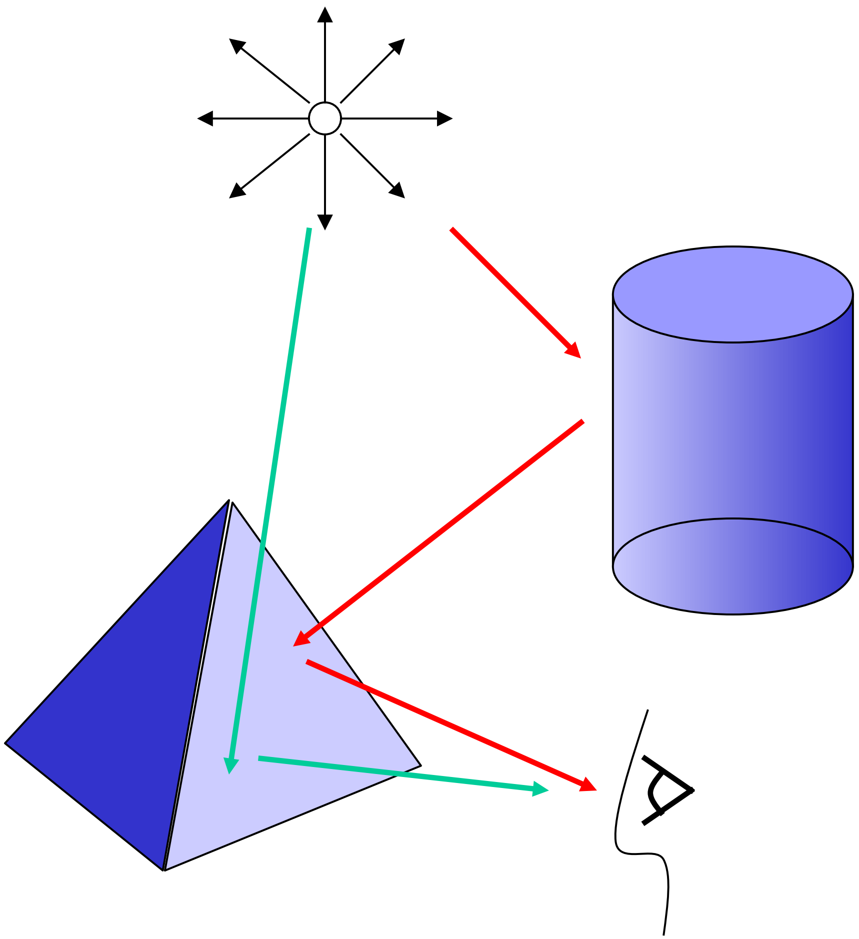
Computer Graphics:
9 - Global Illumination - Raytracing

Prof. Dr. Charles A. Wüthrich,
Fakultät Medien, Medieninformatik
Bauhaus-Universität Weimar
caw AT medien.uni-weimar.de

Global Illumination Models

- Light reflected by a surface is dependent
  - on the surface itself,
  - the direct light sources, and
  - light which is reflected by the other surfaces on the environment towards the current surface (Reflections)
- Note that in local models the third component is modeled through ambient light
- Kajiya introduced an equation describing this



Local vs. Global illumination

- Until now, we have only computed light behaviour as local illumination, except
  - Shadows
  - Environment mapping
- Obviously, the behaviour of light is much richer, and it includes
  - Reflections
  - Refractions
  - More complex effects (fog, colour bleeding…)

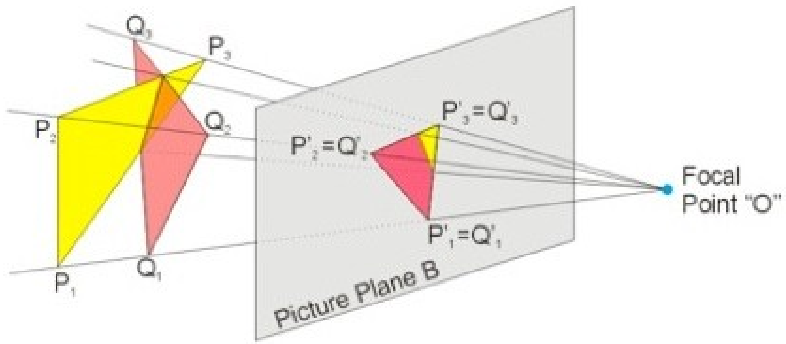
Complex illumination examples

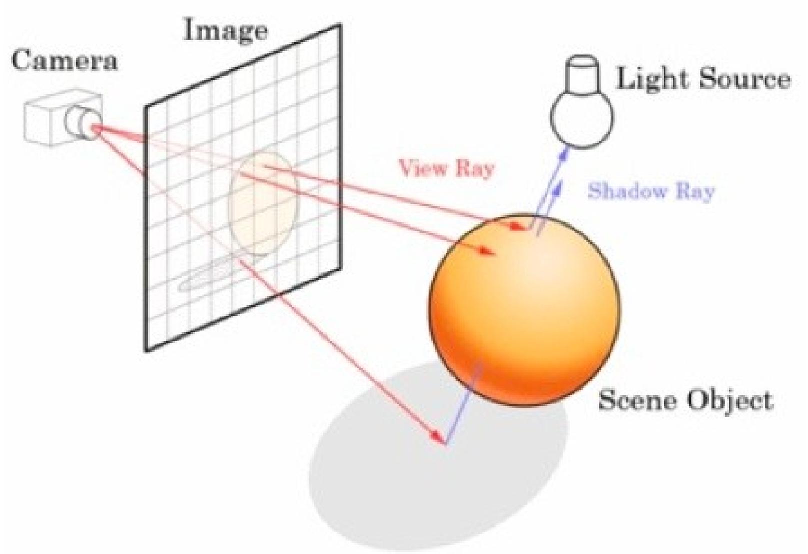
Complex illumination examples

Complex illumination examples

What have we done

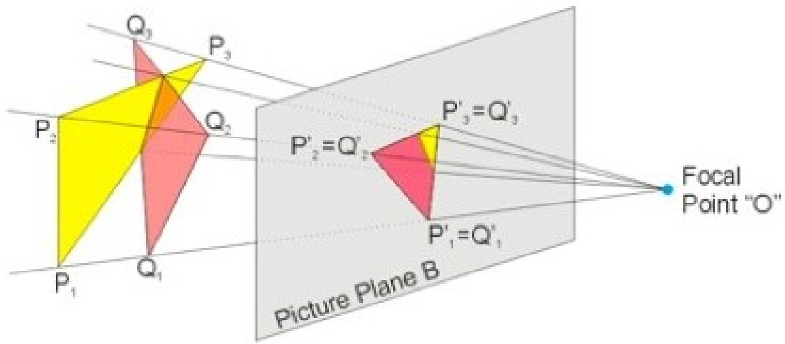
- Until now, we have done the following:
  - Projection,
  - compute hidden surfaces,
  - Add shading,
  - Add shadows
- This we have done starting from the objects in space.

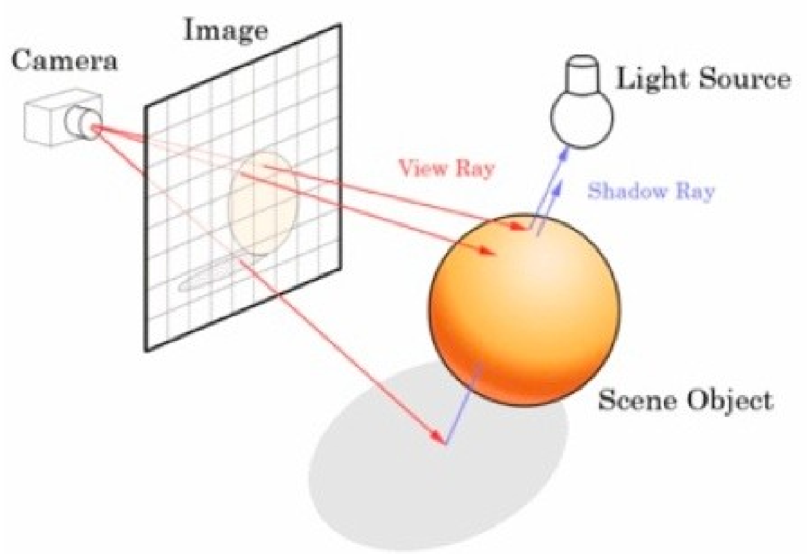




What have we done

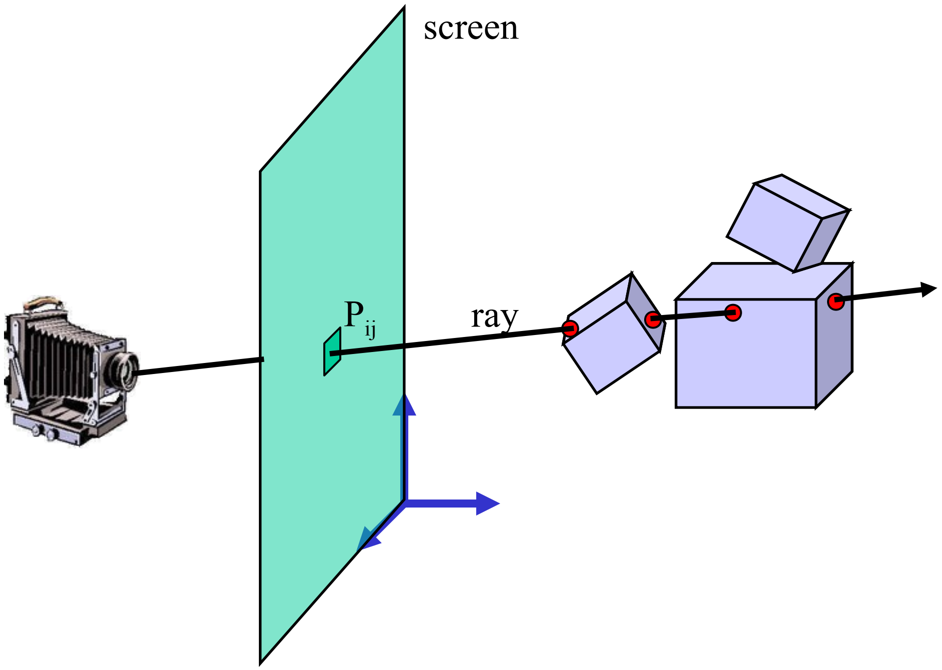
- Until now, we have done the following:
  - Projection,
  - compute hidden surfaces,
  - Add shading,
  - Add shadows
- This we have done starting from the objects in space.
- Why not think at rendering from the point of view of pixels?

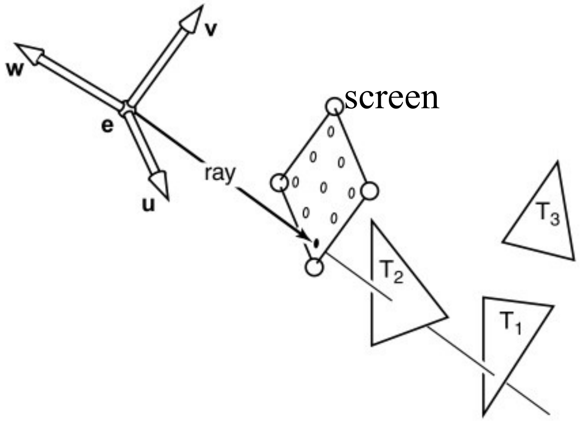




Ray tracing

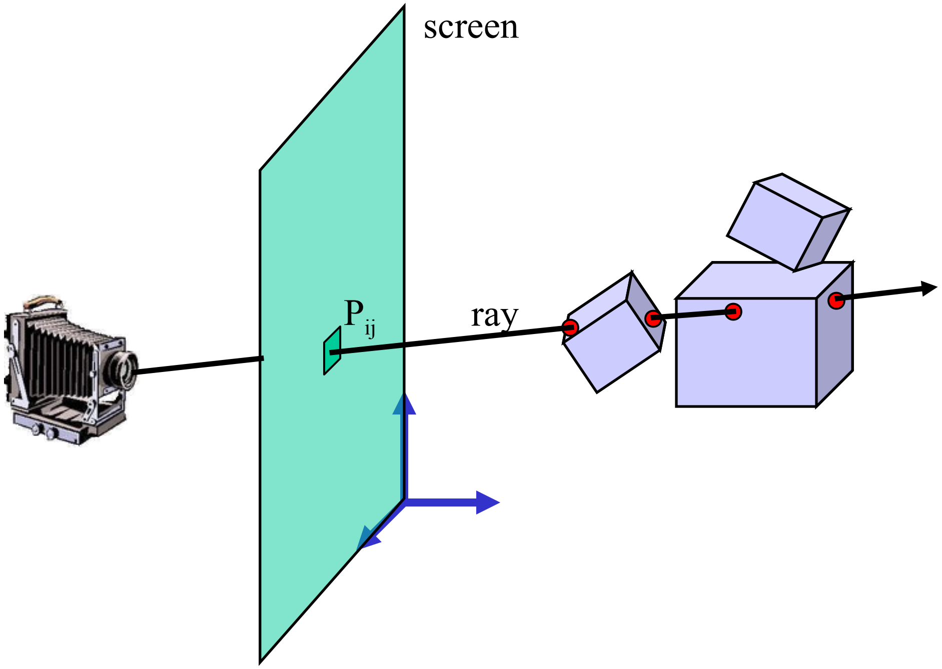
- Let us start thinking:
  - My viewpoint is behind the image plane
  - The image plane is made of pixels
  - What if I shoot a straight line (ray) from the viewpoint through a pixel center into my 3D scene?
  - My ray would intercept objects...

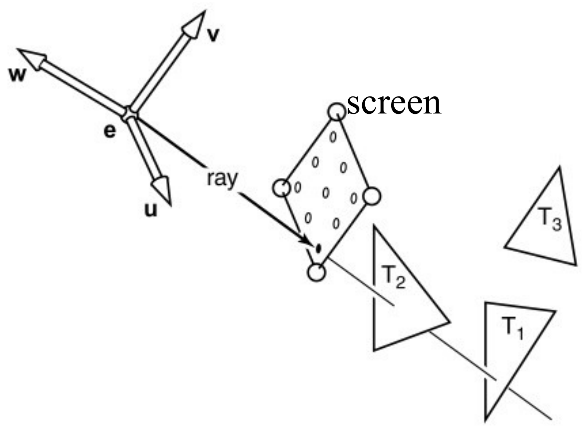




Ray tracing

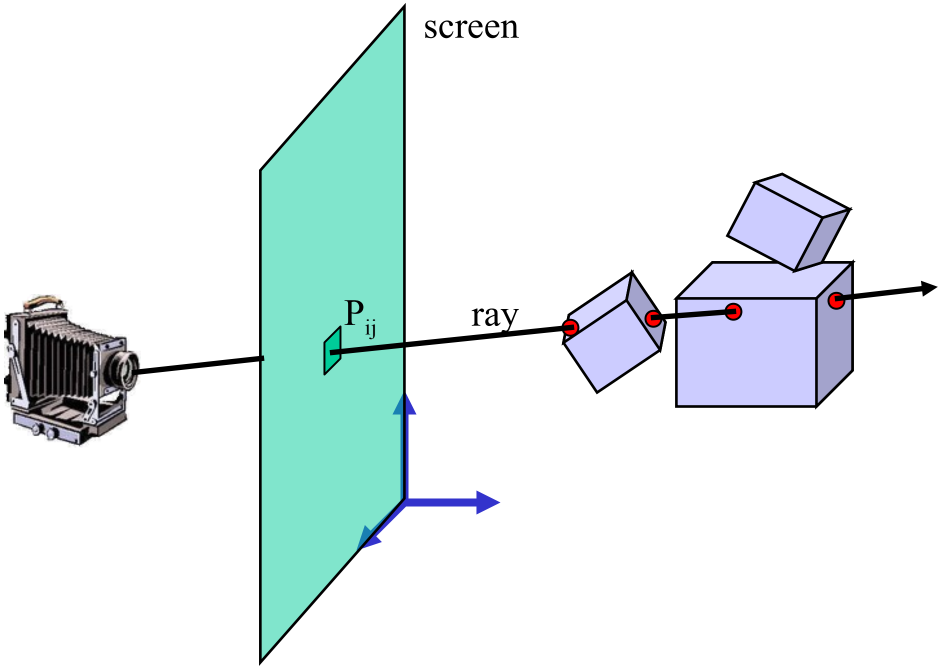
- Let us start thinking:
  - My viewpoint is behind the image plane
  - The image plane is made of pixels
  - What if I shoot a straight line (ray) from the viewpoint through a pixel center into my 3D scene?
  - My ray would meet objects...
  - ... And accumulate light, depending of which objects (polygons) are intercepted...
  - And depending on their light reflection properties (including transparency)

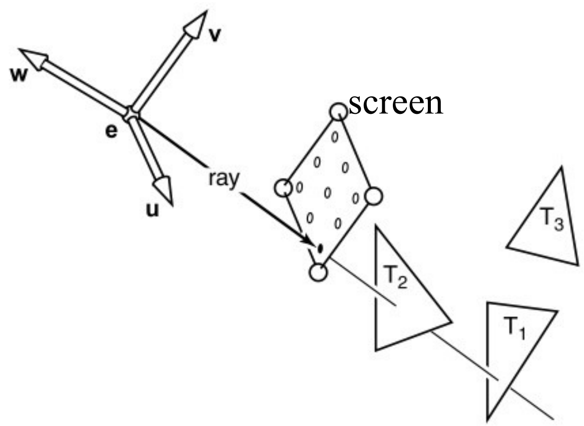




Ray casting

- We cast a ray through the viewpoint and the pixel centers of the screen
- We intersect if with the polygons of the scene
- We sort the polygons intercepted by the ray according to their depth
- We paint the pixel with the color of the closest polygon!

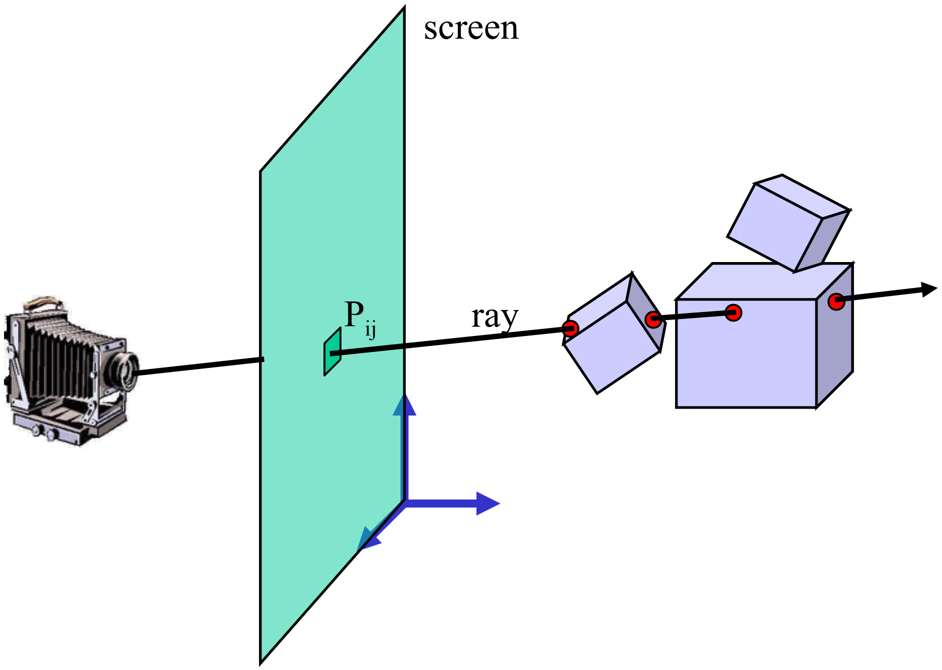


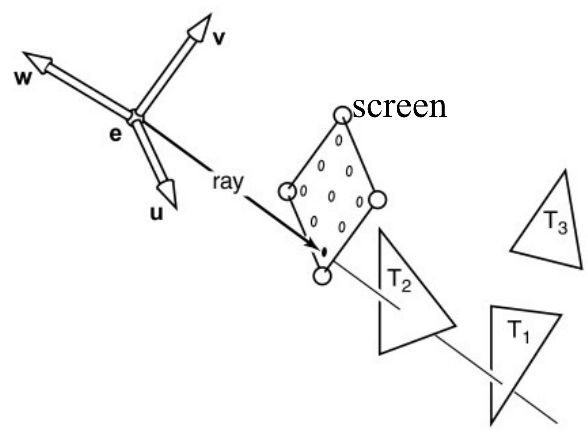


Ray casting

- We cast a ray through the viewpoint and the pixel centers of the screen
- We intersect if with the polygons of the scene
- We sort the polygons intercepted by the ray according to their depth
- We paint the pixel with the color of the closest polygon...

- ... obtaining HIDDEN SURFACE for free!



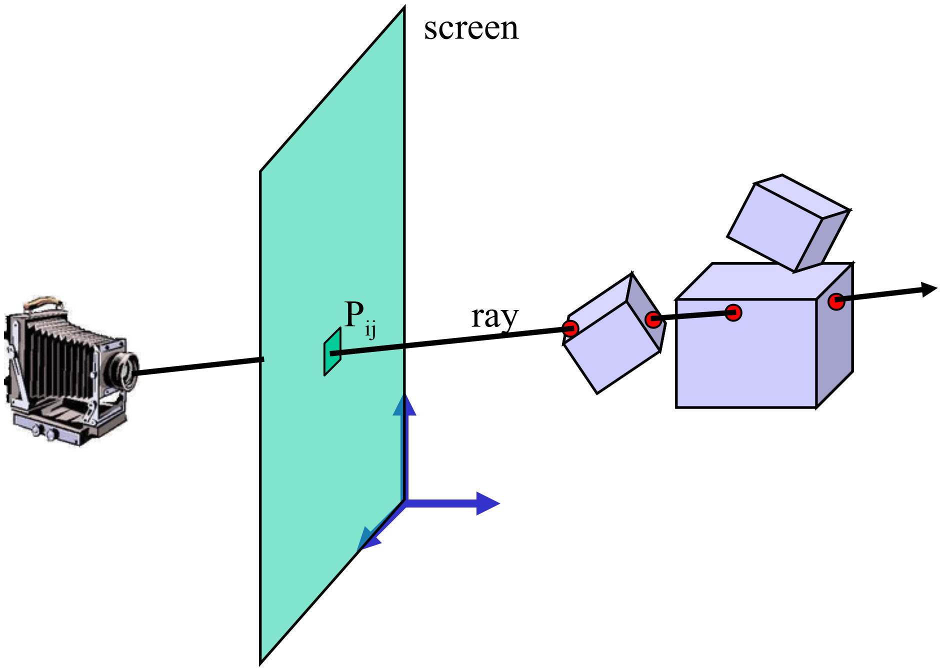


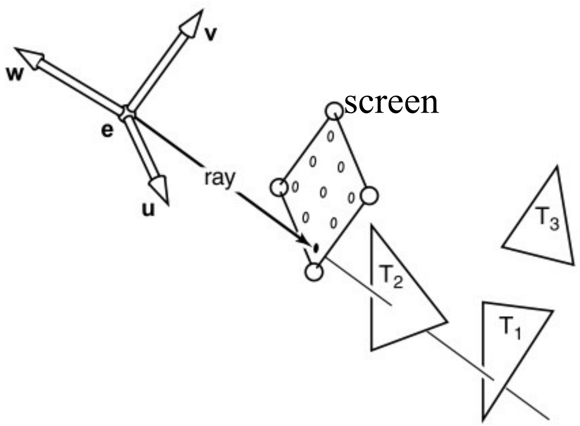
Ray casting

- We cast a ray through the viewpoint and the pixel centers of the screen
- We intersect if with the polygons of the scene
- We sort the polygons intercepted by the ray according to their depth
- We paint the pixel with the color of the closest polygon...

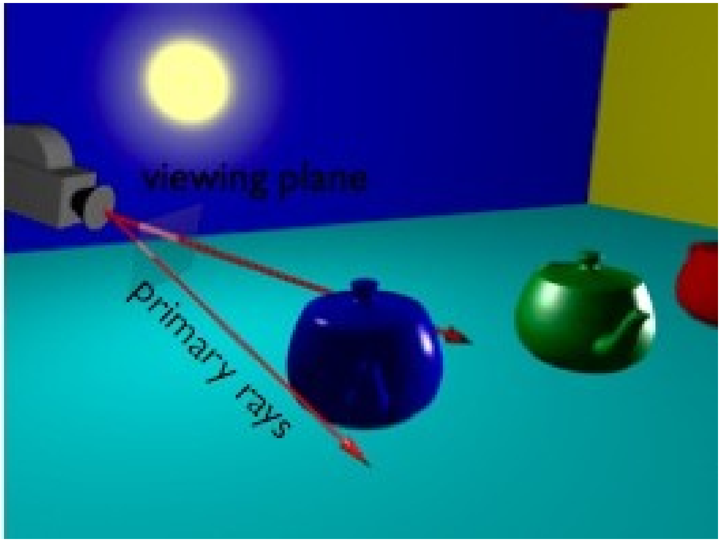
- ...if transparent, we *accumulate* along the ray the light reflection properties of the polys met..

- ...obtaining TRANSPARENCY!

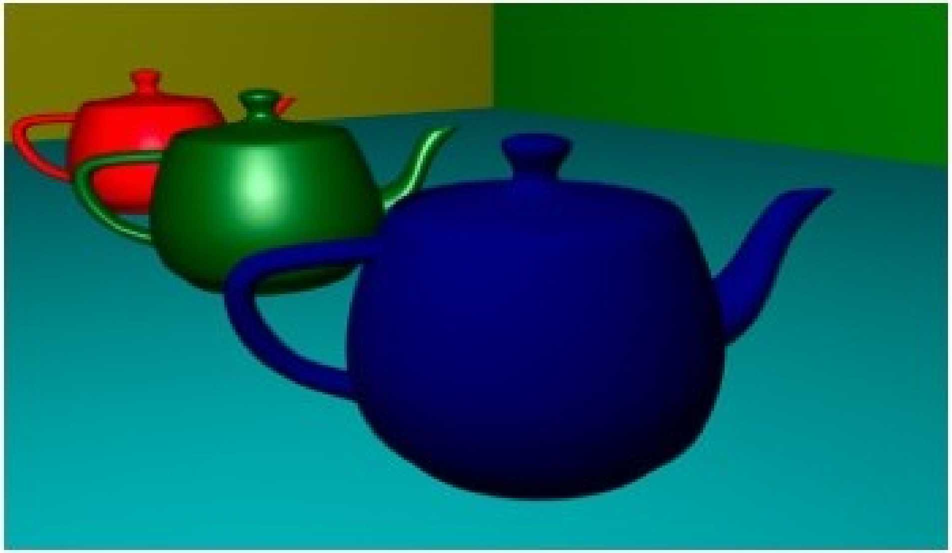




Ray casting



- The rays passing through the screen are called *primary rays.*
- And the method *raycasting* [Appel68]



Ray casting: intersections

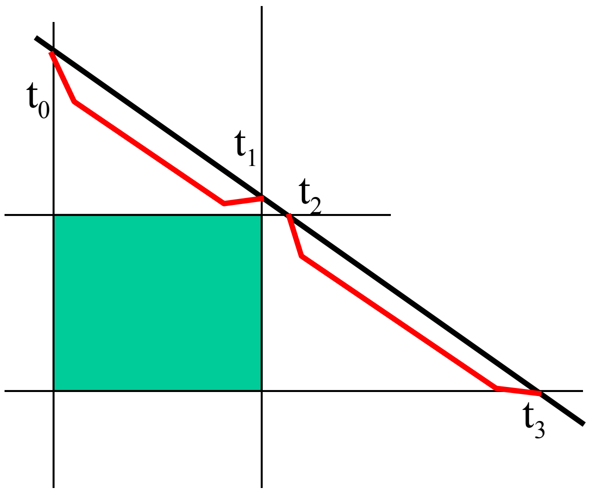
- Equation of line through
  - Viewpoint $V=(x_v,y_v,z_v)$
  - Pixel $P_{ij}=(x_{ij},y_{ij},z_{ij})$:

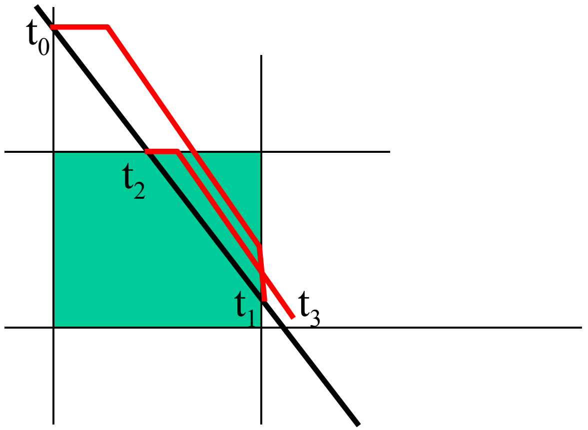
$$r := \begin{cases} x = x_V + t\left(x_{ij} - x_V\right) \\ y = y_V + t\left(y_{ij} - y_V\right) \\ z = z_V + t\left(z_{ij} - z_V\right) \end{cases}$$

- Sphere: substitute into sphere equation and solve system
  - Eq. of sphere with centre (x_C,y_C,z_C) and radius r: $(x-x_C)^2+(y-y_C)^2-(z-z_C)^2=r^2$
  - resulting eq. in t has to be checked for existance of solution

Ray Casting: Intersections

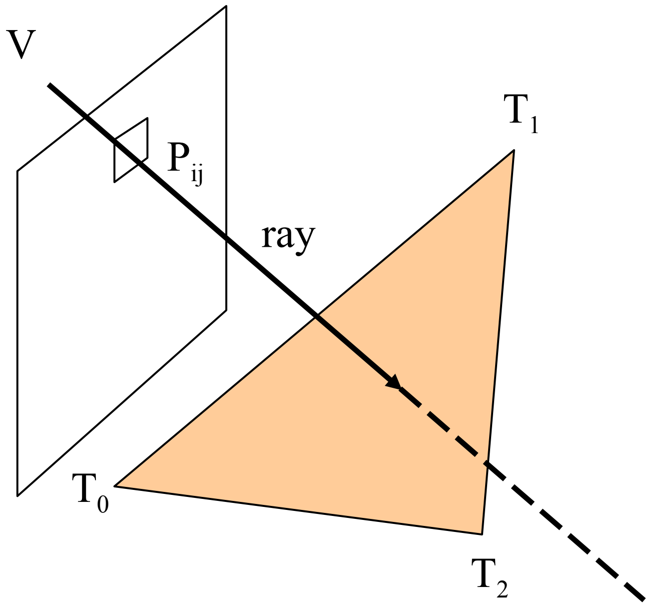
- Boxes (parallel to axes),
  delimited by planes parallel to axes (x=i)
  - Compute intersections with all parallel planes (x,y,z) dir.
  - resolve WRT parameter t
  - analyze intervals and check if they overlap





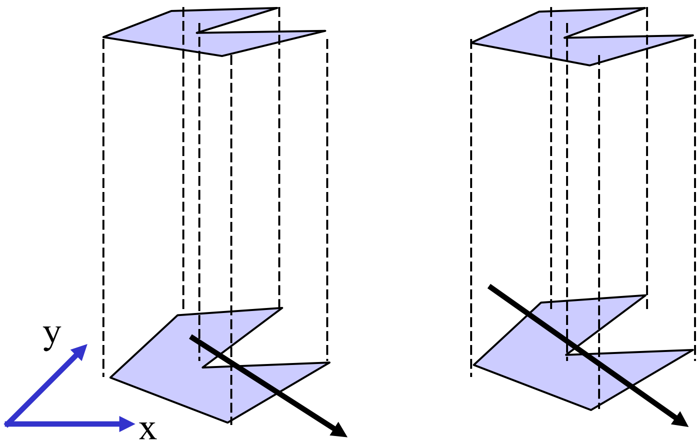
Ray casting: Intersections

- Triangle:
  - My ray passes through the viewpoint and the pixel, so a point P on the ray can be expressed as $P=V+(P_{ij}-V)t$.
  - The triangle points can be viewed in baricentric coordinates, so a point T on the triangle would be
    $T=T_0+\beta(T_1-T_0)+\gamma(T_2-T_0)$
  - By setting equal such equations I compute the intersection point:
    $V+(P_{ij}-V)t=T_0+\beta(T_1-T_0)+\gamma(T_2-T_0)$
  - These are 3 equations in 3 unknowns t, β, γ.



Ray casting: Intersections

- Polygon:
  Project on one major plane
  (check for special cases)
- Use 2D point in polygon:
  - Send ray towards polygon
  - check number of intersections (even or odd)

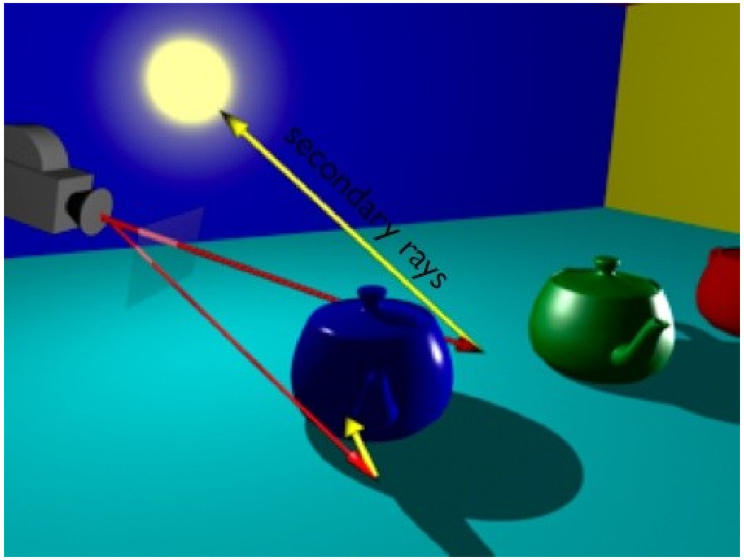


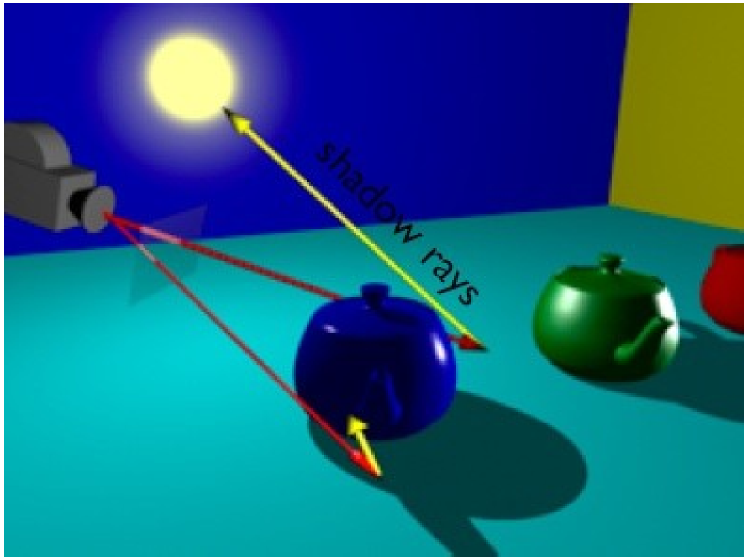
- Quadrics:
  Use their equations and solve against parameter t

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = 0$$

Ray tracing

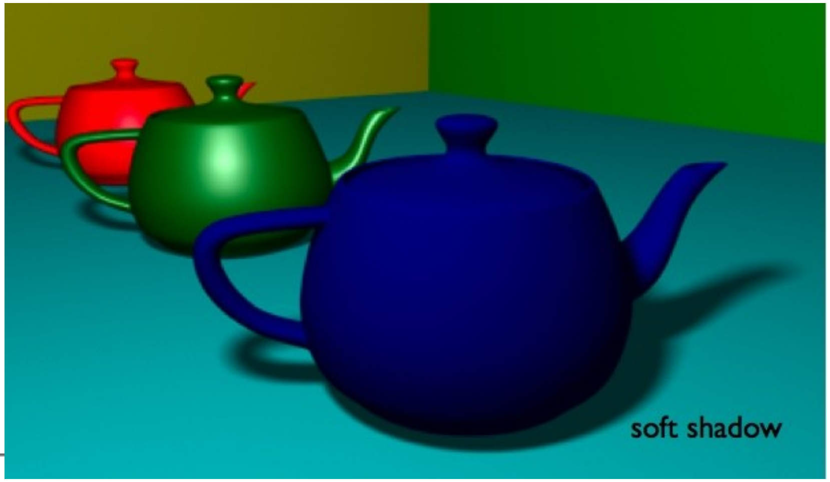
- But now I can add other effects!
  - If I hit with a ray a surface i can lookup if the part being drawn is in shade
  - By shooting a ray from the impact point to the light source I can check if there are objects inbetween, this getting shadows.
  - *Shadows* are almost for free!
  - The rays to the light source are secondary rays
  - They are called shadow rays

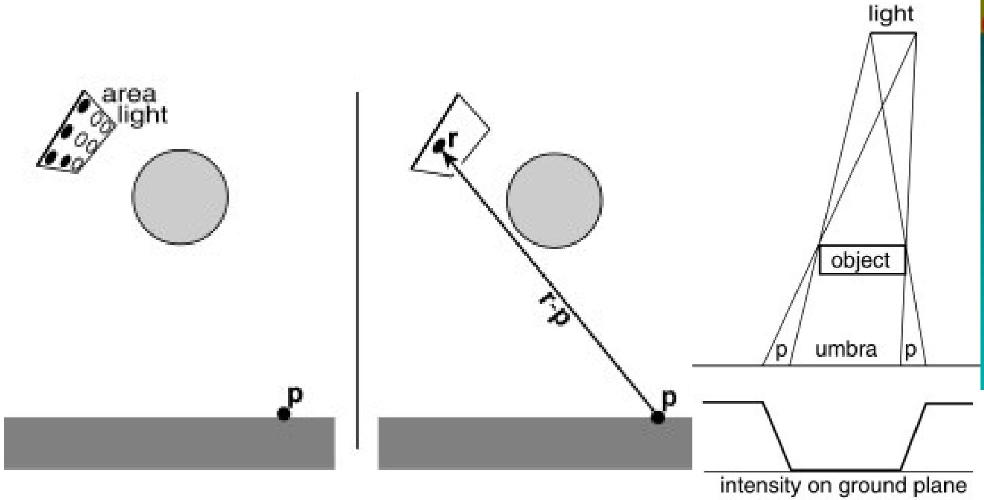




Ray tracing

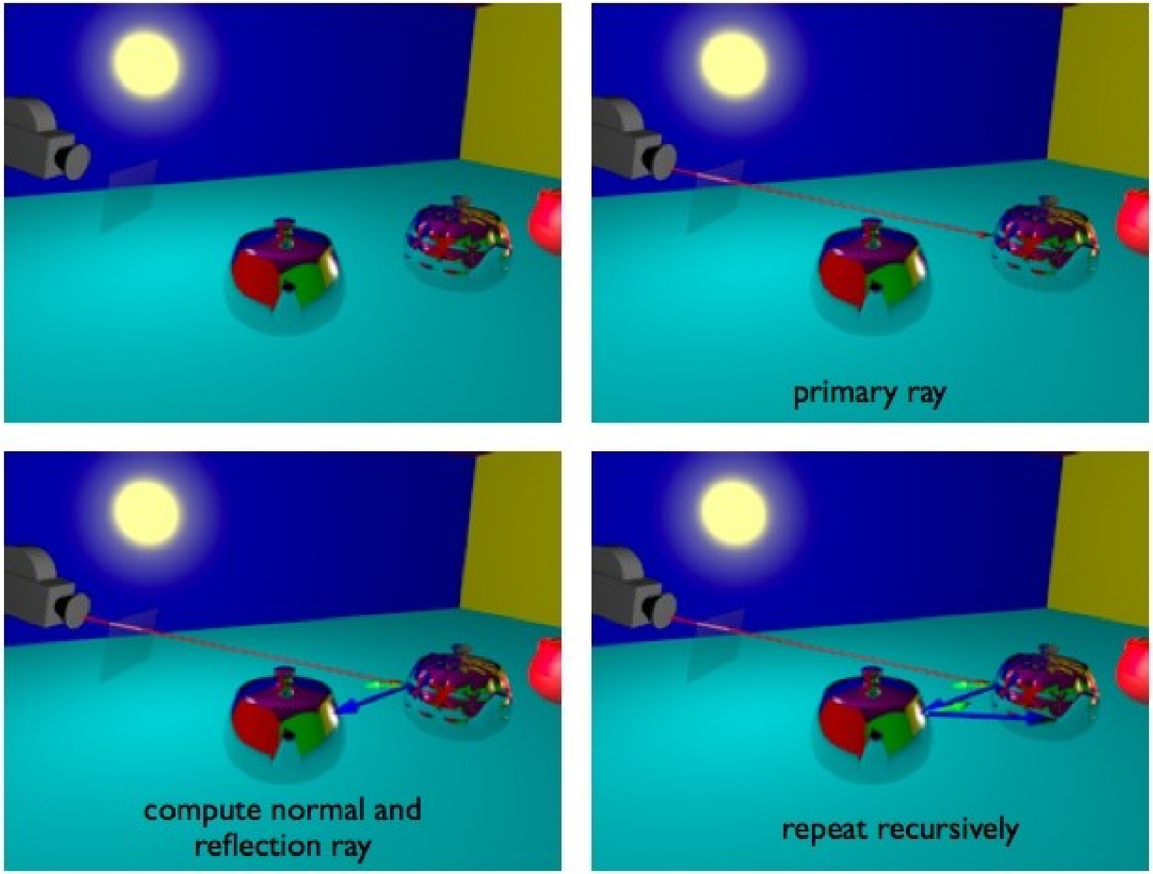
- Shadows can be done in a hard manner or a soft manner (soft shadows)
- In case of area light sources, one interpolates linearly between total occlusion and no occlusion

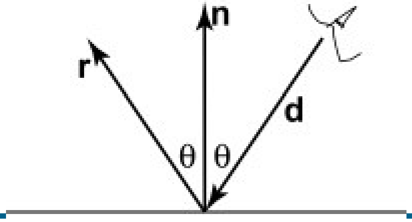




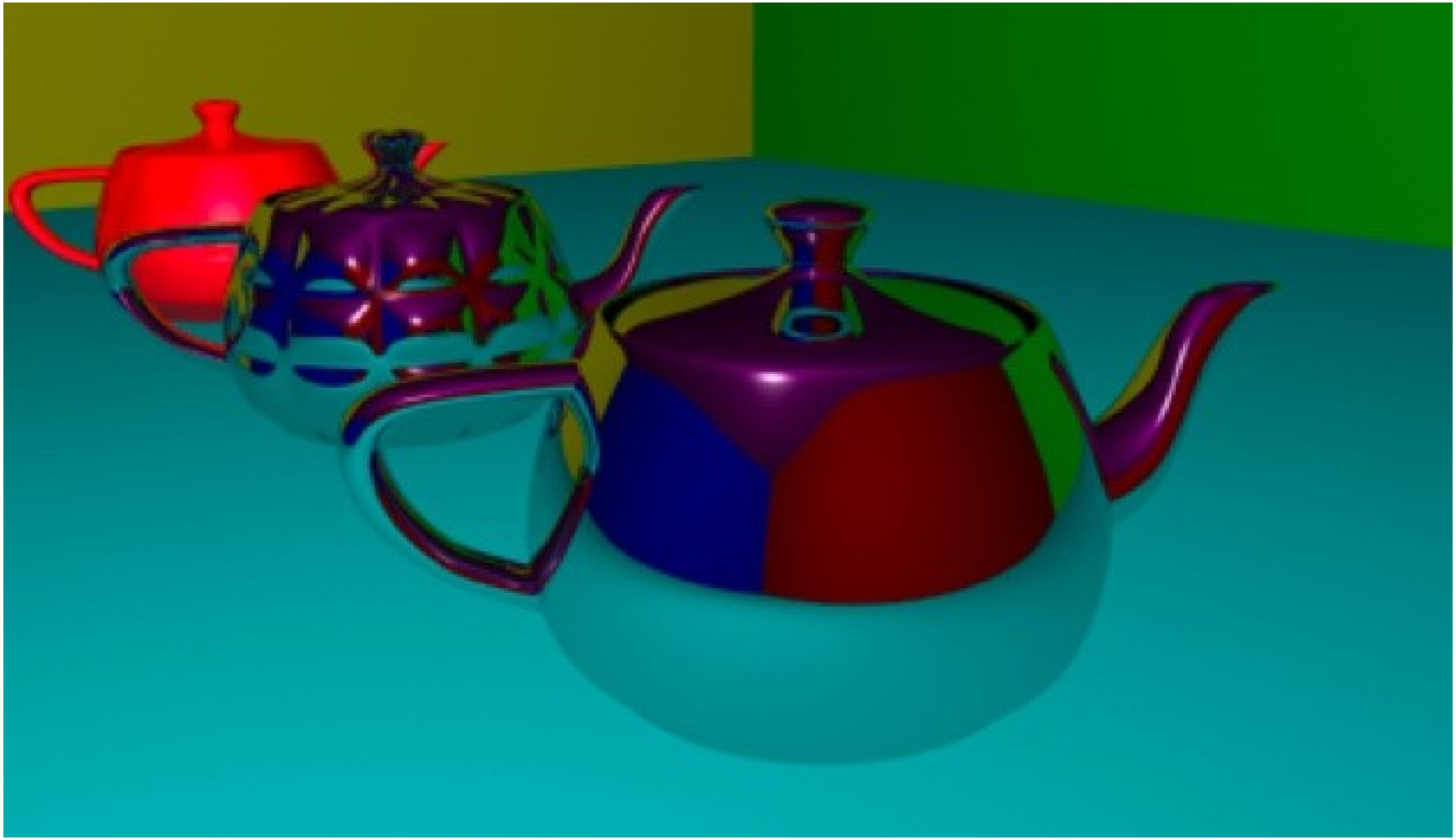
Recursive raytracing: reflections

- When a polygon is hit, reflections can be computed by sending a secondary ray in the environment and computing ist “reflected light contribution” to the color of the pixel.
- Optics laws are used
- By accumulatng recursively, one can simulate multiple reflections



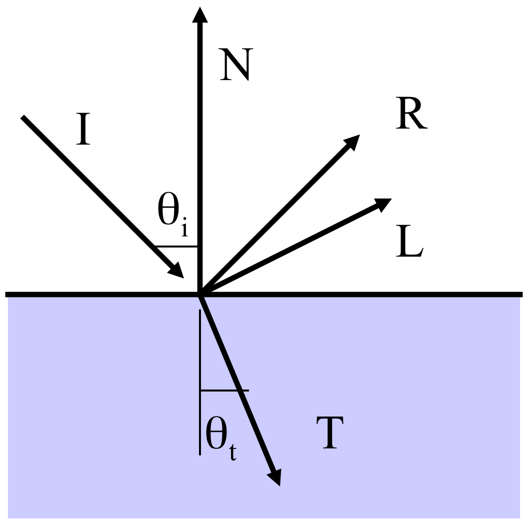


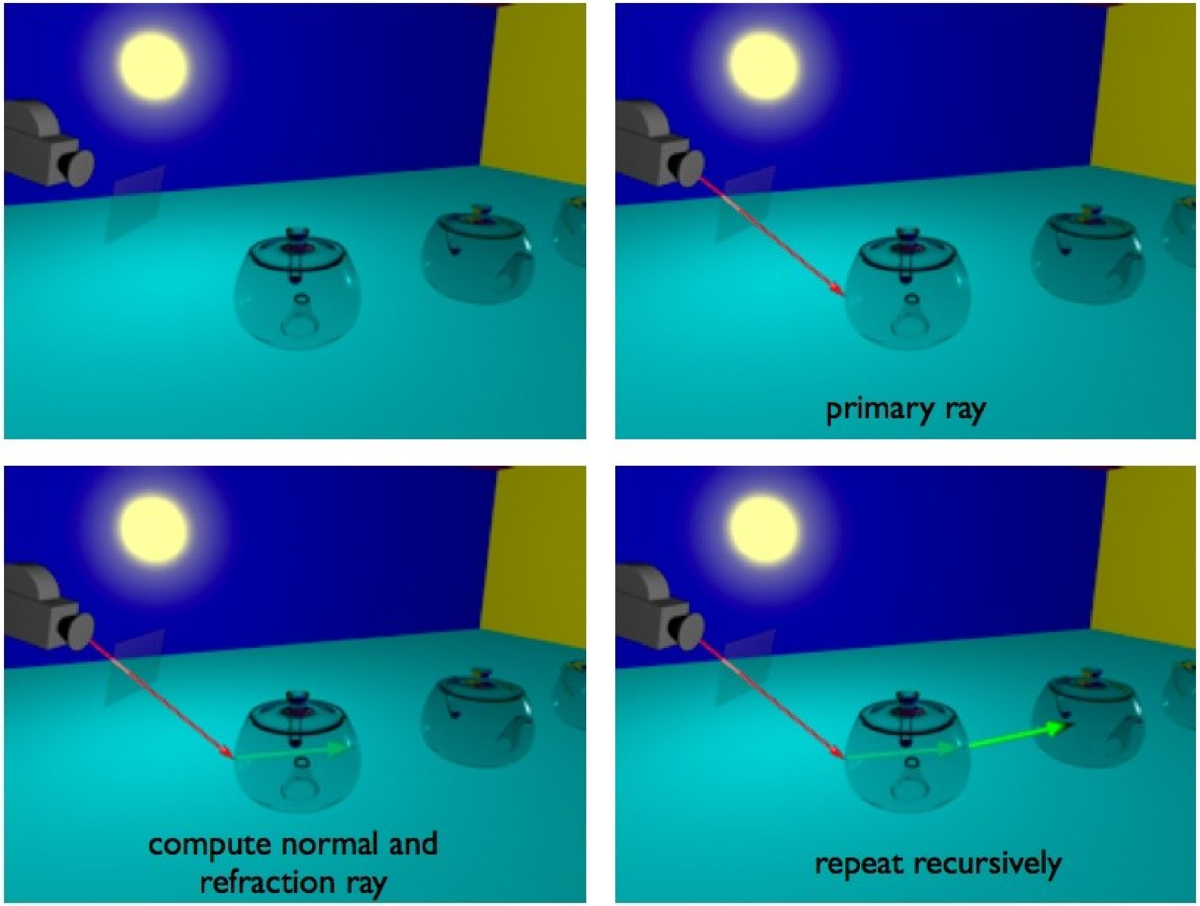
Recursive raytracing: reflections



Recursive raytracing: refractions

- Similarly, one can compute a refraction ray according to Snell's refraction law $\dfrac{\sin\vartheta_i}{\sin\vartheta_t} = \dfrac{\eta_{t\lambda}}{\eta_{i\lambda}}$

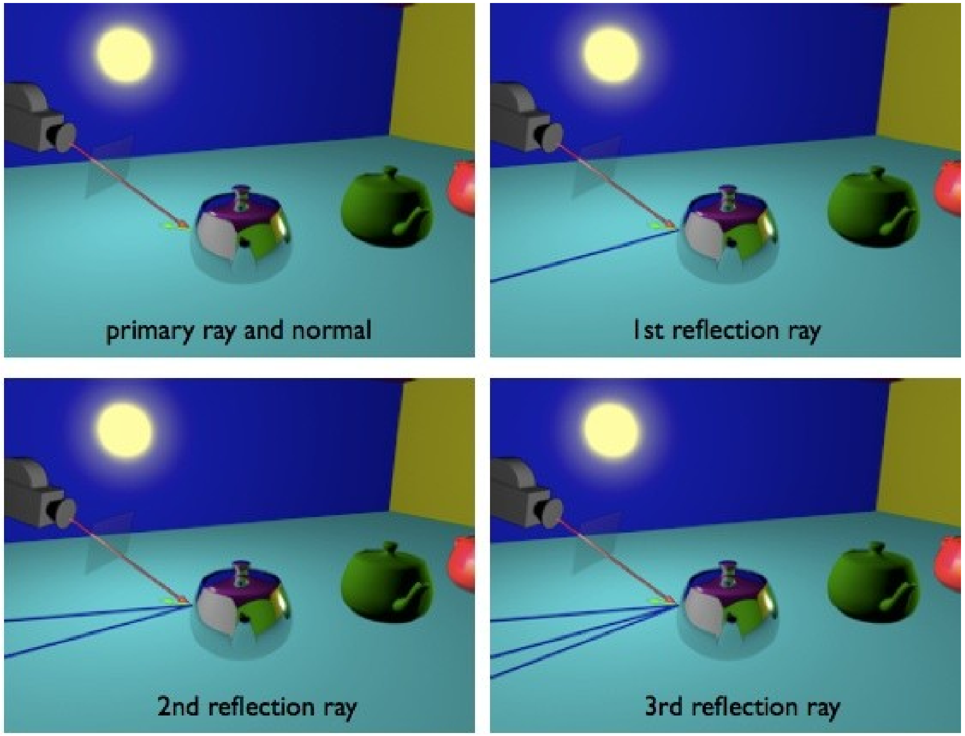


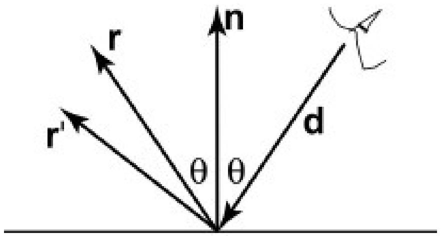


Recursive raytracing: refractions

Stochastic raytracing

- In stochastic raytracing, more random rays are chosen in a direction interval around the main reflection
- This allows with one method:
  - glossy reflections
  - soft shadows
  - antialiasing
- Also called Montecarlo raytracing

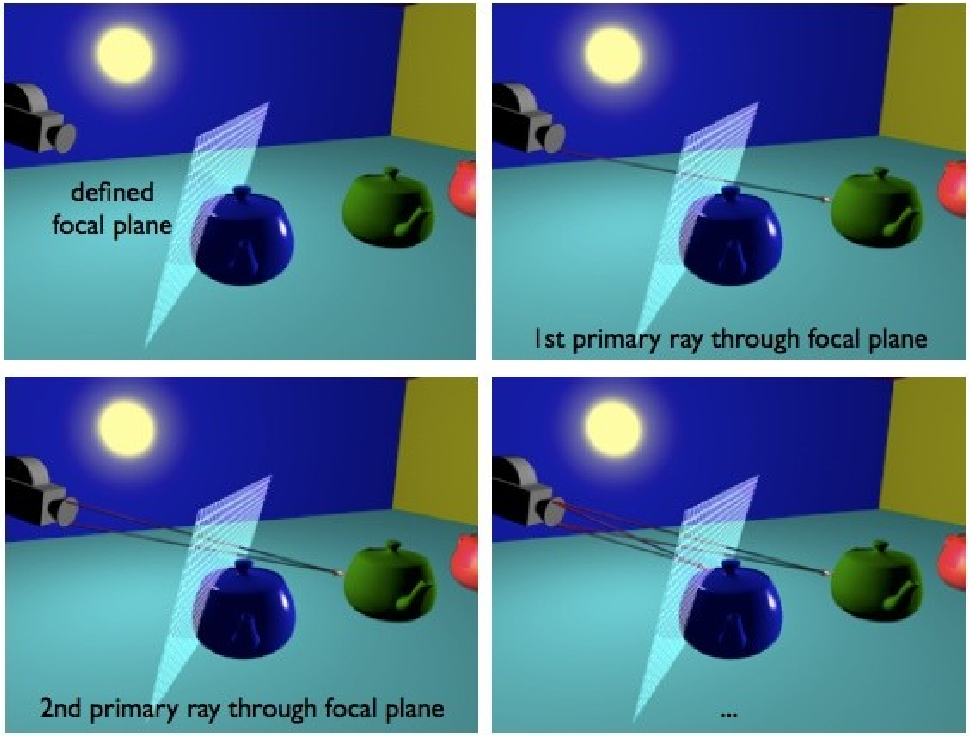


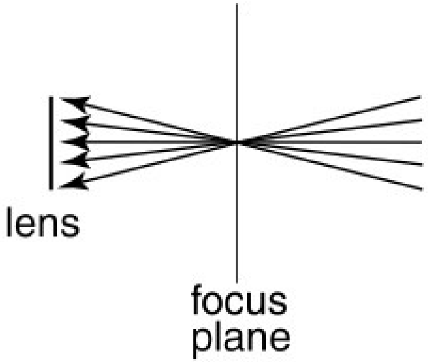


Stochastic raytracing

Stochastic raytracing

- Stochastic raytracing can also be used to simulate the depth of field of cameras
  - Achieved by introducing a focus plane
  - The focus plane for rays blurs the image on the image plane
  - Send stochastic rays to it to simulate blur

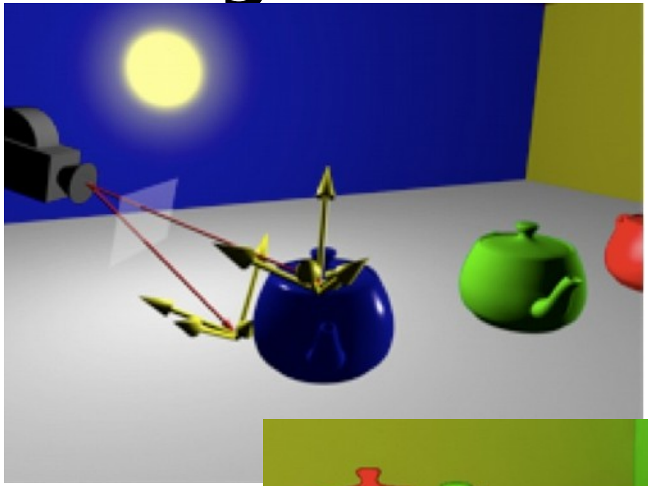


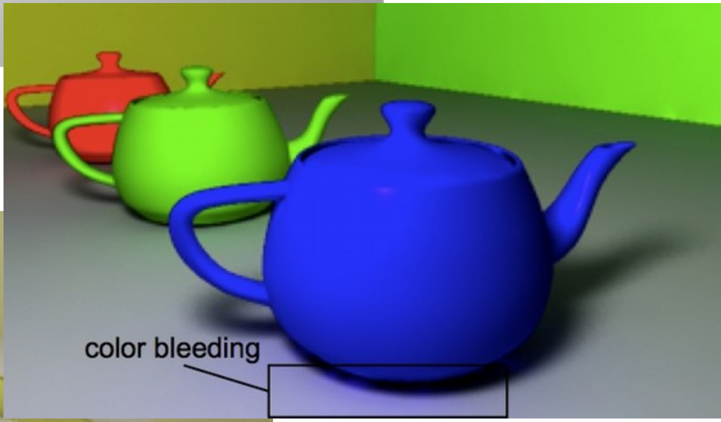


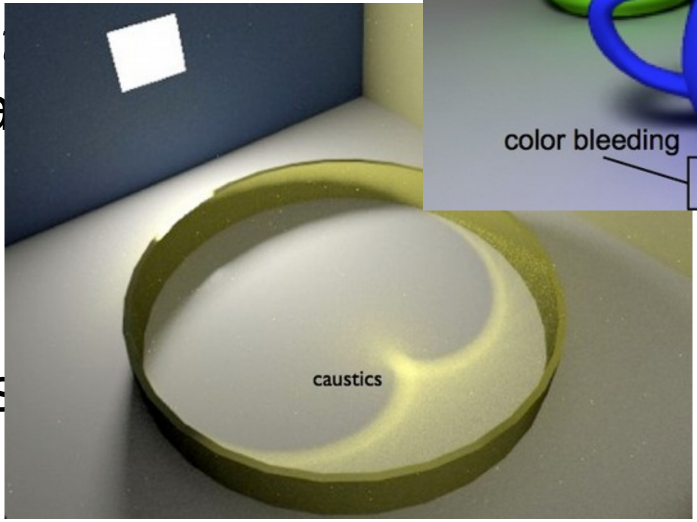
Stochastic raytracing

Path tracing

- On glossy surfaces one can generate random rays too (*path tracing*) in order to simulate diffuse reflections
  - Colour bleeding
  - Caustics
- In *bidirectional pa* *tracing* multiple ra are shot
  - from the eye
  - From light sources
- *Photon mapping* is similar







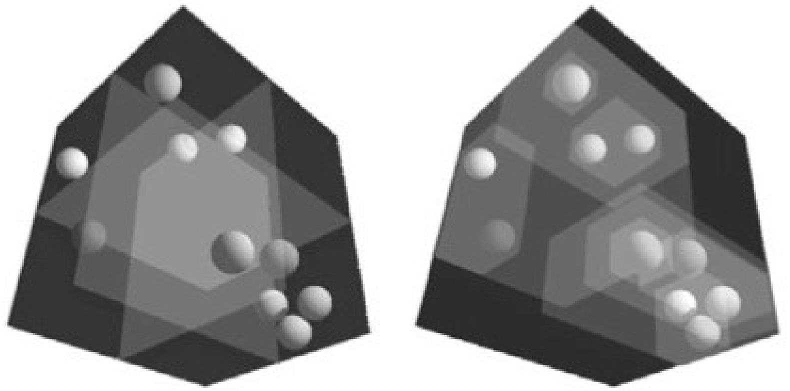
Raytracing efficiency

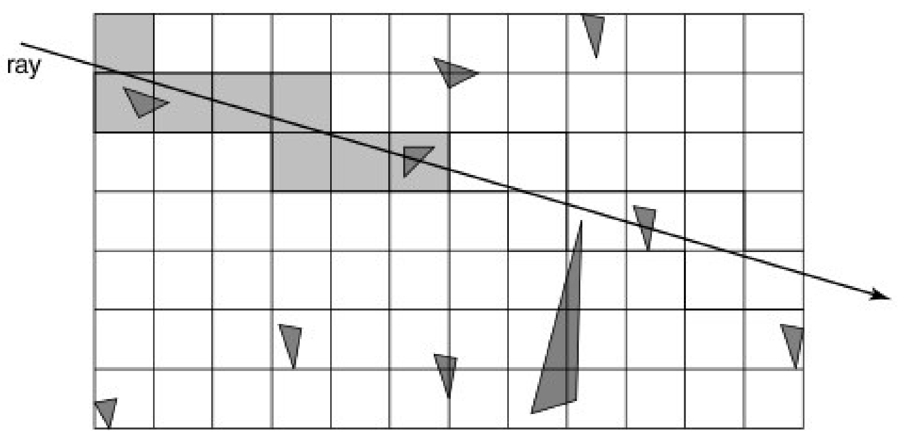
- Raytracing is not very efficient when it comes to simulating caustics and bleeding.
- Every ray has to be intersected with all scene polygons
  - At each intesection, multiple rays are generated
  - This leads to a huge number of rays structured in a tree
  - Such a tree has to be generated for each pixel of the screen
- Recursive generation also implies a stop criterion is needed for the generation of rays

- When do I stop?
  - rays do not hit any objects
  - maximal tree depth is reached (two mirrors)
  - Ray contribution is neglegible (ray damping) (ex. 5%)

Raytracing speedups

- Note that each ray has to be intersected with the whole polygons in the environment
- There are speedups to avoid computing loads of intersections
  - Bounding volumes: complex objects are wrapped in simple volumes (hulls) and intersection ray-object is done first on hull, only if hit is available real intersection is done
  - Hierarchical bounding volumes: bounding volumes are done hierarchically (clusters of objects)
  - Octrees can be used to do intersections, or space can be partitioned in volume units





Conclusion

- Interactive rates (>15fps) for raytracing are being achieved by
  - Implementing in clusters, and distributing rays to processors
  - Doing it on graphics cards, albeit only for raycasting
- Raytracing does model well reflections and refractions, however it is still an incomplete instrument (no colour bleeding from surfaces)
- Raytracing is suitable for parallel machines, and computer clusters (highly parallelizable)
- Often, raytraced pictures are overloaded with Christmas balls and mirrors (questionable aesthetics)
- Take your time to take a look at radiance page on http://www.education.siggraph.org under coursware or http://radsite.lbl.gov/radiance/framew.html

Examples

Courtesy Martin Moeck, Siemens Lighting, 1994

Examples

Courtesy R. Mc Farland, S. ROuten, U. of Indiana

Examples

Kajiyas Rendering Equation

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

- James T. Kajiya, Siggraph '86

- *x*, *x*', *x*'' : Points in the environment
- *I(x,x')* : Light Intensity from x' to x

Kajiyas Rendering Equation

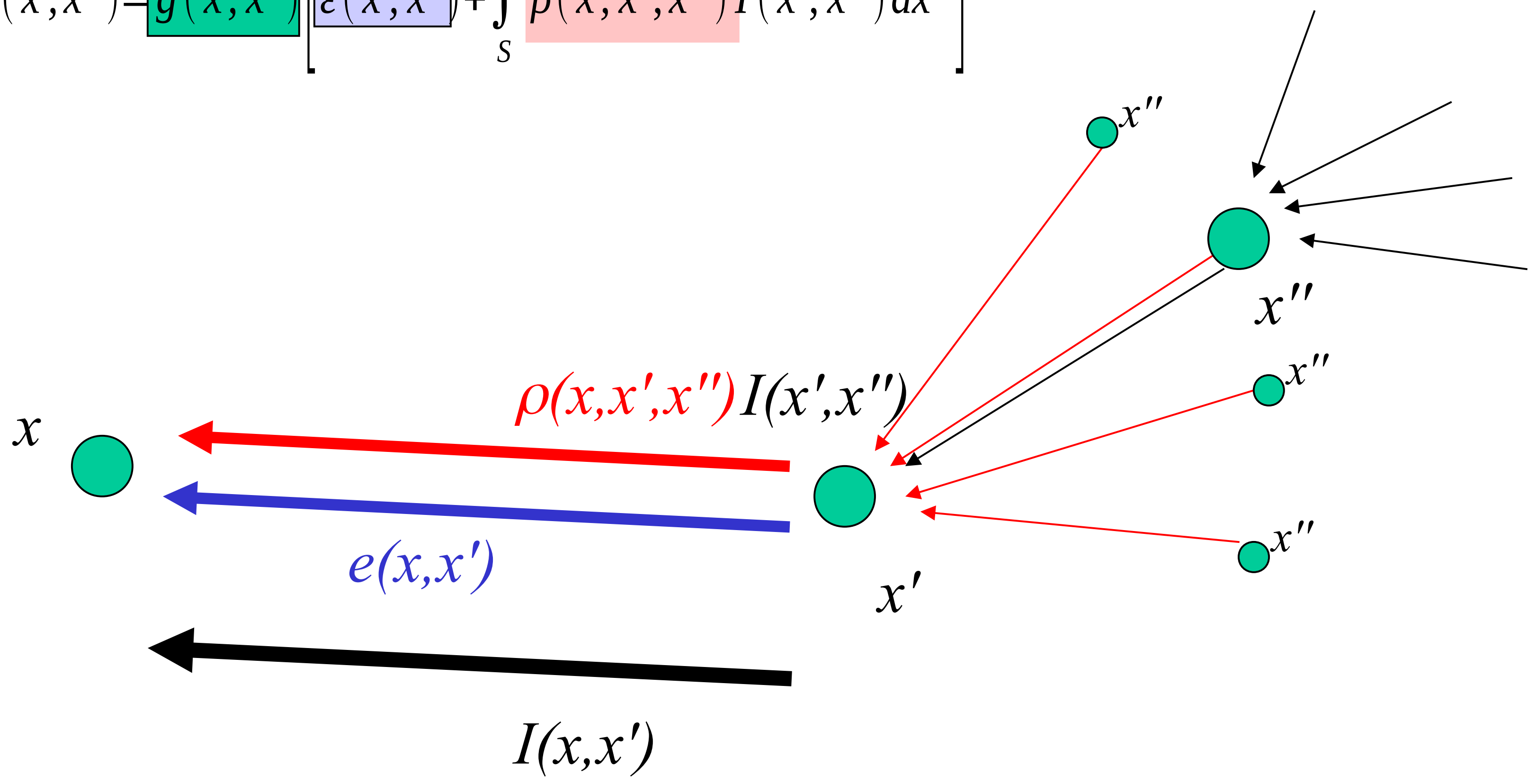
$$I(x,x') = g(x,x')\left[\varepsilon(x,x') + \int_S \rho(x,x',x'') I(x',x'') dx''\right]$$

Where

- $g(x,x')$: Visibility term (geometry factor)
  - $g(x,x')=0$ if x,x' mutually invisible else $g = 1/d(x,x')^2$
- $\varepsilon(x,x')$: Light emitted directly from x' to x
- $\rho(x,x',x'')$: Reflection coefficient
  - Intensity arriving in x, that has been originated at x'', and reflected through x'
- The integral is made on all surfaces in the environment

Kajiyas Rendering Equation

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



Kajiyas Rendering Equation

- Notes:
  - *g(x,x')*ε(x,x')* codes visibility information.
    If x=Viewpoint it is hidden surface computations
  - The rendering equation is computationally very complex, the integral extends to all surfaces in the environment
  - In „partecipating media“, such as foggy environments, the integral is done on all points of the volume considered
  - All Illumination Methods are in some ways solutions to the Kajiya's equation

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

End

+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++