
# Tracing and Ray Tracing

CS535

Daniel G. Aliaga
Department of Computer Science
Purdue University

# Ray Casting and Ray Tracing


- Ray Casting
  - Arthur Appel, started around 1968…
- Ray Tracing
  - Turner Whitted, started around 1980…

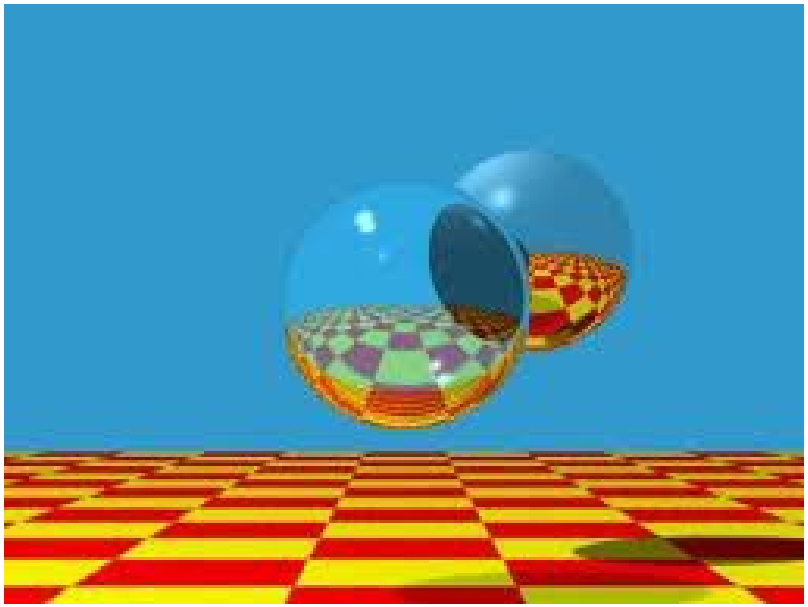
# Ray Tracing


- Turner Whitted, Communications of the ACM, 23(6), 343-349, June 1980

- **ABSTRACT**
  "To accurately render a two-dimensional image of a three-dimensional scene, global illumination information that affects the intensity of each pixel of the image must be known at the time the intensity is calculated. In a simplified form, this information is stored in a tree of "rays" extending from the viewer to the first surface encountered and from there to other surfaces and to the light sources. A visible surface algorithm creates this tree for each pixel of the display and passes it to the shader. The shader then traverses the tree to determine the intensity of the light received by the viewer. Consideration of all of these factors allows the shader to accurately simulate true reflection, shadows, and refraction, as well as the effects simulated by conventional shaders. Anti-aliasing is included as an integral part of the visibility calculations. Surfaces displayed include curved as well as polygonal surfaces."

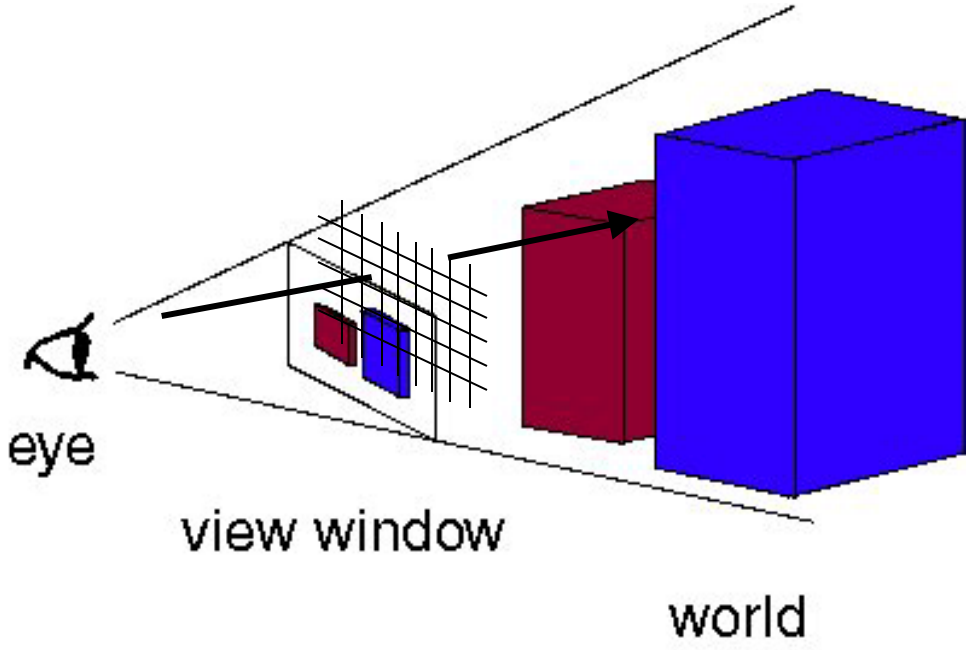
# Ray Tracing

- Turner Whitted, Communications of the ACM, 23(6), 343-349, June 1980



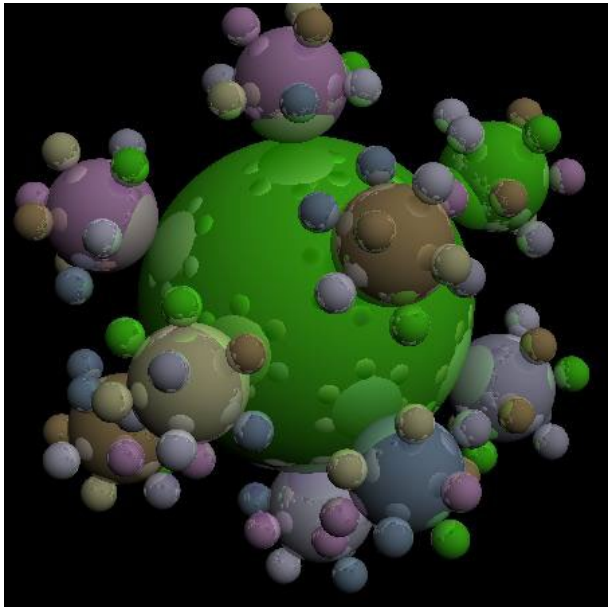
# Ray Casting

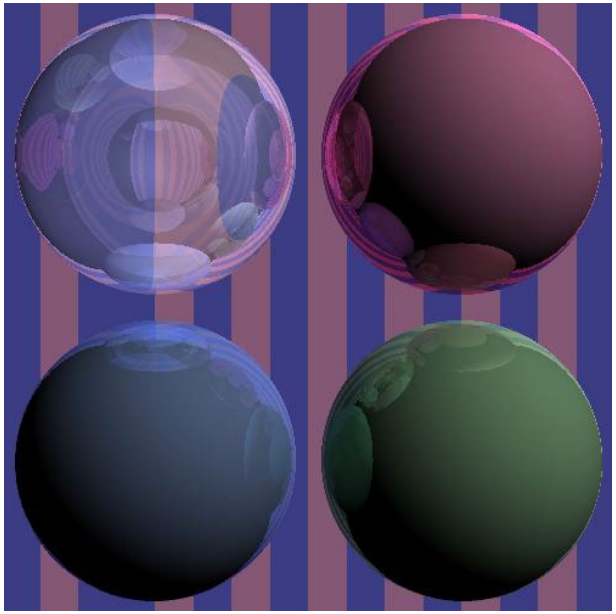
- To “cast” a ray for each pixel into the scene and color the ray based on which object it “hits”
  - This can be interpreted as the inverse of the feed-forward graphics pipeline which effectively “pushes” geometry onto the screen



# Ray Tracing

- In addition to ray-casting, rays can reflect off objects, reflect, refract, identify objects in shadow, and numerous other effects…





# Ray Tracing

- In addition to ray-casting, rays can reflect off objects, reflect, refract, identify objects in shadow, and numerous other effects…

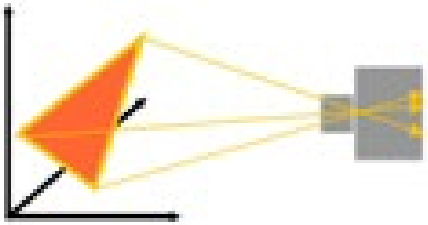
ray trace example

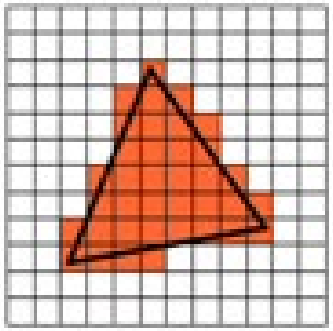

using CUDA/GPU programming

# But there is much more

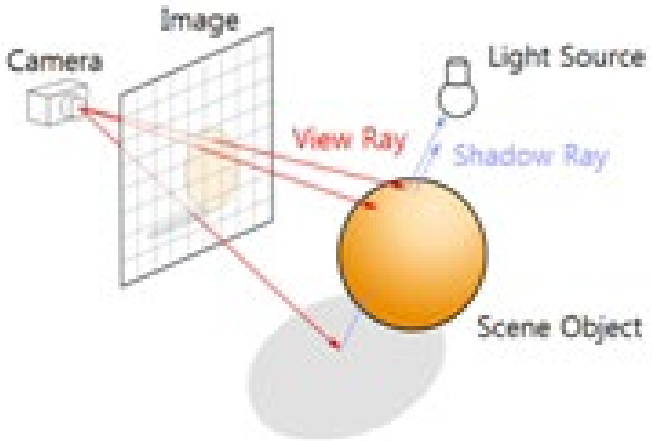

- Path Tracing
- Beam Tracing
- Cone Tracing

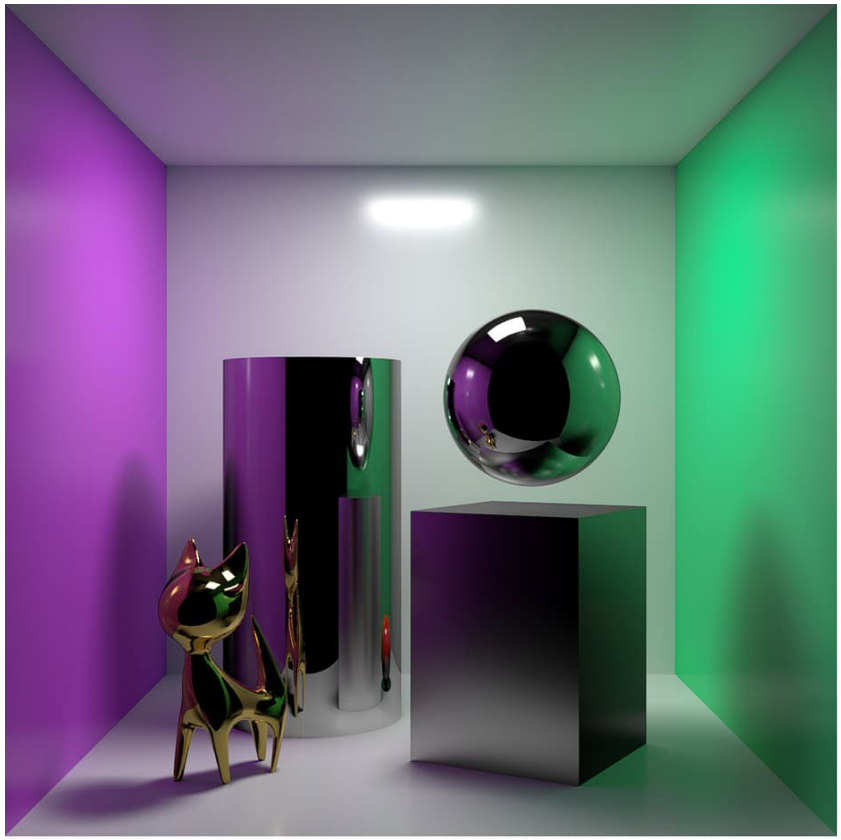
## Rasterization



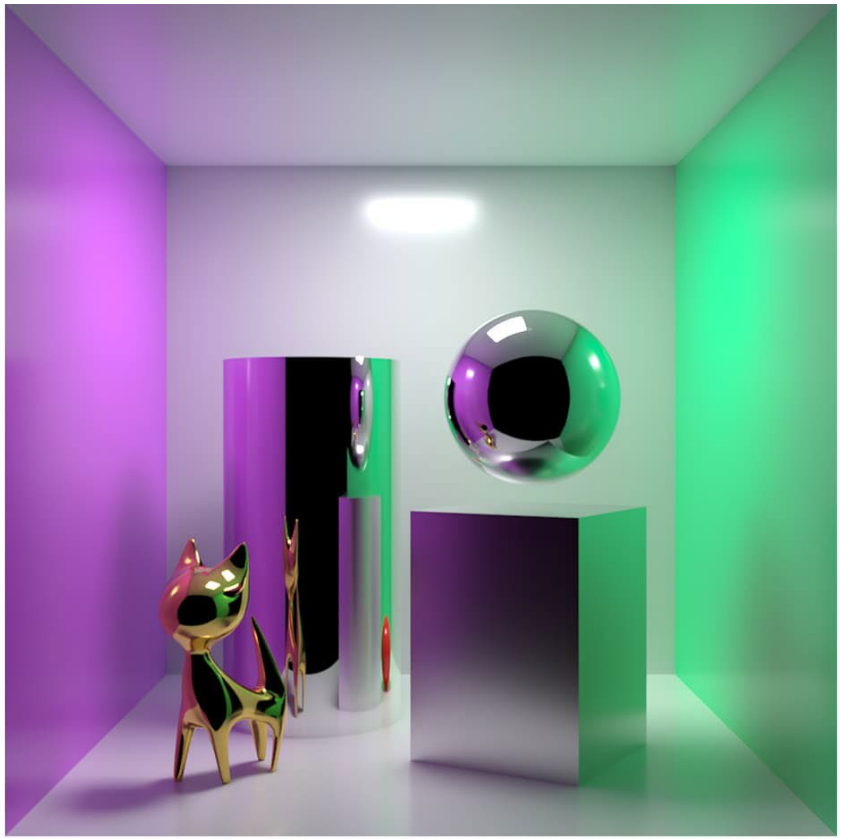


## Ray Tracing

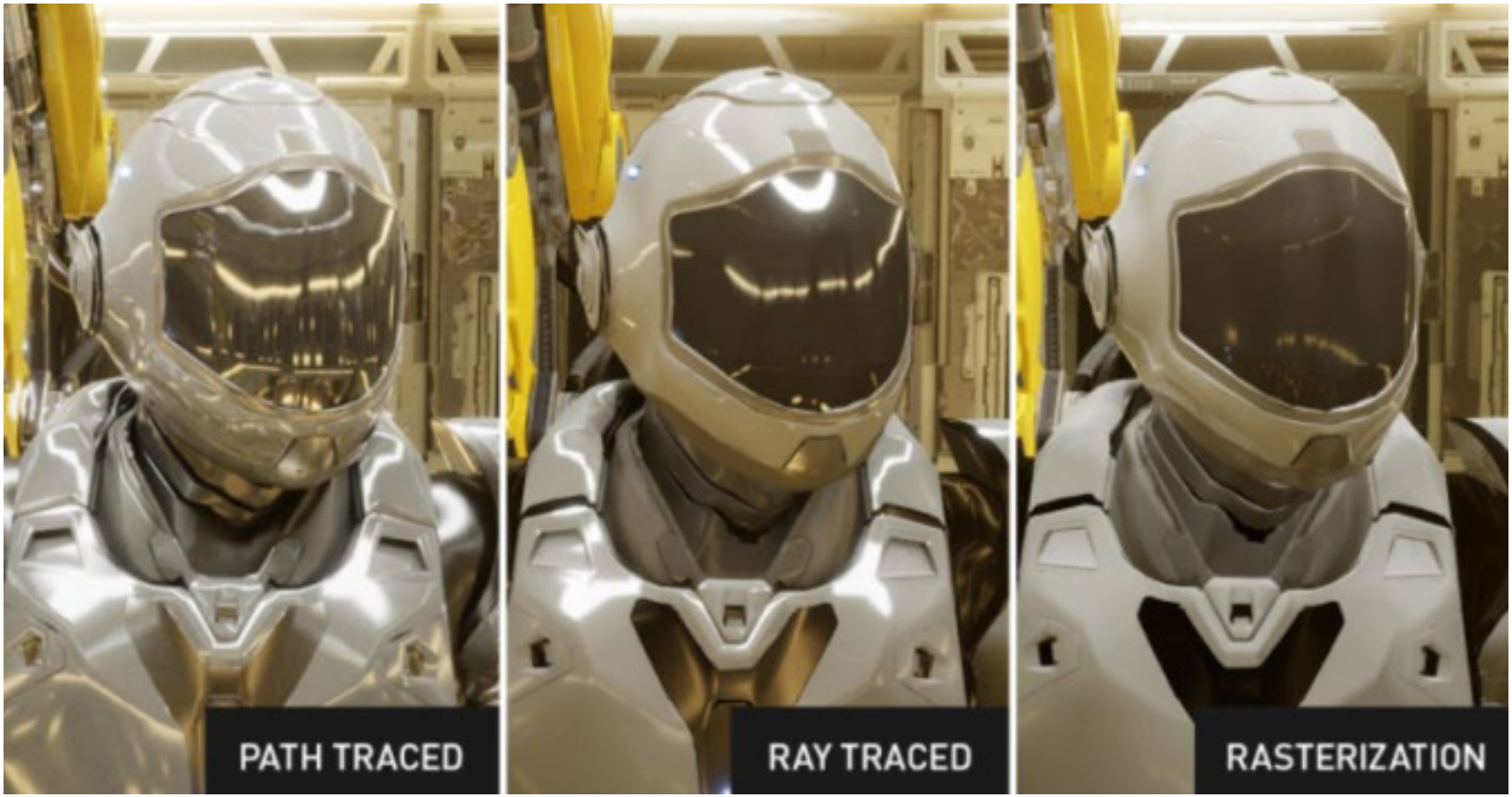


Ray Tracing

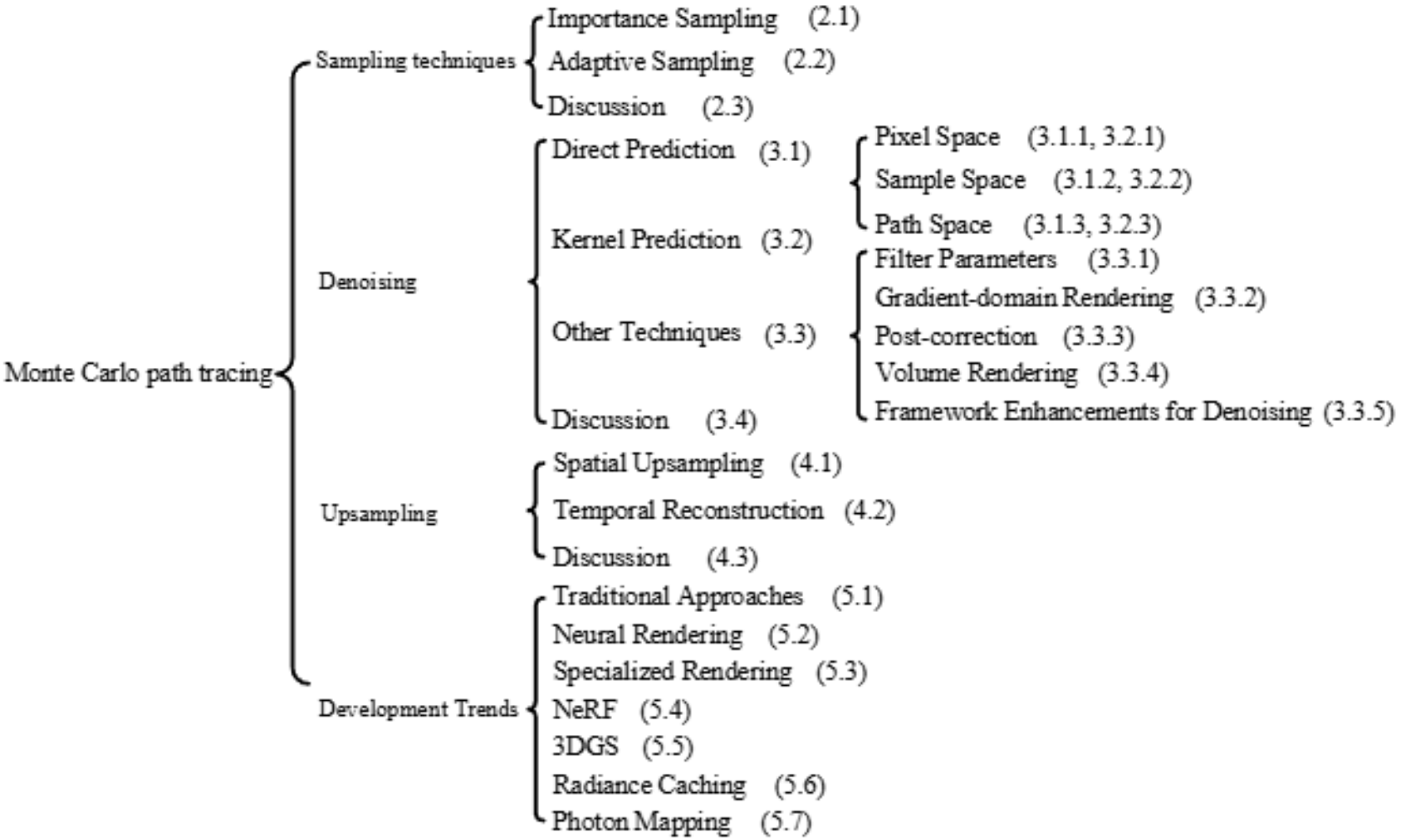


Path Tracing


PATH TRACED
RAY TRACED
RASTERIZATION

Ray Tracing: Comparable Time
With Jittering
Without Jittering
5 Seconds / 9 Shadow Rays
9 Seconds / 25 Shadow Rays
1.84 Seconds / 4 Shadow Rays
3 Seconds / 4 Shadow Rays
Beam Tracing: Reference Quality
5 Seconds
9 Seconds
1.78 Seconds
3 Seconds
Ray Tracing: Comparable Quality
256 Shadow Rays
108 Seconds
81 Seconds
79 Seconds
99 Seconds

# A Survey on Deep Learning for Monte Carlo Path Tracing, Yan et al. 2025



# Ray Tracing


- Fluid Animation (GTX):
  - https://www.youtube.com/watch?v=XleO15ldvBQ
- Star Wars (RTX):
  - https://www.youtube.com/watch?v=J3ue35ago3Y

# Ray Tracers


- POVRay
  - http://www.povray.org
  - https://github.com/POV-Ray/povray
- Compact:
  - https://mzucker.github.io/2016/08/03/miniray.html
- Others:
  - https://en.wikipedia.org/wiki/List_of_ray_tracing_software

# Basic Ray Tracing Algorithm


- Setup image plane
  - Center-of-projection
  - Field-of-view
- Define objects
  - Functional descriptions
    - E.g., sphere, box, f(x,y,z)=0
  - Polygonal descriptions
- Foreach pixel…

# Basic Ray Tracing Algorithm

**Foreach pixel**

**define ray from eye through pixel and into scene**

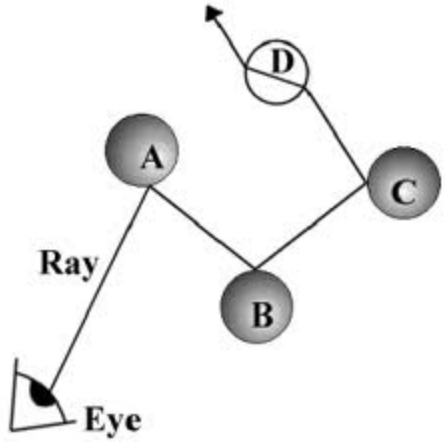
**get intersection with scene**

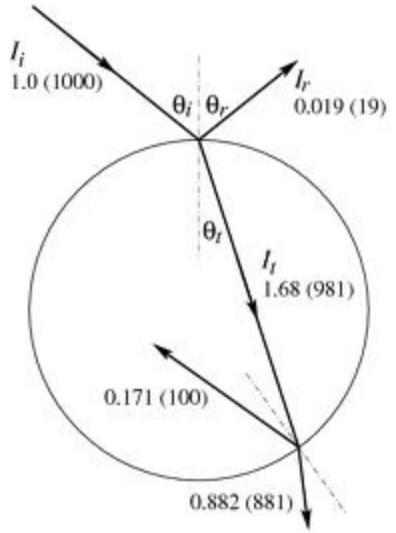
**trace reflected ray…**

**trace refracted ray…**

**trace to-light ray…**

**combine resulting colors…**



# Ray tracing

- Ray tracing (intersection computations)
  - Sphere, plane, polygon, box, quadric
- Reflections
- Refractions
- Shadows

(more on the board…)

# Ray Object Intersections


- A ray is defined as

$R_0 = [x_0, y_0, z_0]$ - origin of ray

$R_d = [x_d, y_d, z_d]$ - direction of ray

Then, a set of points on the ray are defined as

$R(t) = R_0 + R_d * t$ where $t > 0$

If $R_d$ is normalized, then t equals the distance of the ray from origin in World Coordinates,

*Slides: Vivek Puri*

# Ray Sphere Intersection

- Sphere is defined by its center ($S_c = [x_c, y_c, z_c]$) and radius $S_r$

  Thus, the equation of sphere is

$$(x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2 = S_r^2$$

  Substitute equation of ray and solve quadratic eqn for **t**,

  – If discriminant $< 0 \Rightarrow$ No intersection

  – If discriminant $\geq 0 \Rightarrow$ Intersection. Smaller **t** gives the nearest intersection point

*Slides: Vivek Puri*

# Ray Plane Intersection

- Plane is defined by Ax+By+Cz+D=0

  Substitute the equation for ray and solve for **t**,

  $\mathbf{t} = -(AX_0 + BY_0 + CZ_0 + D) / (AX_d + BY_d + CZ_d)$

  $= -(\mathbf{P}_n \cdot \mathbf{R}_0 + D) / (\mathbf{P}_n \cdot \mathbf{R}_d)$

  where $\mathbf{P_n}$ is the plane normal of unit length

  If $\mathbf{P}_n \cdot \mathbf{R}_d = 0 \Rightarrow$ Ray is parallel to the plane

  If $\mathbf{P}_n \cdot \mathbf{R}_d > 0 \Rightarrow$ Normal of plane points away from the ray

  If $\mathbf{t} < 0 \Rightarrow$ Ray intersects plane behind origin

  If $\mathbf{t} > 0 \Rightarrow$ Substitute **t** in equation for intersection point

*Slides: Vivek Puri*

# Ray Polygon Intersection


- Check whether the ray intersects the polygon plane
- Calculate the point of intersection
- Check whether the point is within the polygon. **How?**
  - Cross products
  - Shoot a ray from the point and count edges crossed, if odd then inside
  - or one of several other approaches

*Slides: Vivek Puri*

# Ray Box Intersection

- Transform box vertices ($\mathbf{V_i}$) to align with the coordinate axes
- Solve for intersection with $x_{min}$ and $x_{max}$ planes; similarly solve for Y and Z.
- Check for common interval or intersection

If, $\max(t_{min}) \leq \min(t_{max})$

Then, $t_0 = \max(t_{min})$

$t_1 = \min(t_{max})$

*Slides: Vivek Puri*

# Ray Quadric Intersection


- Class of quadrics (surfaces that can be defined by a quadratic equation) include cylinders, cones, ellipsoids, paraboloids, etc

- The general quadric surface equation is

  $Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J = 0$

- Substitute the equation of ray, we get the form,

  $A_q t^2 + B_q t + C_q = 0$

- Solve for **t** using quadratic formula

*Slides: Vivek Puri*

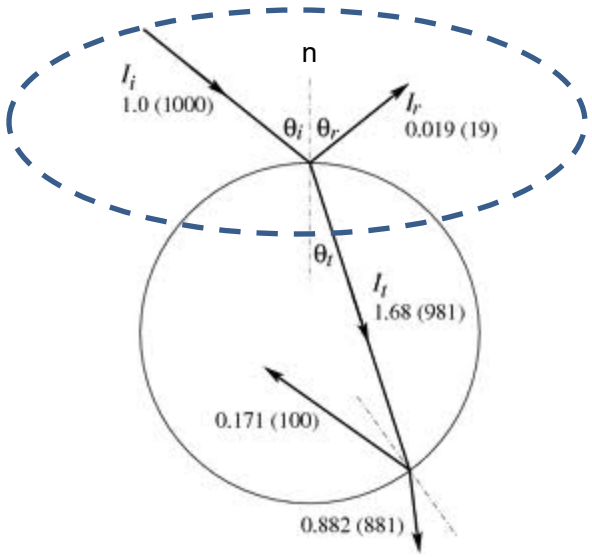
# Ray tracing


- Ray tracing (intersection computations)
- Reflections
- Refractions
- Shadows

(more on the board…)

# Reflections

- Similar to what we saw for shading/illumination
- Recall $\theta_i = \theta_r$
- $l_r = l_i - 2n(n \cdot l_i)$



# Refraction (or Transmission)

- Need refraction indices:

  $k_1$=outside object (e.g., air)

  $k_2$=inside object (e.g., glass)

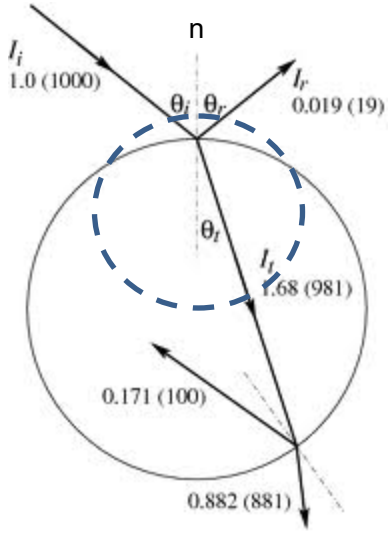
- Snell's Law:

$$k_1 \sin(\theta_i) = k_2 \sin(\theta_t)$$

- After some rearrangement:

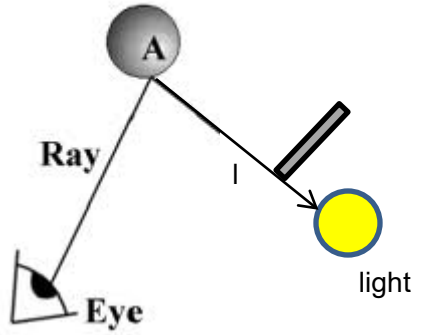
$$l_t = -n\cos(\theta_t) + b\sin(\theta_t)$$

where $b = n \times (n \times l_i)$



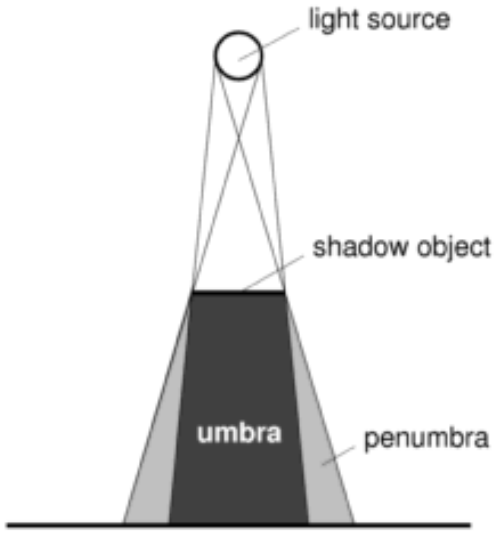
# (Hard) Shadows

- If ray $l$ is unoccluded from surface point to light source, then surface point is illuminated (i.e., not in shadow)
- Use same recursive rayTrace function but cast ray from surface point to light



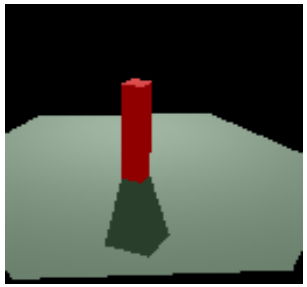
# Soft Shadows

- Lights are actually areas, so use area light sources
- Using areas enables both umbra and penumbra to appear

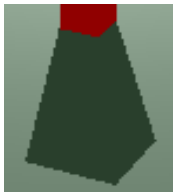


# Soft Shadows

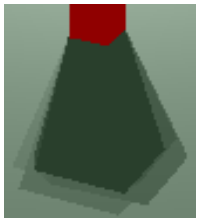
- Lights are actually areas, so use use area light sources
- *Distributed Ray Tracing*
  - Replace light with a collection of point light sources (e.g., up to 50 rays jittered samples)



Example Scene



1 light ray



Uniformly sampled light rays

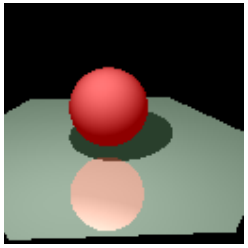
10 light rays

20 light rays

50 light rays

# Distributed Ray Tracing

- Soft shadows (previous slide)
- Glossy Reflections



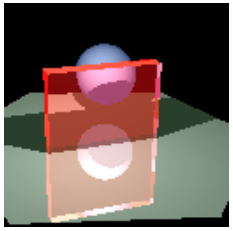
1 ray

10 rays

20 rays

50 rays

- Fuzzy Translucency

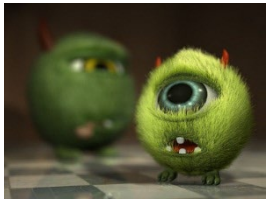


1 ray

10 rays

20 rays

- Defocus

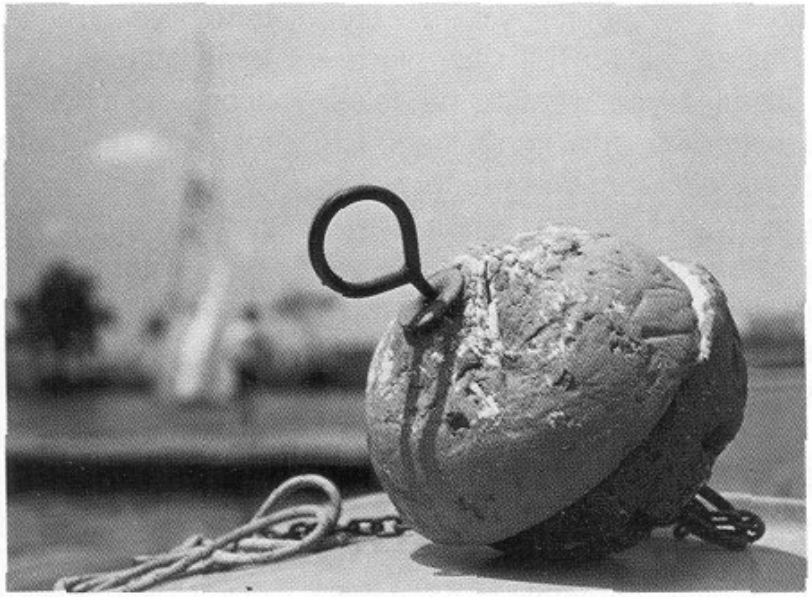



(examples using per-pixel and ray tracing logic)

# Depth of Field

- The area in front of your camera where everything looks sharp and in focus.
  - objects falling within that area will be acceptably-sharp and in focus;
  - objects falling outside the area will be soft and out of focus.

less depth of field

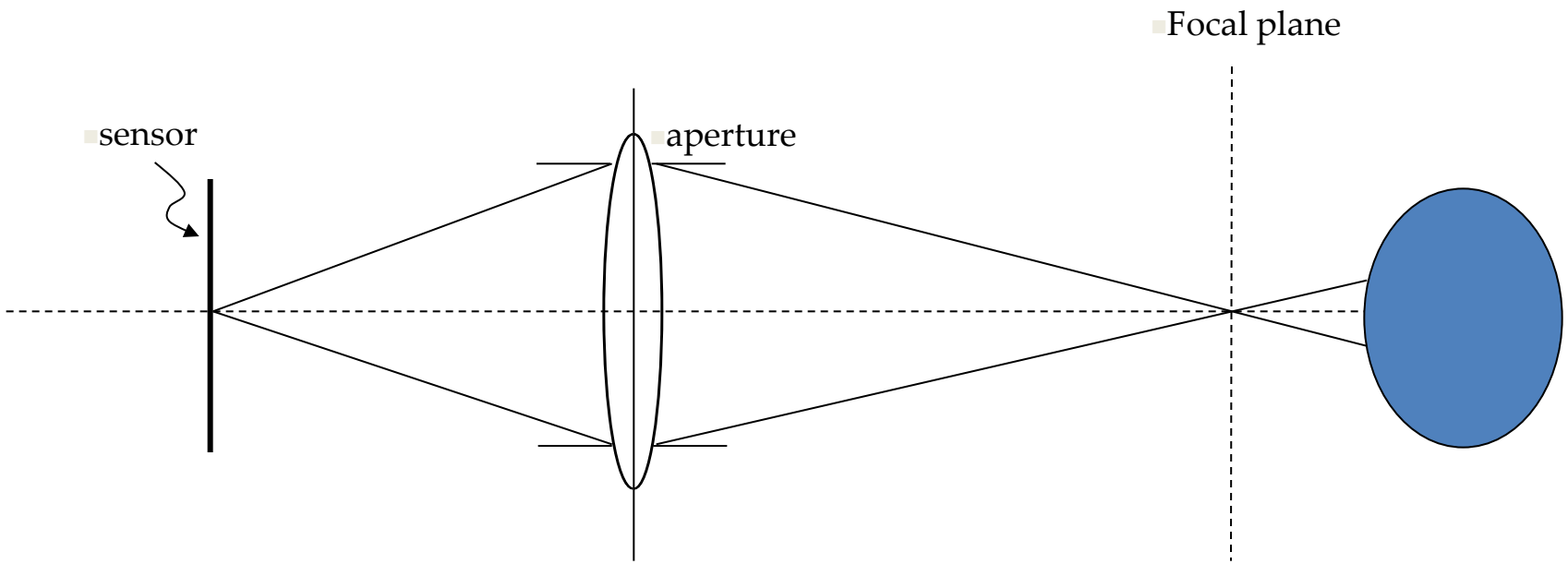


wider aperture

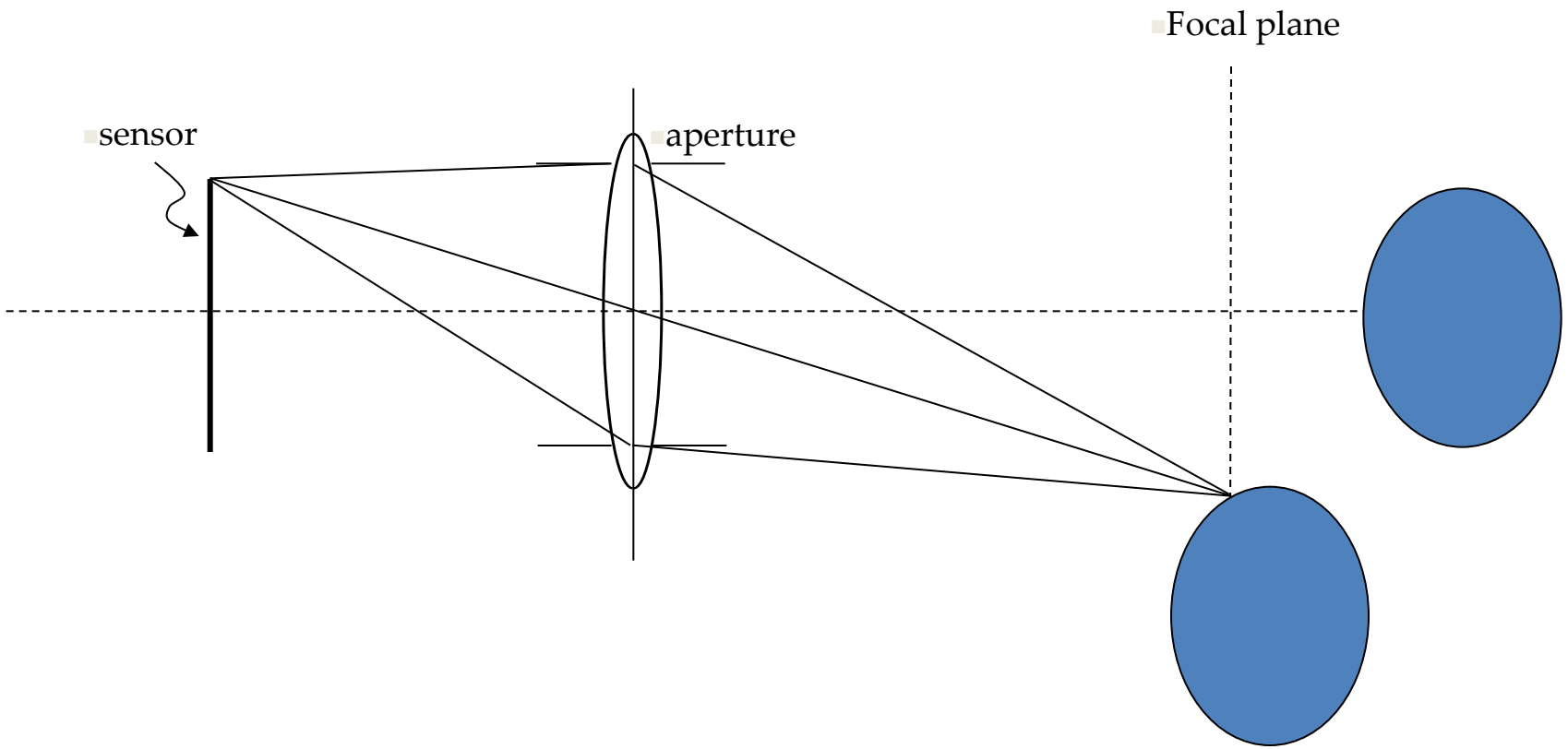
more depth of field

smaller aperture

# Depth of Field

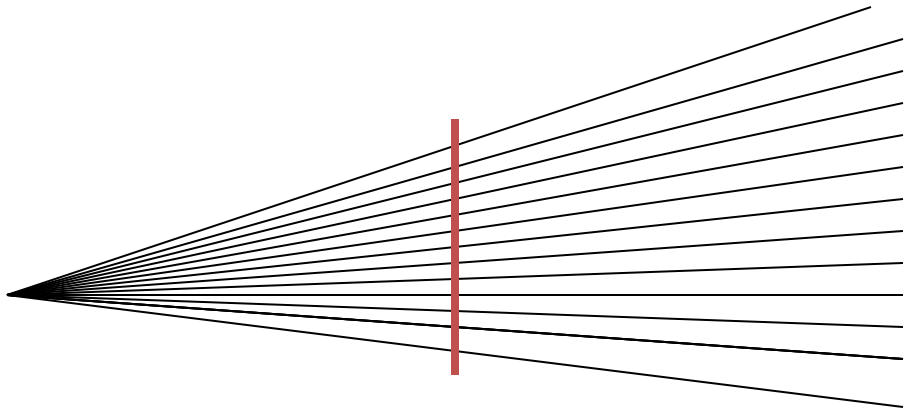


# Depth of Field



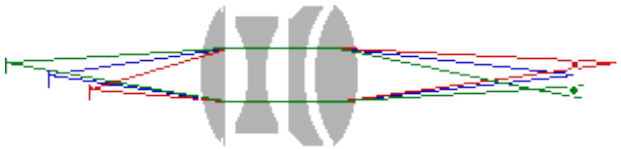
# Computer Graphics
# Camera Models

- Pinhole – ideal camera
- All rays go through single point
- Everything in focus -- unrealistic

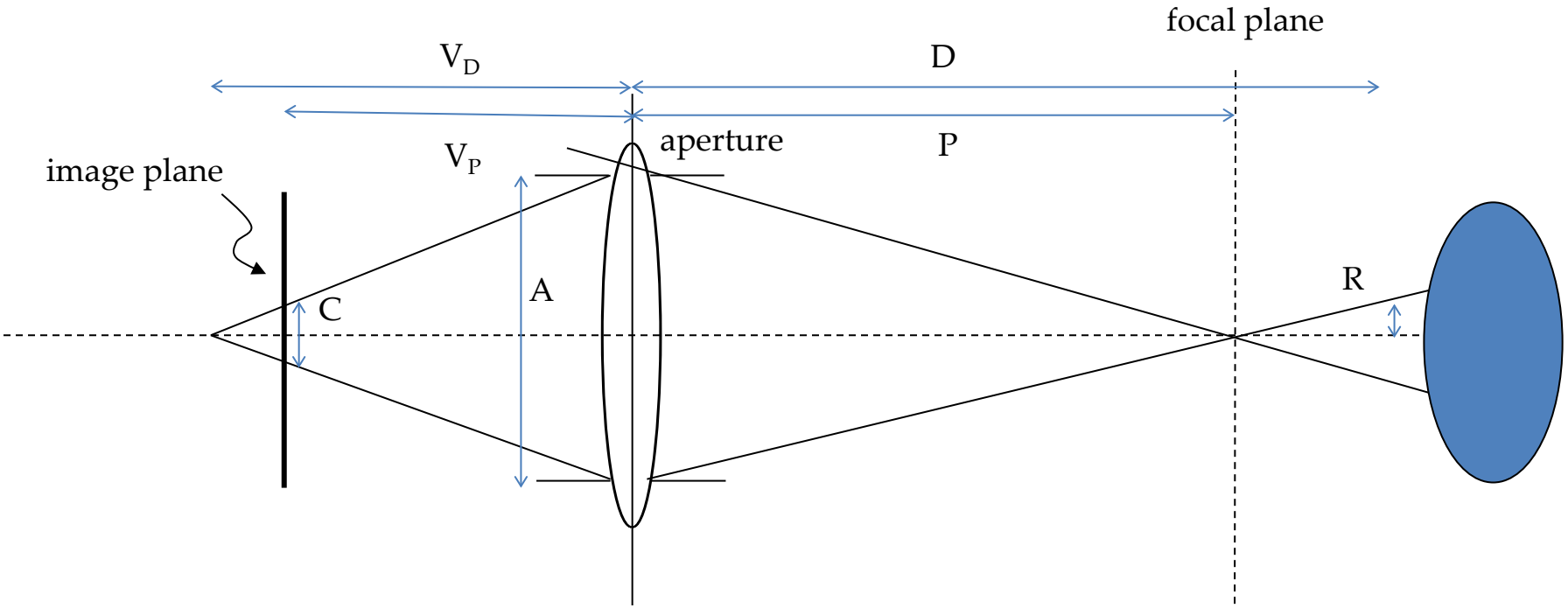


# More Realistic Model

- Lenses with spherical surfaces
- Depth of field control



# Depth of Field

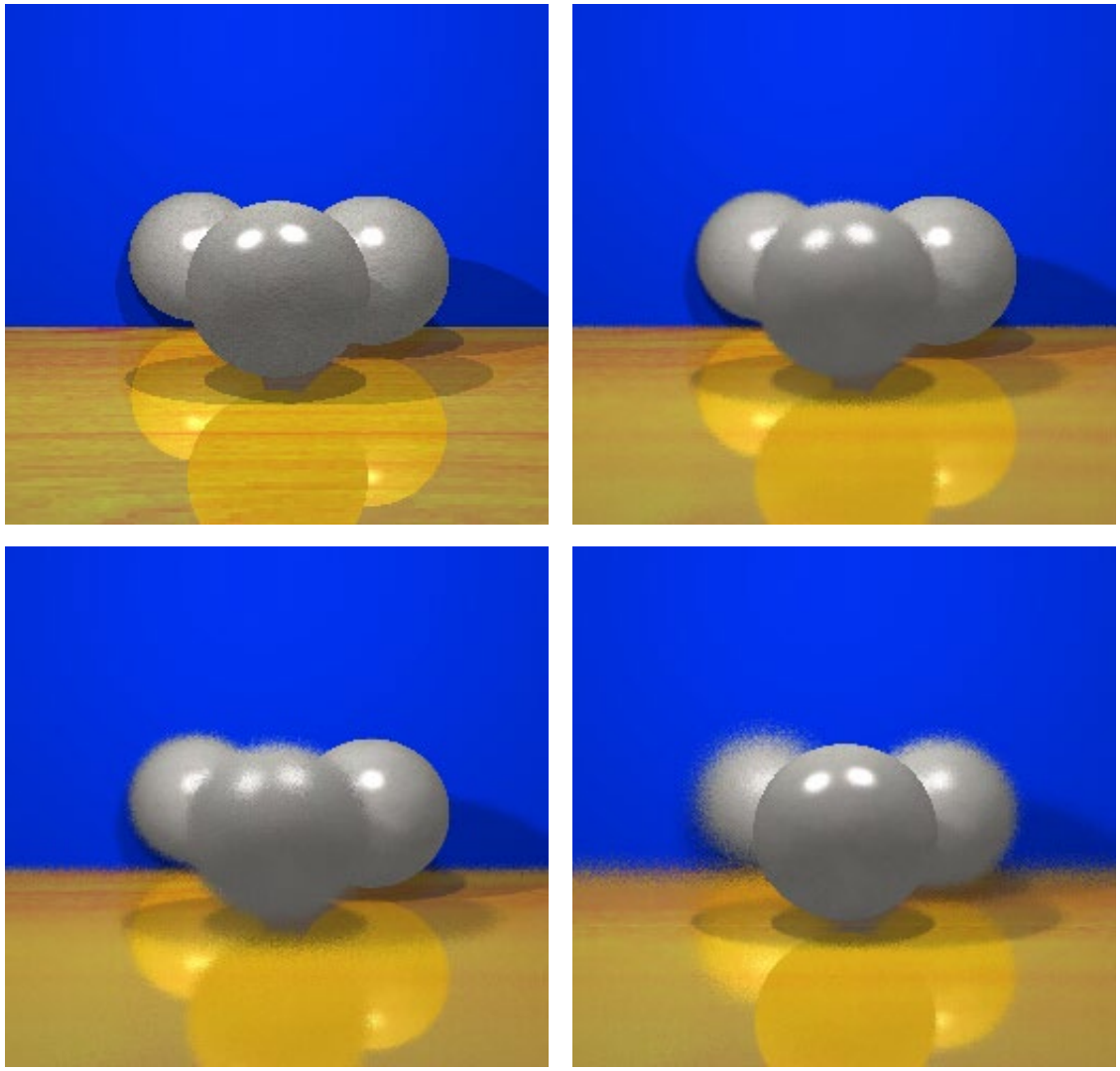



$V_P = AP/(P-A)$ for $P>A$ $\quad C = |V_D - V_P| \, (A/V_P)$ "=circle of confusion"

$V_D = AD/(D-A)$ for $D>A$ $\quad R = 0.5\, A\, (\, |D-P| \,/\, P\,)$

# Depth of Field

Example Results



# Depth of Field: Out of Focus Blur


- How to approximate without actually creating an entire physically based rendering system?
- Basic idea:
  - You want something at distance “P” to have its rays converge
  - So think backwards: how can you distort multiple rays per pixel so that they converge at distance P but not otherwise?