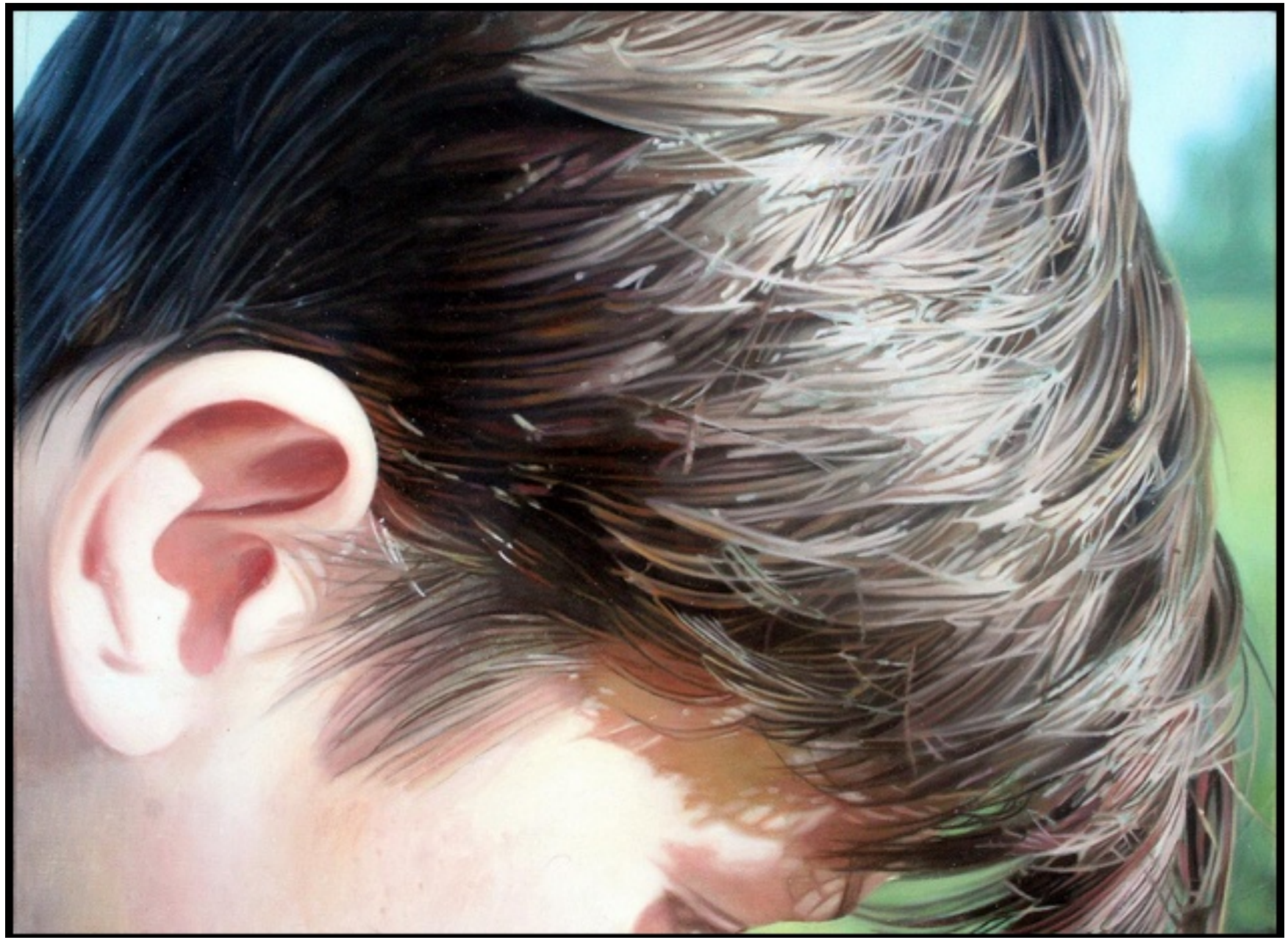
Global Illumination

CS7GV3 – Real-time Rendering

Photorealism



Profil by Jaques Bodin.

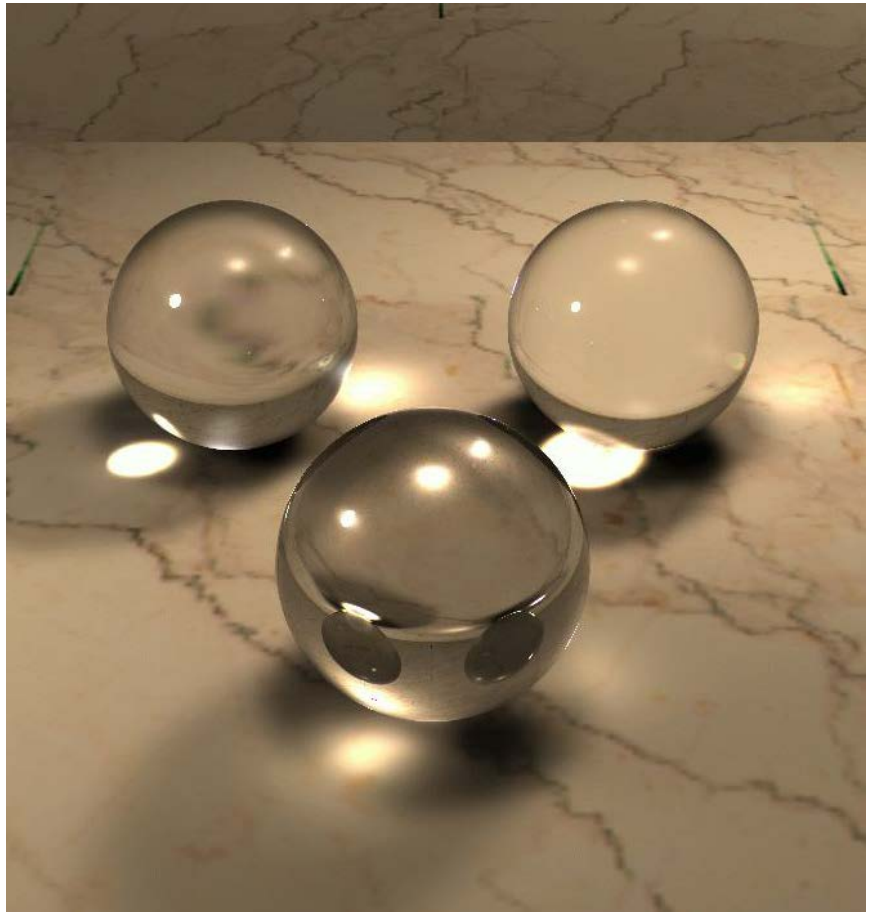
Photorealism artists exploit high detail, depth of field, wide angle, lighting, and focus anomalies

Richard Estes, Supermarket, Columbus Avenue, NY, 2008

Richard Estes. Bus with reflection of the Flatiron building. Oil on board. 36x48in. 1968.

Ray traced rendering in POV-Ray © Gilles Tran

Global Illumination



The Rendering Equation

Kajiya's [1986] representation of light transport in the scene, describing the amount of light going from any point x to another point x':

$$I(x,x') = g(x,x')[\varepsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')dx'']$$

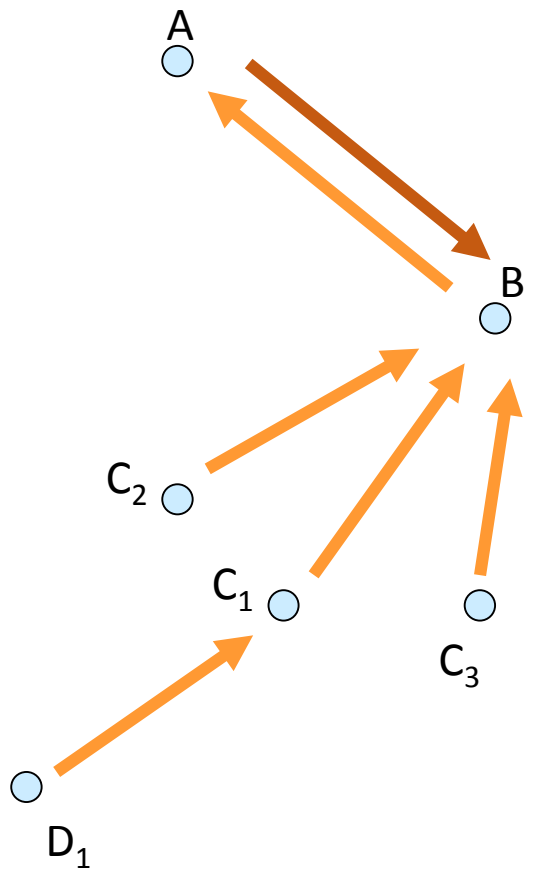
- I(x, x') = intensity of light passing from x to x'
  - (two point transport)
- g(x, x') = $\begin{cases} 0 \text{ if } x \text{ and } x' \text{ are not mutually visible} \\ 1/r^2 \text{ where } r = |\overrightarrow{xx'}| \end{cases}$ Attenuation (distance from x to x')

  (geometry factor)
- *e* (x, x') = intensity of light emitted by x and passing to x'
- r (x, x', x") = bi-directional reflectance scaling factor for light passing from x" to x by reflecting off x'
- S = all surfaces in the scene
- *p* (x, x', x") = is related to the intensity of light scatted from x'' to x by a patch of surface x'

James T. Kajiya. 1986. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '86), David C. Evans and Russell J. Athay (Eds.). ACM, New York, NY, USA, 143-150. DOI=http://dx.doi.org/10.1145/15922.15902

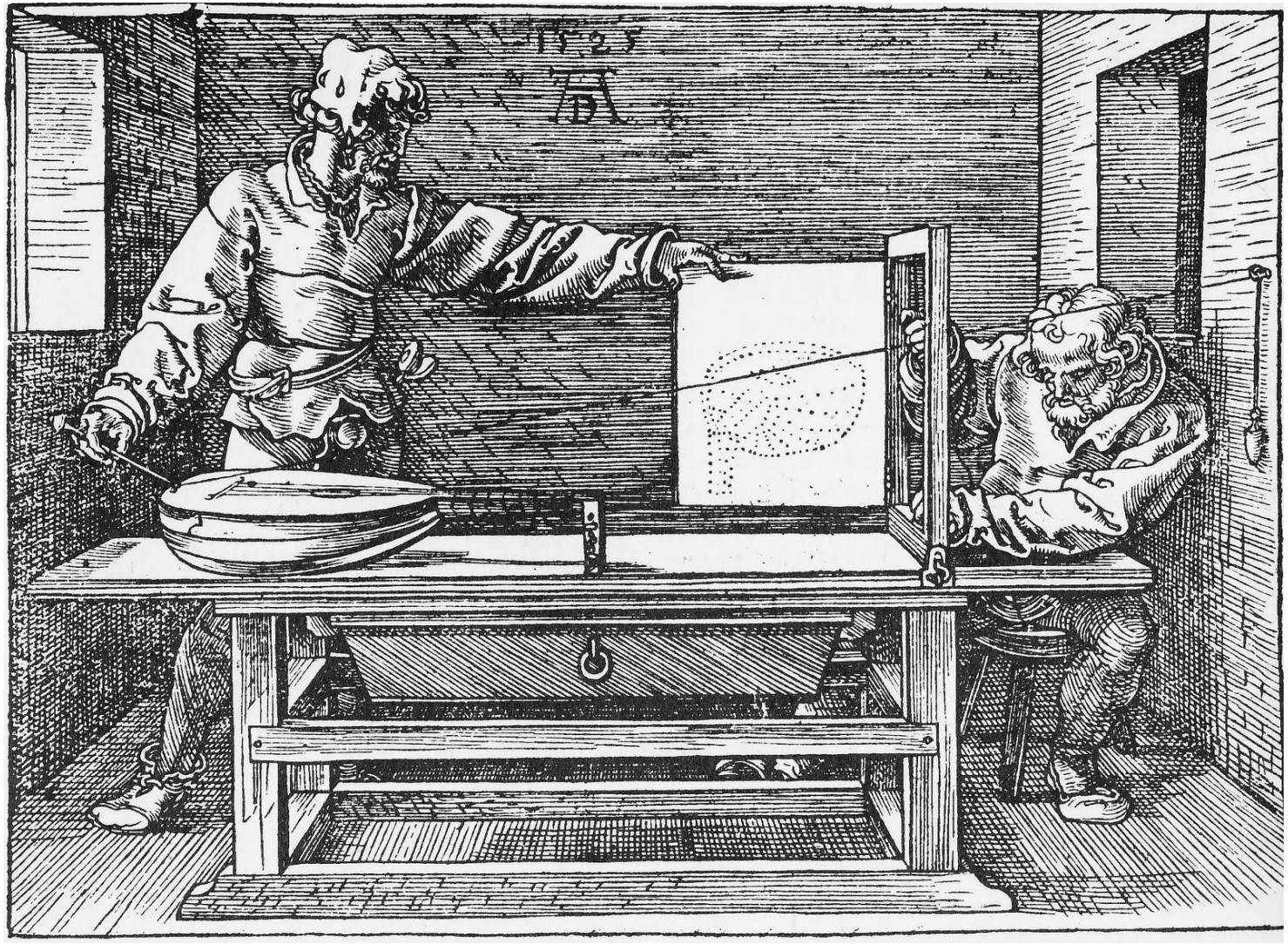
Recursive Light Transport

- Light_transport_from B to A depends on:
  - Emmitance of B
  - Visibility of B from A
  - Distance to B from A
- Also:
  - Reflectance of B
  - Light transport from C_i to B
    - Emmitance of C
    - Visibility of C from B
    - Distance to C from B



Ray-tracing

Perspective



"Man Drawing a Lute" Albrecht Durer (1471-1528)

Ray Tracing

- A recursive view-dependent rendering algorithm
  - Photo-realistic technique
- Current ray tracing methods are attributed to Turner Whitted
- First implementation of ray tracing in computer graphics = Appel (IBM 1968)

Took 74 minutes to render this image

Turner Whitted. 1980. An improved illumination model for shaded display. *Commun. ACM* 23, 6 (June 1980), 343-349. DOI=http://dx.doi.org/10.1145/358876.358882

Ray Tracing

$$I = I_{local} + k_{rg}I + k_{tg}\mathrm{I}$$

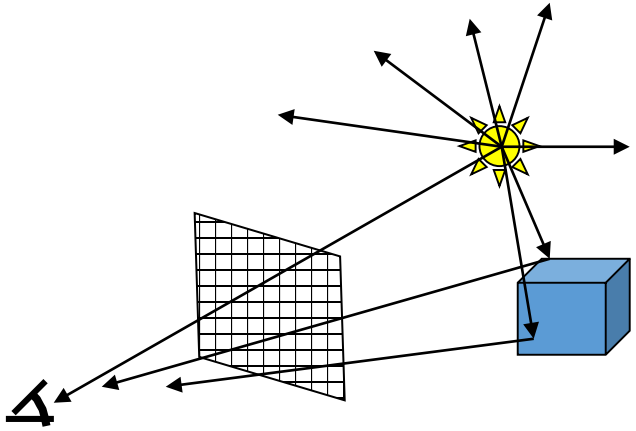
Local term (as in Phong)

Reflected

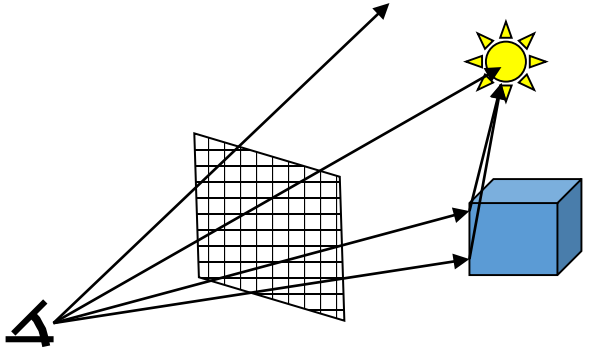
Transmitted

- The Direct Diffuse Term I_{local} is calculated empirically (using Phong illumination)
- In the global ray-traced terms, diffusion of light due to surface imperfections is totally ignored.

Ray Tracing from Eye



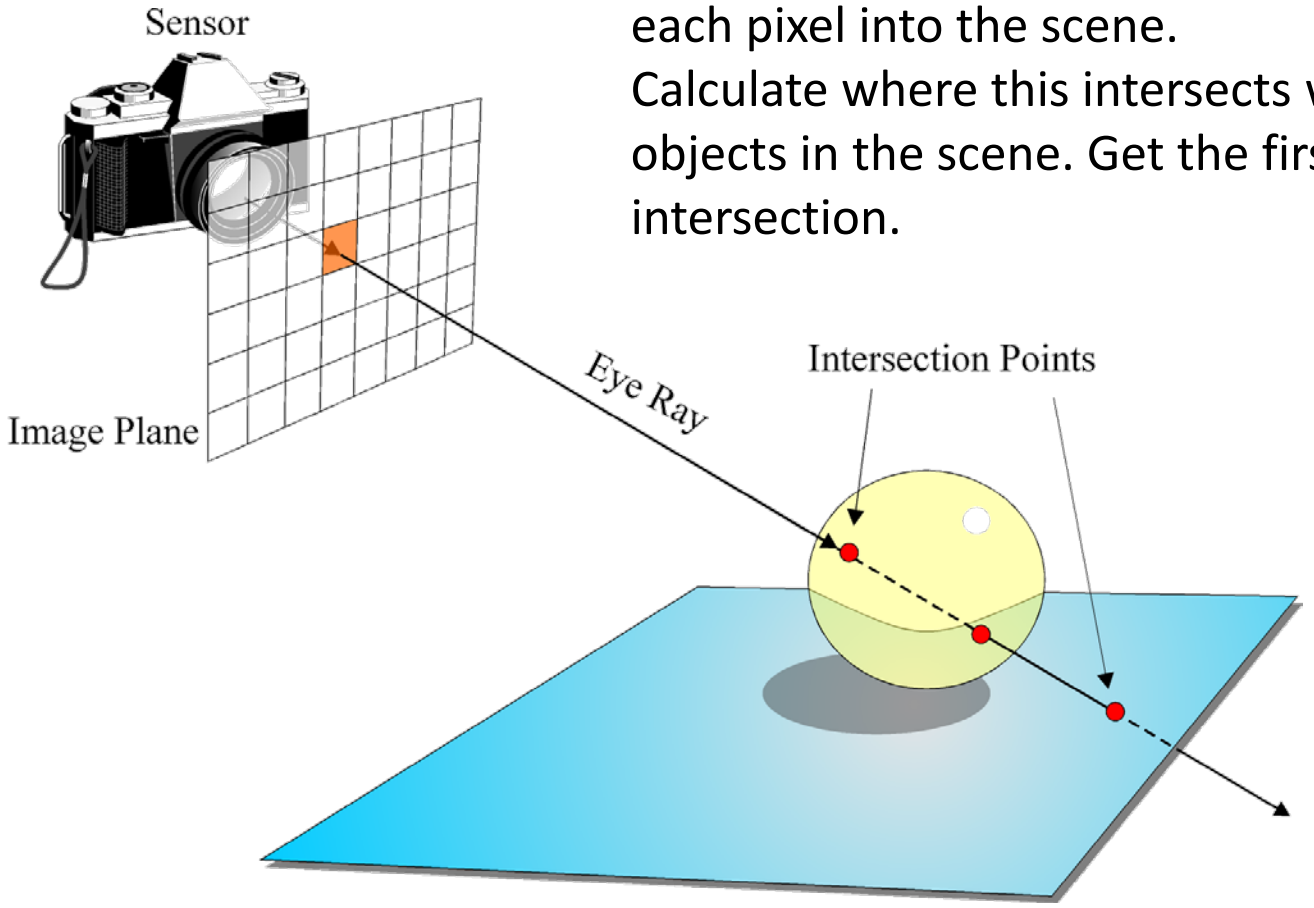
Tracing from light source



Traditional ray-tracing

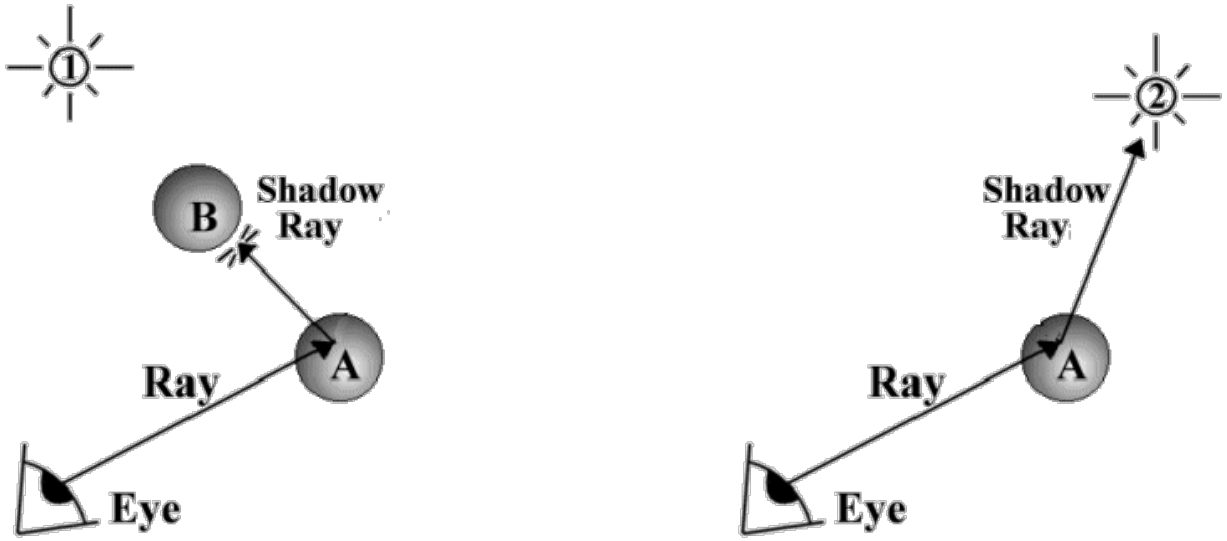
Starting at the light position traces many rays that never reach the eye. Thus the traditional ray-tracing method is to start at the eye and trace rays back-wards to the source.

Eye Ray and Object Intersection



Cast a ray from camera position through each pixel into the scene.
Calculate where this intersects with objects in the scene. Get the first intersection.

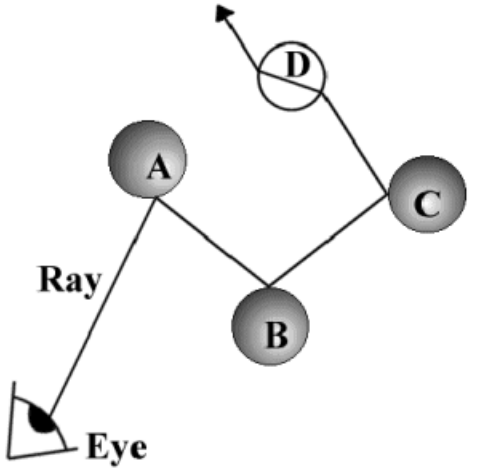
Shadow Rays



- Extend a “Shadow Feeler” (or light ray) and see if it is occluded by an object in the scene
- If so the object is in shadow from this light source (no contribution at this point)
- Otherwise solve the **Phong model** to calculate the contribution of this point to the colour of the pixel.
- If object is diffuse stop here.

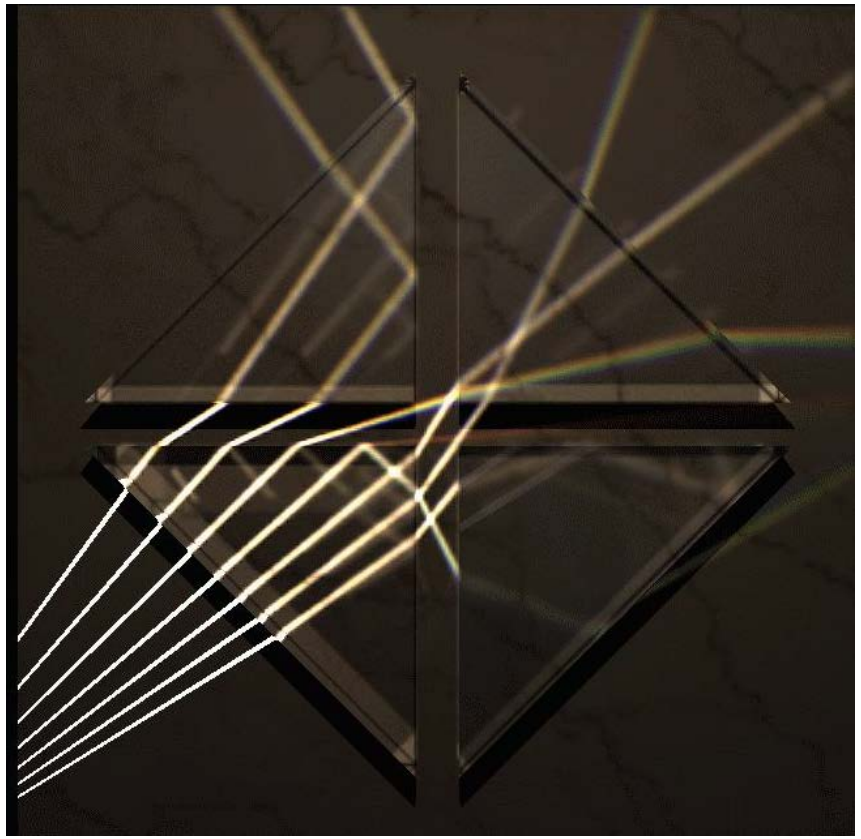
Reflection

- If object is specular THEN create a ray in the direction of perfect specular reflection.



- For this new ray, we will again check whether…
  - it intersects an object,
  - if so is this object in shadow
  - what is the contribution of this object to the colour of the pixel?
- Each ray contributes to the colour of the pixel it originated from

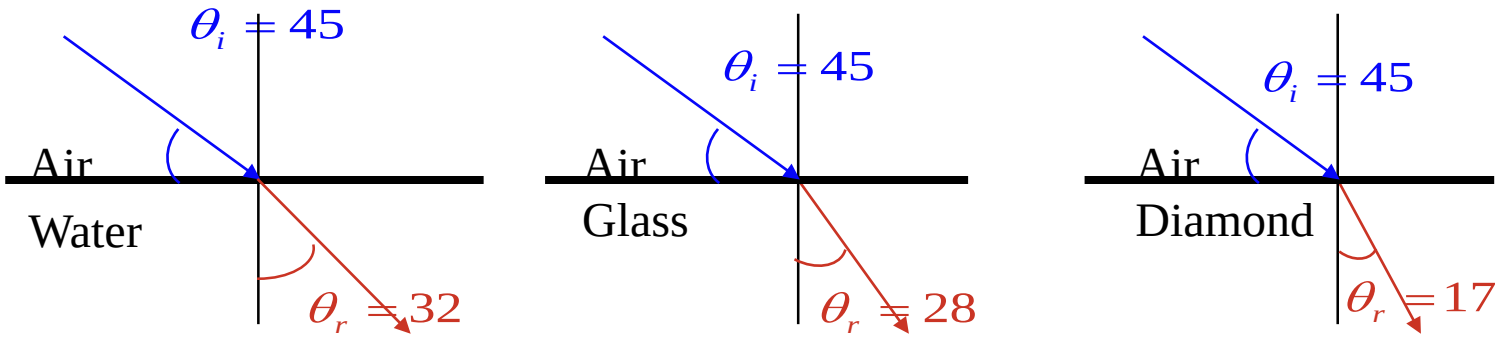
Refraction



- Similarly if the object is transmissive (not-opaque) then generate a transmitted (or refracted) ray and repeat…

The Angle of Refraction

- When light passes from a material of one optical density to another it changes direction.
- The amount by which the direction changes is determined by the optical densities of the two media.

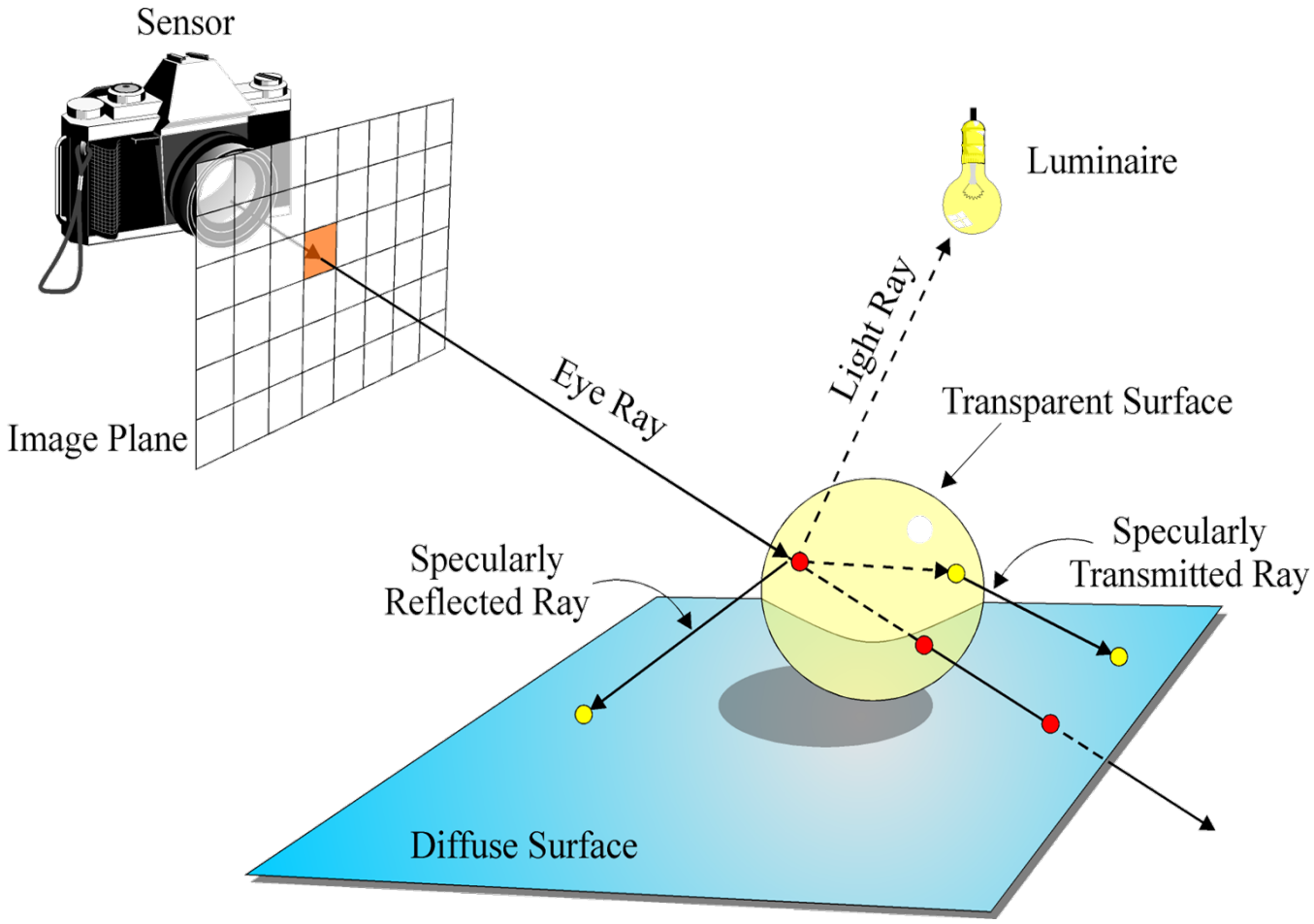


- Optical density (and thus the amount of bending is related to a value we call the index of refraction (ior) of the material

Index of Refraction (ior)

Material	Index of Refraction	
Vacuum	1.0000	<--lowest optical density
Air	1.0003	
Ice	1.31	
Water	1.333	
Ethyl Alcohol	1.36	
Plexiglas	1.51	
Crown Glass	1.52	
Light Flint Glass	1.58	
Dense Flint Glass	1.66	
Zircon	1.923	
Diamond	2.417	
Rutile	2.907	
Gallium phosphide	3.50	<--highest optical density

Ray Tracing Summary



Ray Tracing Algorithm

$$I = \quad I_{local} \qquad + \qquad k_{rg}I \qquad + \quad k_{tg}\mathrm{I}$$

1) Cast a ray
2) Determine Intersections
3) For closest Intersection:
   - Extend light(shadow feeler) ray + calculate local term
   - Spawn Transmitted Ray (step 1)
   - Spawn Reflected Ray (step 1)

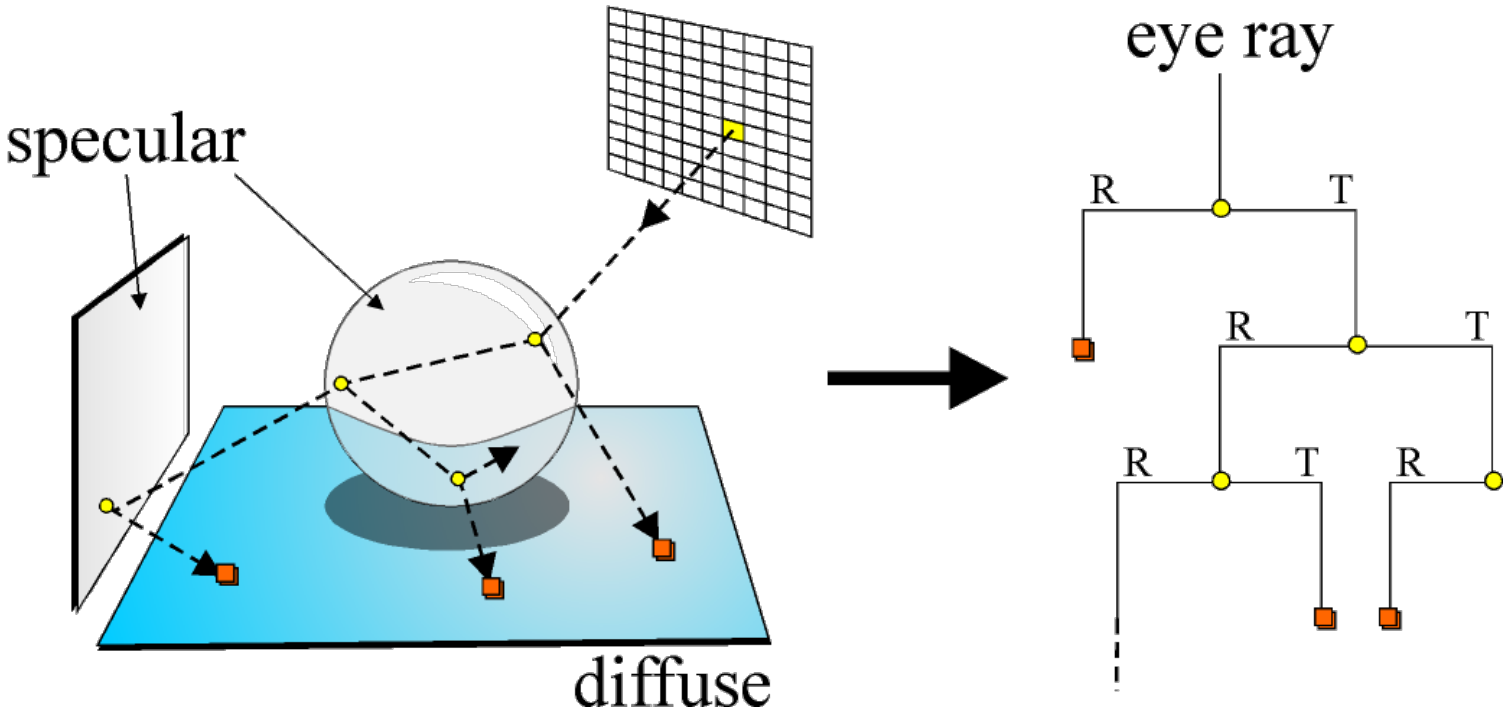
The Ray Tracing Algorithm

```
for each pixel in viewport
{
            determine eye ray for pixel
            intersection = trace(ray, objects)
            colour = shade(ray, intersection)
}
Intersection trace(ray, objects)
{
            for each object in scene
            intersect(ray, object)
            sort intersections
            return closest intersection
}
Colour shade(ray, Intersection)
{
            for each light in scene
                        col += Phong(trace(ray, light))
            col += shade(trace(ray, refl))
            col += shade(trace(ray, refr))
            return col;
}
```

Recursion

Ray Tree

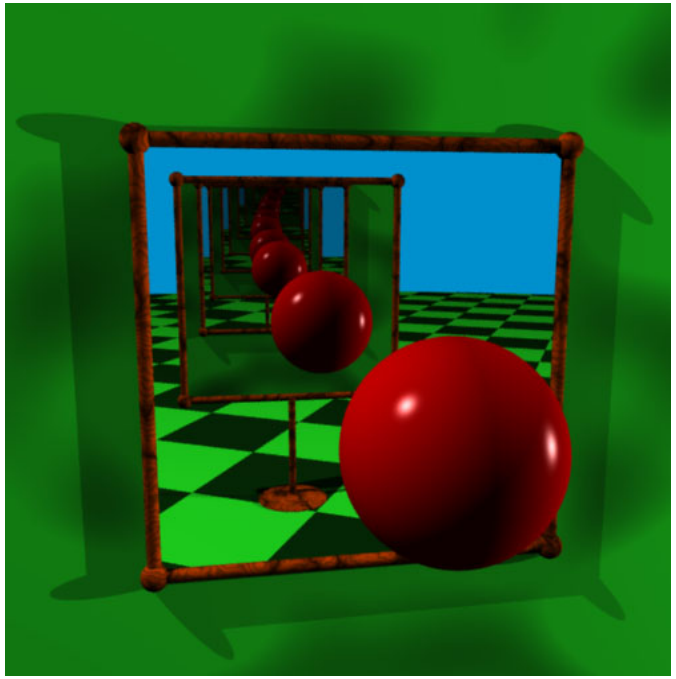
The spawning of reflected and refracted rays can be represented in *tree* structure.



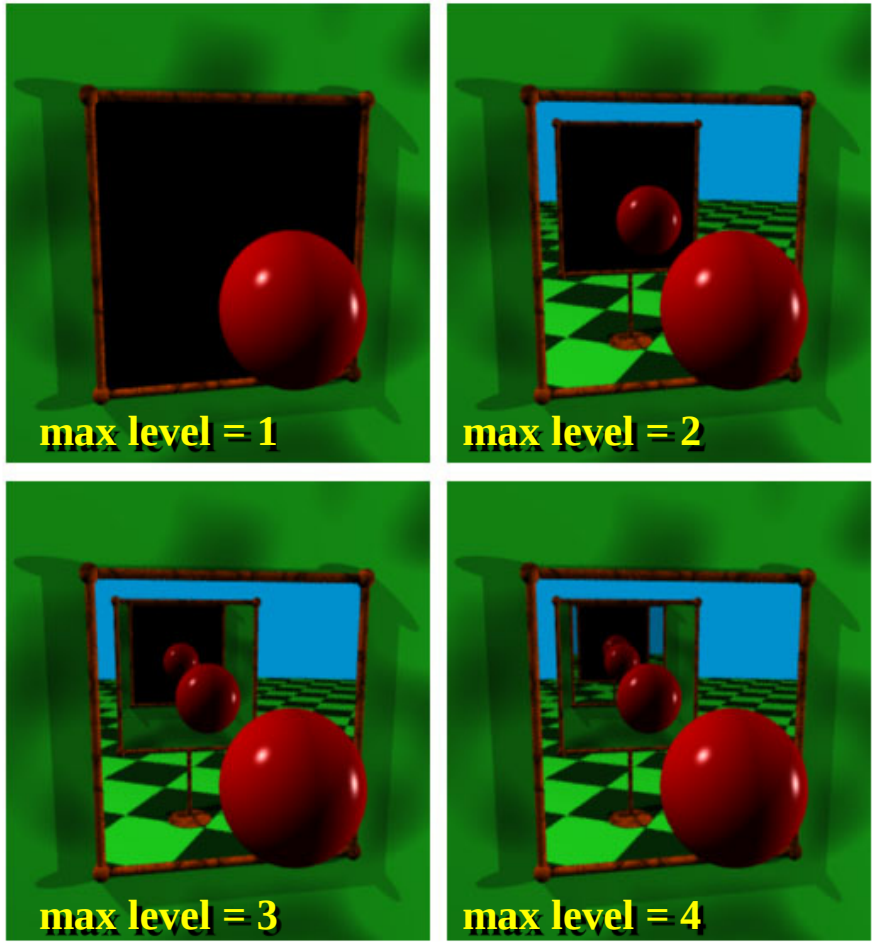
Terminating Recursion

- All of the spawned rays contribute to the pixel that the tree originated from.
- However each new ray contributes less and less to the pixel.
- Unlike in the real-world, we can't keep bouncing around for ever so we stop the recursion at some stage ➔ recursion clipping.
  - Stop after a set number of bounces.
  - Or stop when the contribution becomes less than a certain value.

Recursion Clipping



Very high maximum recursion level



Terminating Recursion

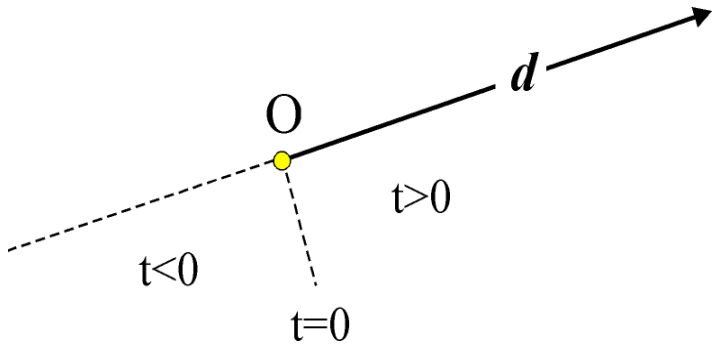
```
for each pixel in viewport
{
    determine eye ray for pixel
    intersection = trace(ray, objects)
    colour = shade(ray, intersection)
}

trace(ray, objects)
{
   for each object in scene
       intersect(ray, object)
   sort intersections
   return closest intersection
}
```

```
colour shade(ray, intersection)
{
    if no intersection
        return background colour

    for each light source
        if(visible)
            colour += Phong contribution

    if(recursion level < maxlevel and surface not diffuse)
    {
        ray = reflected ray
        intersection = trace(ray, objects)
        colour += ρ_refl*shade(ray, intersection)
        ray = transmitted ray
        intersection = trace(ray, objects)
        colour += ρ_trans* shade(ray, intersection)
    }
    return colour
}
```

The Ray

- A ray is mathematically the affine half-space defined by:

$$\mathbf{r} = O + t\vec{d} \quad t \geq 0$$



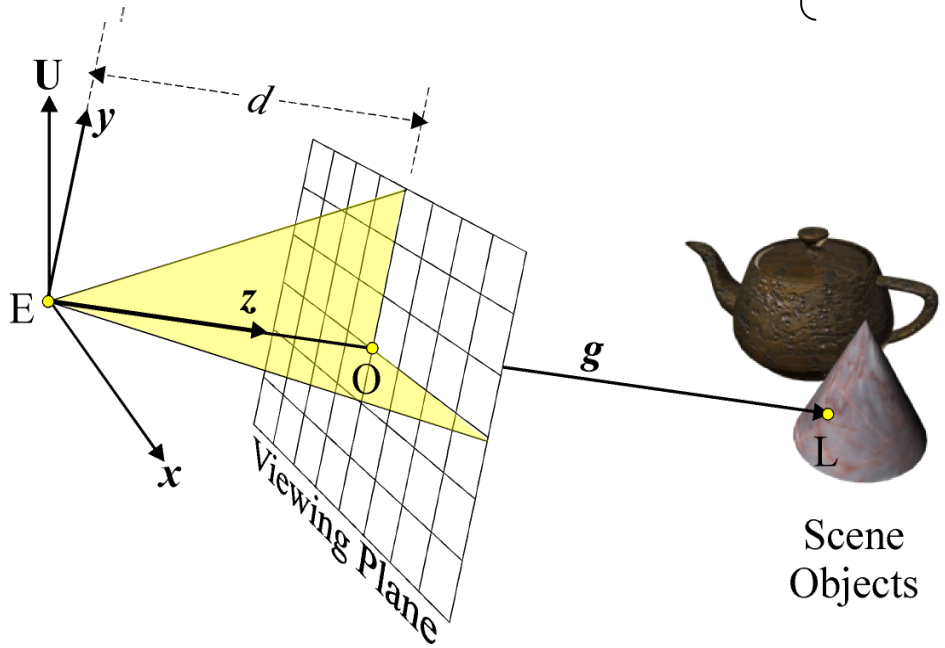
- All points on the ray correspond to some positive value of t, the parametric distance along the ray.
  - If d is normalised then t is the length along the ray of the point.

Ray Casting

- Assuming a pinhole camera model, a view is specified using:

camera view

E	eye/camera position
L	look point (defined gaze direction)
U	up vector (fixes orientation)
(θ, ϕ)	horizontal and vertical field of view angles
d	distance to viewing plane from eye point



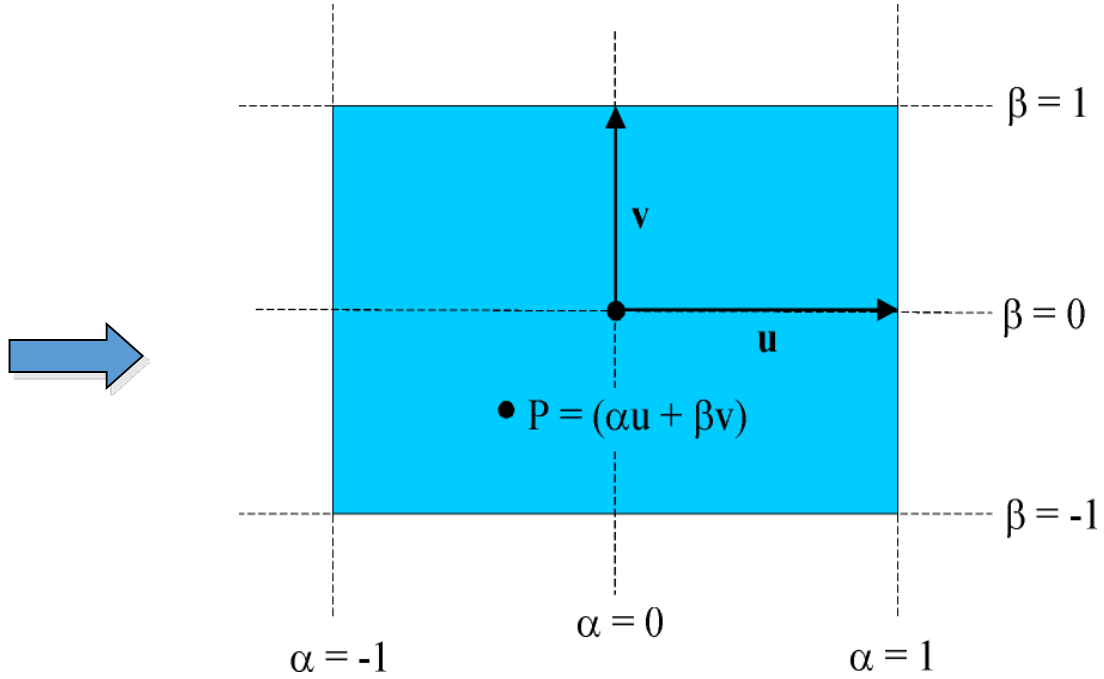
From this calculate the camera coordinate system:

$$
\begin{aligned}
\mathbf{g} &= L - E \\
\mathbf{z} &= \frac{\mathbf{g}}{||\mathbf{g}||} \\
\mathbf{x} &= \frac{\mathbf{z} \times \mathbf{u}}{||\mathbf{z} \times \mathbf{u}||} \\
\mathbf{y} &= \mathbf{x} \times \mathbf{z} \\
O &= E + d\mathbf{z}
\end{aligned}
$$

Eye Ray Construction

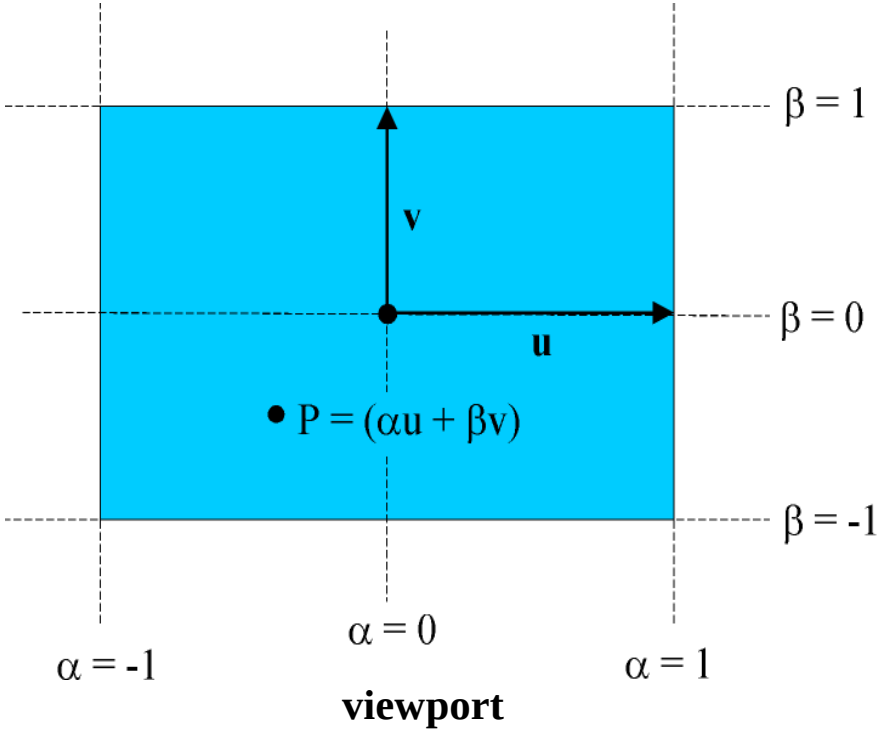
- We need to construct rays originating at the eye and passing through a point on the viewport.
- We need to relate the viewport and window co-ordinate systems.
  - Construct vectors u and v which span the viewing plane
  - Determine origin of viewport (usually defined as the center)
  - all points P on the viewport are linear combinations of u and v with scalars α and β defining the co-ordinate of the point:

$$P = \alpha\vec{u} + \beta\vec{v}$$

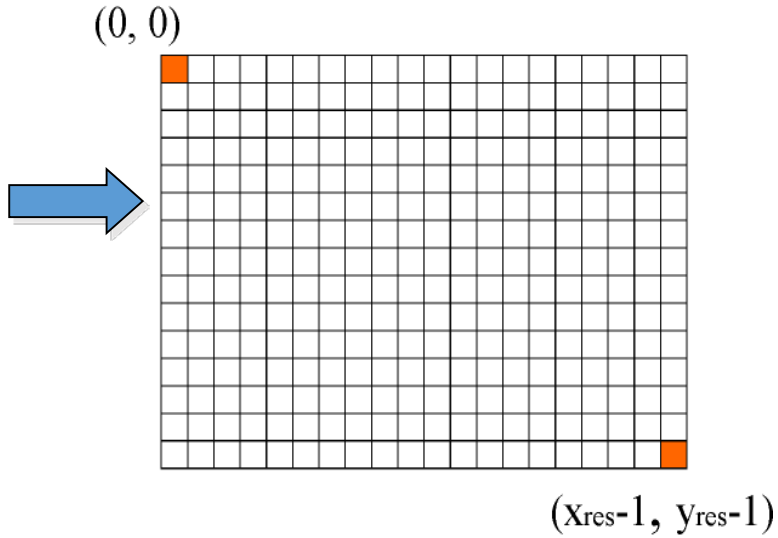


Eye Ray Construction

We need to construct rays originating at the eye and passing through points P_i on the viewport.



viewport



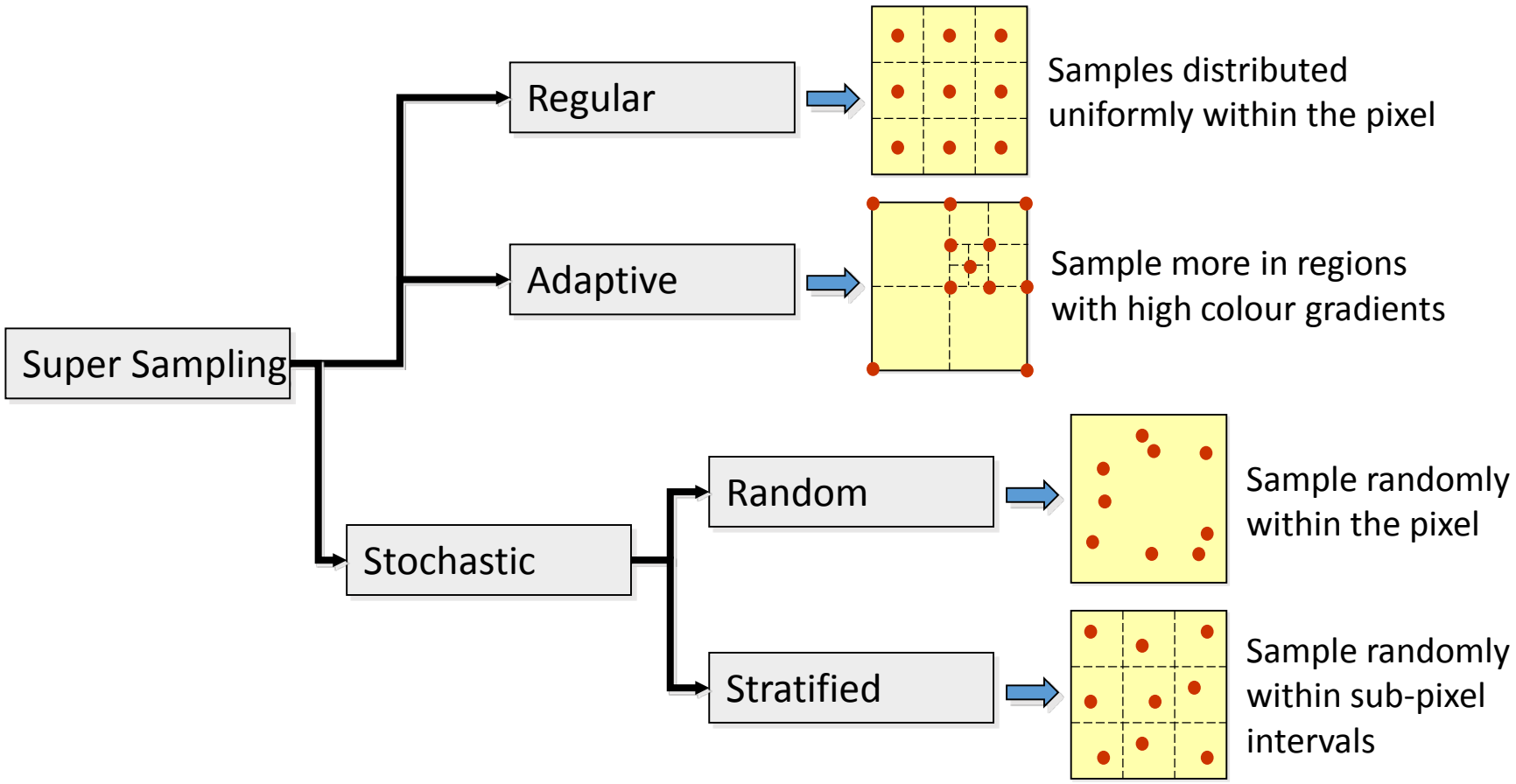
window

$$\alpha = \frac{2(x + 0 \cdot 5)}{x_{res}} - 1 \quad \beta = 1 - \frac{2(y + 0 \cdot 5)}{y_{res}}$$

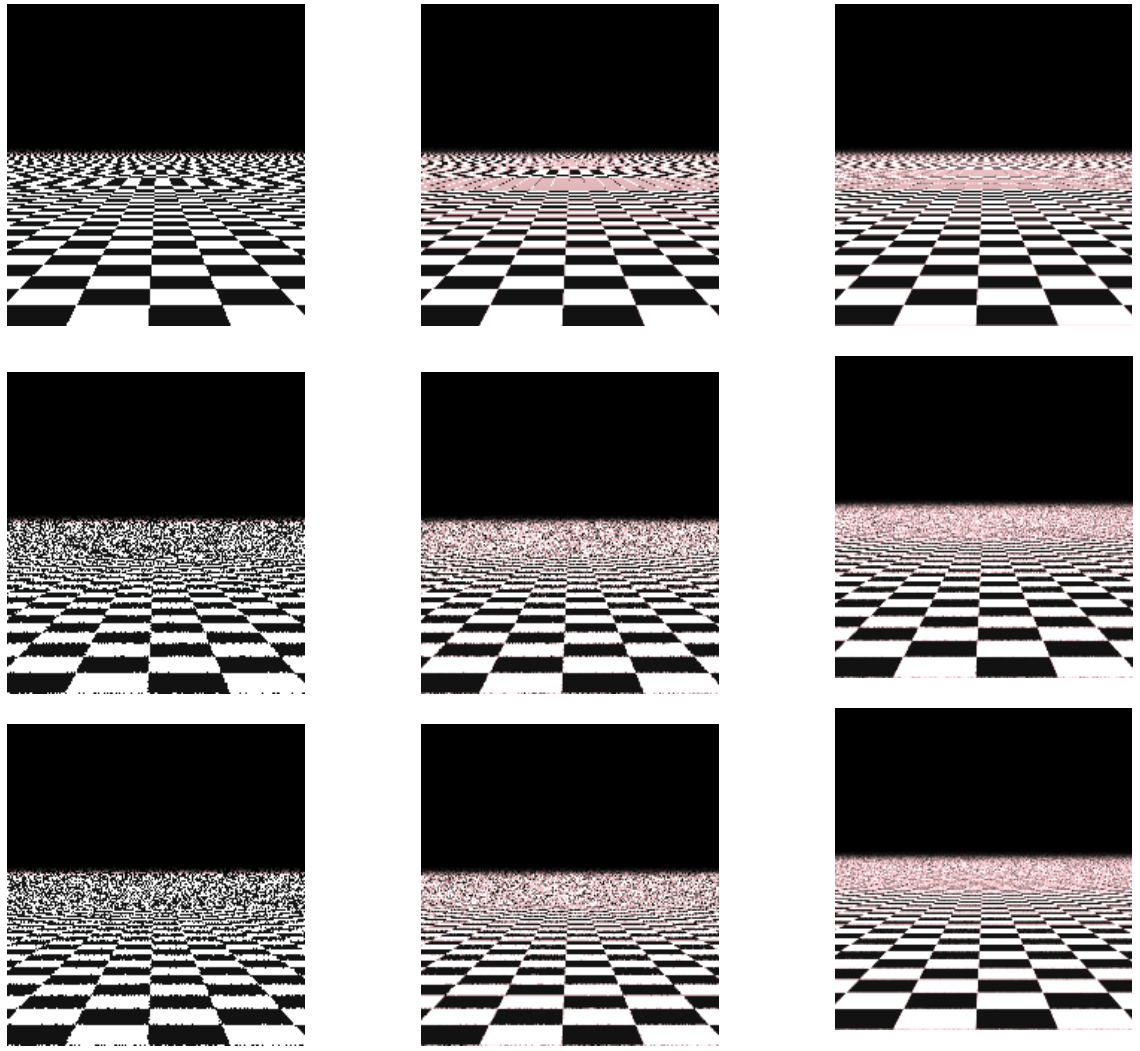
- Pixel center used for ray direction.
- Y value is inverted

Super Sampling Schemes

One ray per pixel is usually not sufficient
Send *n* rays per pixel and filter the result (usually take a weighted average):

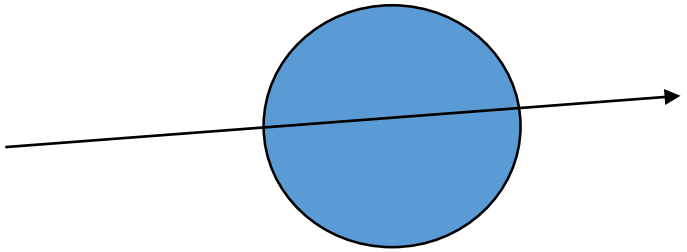


Super Sampling

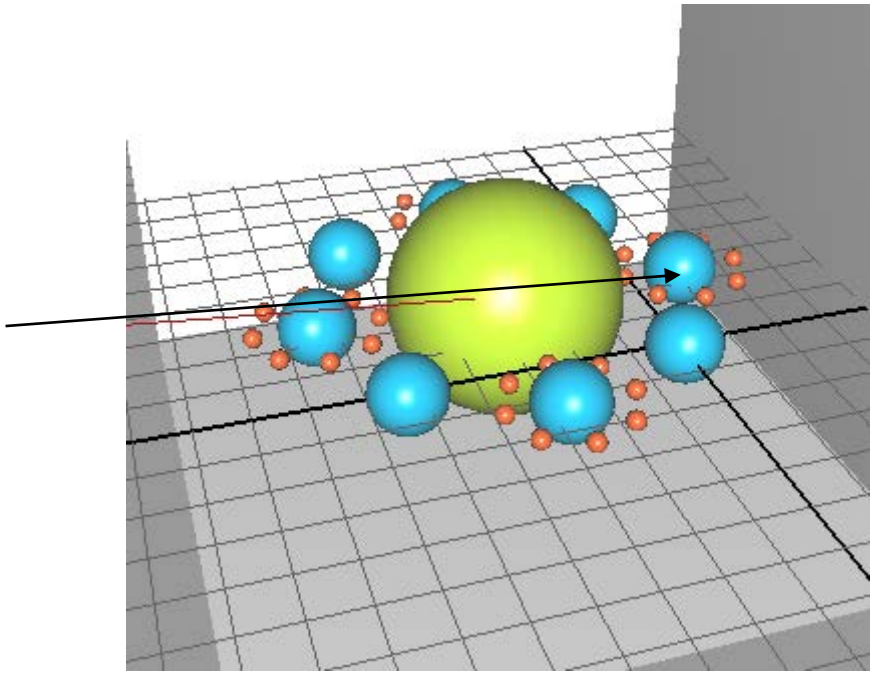


Object Intersection

- Intersection testing is solved mathematically. But is an inherently expensive geometrical problem. 90% of computation time.

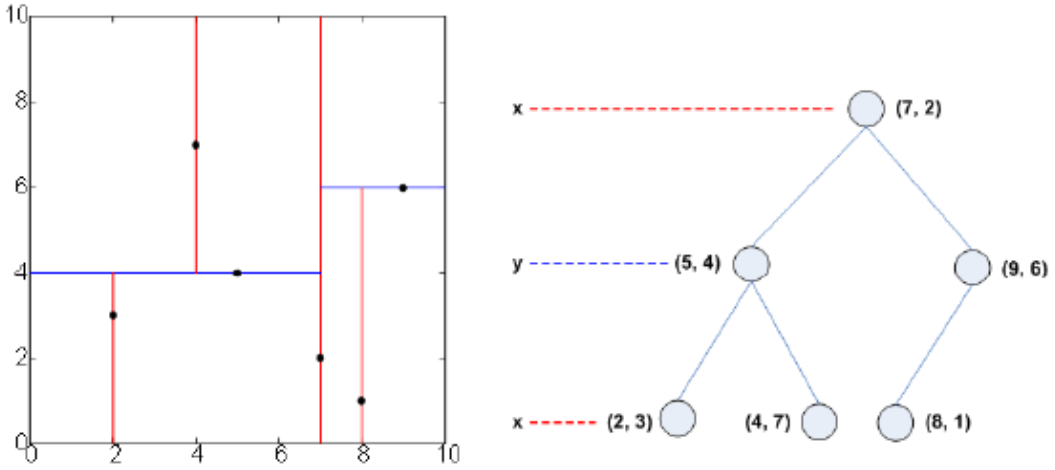


Where does ray intersect object?

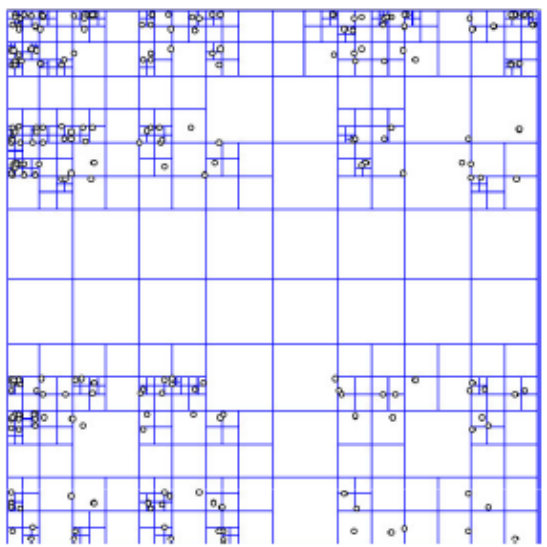


Where does ray intersect scene?

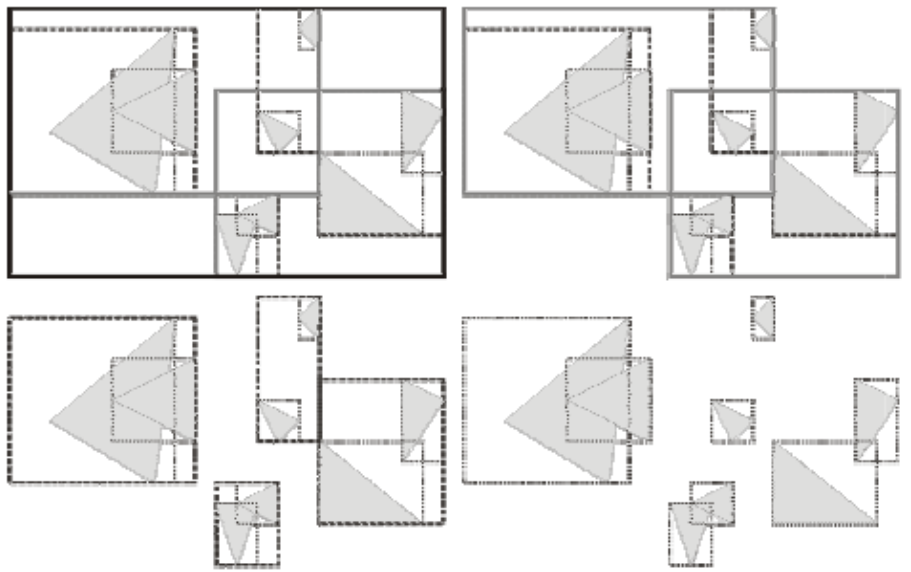
Acceleration Strategies



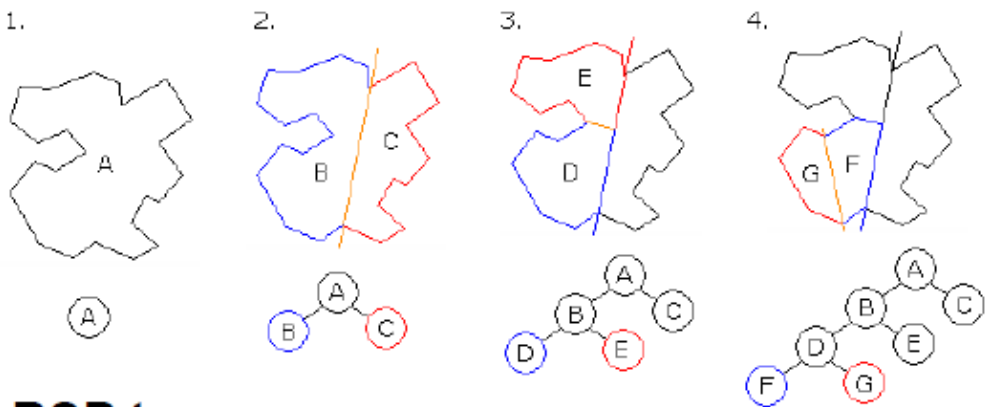
***k*d-tree**



Quadtree / Octree



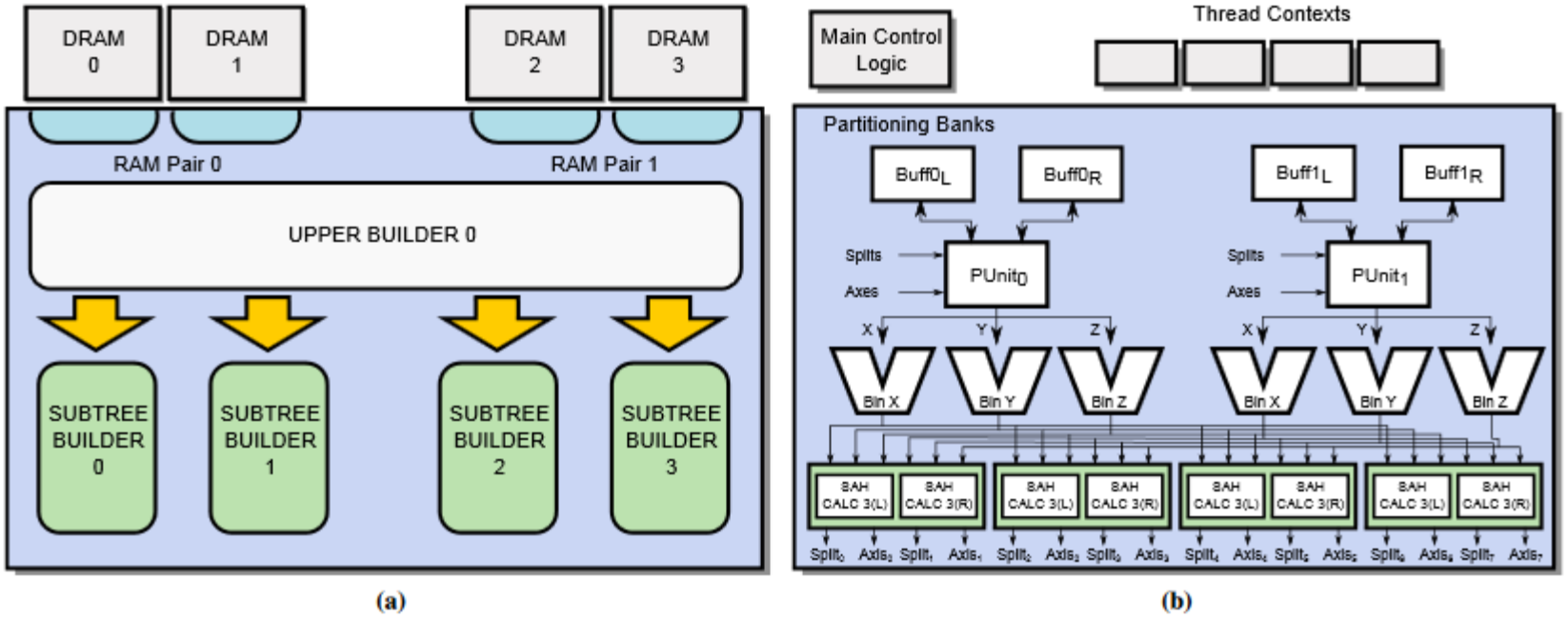
Bounding volume hierarchy



BSP tree

A hardware unit for fast SAH-optimised BVH construction

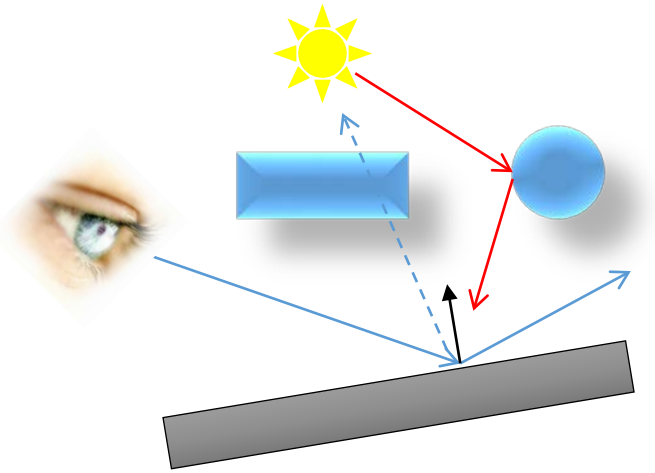
- Specialised microarchitecture
- Construction of binned SAH BVHs



Michael J. Doyle, Colin Fowler, and Michael Manzke. 2013. A hardware unit for fast SAH-optimised BVH construction. *ACM Trans. Graph.* 32, 4, Article 139 (July 2013), 10 pages. DOI: https://doi.org/10.1145/2461912.2462025

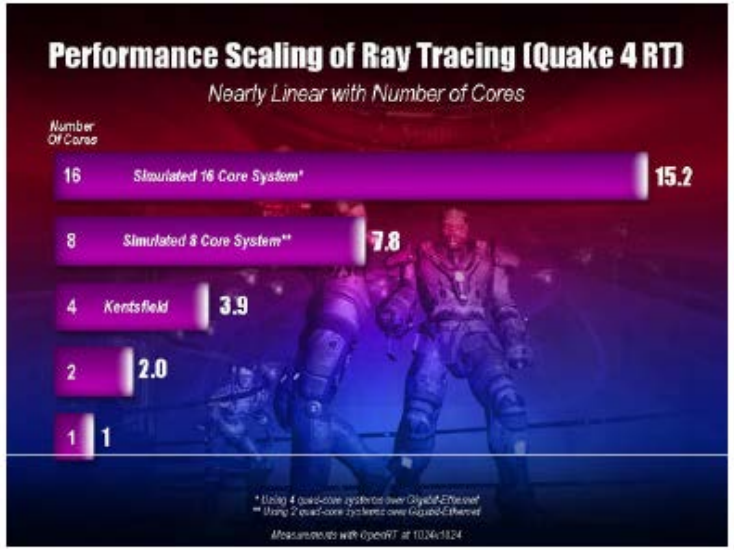
Ray-tracing Limitations

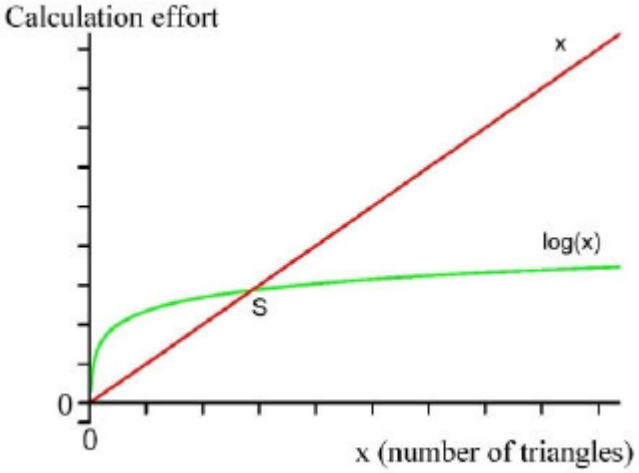
- Some global effects not fully possible
  - Correct shadows & caustics of refractive objects
  - Indirect illumination from reflective objects
  - Diffuse global reflection – colour bleeding
- Slow (well we get better at it)

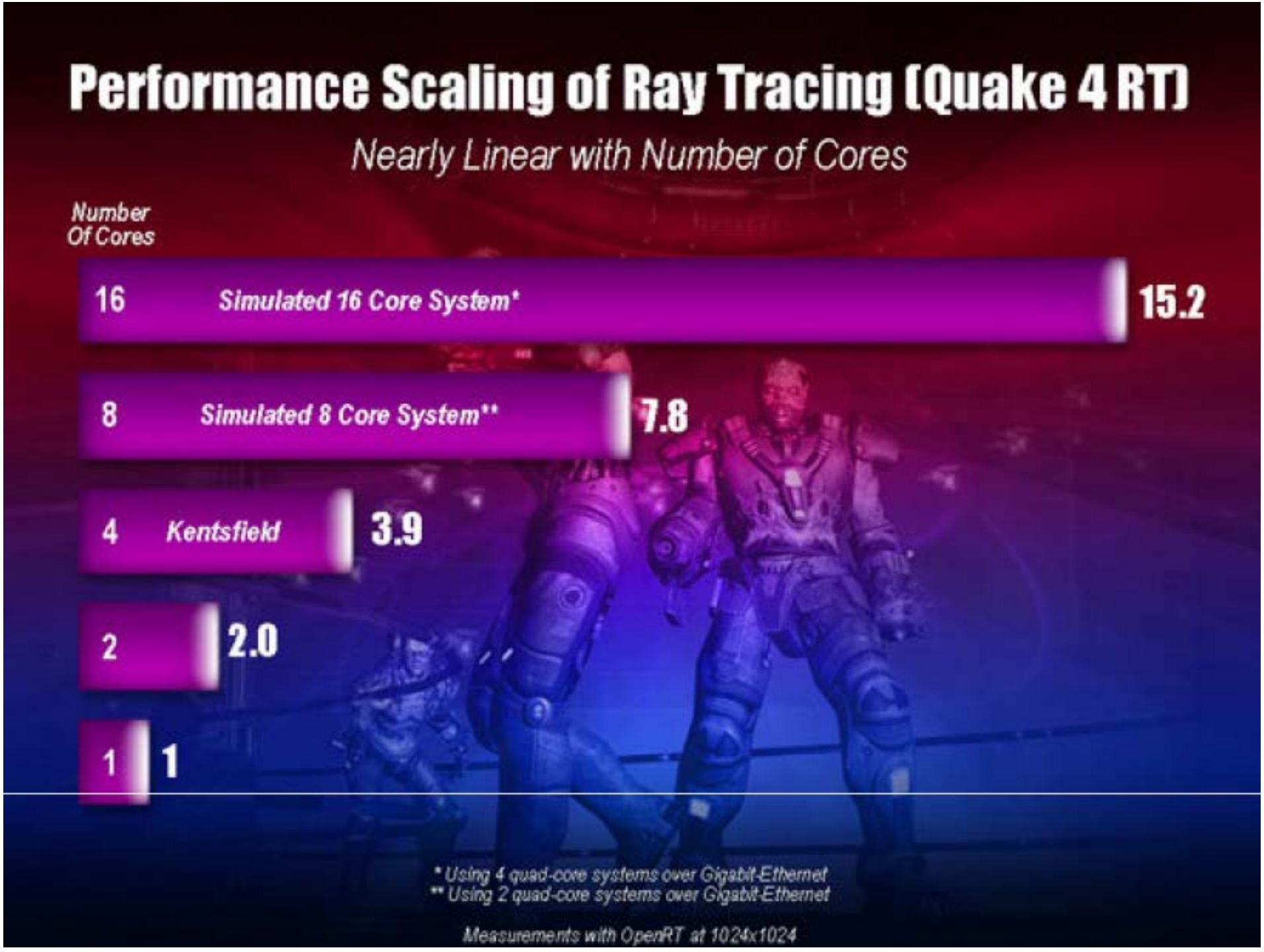


Realtime Ray-tracing

- Advantages:
  - Scales well on multi-core
  - Accurate shadows
  - Pixel accurate reflection, refraction
  - Simple portals
  - Scales logarithmically with geometry complexity
  - Trivial instancing
  - Photo-realism is a solved problem largely using ray tracing





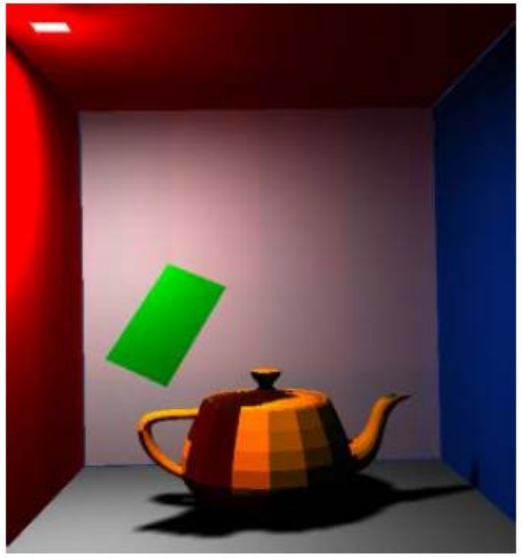

Performance Scaling of Ray Tracing (Quake 4 RT)
Nearly Linear with Number of Cores
Number Of Cores
16
Simulated 16 Core System*
15.2
8
Simulated 8 Core System**
7.8
4
Kentsfield
3.9
2
2.0
1
1
* Using 4 quad-core systems over Gigabit-Ethernet
** Using 2 quad-core systems over Gigabit-Ethernet
Measurements with OpenRT at 1024x1024

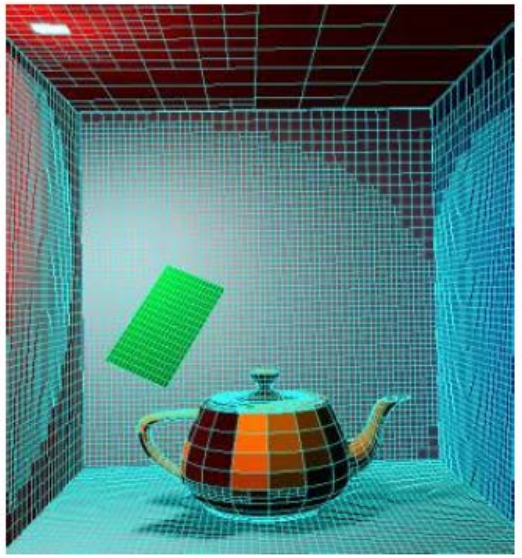
Recent Advances

- Embree ray tracing kernels: overview and new features
  - Attila T. Áfra, Ingo Wald, Carsten Benthin, and Sven Woop. 2016. Embree ray tracing kernels: overview and new features. In ACM SIGGRAPH 2016 Talks (SIGGRAPH '16). ACM, New York, NY, USA, Article 52, 2 pages. DOI: https://doi.org/10.1145/2897839.2927450
- OptiX: a general purpose ray tracing engine
  - Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. 2010. OptiX: a general purpose ray tracing engine. In *ACM SIGGRAPH 2010 papers* (SIGGRAPH '10), Hugues Hoppe (Ed.). ACM, New York, NY, USA, Article 66, 13 pages. DOI: https://doi.org/10.1145/1833349.1778803

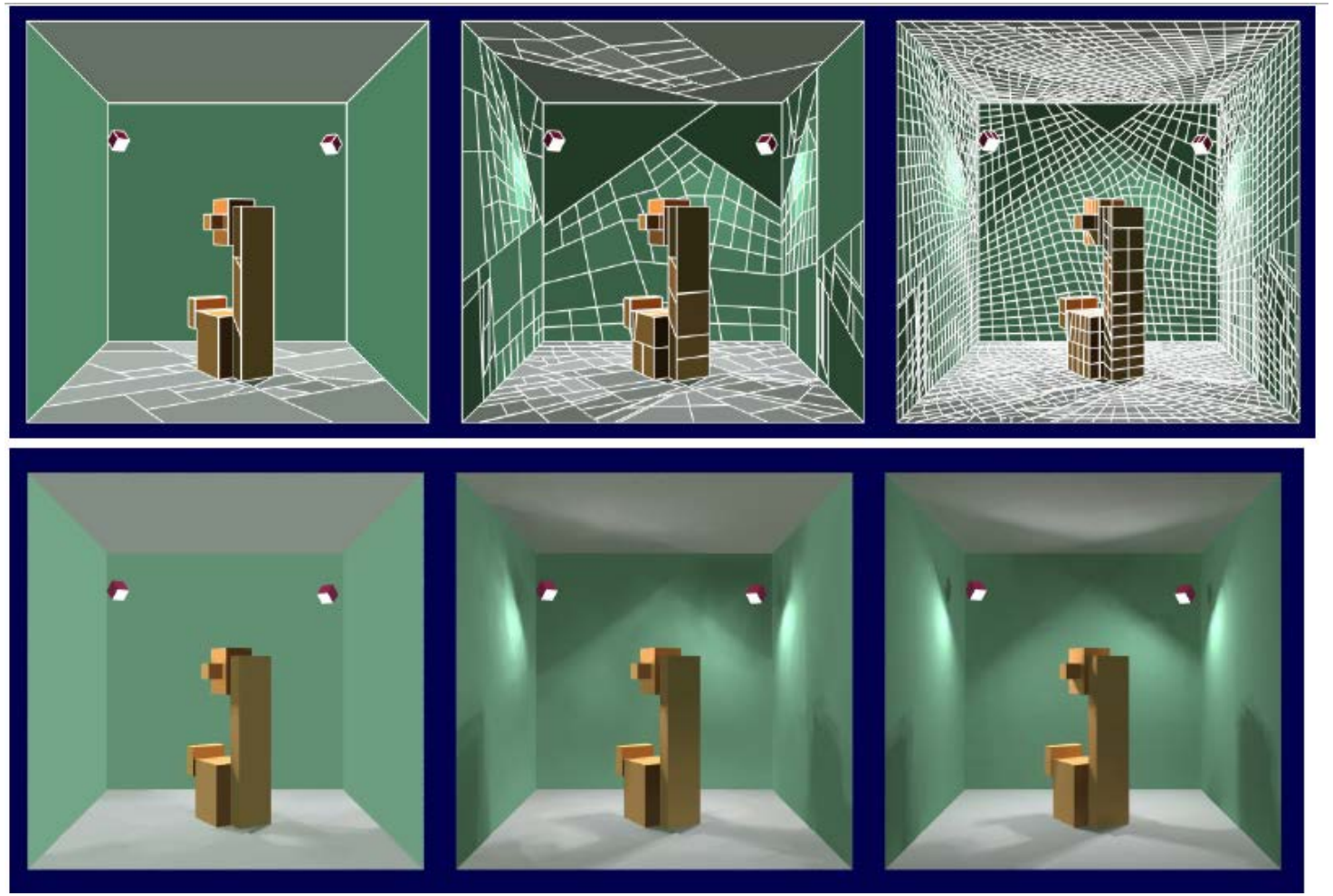
Radiosity

- Radiance transfer modelled on patches of the scene (finite element discretization of the scene)
  - Illumination calculated recursively based transfer of radiant energy between patches
  - Assumption of diffuse light-material interaction





Progressive Radiosity

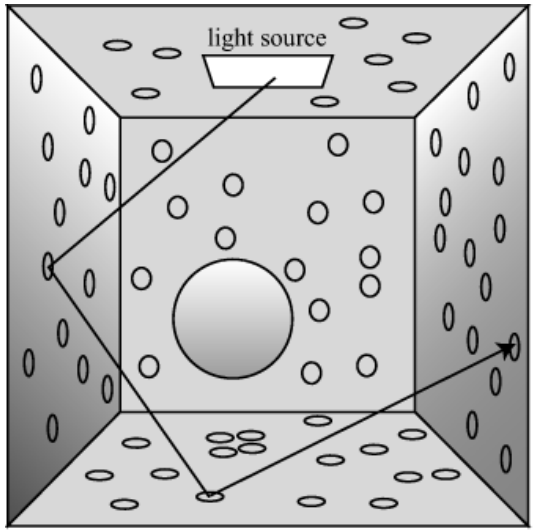


Radiosity

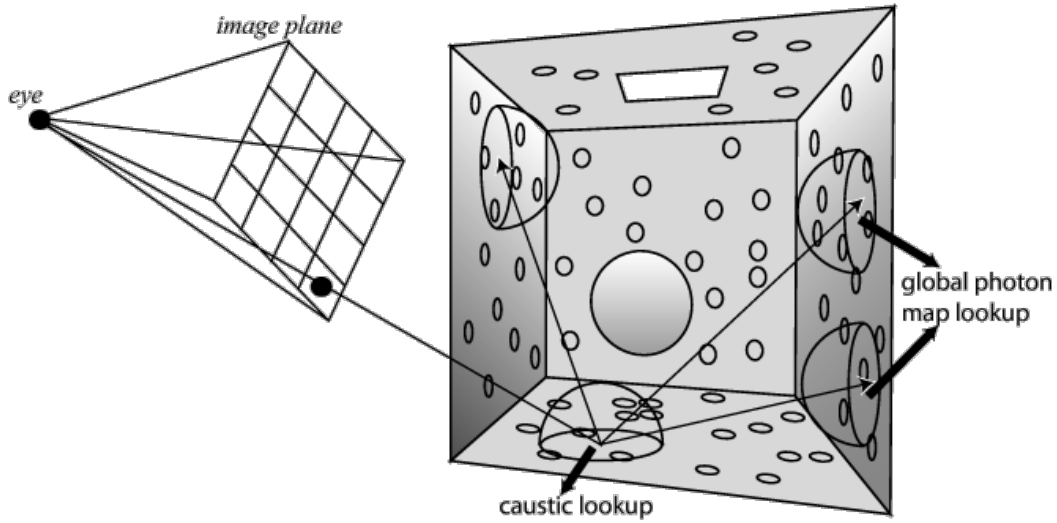
- Provides soft shadows, color bleeding and indirect illumination for free;
- However,
  - does not handle specular reflection,
  - has difficulty in processing transparency (i.e., reflection and refraction),
  - requires the scene to be subdivided
  - a second pass (e.g., ray tracing) is needed to produce reflection and refraction.
  - is very time consuming.
- View independent – useful for pre-process

Photon Mapping

- Two-pass global illumination technique [Jensen'96]
  1. Use photons to simulate transport of individual photon energy. Pre-trace from lights to build the Photon Map
     - N.B. Photon Map is not an image
  2. Trace paths from viewer to see where this is picked up



Pass 1: Shoot Photons

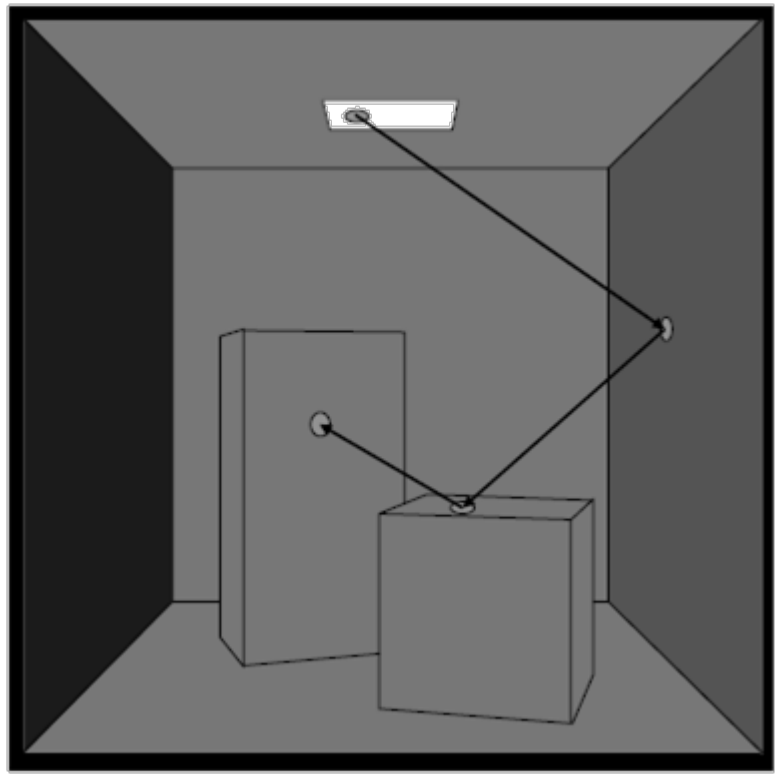


Pass 2: Find Nearest Neighbors

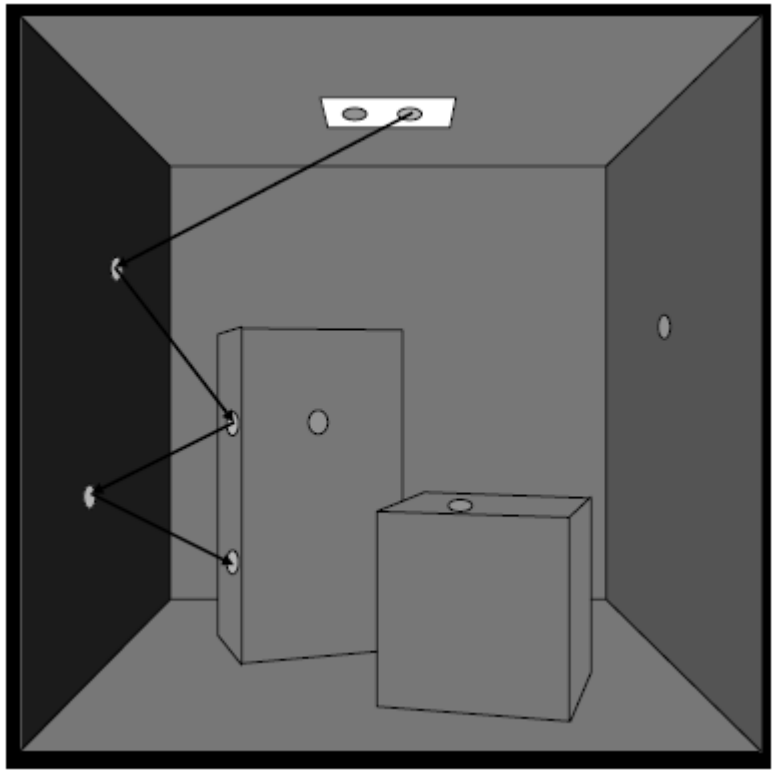
Photon Mapping

- Accounts for
  - color bleeding,
  - soft shadows,
  - indirect illumination,
  - scattering by participating media
  - caustic effects
- Works with arbitrary geometry
- Low memory consumption
- Correct rendering results (although can introduce noise)

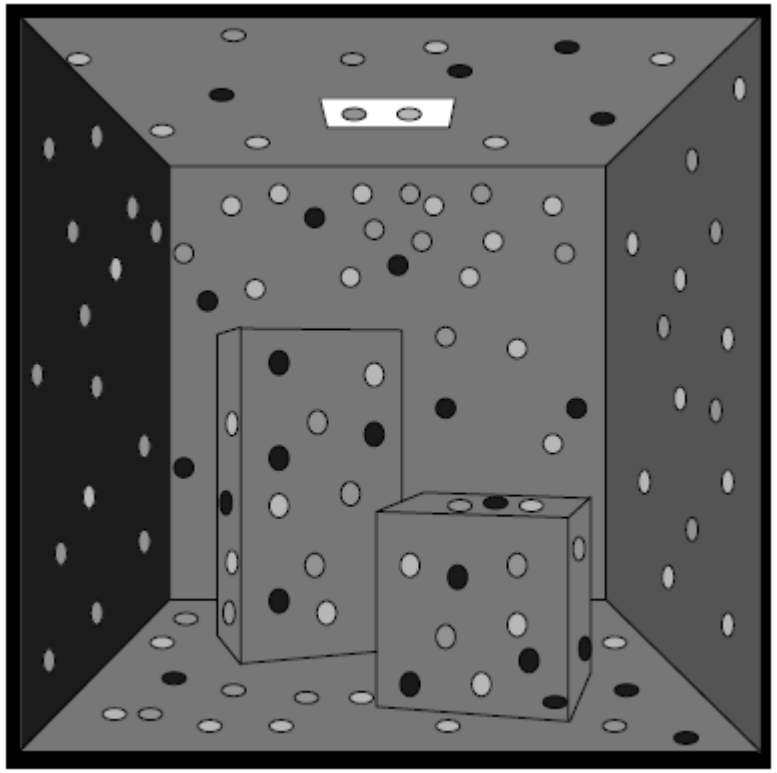
First Pass



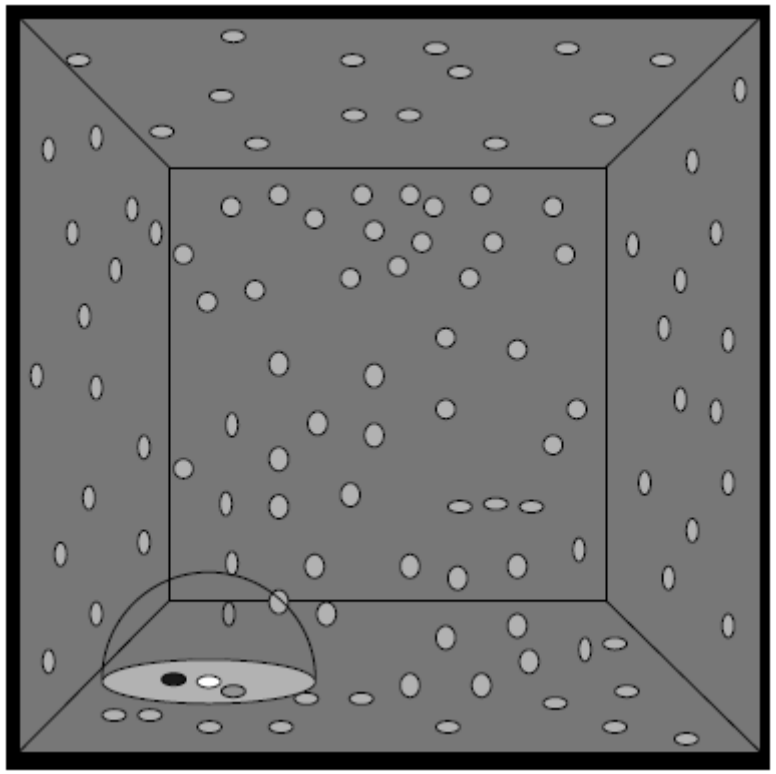
First Pass



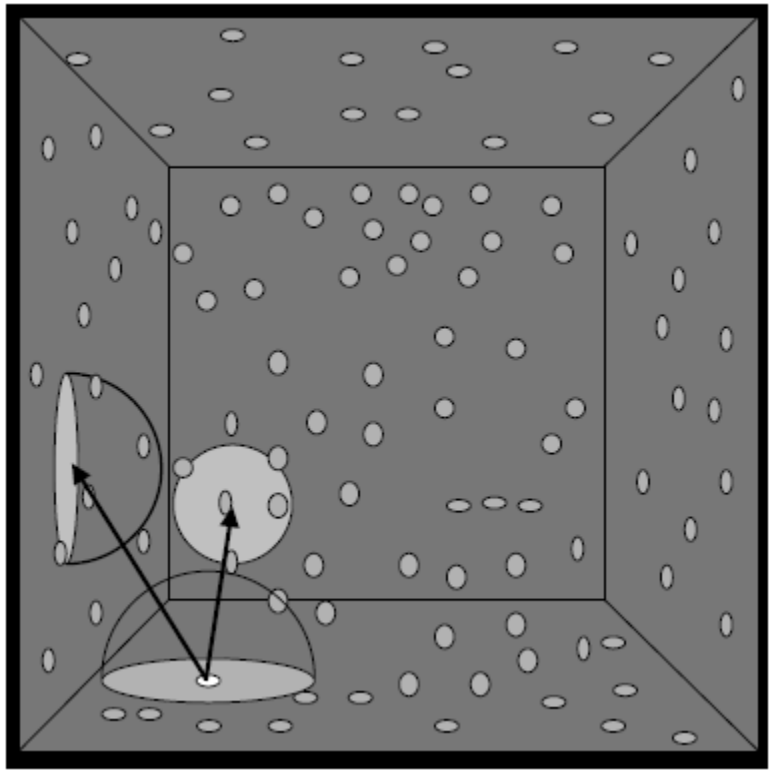
First Pass



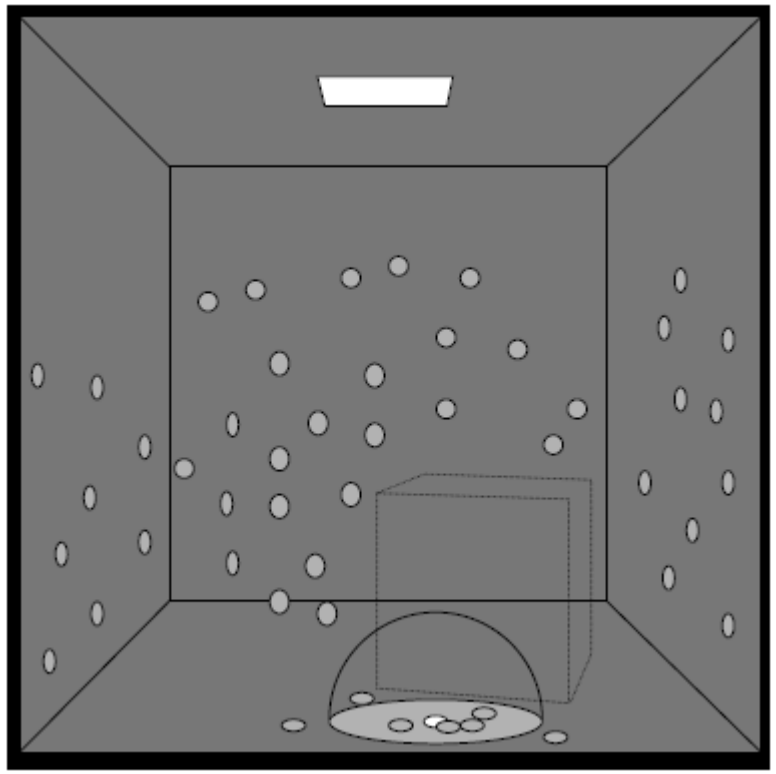
Second Pass

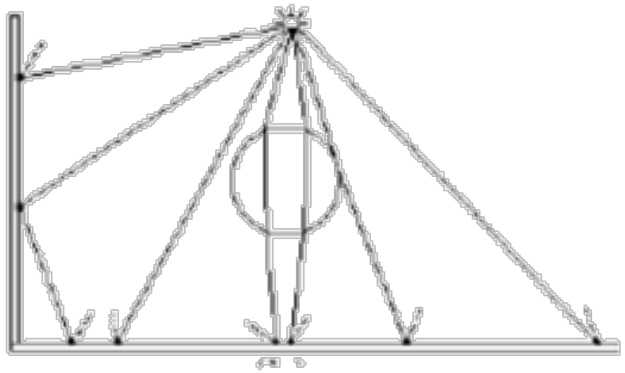


Second Pass

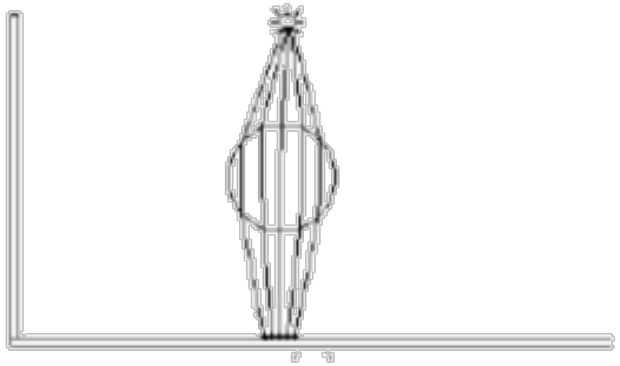


Second Pass





Global photon map



Caustic photon map

Photon Mapping Refs

- Jensen's PhD published as a book: Jensen, Henrik W., Realistic Image Synthesis Using Photon Mapping, A K Peters, Ltd., Massachusetts, 2001
- Seminal Paper: Jensen, Henrik W., Global Illumination using Photon Maps, in Rendering Techniques 1996 (7th Eurographics Symposium on Rendering)
  - http://sites.fas.harvard.edu/~cs278/papers/pmap.pdf
- A summaryby Zack Waters:
  - http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/photon_mapping/PhotonMapping.html
- **Online Interactive applet** with code comparing ray-tracing and photon mapping by Grant Schindler
  - http://www.cc.gatech.edu/~phlosoft/photon/