  - Integrate the point-to-point form factor
- Permits the computation of the patch-to-patch form factor, as opposed to point-to-patch



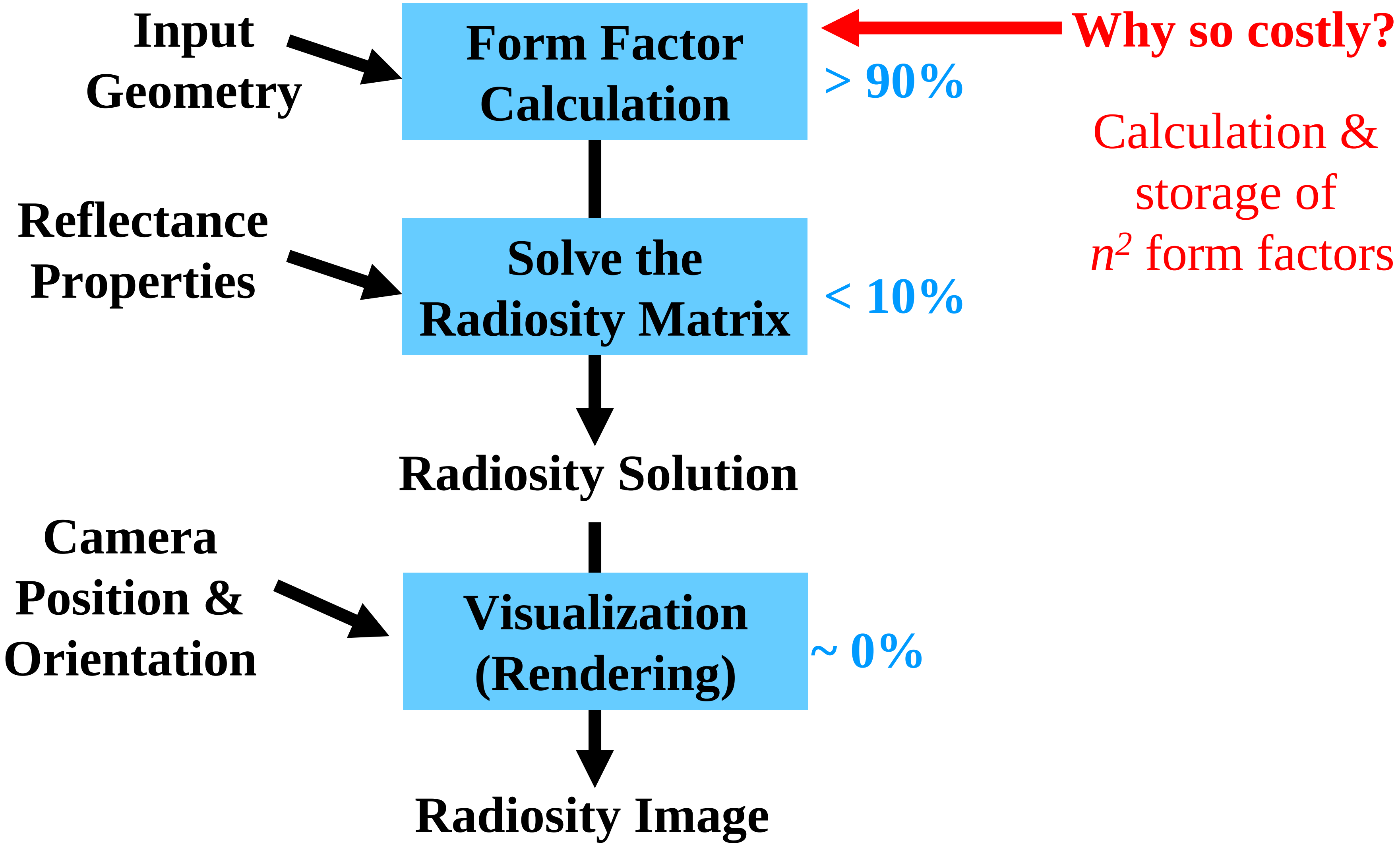


Lightscape **http://www.lightscape.com**

Radiosity

- Why Radiosity
  - The Cornell Box
  - Radiosity vs. Ray Tracing
- Global Illumination: The Rendering Equation
- Radiosity Equation/Matrix
- Calculating the Form Factors
- Progressive Radiosity

Stages in a Radiosity Solution



Input Geometry

Form Factor Calculation

> 90%

Why so costly?

Calculation & storage of n^2 form factors

Reflectance Properties

Solve the Radiosity Matrix

< 10%

Radiosity Solution

Camera Position & Orientation

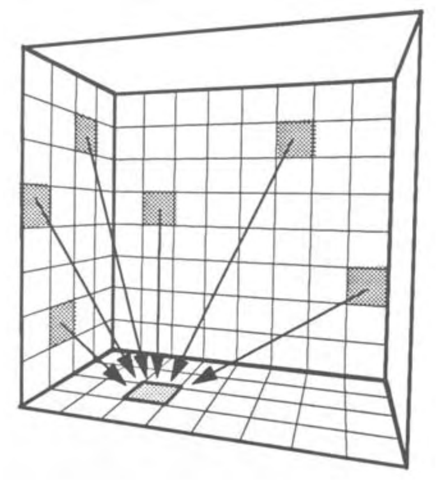
Visualization (Rendering)

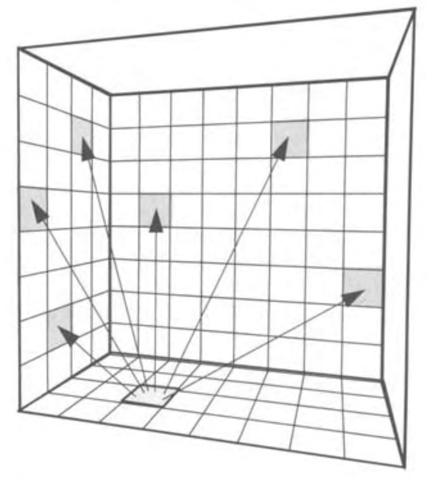
~ 0%

Radiosity Image

Progressive Refinement

- Goal: Provide frequent and timely updates to the user during computation
- Key Idea: Update the entire image at every iteration, rather than a single patch
- How? Instead of summing the light received by one patch, distribute the radiance of the patch with the most *undistributed radiance*.

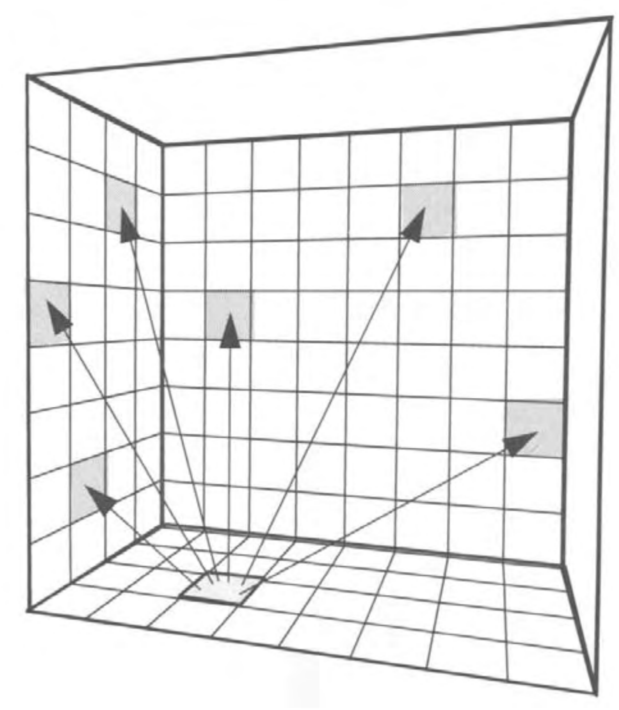




Reordering the Solution for PR

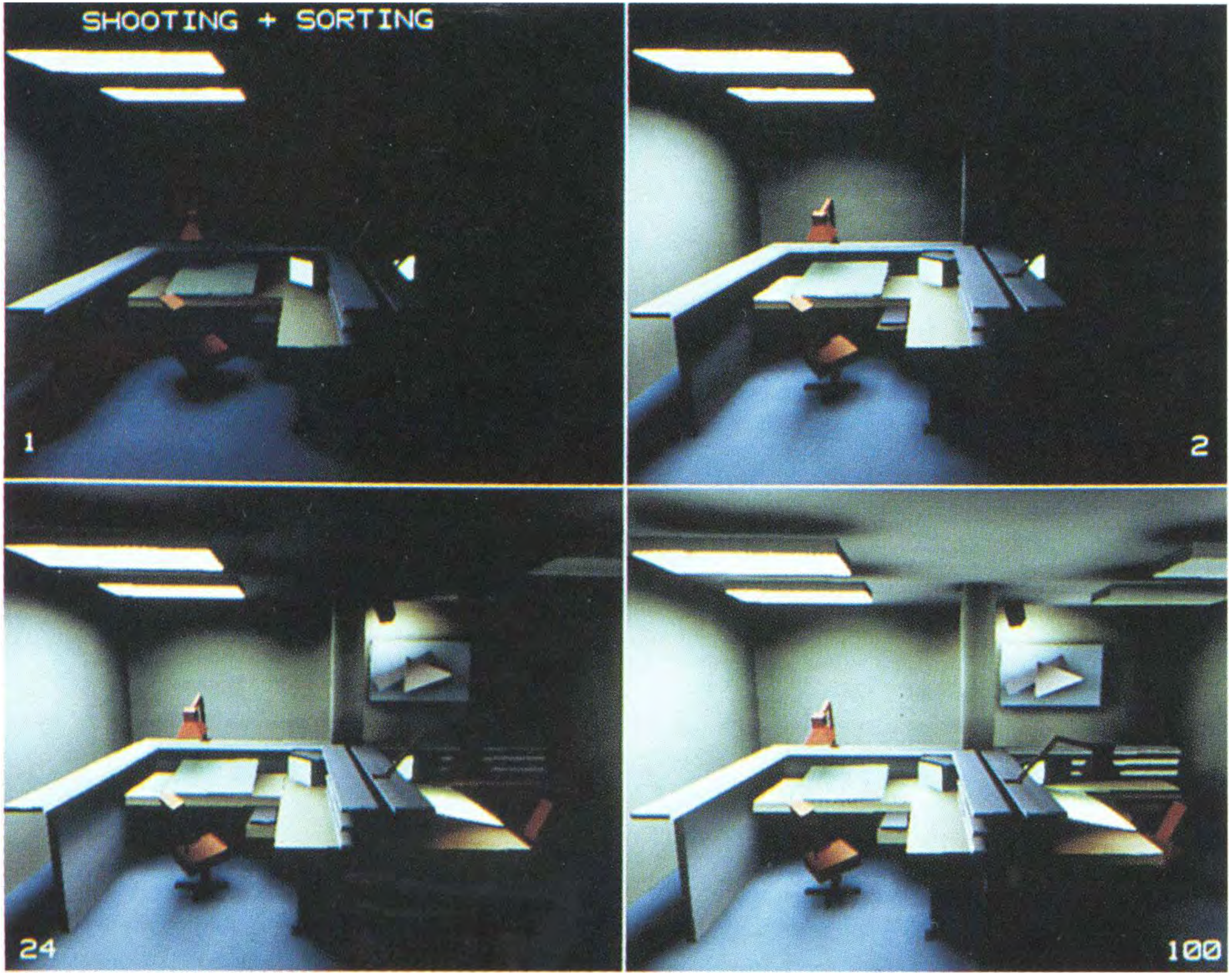
Shooting: the radiosity of all patches is updated for each iteration:

$$\begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ \\ \vdots \\ B_n \end{bmatrix} + \begin{bmatrix} \cdots & \rho_1 F_{1i} & \cdots \\ \cdots & \rho_2 F_{2i} & \\ & & \\ & & \\ & & \\ \cdots & \rho_n F_{ni} & \cdots \end{bmatrix} \begin{bmatrix} \\ \vdots \\ B_i \\ \vdots \\ \\ \\ \end{bmatrix}$$

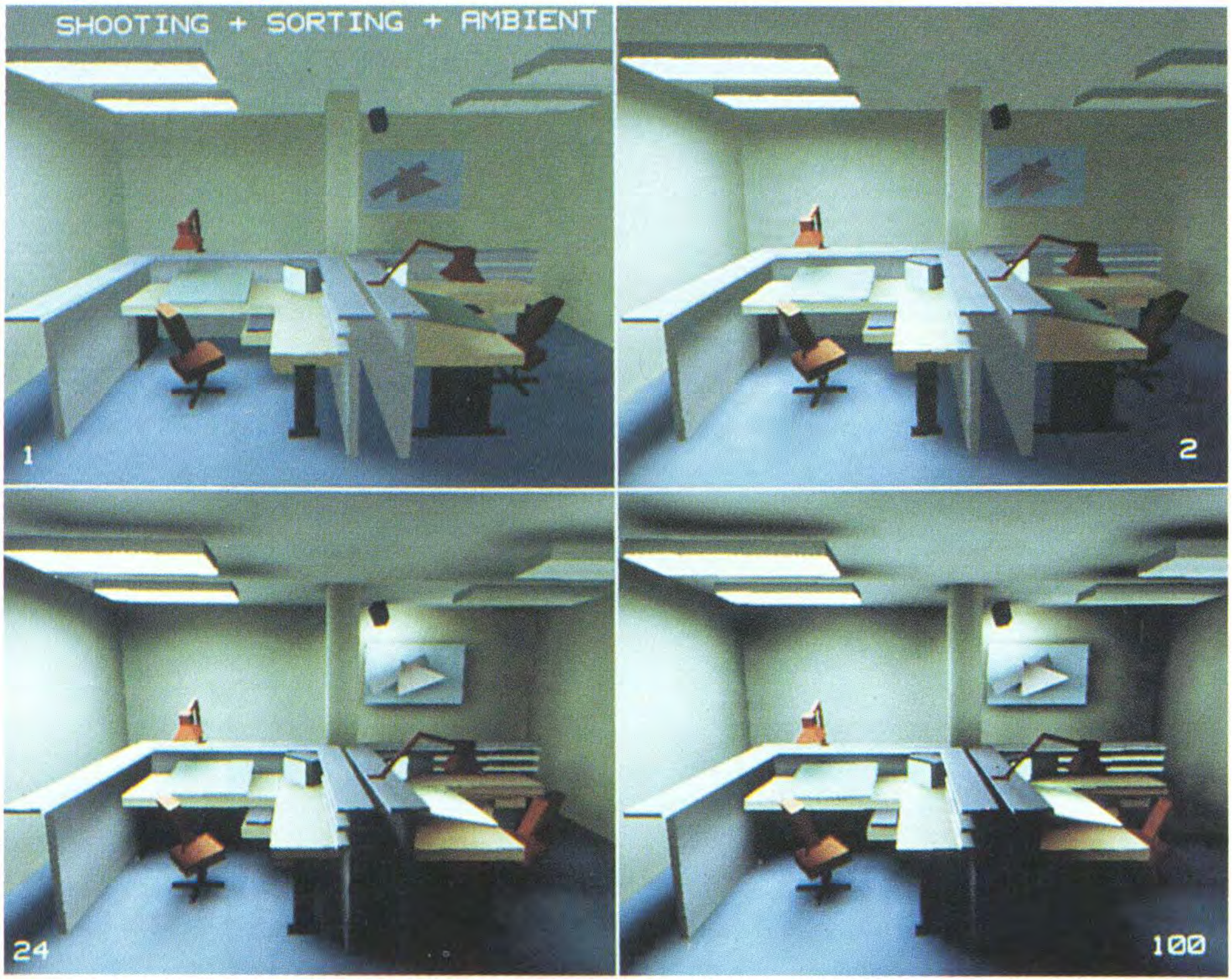


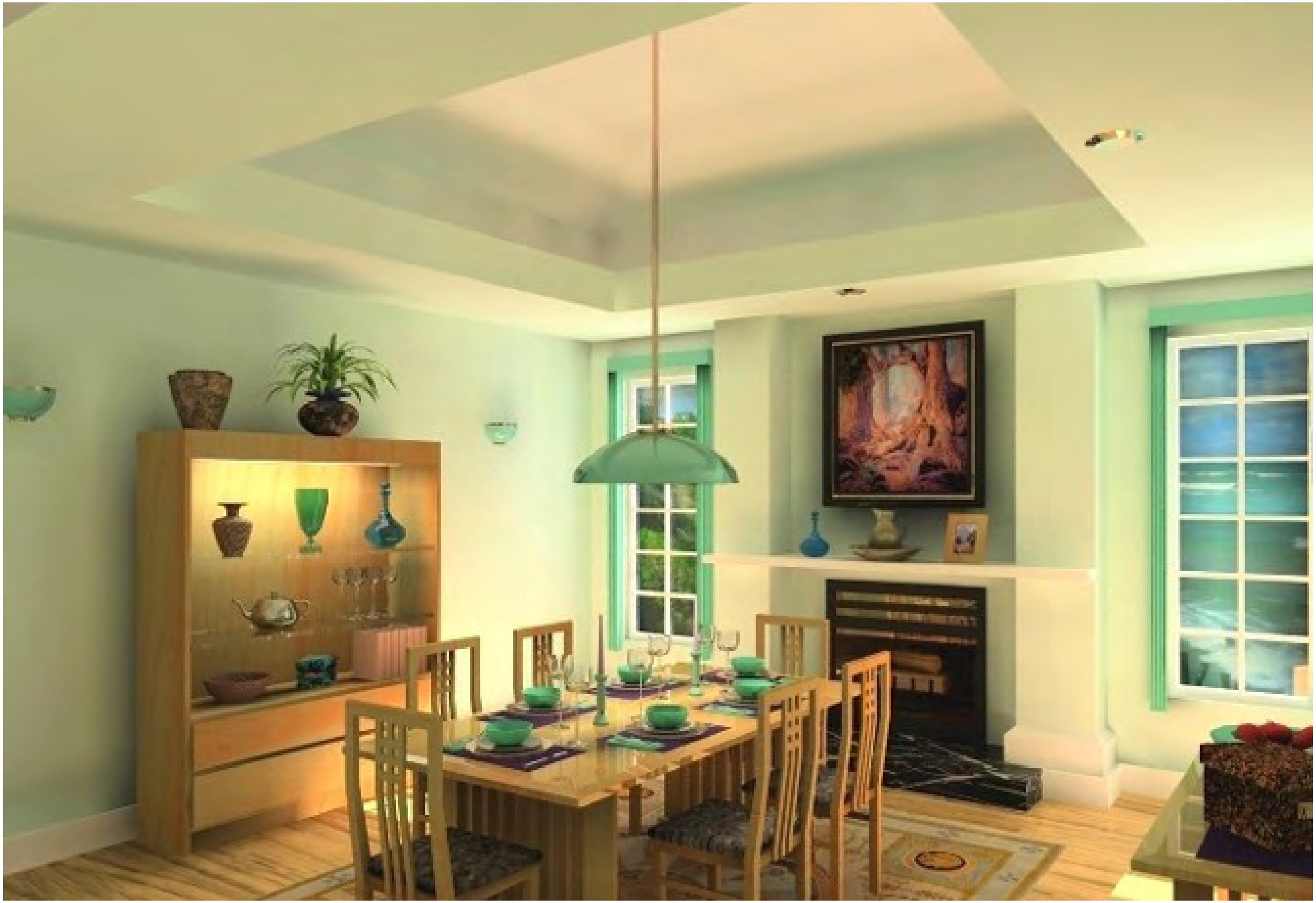
This method is fundamentally a Southwell relaxation

Progressive Refinement w/out Ambient Term



Progressive Refinement with Ambient Term





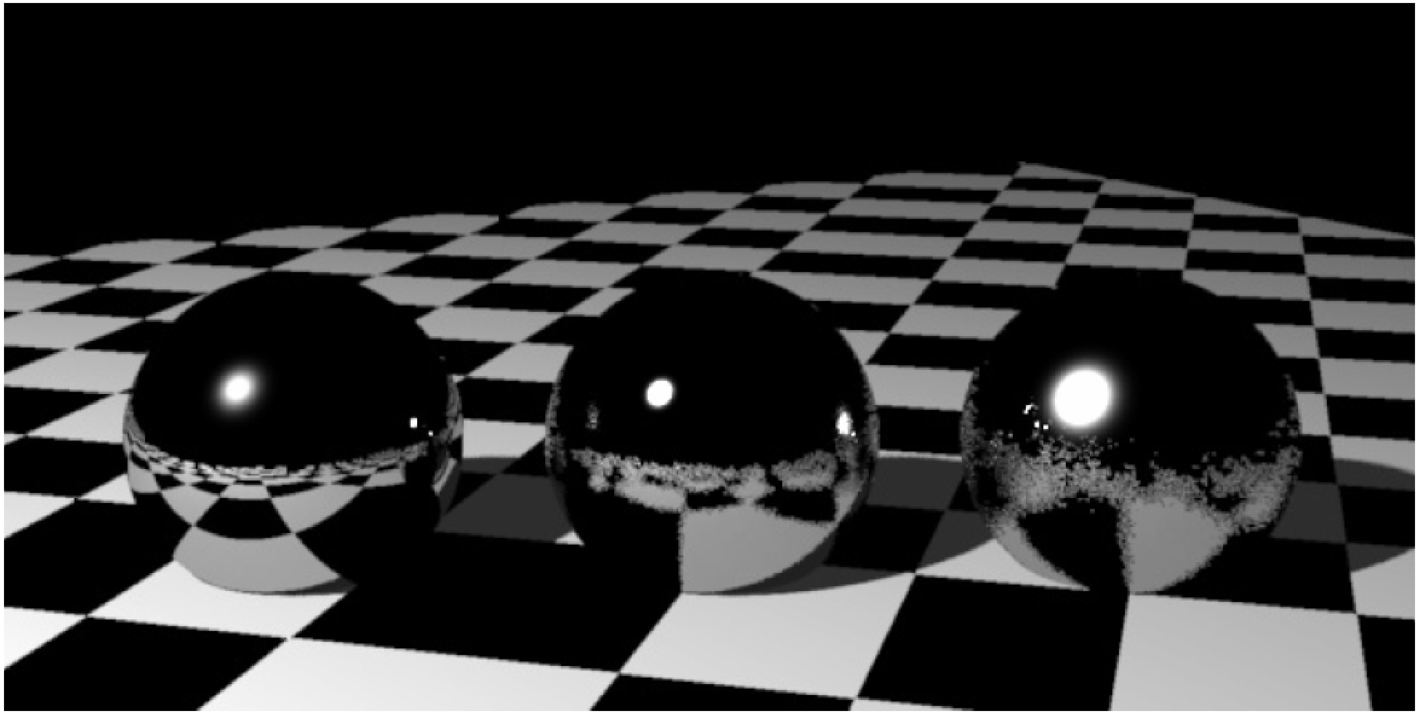
Lightscape **http://www.lightscape.com**

Monte Carlo Methods

- Expected value and variance
- Analysis of Monte-Carlo integration
- Monte-Carlo in graphics
- Importance sampling
- Stratified sampling
- Global illumination
- Advanced Monte-Carlo rendering

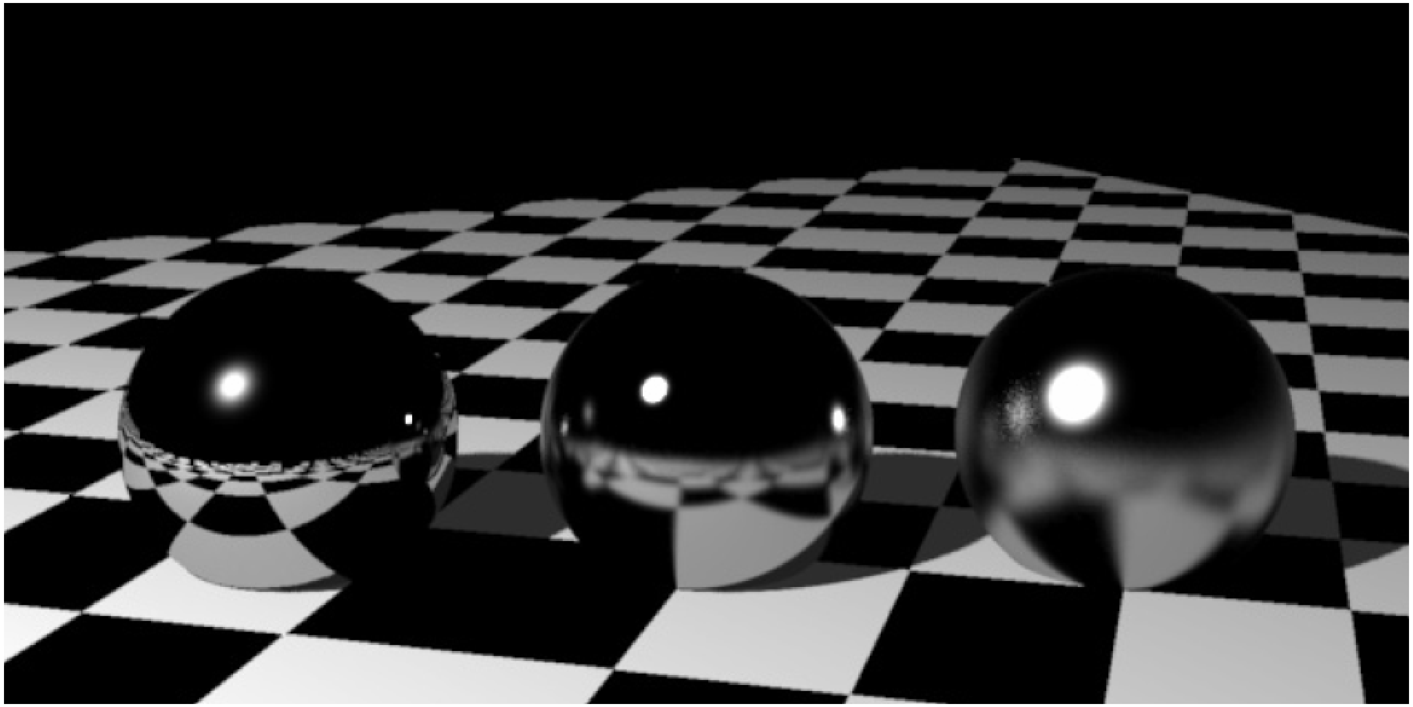
A little bit of eye candy for motivation

- Glossy material rendering
- Random reflection rays around mirror direction
  - 1 sample per pixel



A little bit of eye candy for motivation

- Glossy material rendering
- Random reflection rays around mirror direction
  - 256 sample per pixel



Monte Carlo Images

- Image from the ARNOLD Renderer by Marcos Fajardo

98
MAJO
DIVERSI

Expected value

$$E[x] = \int_{-\infty}^{\infty} x p(x) dx$$

$$E[f(x)] = \int_{-\infty}^{\infty} f(x) p(x) dx$$

- Expected value is linear
- $E[f_1(x) + a\, f_2(x)] \;=\; E[f_1(x)] + a\, E[f_2(x)]$

Variance

$$\sigma^2 = E[(x - E[x])^2] = \int_{-\infty}^{\infty} (x - E[x])^2 p(x) dx$$

- Measure of deviation from expected value
- Expected value of square difference (MSE)
- Standard deviation σ:
  square root of variance (notion of error, RMS)

Variance

$$\sigma^2 = E[(x - E[x])^2] = E[x^2] - (E[x])^2$$

- Proof: $\sigma^2 = E[(x - E[x])^2]$

$$= E[x^2 - 2xE[x] + E[x]^2]$$

- Note that E[x] is a constant. By linearity of E we have:

$$\sigma^2 = E[x^2] - (2E[x])E[x] + (E[x])^2$$

$$\sigma^2 = E[x^2] - (E[x])^2$$

Monte Carlo Methods

- Expected value and variance
- Analysis of Monte-Carlo integration
- Monte-Carlo in graphics
- Importance sampling
- Stratified sampling
- Global illumination
- Advanced Monte-Carlo rendering

Monte Carlo integration

- Function *f(x)* of x 2 [a b] $I = \int_a^b f(x)dx$
- We want to compute

- Consider a random variable *x*
- If *x* has uniform distribution, *I=E[f(x)]*
  - By definition of the expected value

Sum of Random Variables

- Use *N* independent identically-distributed (IID) variables x_i
  - Share same probability (uniform here)
- Define

$$F_N = \frac{1}{N} \sum_{j=1}^{n} f(x_i)$$

- By linearity of the expectation:
  $E[F_N] = E[f(x)]$

Study of variance

$$\sigma^2[F_N] = \sigma^2\left[\sum_{j=1}^{n} \frac{f(x_i)}{N}\right]$$

- Recall $\sigma^2[x+y] = \sigma^2[x] + \sigma^2[y] + 2\,\mathrm{Cov}[x,y]$
  - – We have independent variables: Cov[xi, xj]=0 if i $\neq$ j

□ $\sigma^2[ax] = a^2\,\sigma^2[x]$

$$\boxed{\sigma^2[F_N] = \frac{\sigma^2[f(x)]}{N}}$$

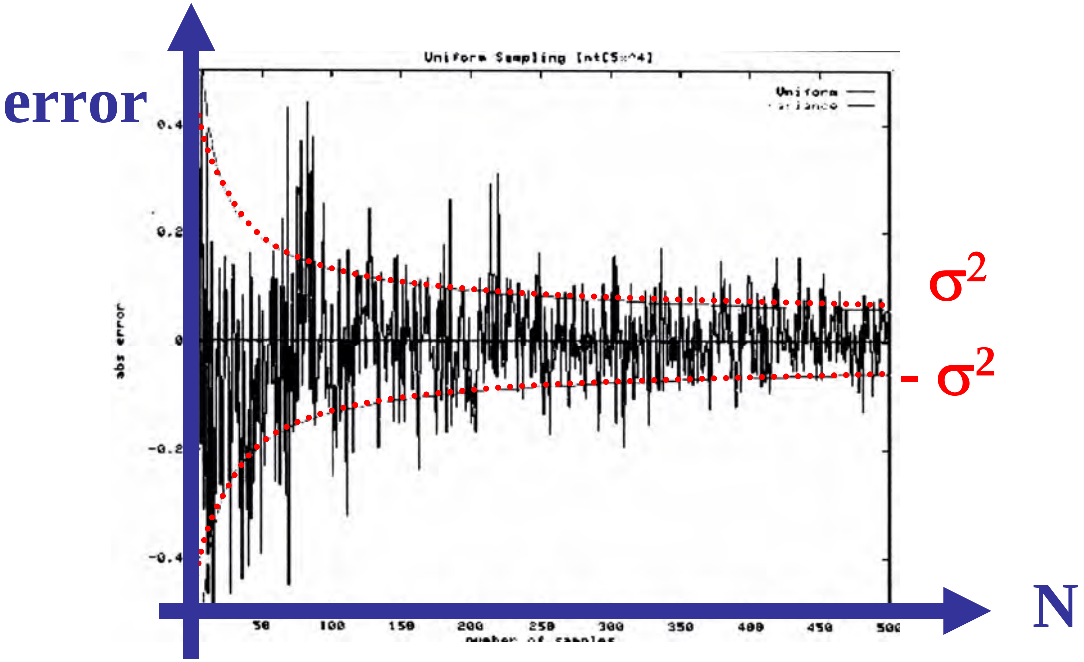
- i.e. stddev σ (error) decreases by $\boxed{\sqrt{N}}$

Example

$$I = \int_0^1 5x^4 dx$$

- We know it should be 1.0

- In practice with uniform samples:



Monte Carlo integration with probability

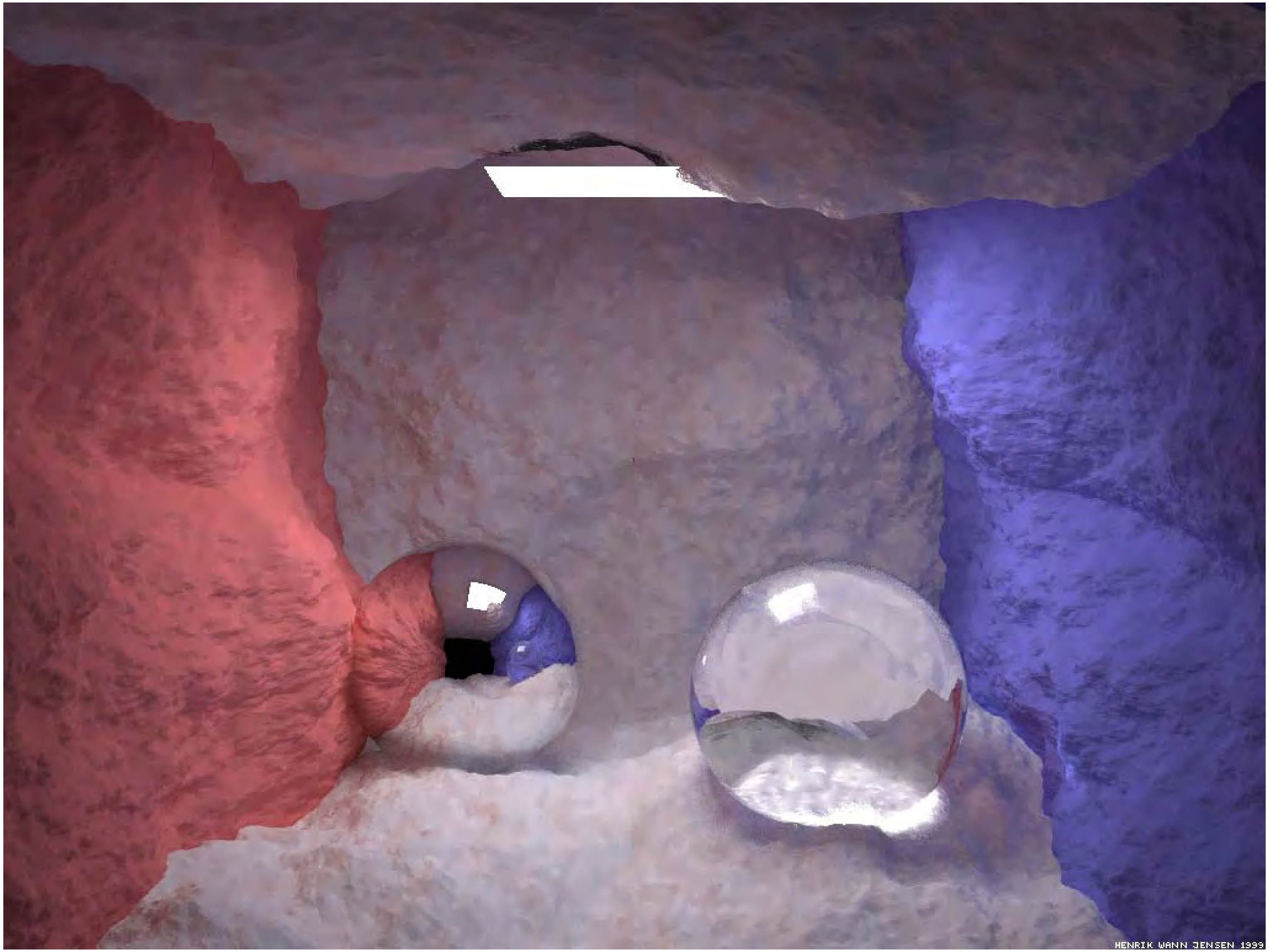
- Consider N random samples over domain **with probability *p(x)***
- Define estimator < I > as:

$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

- Probability *p* allows us to sample the domain more smartly

Monte-Carlo Recap

- Expected value is the integrand
  - Accurate “on average”
- Variance decrease in 1/N
  - Error decreases in $1/\sqrt{n}$


HENRIK WANN JENSEN 1999

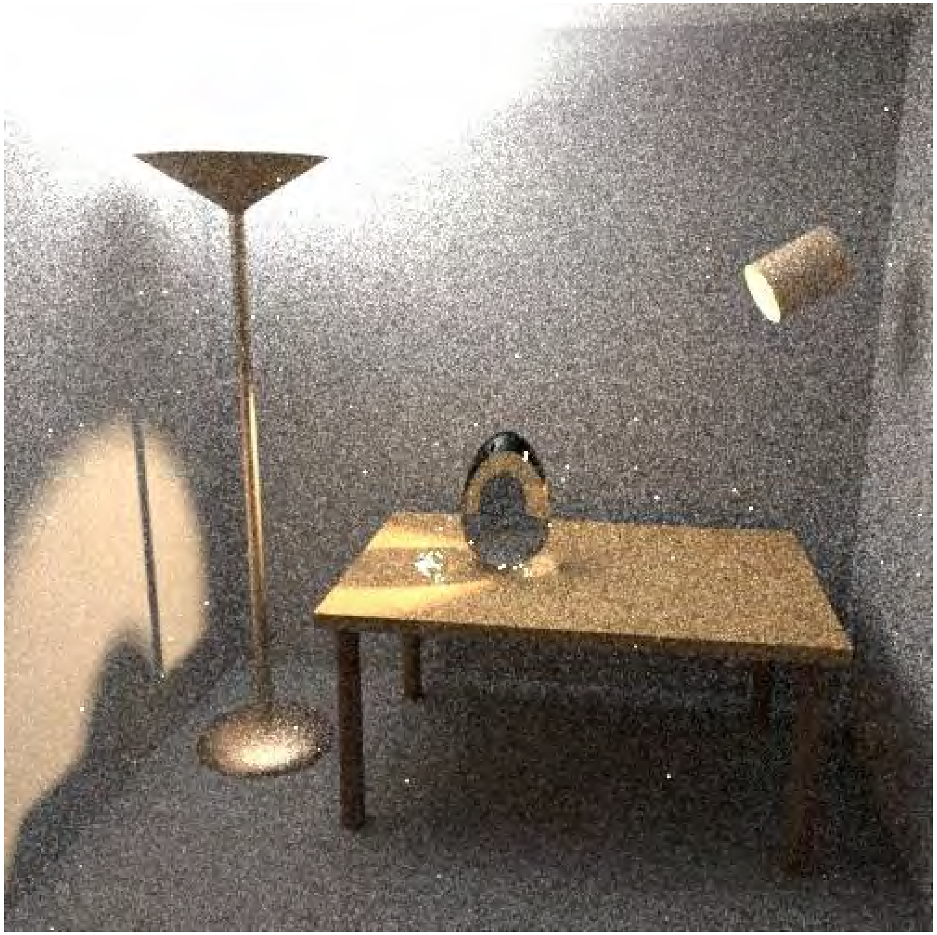
Advantages of MC Integration

- Few restrictions on the integrand
  - Doesn’t need to be continuous, smooth, ...
  - Only need to be able to evaluate at a point
- Extends to high-dimensional problems
  - Same convergence
- Conceptually straightforward
- Efficient for solving at just a few points

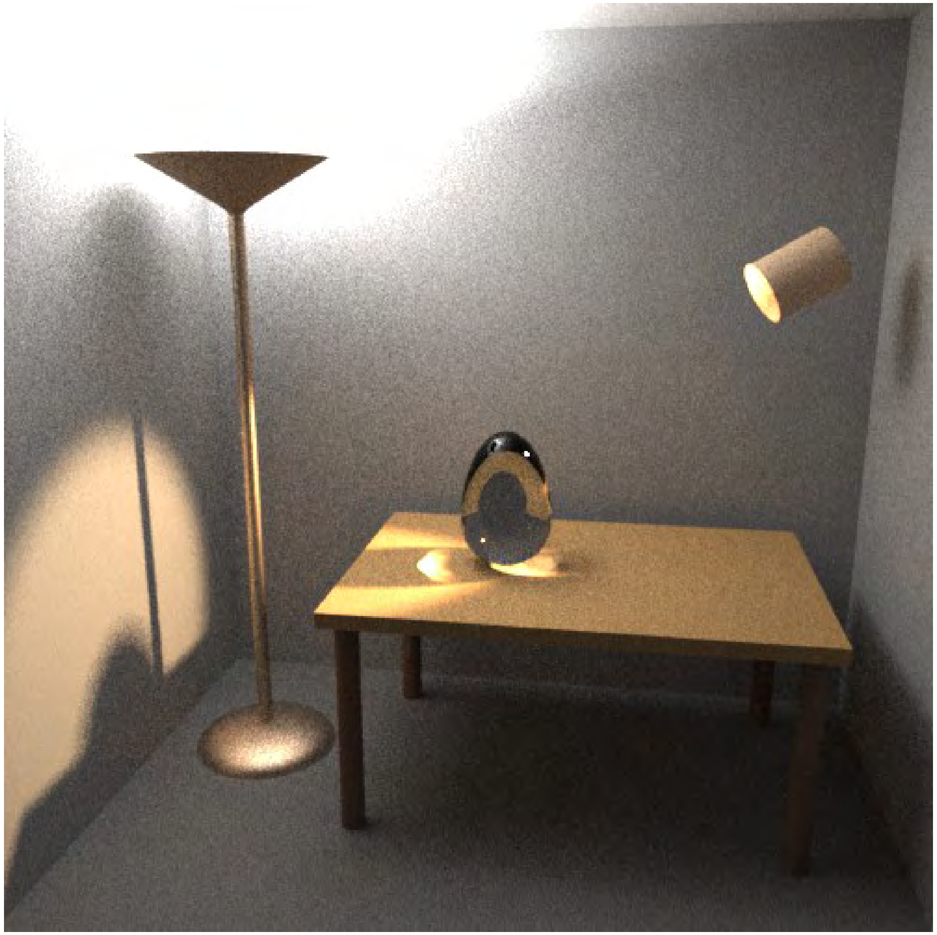
Disadvantages of MC

- Noisy
- Slow convergence
- Good implementation is hard
  - Debugging code
  - Debugging maths
  - Choosing appropriate techniques

- Images by Veach and Guibas



Naïve sampling strategy



Optimal sampling strategy

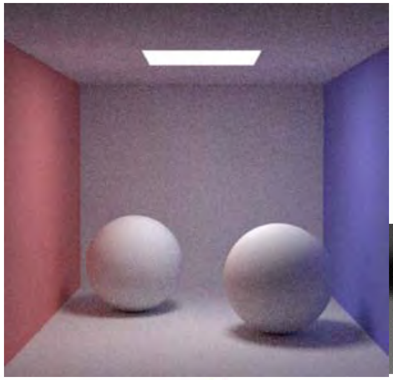
Today’s lecture

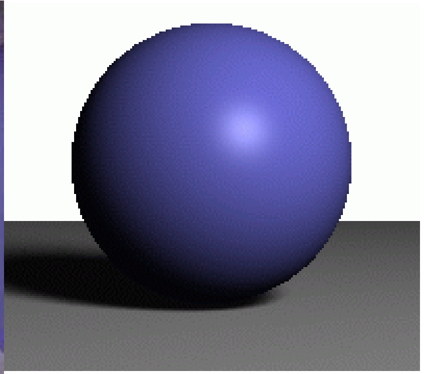
- Expected value and variance
- Analysis of Monte-Carlo integration
- Monte-Carlo in graphics
- Importance sampling
- Stratified sampling
- Global illumination
- Advanced Monte-Carlo rendering

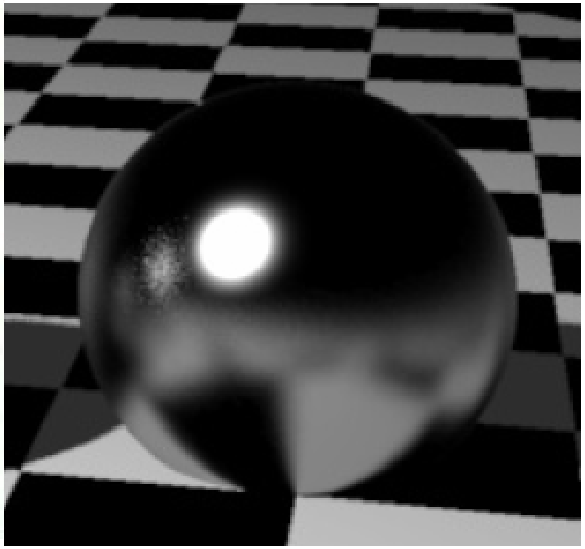
What can we integrate?

$$\int\int\int\int\int L(x, y, t, u, v)\,dx\,dy\,dt\,du\,dv$$

- Pixel: antialiasing
- Light sources: Soft shadows
- Lens: Depth of field
- Time: Motion blur
- BRDF: glossy reflection
- Hemisphere: indirect lighting







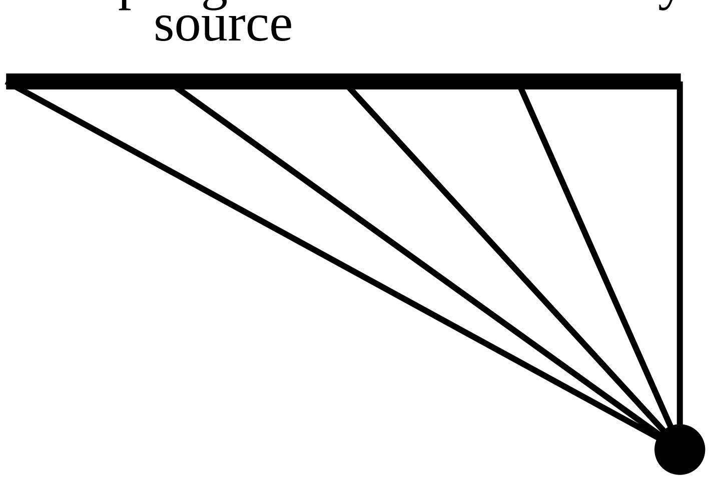
Domains of integration

- Pixel, lens (Euclidean 2D domain)
- Time (1D)
- Hemisphere
  - Work needed to ensure uniform probability
- Light source
  - Same thing: make sure that the probabilities and the measures are right.

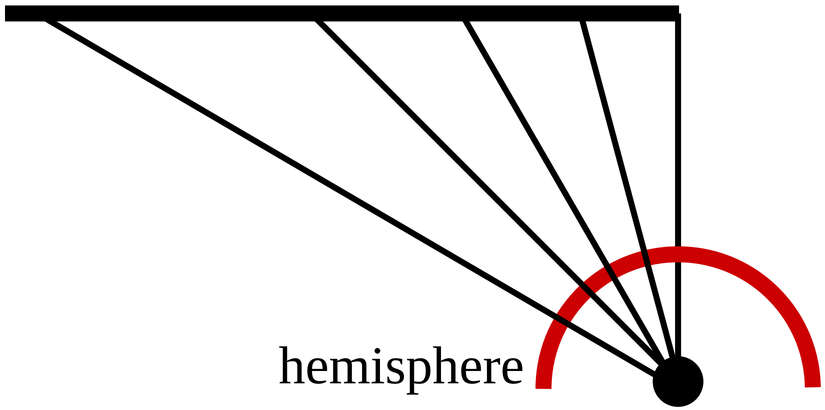
Example: Light source

- Integrate over surface or over angle
- Be careful to get probabilities and integration measure right!

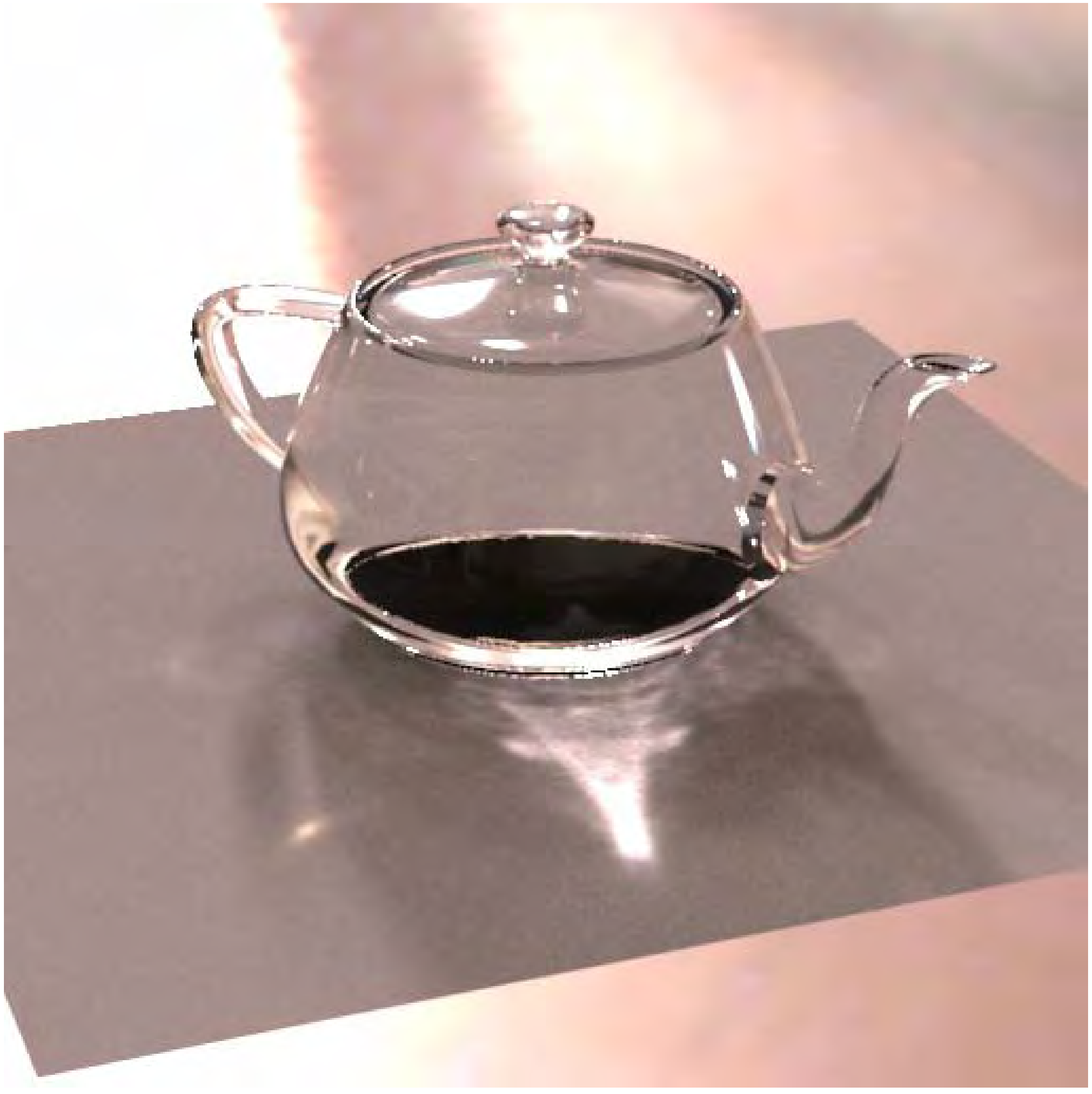
Sampling the source uniformly



Sampling the hemisphere uniformly



- Image by Henrik Wann Jensen



Today’s lecture

- Expected value and variance
- Analysis of Monte-Carlo integration
- Monte-Carlo in graphics
- Importance sampling
- Stratified sampling
- Global illumination
- Advanced Monte-Carlo rendering

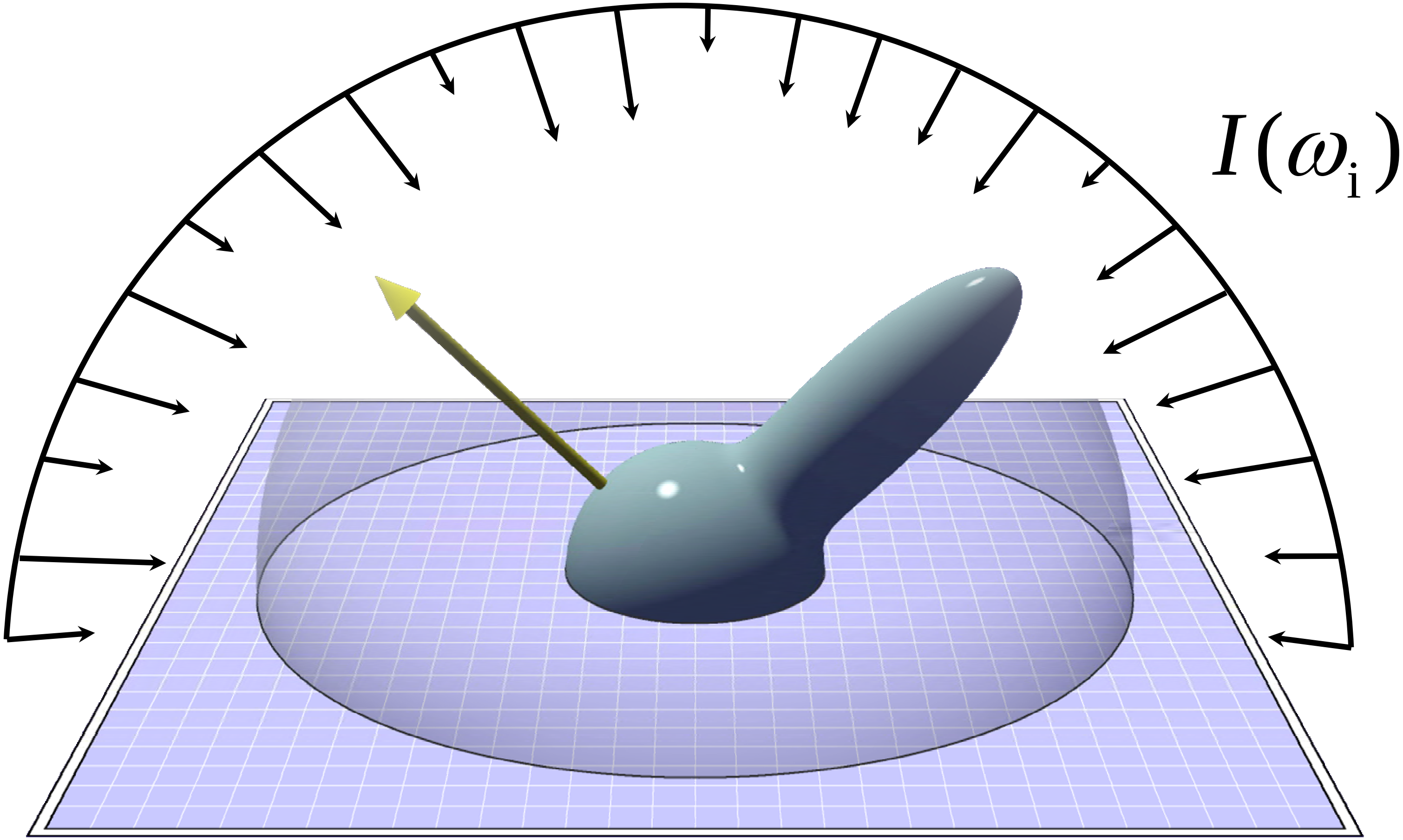
Important issues in MC rendering

- Reduce variance!
- Choose a smart probability distribution
- Choose smart sampling patterns

- And of course, cheat to make it faster without being noticed

Example: Glossy rendering

- Integrate over hemisphere
- BRDF times cosine times incoming light

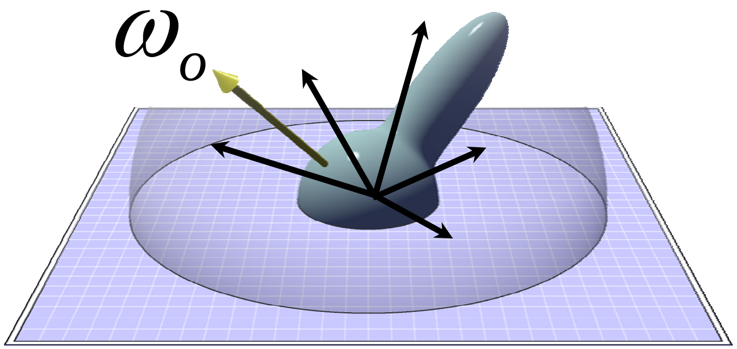


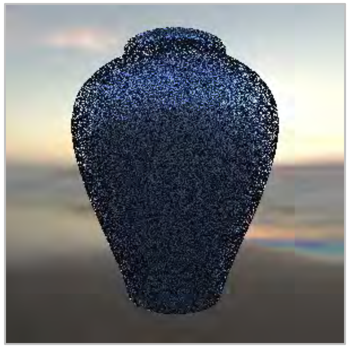
Slide courtesy of Jason Lawrence

Sampling a BRDF

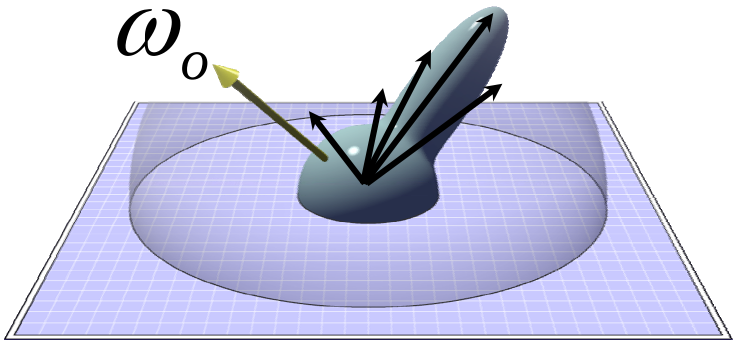
5 Samples/Pixel

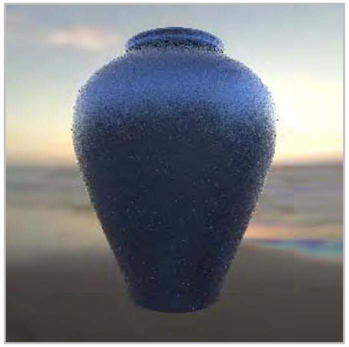
$U(\omega_i)$





$P(\omega_i)$



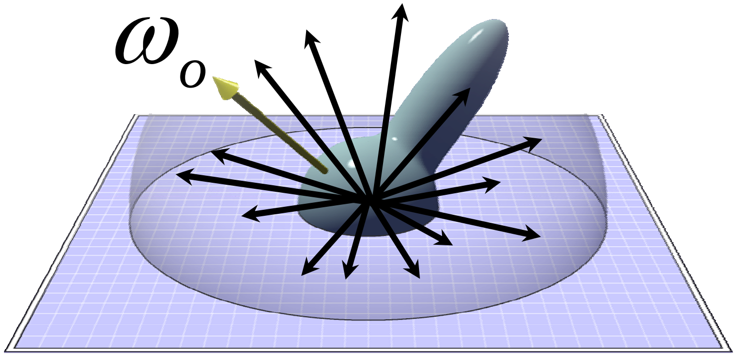


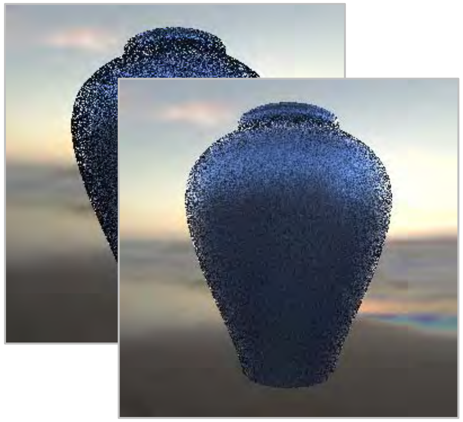
Slide courtesy of Jason Lawrence

Sampling a BRDF

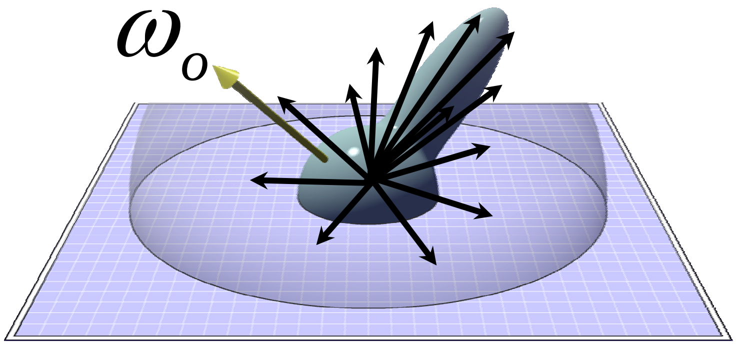
25 Samples/Pixel

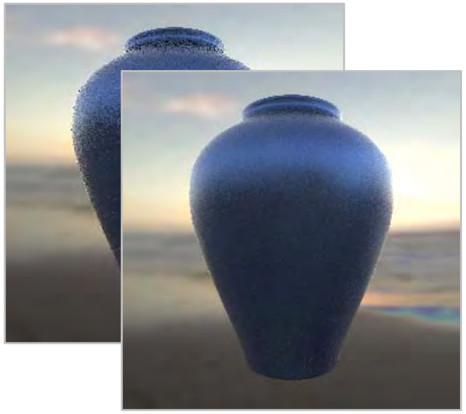
$U(\omega_i)$





$P(\omega_i)$

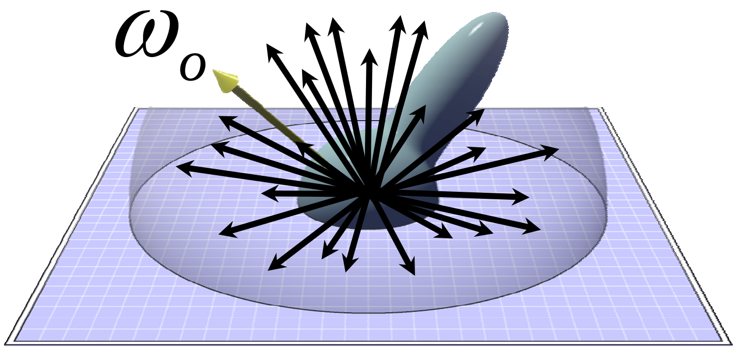




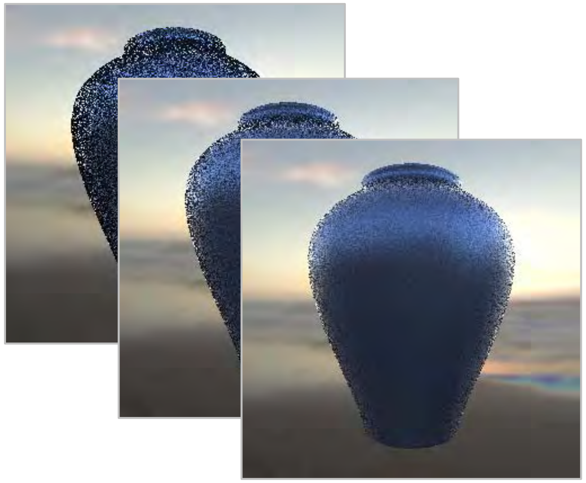
Slide courtesy of Jason Lawrence

Sampling a BRDF

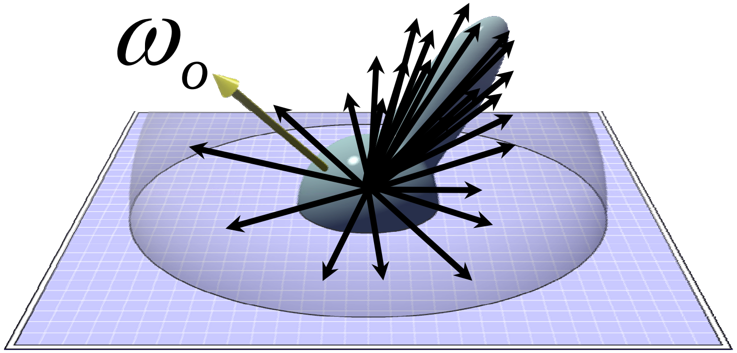
$U(\omega_i)$

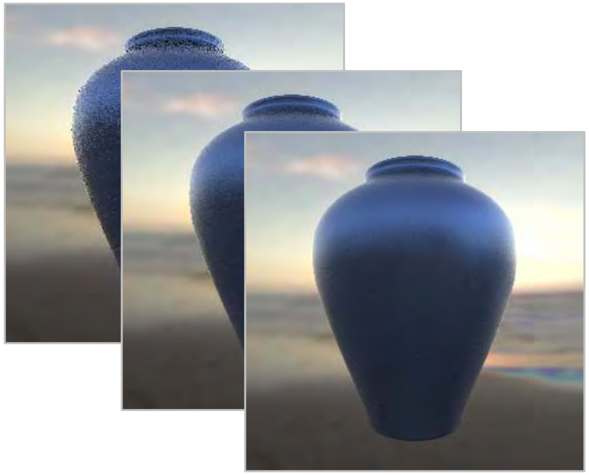


75 Samples/Pixel



$P(\omega_i)$



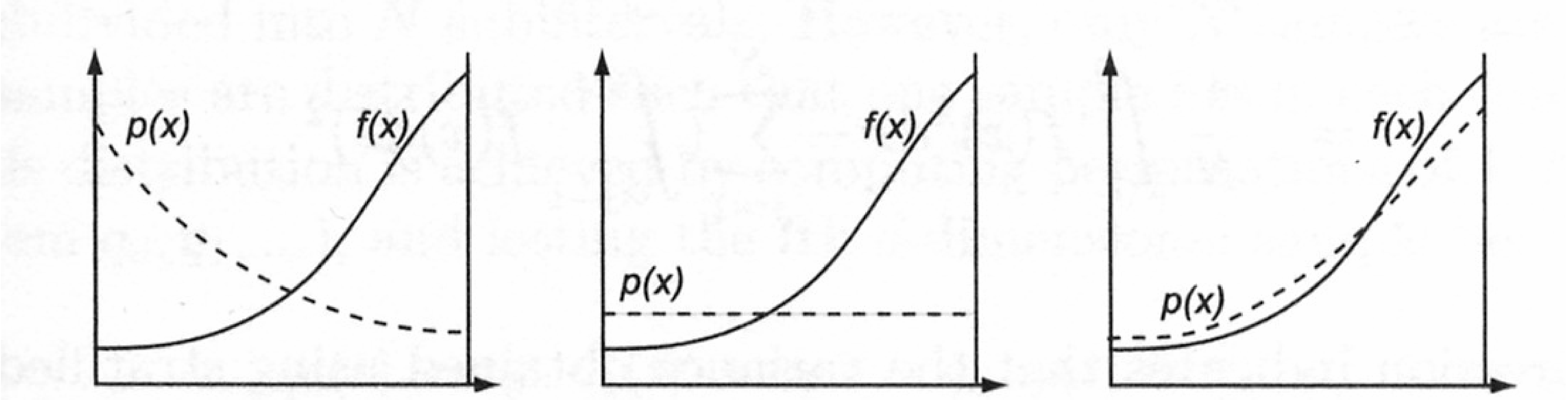


Slide courtesy of Jason Lawrence

Importance sampling

$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

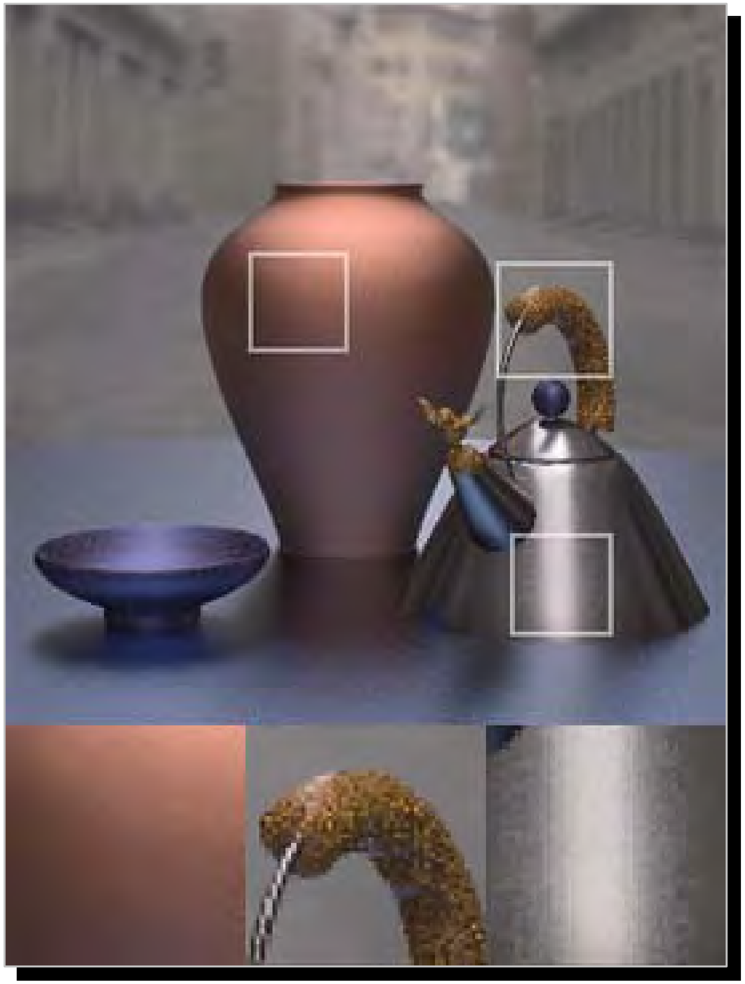
- Choose *p* wisely to reduce variance
  - *p* that resembles *f*
  - Does not change convergence rate (still sqrt)



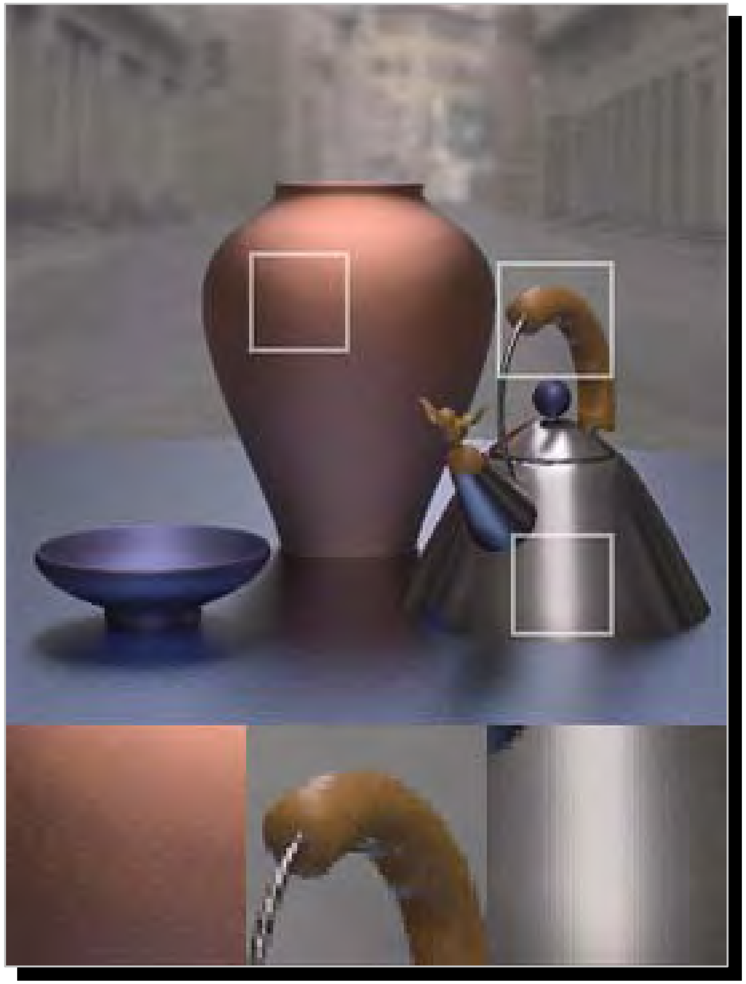
bad uniform good

Questions?

1200 Samples/Pixel



Traditional importance function

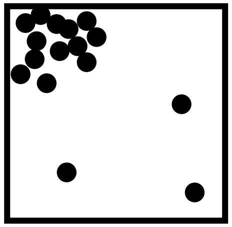


Better importance by Lawrence et al.

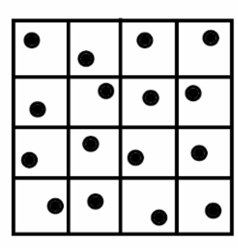
Today’s lecture

- Expected value and variance
- Analysis of Monte-Carlo integration
- Monte-Carlo in graphics
- Importance sampling
- Stratified sampling
- Global illumination
- Advanced Monte-Carlo rendering

Stratified sampling



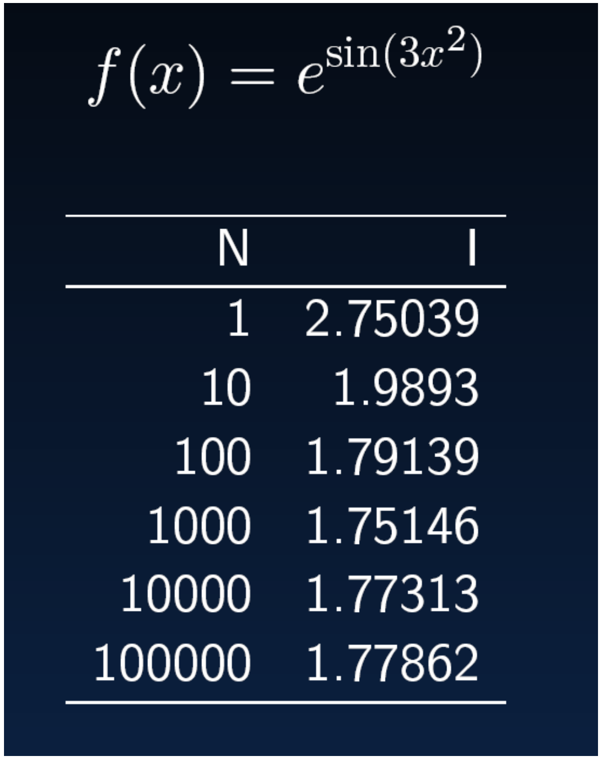
- With uniform sampling, we can get unlucky
  - E.g. all samples in a corner
- To prevent it, subdivide domain Ω into non-overlapping regions Ω_i
  - Each region is called a stratum



- Take one random sample per Ω_i

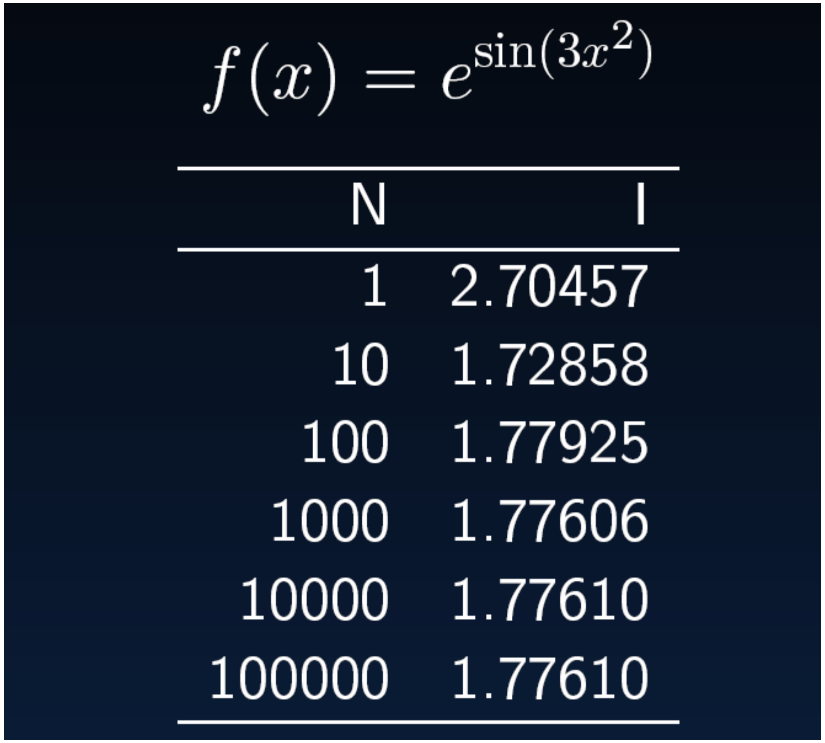
Example

- Borrowed from Henrik Wann Jensen



$$f(x) = e^{\sin(3x^2)}$$

N	I
1	2.75039
10	1.9893
100	1.79139
1000	1.75146
10000	1.77313
100000	1.77862



$$f(x) = e^{\sin(3x^2)}$$

N	I
1	2.70457
10	1.72858
100	1.77925
1000	1.77606
10000	1.77610
100000	1.77610

Unstratified

$$O(1/\sqrt{N})$$

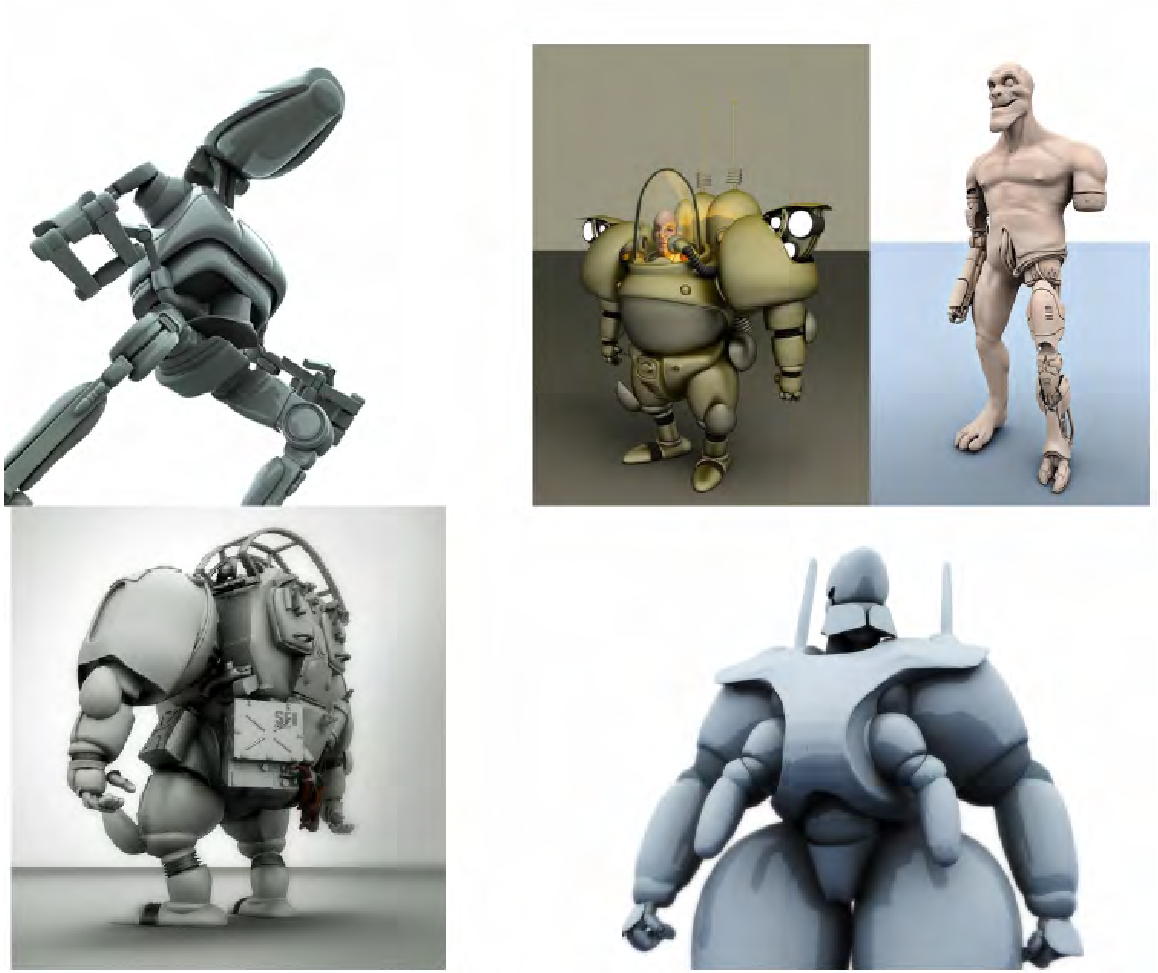
Stratified

$$O(1/N)$$

Stratified sampling - bottomline

- Cheap and effective
- Typical example: jittering for antialiasing
  - Signal processing perspective:
    better than uniform because less aliasing (spatial patterns)
  - Monte-Carlo perspective: better than random because lower variance (error for a given pixel)

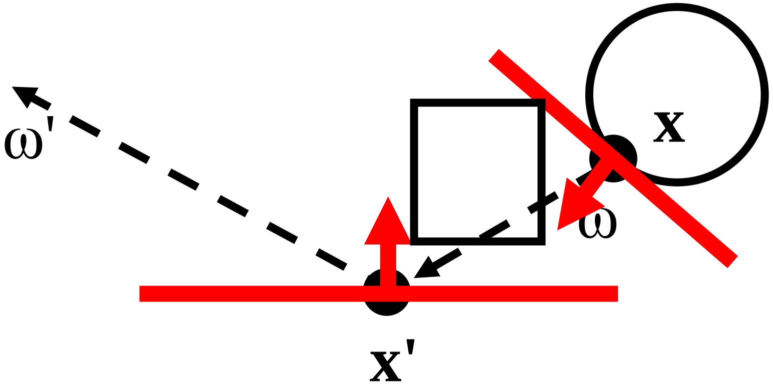
- Image from the ARNOLD Renderer by Marcos Fajardo



Monte Carlo Methods

- Expected value and variance
- Analysis of Monte-Carlo integration
- Monte-Carlo in graphics
- Importance sampling
- Stratified sampling
- Global illumination
- Advanced Monte-Carlo rendering

Recall The Rendering Equation



$$L(x',\omega') = E(x',\omega') + \int \rho_{x'}(\omega,\omega')L(x,\omega)G(x,x')V(x,x')\, dA$$

emission

BRDF

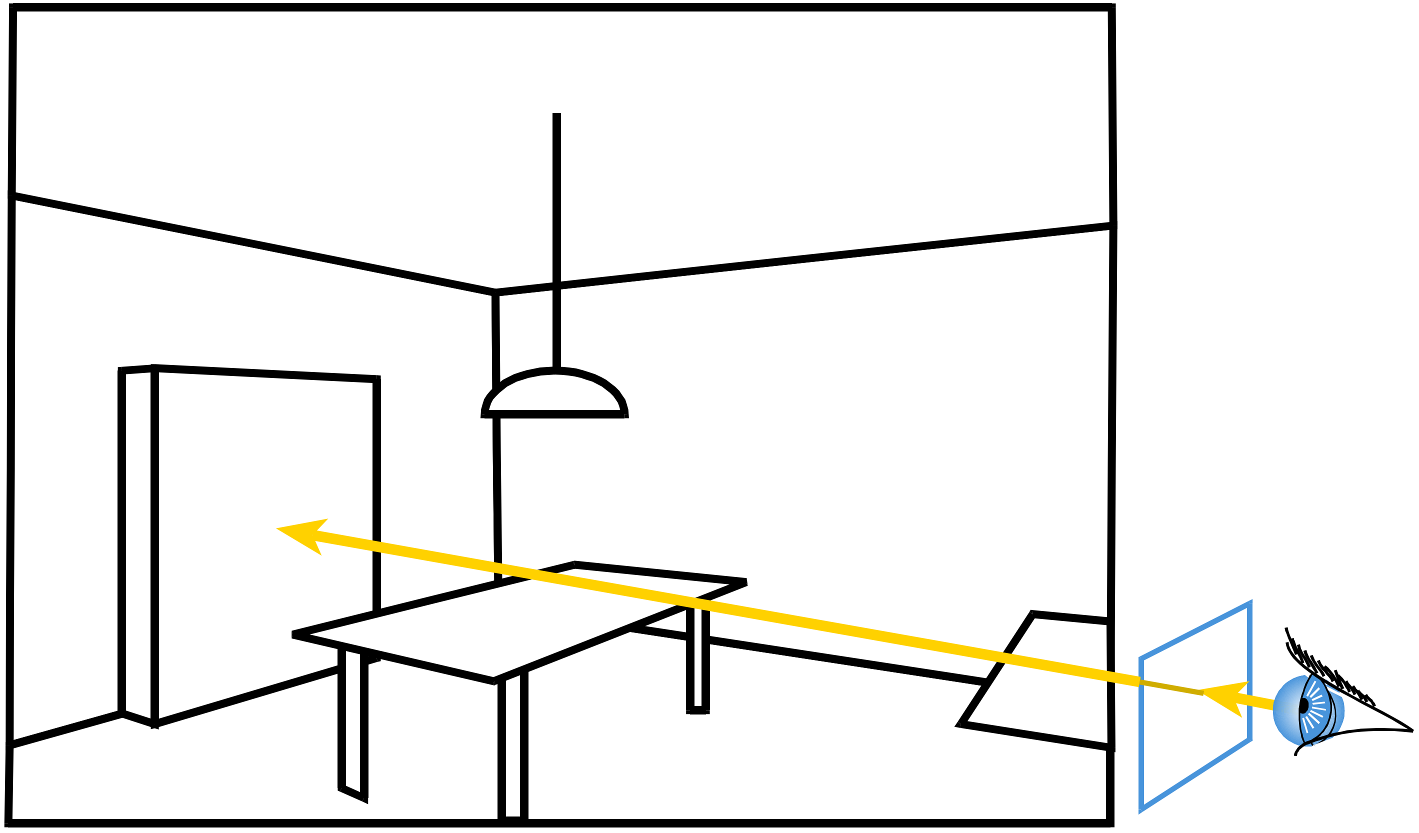
Incoming light

Geometric term

visibility

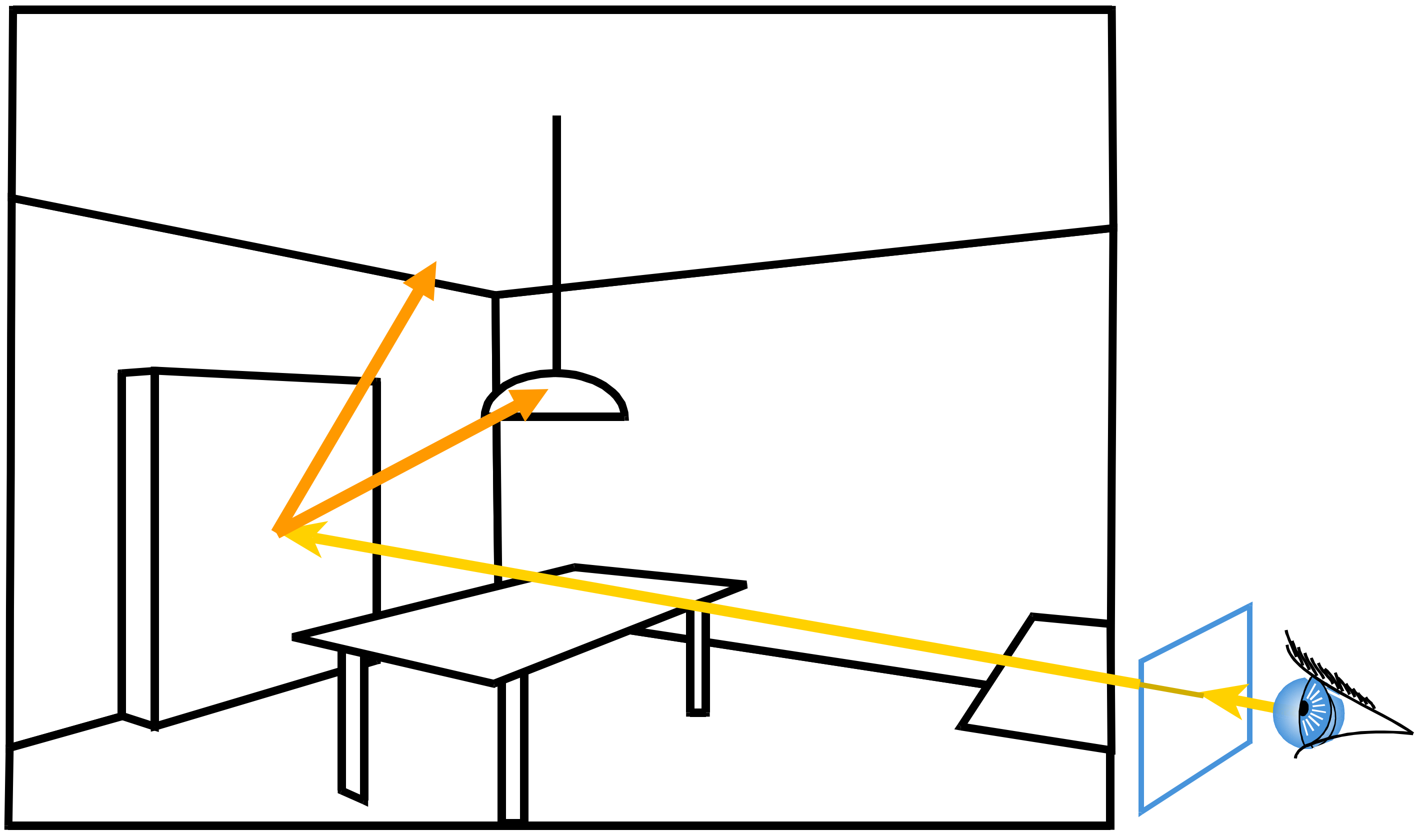
Ray Casting

- Cast a ray from the eye through each pixel



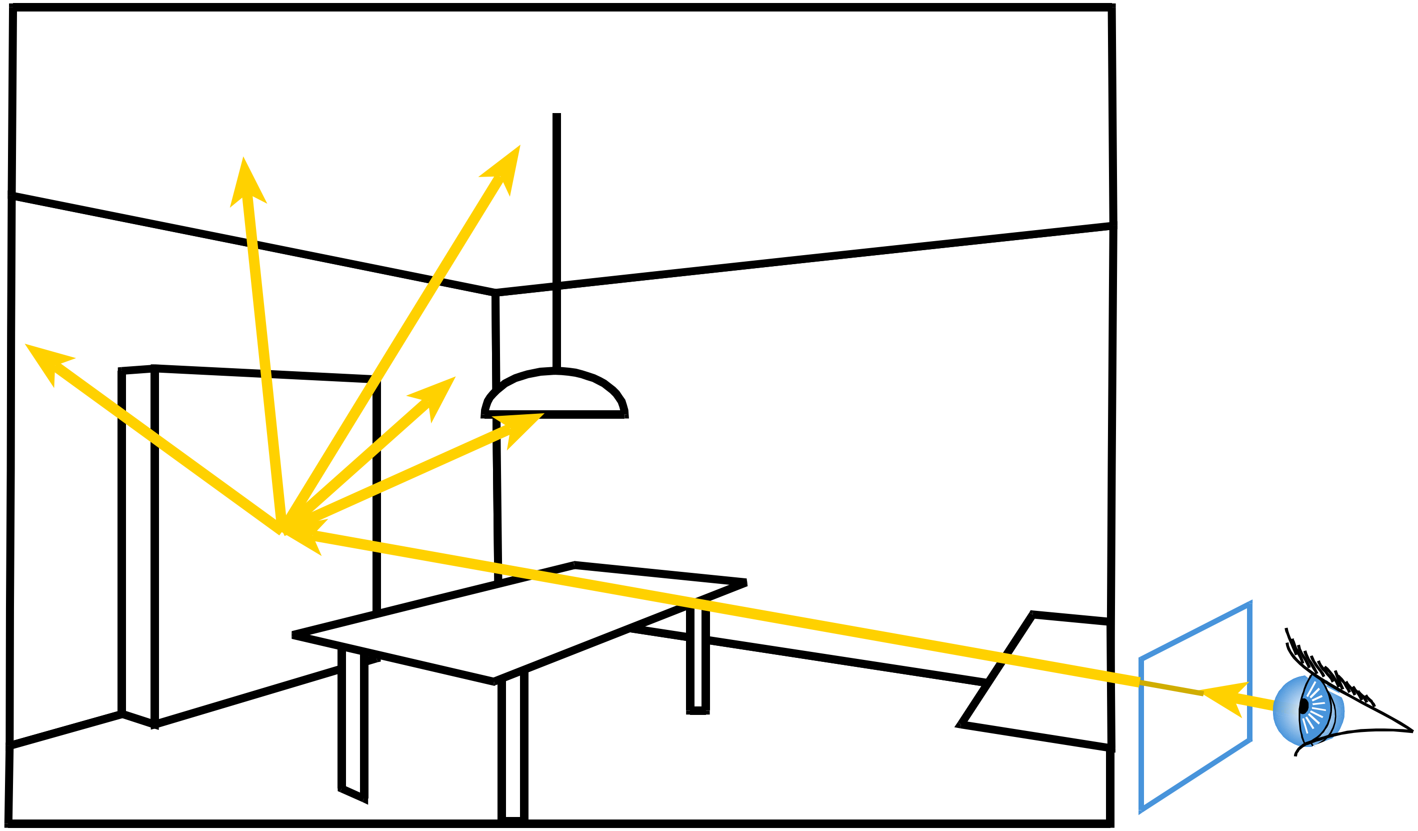
Ray Tracing

- Cast a ray from the eye through each pixel
- Trace secondary rays (light, reflection, refraction)



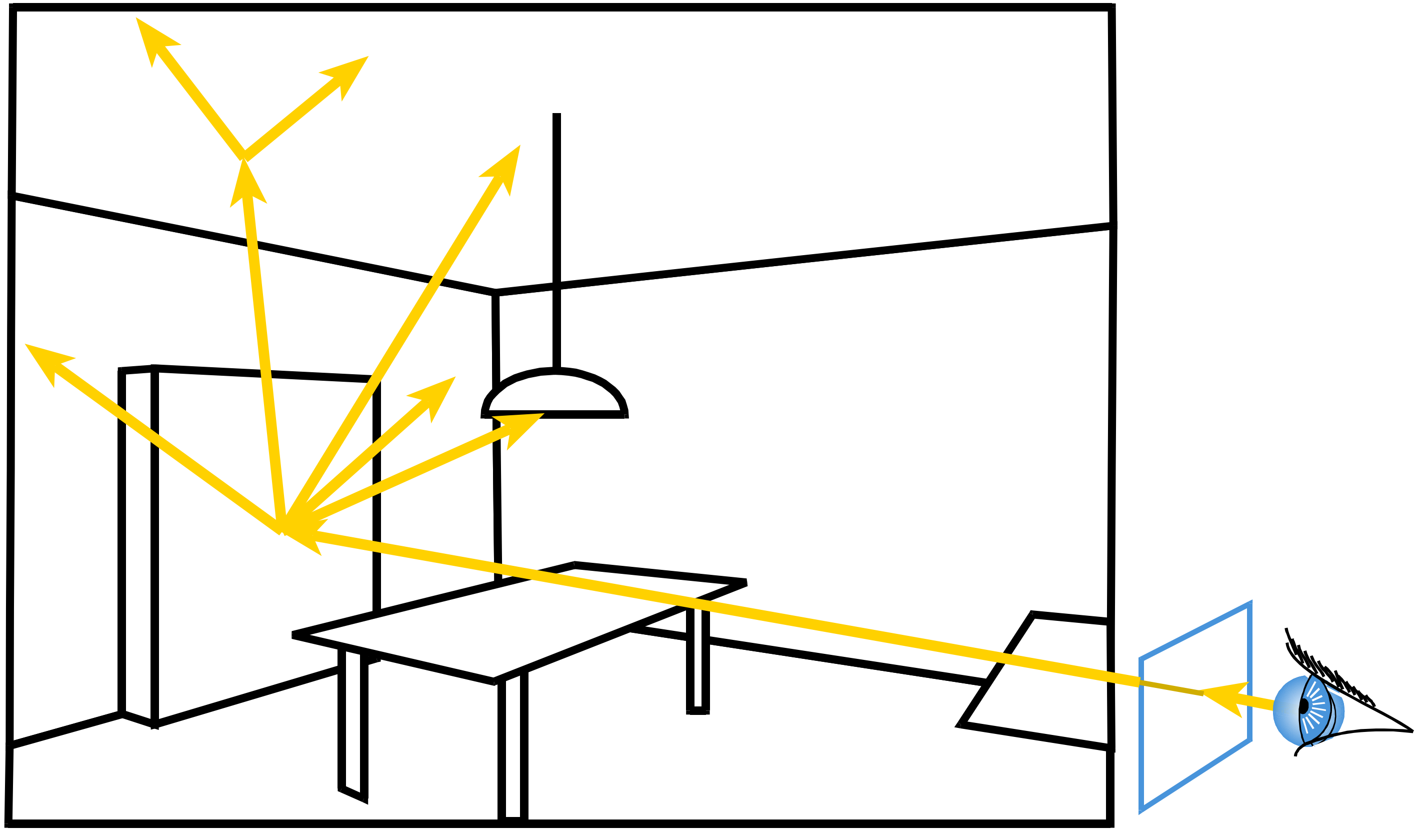
Monte-Carlo Ray Tracing

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
  - Accumulate radiance contribution



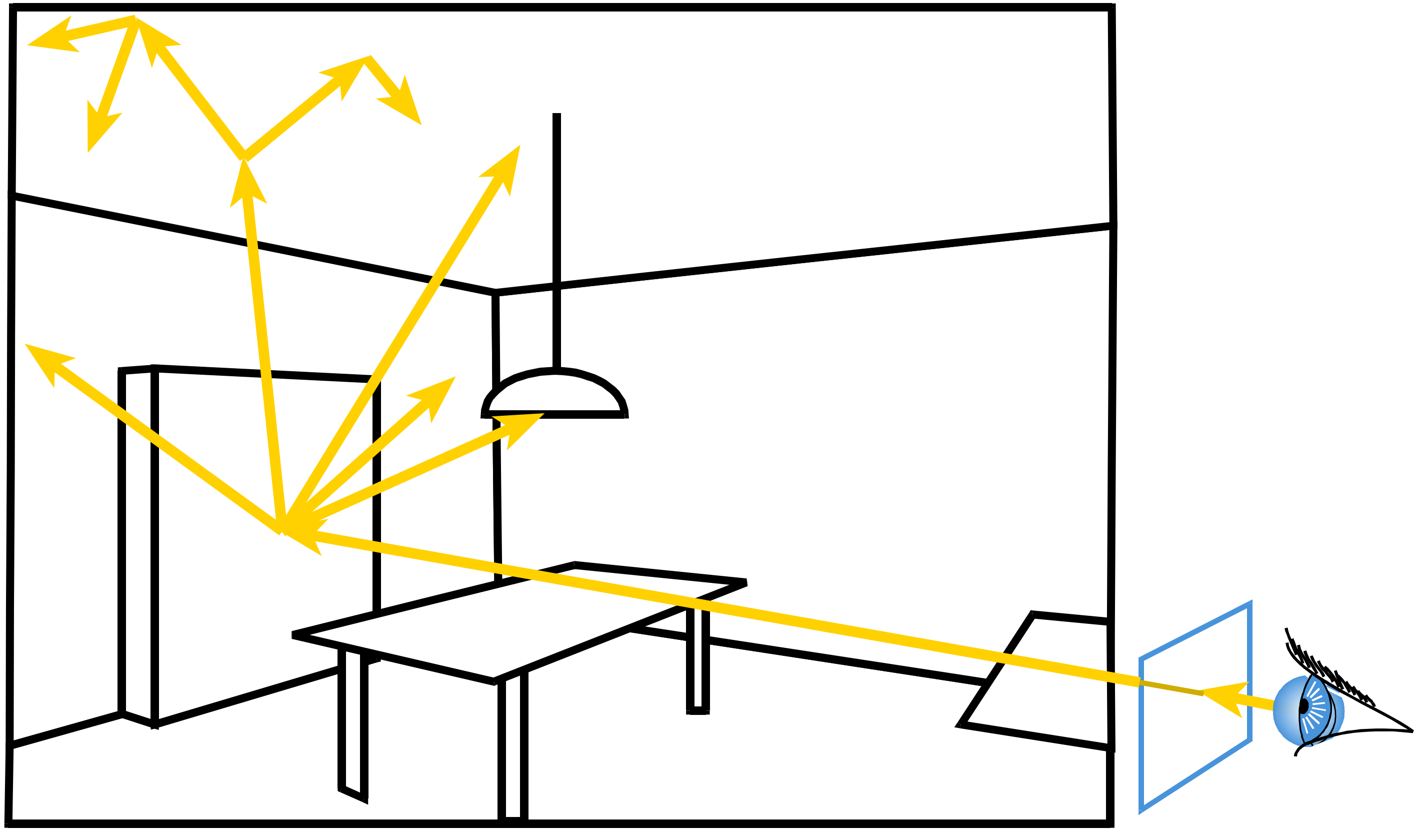
Monte-Carlo Ray Tracing

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse



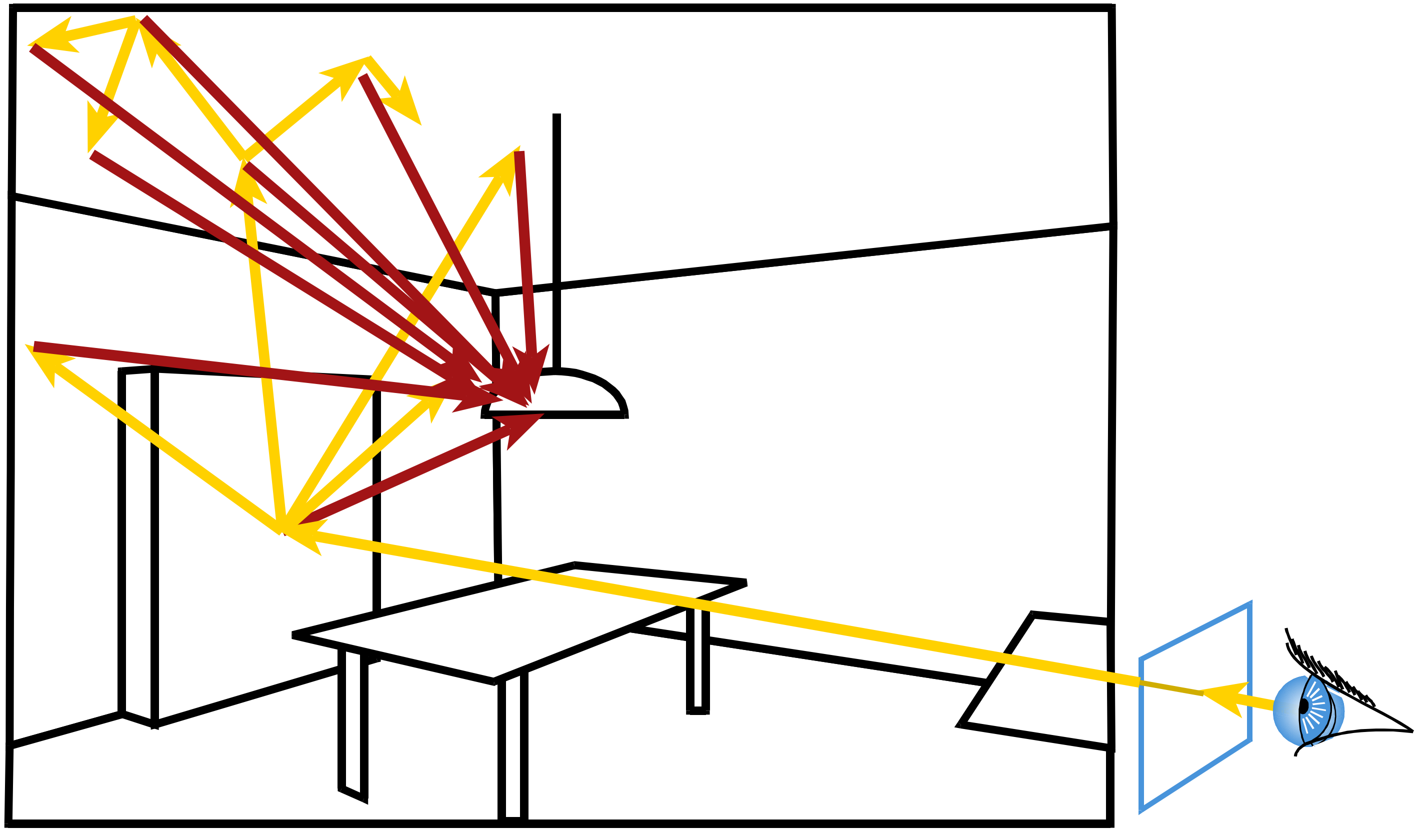
Monte-Carlo

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse

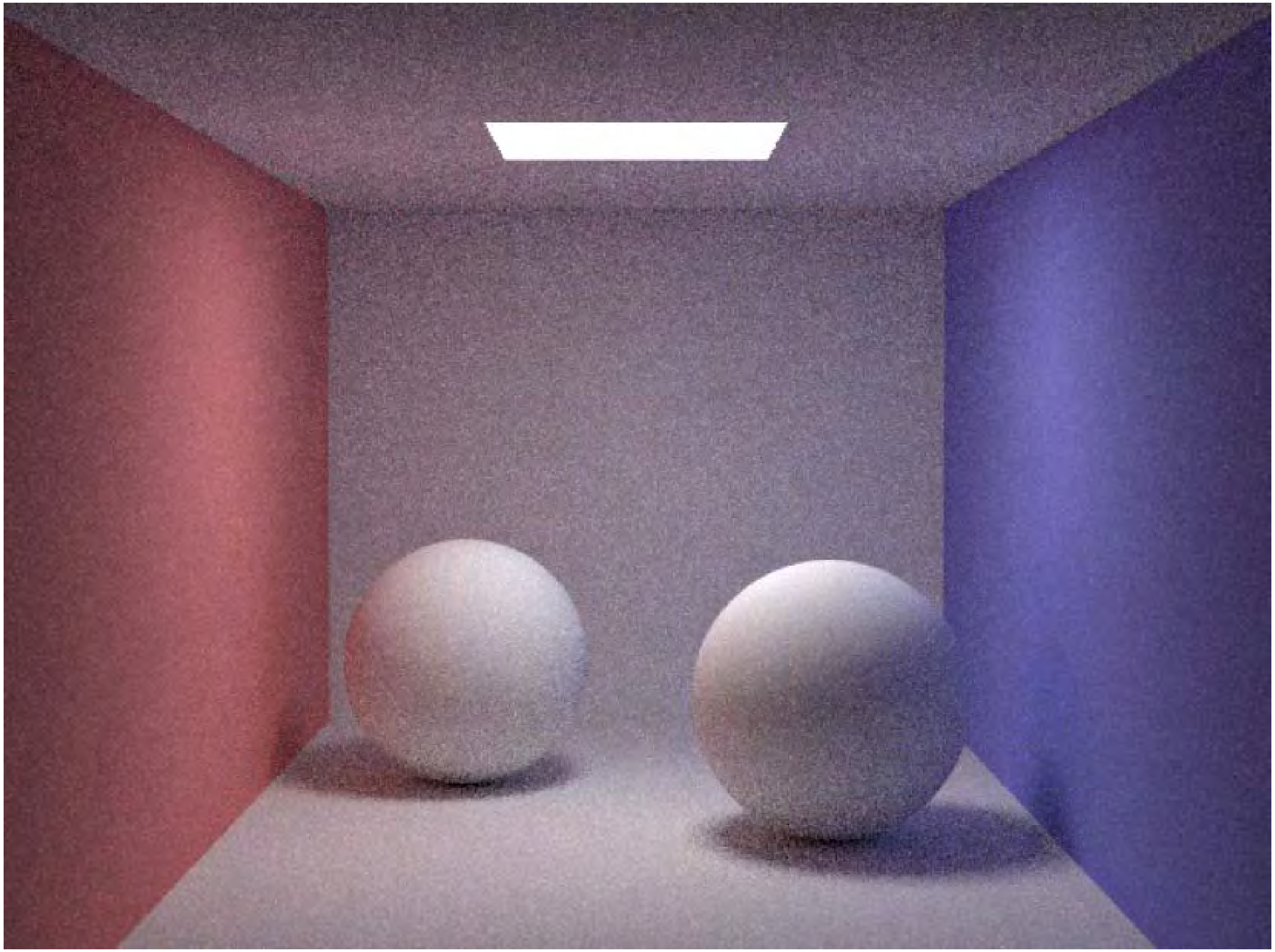


Monte-Carlo

- Systematically sample primary light

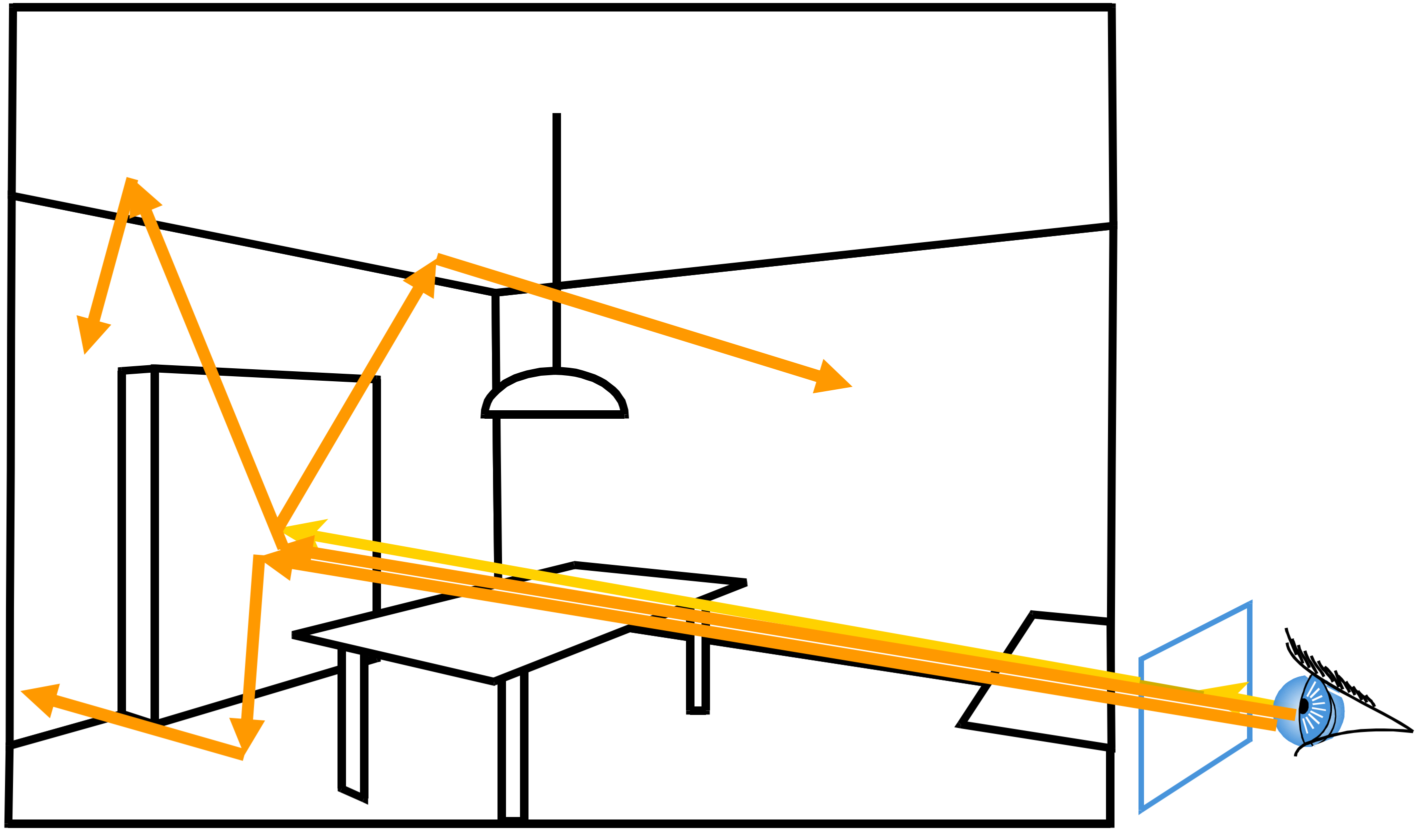


Results



Monte Carlo Path Tracing

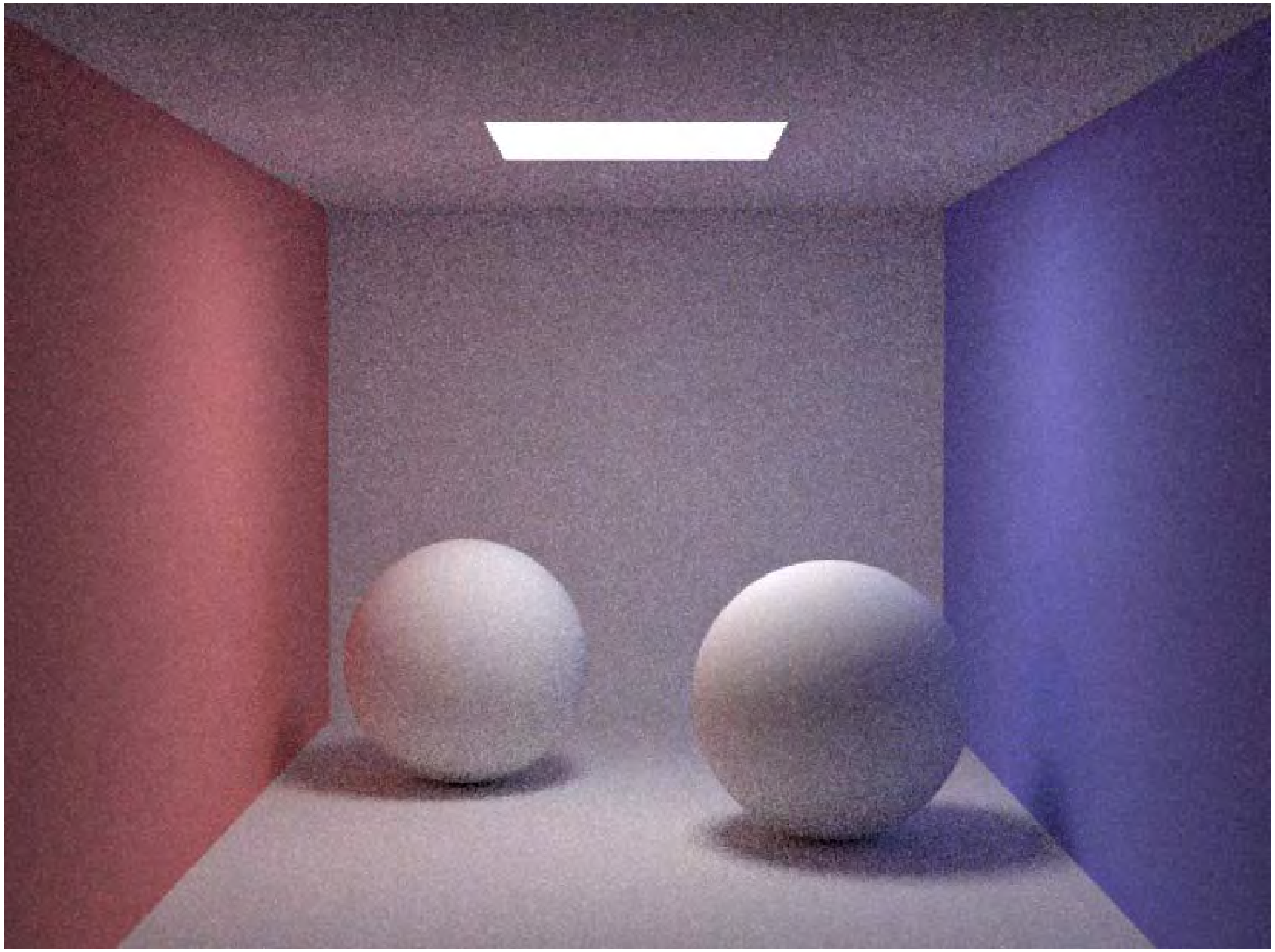
- Trace only one secondary ray per recursion
- But send many primary rays per pixel
- (performs antialiasing as well)



Results

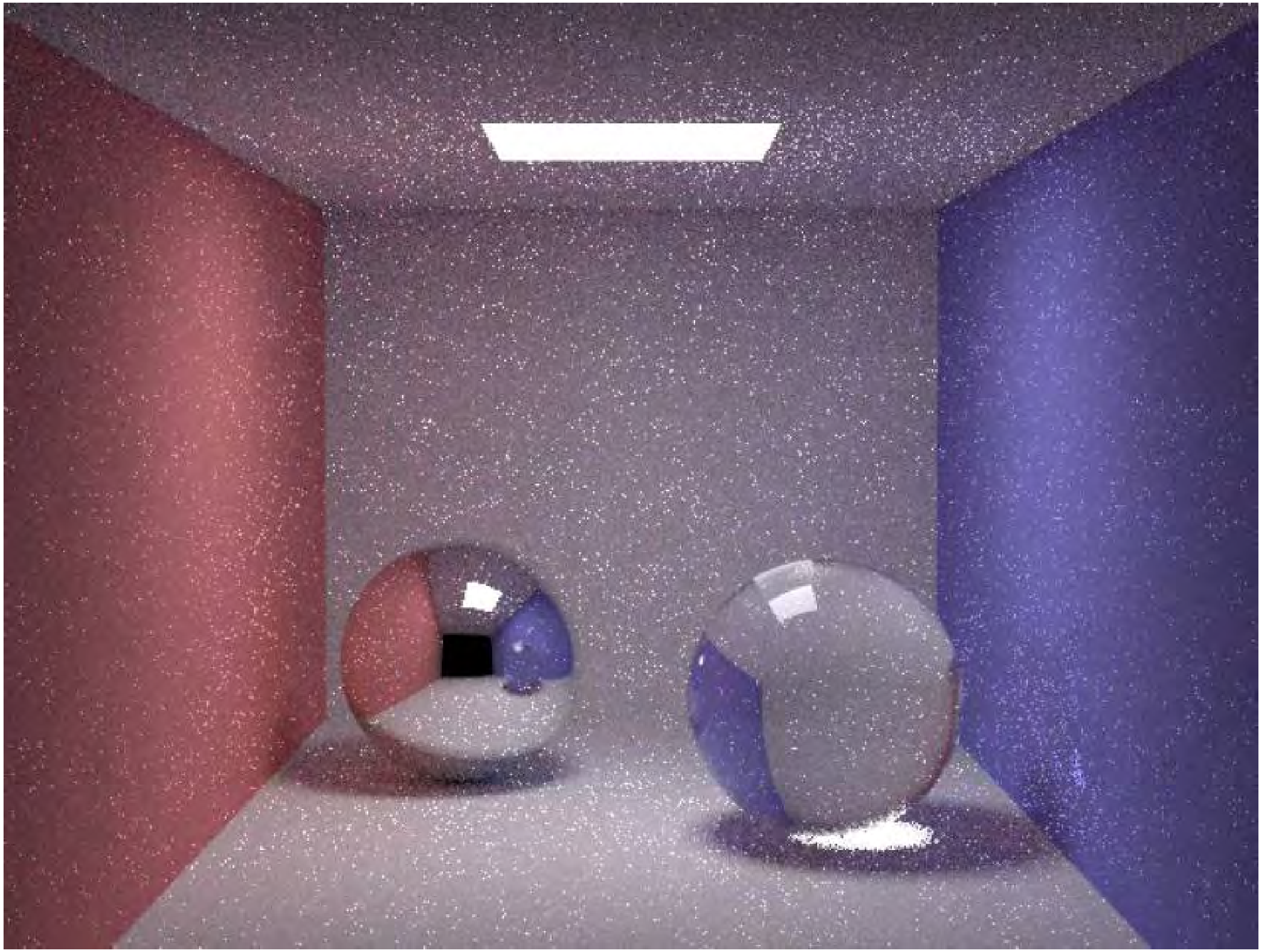
- 10 paths/pixel

Think about it : we compute an
infinite-dimensional integral with 10 samples!!!



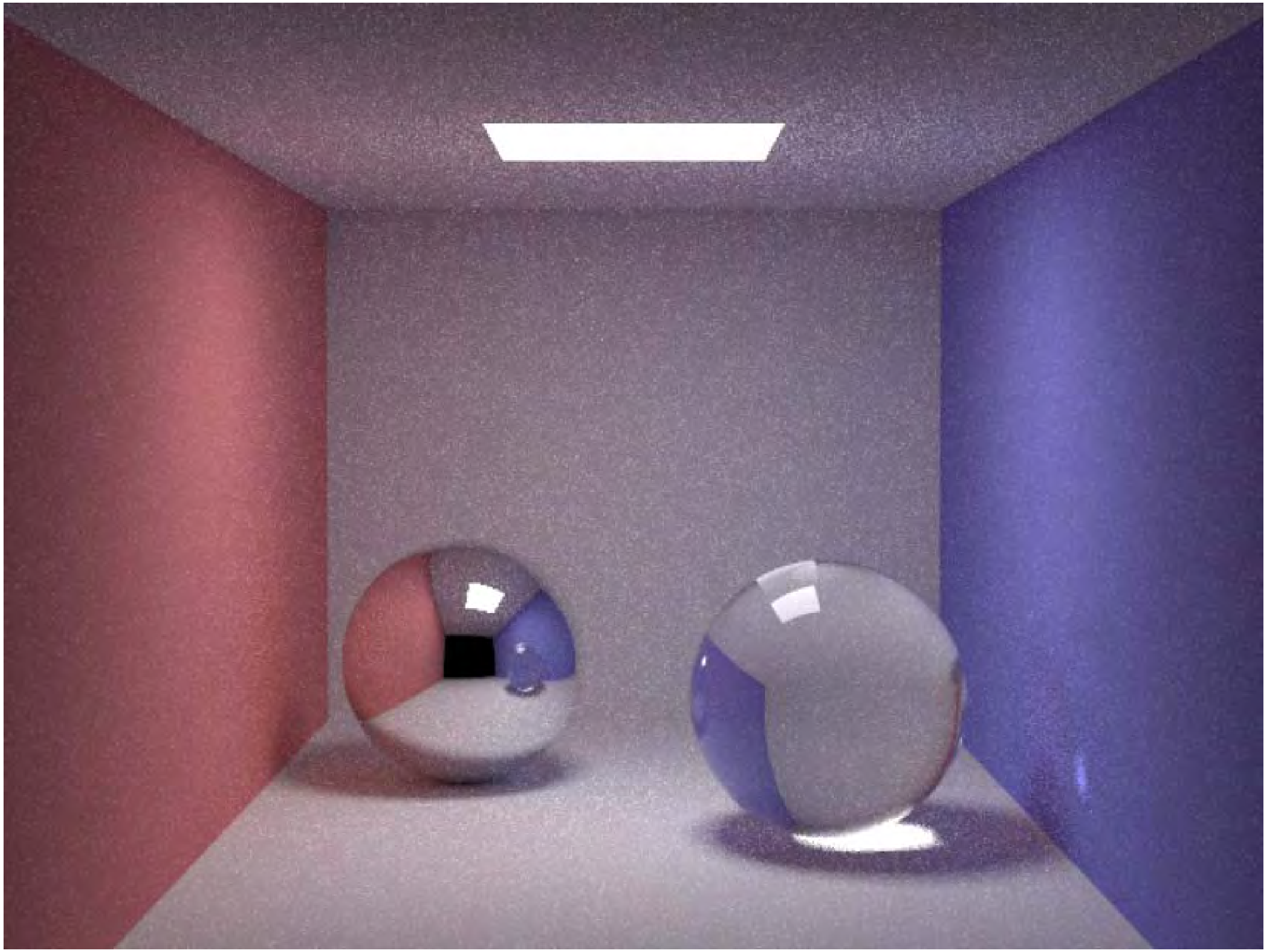
Results: glossy

- 10 paths/pixel

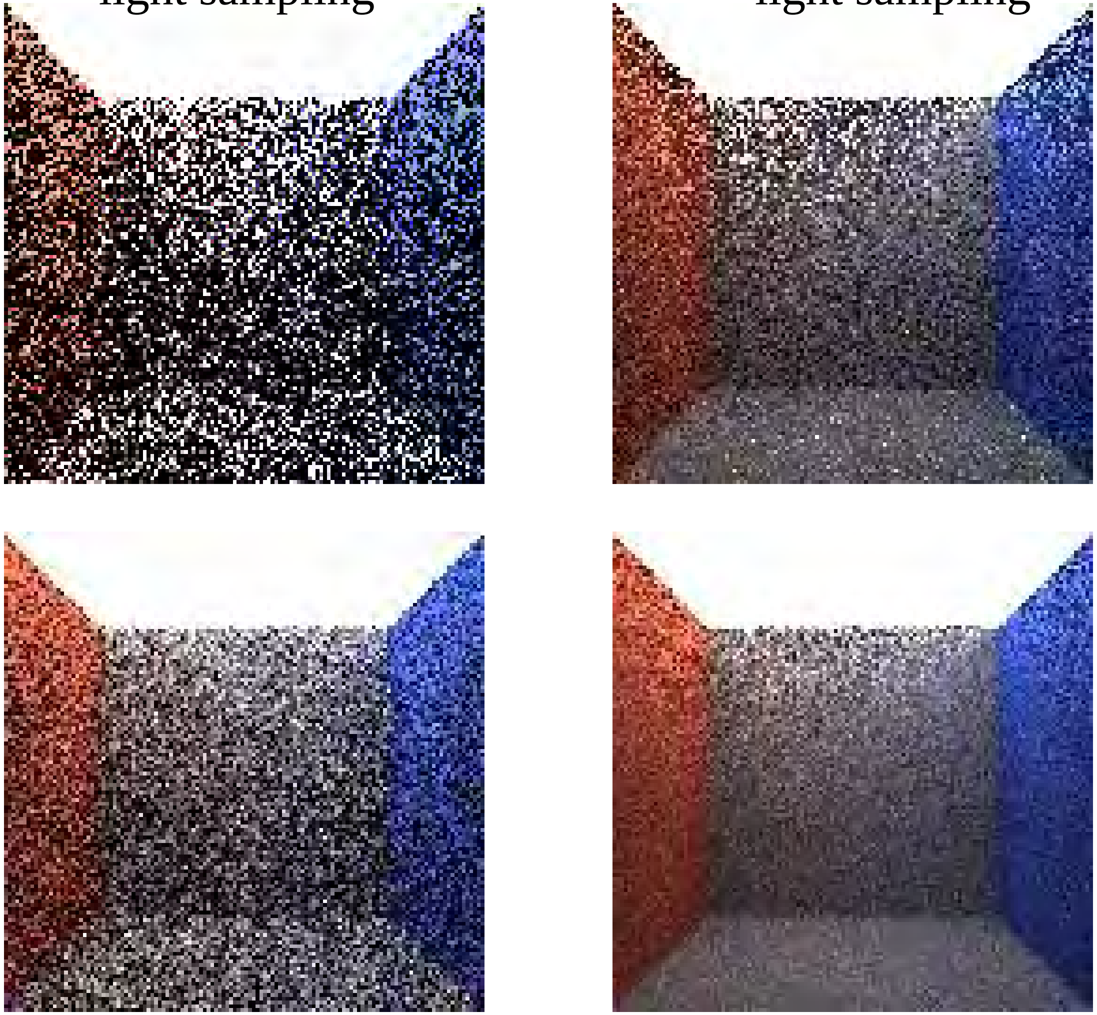


Results: glossy

- 100 paths/pixel



Importance of sampling the light



Without explicit light sampling

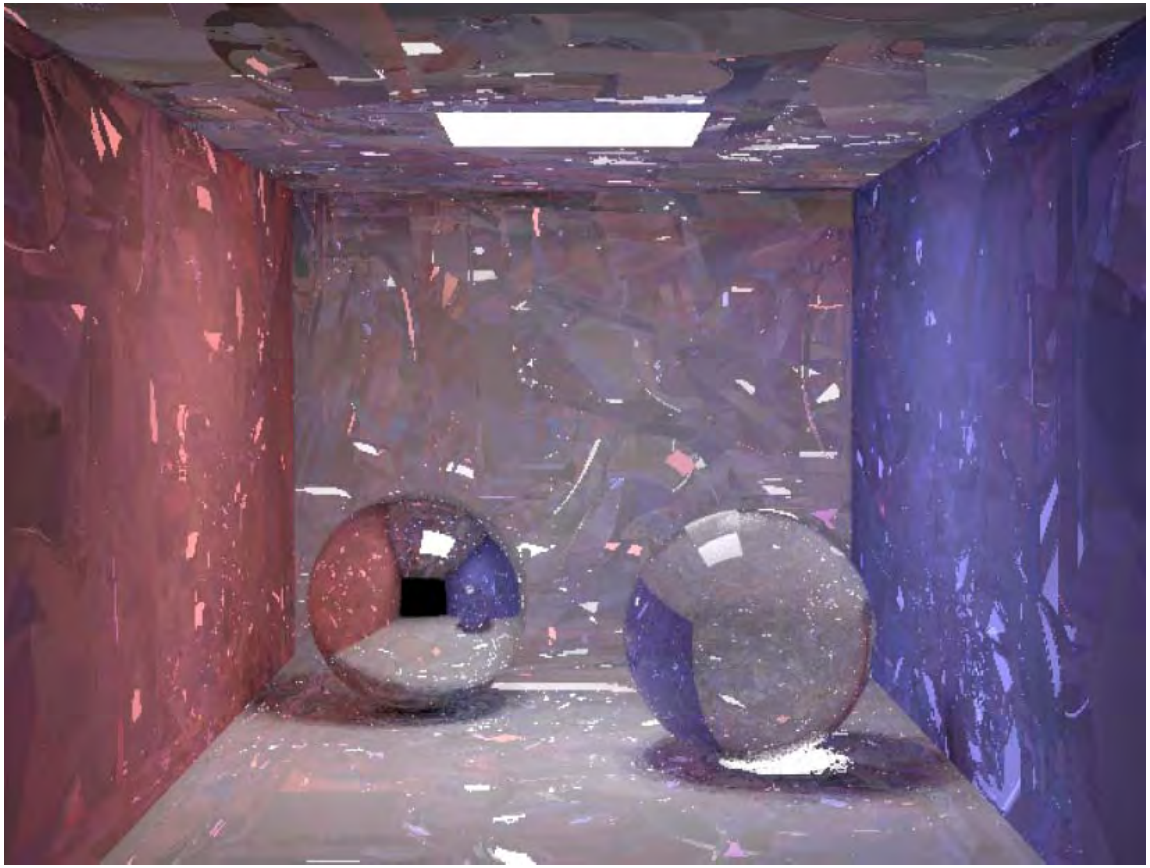
With explicit light sampling

1 path per pixel

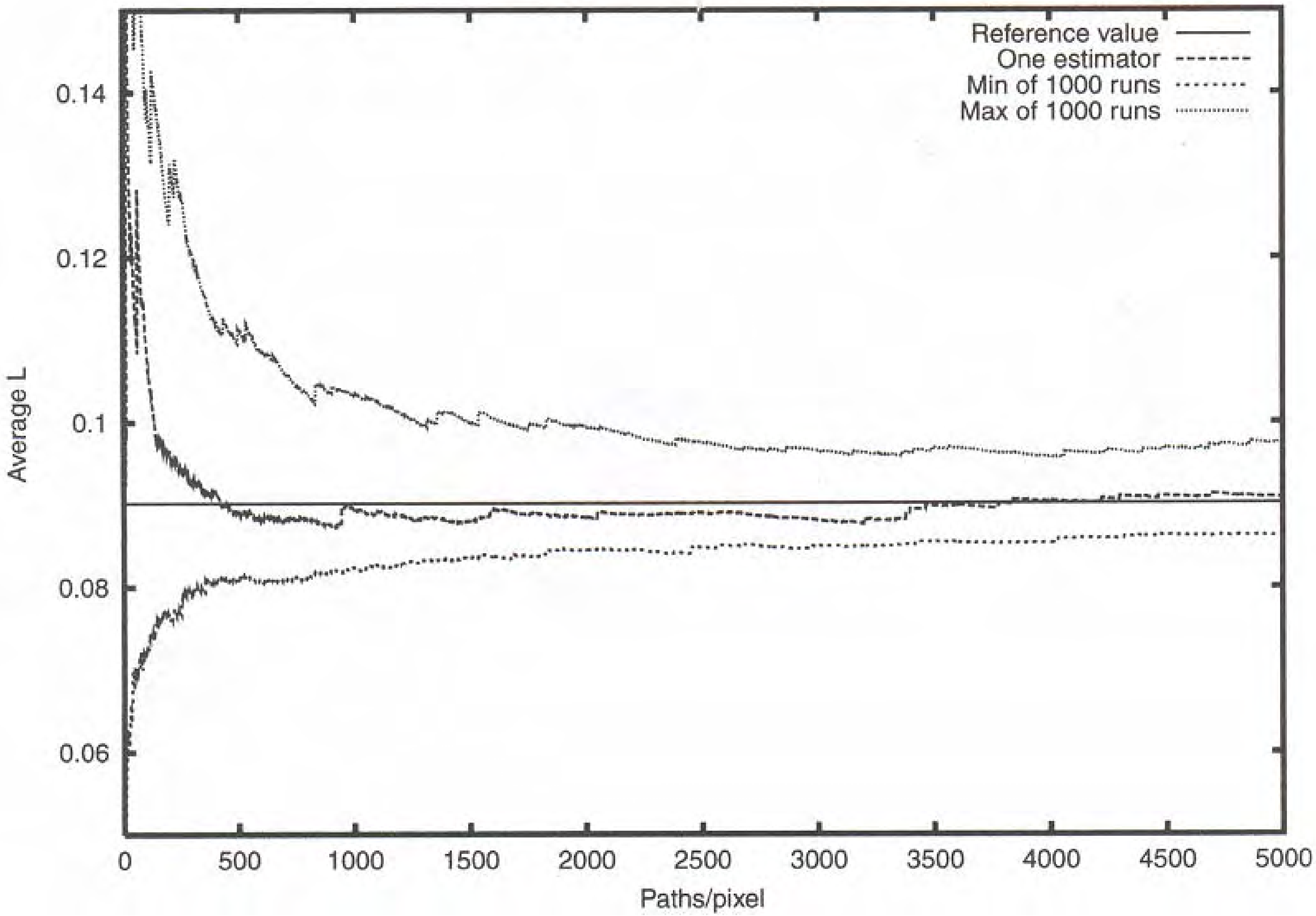
4 path per pixel

Why use random numbers?

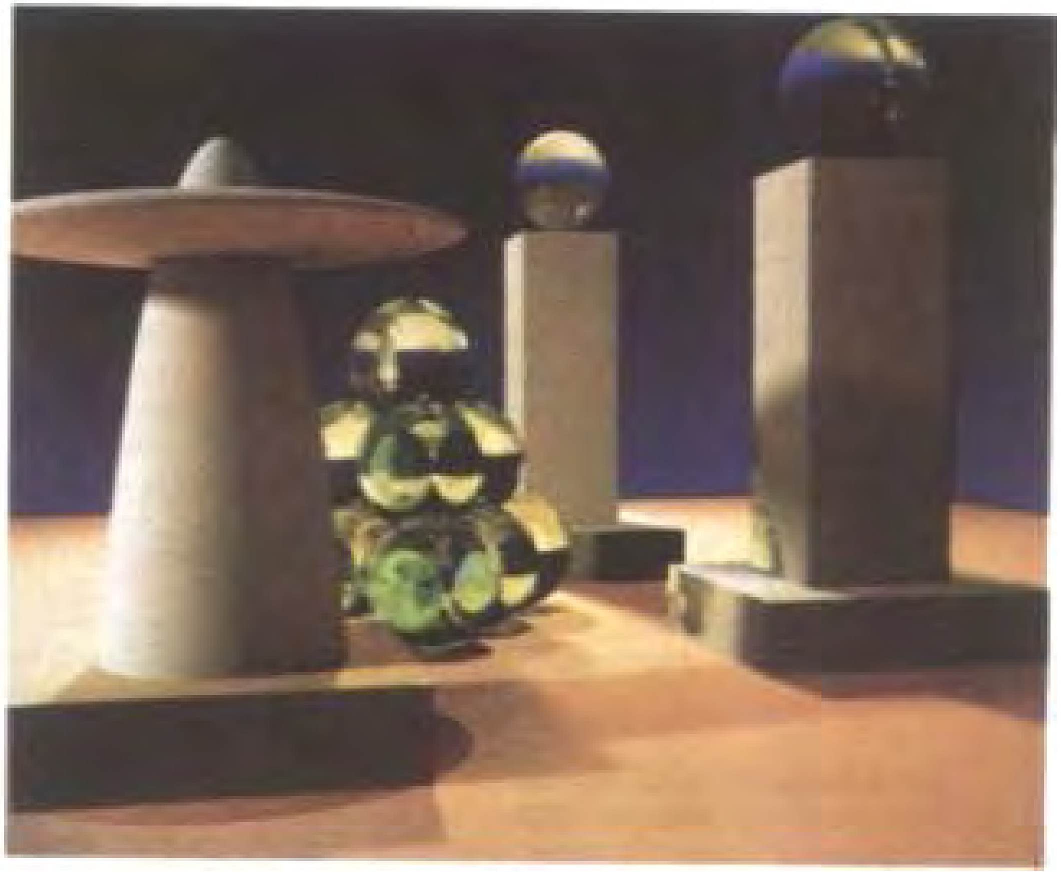
- Fixed random sequence
- We see the structure in the error



Convergence speed

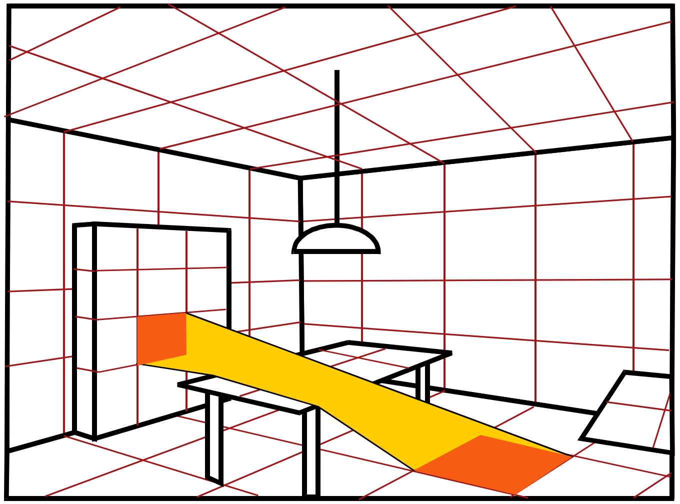


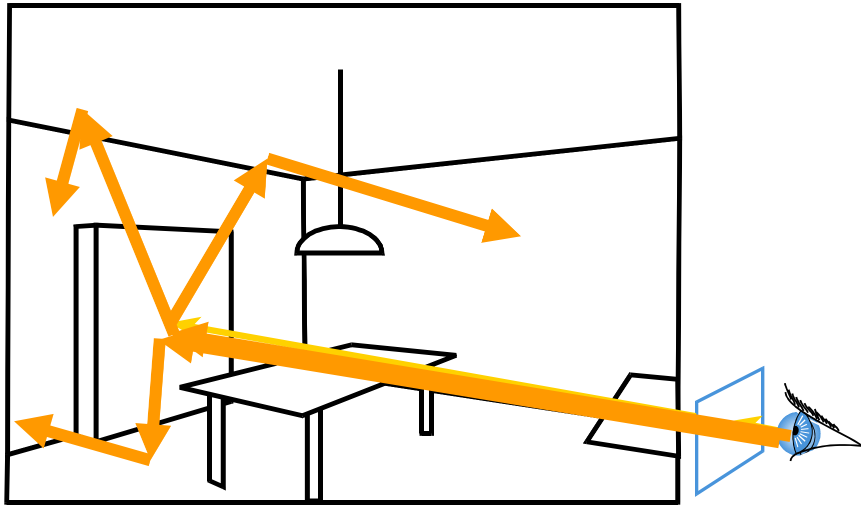
- Vintage path tracing by Kajiya (1986, introduction of the rendering equation)



Radiosity vs. Monte Carlo

- We have an integral equation on an infinite space
- Finite elements (Radiosity)
  - Project onto finite basis of functions
  - Linear system
  - View-independent (no angular information)
- Monte Carlo
  - Probabilistic sampling
  - View-dependent (but angular information)



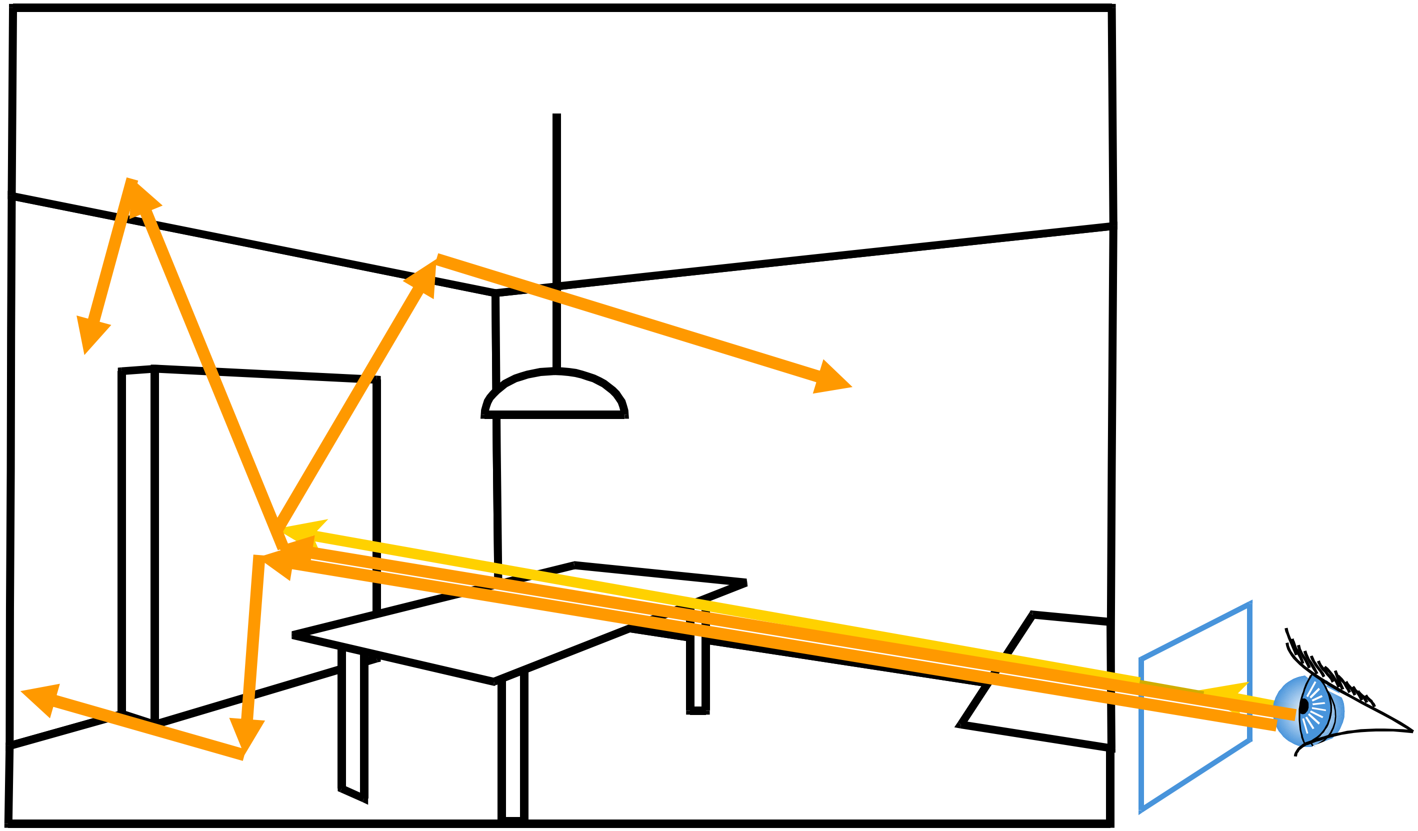


Today's lecture

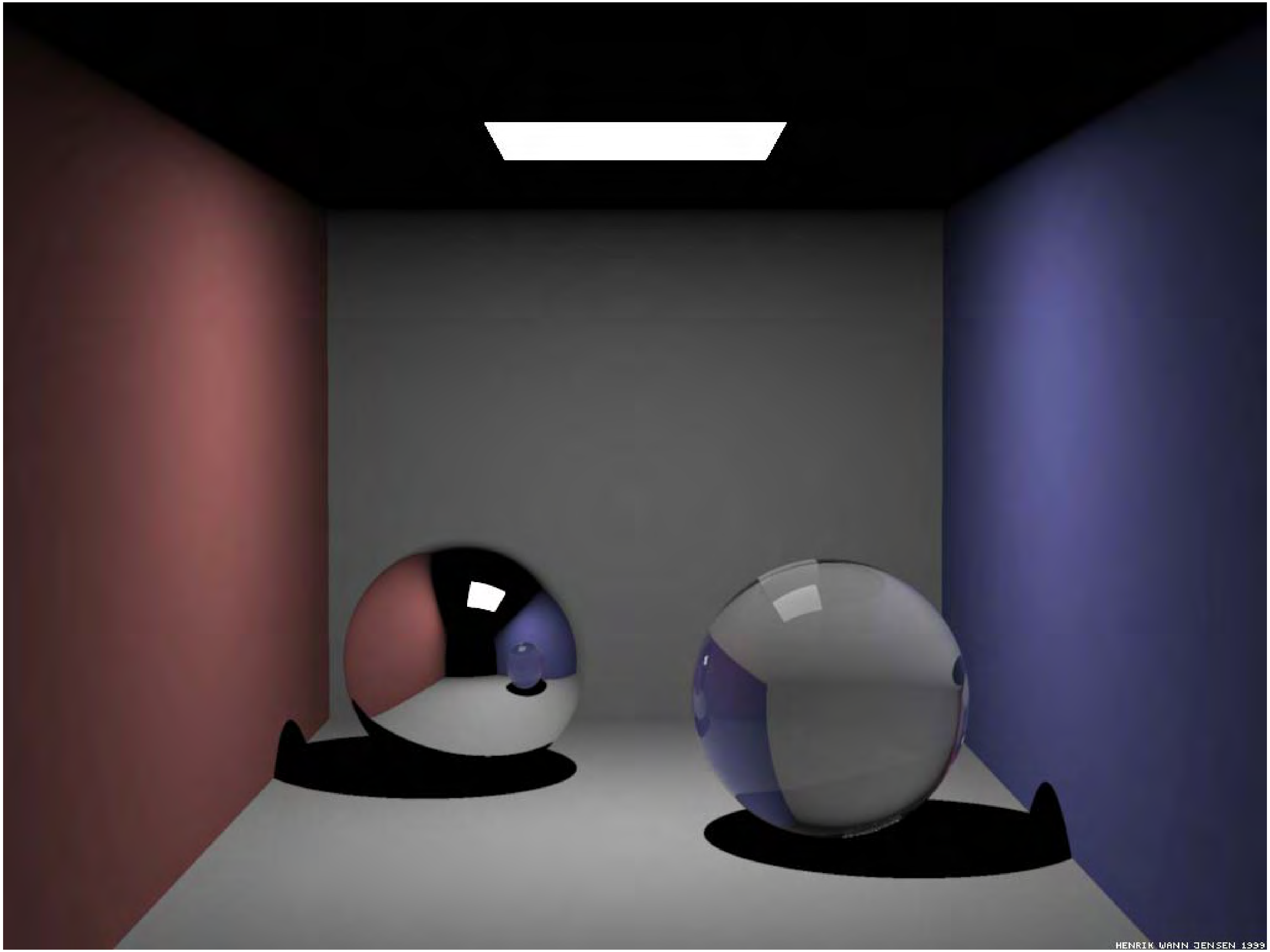
- Expected value and variance
- Analysis of Monte-Carlo integration
- Monte-Carlo in graphics
- Importance sampling
- Stratified sampling
- Global illumination
- Advanced Monte-Carlo rendering

Path Tracing is costly

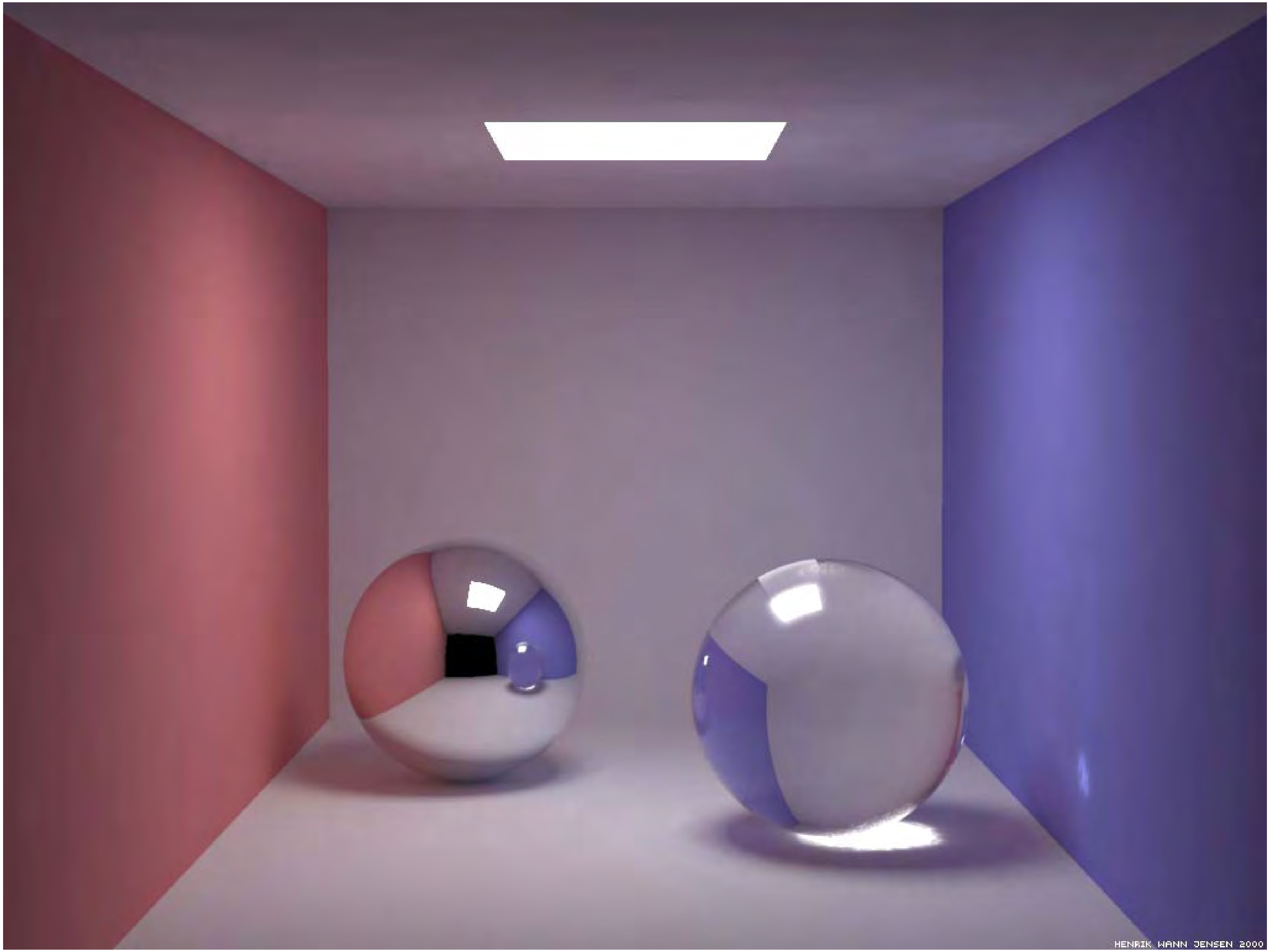
- Needs tons of rays per pixel



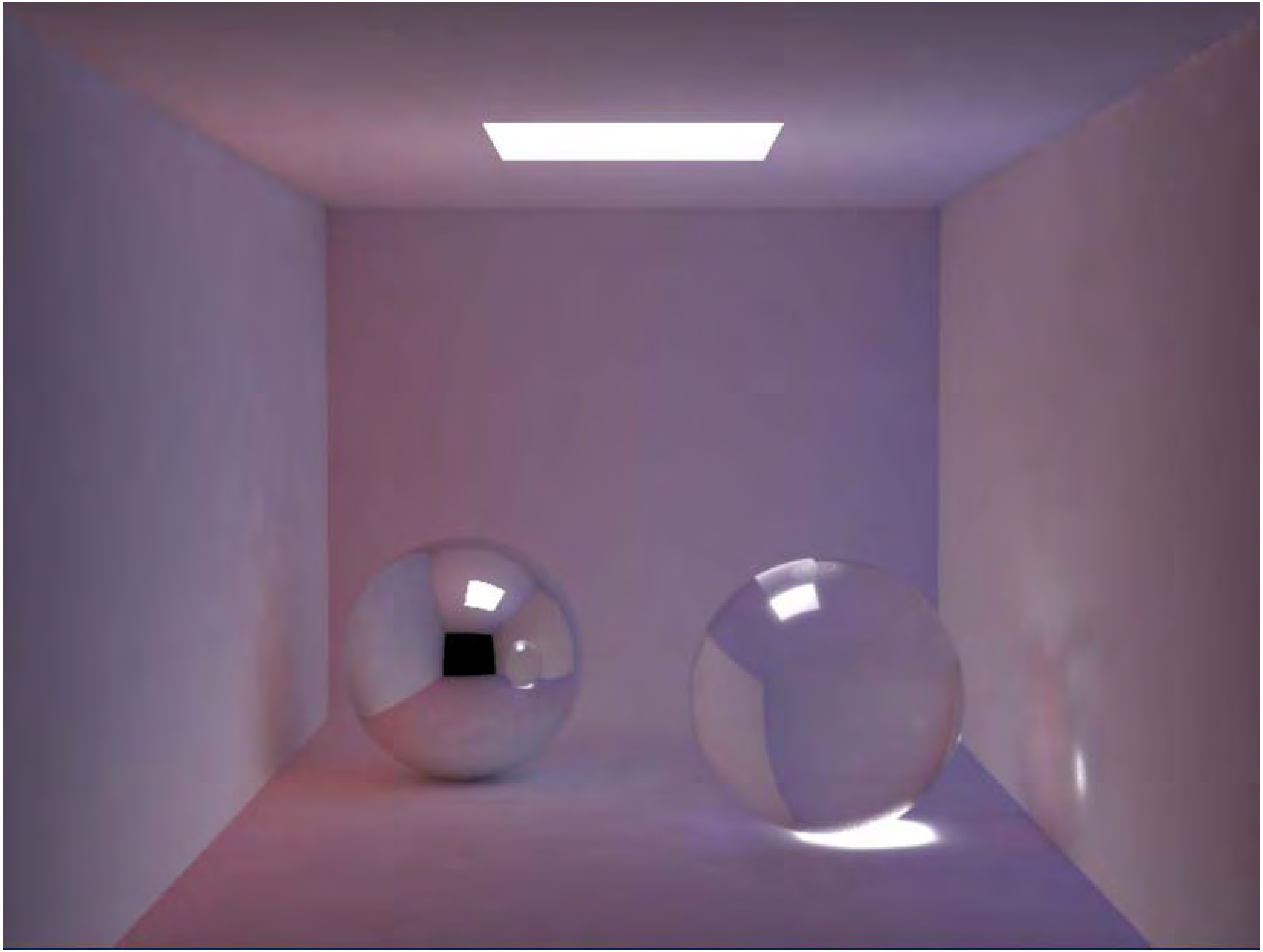
Direct illumination



Global Illumination

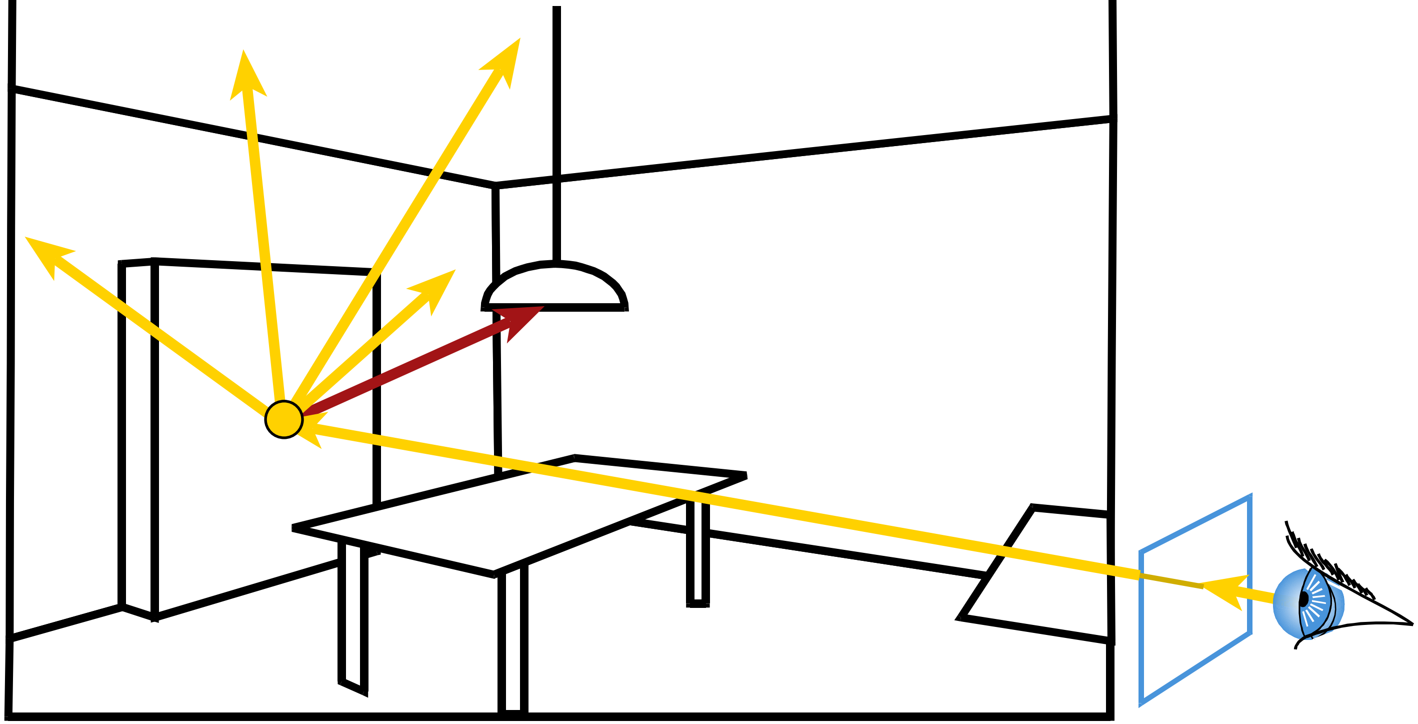


Indirect illumination: smooth



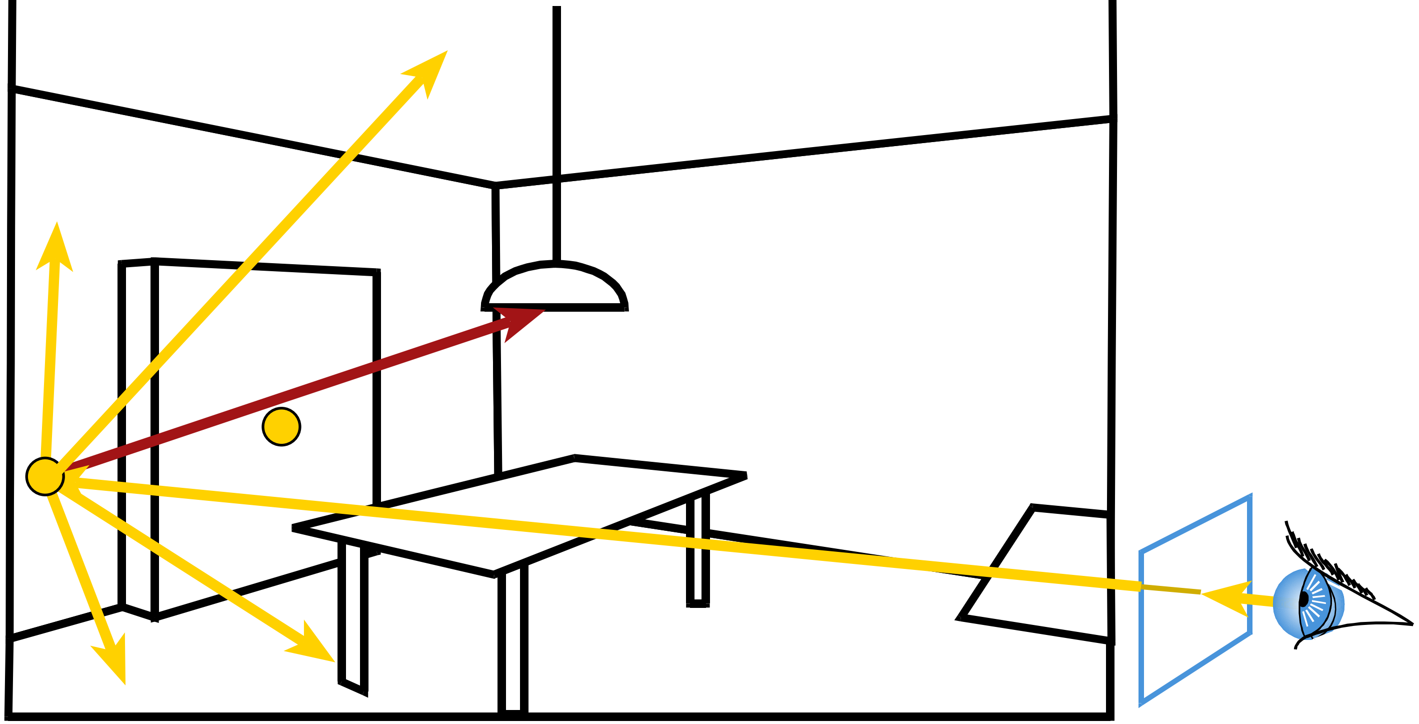
Irradiance cache

- The indirect illumination is smooth



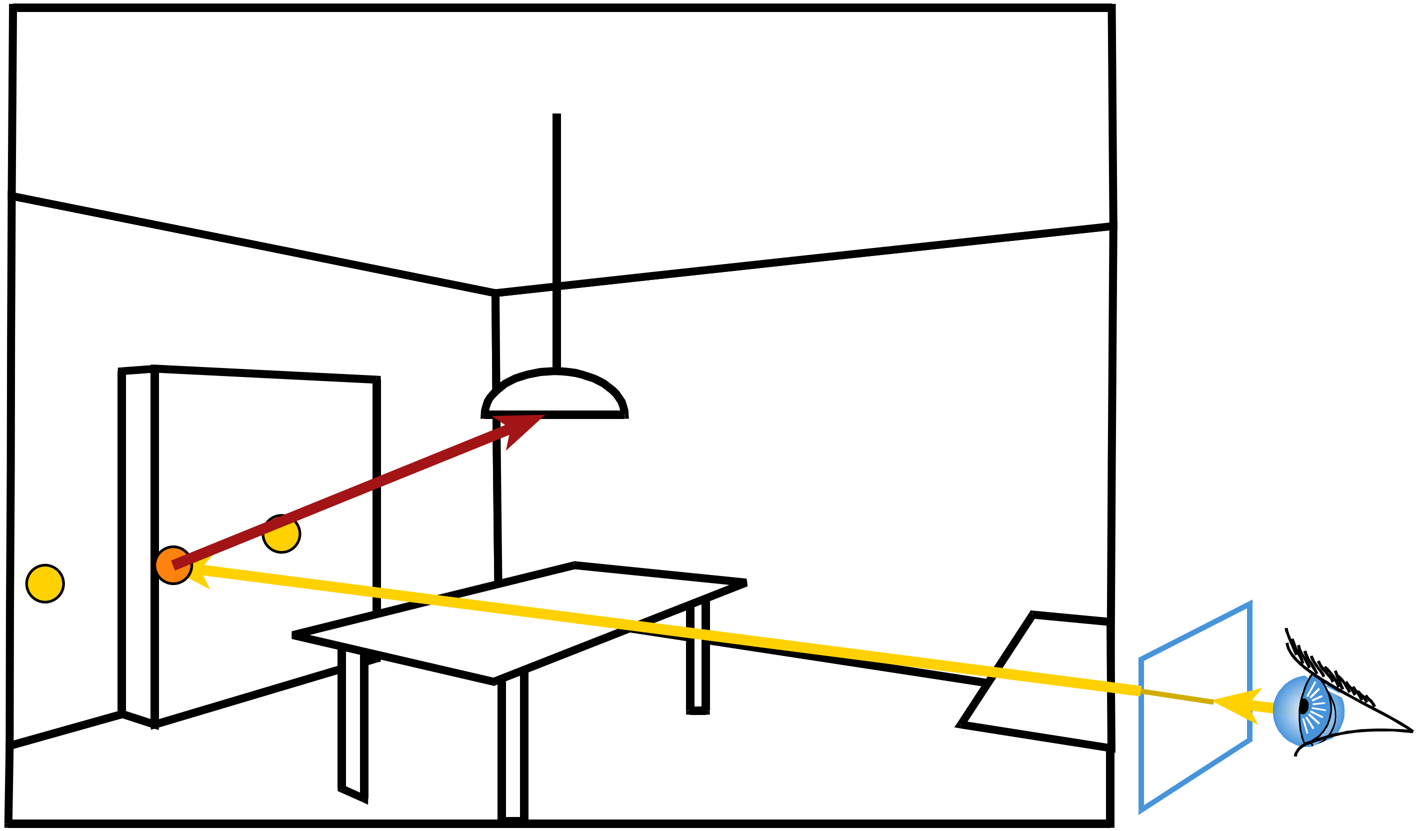
Irradiance cache

- The indirect illumination is smooth



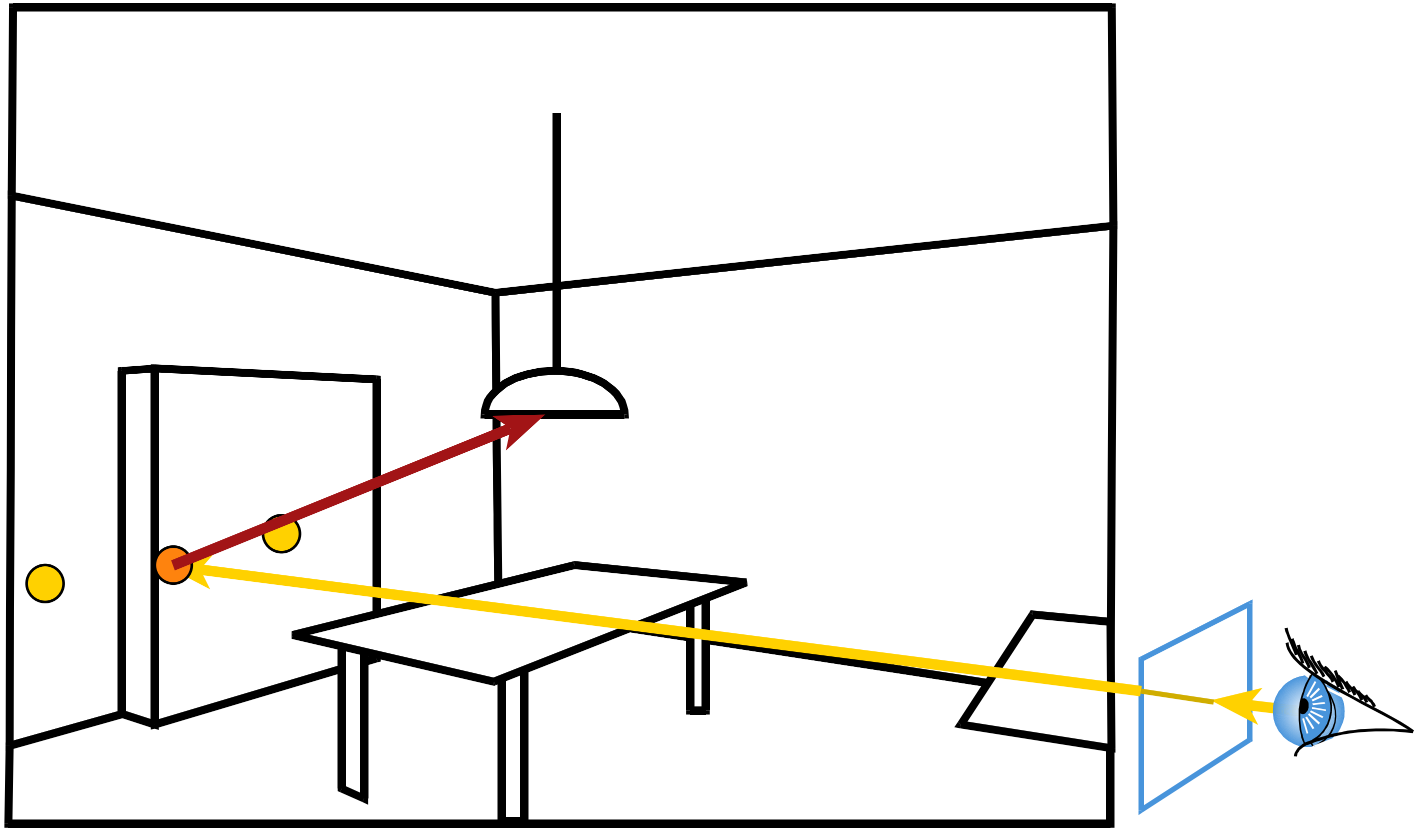
Irradiance cache

- The indirect illumination is smooth
- Interpolate nearby values



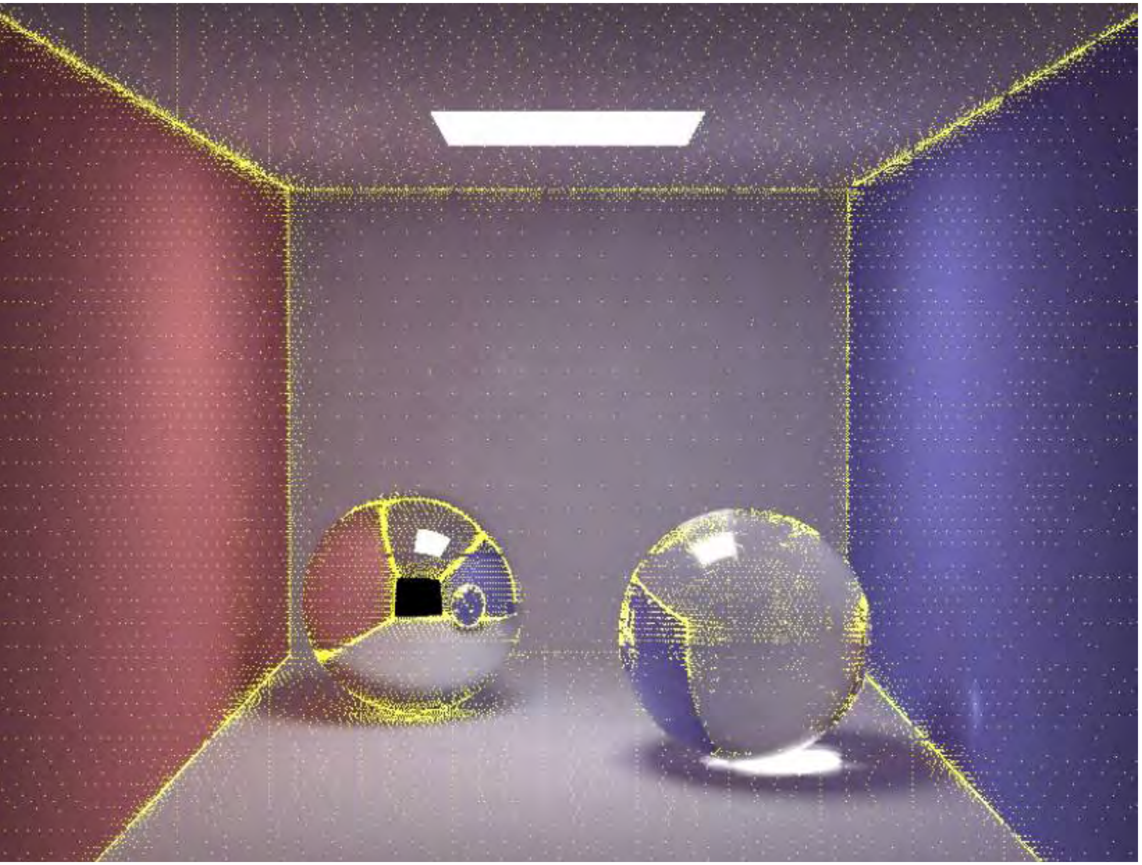
Irradiance cache

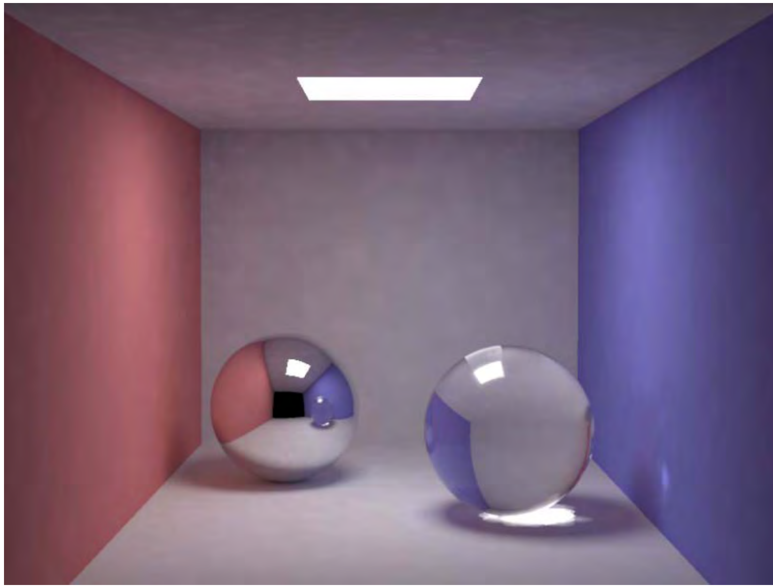
- Store the indirect illumination
- Interpolate existing cached values
- But do full calculation for direct lighting



Irradiance caching

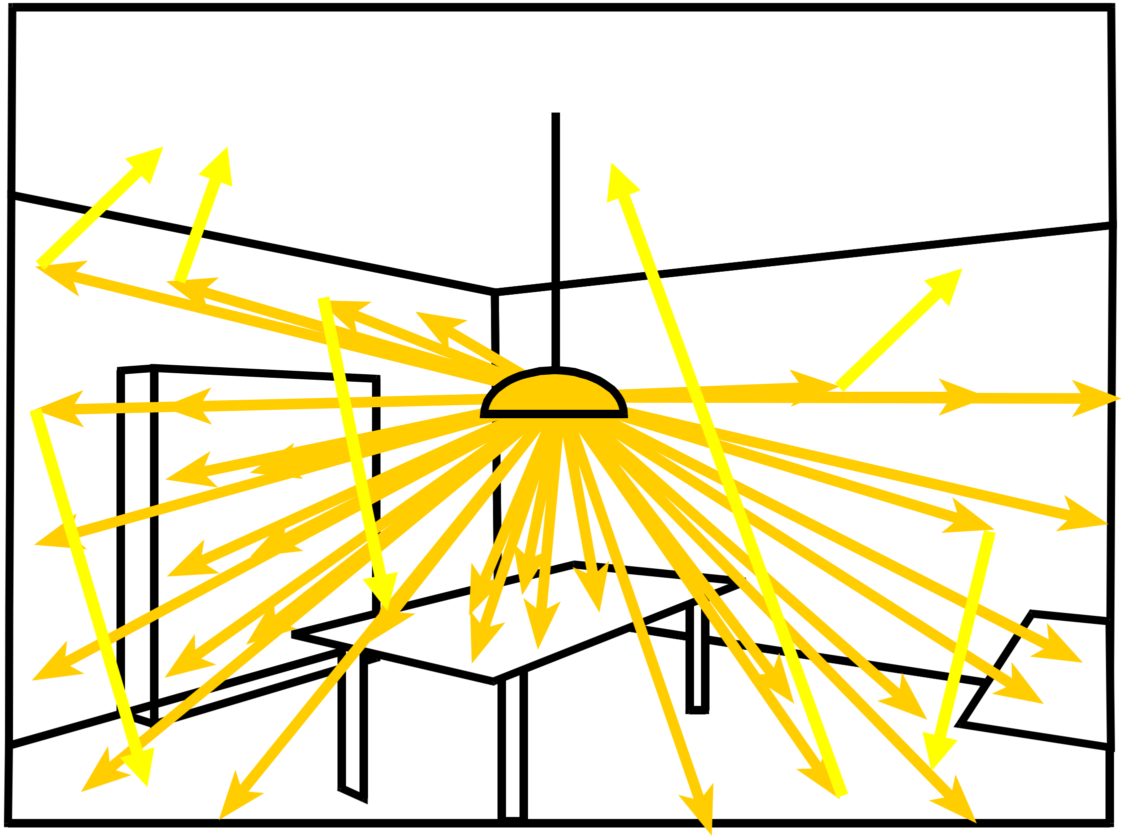
- Yellow dots:
  computation of indirect diffuse contribution





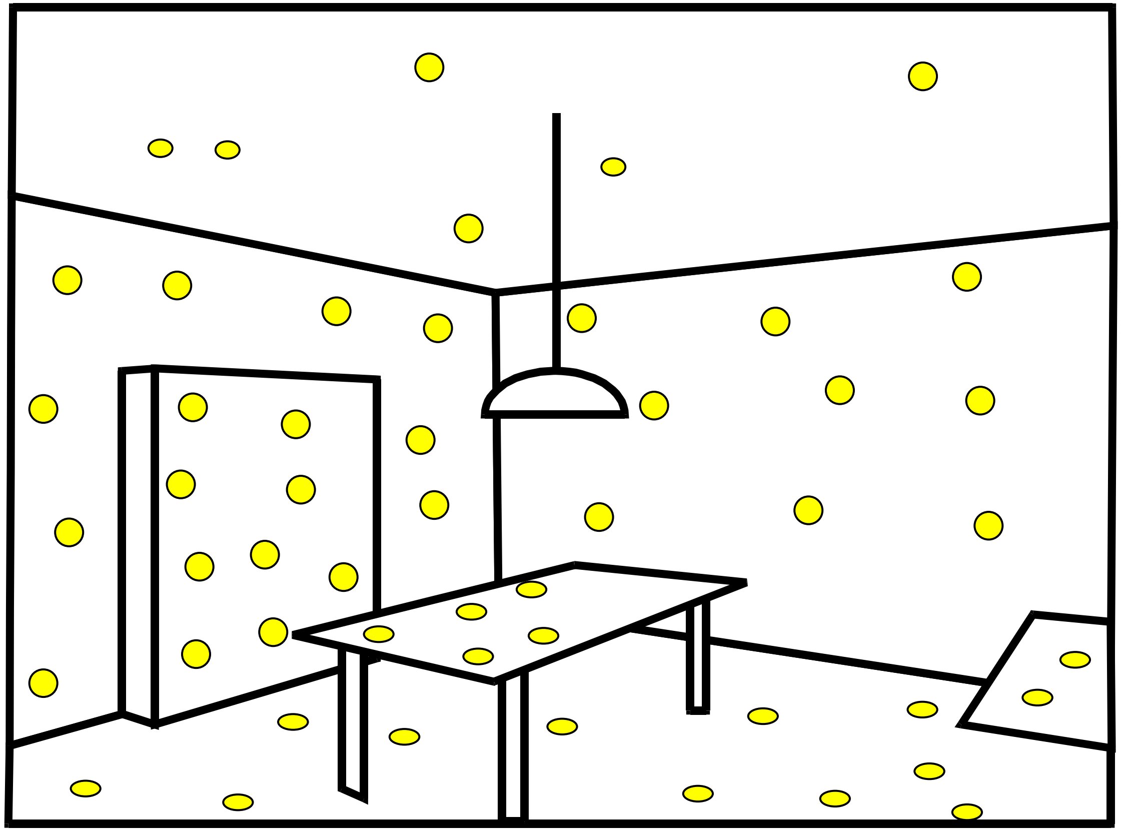
Photon mapping

- Preprocess: cast rays from light sources
- Store photons



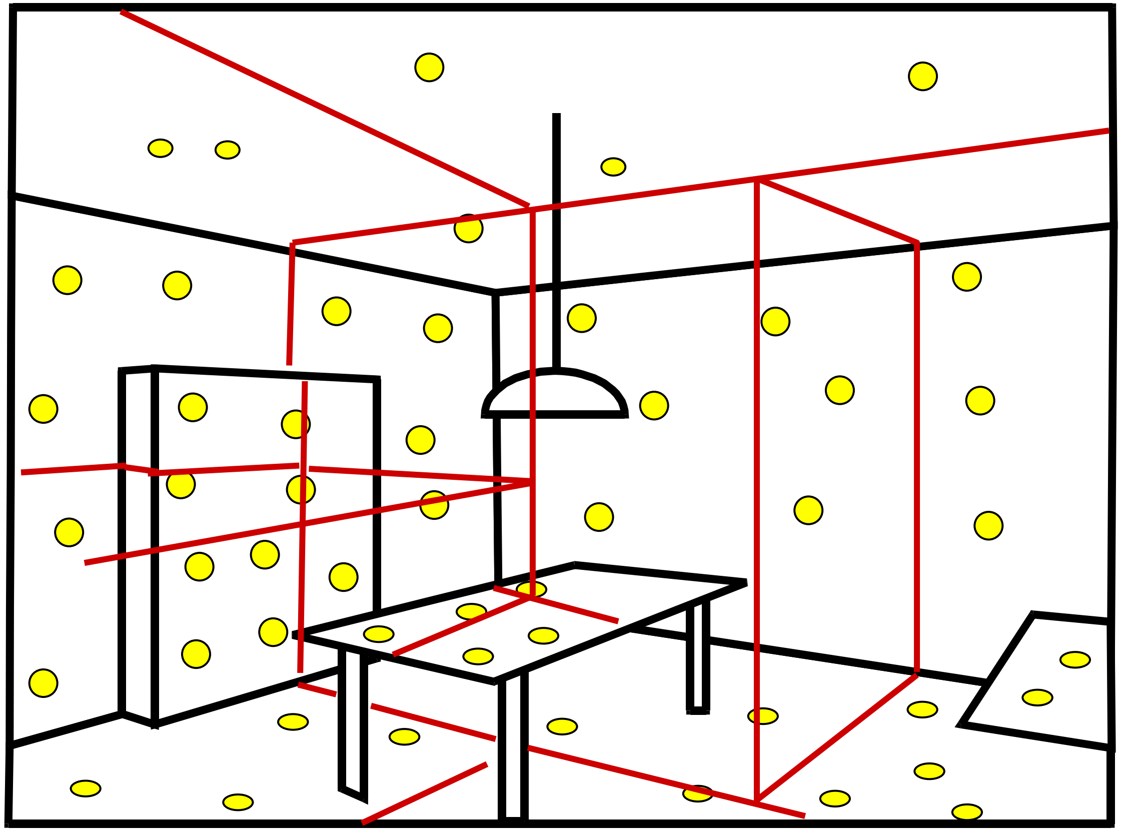
Photon mapping

- Preprocess: cast rays from light sources
- Store photons (position + light power + incoming direction)



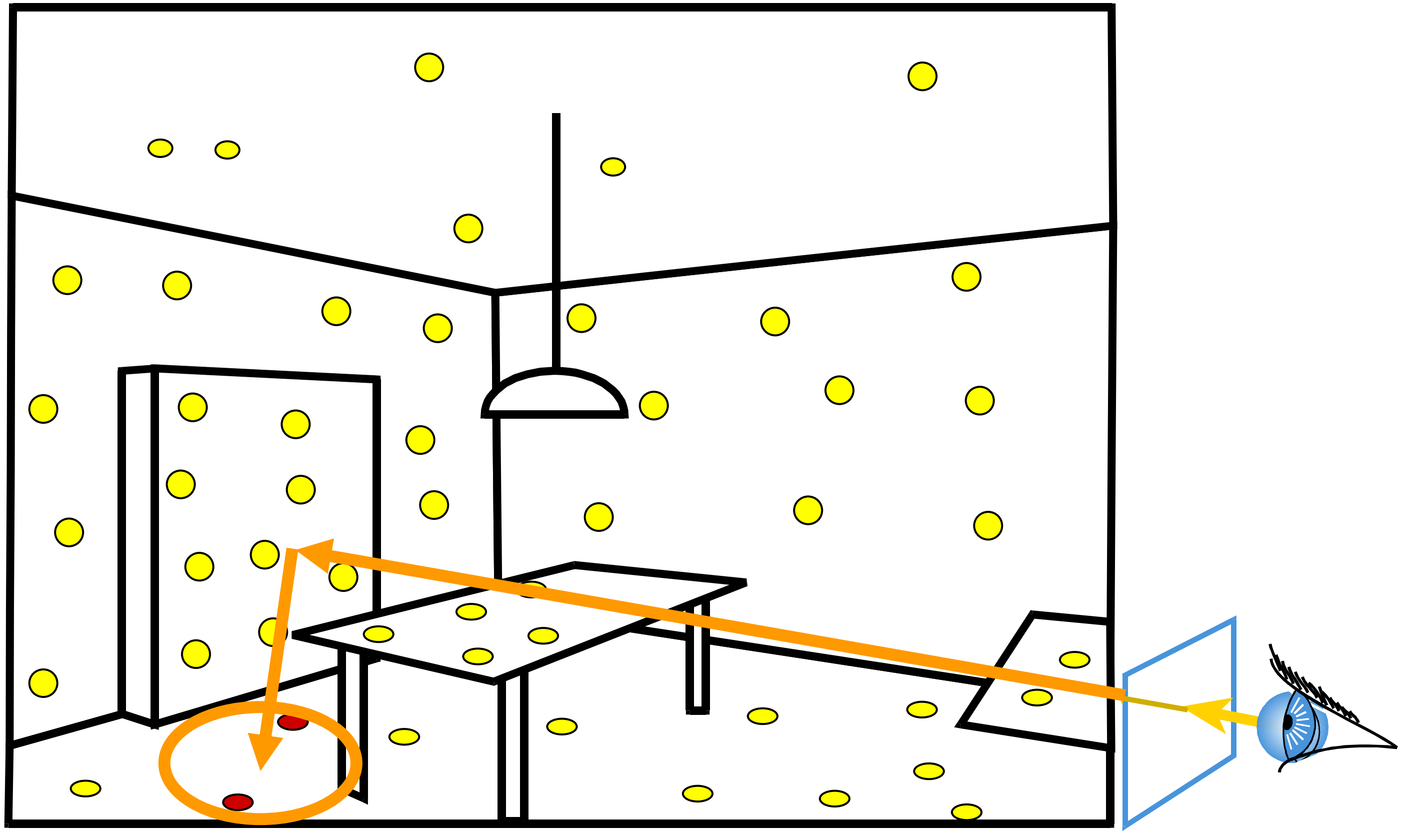
Photon map

- Efficiently store photons for fast access
- Use hierarchical spatial structure (kd-tree)

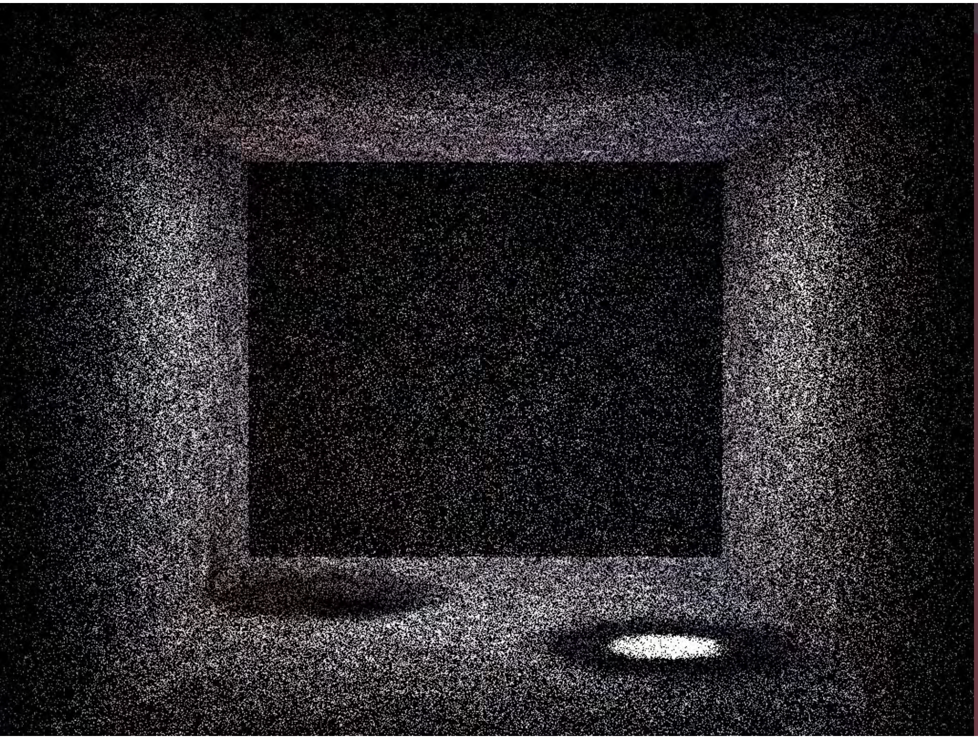


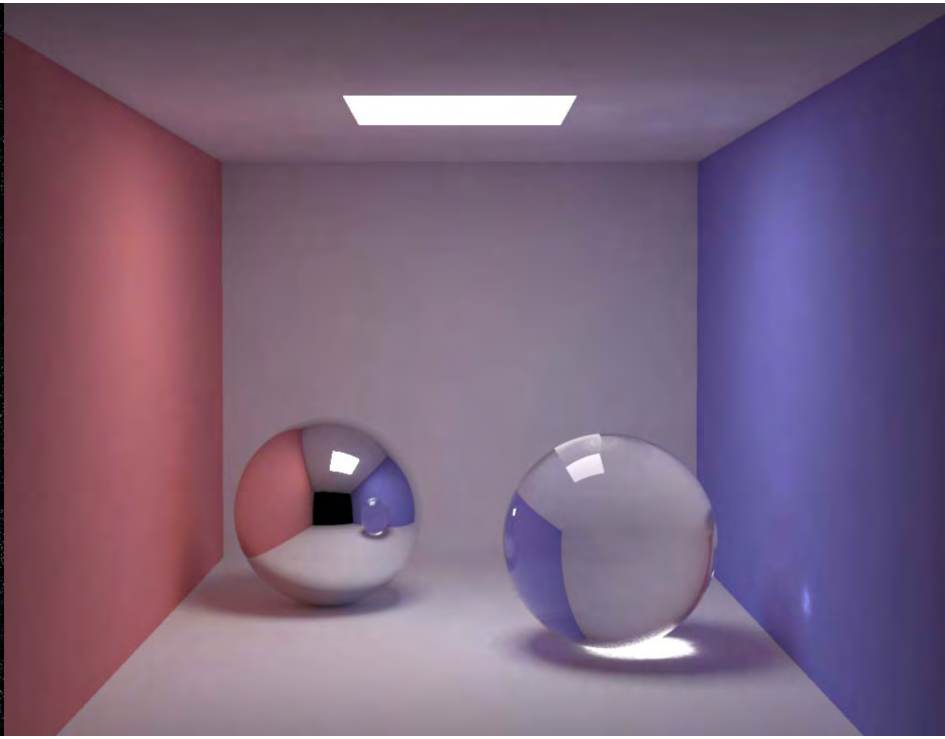
Photon mapping - rendering

- Cast primary rays
- For secondary rays
  - reconstruct irradiance using adjacent stored photon
  - Take the k closest photons
- Combine with irradiance caching and a number of other techniques



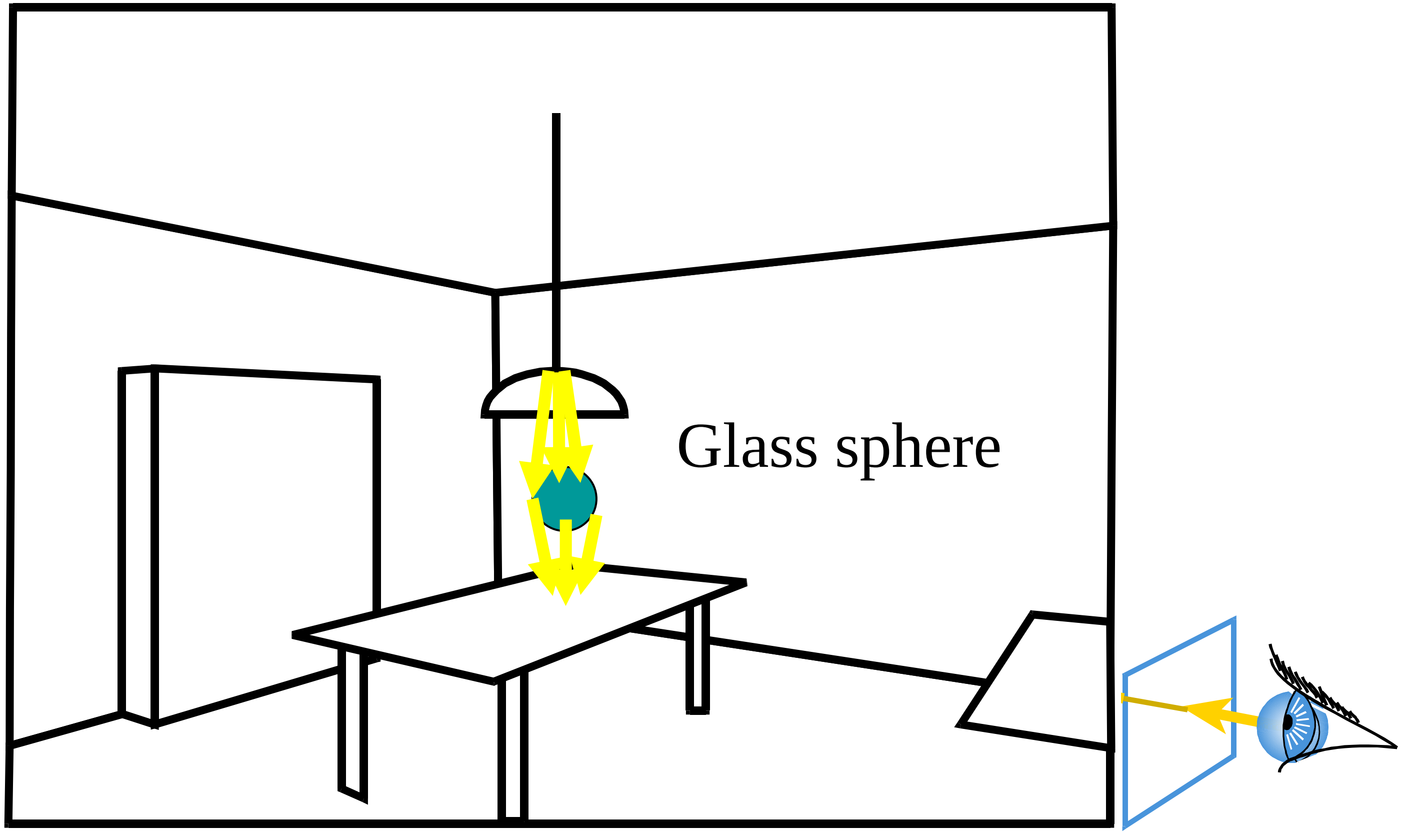
Photon map results



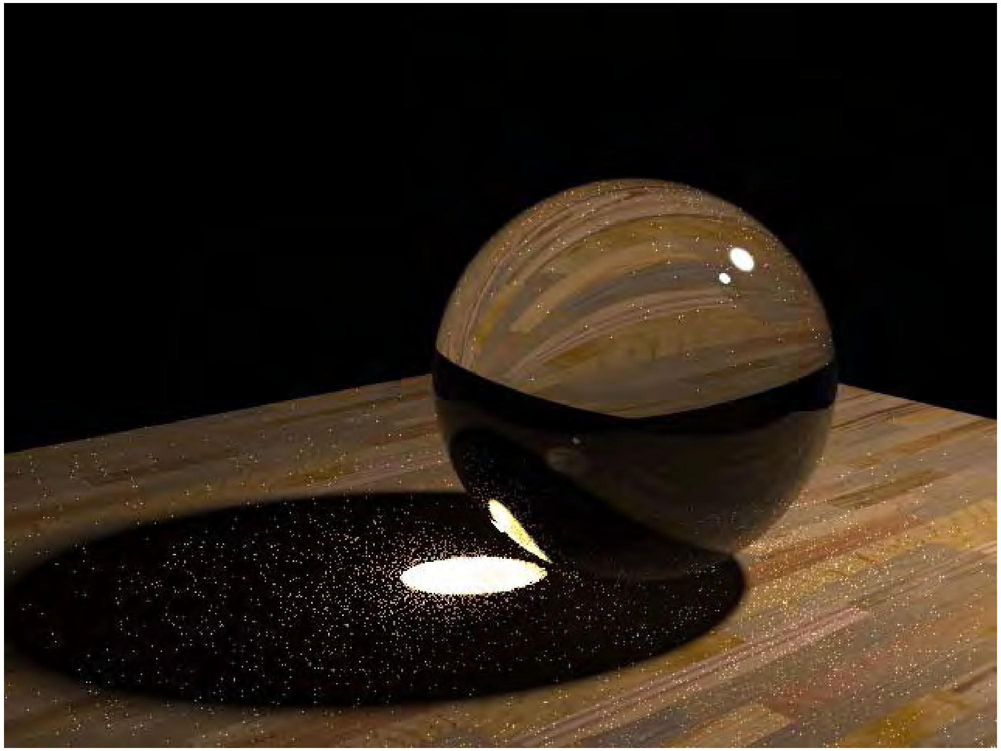


Photon mapping - caustics

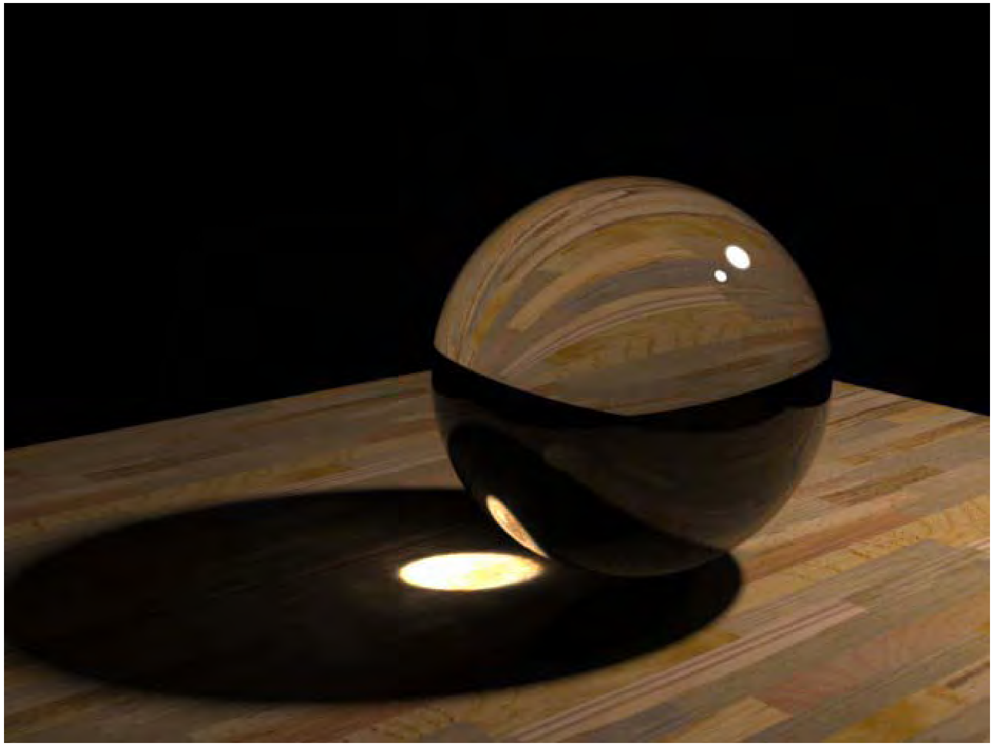
- Special photon map for specular reflection and refraction



- 1000 paths/pixel

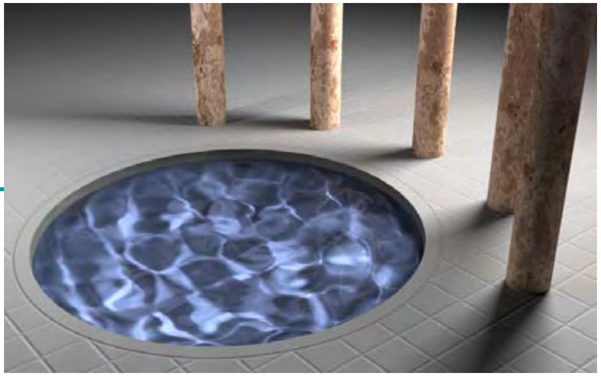


- Photon mapping



References

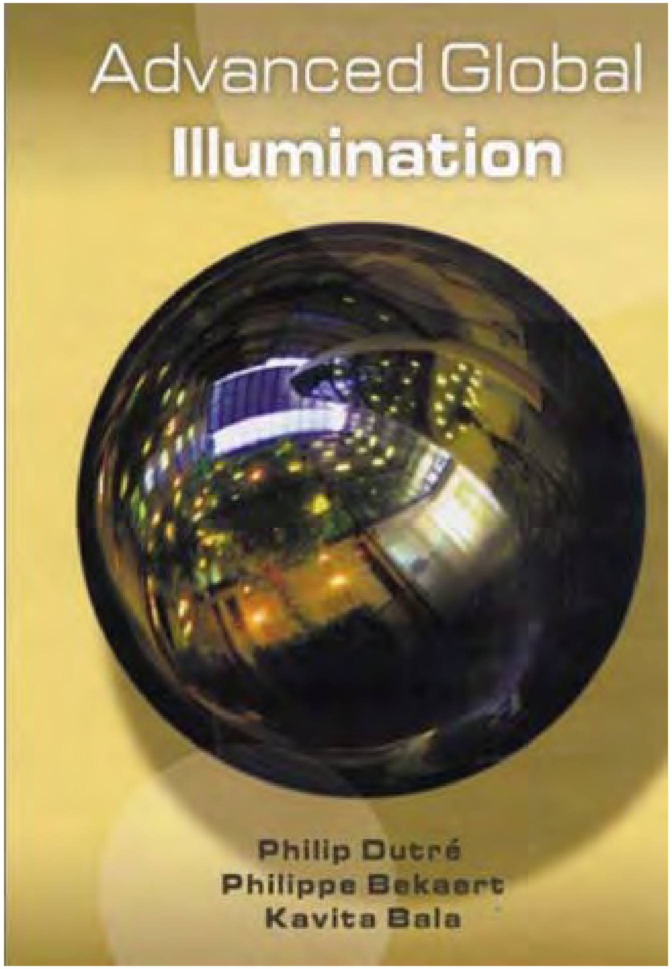
- Eric Veach's PhD dissertation
  http://graphics.stanford.edu/papers/veach_thesi

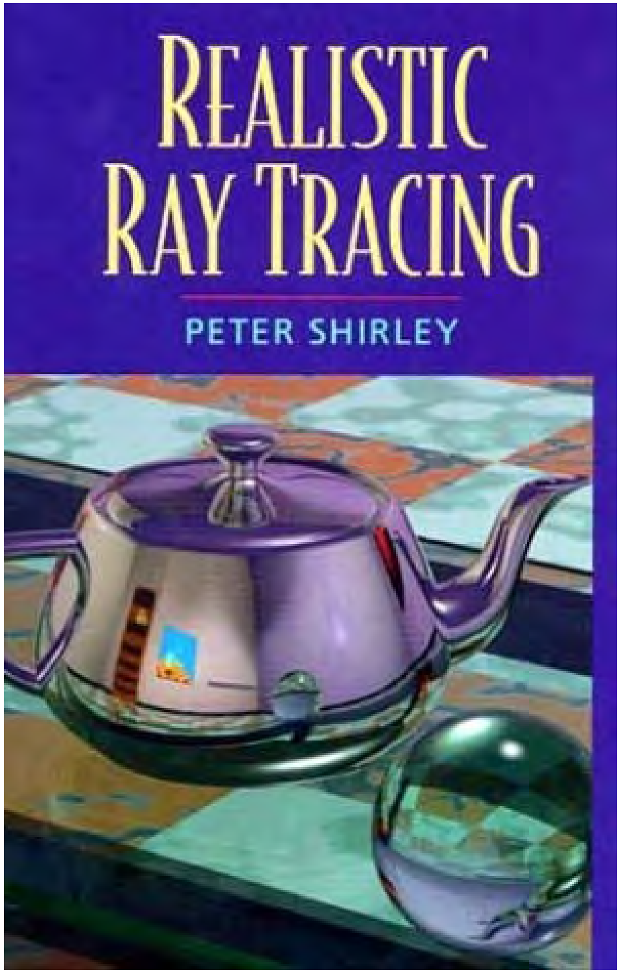


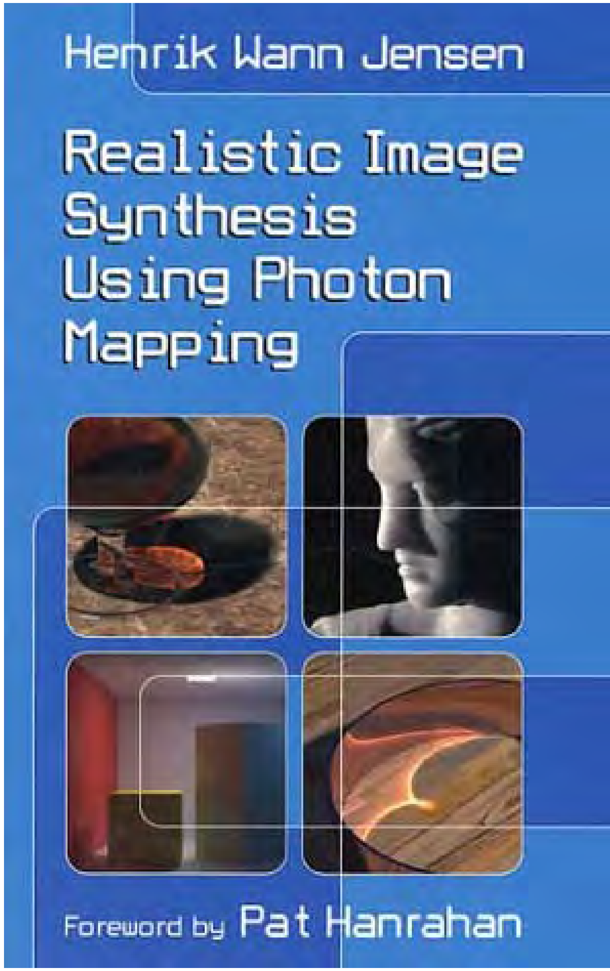
- Physically Based Rendering
  by Matt Pharr, Greg Humphreys

MATT PHARR ◦ GREG HUMPHREYS
PHYSICALLY BASED
RENDERING
FROM THEORY TO
IMPLEMENTATION
SERIES IN INTERACTIVE 3D TECHNOLOGY

References


Advanced Global Illumination
Philip Dutré
Philippe Bekaert
Kavita Bala


REALISTIC RAY TRACING
PETER SHIRLEY

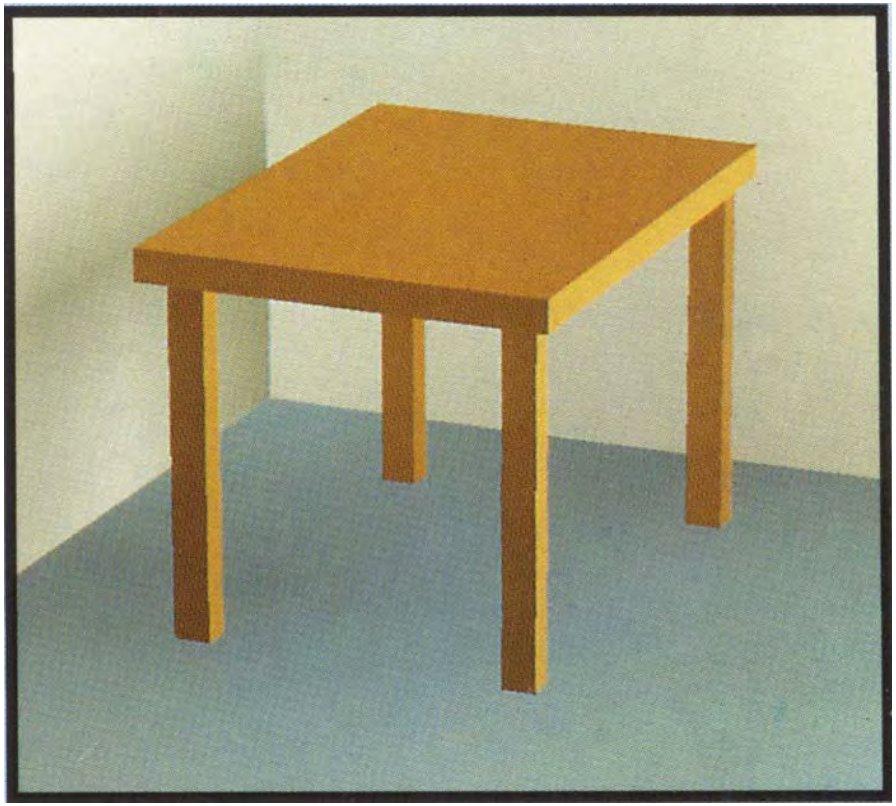

Henrik Wann Jensen
Realistic Image Synthesis Using Photon Mapping
Foreword by Pat Hanrahan

Advanced Topics

- Advanced Radiosity
  - Adaptive Subdivision
  - Discontinuity Meshing
  - Hierarchical Radiosity
  - Other Basis Functions

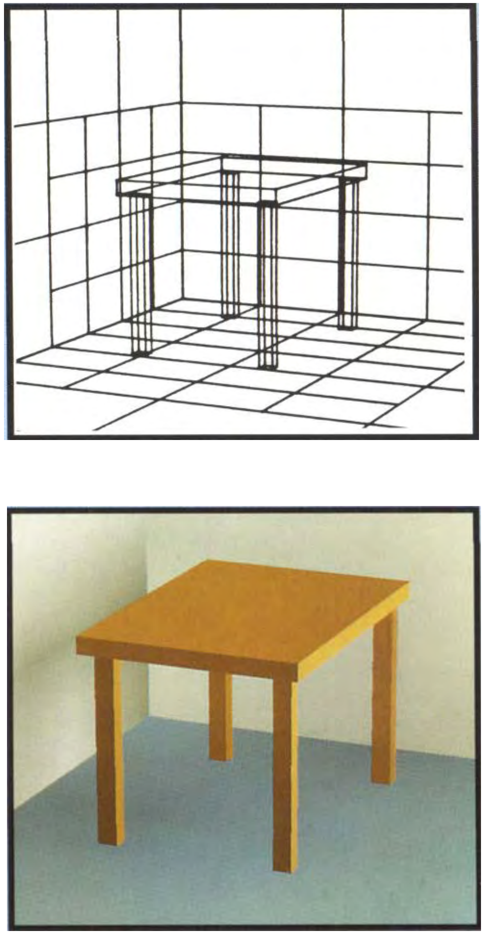
Increasing the Accuracy of the Solution

What's wrong with this picture?

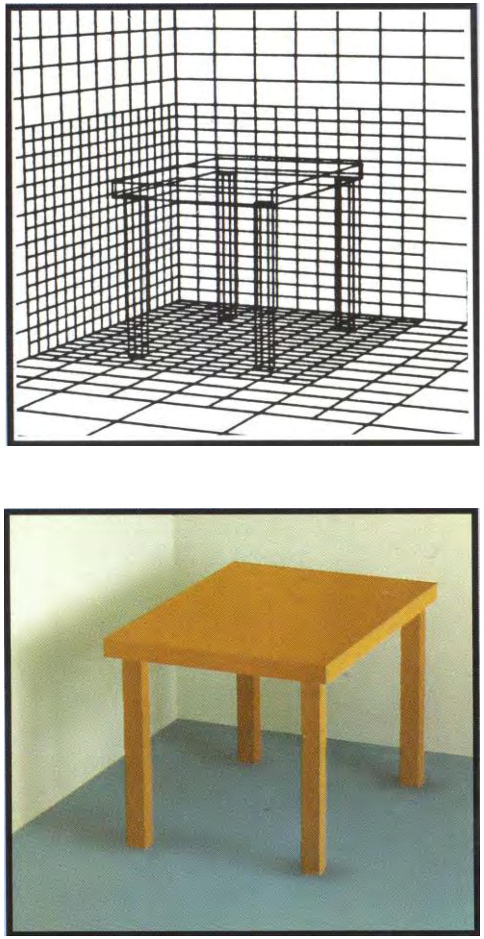


- The quality of the image is a function of the size of the patches
- The patches should be *adaptively subdivided* near shadow boundaries, and other areas with a high radiosity gradient
- Compute a solution on a uniform initial mesh, then refine the mesh in areas that exceed some error tolerance

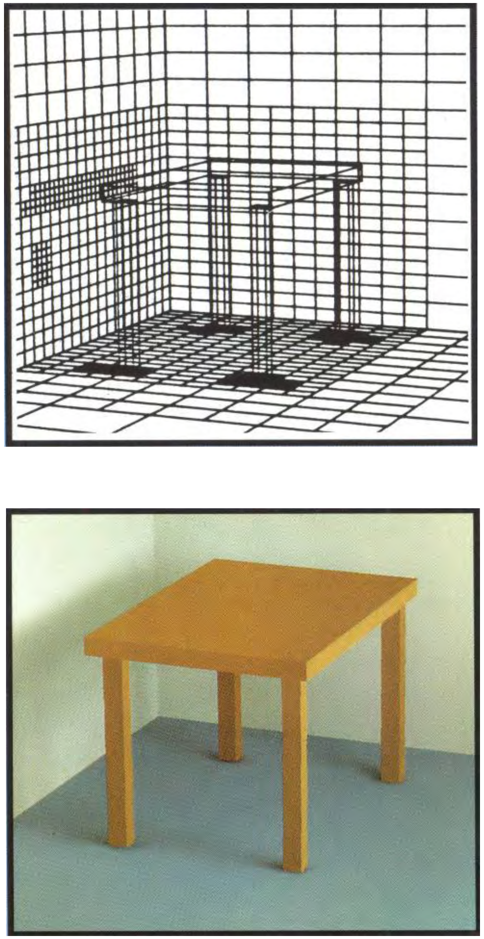
Adaptive Subdivision of Patches



Coarse patch solution
(145 patches)



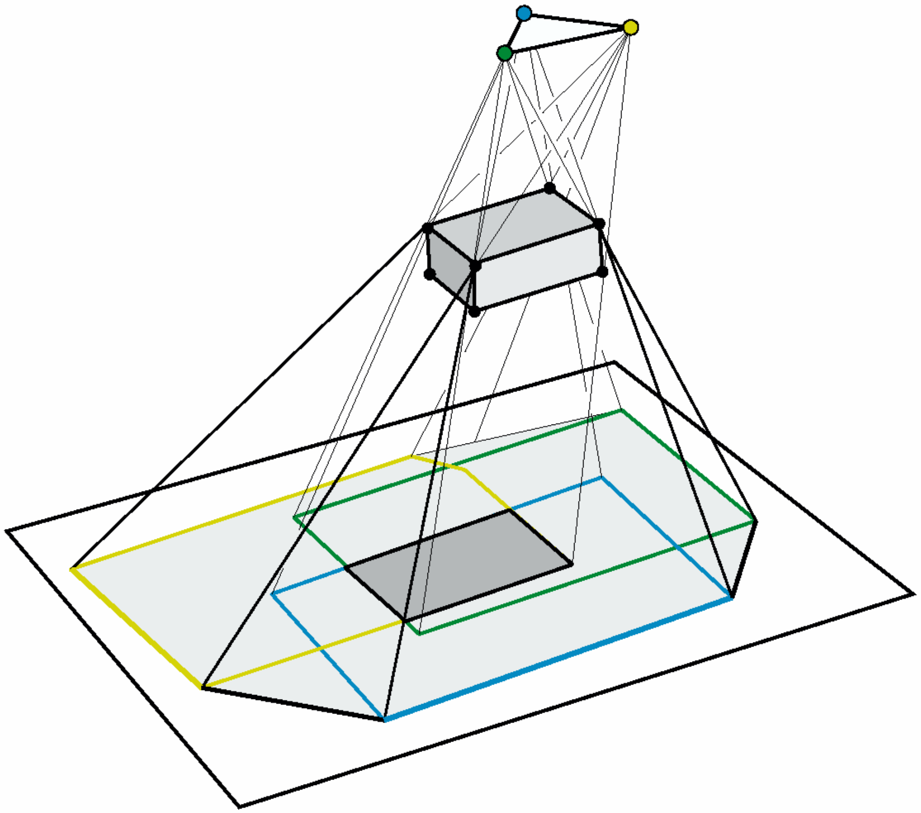
Improved solution
(1021 subpatches)



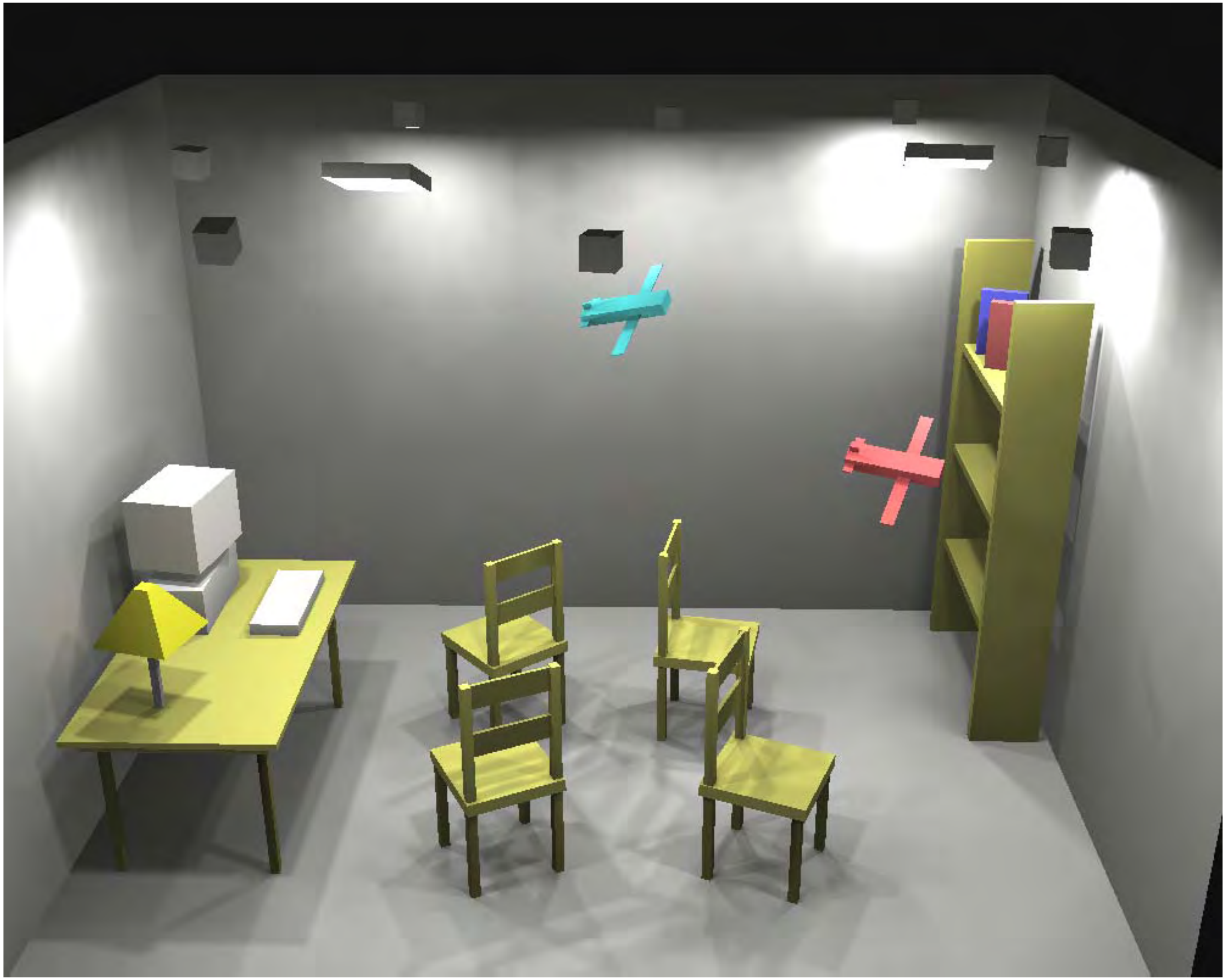
Adaptive subdivision
(1306 subpatches)

Discontinuity Meshing

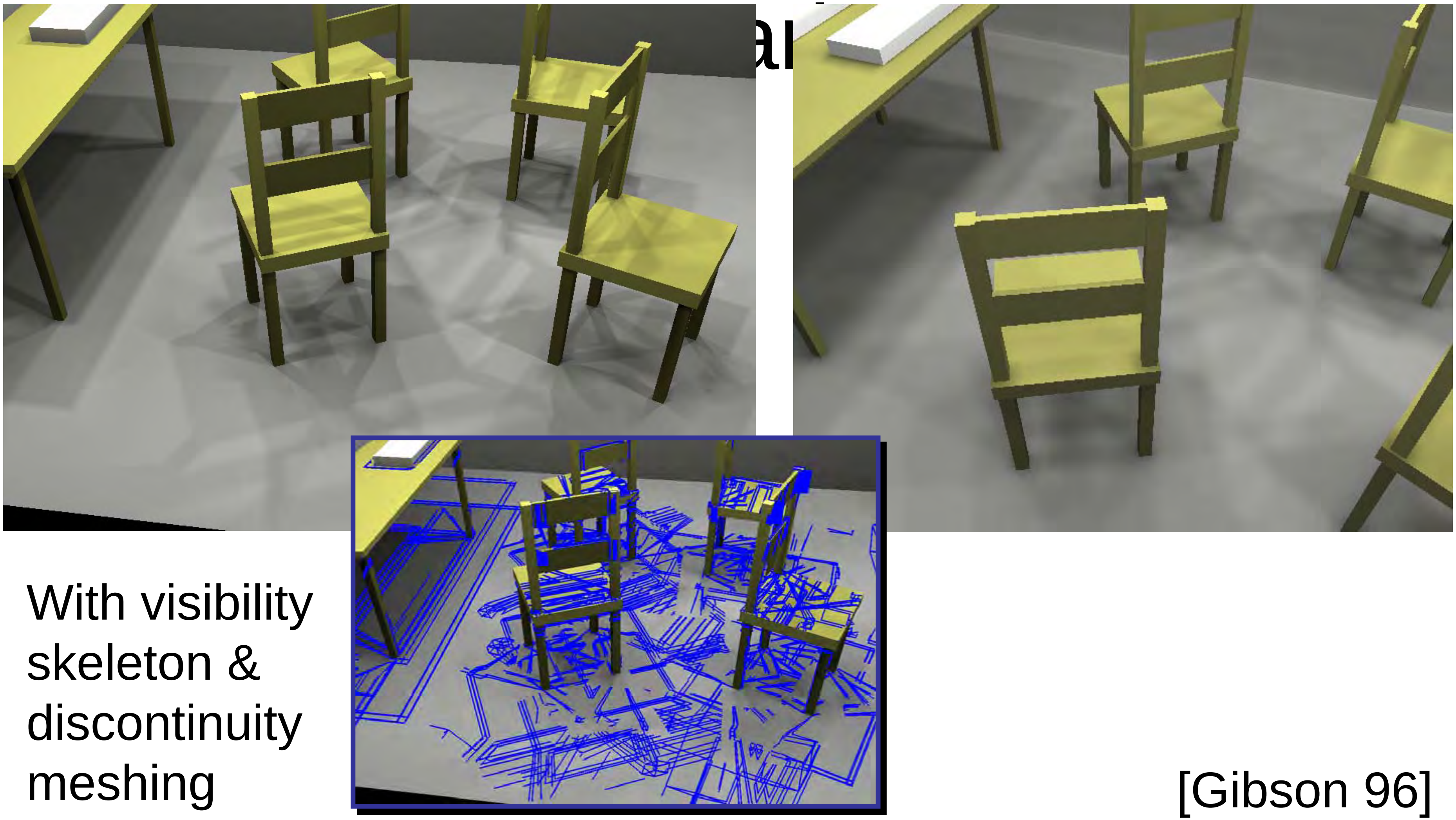
- Limits of umbra and penumbra
  - Captures nice shadow boundaries
  - Complex geometric computation
  - The mesh is getting complex



Discontinuity Meshing



Discontinuity Meshing



With visibility skeleton & discontinuity meshing

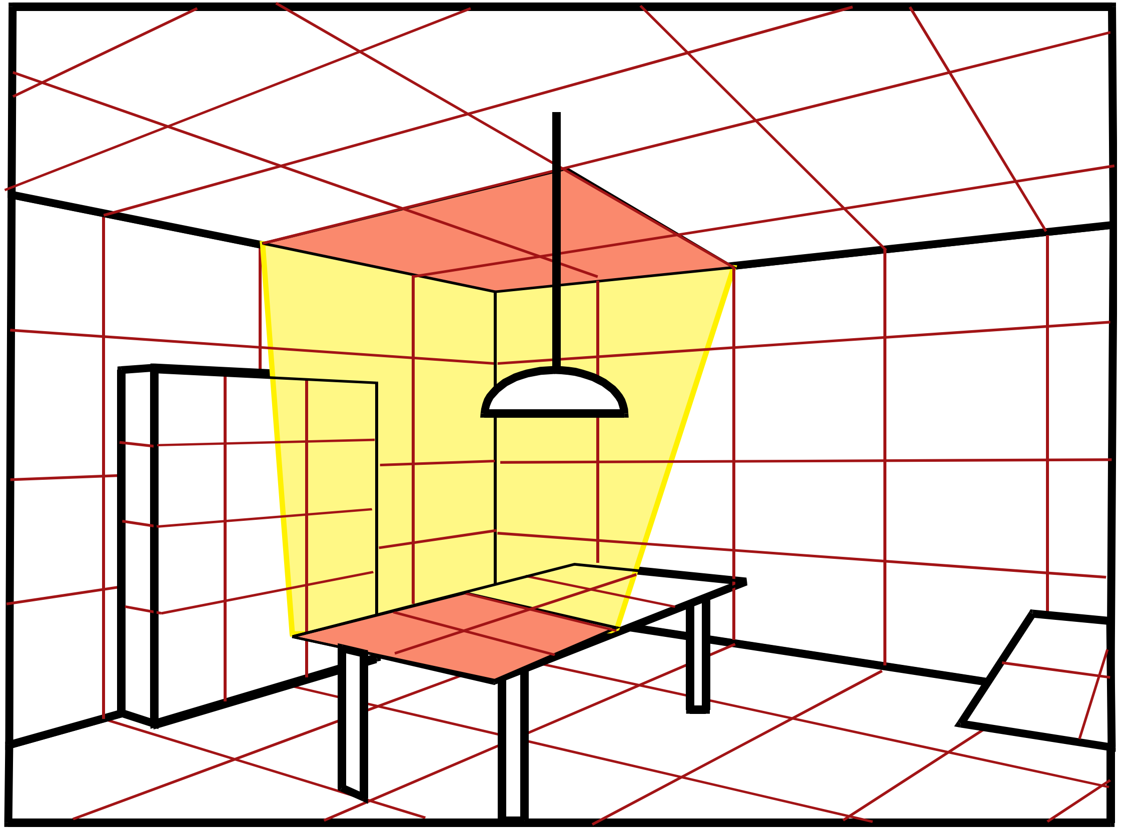
10 minutes 23 seconds

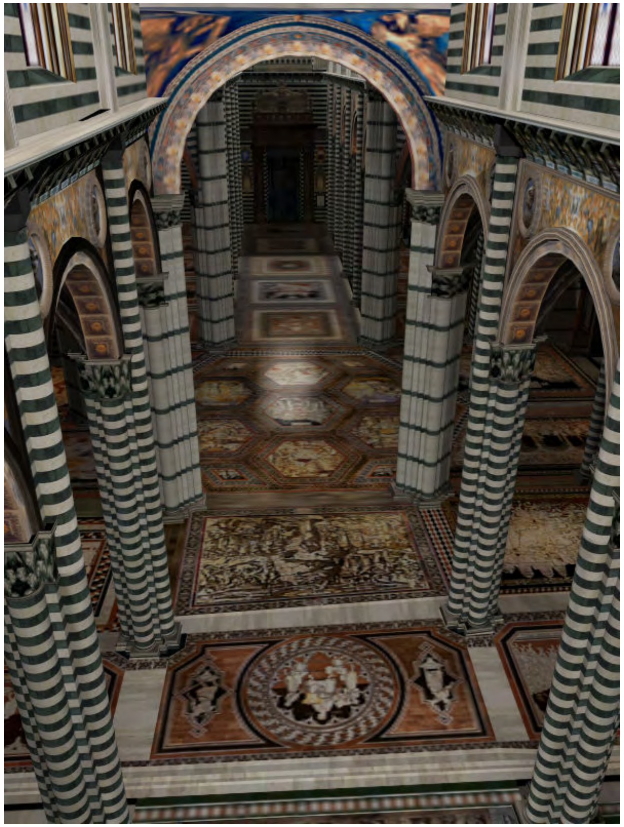
[Gibson 96]

1 hour 57 minutes

Hierarchical Approach

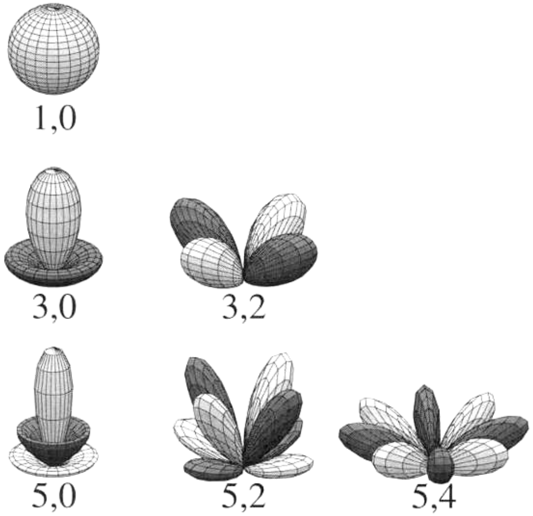
- Group elements when the light exchange is not important
  - Breaks the quadratic complexity
  - Control non trivial, memory cost

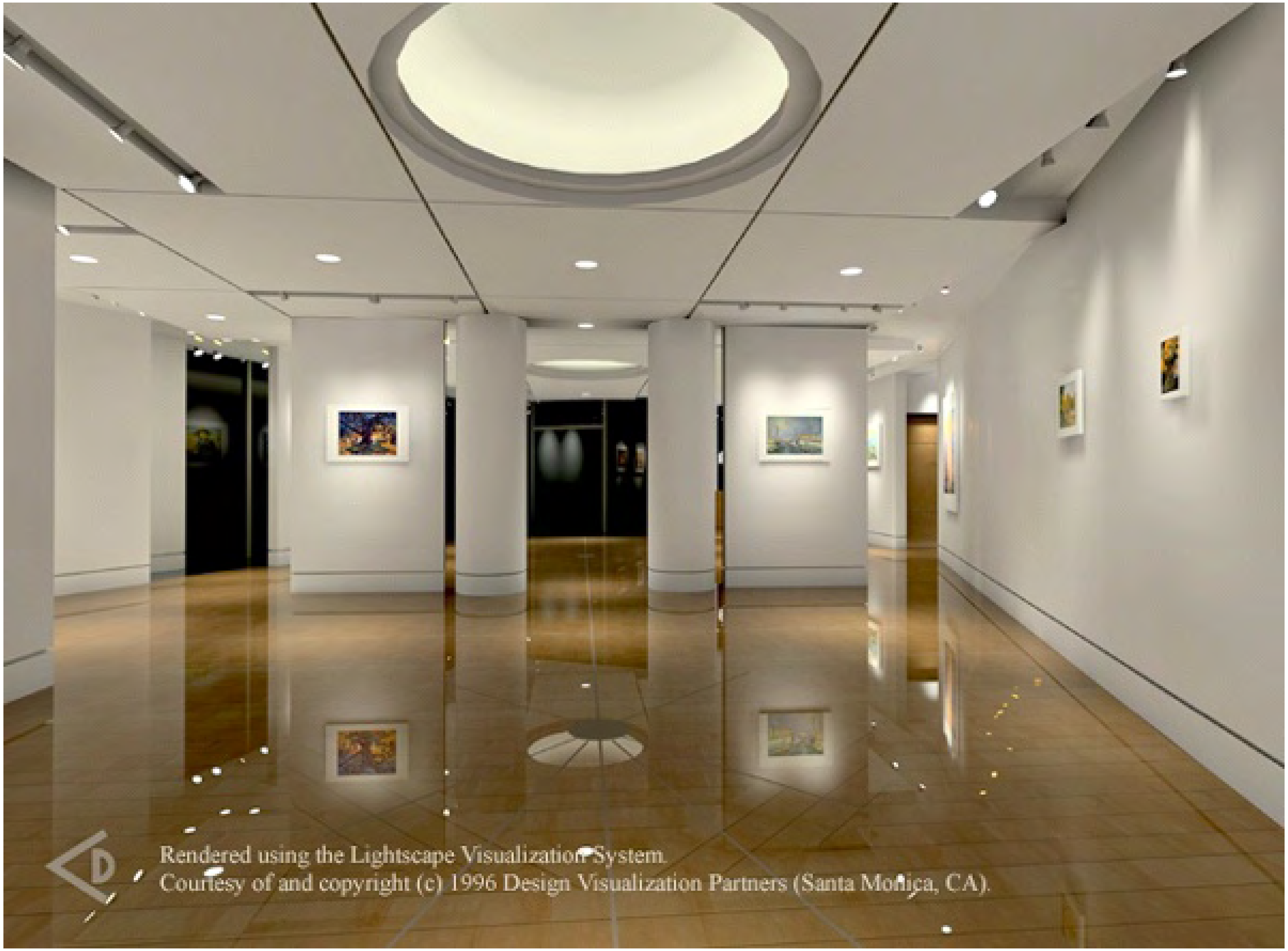




Other Basis Functions

- Higher order (non constant basis)
  - Better representation of smooth variations
  - Problem: radiosity is discontinuous (shadow boundary)
- Directional basis
  - For non-diffuse finite elements
  - E.g. spherical harmonics




Rendered using the Lightscape Visualization System.
Courtesy of and copyright (c) 1996 Design Visualization Partners (Santa Monica, CA).

Lightscape **`http://www.lightscape.com`**

Radiosity today

- Used in architectural simulation (Lightscape software)
- Used for game lighting preprocessing (light maps)

- Not as hot a research topic
  - Monte-Carlo Ray-tracing is hotter (more general)
  - But “pre-computed radiance transfer” is very close: idea of projecting onto simpler basis functions (used e.g. in Max Payne 2)

Practical problems with radiosity

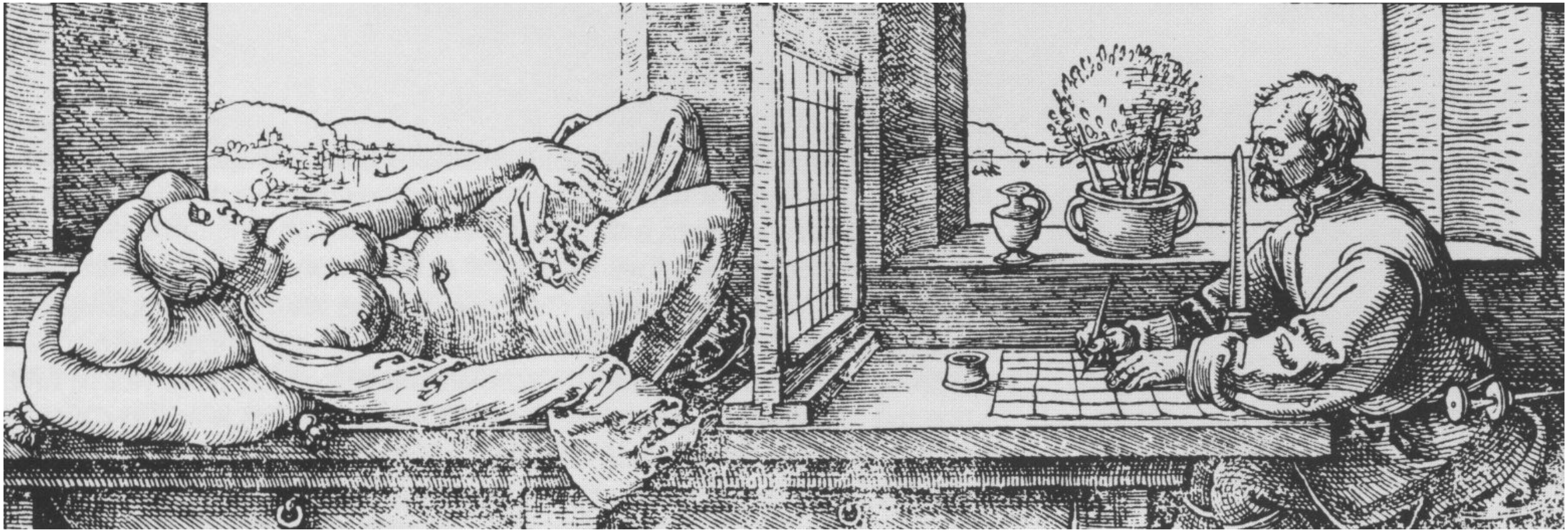
- Meshing (memory, robustness)
- Form factors (computation)
- Diffuse limitation (extension to specular takes too much memory)

- Fast extensions (hierarchical) can be hard to control

Fin

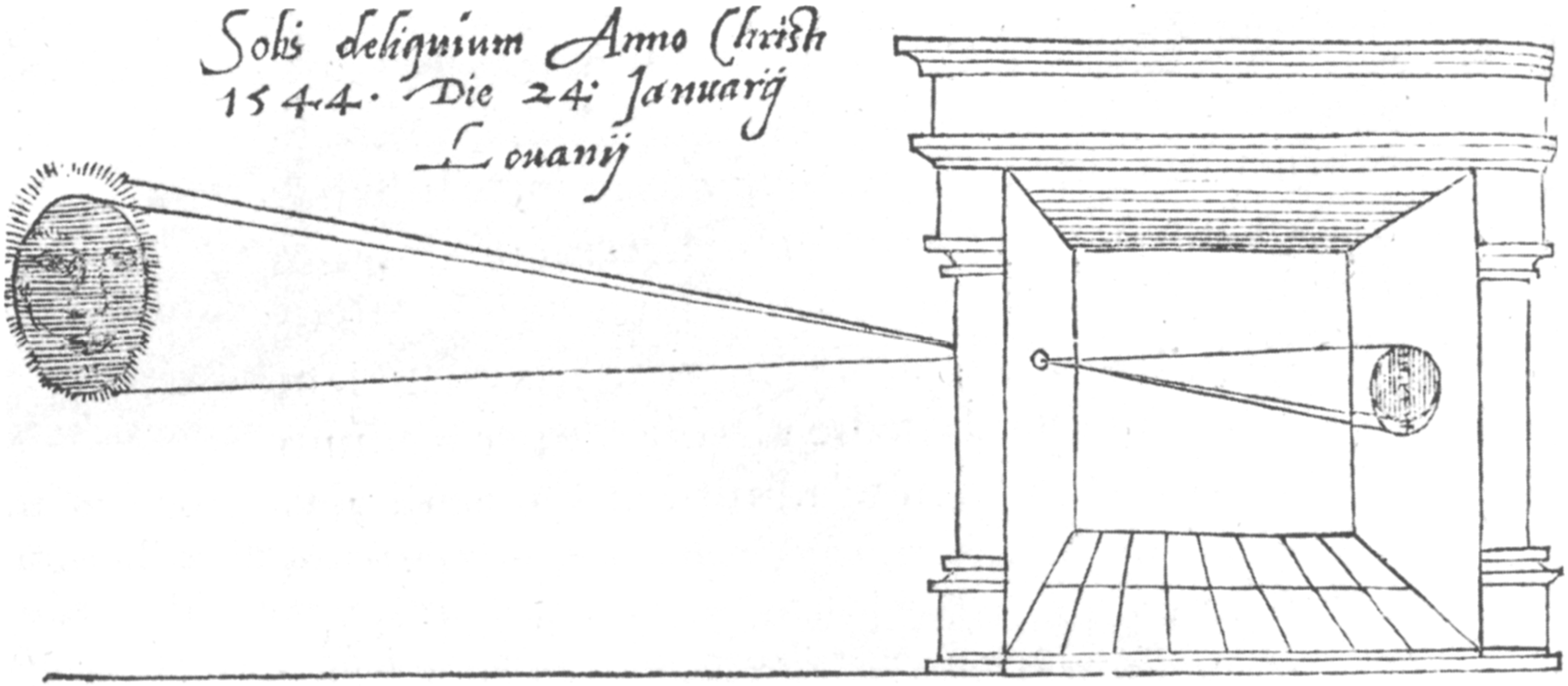
Durer’s Ray casting machine

- Albrecht Durer, 16th century

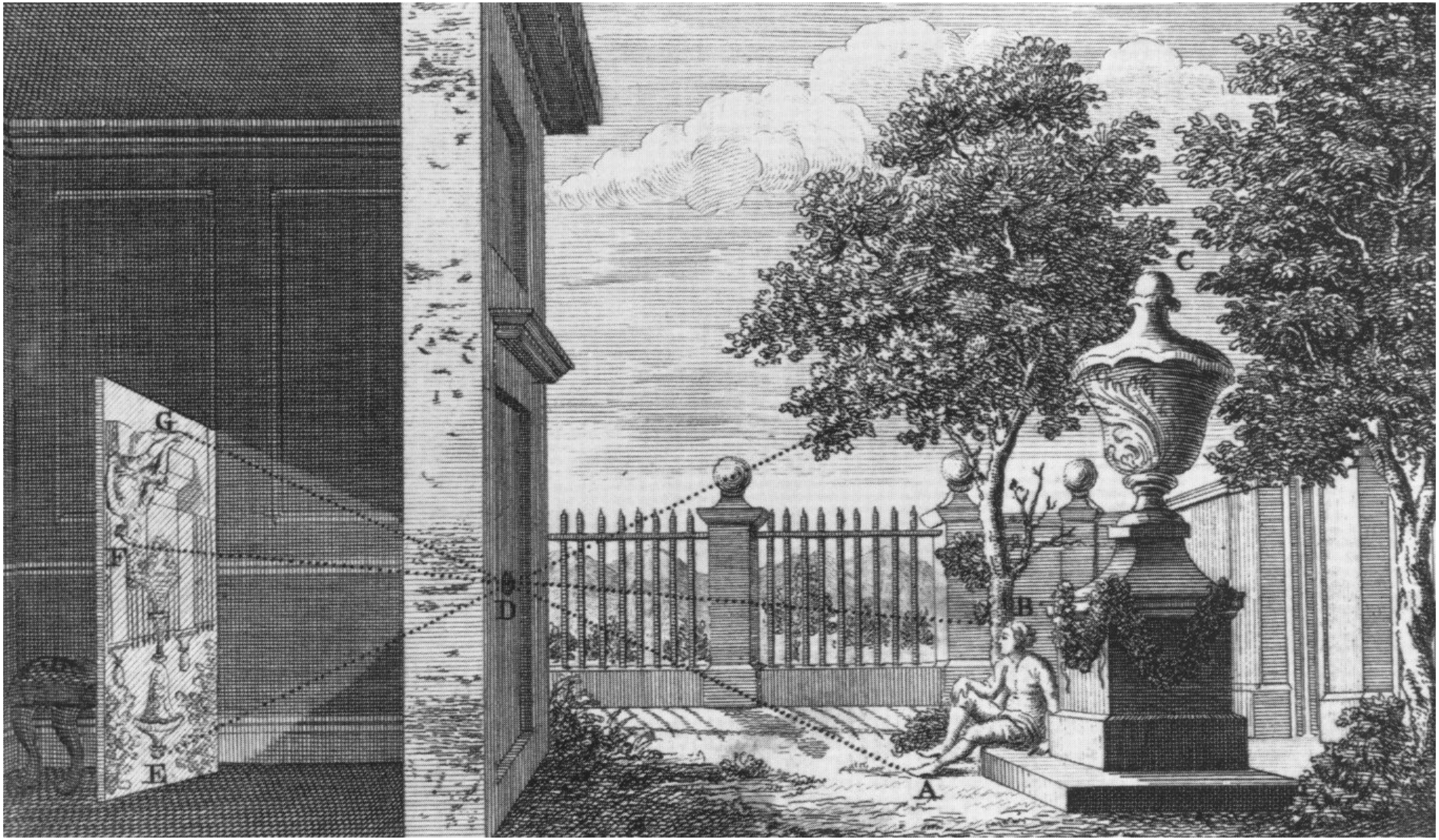


Oldest illustration

- From. R. Gemma Frisius, 1545

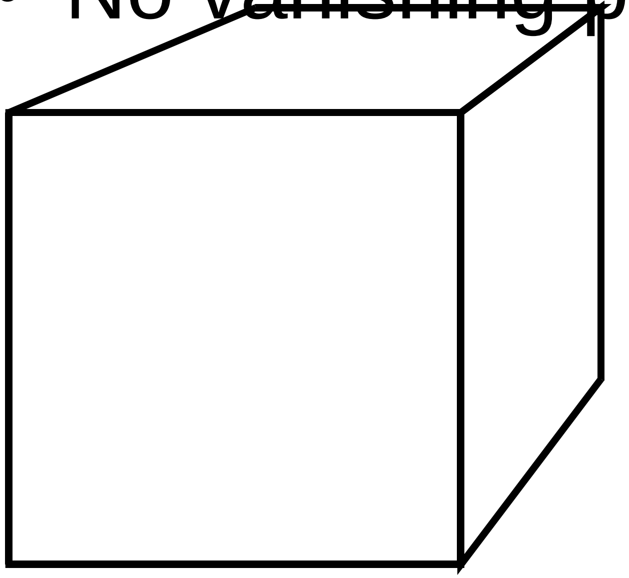


Camera Obscura

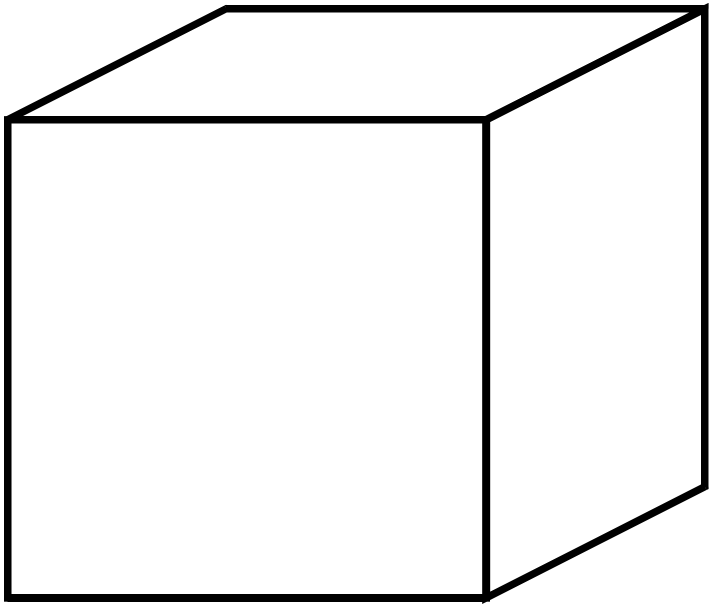


Orthographic camera

- Parallel projection
- No foreshortening
- No vanishing point

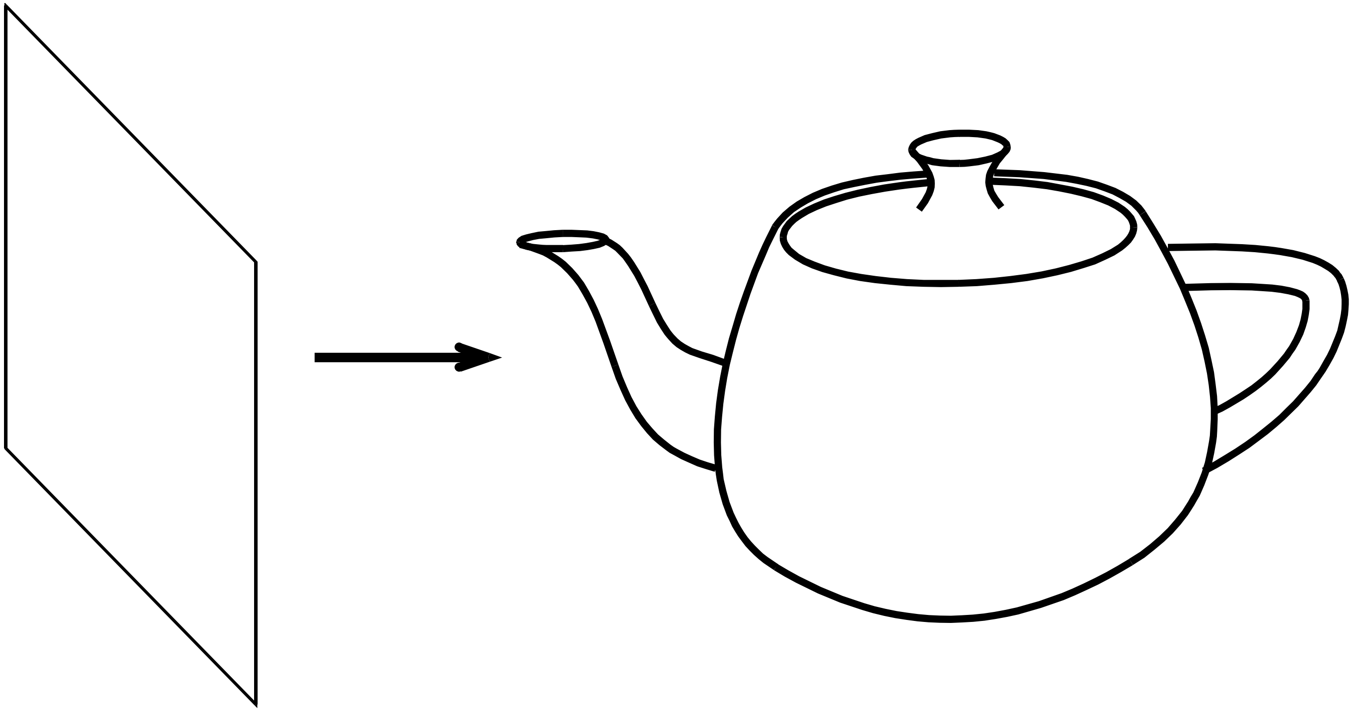


perspective



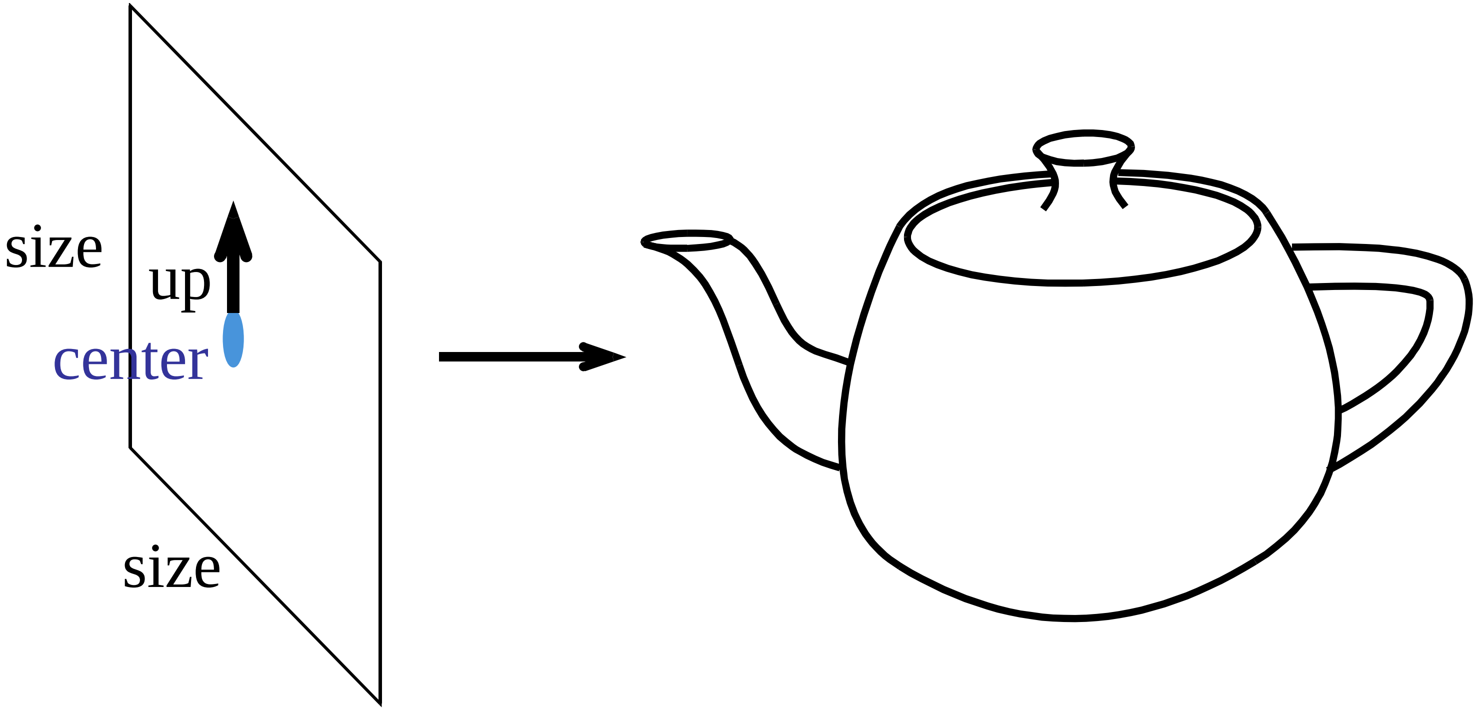
orthographic

Orthographic camera description



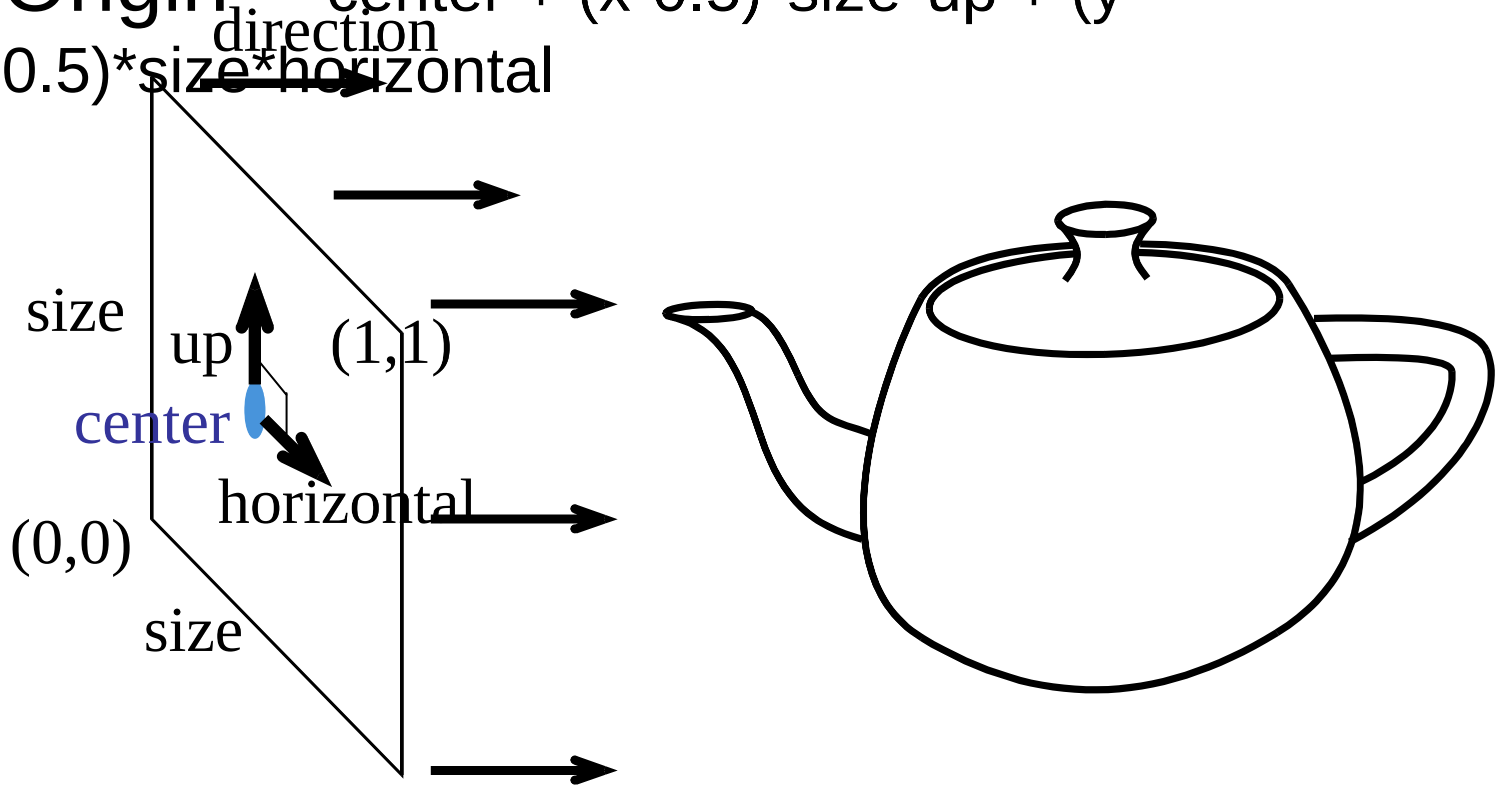
Orthographic camera description

- Direction
- Image center
- Image size
- Up vector



Orthographic ray generation

- Direction is constant
- Origin = center + (x-0.5)*size*up + (y-0.5)*size*horizontal



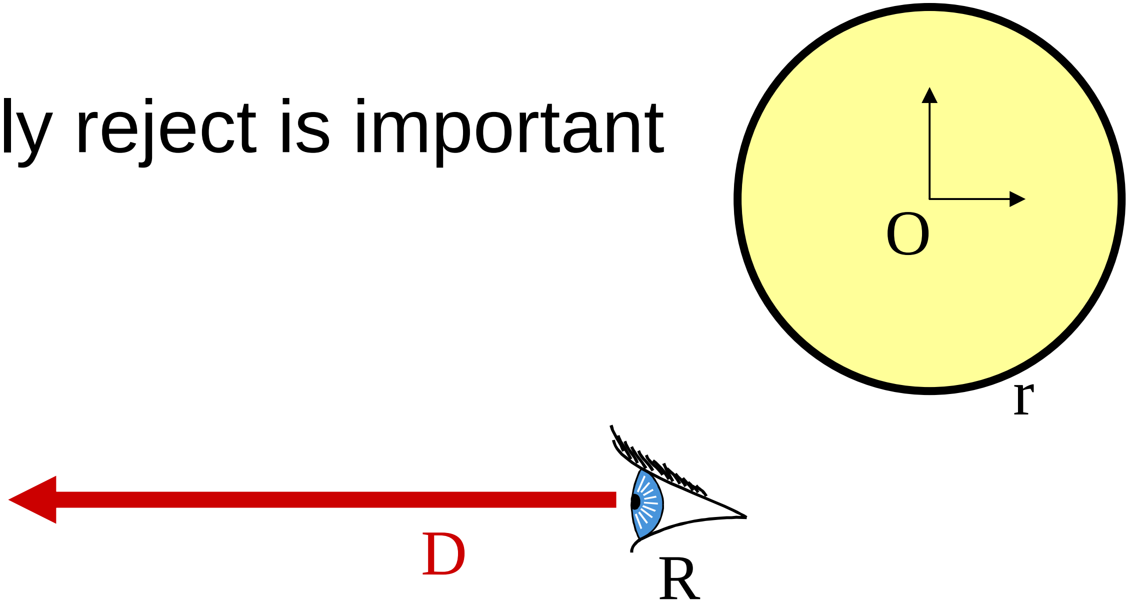
Other weird cameras

- E.g. fish eye, omnimax, panorama

Geometric ray-sphere intersection

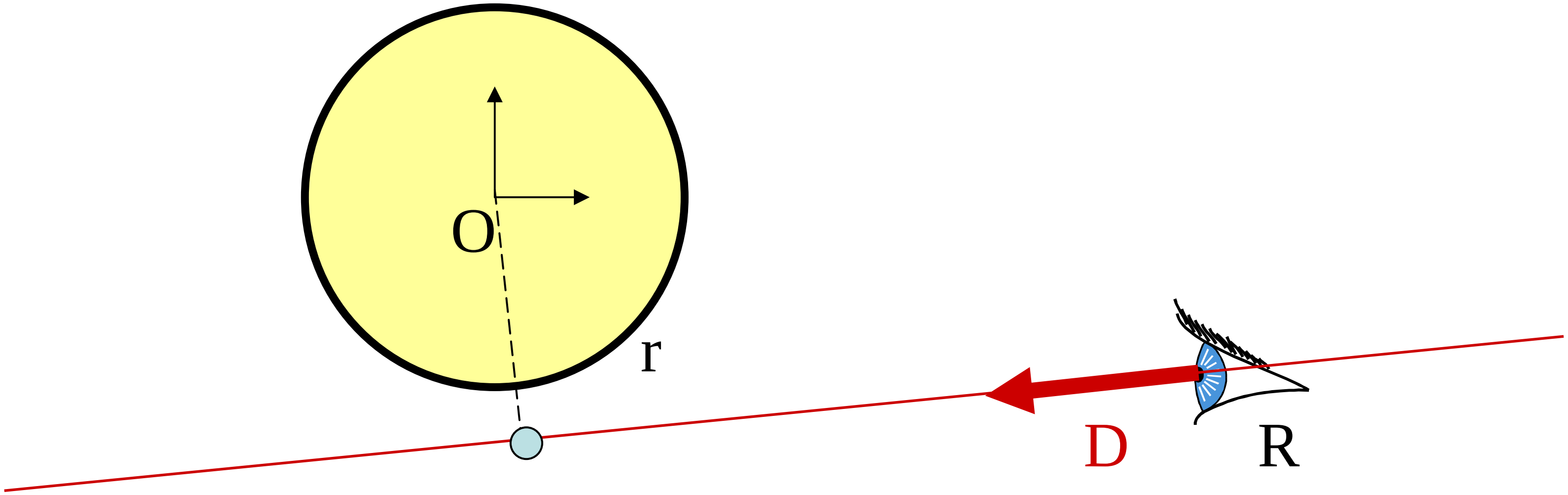
- Try to shortcut (easy reject)
- e.g.: if the ray is facing away from the sphere
- Geometric considerations can help

- In general, early reject is important



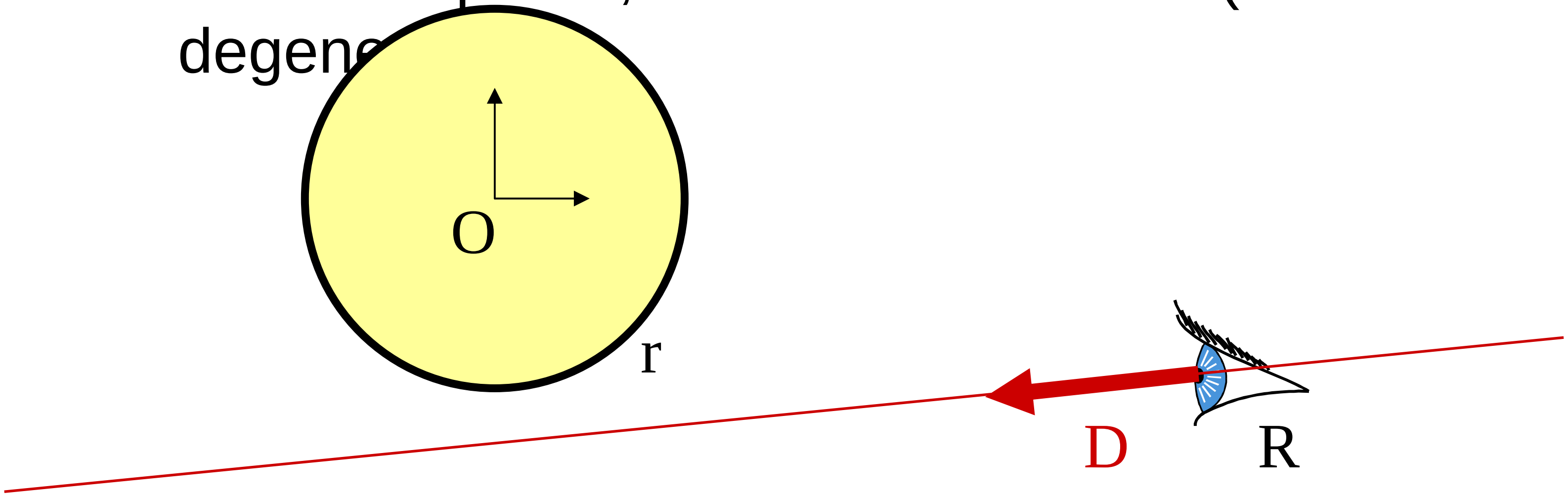
Geometric ray-sphere intersection

- What geometric information is important?
  - Inside/outside
  - Closest point
  - Direction



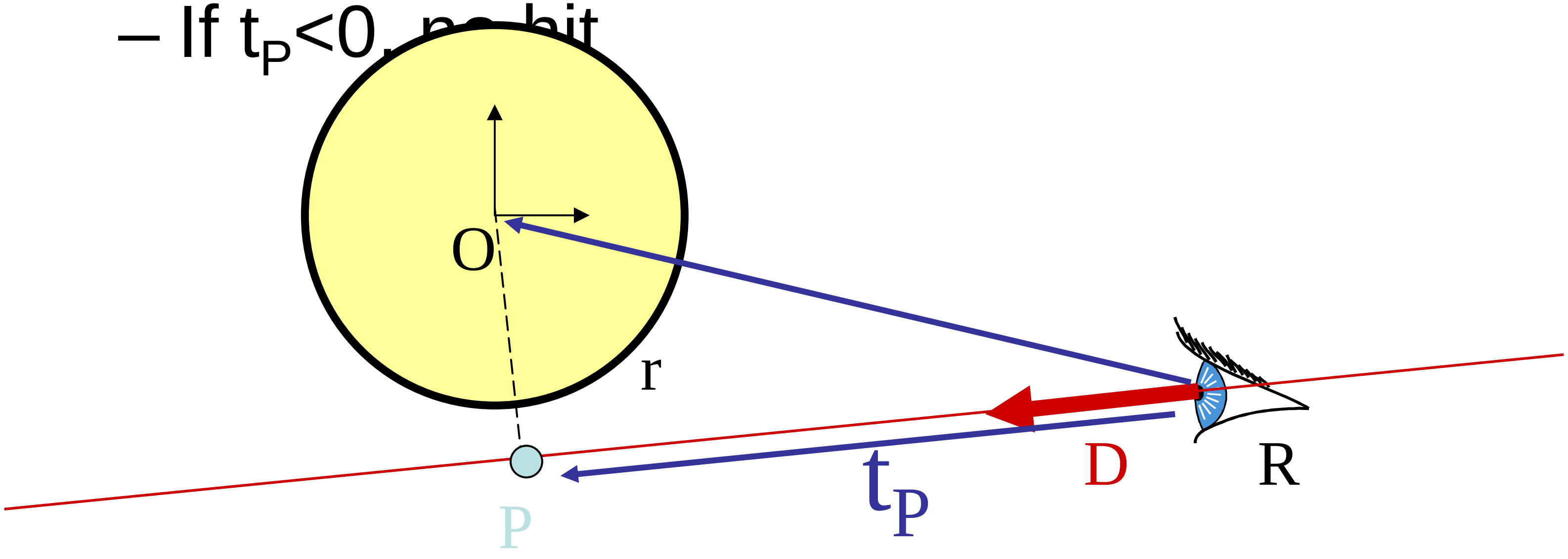
Geometric ray-sphere intersection

- Find if the ray's origin is outside the sphere
  - $R^2 > r^2$
  - If inside, it intersects
  - If on the sphere, it does not intersect (avoid degene



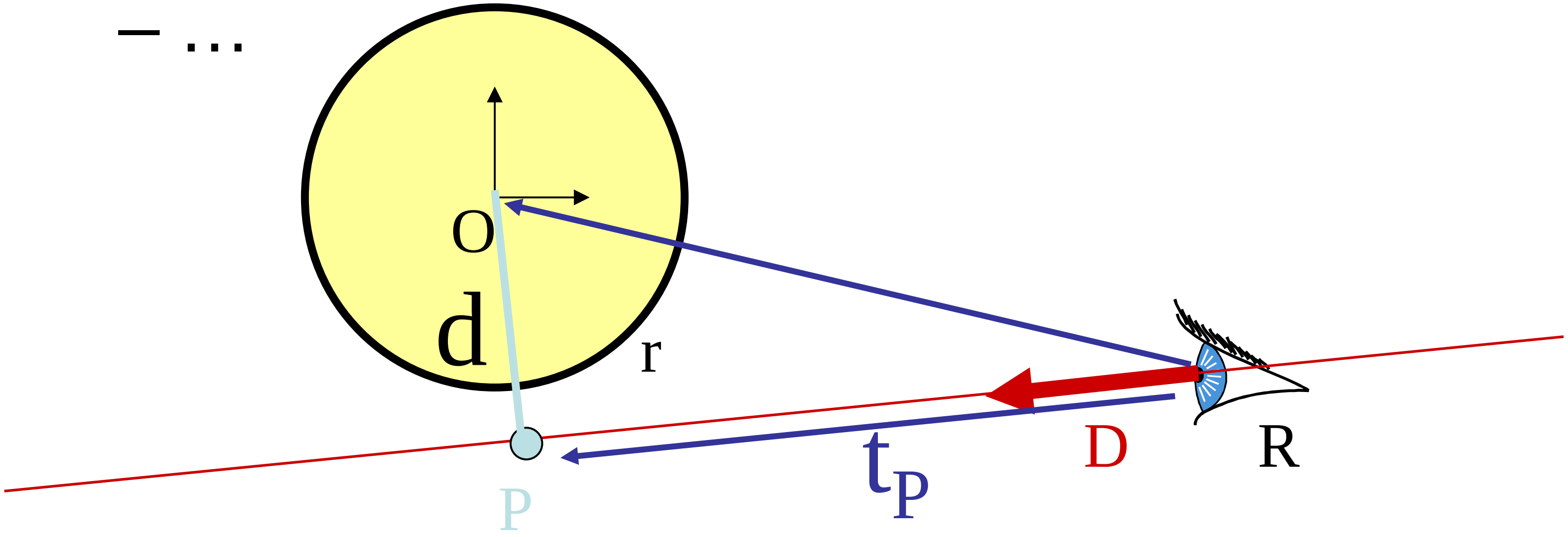
Geometric ray-sphere intersection

- Find if the ray's origin is outside the sphere
- Find the closest point to the sphere center
  - t_P=RO.D
  - If t_P<0, no hit



Geometric ray-sphere intersection

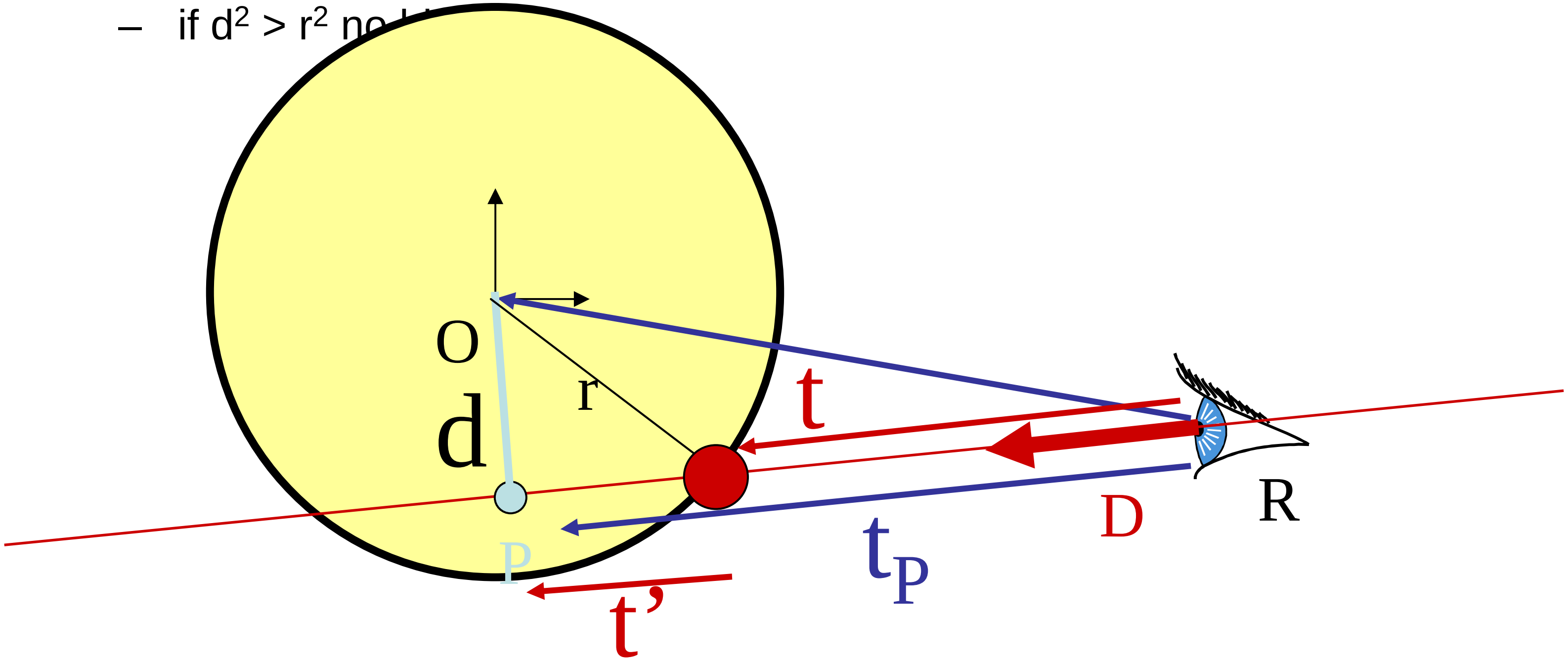
- Find if the ray's origin is outside the sphere
- Find the closest point to the sphere center
  - If $t_P<0$, no hit
- Else find squared distance d^2
  - Pythagoras: $d^2=R^2-t_P^2$
  - if $d^2 > r^2$, no hit
  - …



Geometric ray-sphere intersection

- Find if the ray's origin is outside the sphere
- Find the closest point to the sphere center
  - If $t_P<0$, no hit
- Else find squared distance d^2
  - if $d^2 > r^2$ no h

- If outside $t = t_P - t'$
  - $t'^2+d^2=r^2$
- If inside $t = t_P + t'$

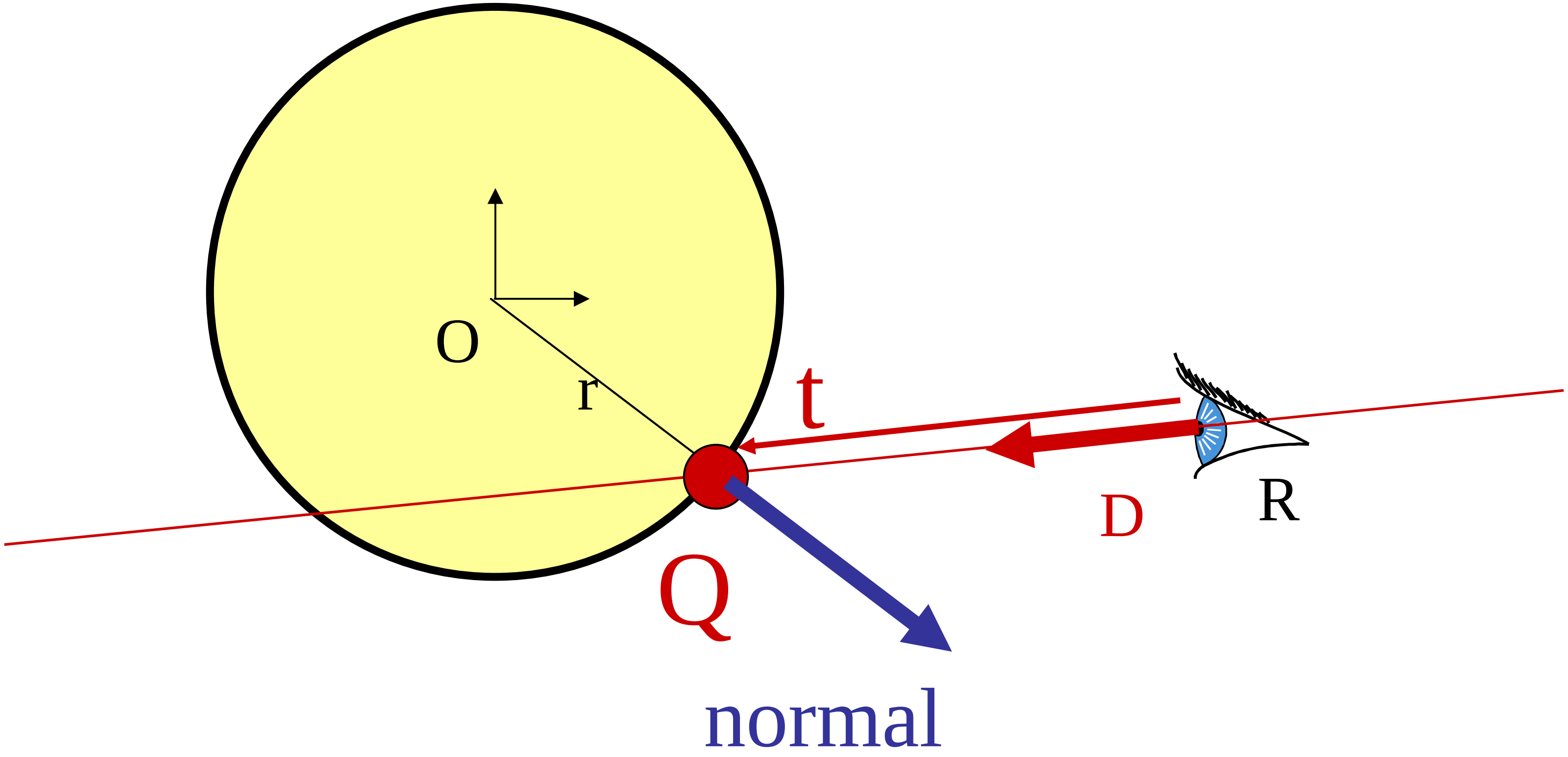


Geometric vs. algebraic

- Algebraic was more simple
  (and more generic)
- Geometric is more efficient
  - Timely tests
  - In particular for outside and pointing away

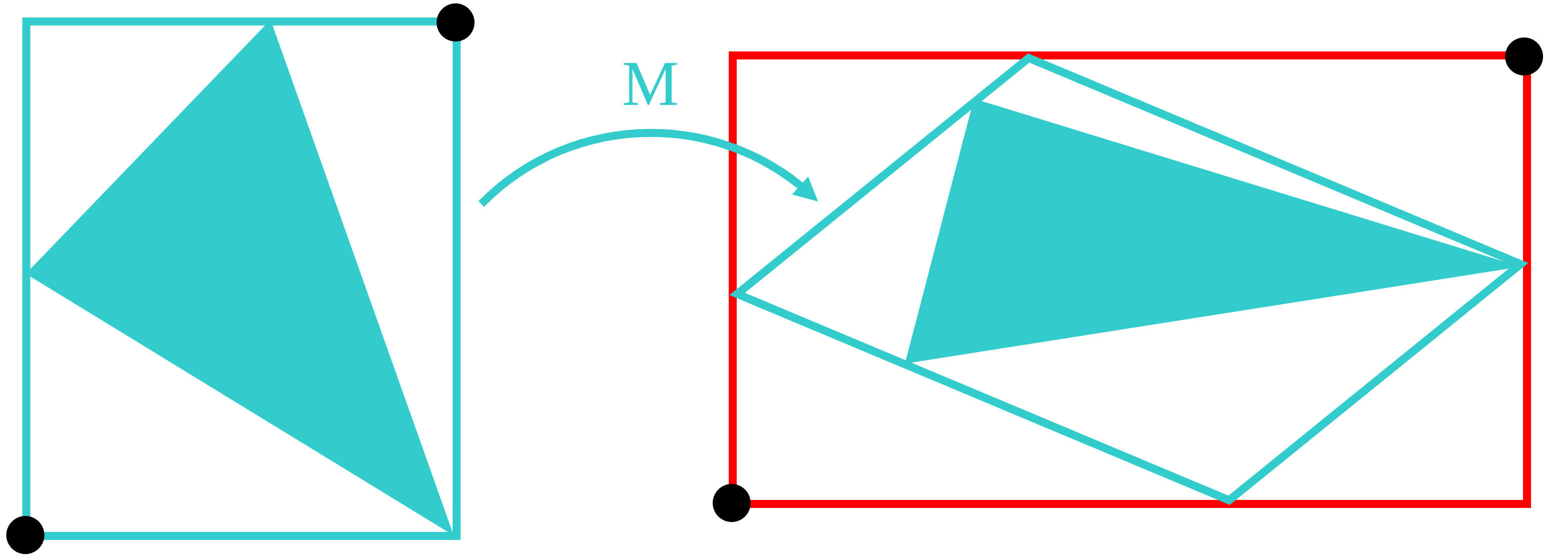
Normal

- Simply Q/||Q||



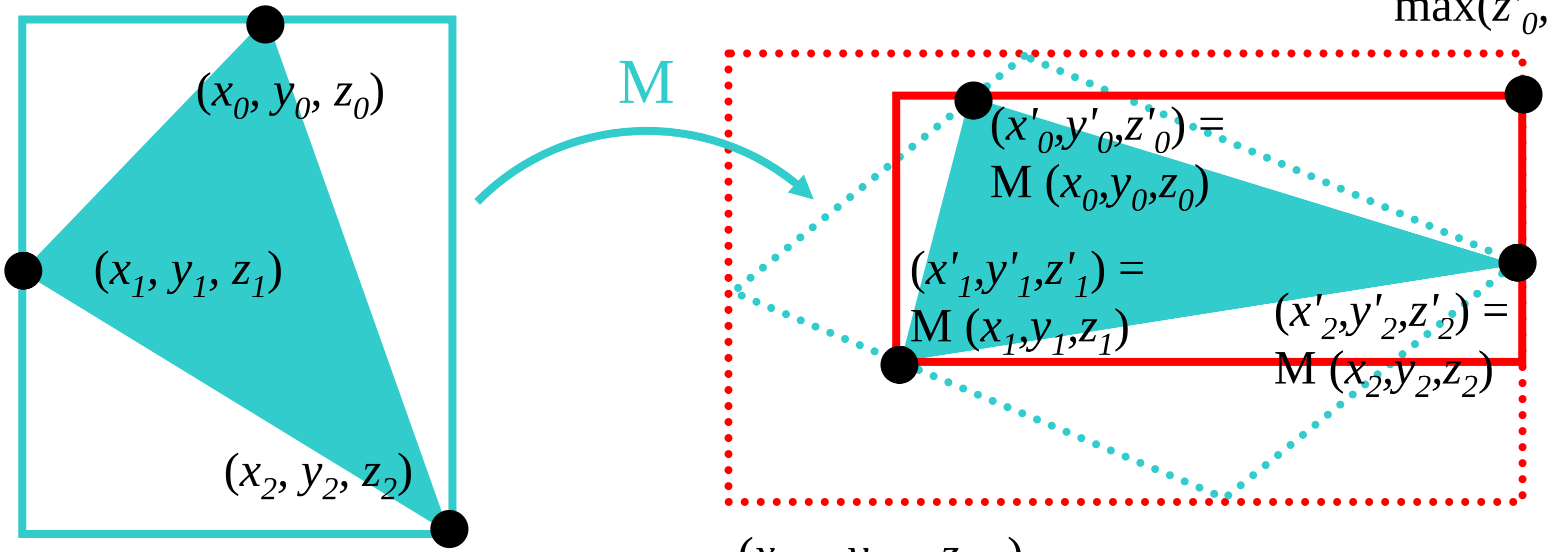
Special Case: Transformed Triangle

Can we do better?



Special Case: Transformed Triangle

$(x_{max}, y_{max}, z_{max})$
$= (\max(x'_0,x'_1,x'_2),$
$\max(y'_0,y'_1,y'_2),$
$\max(z'_0,z'_1,z'_2))$



(x_0, y_0, z_0)

(x_1, y_1, z_1)

(x_2, y_2, z_2)

M

$(x'_0,y'_0,z'_0) =$
$\mathrm{M}\ (x_0,y_0,z_0)$

$(x'_1,y'_1,z'_1) =$
$\mathrm{M}\ (x_1,y_1,z_1)$

$(x'_2,y'_2,z'_2) =$
$\mathrm{M}\ (x_2,y_2,z_2)$

$(x_{min}, y_{min}, z_{min})$
$= (\min(x'_0,x'_1,x'_2),$
$\min(y'_0,y'_1,y'_2),$
$\min(z'_0,z'_1,z'_2))$

Non-linearity of variance

$$\sigma^2 = E[(x-E[x])^2] = \int_{-\infty}^{\infty}(x-E[x])^2 p(x)dx$$

- Variance is not linear !!!!

$a^2\ \sigma^2[x]$

□ $\sigma^2[ax]=$

Non-linearity of variance

- Consider two random variable x and x

$$\sigma^2(x_1+x_2) = E[(x_1+x_2)^2] - (E[x_1+x_2])^2$$
$$= E[x_1^2 + 2x_1x_2 + x_2^2] - (E[x_1] + E[x_2])^2$$
$$= E[x_1^2]+2E[x_1x_2]+E[x_2^2]-E[x_1]^2-2E[x_1]E[x_2]-E[x_2]^2$$
$$= \sigma^2[x_1] + \sigma^2[x_2] + 2E[x_1x_2] - 2E[x_1]E[x_2]$$

- We define the covariance

Cov[x_1,x_2] = E[x_1x_2] - E[x_1] E[x_2]

□ $\sigma^2[x_1+x_2]$ = $\sigma^2[x_1] + \sigma^2[x_2] + 2$ Cov[x_1,x_2]

Non-linearity of variance, covariance

- Consider two random variable x_1 and x_2
- We define the covariance
  $\mathrm{Cov}[x_1,x_2] \;=\; E[x_1x_2] - E[x_1]\,E[x_2]$
  - Tells how much they are big at the same time
  - **Null if variables are independent**

□ $\sigma^2[x_1+x_2] \;=\; \sigma^2[x_1] + \sigma^2[x_2] + 2\,\mathrm{Cov}[x_1,x_2]$

Recap

- Expected value is linear
  - $E[ax_1+bx_2]=aE[x_1]+bE[x_2]$
- Variance is not
- For two independent variables
  - $\sigma^2[x_1+x_2]=\sigma^2[x_1]+\sigma^2[x_2]$
  - If not independent, needs covariance
- $\sigma^2[ax]=a^2\sigma^2[x]$