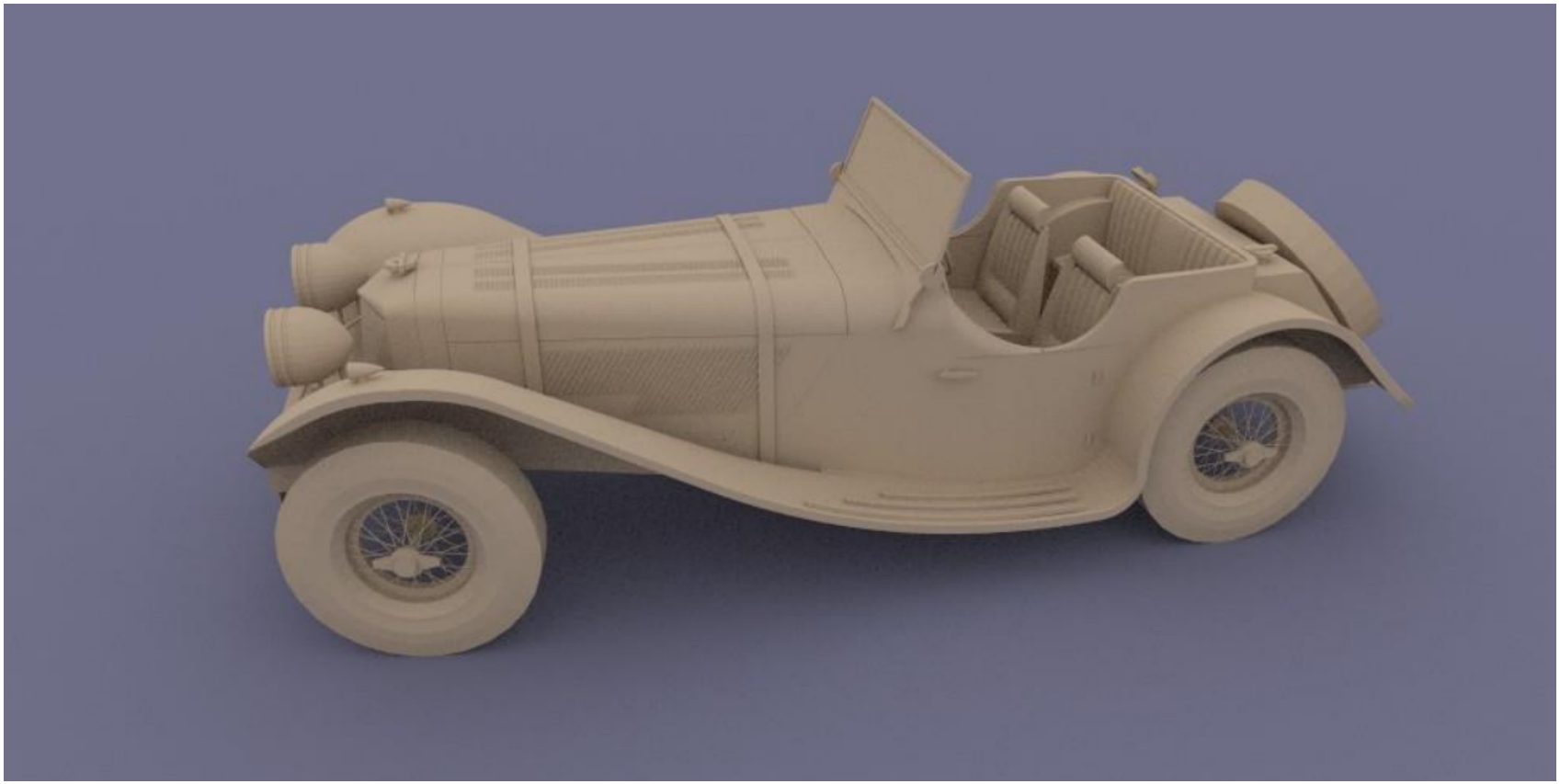
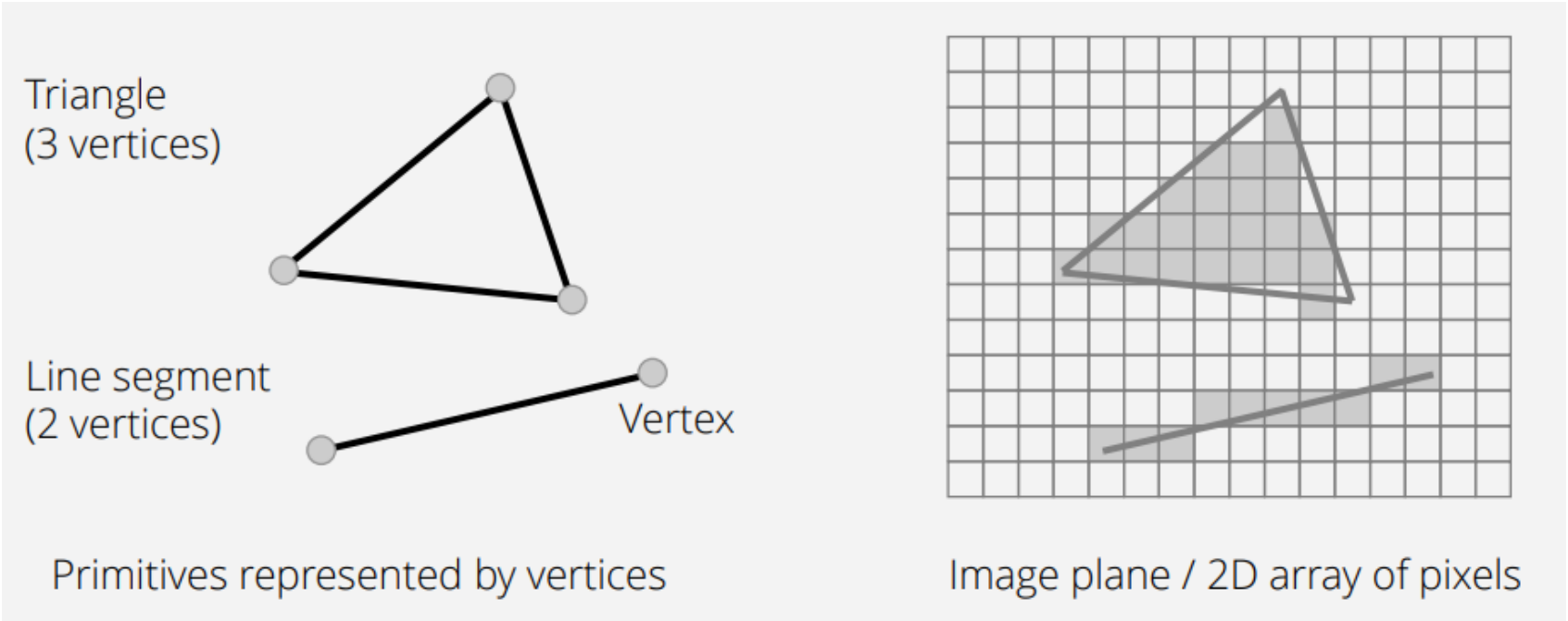
CS 461 - Computer Graphics

Ray Casting & Ray Tracing

3D World

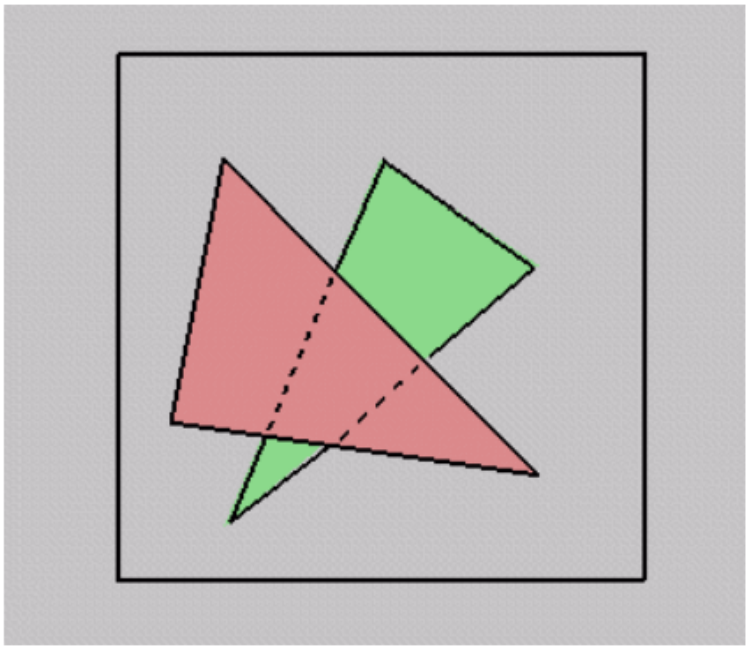


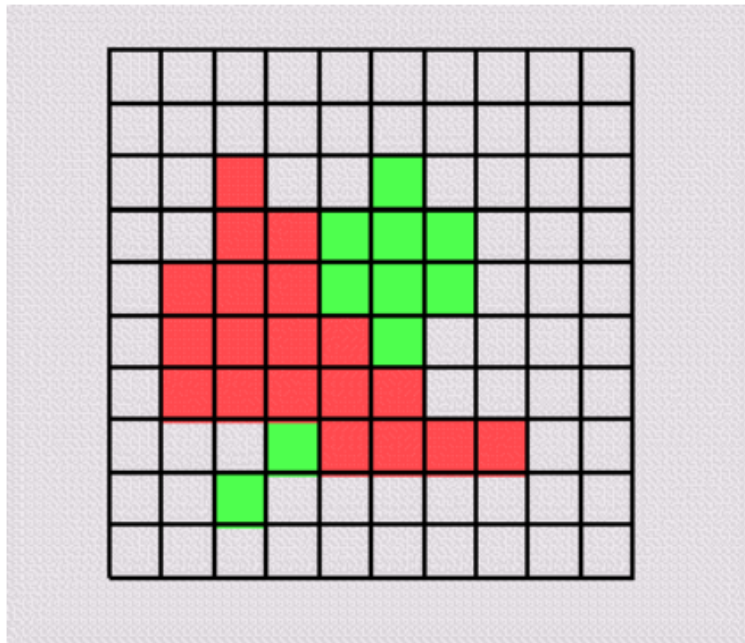
Rasterization



Z-Buffer

Rasterization





Ray casting

3D parametric line

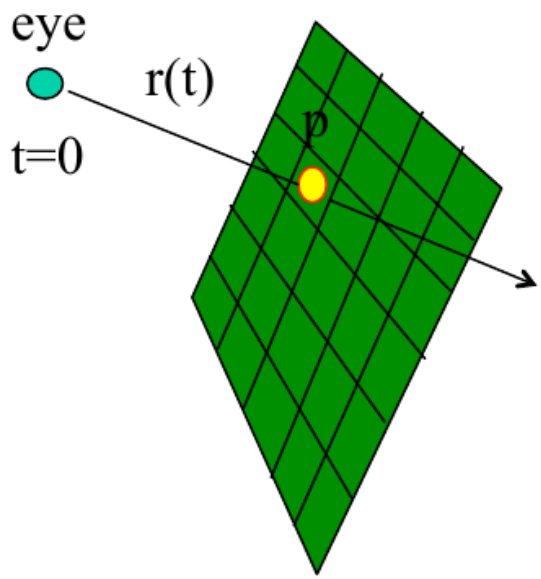
$$p(t) = \text{eye} + t\,(s - \text{eye})$$

r(t): ray equation

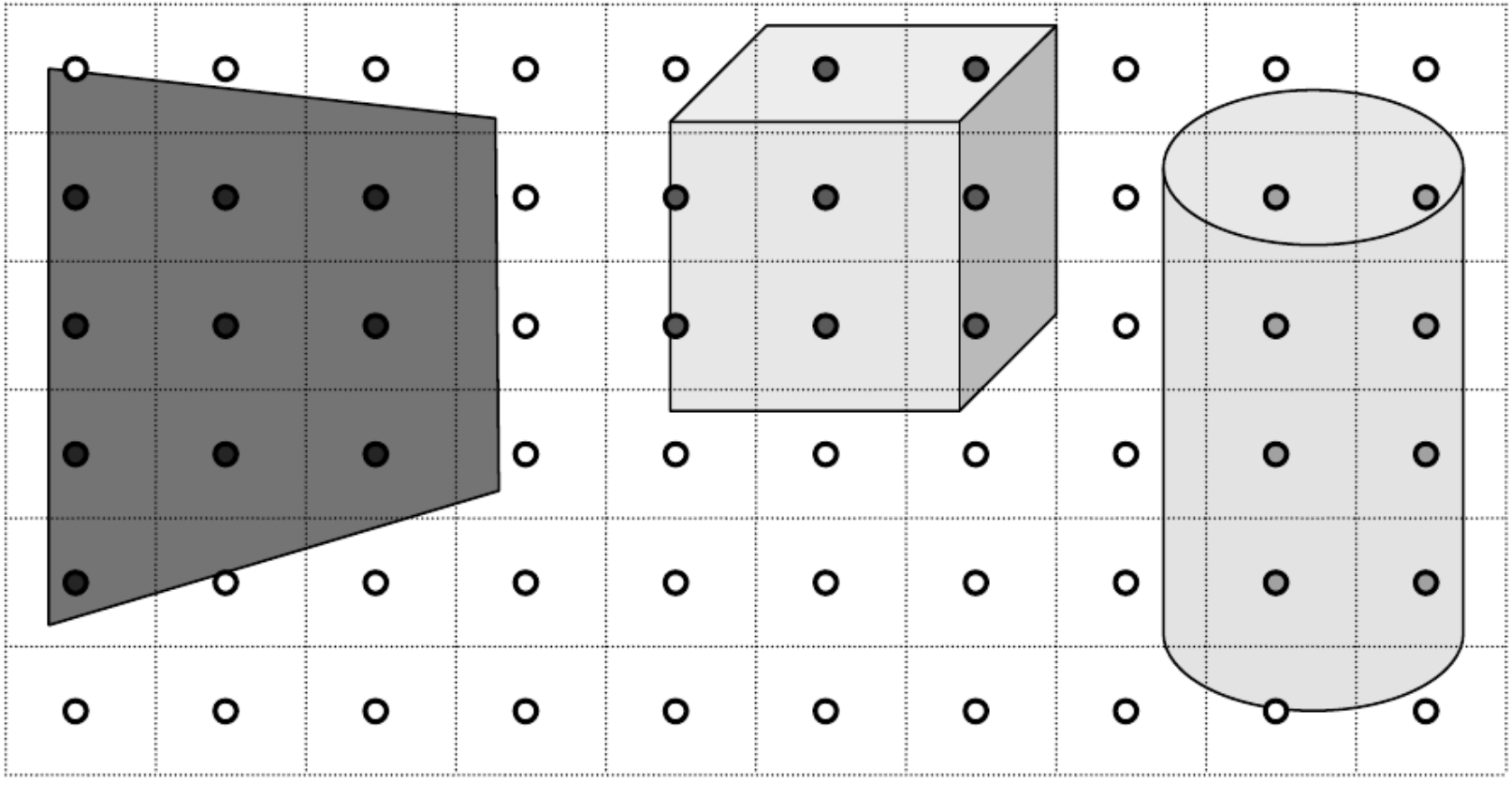
eye: eye (camera) position

s: pixel position

t: ray parameter

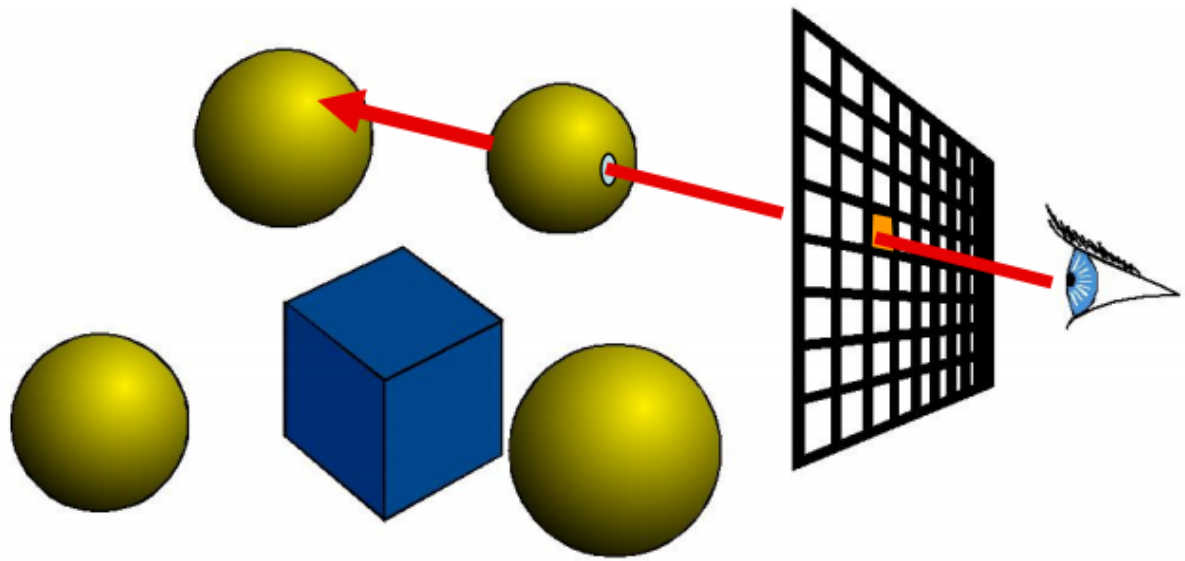


Ray-casting



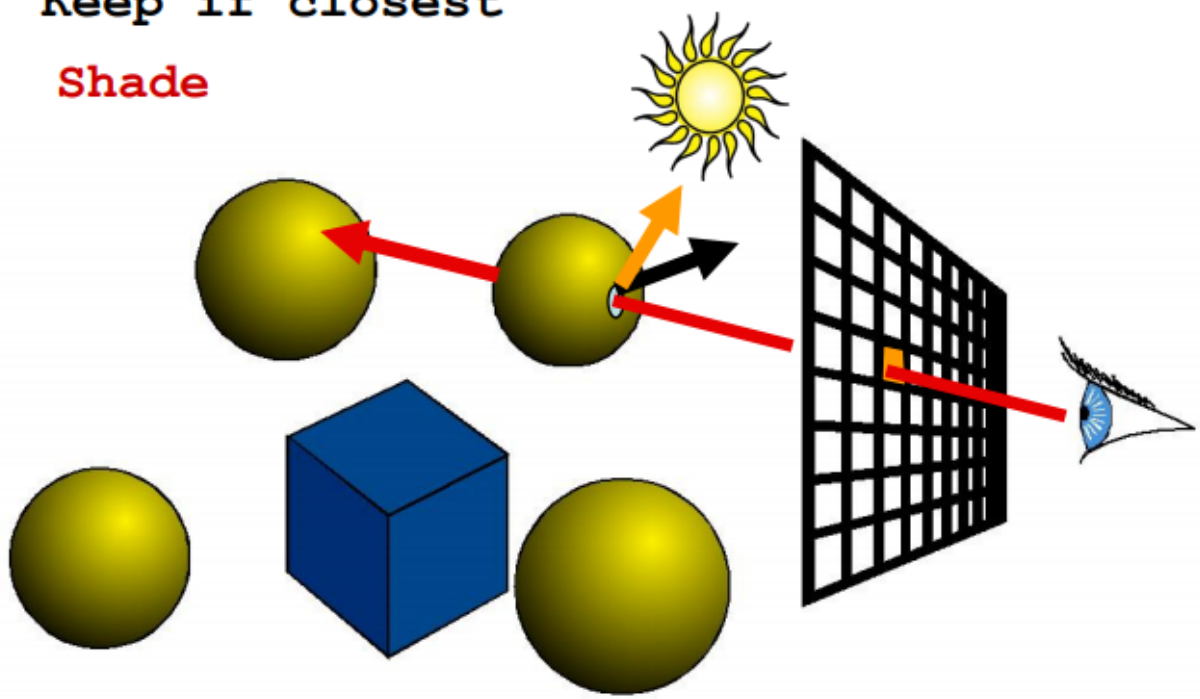
Ray-casting algorithm

```
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 algorithm

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



Ray-Sphere intersection

$$ray \equiv P = P_0 + P_1 t$$

$$sphere \equiv (P - C)(P - C) - r^2 = 0$$

Substitute

$$ray \equiv P = P_0 + P_1 t$$

$$sphere \equiv (P_0 + P_1 t - C)(P_0 + P_1 t - C) - r^2 = 0$$

Simplify

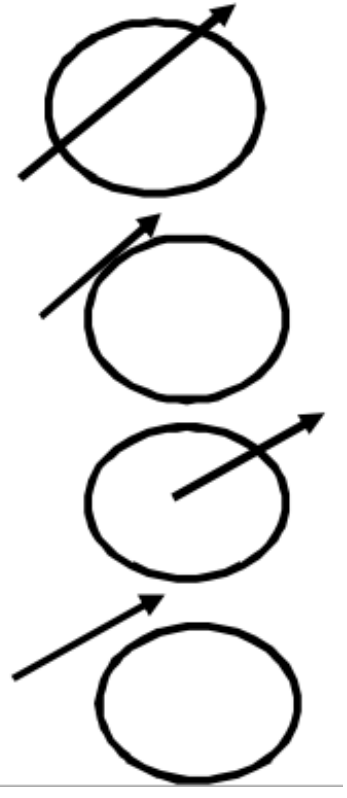
$$t^2 (P_1 P_1) + 2t\, P_1 (P_0 - C) + (P_0 - C)(P_0 - C) - r^2 = 0$$

Ray-Sphere intersection

$$t^2(P_1P_1)+2t\,P_1(P_0-C)+(P_0-C)(P_0-C)-r^2=0$$

Solve quadratic equations for t

- 2 real positive roots: pick smaller root
- Both roots same: tangent to sphere
- One positive, one negative root: ray origin inside sphere (pick + root)
- Complex roots: no intersection (check discriminant of equation first)



Ray-Triangle intersection

- One approach: Ray-Plane intersection, then check if inside triangle
- Plane equation:

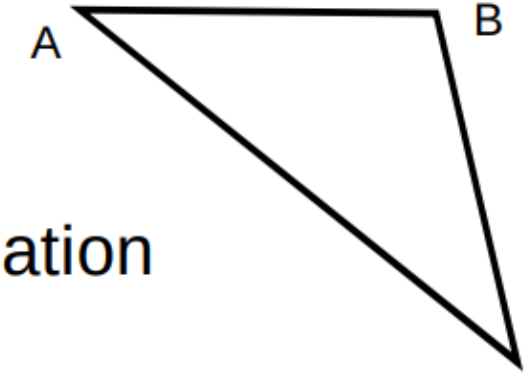
$$plane \equiv Pn - An = 0$$

- Combine with ray equation

$$ray \equiv P = P_0 + P_1 t$$

$$(P_0 + P_1 t)n = An$$

$$n = \frac{(C-A)\times(B-A)}{|(C-A)\times(B-A)|}$$



$$t = \frac{An - P_0 n}{P_1 n}$$

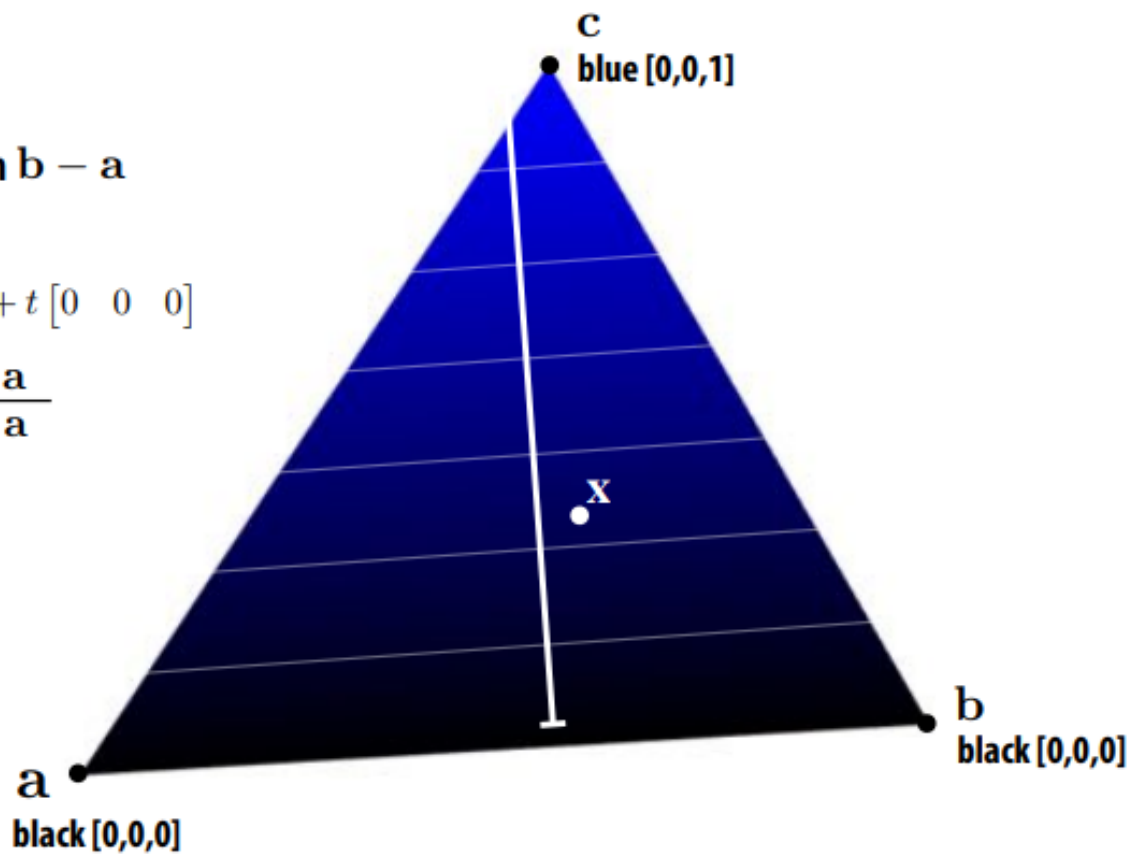
- Explain Ray-Box intersection

Barycentric coordinates

Color depends on distance from $\mathbf{b} - \mathbf{a}$

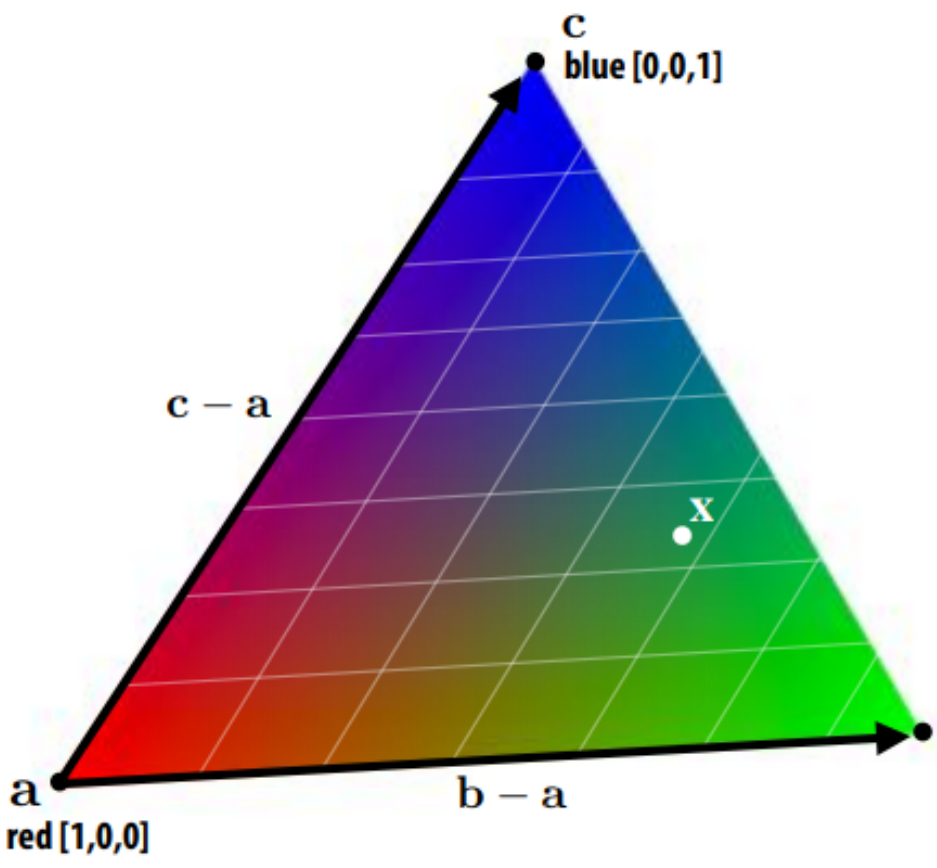
color at $\mathbf{x} = (1-t)\begin{bmatrix}0 & 0 & 1\end{bmatrix} + t\begin{bmatrix}0 & 0 & 0\end{bmatrix}$

$$t = \frac{\text{distance from } \mathbf{x} \text{ to } \mathbf{b} - \mathbf{a}}{\text{distance from } \mathbf{c} \text{ to } \mathbf{b} - \mathbf{a}}$$



How can we interpolate in 2D between three values?

Barycentric coordinates



$\mathbf{b}-\mathbf{a}$ and $\mathbf{c}-\mathbf{a}$ form a non-orthogonal basis for points in triangle (origin at $\mathbf{a}$)

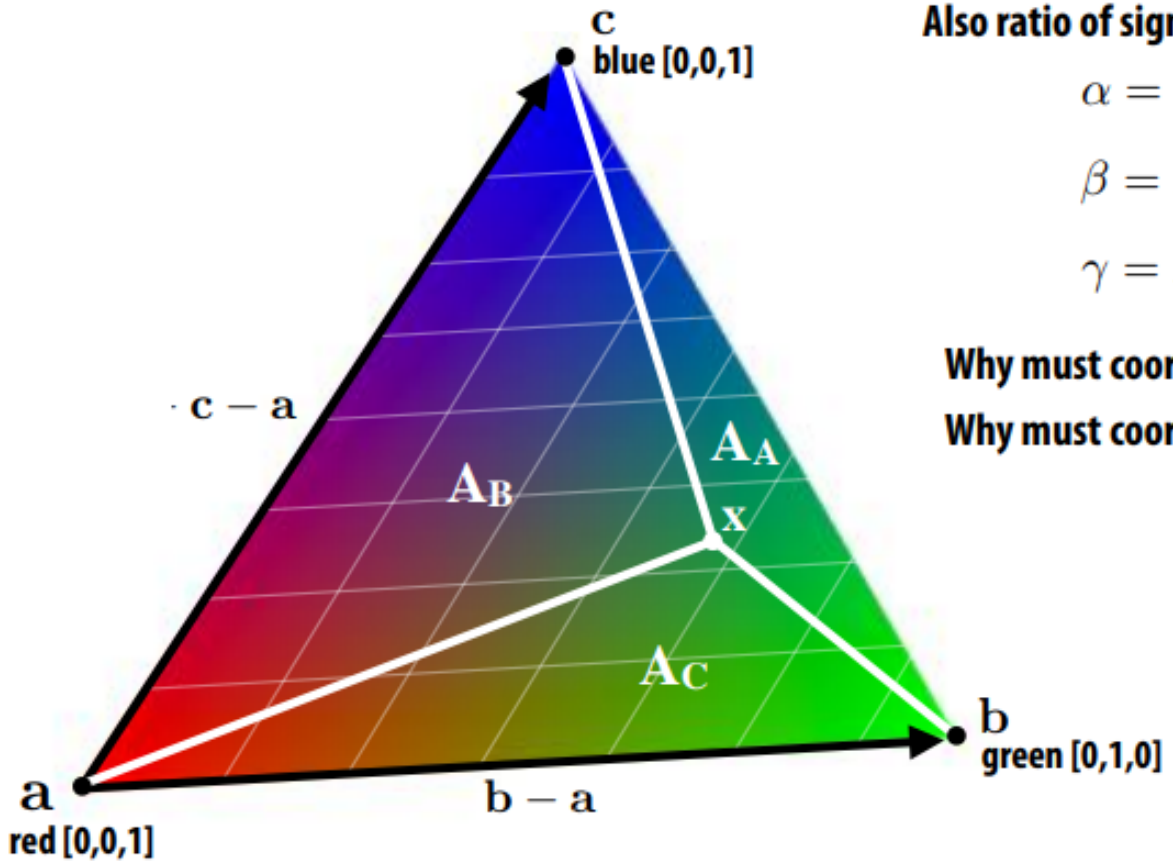
$$\begin{aligned}\mathbf{x} &= \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a}) \\ &= (1-\beta-\gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\ &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}\end{aligned}$$

$$\alpha + \beta + \gamma = 1$$

Color at $\mathbf{x}$ is linear combination of color at three triangle vertices.

$$\mathbf{x}_{\text{color}} = \alpha\mathbf{a}_{\text{color}} + \beta\mathbf{b}_{\text{color}} + \gamma\,\mathbf{c}_{\text{color}}$$

Barycentric coordinates



Also ratio of signed areas:

$$\alpha = A_A/A$$

$$\beta = A_B/A$$

$$\gamma = A_C/A$$

Why must coordinates sum to one?

Why must coordinates be between 0 and 1?

Rendering vs Ray-casting

- Rasterization:
  - Proceeds in triangle order (never have to store in entire scene, naturally supports unbounded size scenes)
  - Store depth buffer (random access to regular structure of fixed size)
- Ray casting:
  - Proceeds in screen sample order
  - Never have to store depth buffer (just current ray)
  - Natural order for rendering transparent surfaces (process surfaces in the order the are encountered along the ray: front-to-back or back-to-front)
  - Must store entire scene (random access to irregular structure of variable size: depends on complexity and distribution of scene)
- Conceptually, compared to rasterization approach, ray casting is just a reordering of loops + math in 3D

CGI effects

Photo realistic rendering

Photo realistic rendering

Photo realistic rendering

Photo realistic rendering

Photo realistic rendering

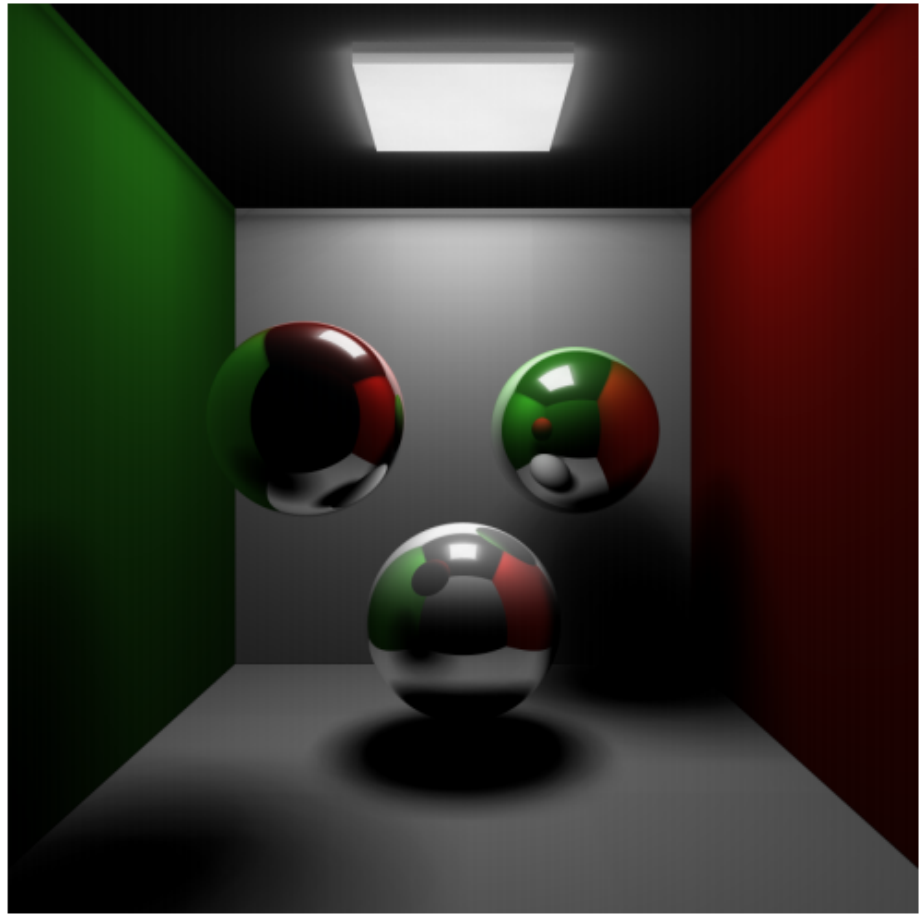
Photo realistic rendering

Photo realistic rendering

Photo realistic rendering

Complex reflections

Soft shadows

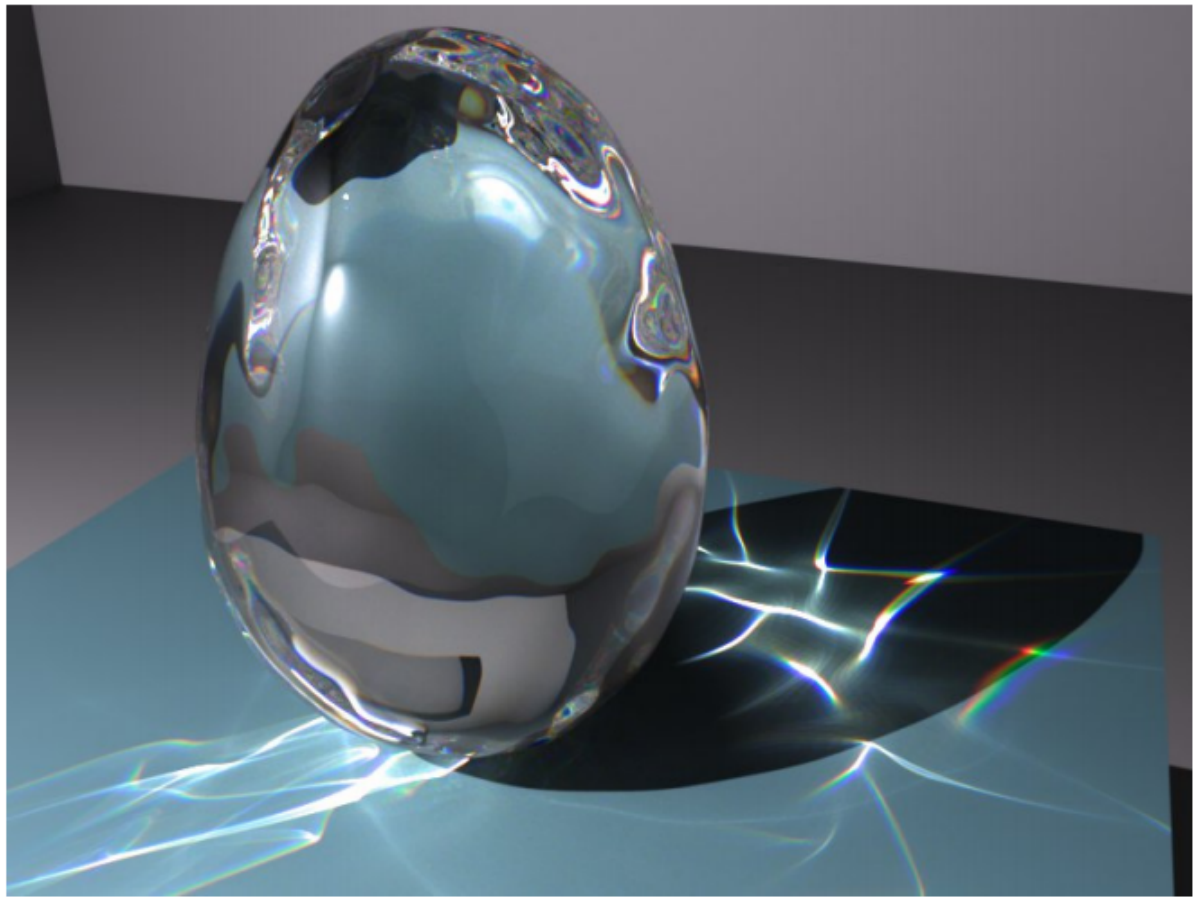


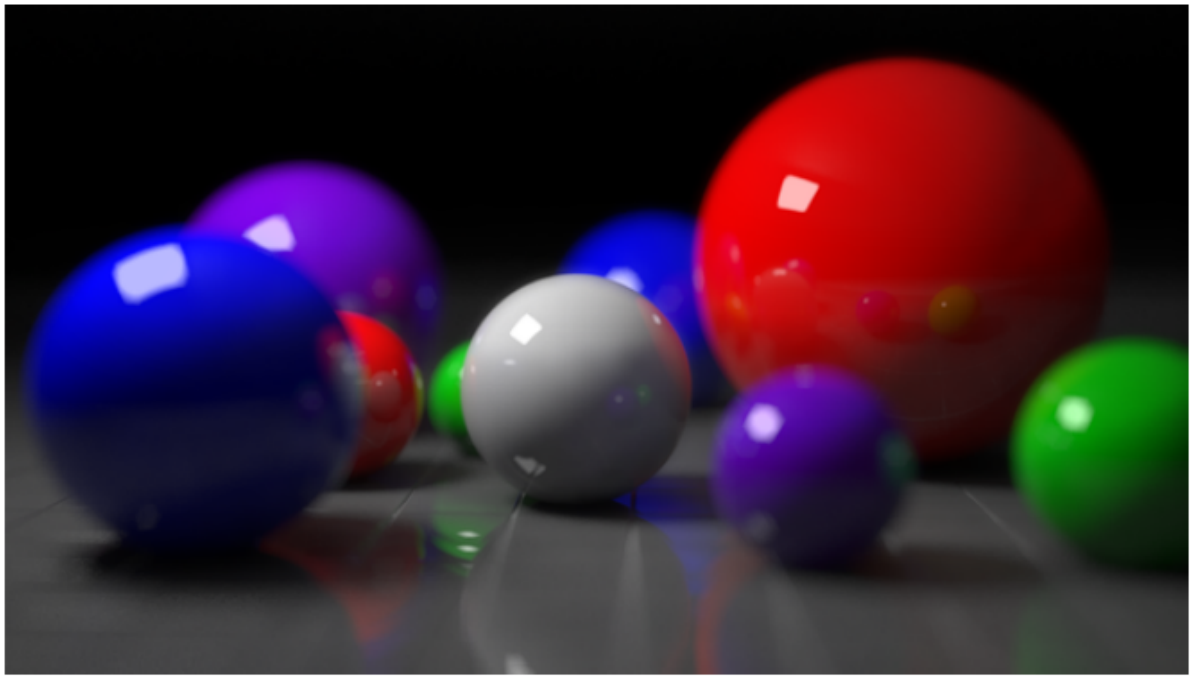
Transparency

Refraction

Caustics

Dispersion





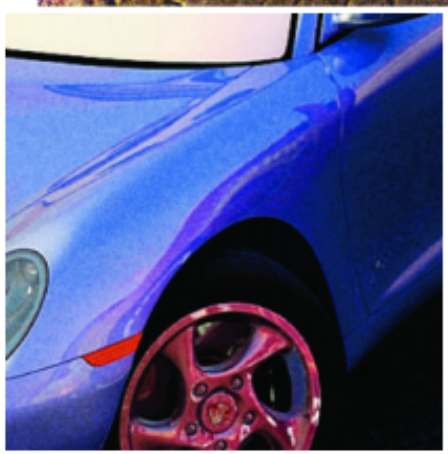
Depth of field

Defocus

Color bleeding

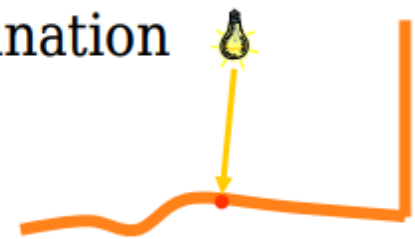
Subsurface scattering

Cars (Pixar)



Global illumination

Direct illumination



Geometric optics

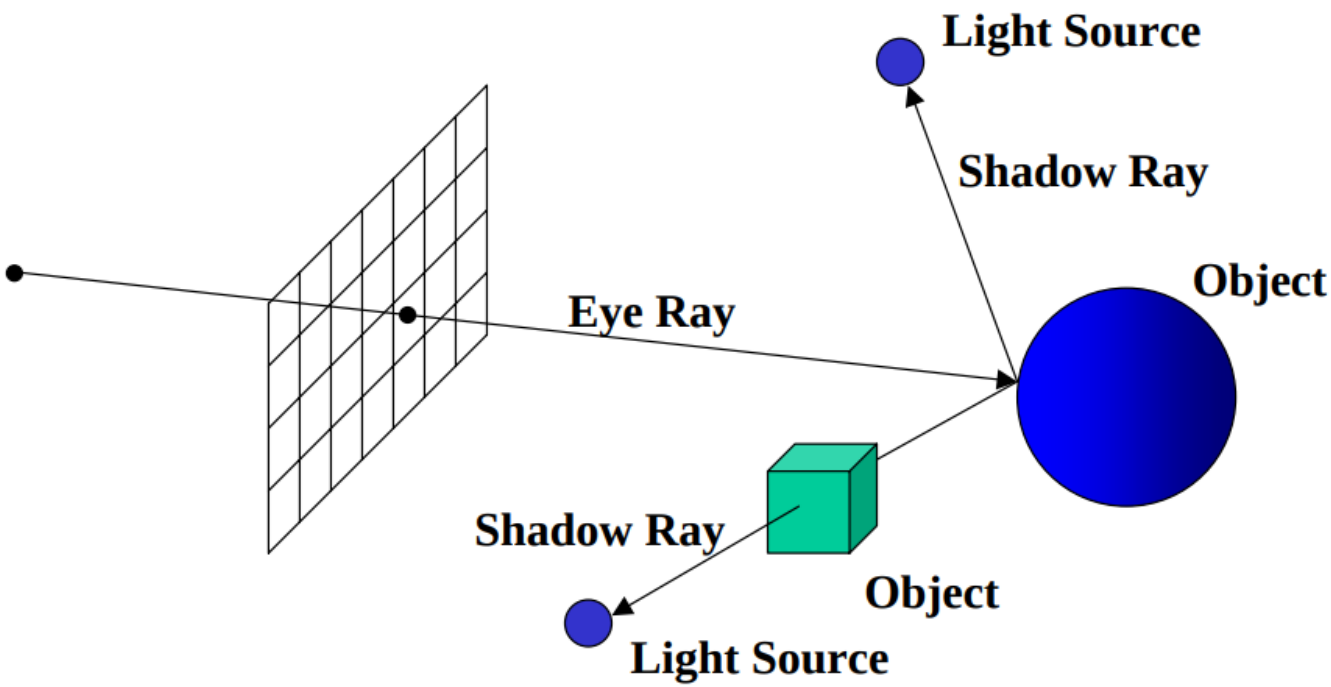
- Modern theories of light treat it as both a wave and a particle.
- We will take a combined and somewhat simpler view of light - the view of geometric optics.
- Here are the rules of geometric optics:
  - Light is a flow of photons with wavelengths. We'll call these flows "light rays."
  - Light rays travel in straight lines in free space.
  - Light rays do not interfere with each other as they cross.
  - Light rays obey the laws of reflection and refraction.
  - Light rays travel from the light sources to the eye, but the physics is invariant under path reversal (reciprocity)

Ray-tracing

- A term from optics
- A “physical” simulation of the particle theory of light
- In the 1960s, ray tracing seemed like a great idea, but nobody could do it well enough to beat cheaper image synthesis methods.
- These days, we can follow the simulation deeply enough to get great results!
- But there are some visual phenomena that ray tracing cannot do.

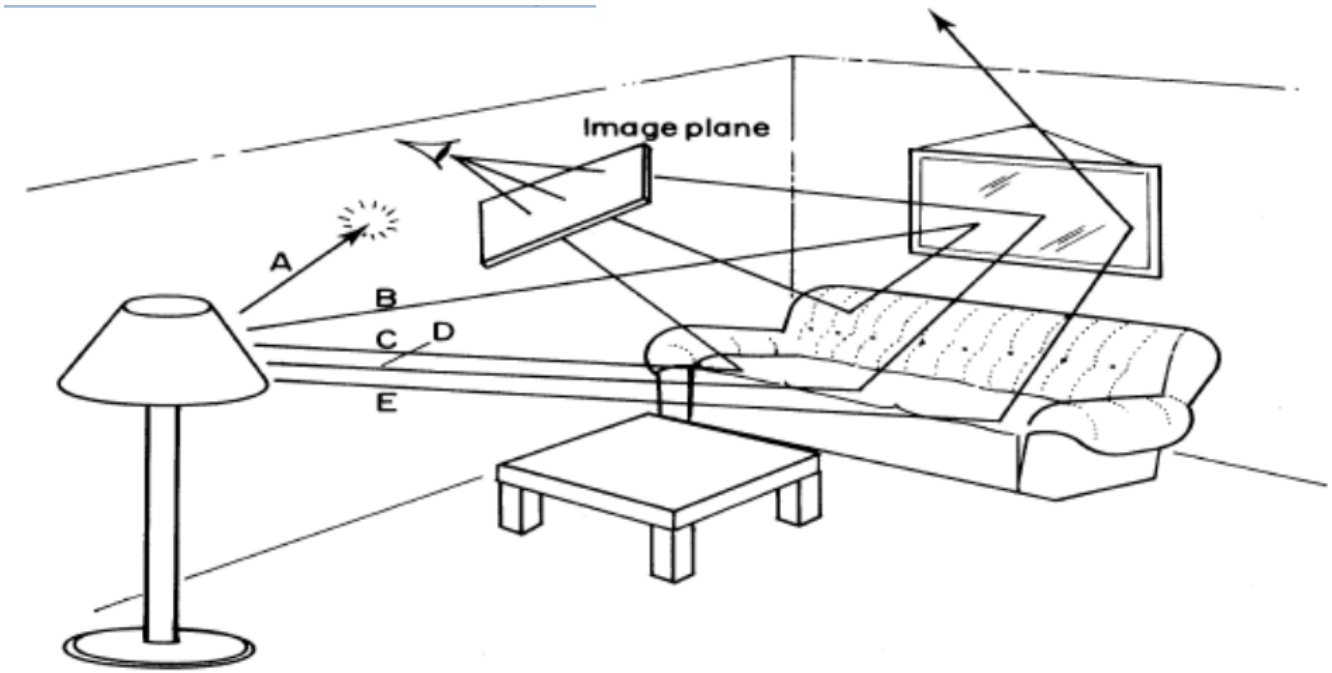
Ray-tracing

- Ray casting:
  - for each pixel in the projection plane, find the object visible at that pixel and
  - color that pixel according to the object color
- What does this model miss? What if the object is reflective?



Forward ray-tracing

- Rays emanate from light sources and bounce around in the scene.
- Rays that pass through the projection plane and enter the eye contribute to the final image.
- What is missing?

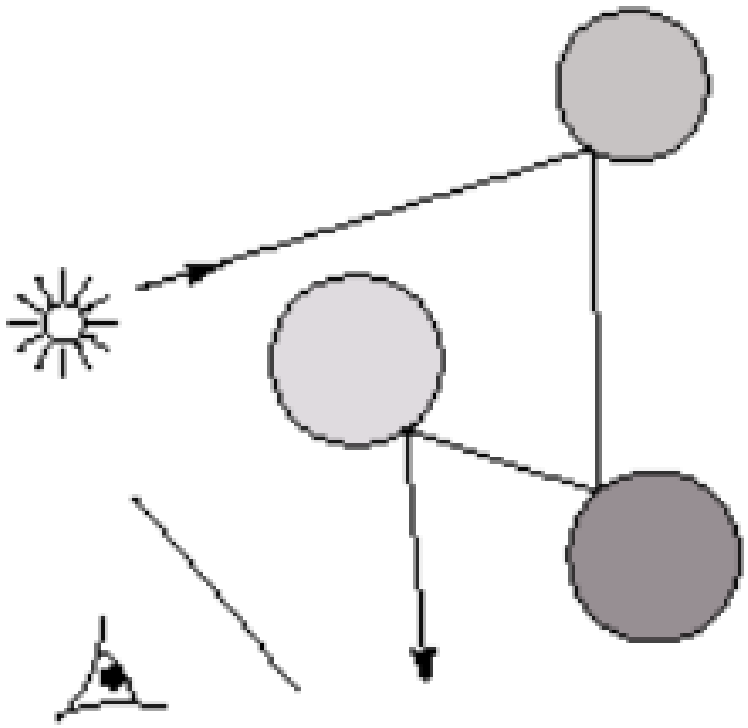


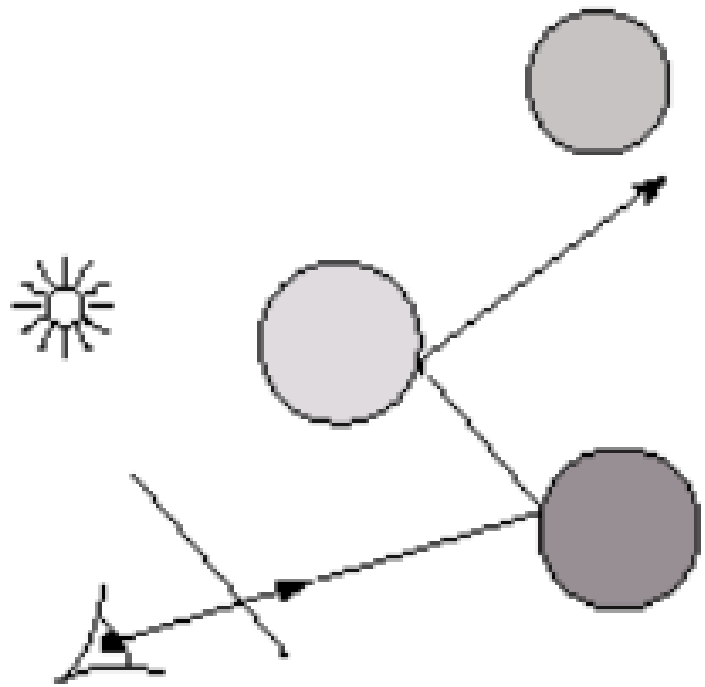
Backward ray-tracing

- (Efficiency!) Rather than propagating rays indiscriminately from light sources, we'd like to ask "which rays will definitely contribute to the final image?"
- We can get a good approximation of the answer by firing rays from the eye, through the projection plane and into the scene
  - These are the paths that light must have followed to affect the image

Eye vs Light ray-tracing

- Where does light begin?
- At the light: light ray tracing (a.k.a. forward ray tracing or photon tracing)
- At the eye: ray tracing (a.k.a. backward ray tracing)





Precursor

- Local illumination - Cast one eye ray, then shade according to light
- Appel (1968) - Cast one eye ray + one ray to light

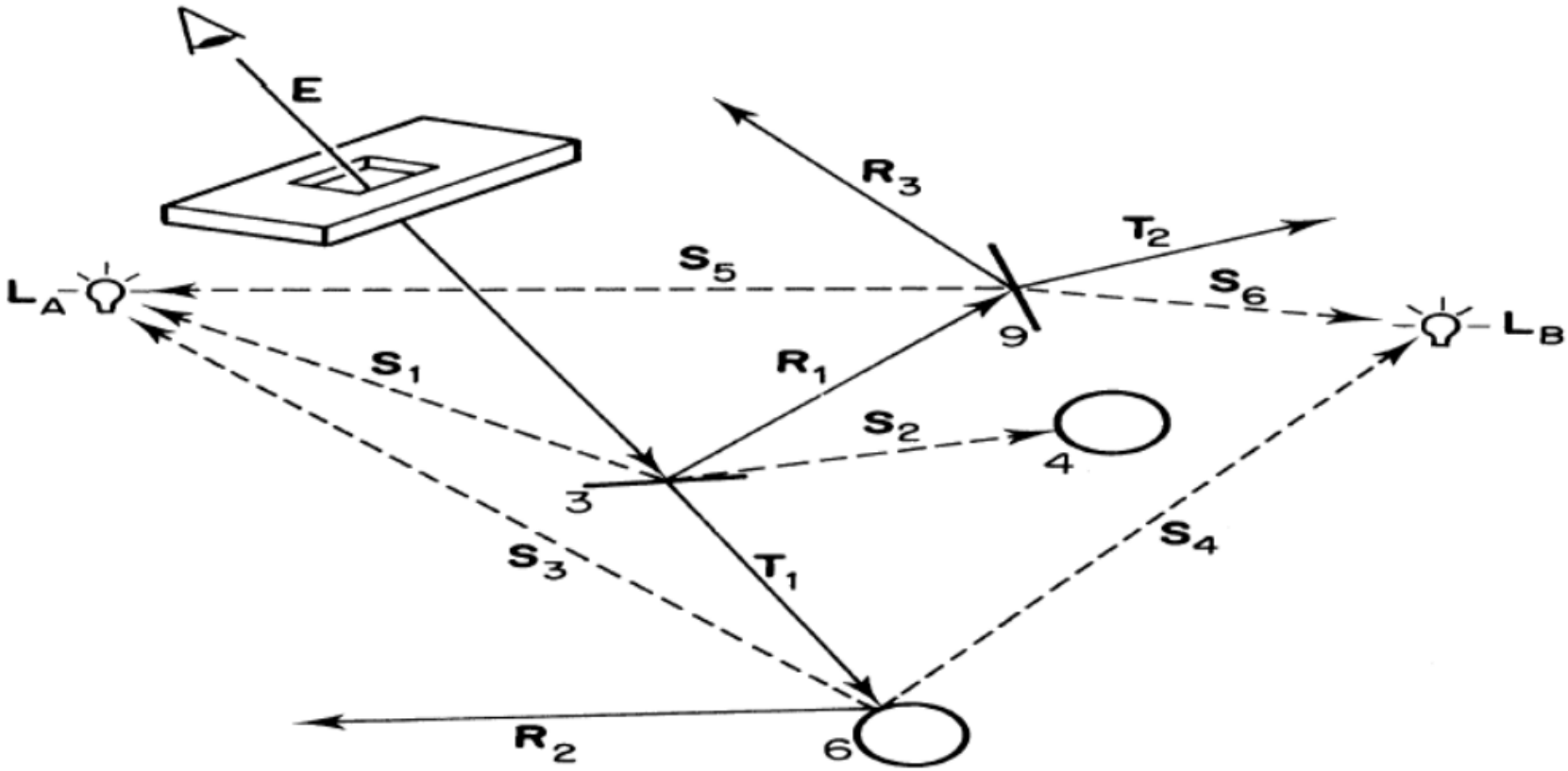
Whitted ray-casting

- In 1980, Turner Whitted introduced ray tracing to the graphics community.
- Combines backward ray tracing + rays to light
- Recursively traces rays
- Algorithm:
  - For each pixel, trace a eye (primary) ray in direction V to the first visible surface (or ray casting)
  - For each intersection, trace secondary rays:
    - Shadow rays in directions to light sources.
    - Reflected ray in direction
    - Refracted ray or transmitted ray in direction

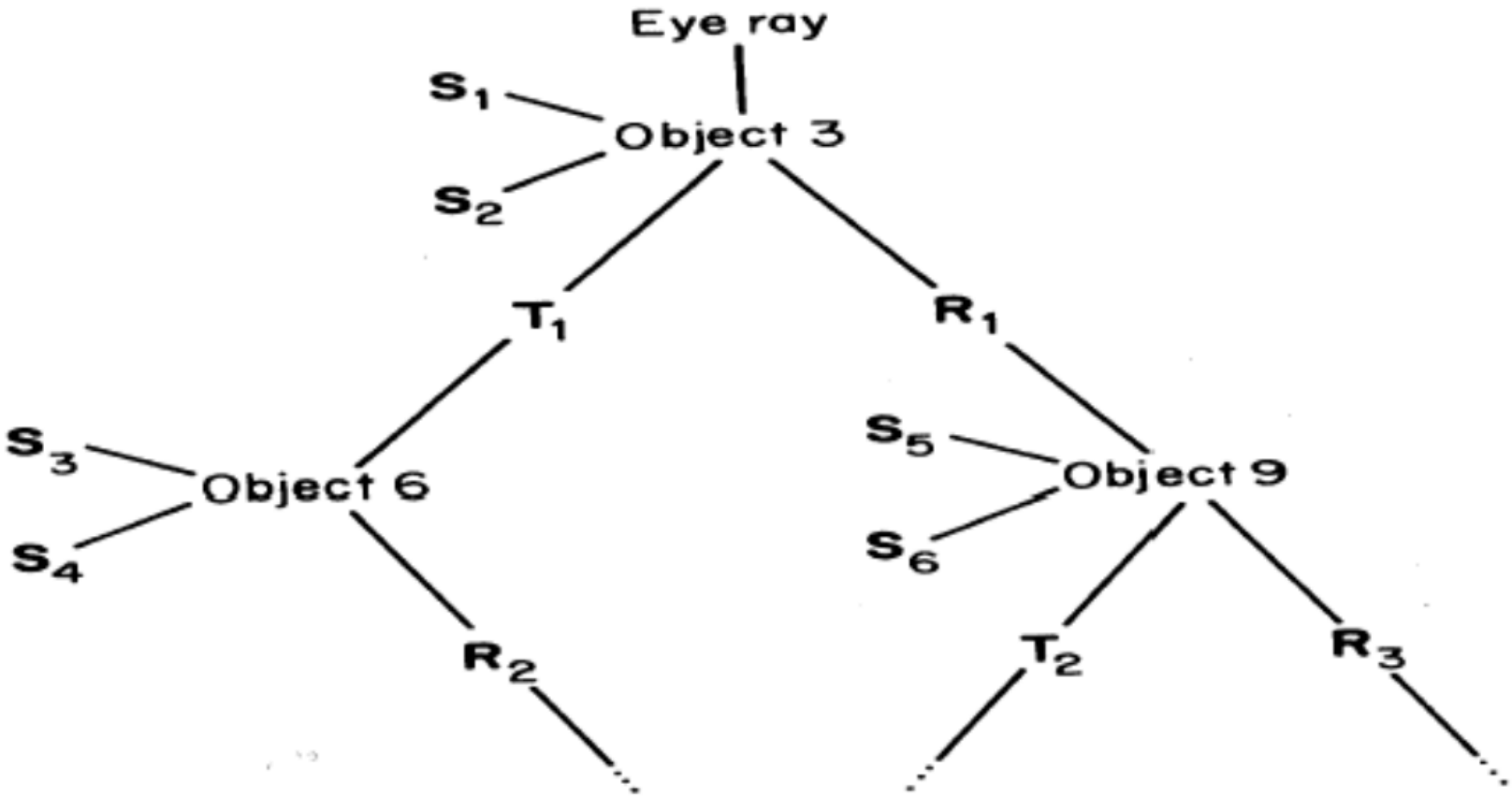
Types of rays

- Three kinds of rays:
  - Shadow rays
  - Reflection rays
  - Transparency rays

Example of ray-tracing

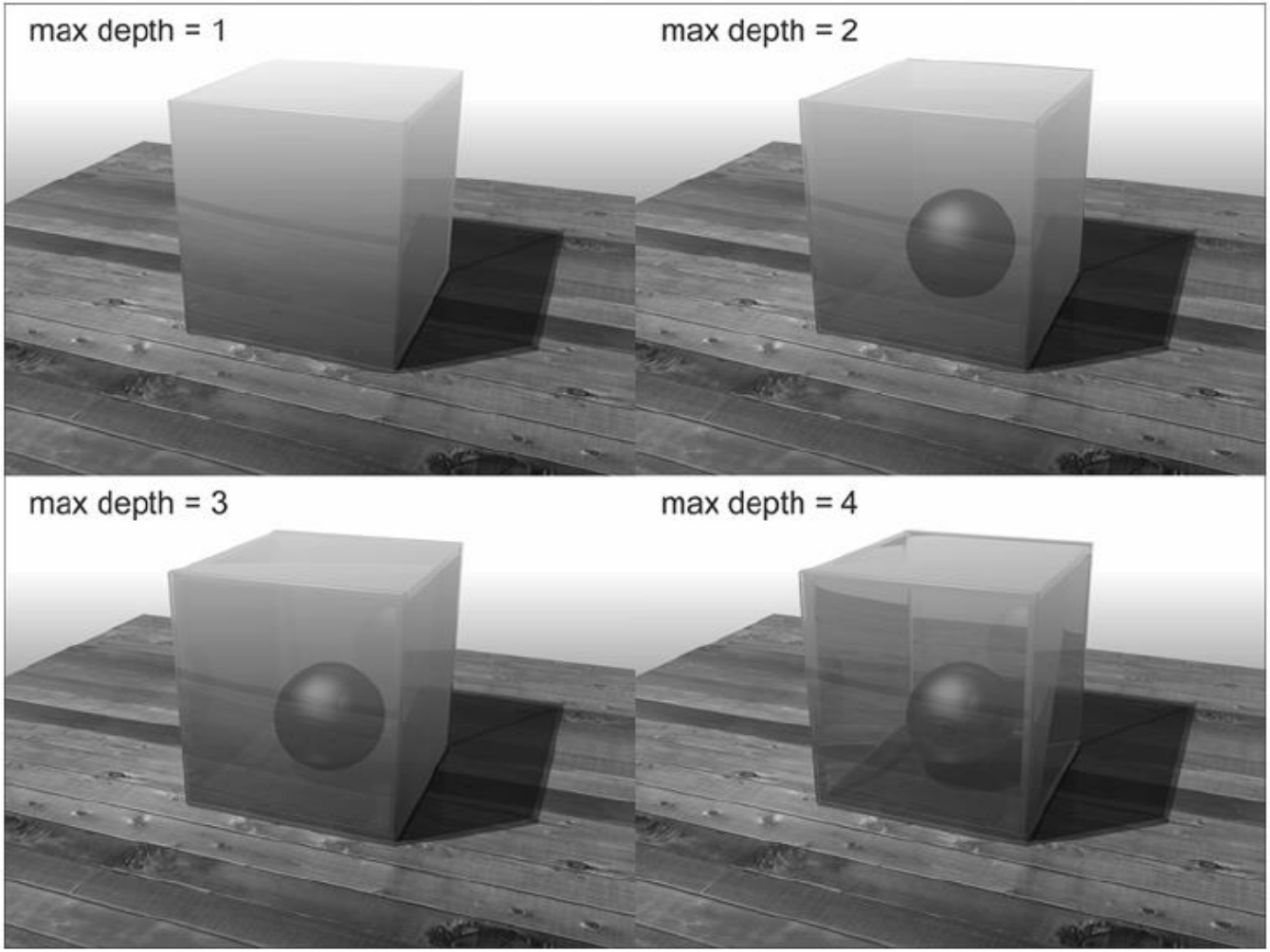


Ray tree



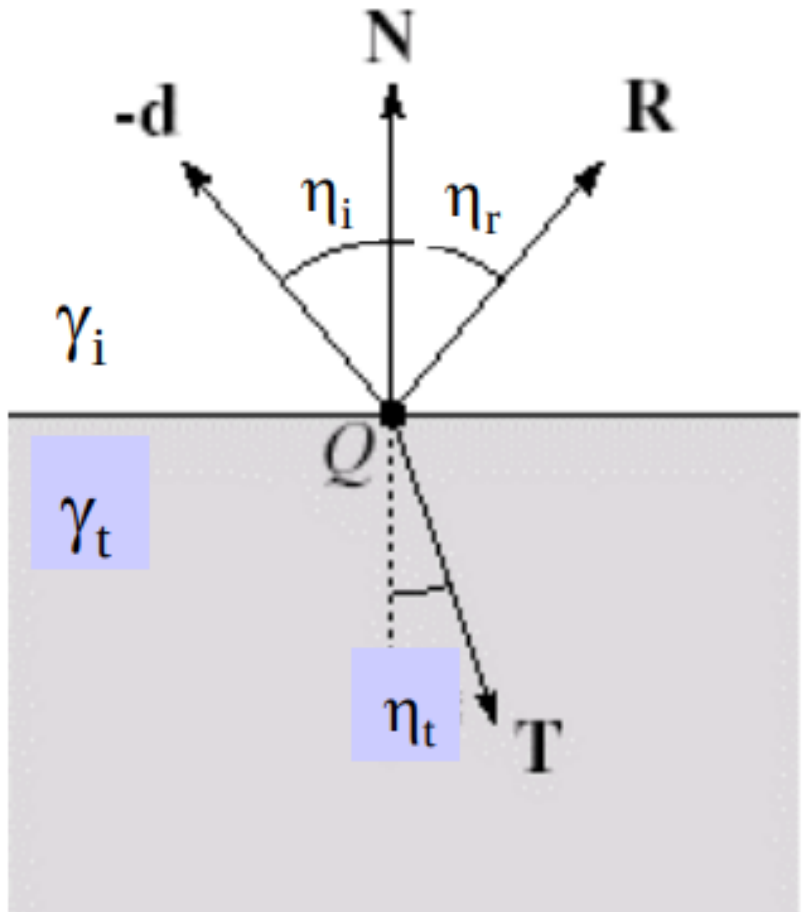
- Stopping conditions

Ray-tracing depth



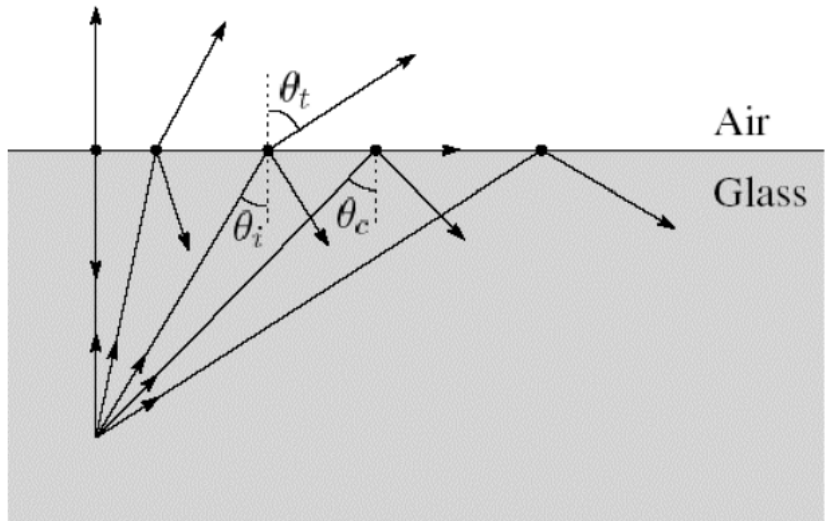
Reflection and Transmission

- Law of reflection $\eta_i = \eta_r$
- Snell's law of refraction: $\gamma_i sin\eta_i = \gamma_t sin\eta_t$
- Where γ_i and γ_r are the indices of refraction



Total internal reflection

- The equation for the angle of refraction can be computed from Snell's law.
- What happens when $\gamma_i > \gamma_t$
- When η_t is exactly 90 deg, we say that η_i "critical angle" η_c
- For $\eta_i > \eta_c$, no rays are transmitted, and only reflection occurs, a phenomenon known as "total internal reflection" or TIR



Recursive ray-tracing algorithm

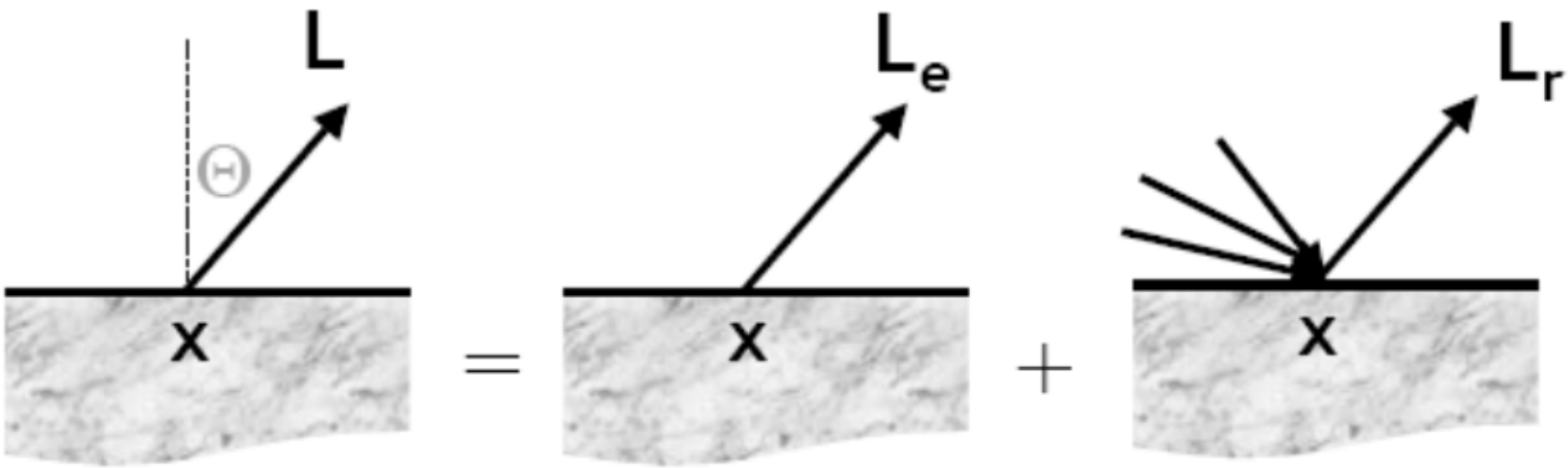
```
for each pixel p do
    r = ray from the eye through p;
    color of p = Trace(r, 0);
endfor
```

Recursive ray-tracing algorithm

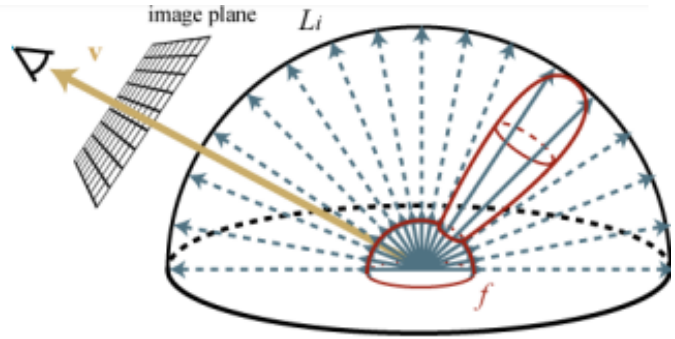
```
Trace(r, d)
    if d > d_max then return background color; endif
    q = Intersect(r);                          // q: object surface point
    if q = null then
      return background color;
    endif
    c = AccLightSource(q);                                  // c: color
    if object (q) is reflective (coherently) then
      r_r = ray towards inverse direction of reflection;
      c += Trace(r_r, d + 1);
    endif
    if object (q) is refractive (coherently) then
      r_t = ray towards inverse direction of refraction;
      c += Trace(r_t, d + 1);
    endif
    return c;
end
```

Rendering equation

- RE describes energy transport in a scene
- Input:
  - light sources
  - geometry of surfaces
  - reflectance characteristics of surfaces
- Output: value of radiance at all surface points and in all directions

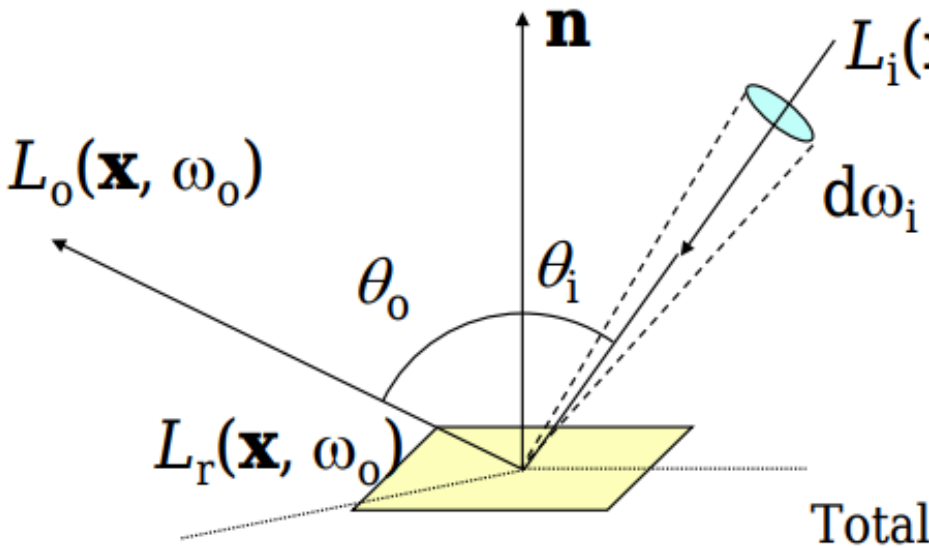


$$L(x \to \Theta) = L_e(x \to \Theta) + L_r(x \to \Theta)$$



- "Sum" (integral) of contributions over the hemisphere:

$$L_r(\mathbf{x}, \omega_o) = \int_{H(\mathbf{x})} L_i(\mathbf{x}, \omega_i) \cdot f_r(\mathbf{x}, \omega_i \rightarrow \omega_o) \cdot \cos\theta_i \, d\omega_i$$



Emitted radiance

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + L_r(\mathbf{x}, \omega_o)$$

Total outgoing rad. Reflected rad.

BSDF

- The fraction of light that is neither absorbed into nor transmitted through a given surface is called the reflectance of a surface

$$\rho(x, \psi_i, \lambda) = \frac{\Phi_r(x, \psi, \lambda)}{\Phi_i(x, \psi_i, (\lambda)}$$

- Similarly - transmittance of a surface
- The reflectance and the transmittance do not describe the spatial distribution of the reflected and transmitted light
- Bidirectional Surface-scattering Distribution Function (BSDF/BDF/BSSDF)
  - BRDF - Reflectance
  - BTDF - Transmittance

BSDF

$$f(x, \psi_i, \psi, \lambda) = \frac{dL(x, \psi, \lambda)}{L_i(x, \psi_i, \lambda)\, d\vec{\omega}_i\, cos\theta_i}$$

where:

$f(x, \psi_i, \psi, \lambda)$	=	BDF of the surface at x,
$dL(x, \psi, \lambda)$	=	radiance propagated at x and in a direction ψ,
$L_i(x, \psi_i, \lambda)$	=	incident radiance at x and in a direction ψ_i,
θ_i	=	angle between the surface normal at x_i and the direction ψ_i,
$d\vec{\omega}_i$	=	differential solid angle at which L_i arrives at x.

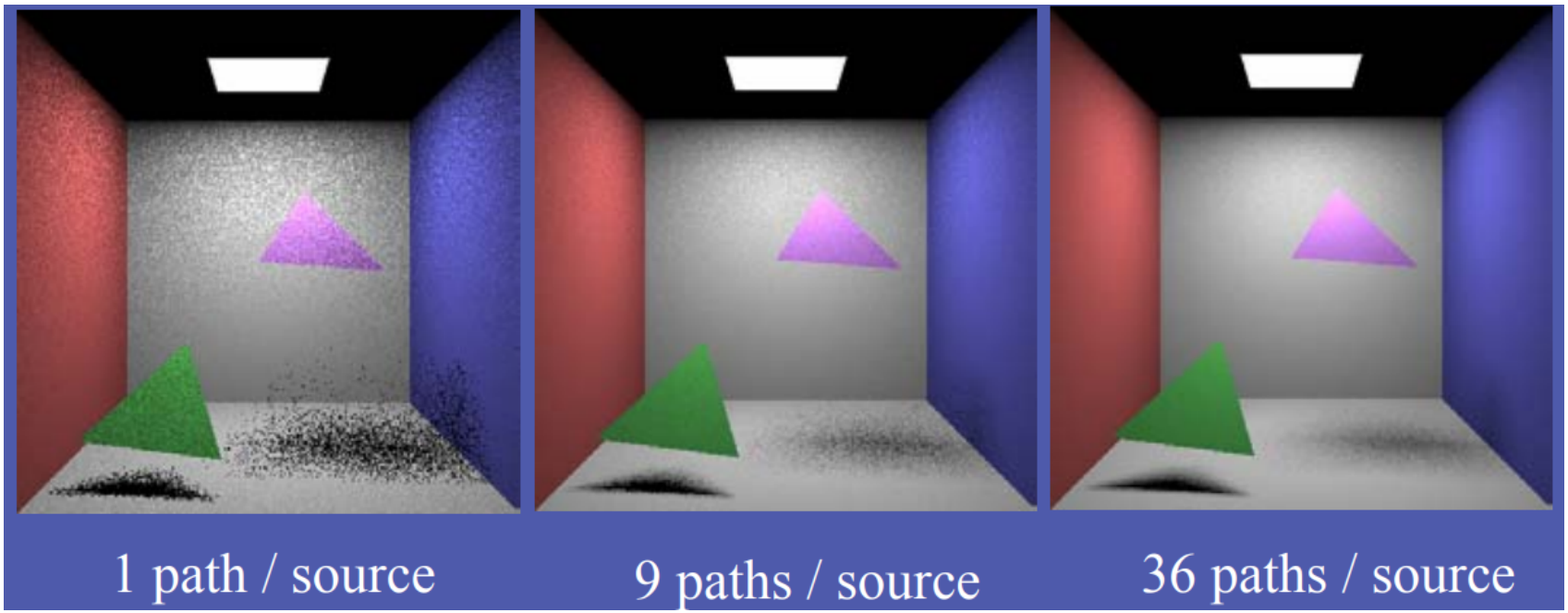
$$L_{\mathrm{o}}(X,\hat{\omega}_{\mathrm{o}}) = L_{\mathrm{e}}(X,\hat{\omega}_{\mathrm{o}}) + \int_{\mathbf{S}^2} L_{\mathrm{i}}(X,\hat{\omega}_{\mathrm{i}})\, f_X(\hat{\omega}_{\mathrm{i}},\hat{\omega}_{\mathrm{o}})\, |\hat{\omega}_{\mathrm{i}} \cdot \hat{n}|\, d\hat{\omega}_{\mathrm{i}}$$

Outgoing light Emitted light Incoming light Material Lambert

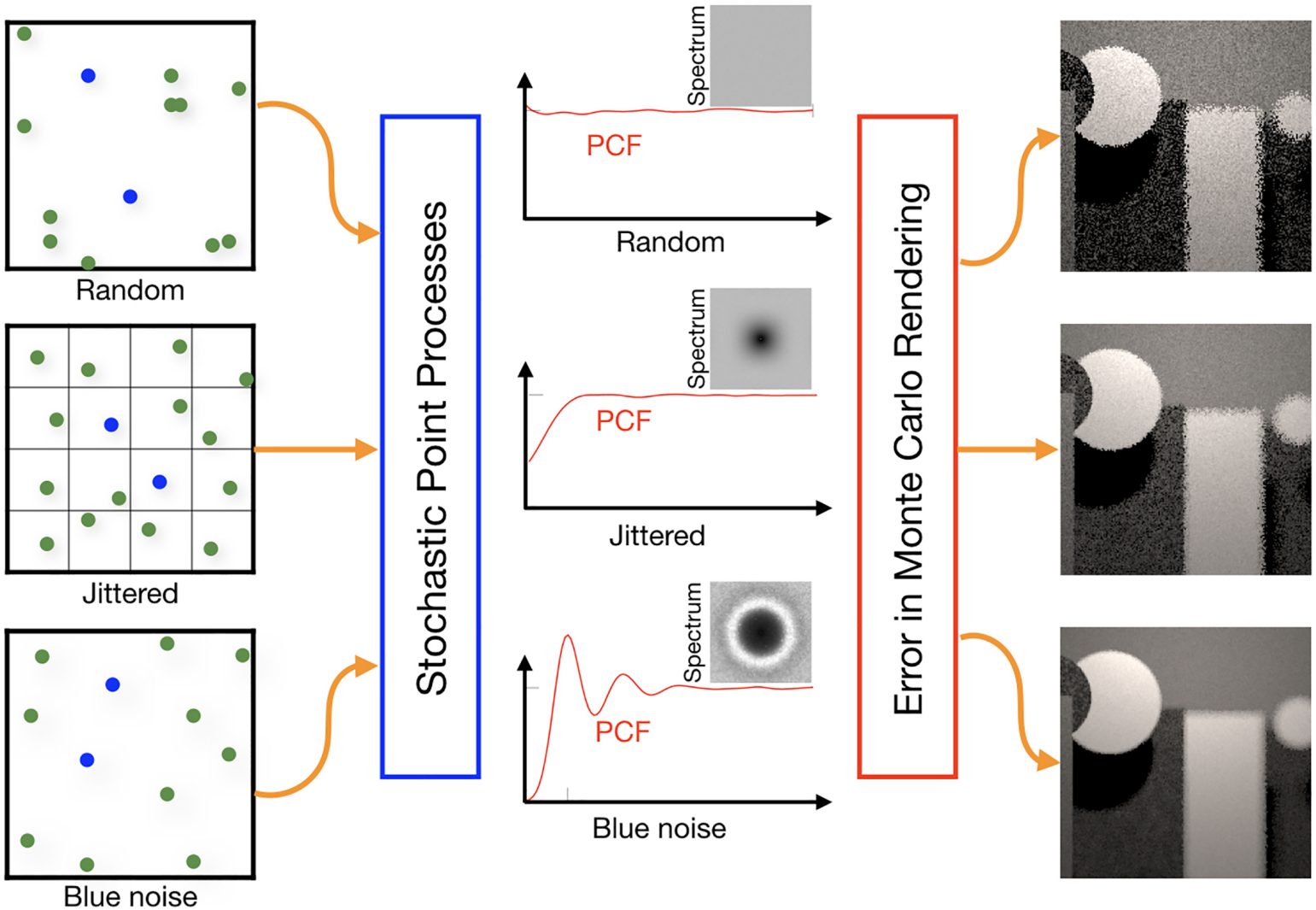
Monte-Carlo rendering

- Monte Carlo integration: uses sampling to estimate the values of integrals It only estimate the values of integrals. It only requires to be able to evaluate the integrand at arbitrary points making it arbitrary points, making it easy to implement and applicable to many problems

Monte-Carlo rendering



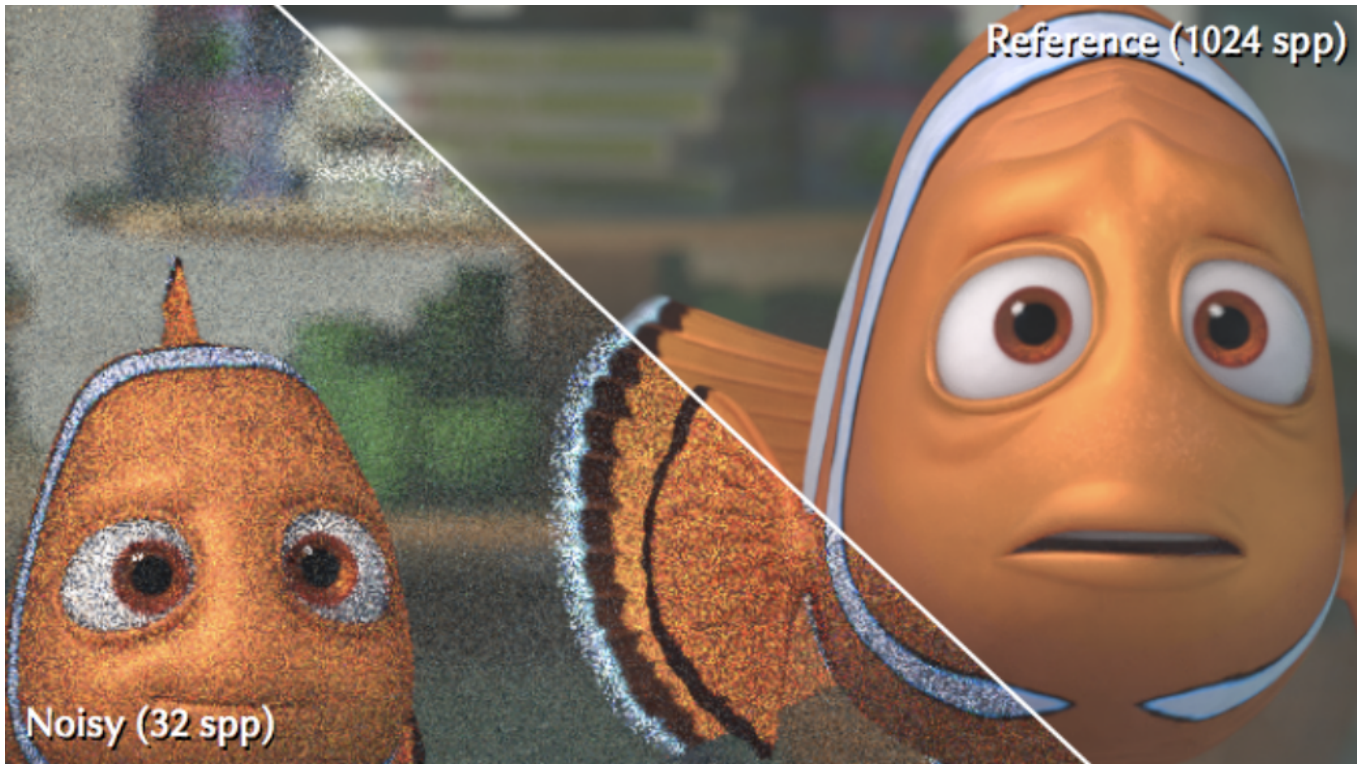
Monte-Carlo rendering



Monsters University

- 5.5M pieces of animated hair
- 29 hours to render a single frame
- 100 million CPU hours to render the film in its final form

Noise removal

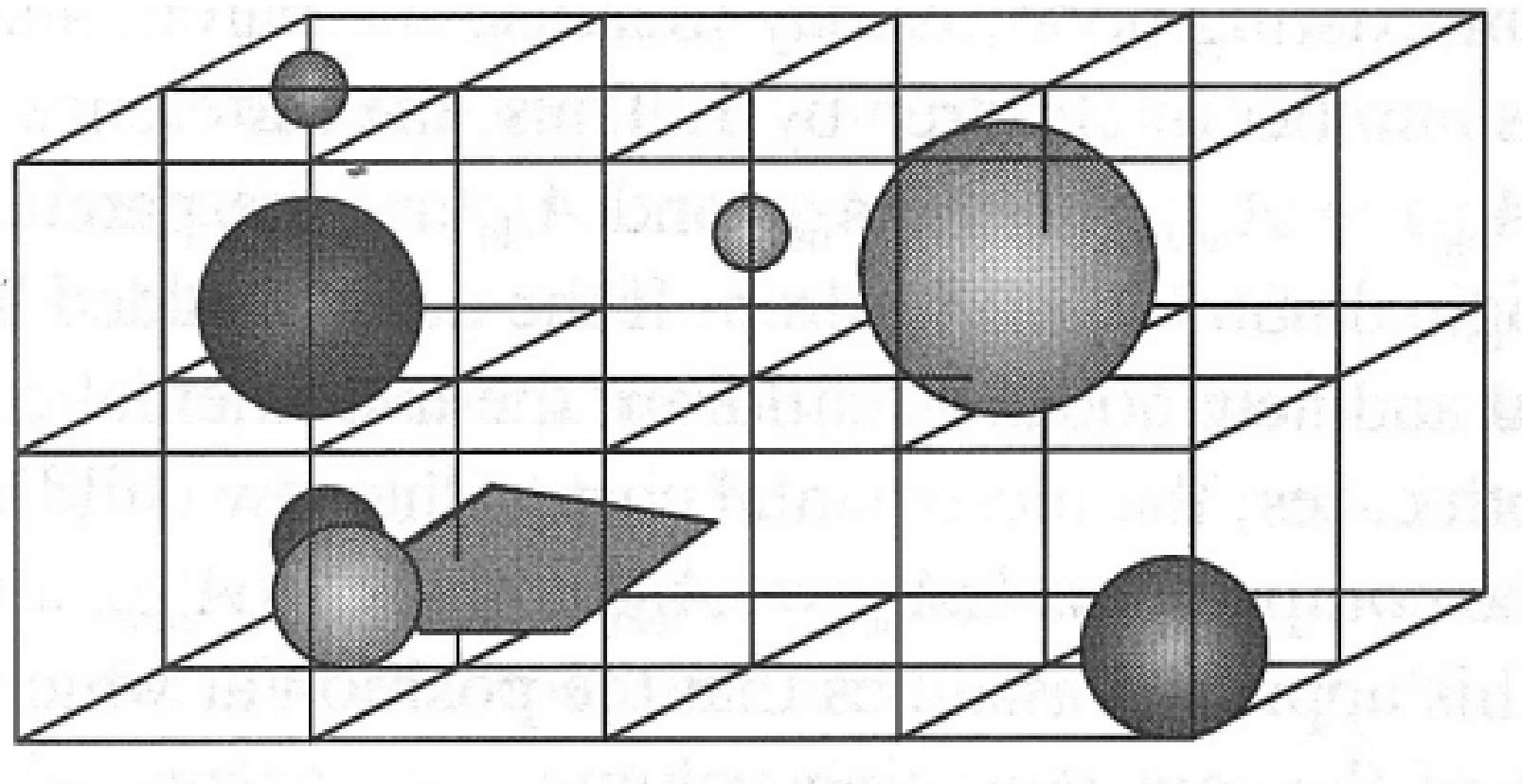


- Denoising with Kernel Prediction and Asymmetric Loss Functions
- Kernel-predicting Convolutional Networks for Denoising Monte Carlo Renderings

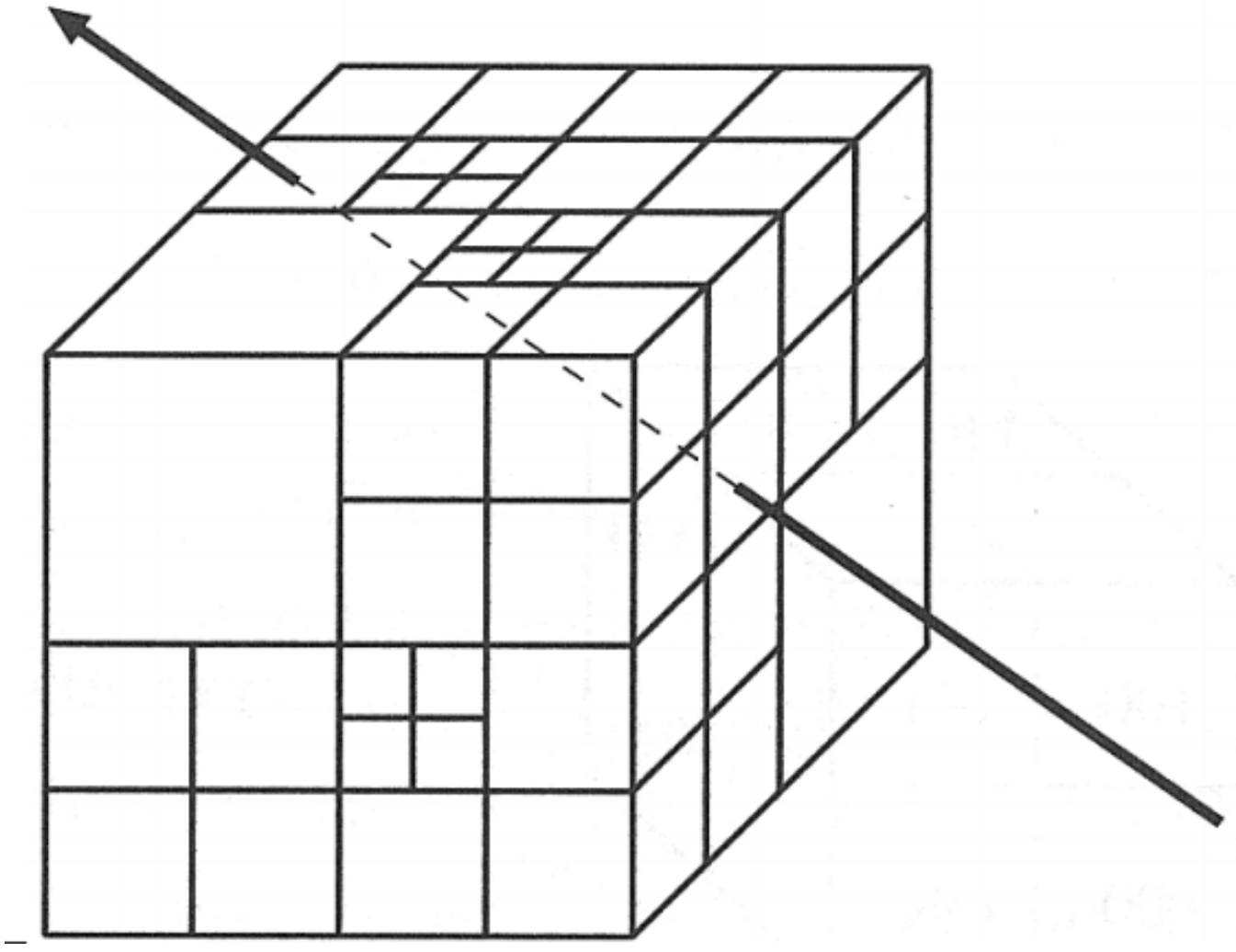
Space partitioning

- Grid
- Octree
- Bounding Volumes
- Bounding Volume Hierarchies

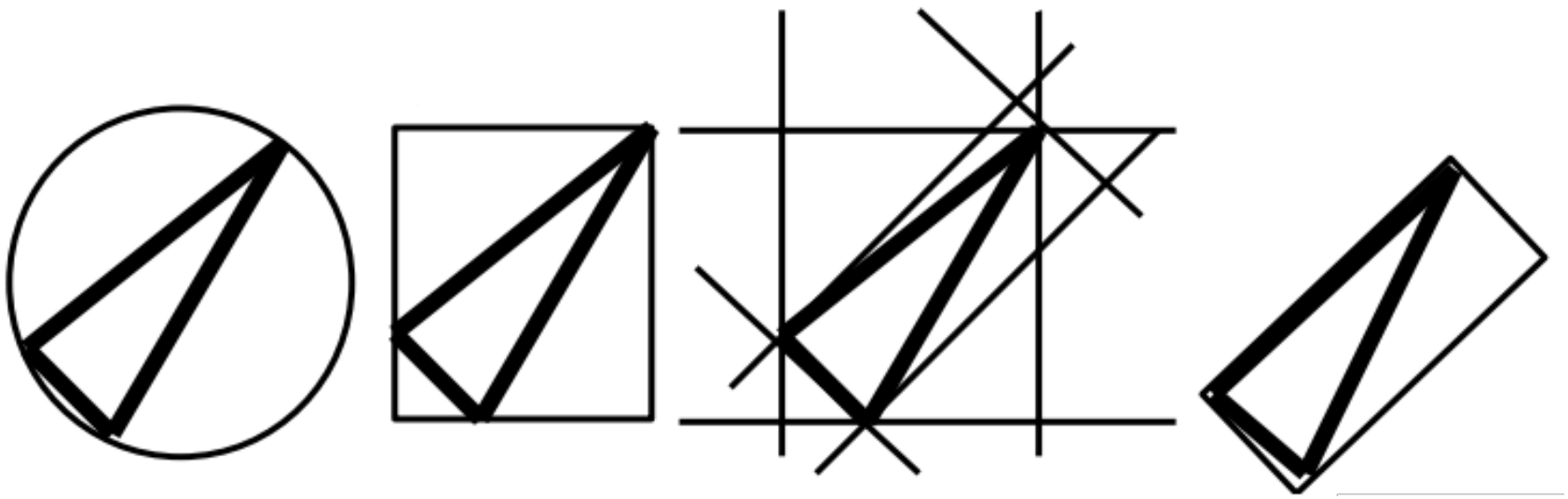
Grid



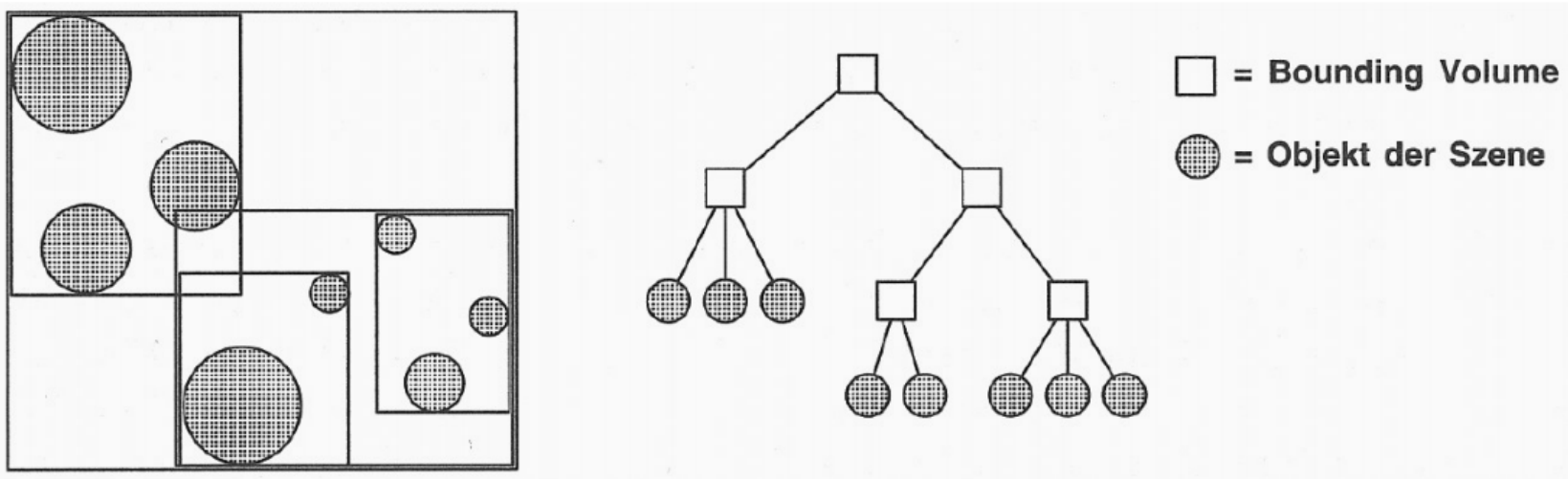
Octree



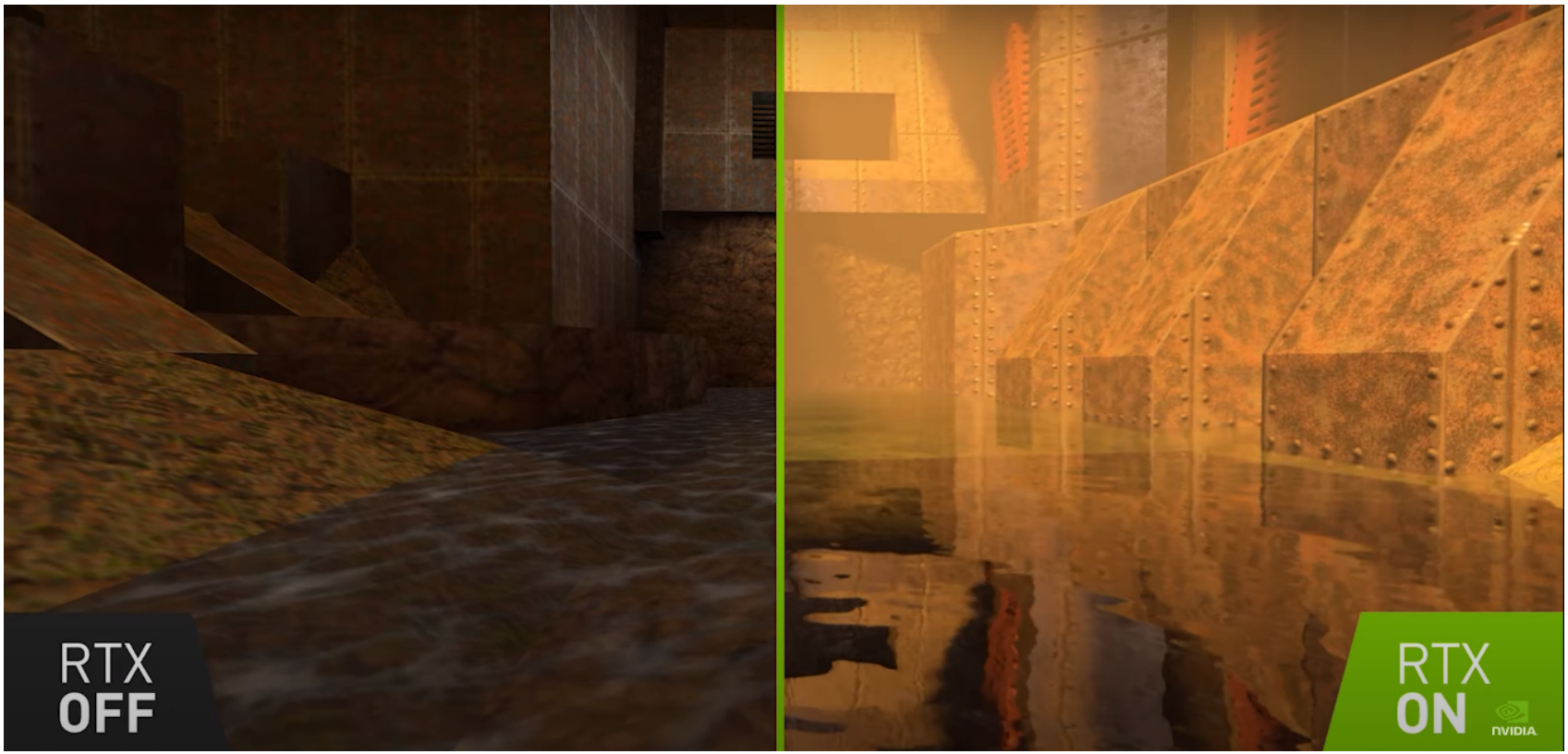
Bounding Volumes



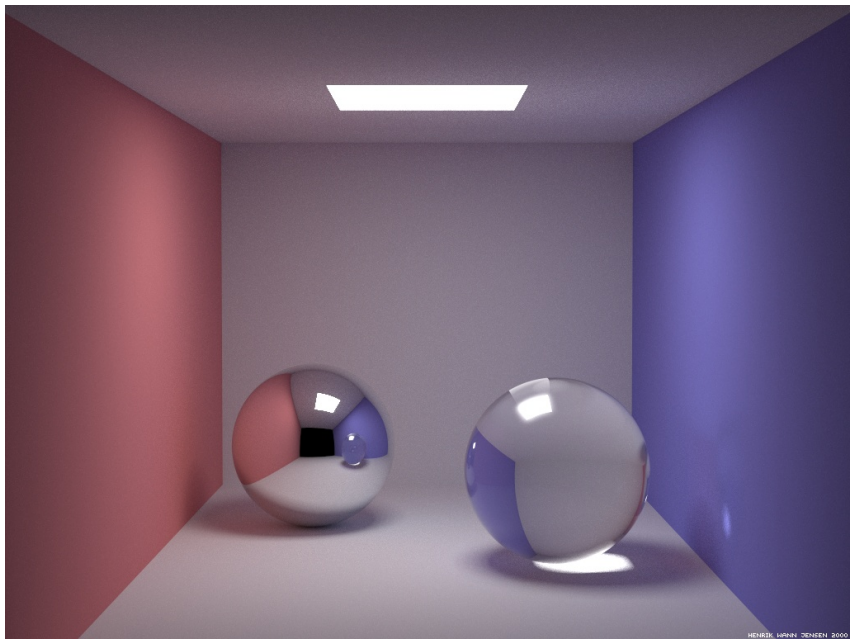
Boundng Volume Hierarchies



RTX



Lab Assignment 3



- Weightage - 40 marks
- Scene should contain at least 3 spheres (one textured, one glossy, one plastic) + its shadows [10 marks each], if possible, a glass sphere as well
- Free to use any related(OpenGL) libraries
- Deadline - 11th midnight

Next class

- 2^{nd} November 9 to 10
- Topic : Introduction to animation
- Need extension for projects?