EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

FACULTY OF
SCIENCE
Computer Graphics

Computer Graphics (Graphische Datenverarbeitung)

- Ray Tracing I -

WS 2021/22

Hendrik Lensch

Overview

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Last Lecture
  - Introduction
  - Math Primer
- Today
  - Ray Tracing I
    - Background
    - Basic ray tracing
    - What is possible
    - Recursive ray tracing algorithm
- Next Lecture
  - Ray Tracing II: Spatial indices

Background

What do we want, what do we need?

Photorealistic Rendering

- Long standing goal in computer graphics
- Ingredients:
  - Camera model
  - Scene model
  - Illumination model
  - Rendering algorithm

Camera Model

- Perspective

Camera Model

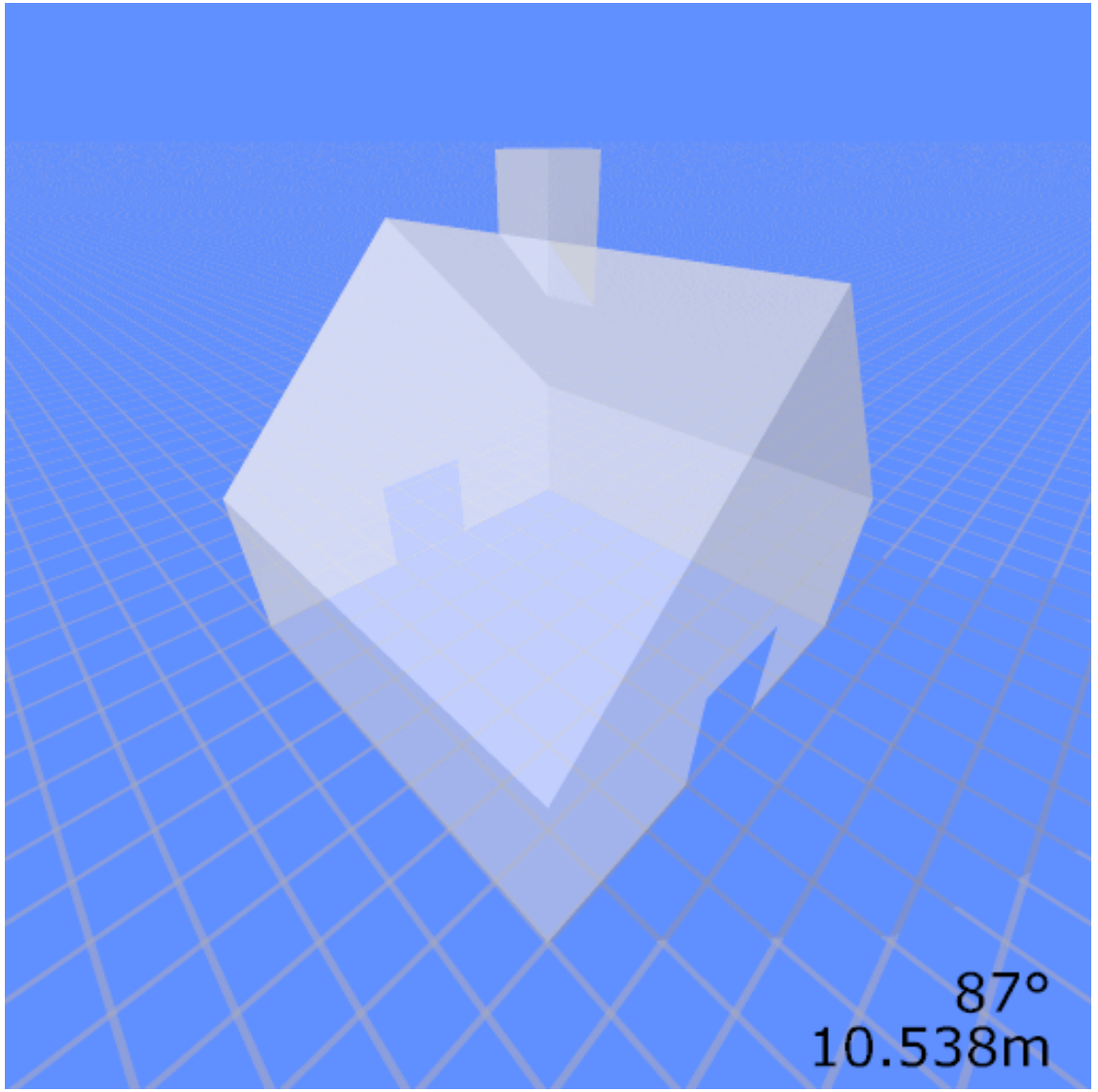
- Perspective

Camera Model

- Perspective

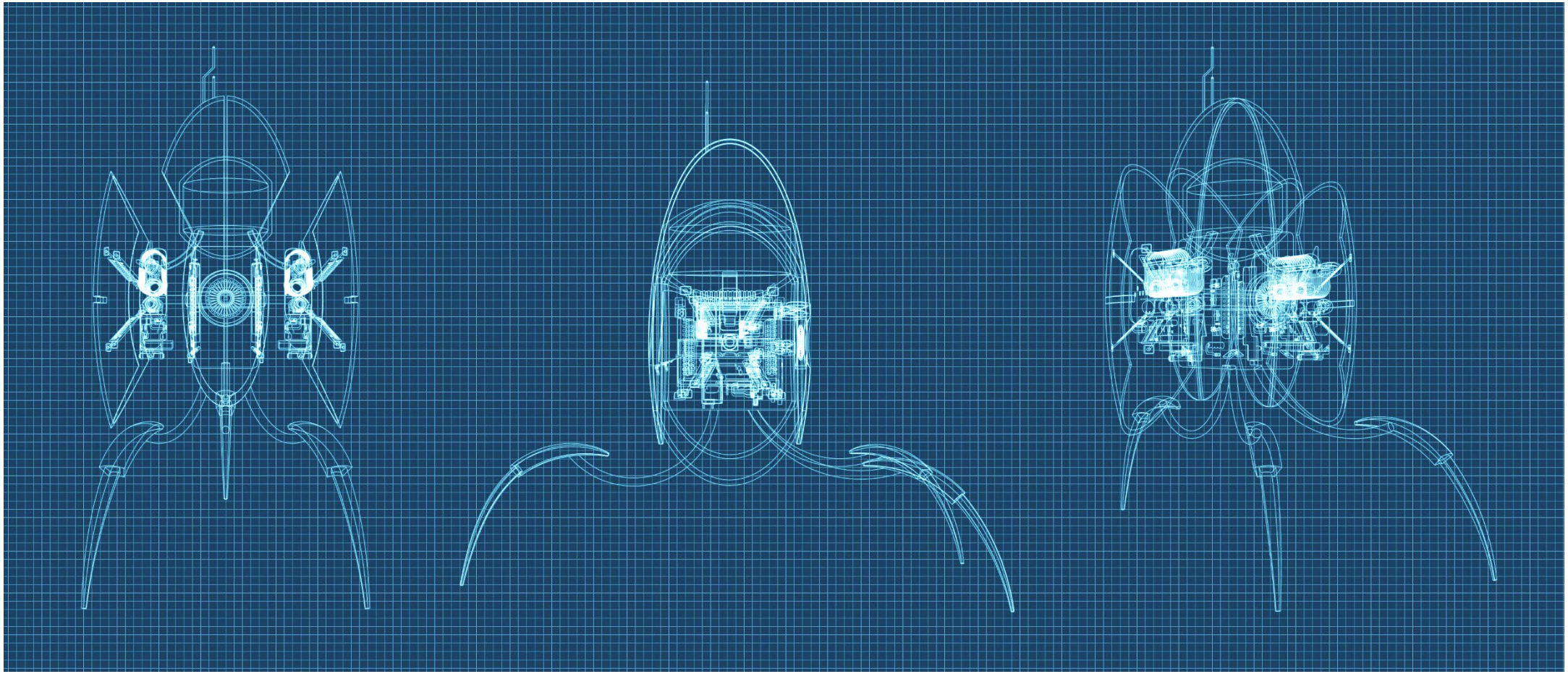
Camera Model

- Perspective projection
- Orthographic projection



Scene Model

- 3D geometry

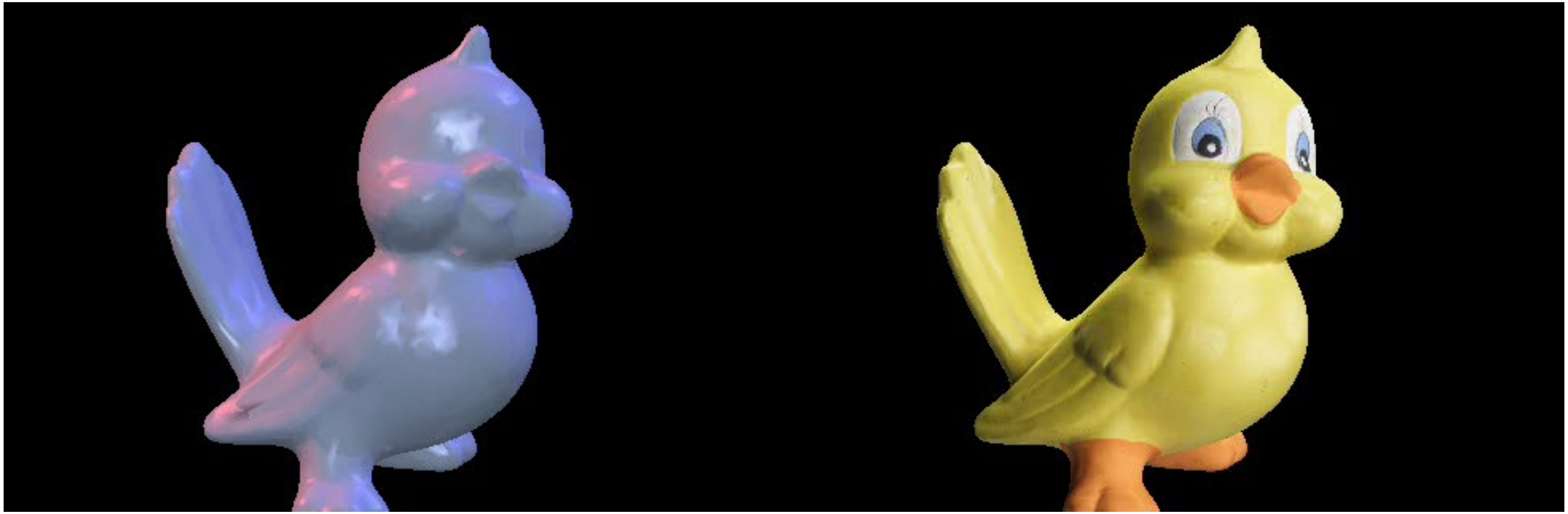


Scene Model

- 3D geometry
- Color
- Texture
- Material

Scene Model

- 3D geometry
- Color
- Texture
- Material

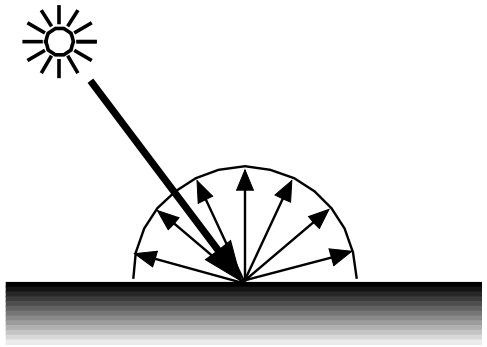


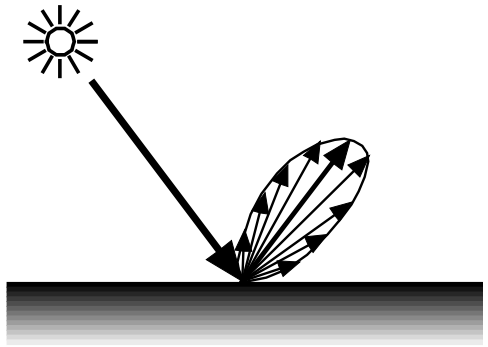
Illumination Model: Local Reflections

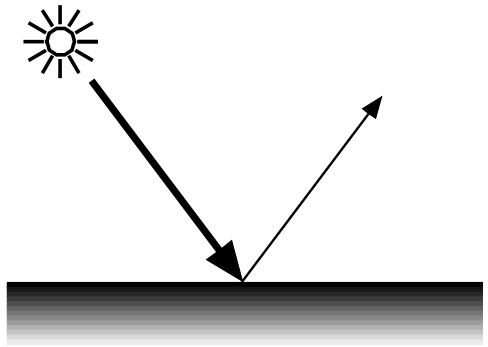
diffuse

glossy

mirror



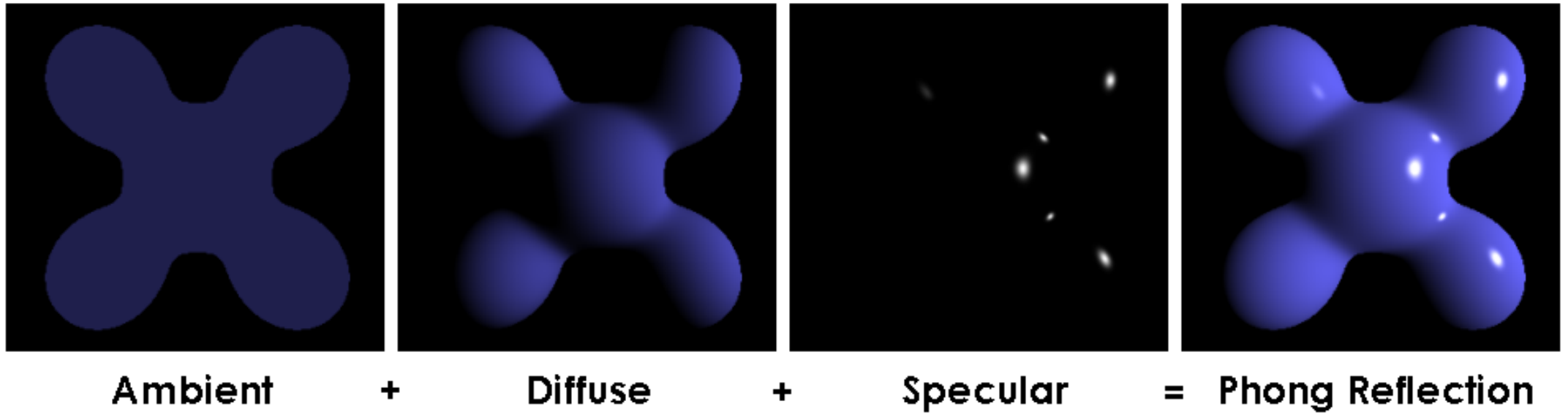




Local vs. Global Light Transport

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

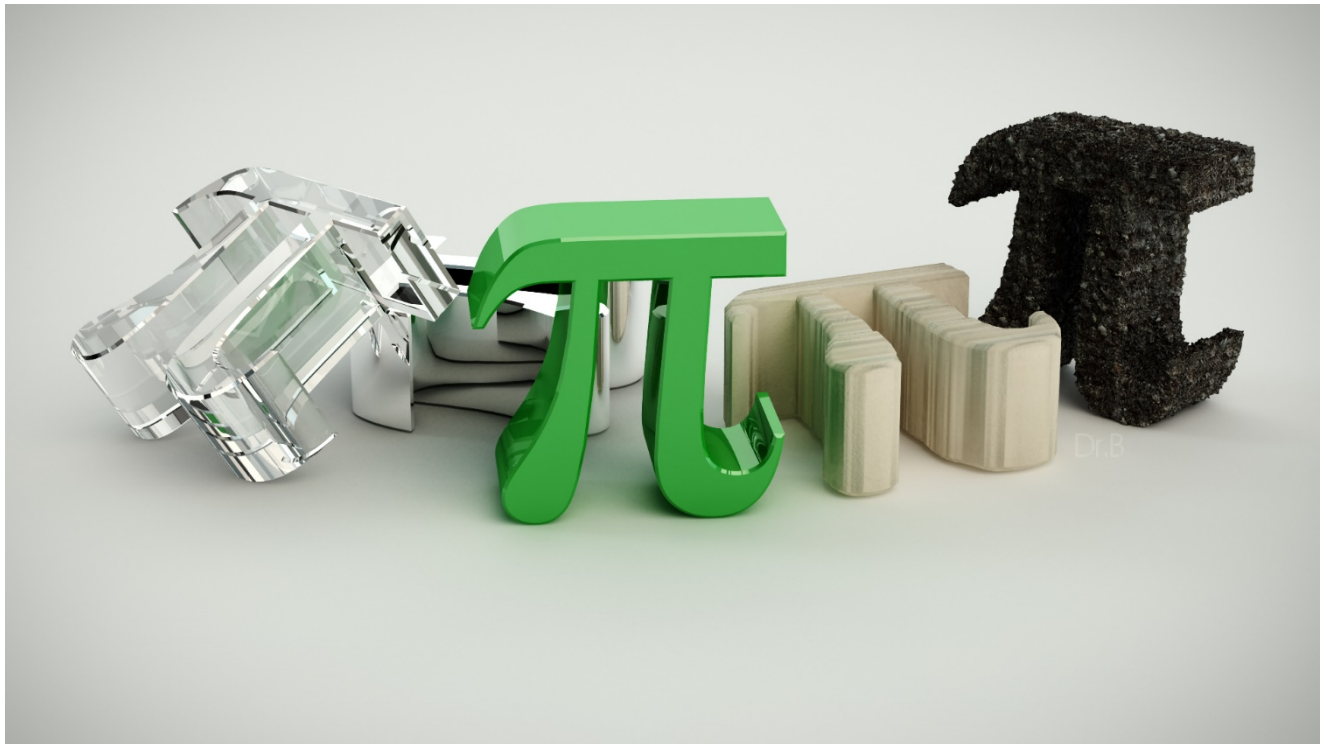
- Local Illumination Models
  - e.g. Phong
  - Light reflects on exactly one surface between light source and eye
  - Indirect light might be included with a constant term



Local vs. Global Light Transport

- Local Illumination Models
  - e.g. Phong
  - Light reflects on exactly one surface between light source and eye
  - Indirect light might be included with a constant term

- Global Illumination Models
  - e.g. Ray Tracing or Radiosity
  - Simulate and measure light transportation through the scene
  - Light interacts between objects and other objects or the environment



Global Illumination Effects

- Light scatters multiple times
  - Shadows
  - (Inter-) Reflections
  - Refractions
  - Caustics
  - …

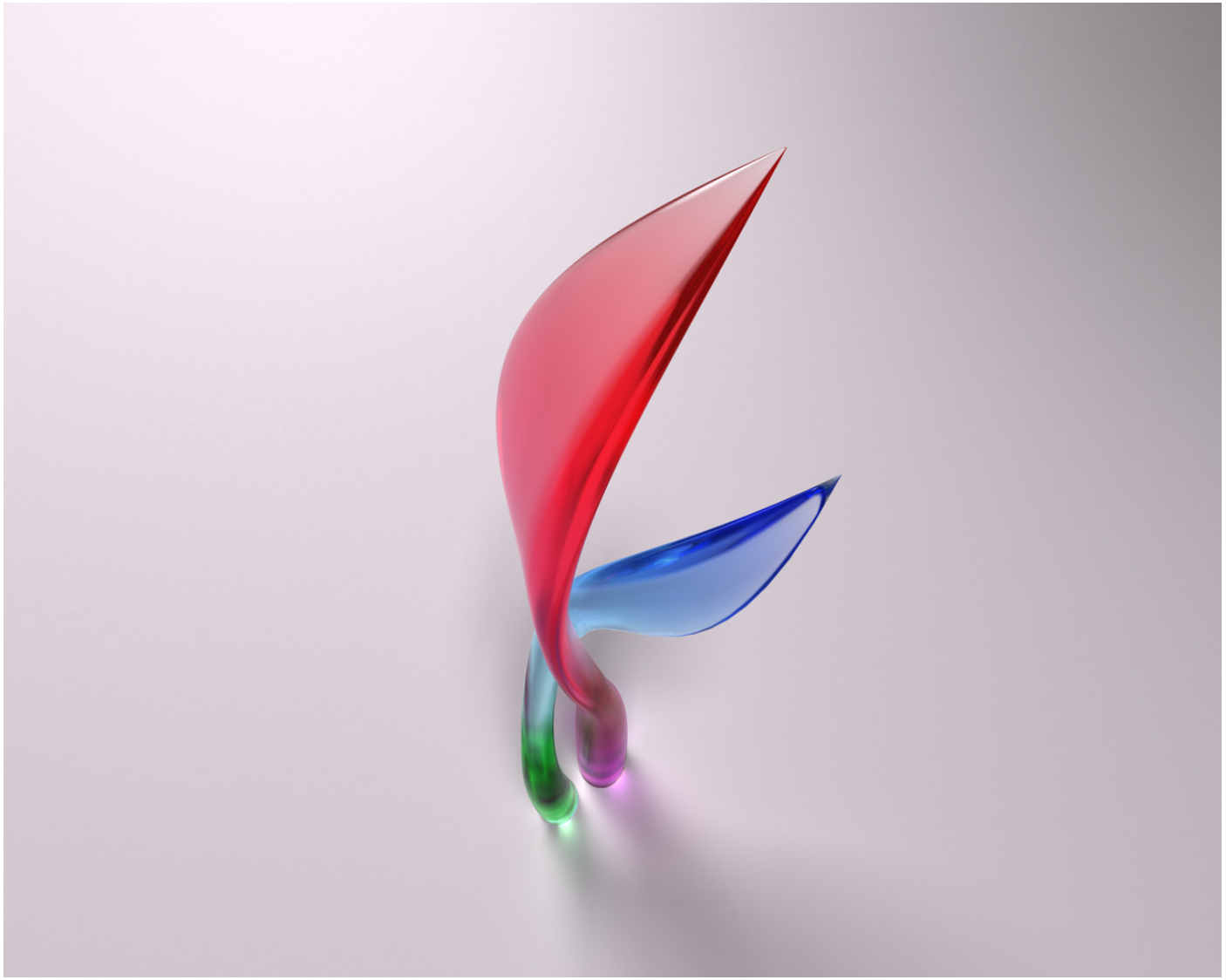


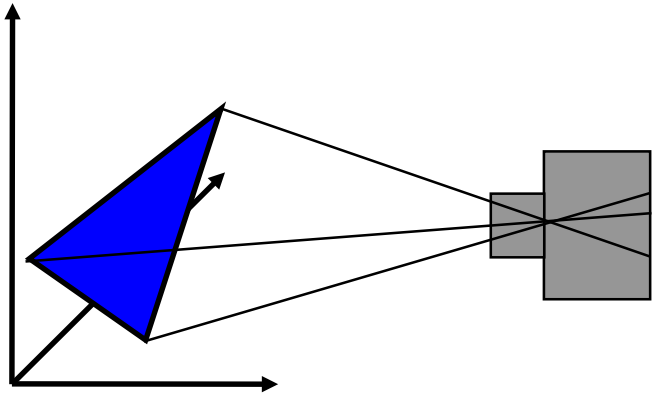
Image Formation

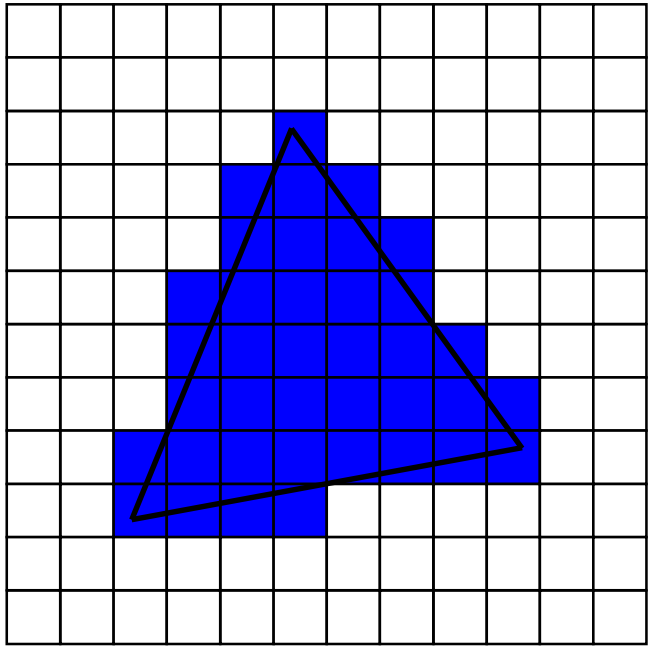
Why do images look like they do?

Image Formation: Rasterization

- Primitive operation of all interactive graphics, so far…
  - Scan convert a single triangle at a time

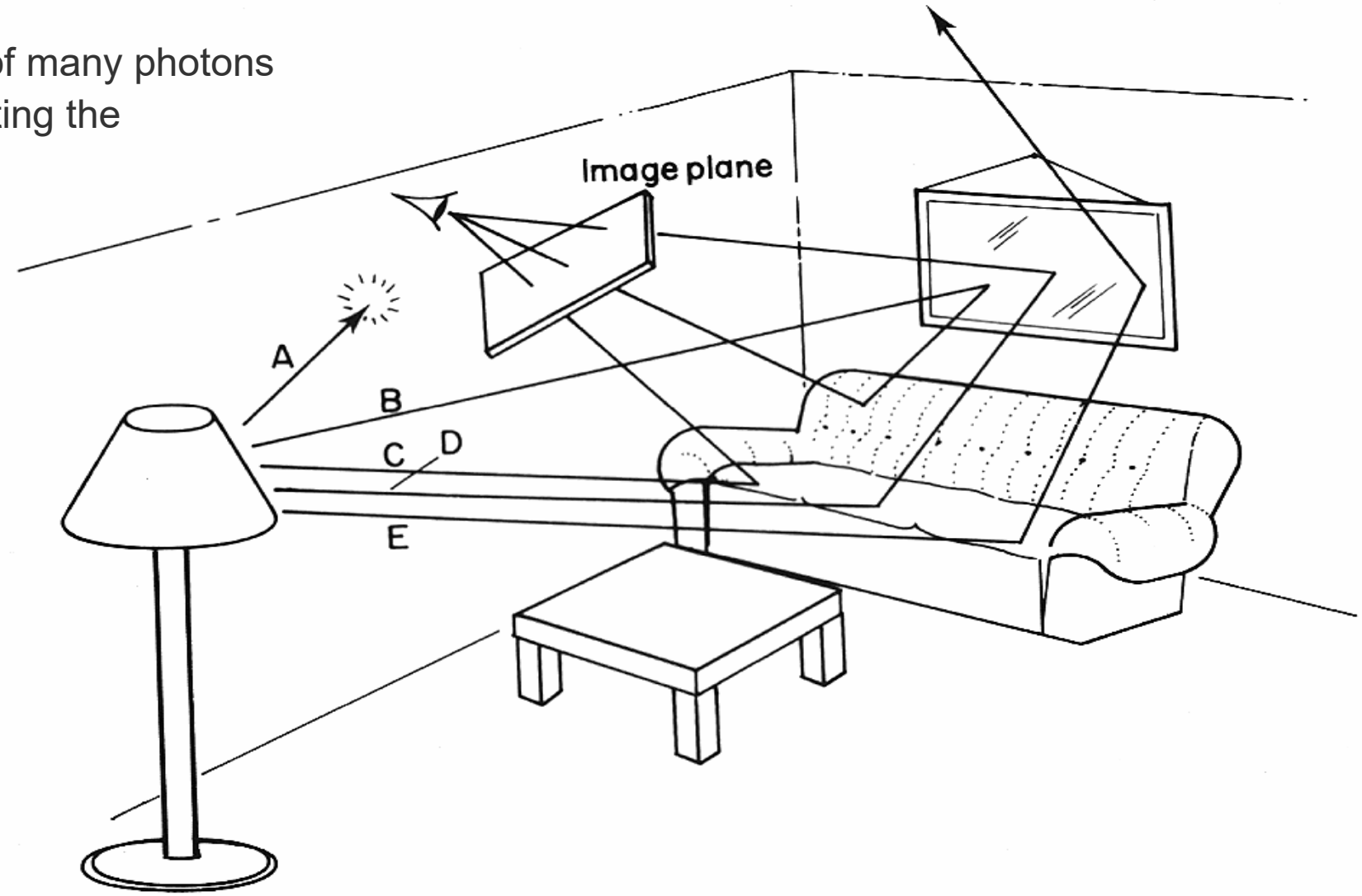
- Sequentially processes every triangle individually
  - Easy to account for direct reflection – local effects
  - But most effects need access to the world:
    - Shadows, Reflection, Global Illumination





Tracing the Paths of Light

- Nature:
  - Follow the path of many photons
  - Record those hitting the film in a camera



Light Transport

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Light Distribution in a Scene
  - Dynamic equilibrium
  - Newly created, scattered, and absorbed photons

- Forward Light Transport:
  - Start at the light sources
  - Shoot photons into scene
  - Reflect at surfaces (according to some reflection model)
  - Wait until they are absorbed or hit the camera (very seldom)
  - Nature: massive parallel processing at the speed of light

- Backward Light Transport:
  - Start at the camera
  - Trace only paths that transport light towards the camera
  - **Ray tracing**

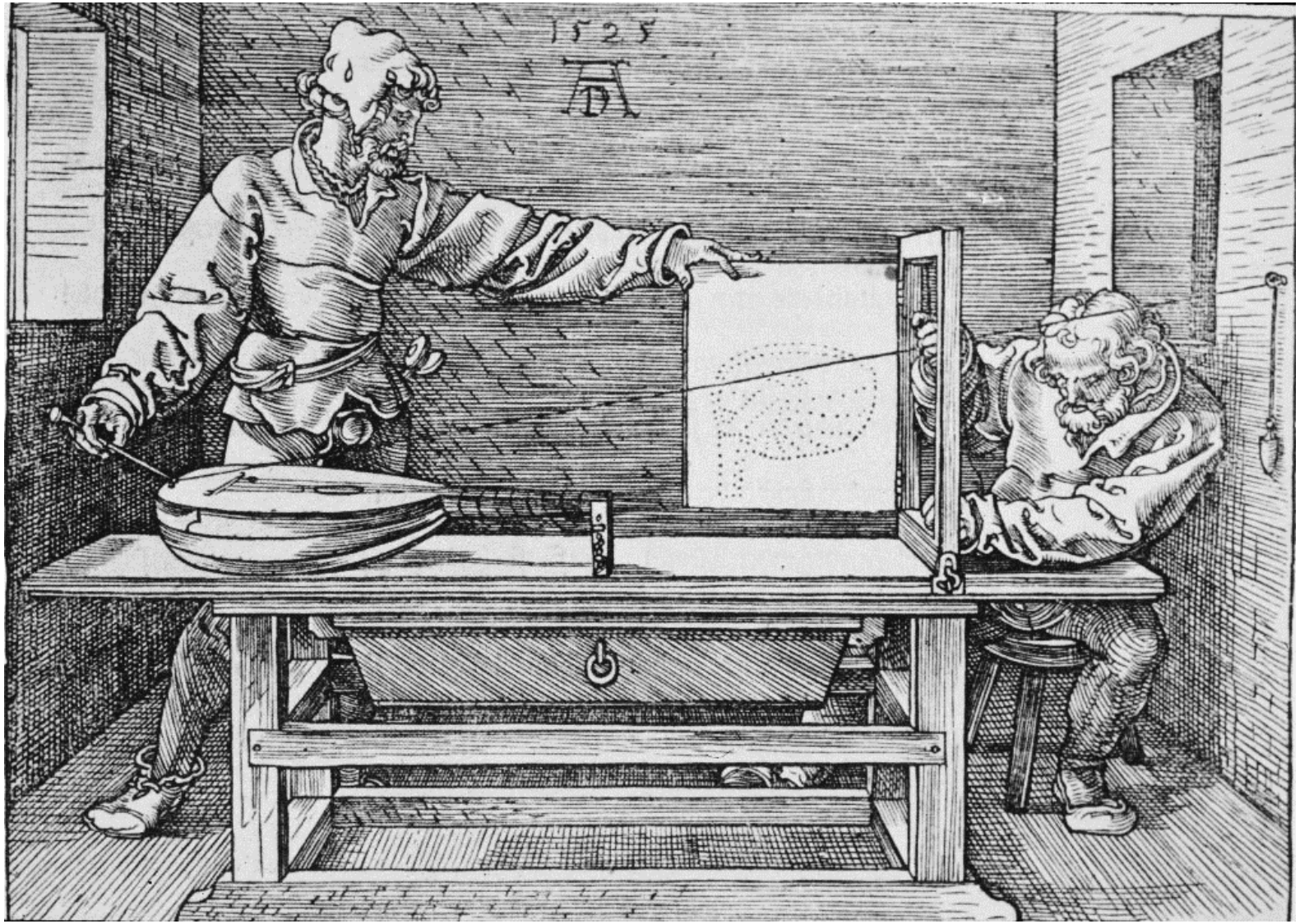
Ingredients

- Surfaces
  - 3D geometry of objects in a scene
- Surface reflectance characteristics
  - Color, absorption, reflection, refraction, subsurface scattering
  - Local property, may vary over surface
  - Mirror, glass, glossy, diffuse, …
- Illumination
  - Position, characteristics of light emitters
  - Repeatedly reflected light → indirect illumination
- Assumption: air/empty space is totally transparent
  - Excludes any scattering effects in participating media volumes
  - Would require solving a much more complex problem
  - In contrast to volume rendering, participating media

Ray Tracing

- The Ray Tracing Algorithm
  - One of the two fundamental rendering algorithms (vs. Rasterization)
- Simple and intuitive
  - Easy to understand and implement
- Powerful and efficient
  - Many optical global effects
    - shadows, reflection, refraction, and others
  - Efficient real-time implementation in SW and HW
- Scalability
  - Can work in parallel and distributed environments
  - Inherent logarithmic scalability with scene size: O(log n) vs. O(n)
  - Output sensitive and demand driven
- Not new
  - Light rays: Empedocles (492-432 BC), Renaissance (Dürer, 1525)
  - Uses in lens design, geometric optics, …

Ray Tracing

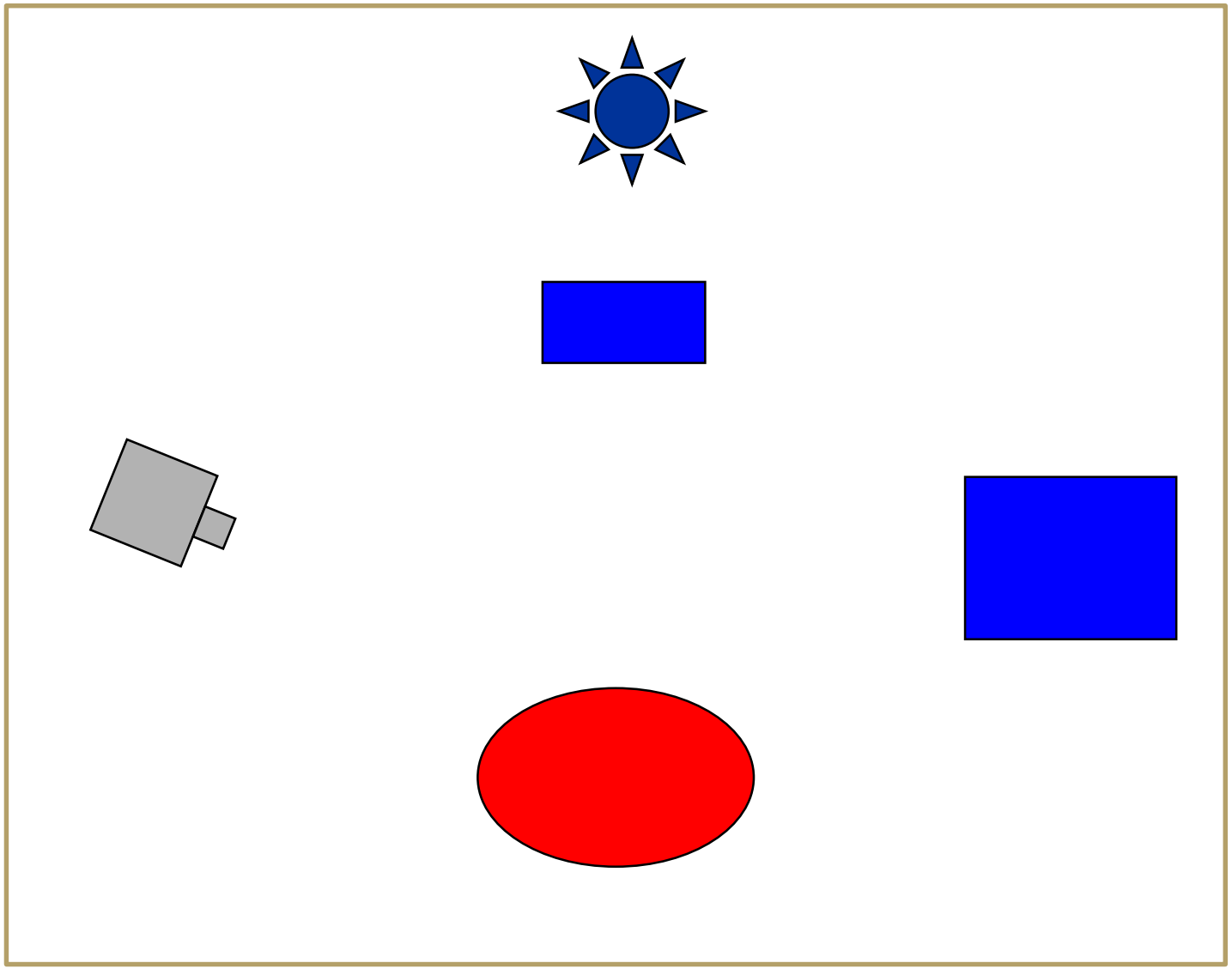


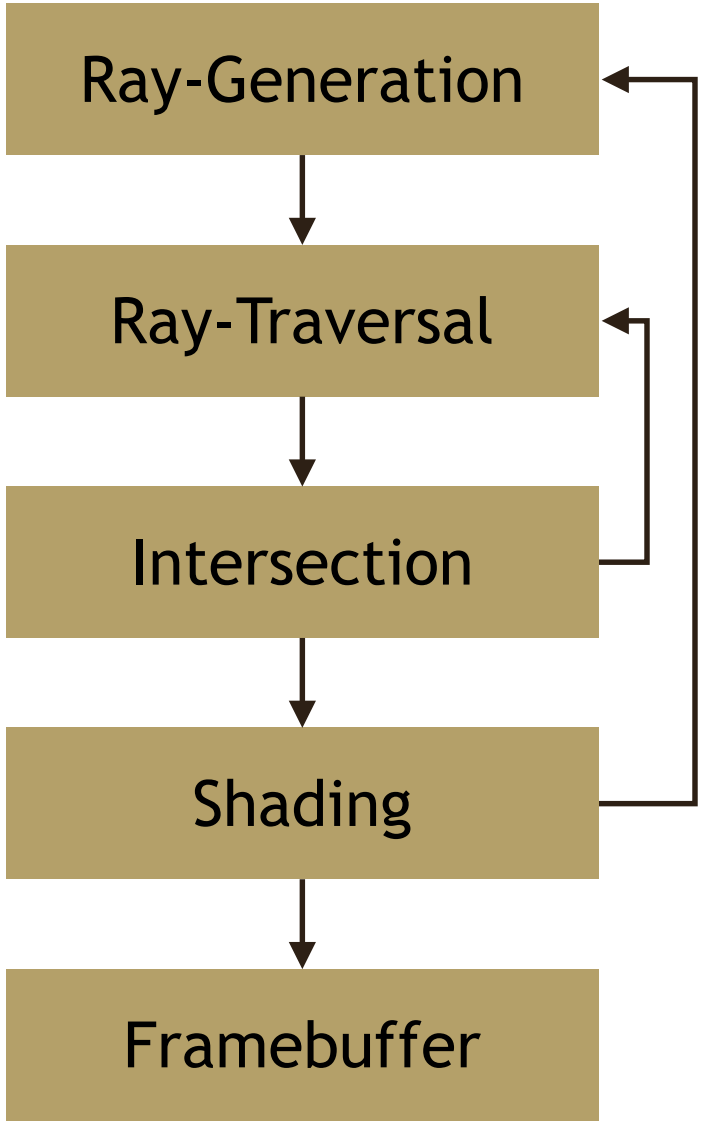
Ray tracing in a woodcut
by Albrecht Dürer, 1525

Global Illumination Effects

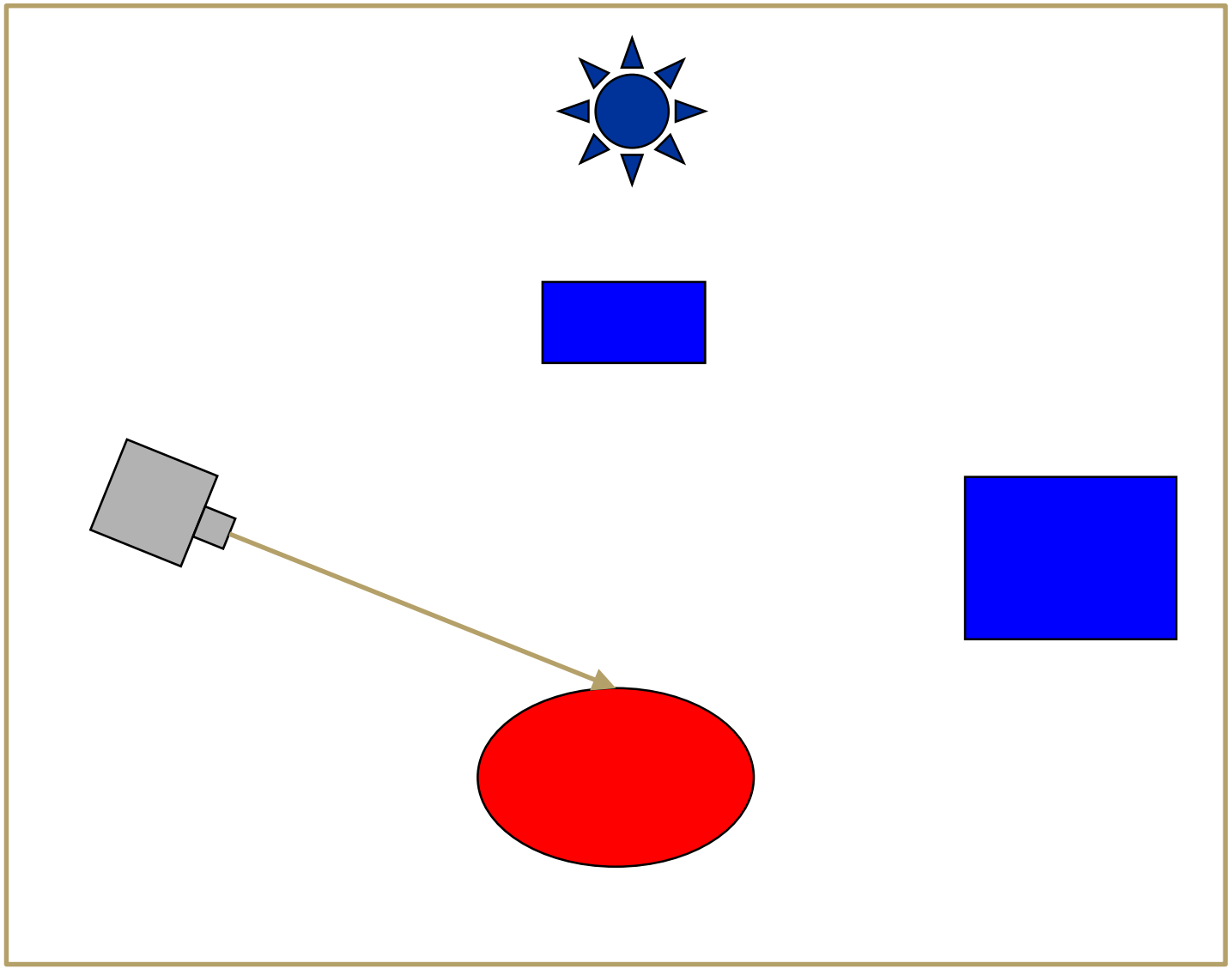
- Highly realistic images
  - Ray tracing enables *correct* simulation of light transport

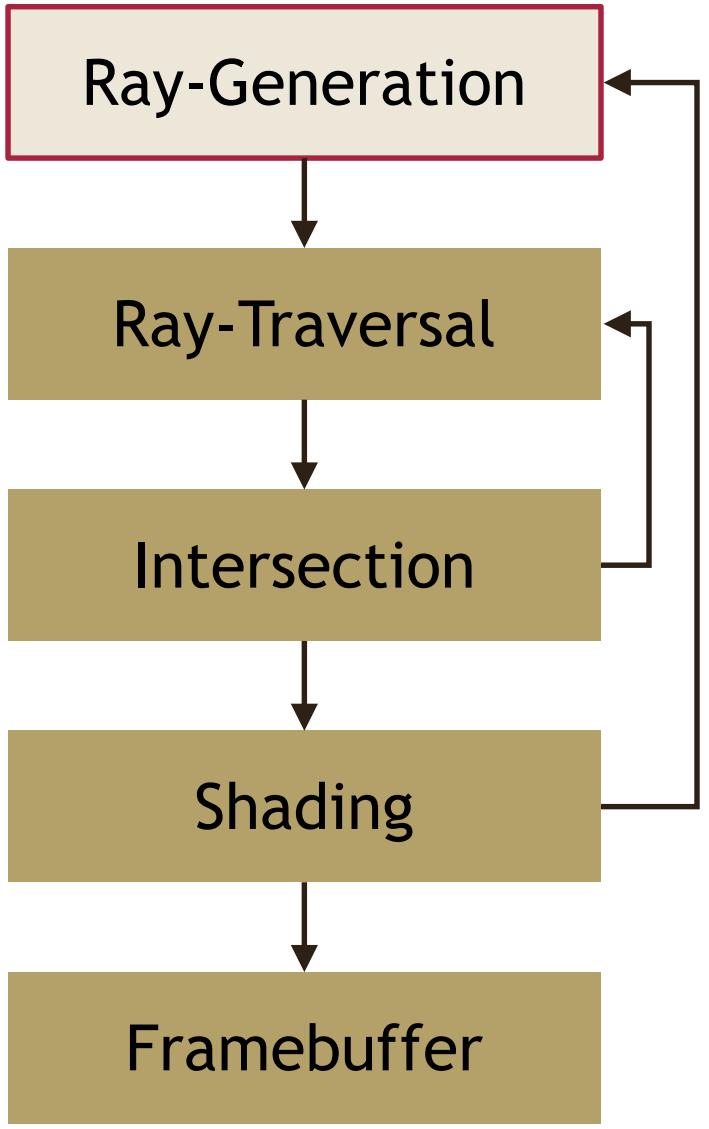
Ray Tracing Pipeline



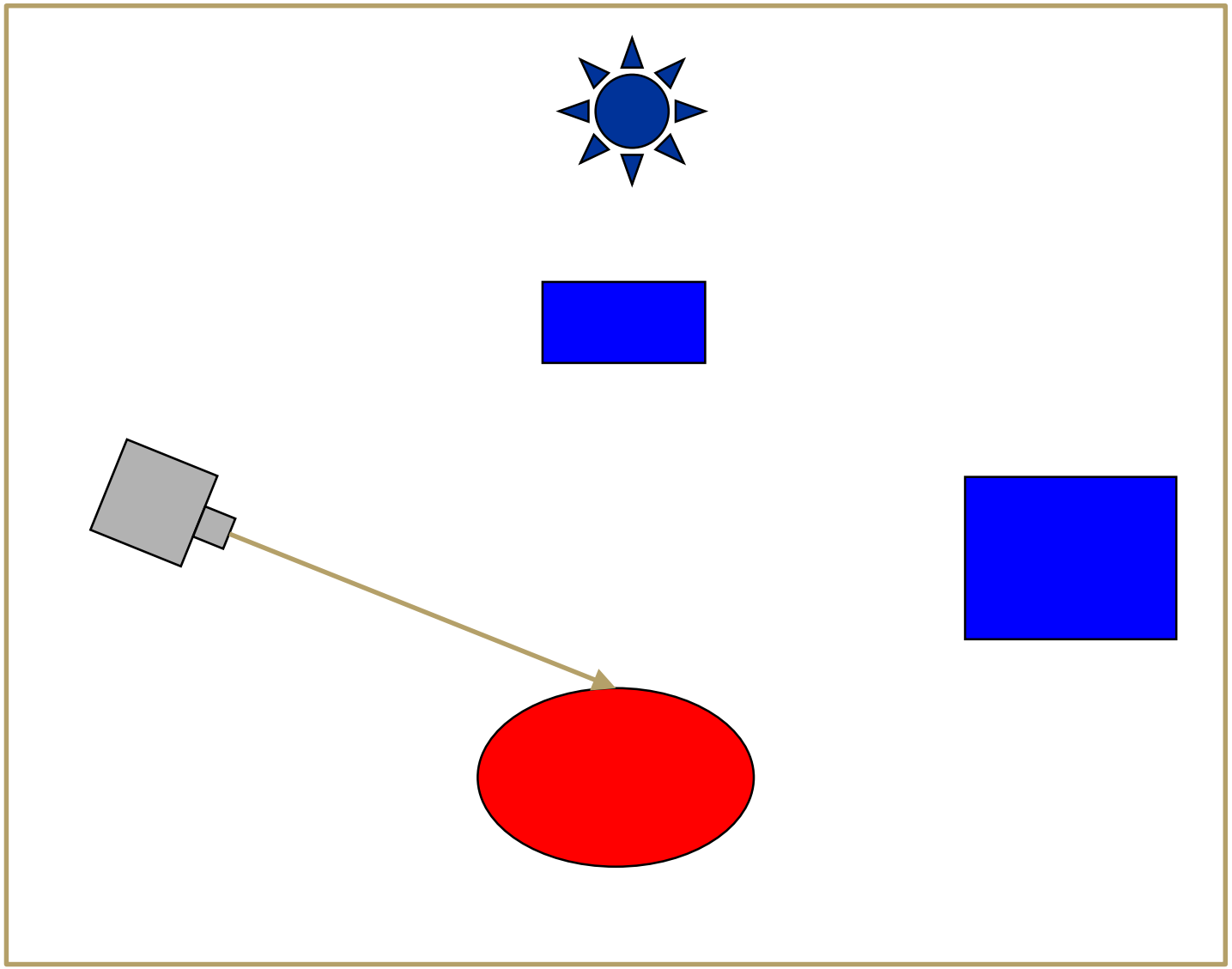


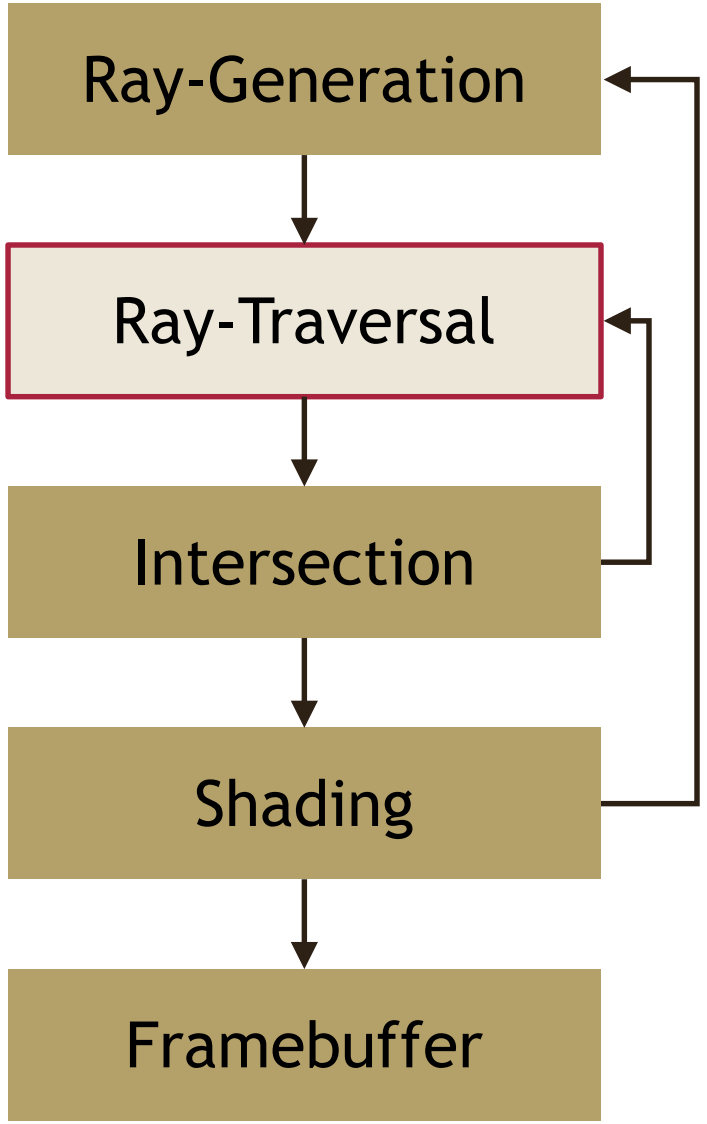
Ray Tracing Pipeline



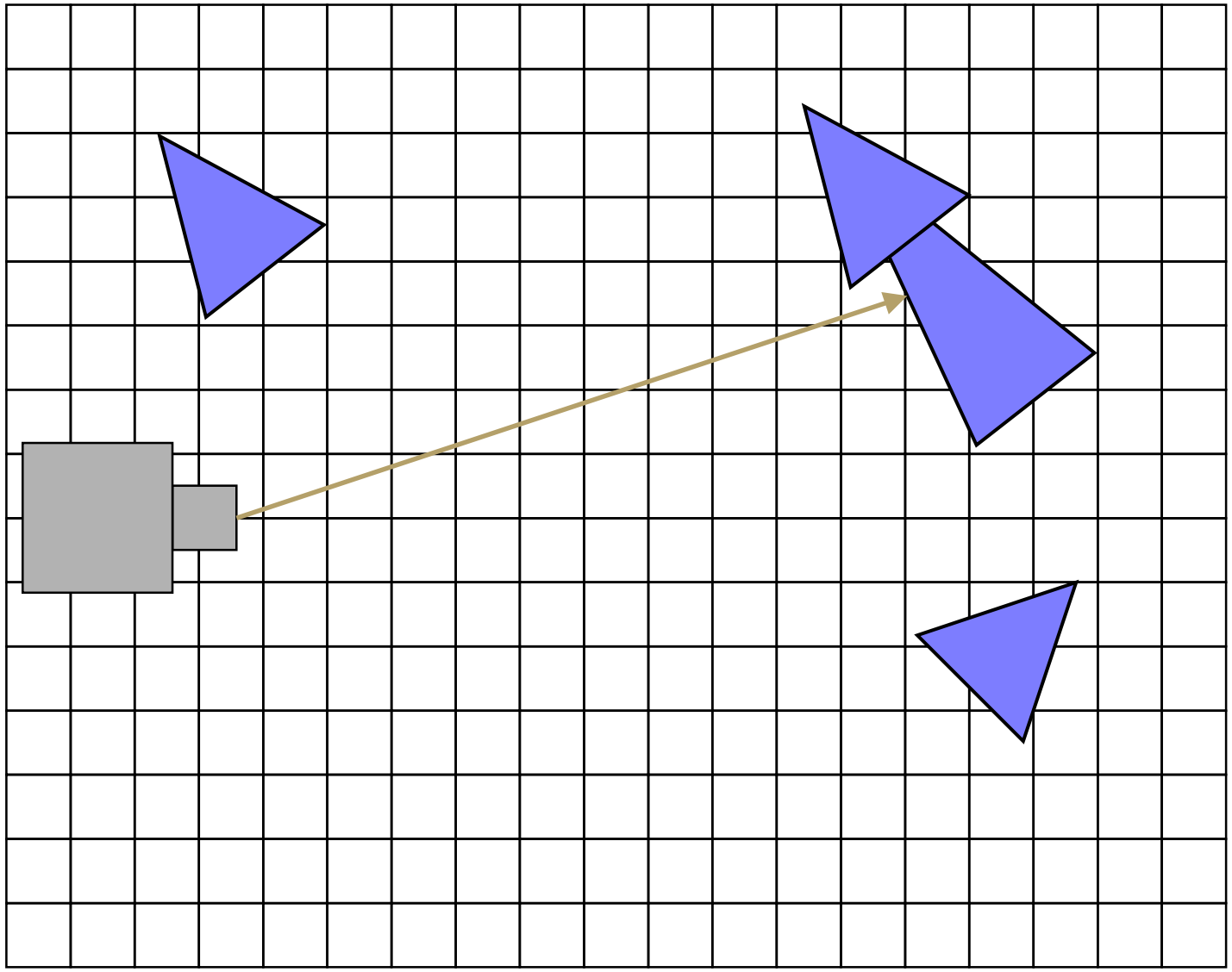


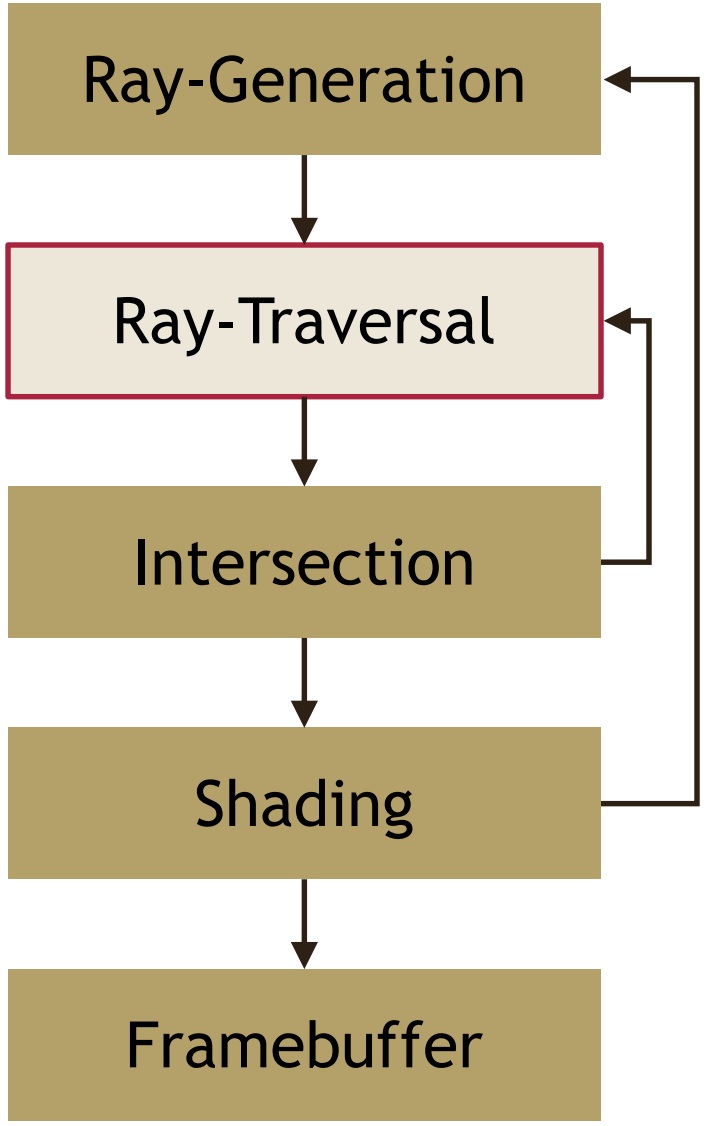
Ray Tracing Pipeline



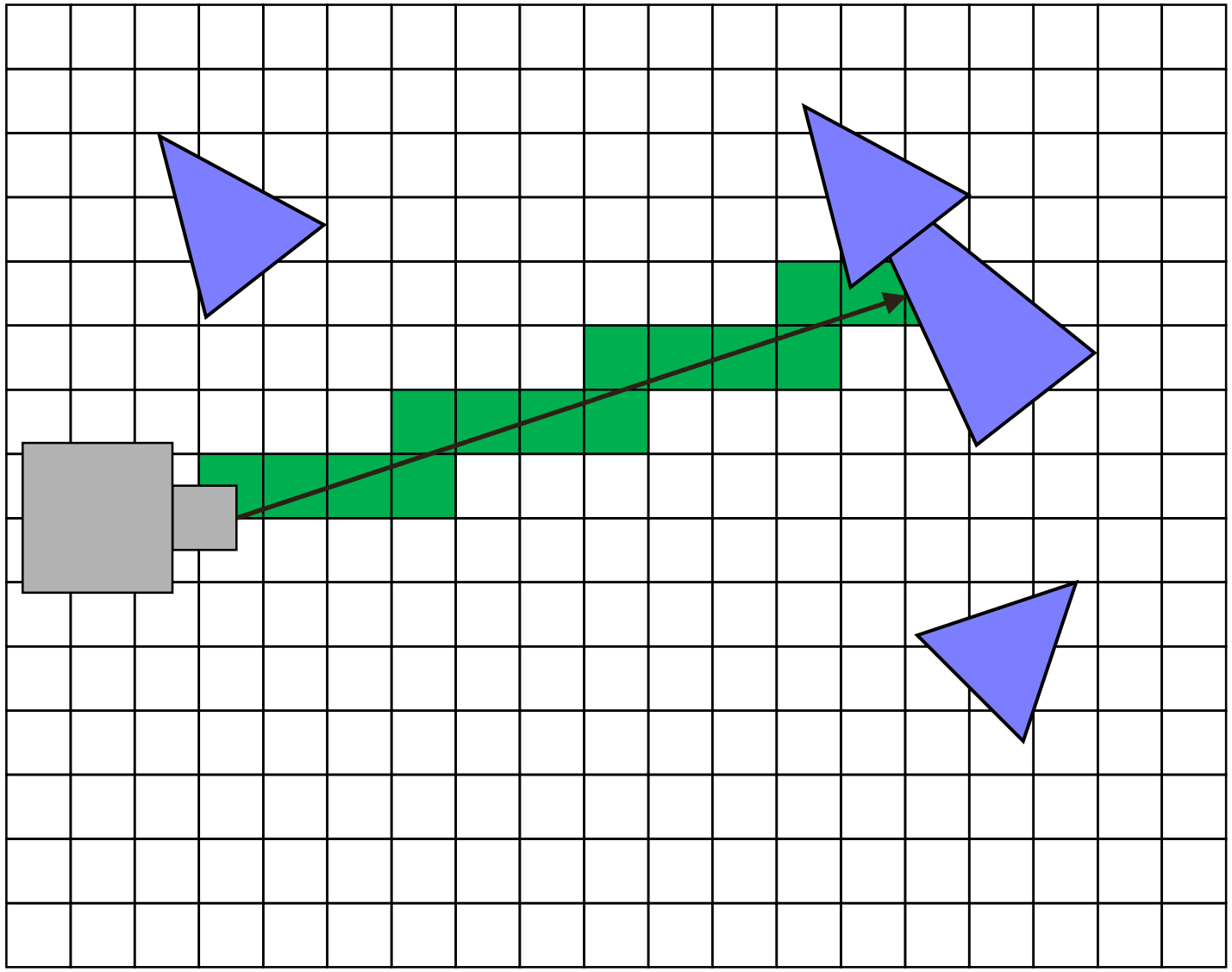


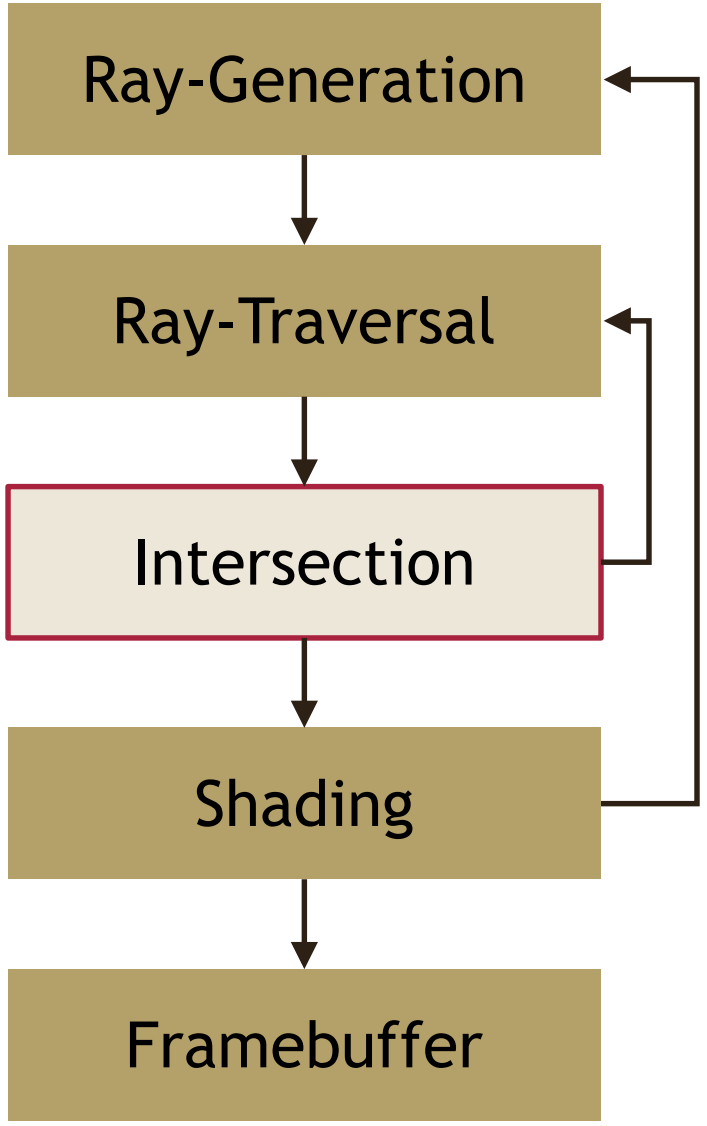
Ray Tracing Pipeline



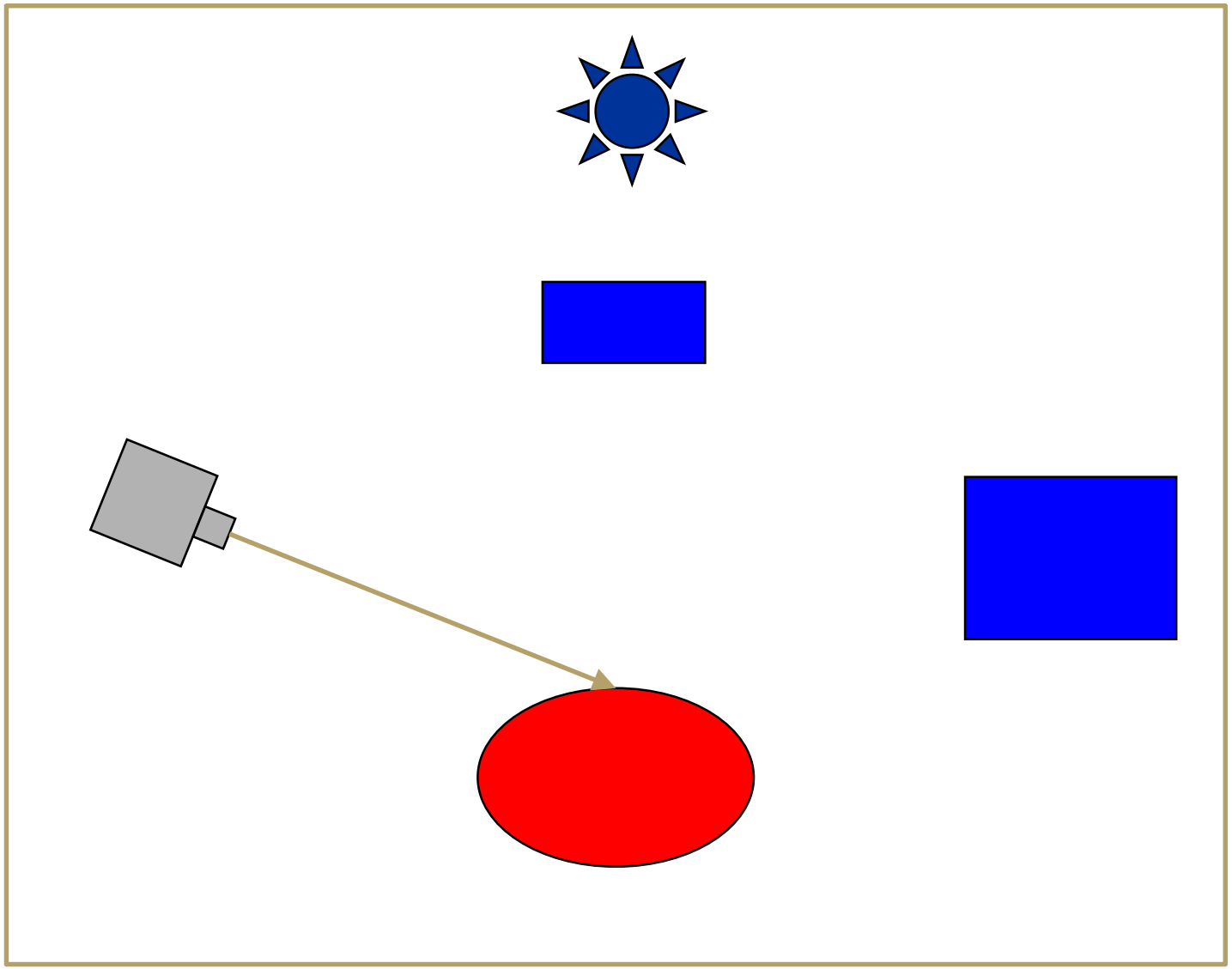


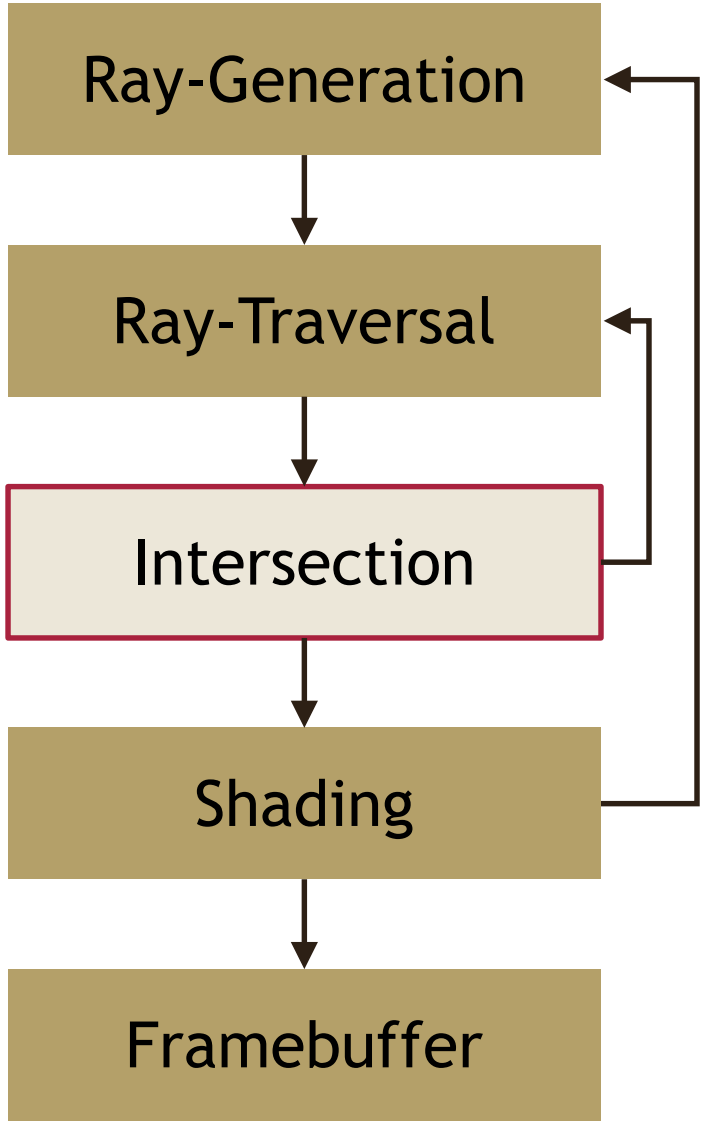
Ray Tracing Pipeline



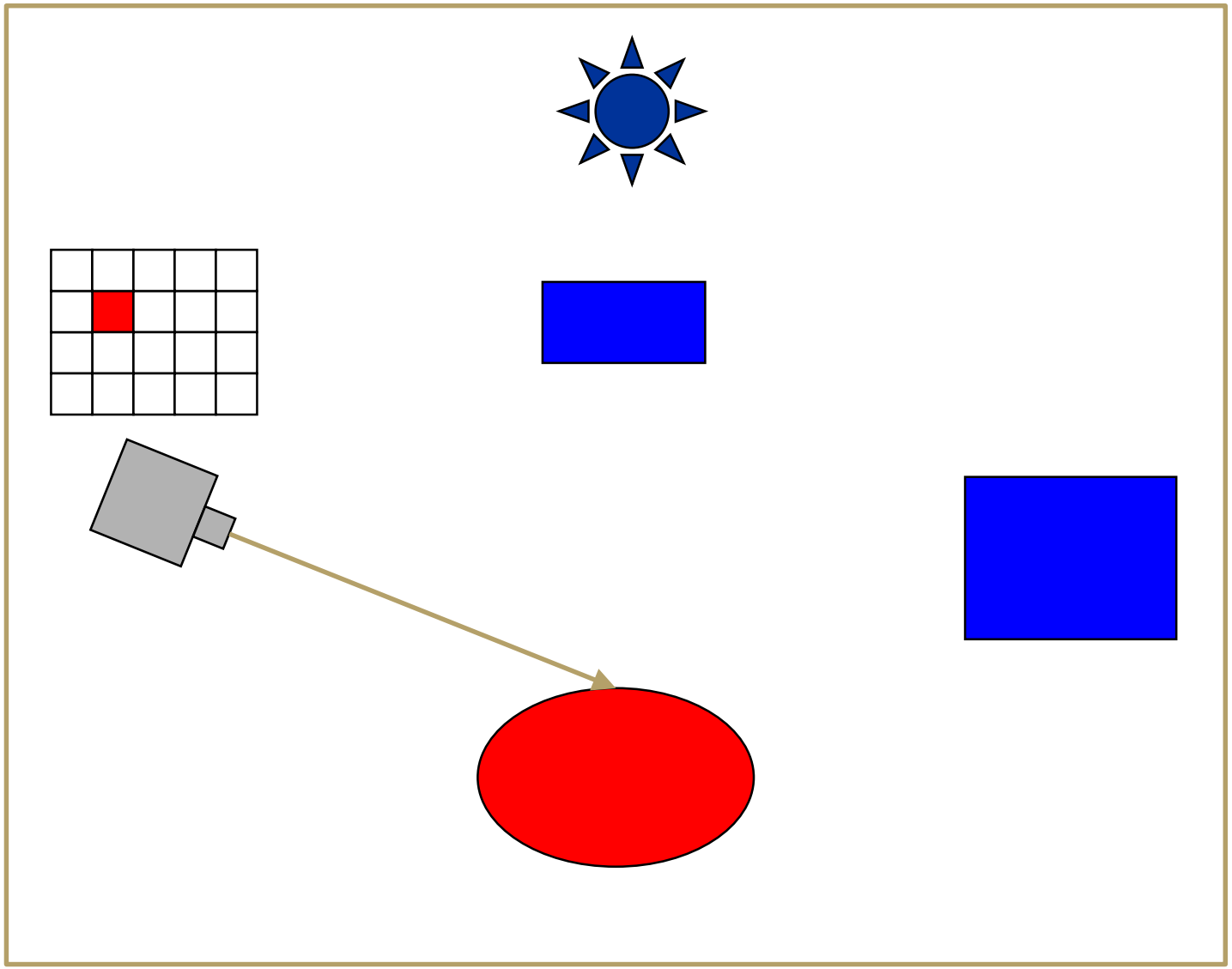


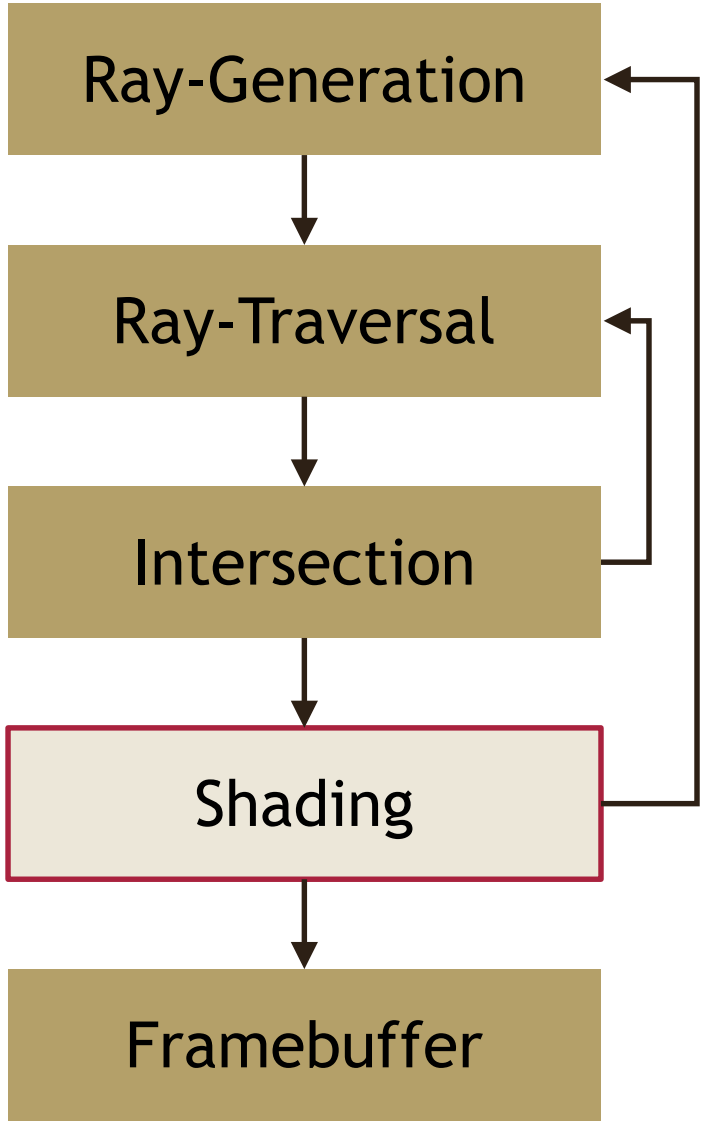
Ray Tracing Pipeline



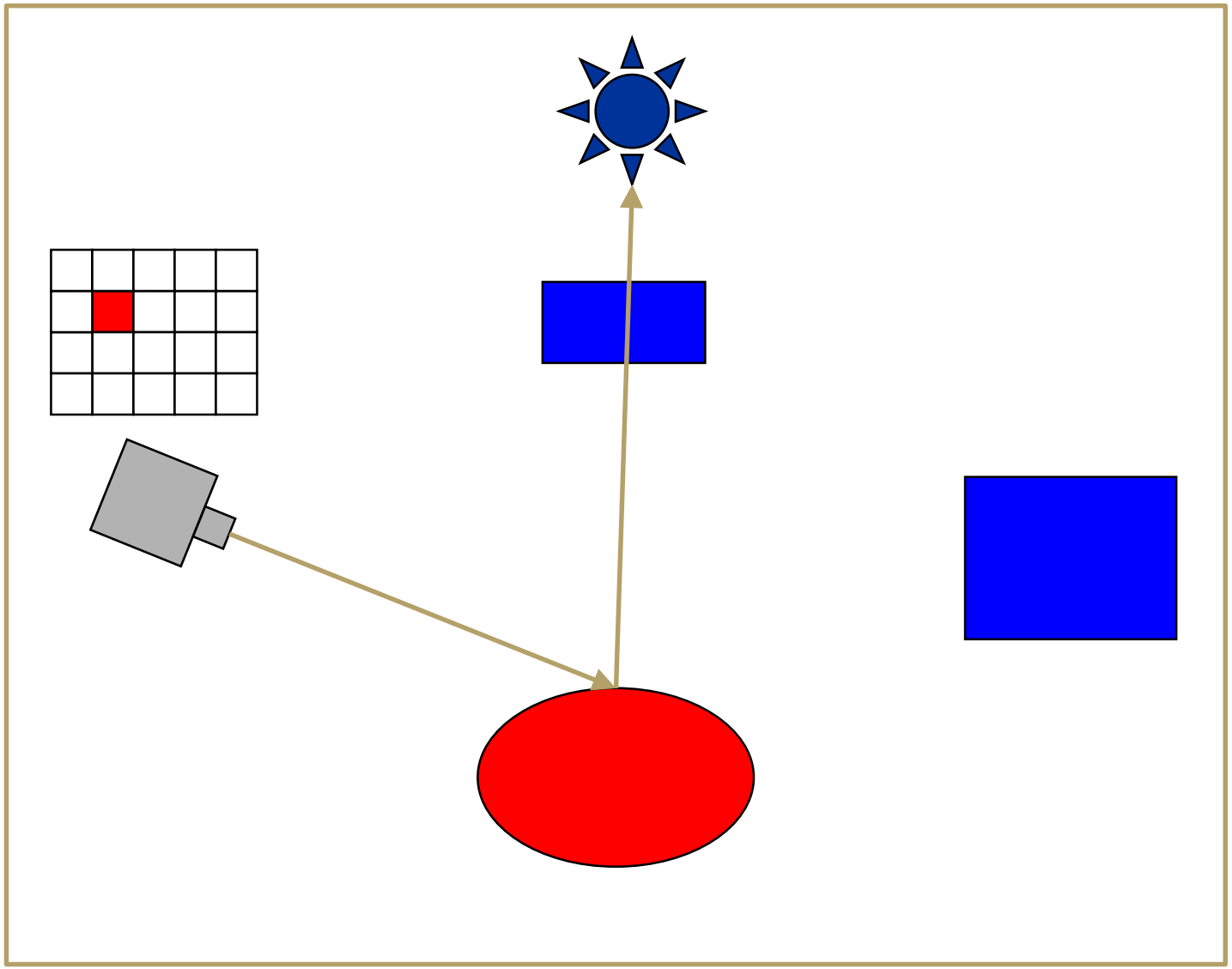


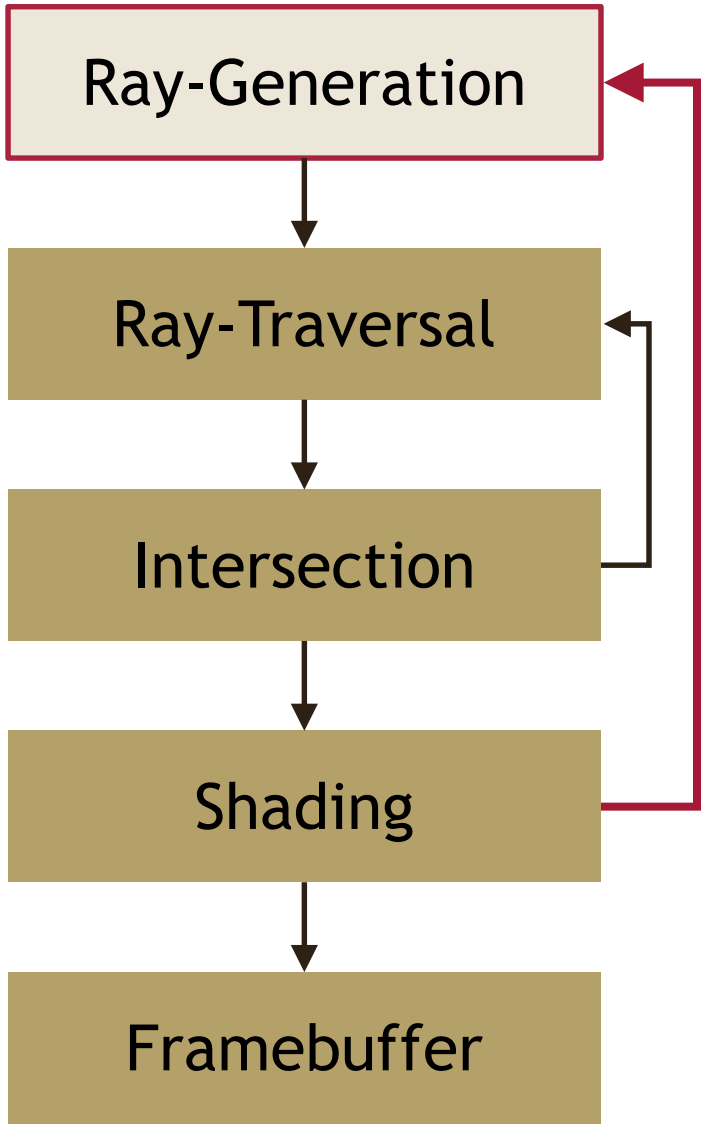
Ray Tracing Pipeline



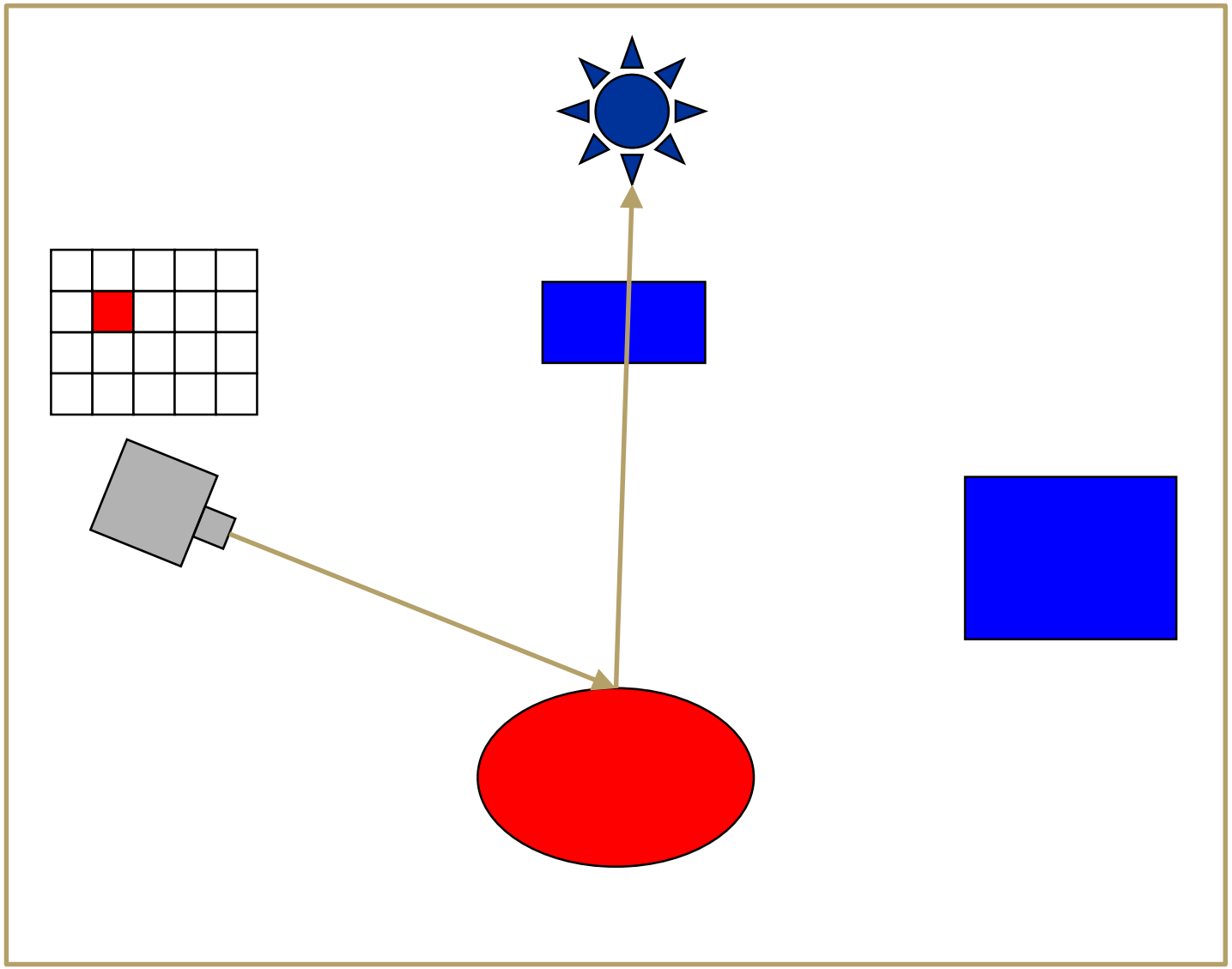


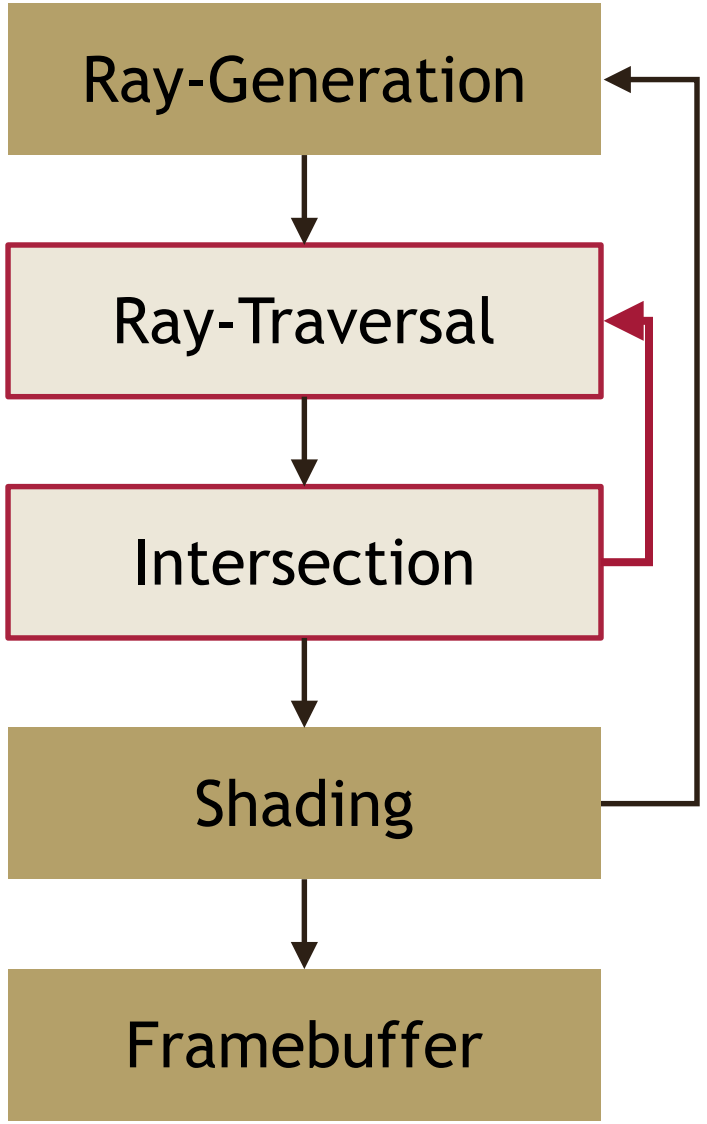
Ray Tracing Pipeline



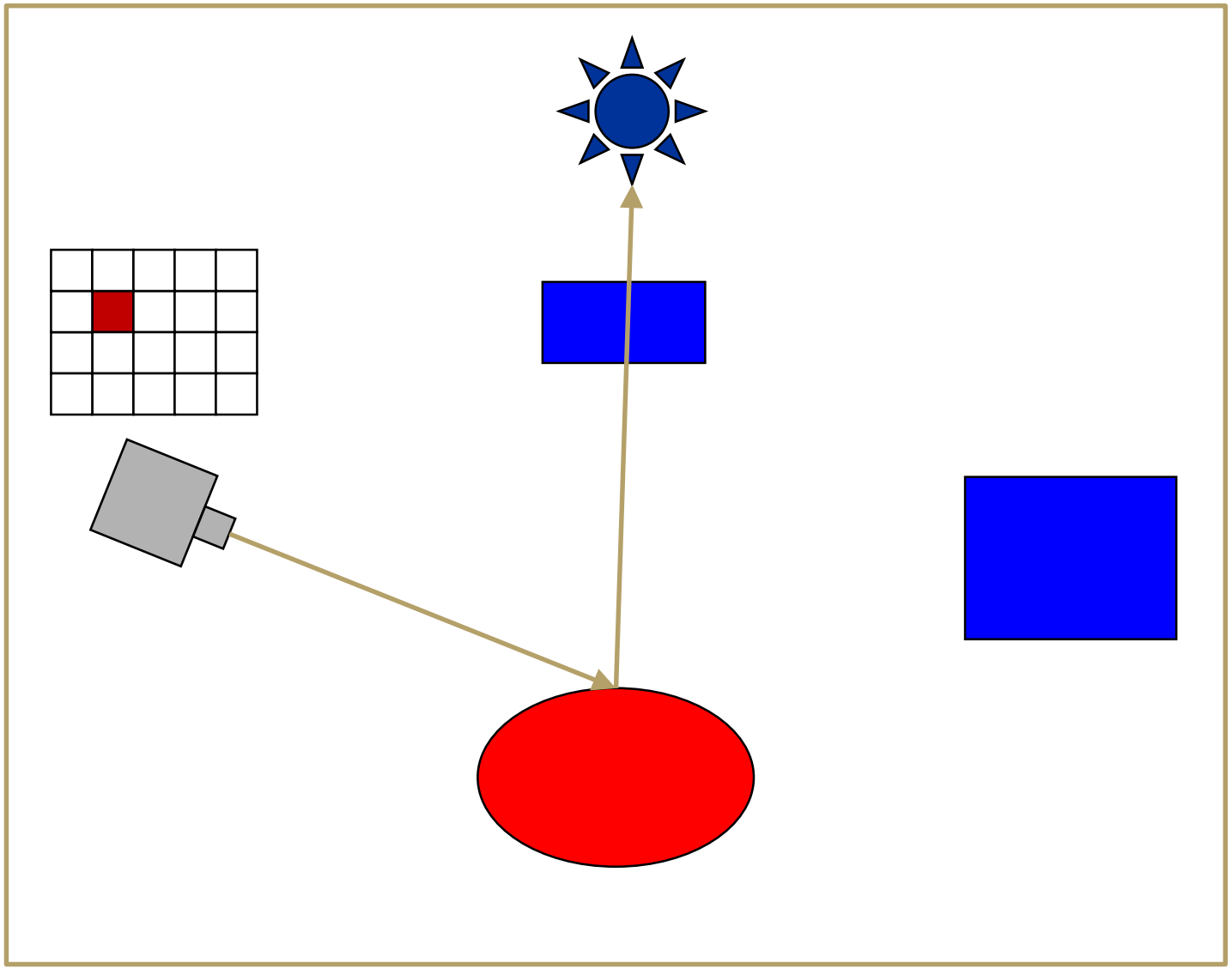


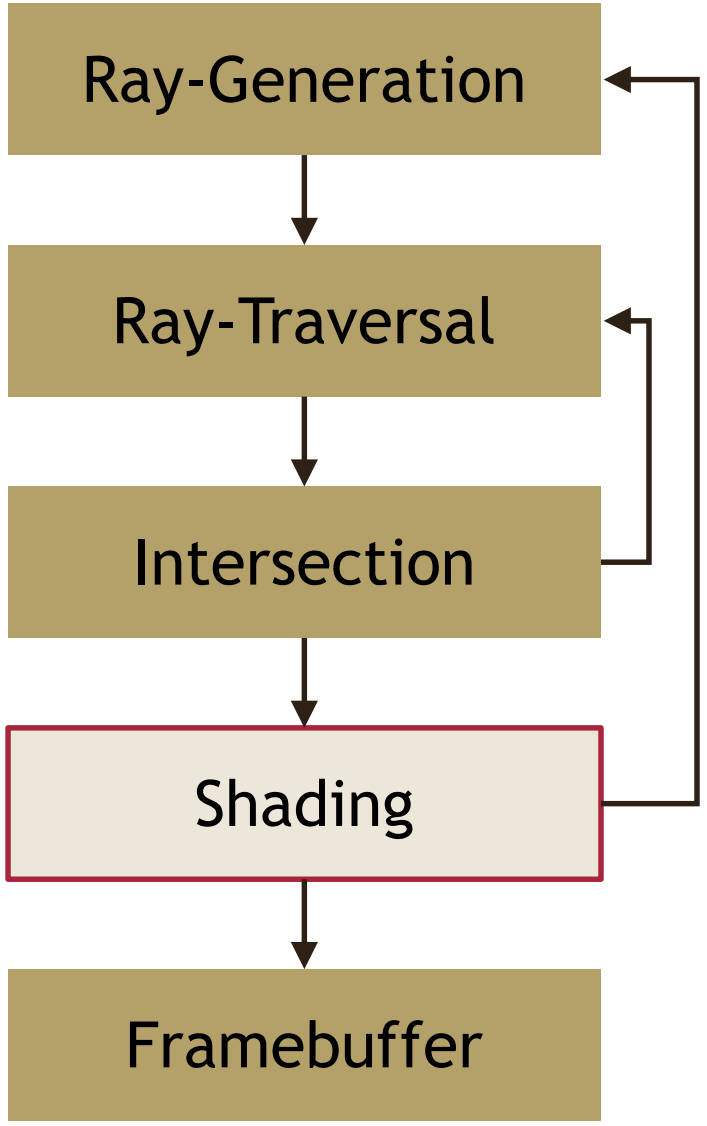
Ray Tracing Pipeline



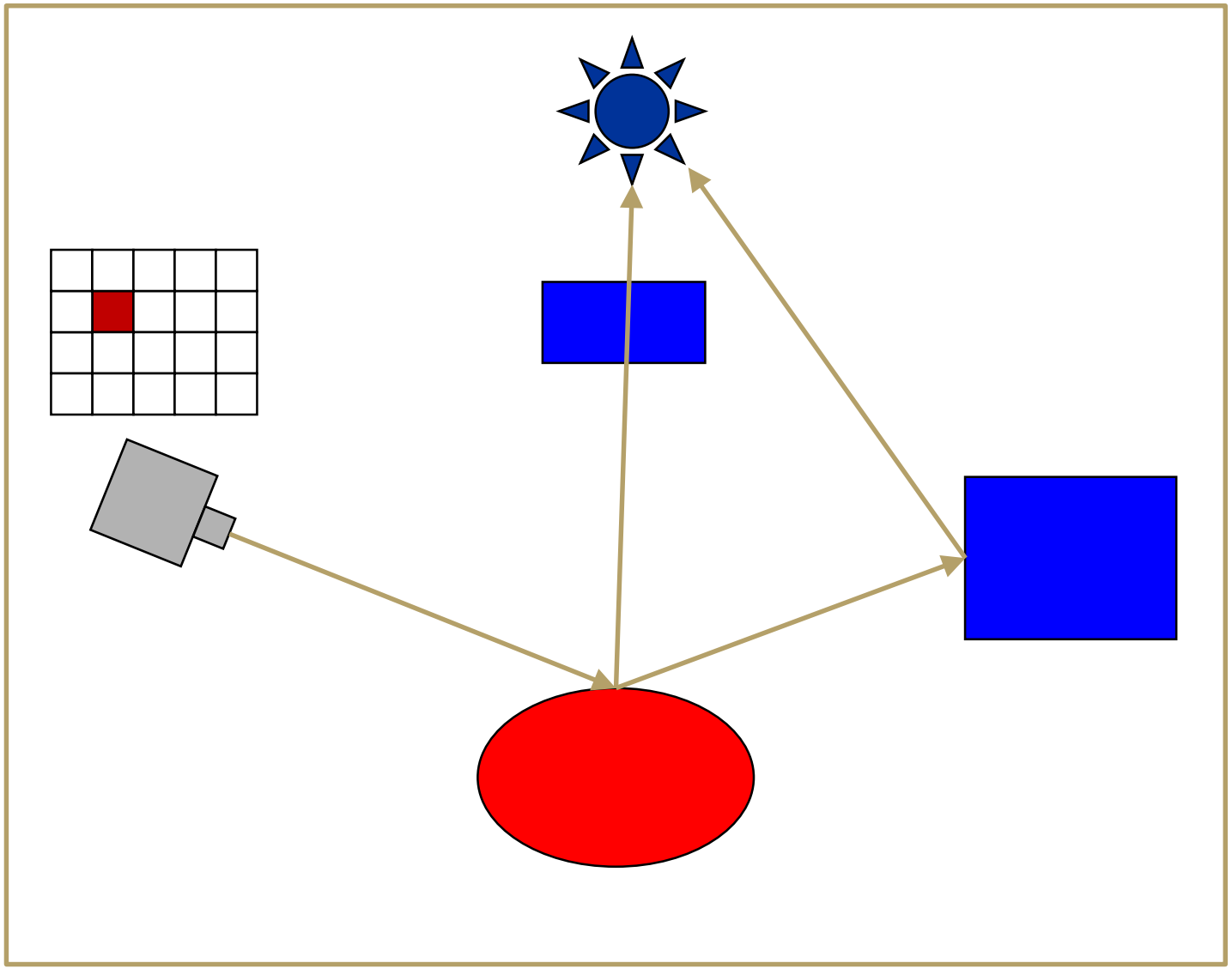


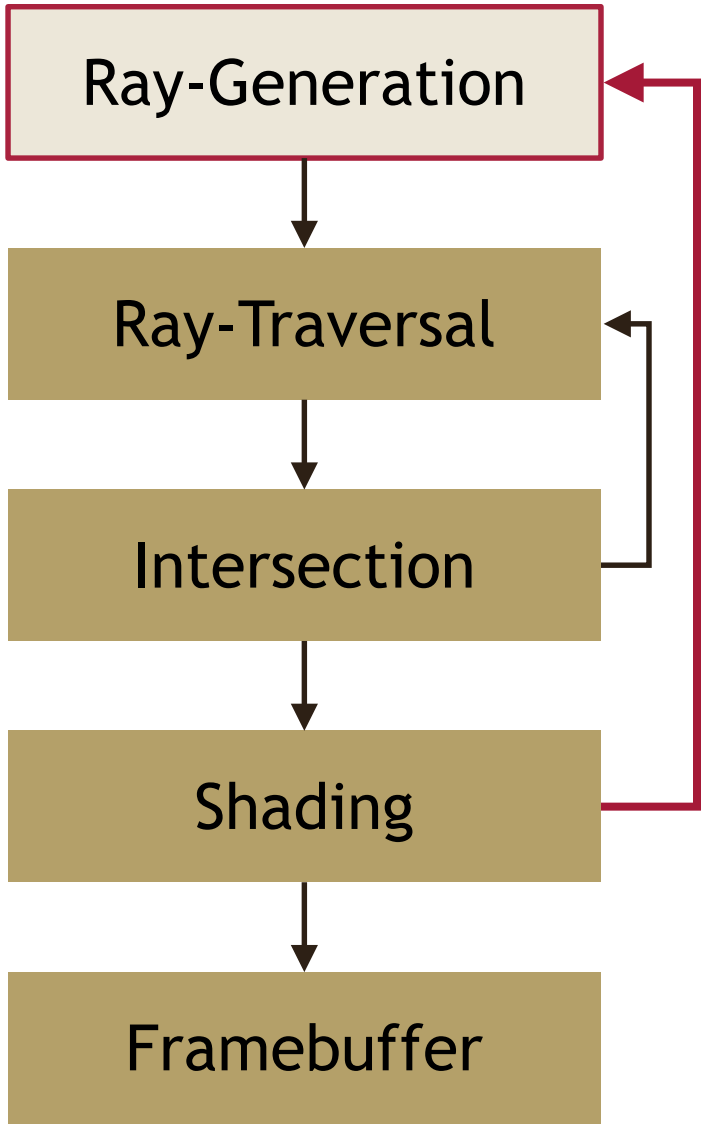
Ray Tracing Pipeline



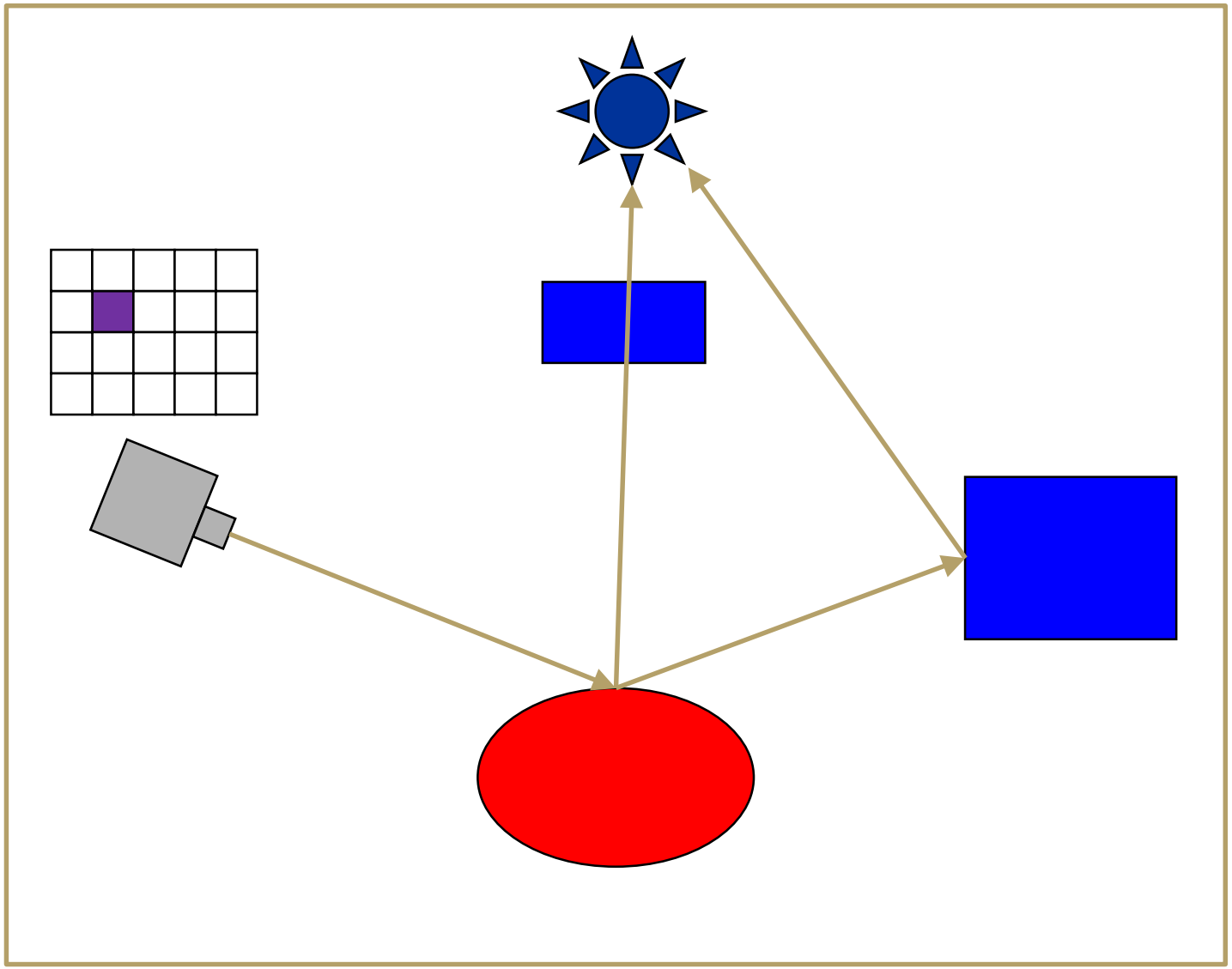


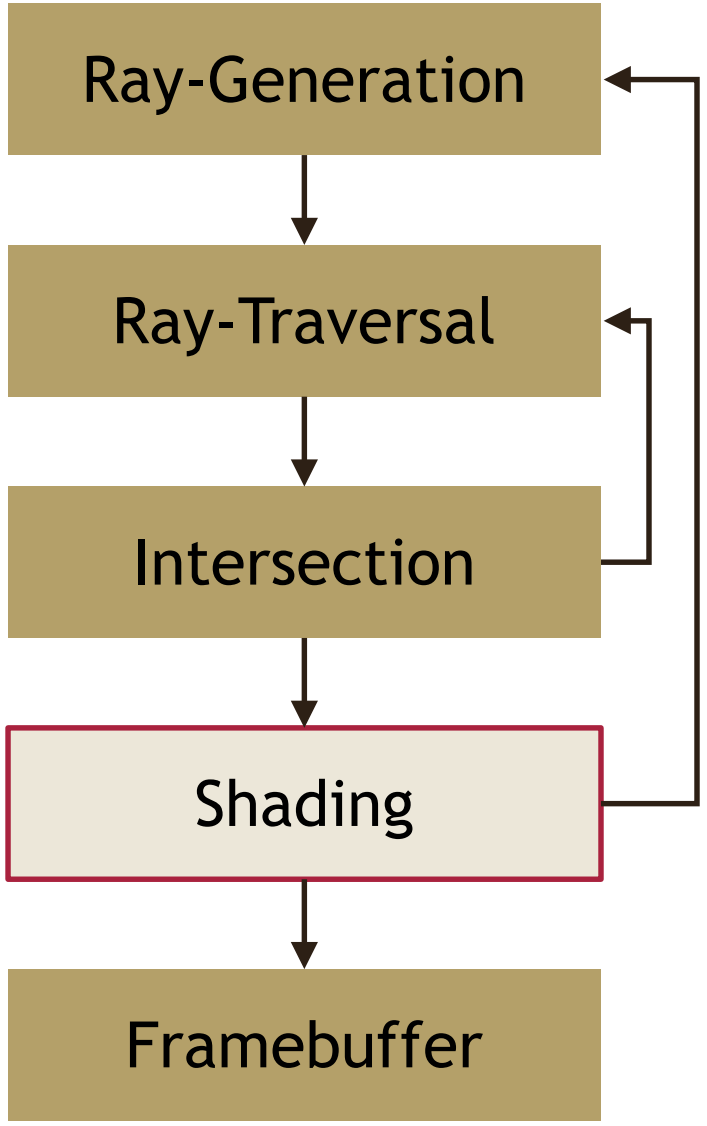
Ray Tracing Pipeline



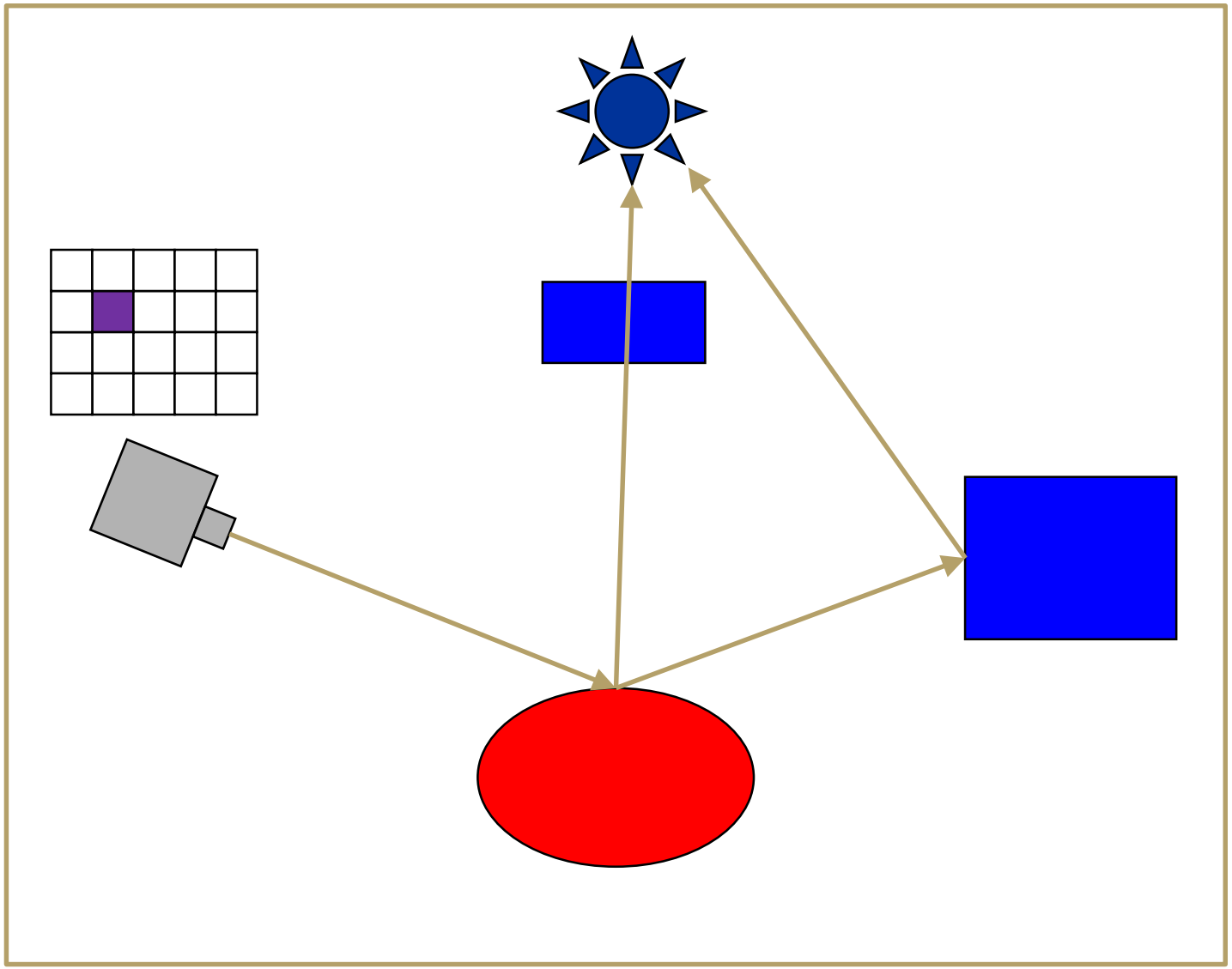


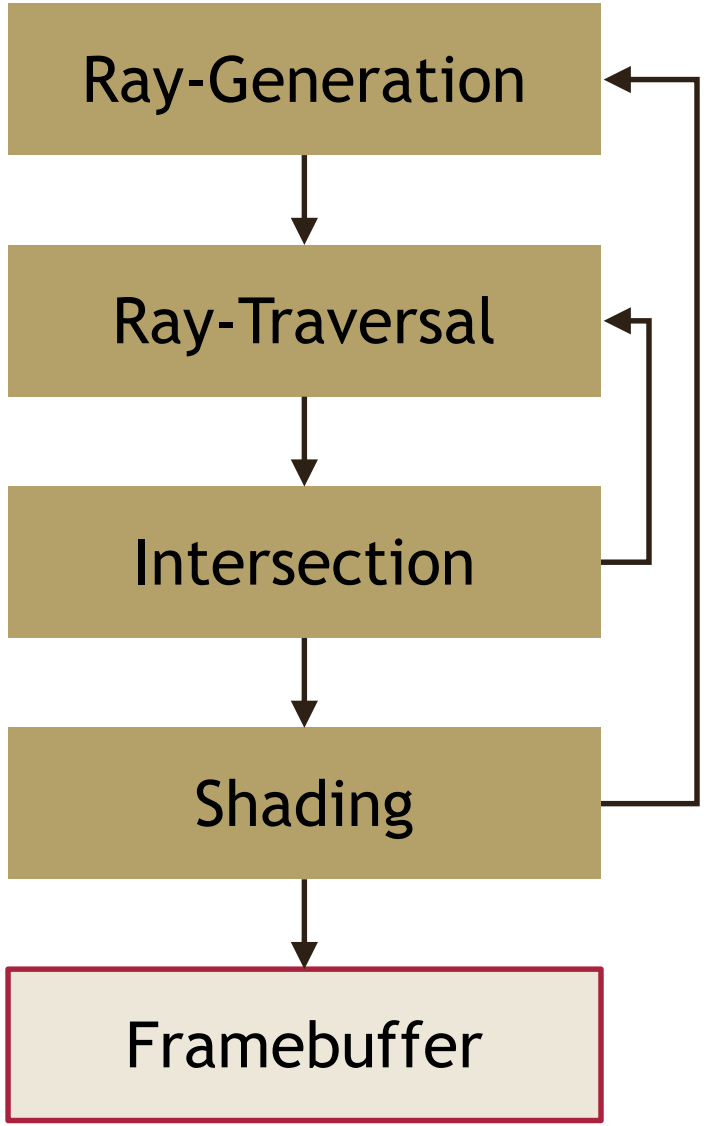
Ray Tracing Pipeline





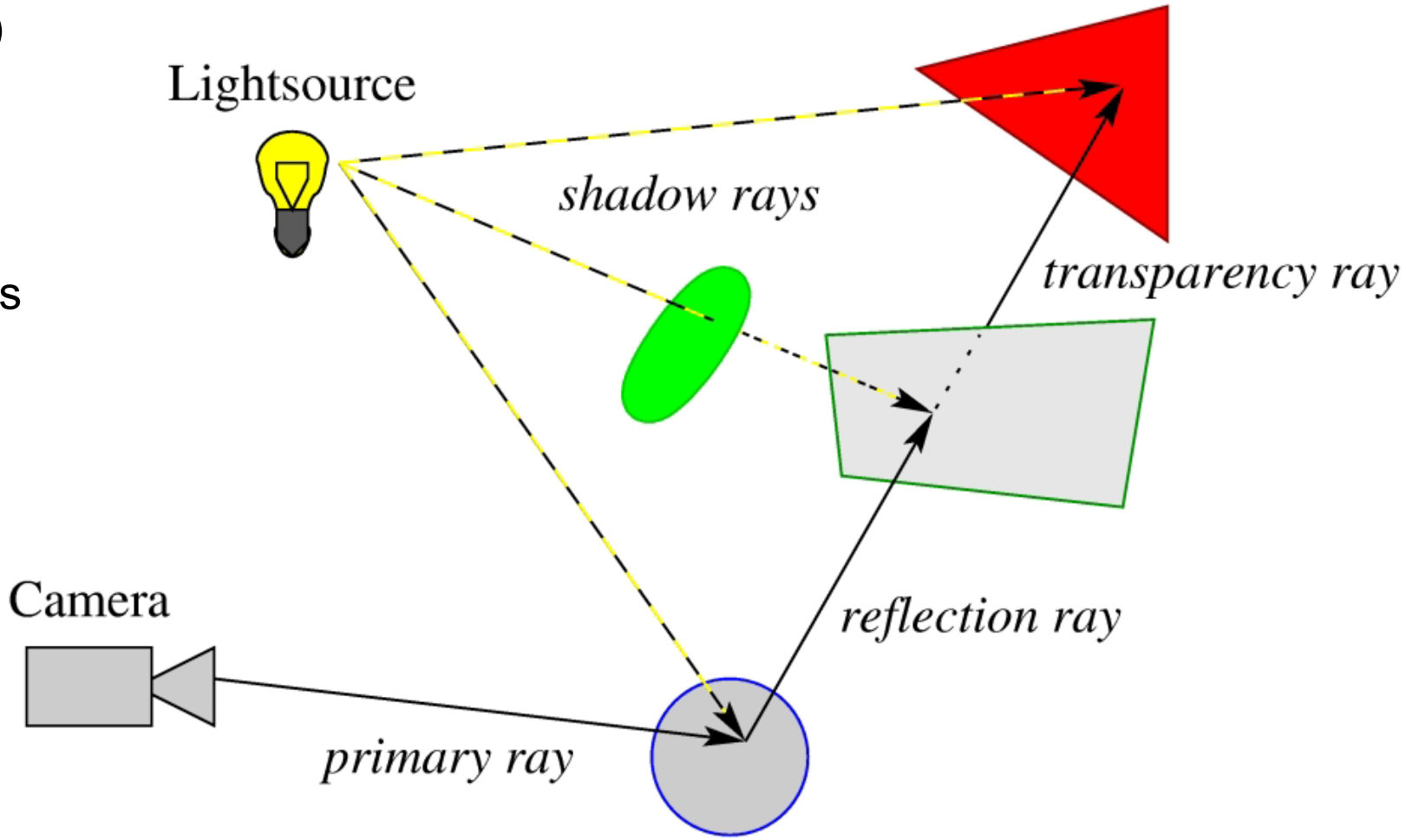
Ray Tracing Pipeline





Ray Tracing

- Global effects
- Parallel (as nature)
- Fully automatic
- Demand driven
- Per pixel operations
- Highly efficient



Ray Tracing

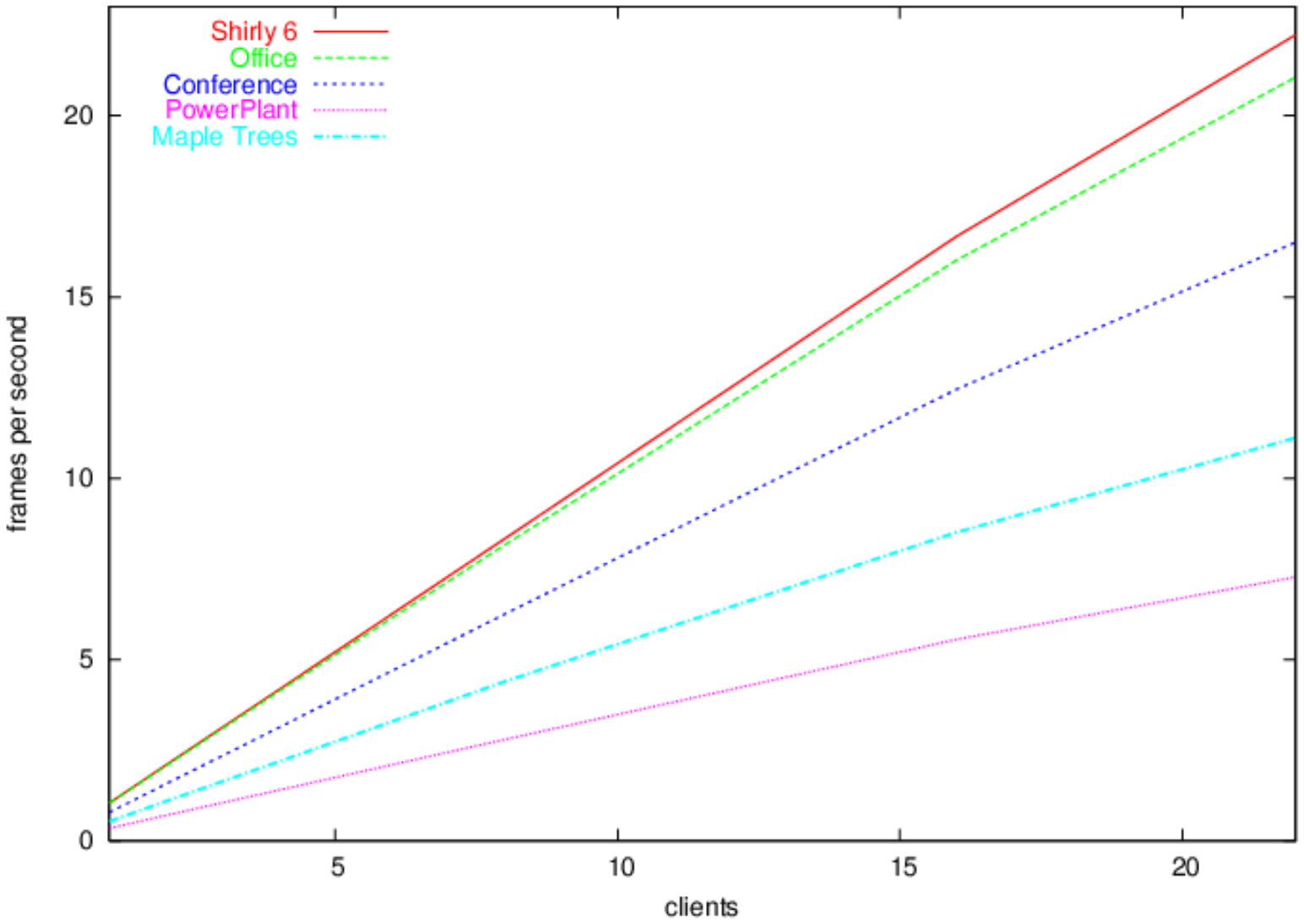
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Offline rendering
  - Only used as an off-line technique
  - Was computationally far too demanding
  - Rendering times of minutes and hours

- Interactive ray tracing (started around 2000)
  - Interactive ray tracing on supercomputers [Parker, U. Utah‘98]
  - Interactive ray tracing on PCs [Wald‘01]
  - Distributed ray tracing on PC clusters [Wald’01]
  - Ray tracing as cloud computing

Scalability

- Highly Scalable
  - Linear in number of
    - Pixels
    - Rays
    - Processors



Enabled by Ray Tracing

- Models Physics of Global Light Transport
  - Dependable, physically-correct visualization

Huge & Realistic 3D Models

Outdoor environment:
~365,000 plants,
~1.5 billion triangles

Rendered in realtime
with skylight illum.
on PC cluster

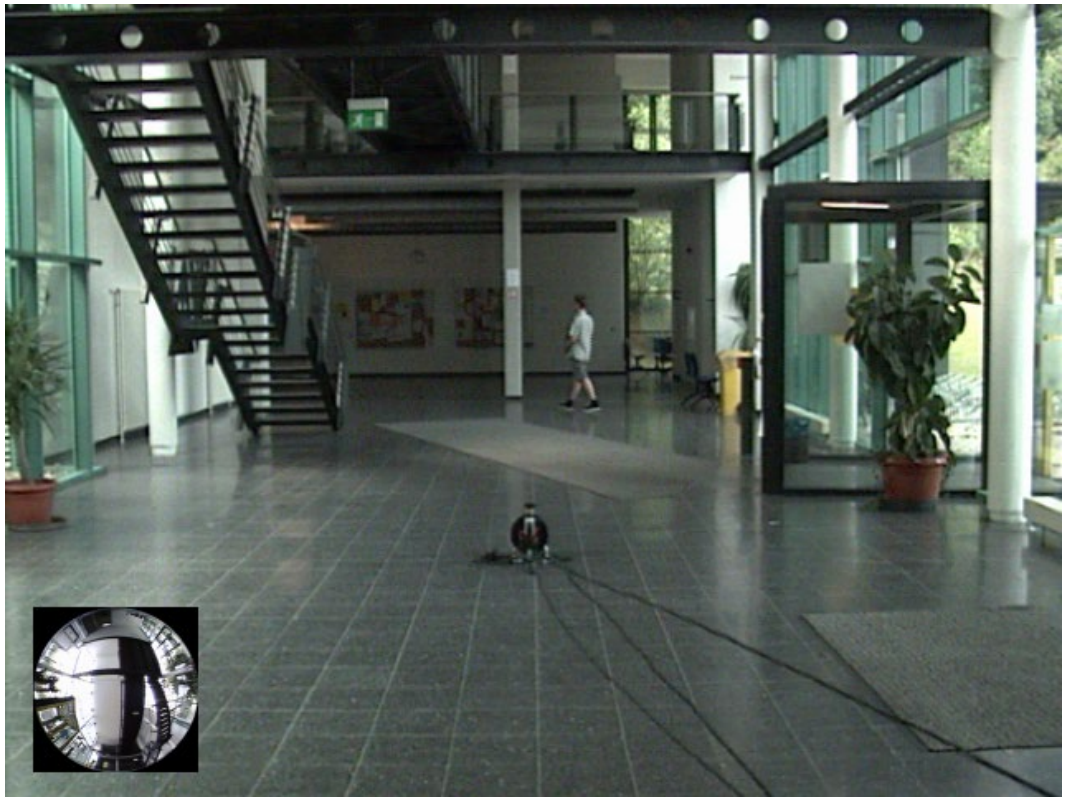
Realtime Lighting Simulation

Measured Materials

- Texture and reflection

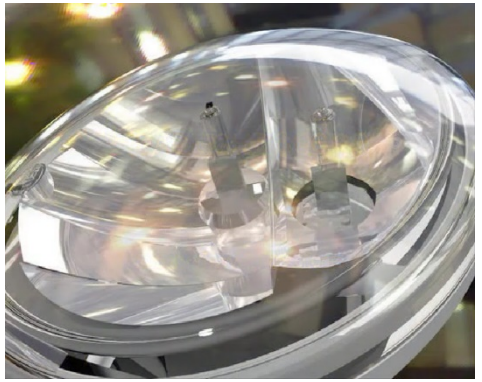
Realistic Visualization: VR/AR

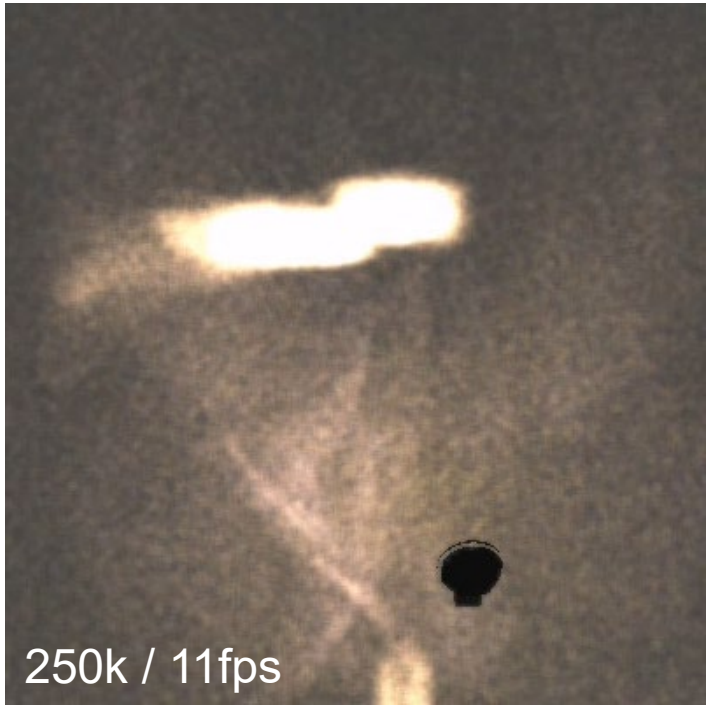
- Lighting measured from environment

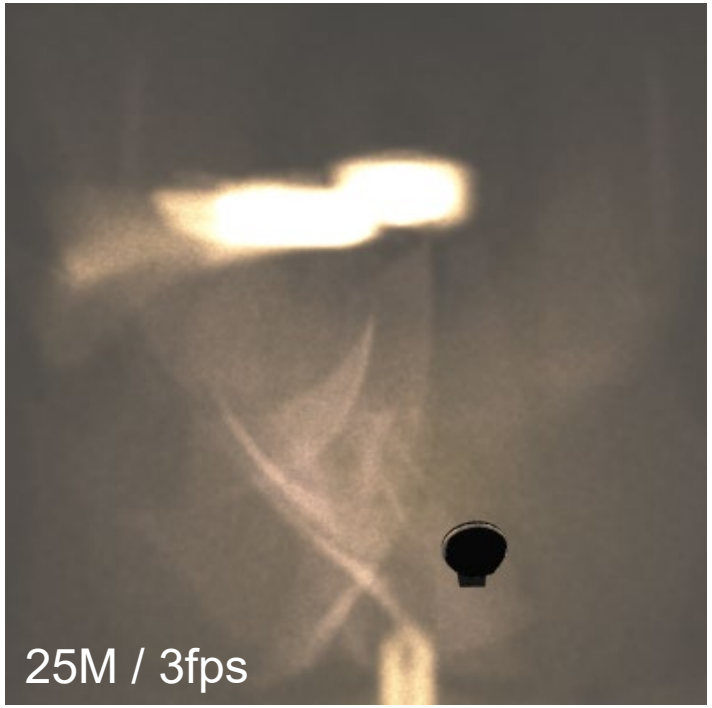


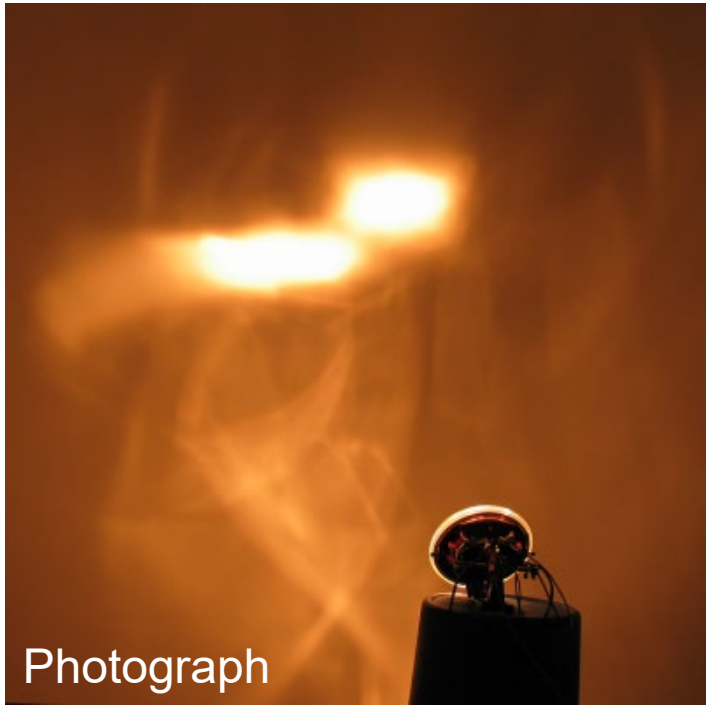
Lighting Simulation

- Complex scattering
- Highly accurate results



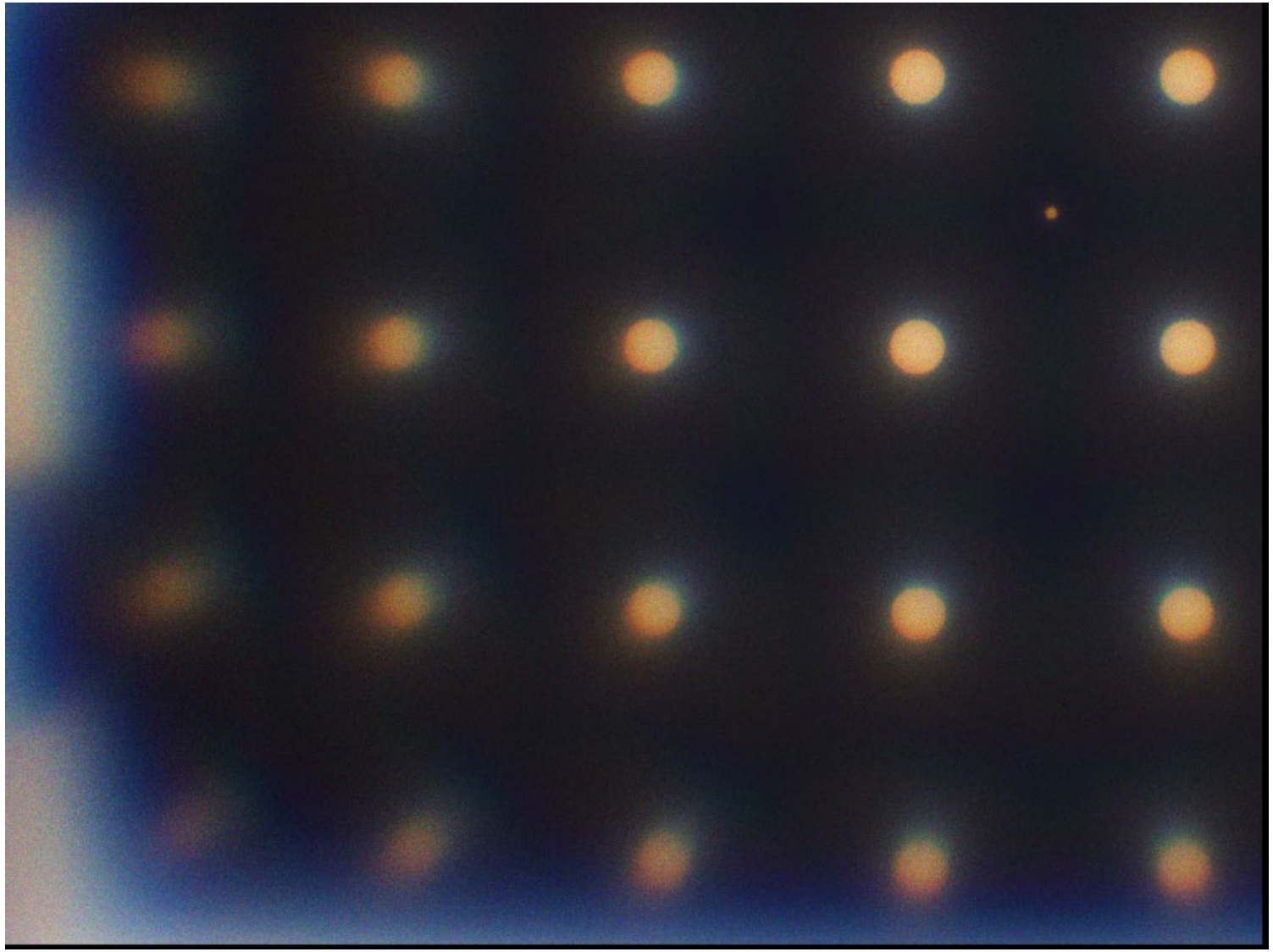






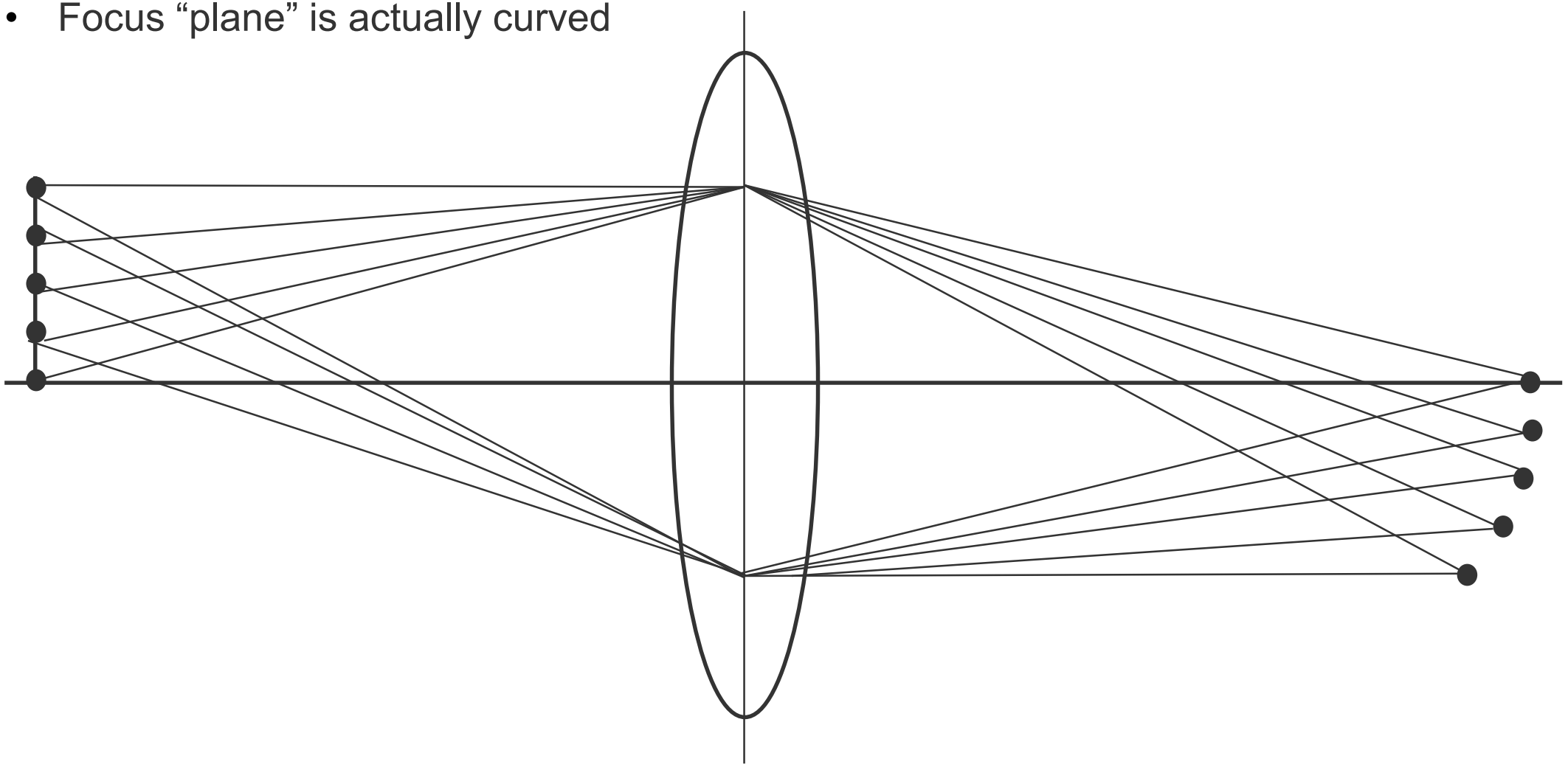
Camera Simulation - Dispersion

- Curvature of field
- Coma
- Chromatic aberration



Camera Simulation - Curvature of Field

- Focus “plane” is actually curved



Camera Simulation – Artificial Artifacts

- Bloom
- Smear

Lens Simulation

- Focus
- Depth of Field

Fundamental Ray Tracing Steps

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Generation of primary rays
  - Rays from viewpoint along viewing directions into 3D scene
  - (At least) one ray per picture element (pixel)

- Ray tracing
  - Traversal of spatial index structures
  - Intersection of ray with scene geometry

- Shading
  - From intersection, determine “light color” sent along primary ray
  - Determines “pixel color”
  - Needed
    - Local material color and reflection properties
      - Object texture
    - Local illumination of intersection point
      - Can be hard to determine correctly

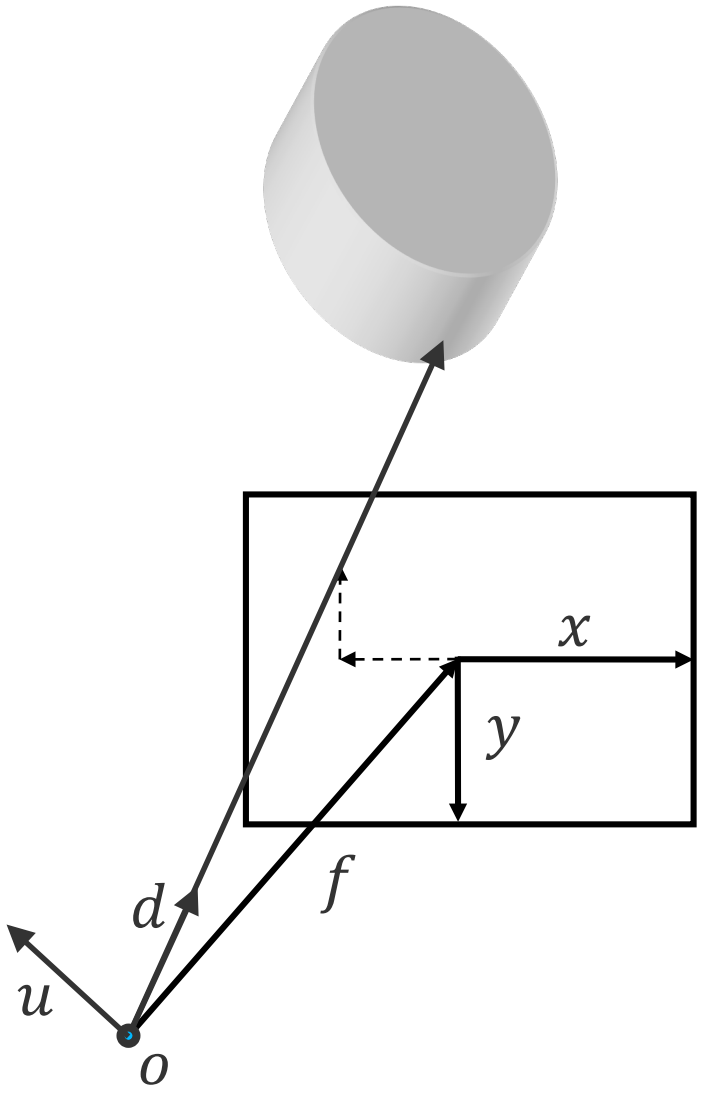
Intersections

Where does the ray hit an object?

Perspective Camera Model

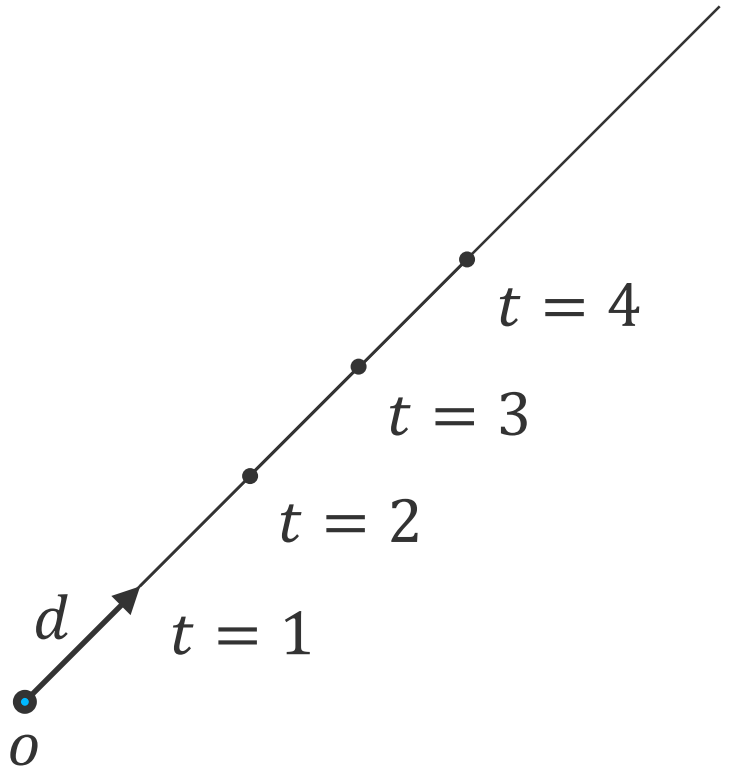
- Definition of the pinhole camera
  - o : Origin (point of view)
  - f : Vector to center of view (focal length)
  - u : Up-vector of camera orientation, in one plane with y vector
  - x, y : Span half the viewing window (frustum) relative to coordinate system (o, f, u)
  - $xres, yres$: Image resolution

```
for j in range(yres):
    for i in range(xres):
        d = f+2*(i/xres-0.5)*x
              +2*(j/yres-0.5)*y
        d = d/norm(d) # normalize
        col = trace(o, d)
        write_pixel(i, j, col)
```



Ray and Object Representation

- Ray in space: $\mathrm{r}(t) = o + td$
  - $o = (o_x, o_y, o_z)^T$
  - $d = (d_x, d_y, d_z)^T$

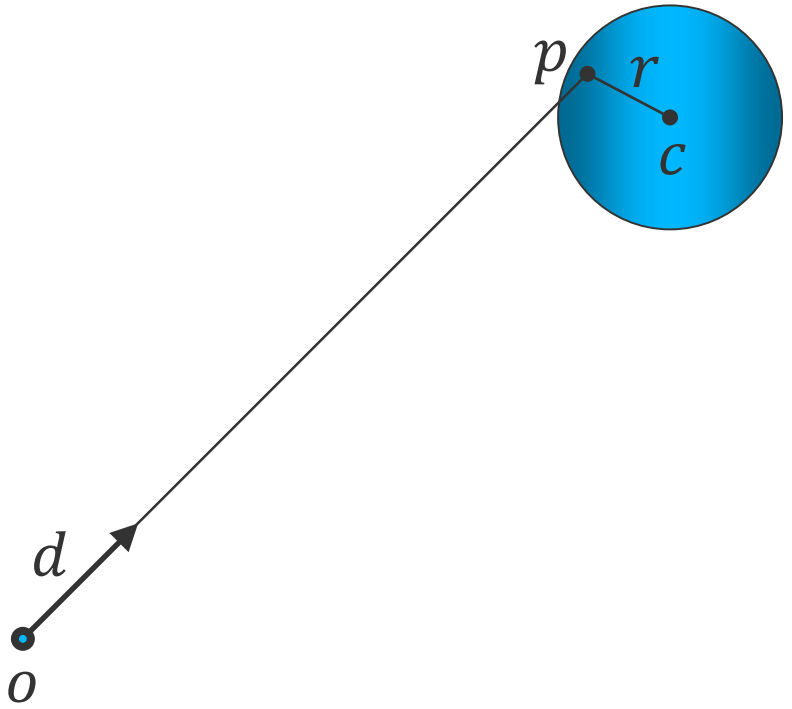


ray parameterization

Ray and Object Representation

- Ray in space: $\mathrm{r}(t) = o + td$
  - $o = (o_x, o_y, o_z)^T$
  - $d = (d_x, d_y, d_z)^T$

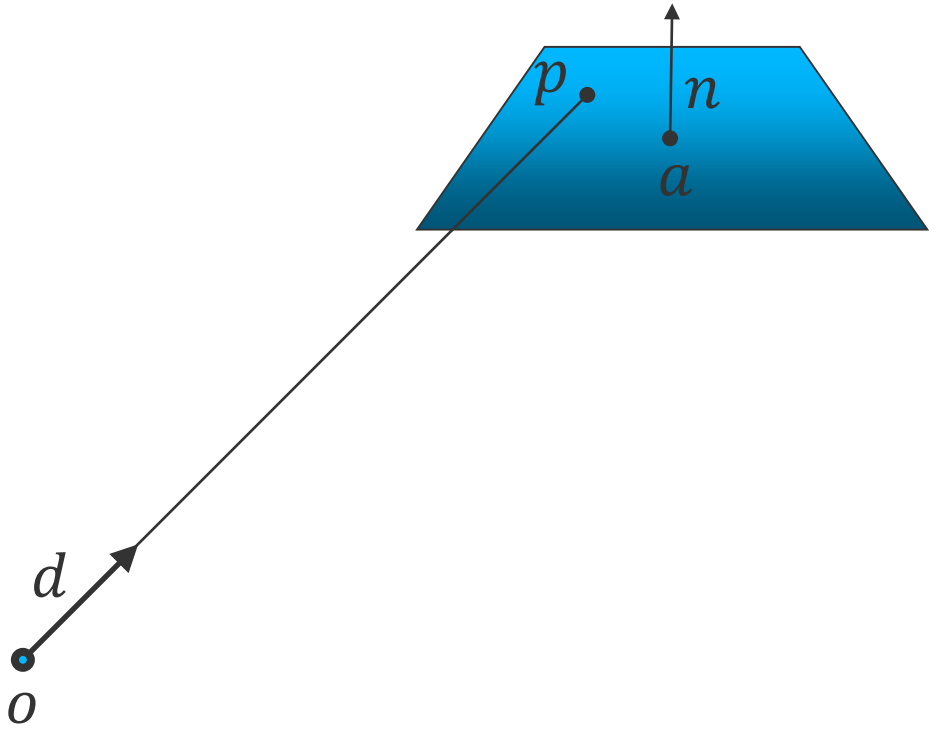
- Scene geometry
  - Sphere: $0 = (p - c)\cdot(p - c) - r^2$
    - c : sphere center
    - r : sphere radius
    - p : any surface point



Ray and Object Representation

- Ray in space: $\mathrm{r}(t) = o + td$
  - $o = (o_x, o_y, o_z)^T$
  - $d = (d_x, d_y, d_z)^T$

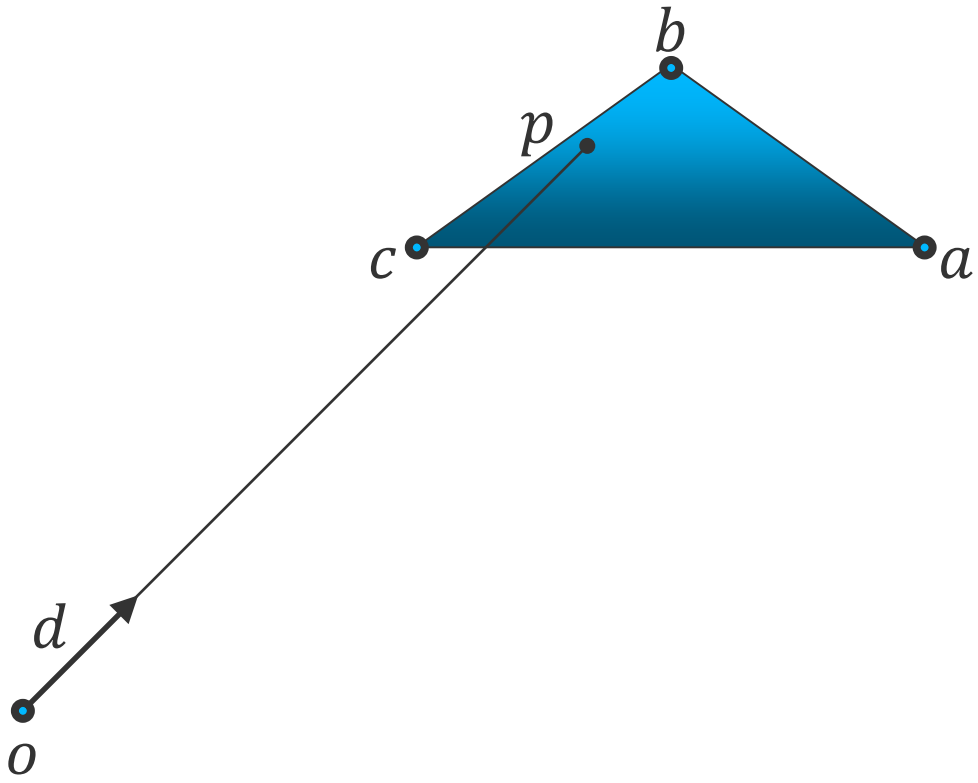
- Scene geometry
  - Sphere: $0 = (p - c)\cdot(p - c) - r^2$
    - c : sphere center
    - r : sphere radius
    - p : any surface point
  - Plane: $0 = (p - a)\cdot n$
    - Implicit definition
    - n : surface normal
    - a : one given surface point
    - p : any surface point



Ray and Object Representation

- Ray in space: $\mathrm{r}(t) = o + td$
  - $o = (o_x, o_y, o_z)^T$
  - $d = (d_x, d_y, d_z)^T$

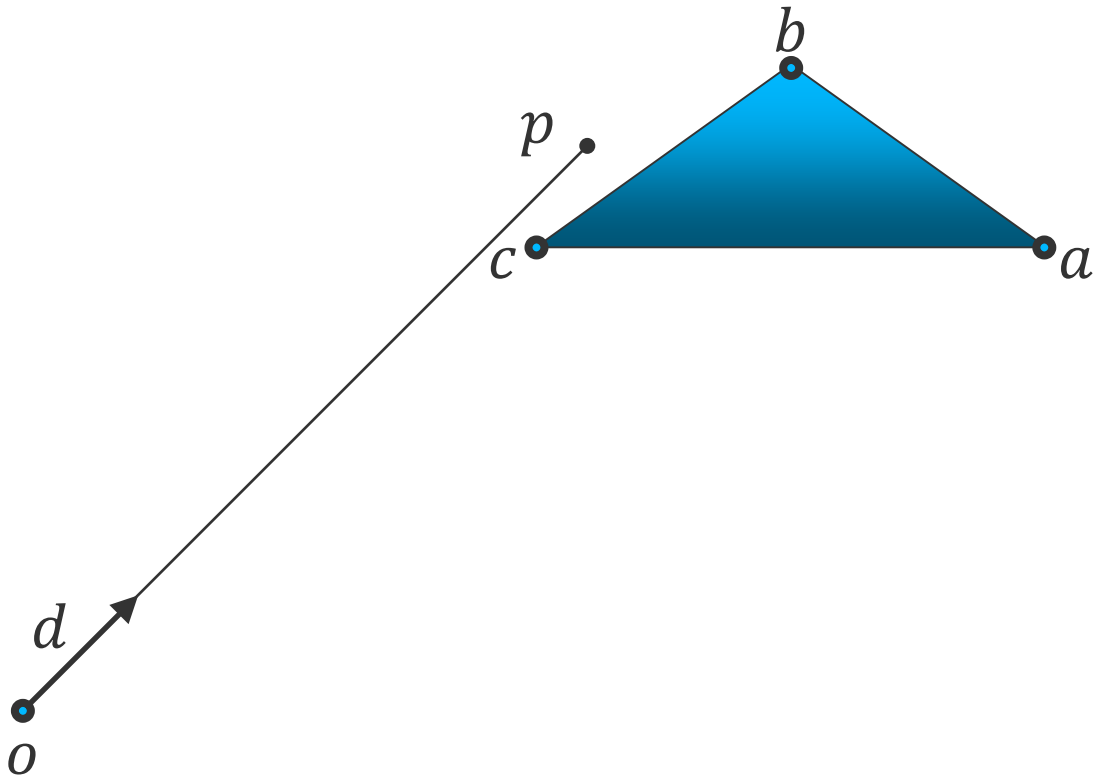
- Scene geometry
  - Sphere: $0 = (p - c)\cdot(p - c) - r^2$
    - c : sphere center
    - r : sphere radius
    - p : any surface point
  - Plane: $0 = (p - a)\cdot n$
    - Implicit definition
    - n : surface normal
    - a : one given surface point
    - p : any surface point
  - Triangle:
    - Plane intersection
      plus barycentric coordinates



Ray and Object Representation

- Ray in space: $\mathrm{r}(t) = o + td$
  - $o = (o_x, o_z, o_z)^T$
  - $d = (d_x, d_z, d_z)^T$

- Scene geometry
  - Sphere: $0 = (p - c)\cdot(p - c) - r^2$
    - c : sphere center
    - r : sphere radius
    - p : any surface point
  - Plane: $0 = (p - a)\cdot n$
    - Implicit definition
    - n : surface normal
    - a : one given surface point
    - p : any surface point
  - Triangle:
    - Plane intersection
      plus barycentric coordinates

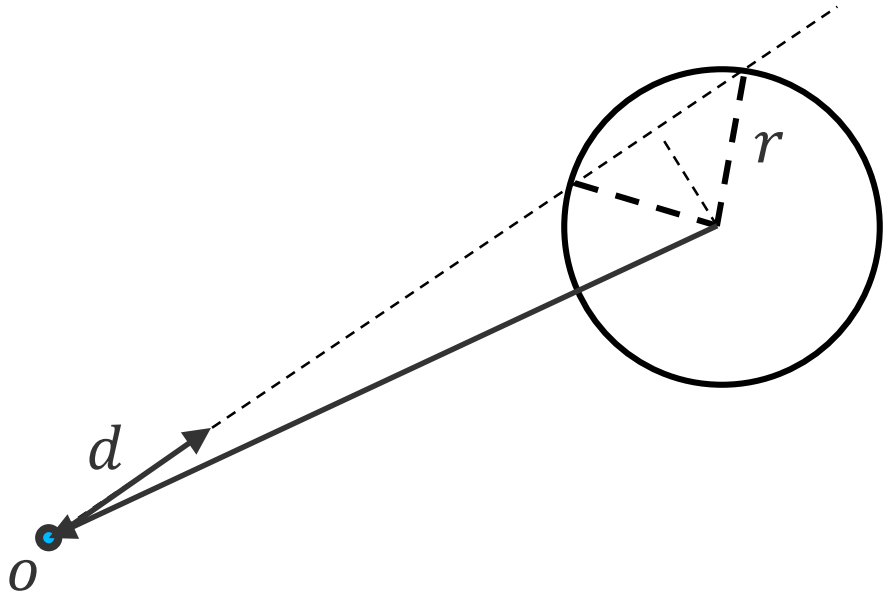


Intersection Ray – Sphere

- Sphere
  - Given a unit sphere at the origin $0 = x^2 + y^2 + z^2 - 1$
  - Given a ray $r(t) = o + td$

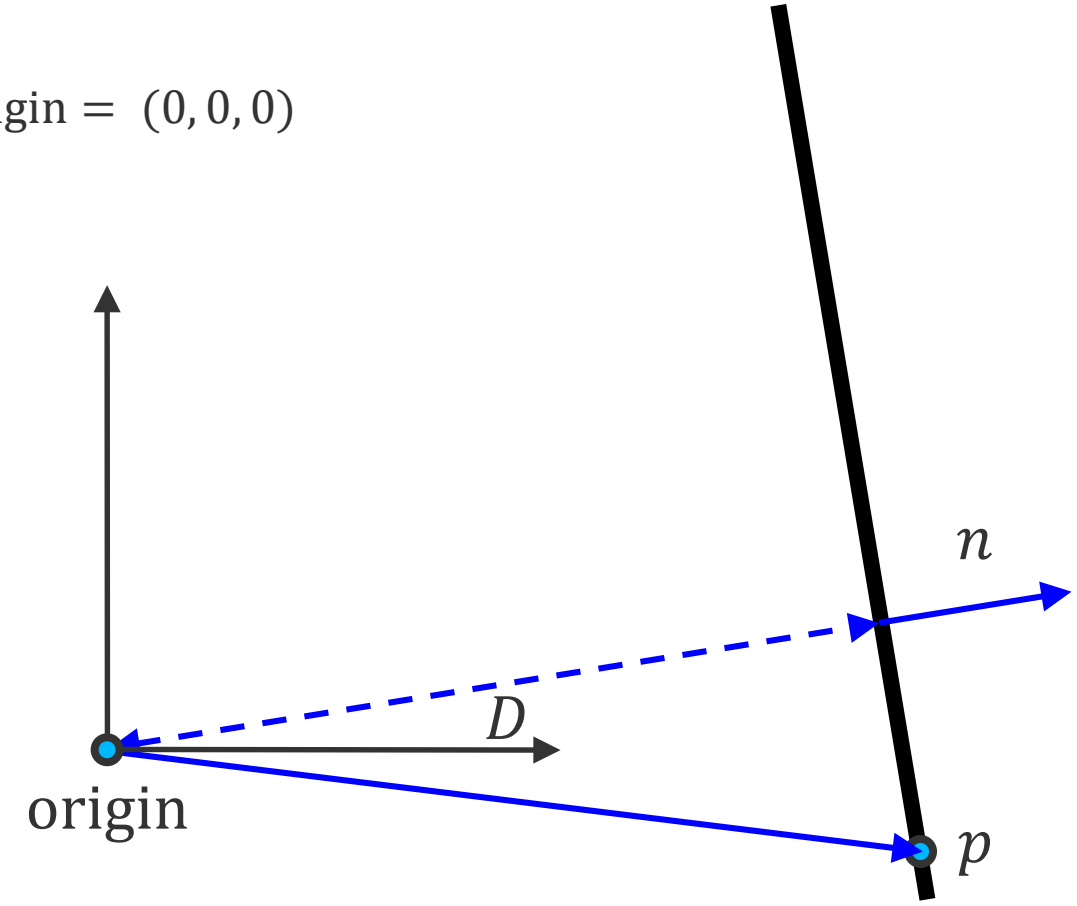
  - Substituting the ray into the equation for the sphere gives
    - $0 = t^2\left(d_x^2 + d_y^2 + d_z^2\right) + 2t\left(o_x d_x + o_y d_y + o_z d_z\right) + \left(o_x^2 + o_y^2 + o_z^2\right) - 1$
    - Easily solvable with standard techniques
    - But beware of numerical imprecision

  - Alternative: Geometric construction
    - Ray and center span a plane
    - Simple 2D construction



Intersection Ray – Plane

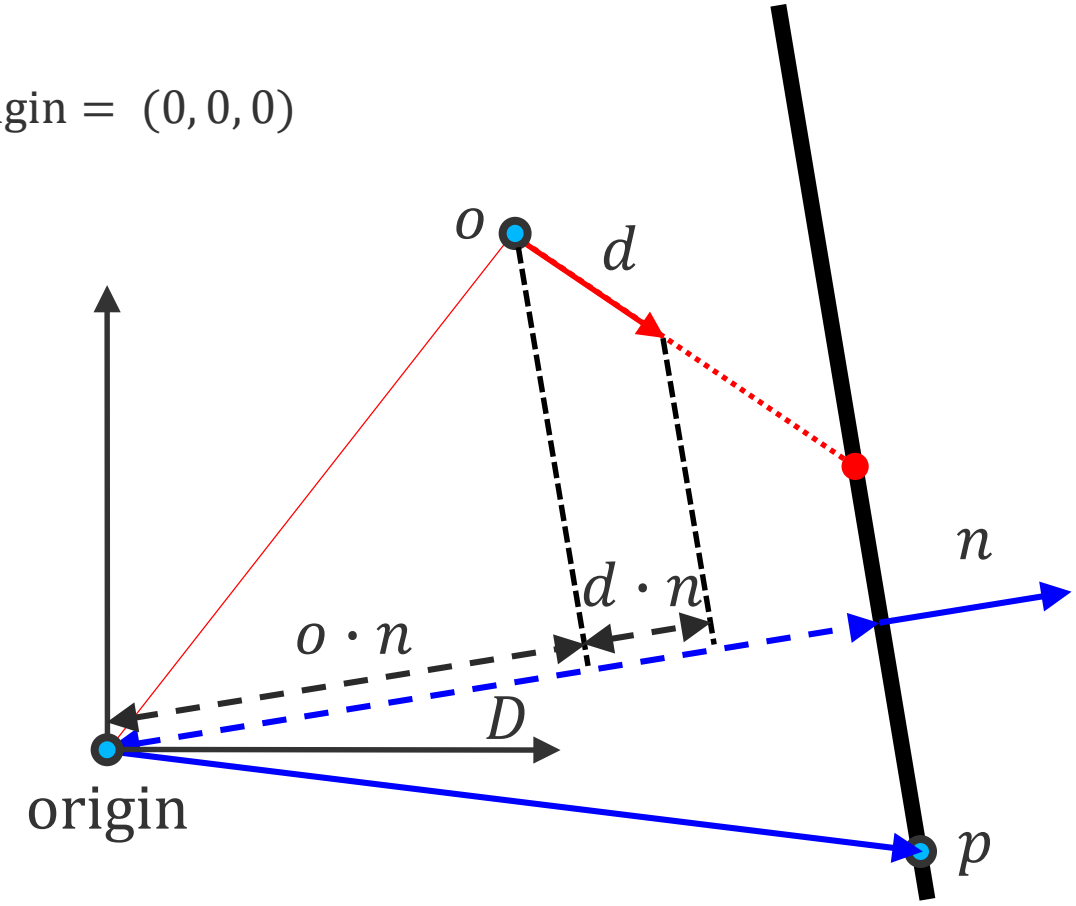
- Plane: Implicit representation (Hesse form)
  - Plane equation: $0 = p \cdot n - D$
    - n : Normal vector $|n| = 1$
    - p : Point on the plane
    - D : Normal distance of plane from origin $= (0, 0, 0)$



Intersection Ray – Plane

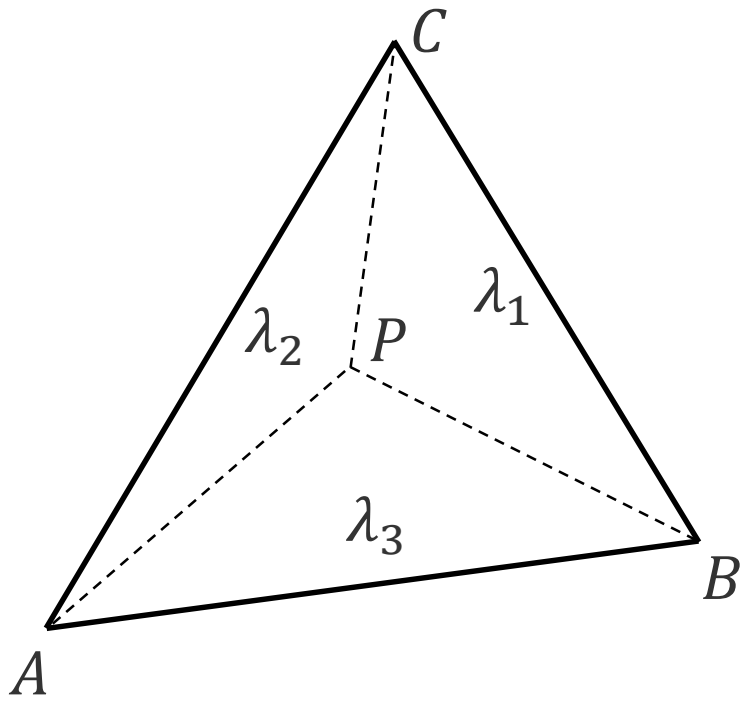
- Plane: Implicit representation (Hesse form)
  - Plane equation: $0 = p \cdot n - D$
    - n : Normal vector $|n| = 1$
    - p : Point on the plane
    - D : Normal distance of plane from origin $= (0, 0, 0)$

- Two possible approaches
  - Geometric
  - Algebraic
    - Substitute ray $\mathrm{r(t)} = o + td$ for p
    - $(o + td) \cdot n - D = 0$
    - Solving for t gives $t = \frac{D - o \cdot n}{d \cdot n}$



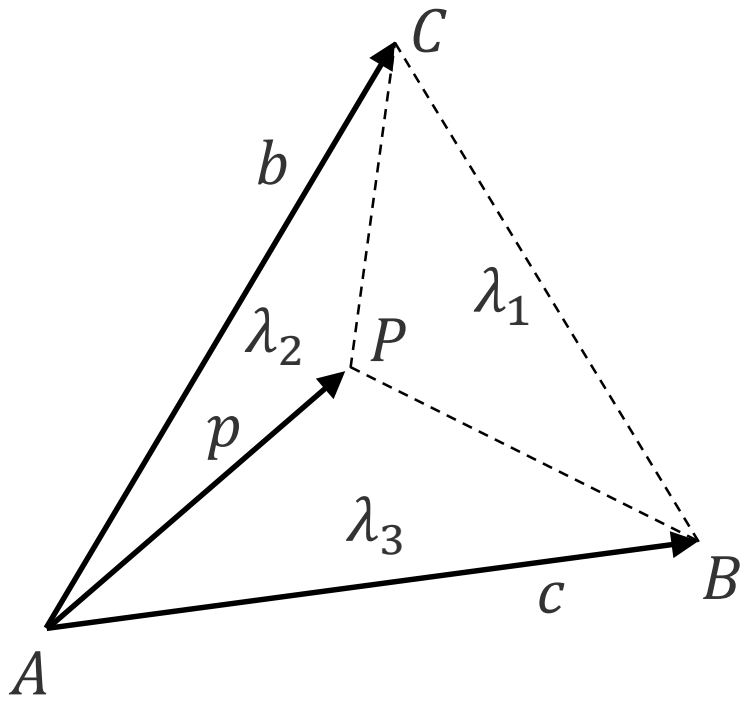
Intersection Ray – Triangle (1)

- Barycentric coordinates
  - Non-degenerate triangle ABC
  - Ratio of signed areas: $\lambda_1 = \Delta(BCP)/\Delta(ABC)$
    $\lambda_2 = \Delta(CAP)/\Delta(ABC)$
    $\lambda_3 = \Delta(ABP)/\Delta(ABC)$
  - Every point P in the plane can be described using
    - $P = \lambda_1 A + \lambda_2 B + \lambda_3 C$
  - $\boldsymbol{\lambda_1 + \lambda_2 + \lambda_3 = 1}$
    - For fixed λ_3, P may move parallel to AB
    - For $\gamma_1 + \gamma_2 = 1$ and $0 < \gamma_3 < 1$
      - $P = (1 - \gamma_3)(\gamma_1 A + \gamma_2 B) + \gamma_3 C$
    - P moves between C and AB
- Point is in triangle, iff all $0 \leq \lambda_i$



Calculation of Barycentric Coordinates 2D

- Hitpoint on 2D Plane P
  - $b = C - A$
  - $c = B - A$
  - $p = P - A$
  - $\lambda_2 = \frac{b_x p_y - b_y p_x}{b_x c_y - b_y c_x}$
  - $\lambda_3 = \frac{p_x c_y - p_y c_x}{b_x c_y - b_y c_x}$
  - $\lambda_1 = 1 - \lambda_2 - \lambda_3$



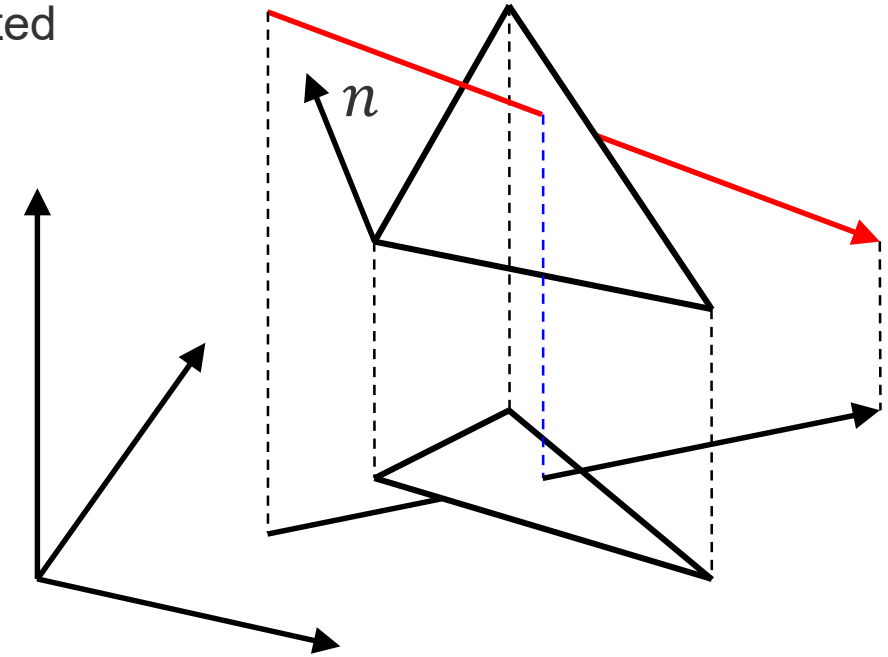
Intersection Ray – Triangle (1)

- Compute intersection with triangle plane

- Given the 3D intersection point
  - Project point into xy, xz, yz coordinate plane
  - Use coordinate plane that is most aligned
    - xy: if n_z is maximal, etc.
  - Coordinate plane and 2D vertices can be pre-computed

- Compute barycentric coordinates in 2D

- Test for positive BCs



Intersection Ray – Triangle (1)

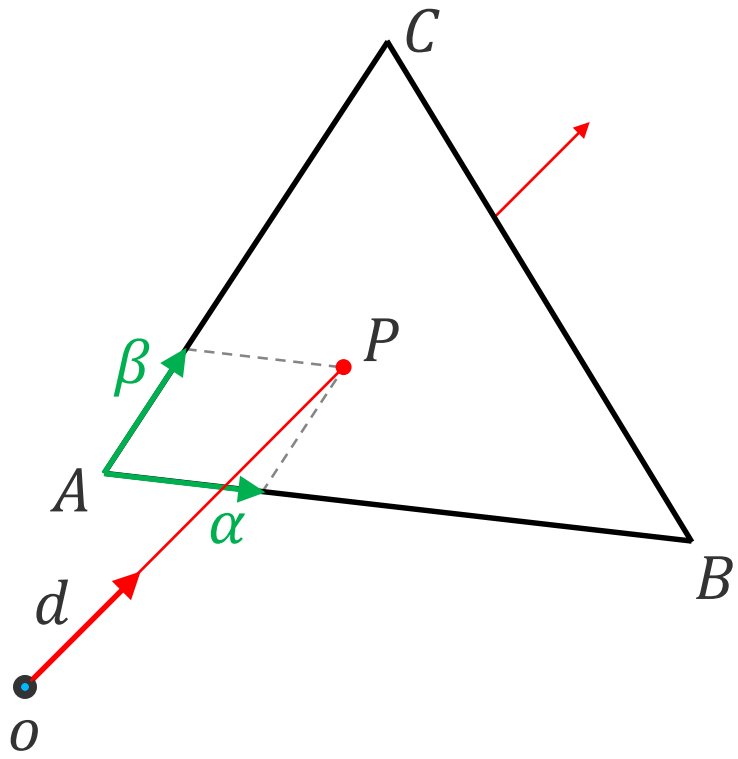
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

```
# calc edges and normal
b = B-A
c = C-A
n = cross(c,b) # normalize
D = dot(A,n)

# distance test
t_plane = (D-dot(o,n)) / dot(d,n)
if (t_plane < eps or t_plane > t_max):
    return NO_HIT

# determine projection dimensions
if (abs(n[0]) > abs(n[1])):
        #x                                  #z
    k = 0 if (abs(n[0]) > abs(n[2])) else 2
else:
        #y                                  #z
    k = 1 if (abs(n[1]) > abs(n[2])) else 2

u = (k+1) % 3
v = (k+2) % 3
```

```
# calculate hitpoint
p[u] = O[u] + t_plane * d[u] - A[u]
p[v] = O[v] + t_plane * d[v] - A[v]

beta = (b[u]*p[v] - b[v]*p[u]) /
       (b[u] * c[v] - b[v] * c[u])

if beta < 0:
    return NO_HIT

gamma = (c[v]*p[u]-c[u]*p[v] /
        (b[u] * c[v] - b[v] * c[u])

if gamma < 0:
    return NO_HIT

if (beta + gamma) > 1:
    return NO_HIT;

return HIT(t_plane, beta, gamma)
```

Intersection Ray – Triangle (2)

- Check if point is inside triangle parametrically
- Compute α, β
  - $P = \alpha(B - A) + \beta(C - A)$
- Check if point inside triangle
  - $0 \leq \alpha$
  - $0 \leq \beta$
  - $0 \leq \alpha + \beta \leq 1$

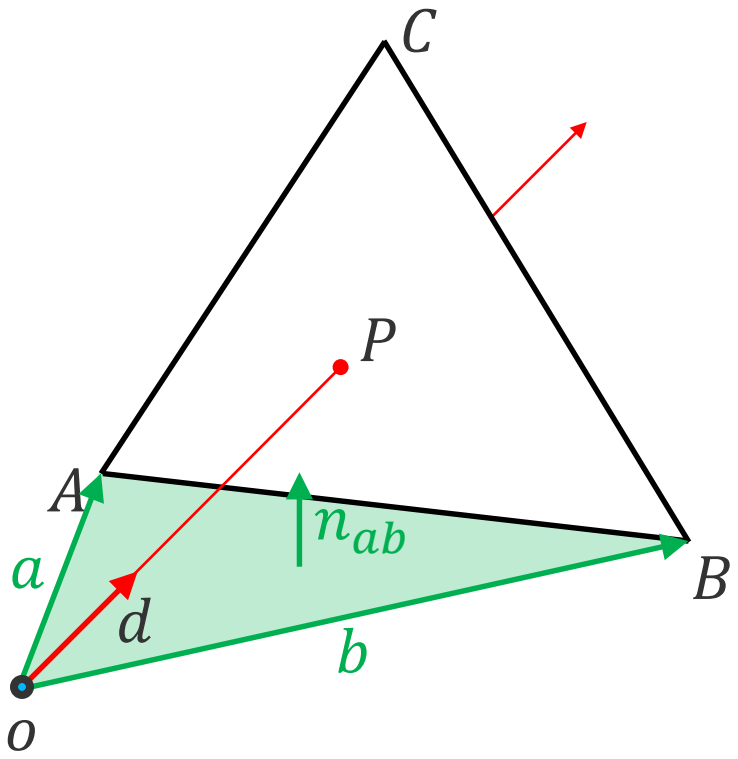


Intersection Ray – Triangle (3)

- Check if point is inside triangle algebraically

- Compute
  - $\langle a, b, c \rangle = \langle A, B, C \rangle - o$
  - $n_{ab} = \frac{a \times b}{|a \times b|}$
  - $n_{bc} = \frac{b \times c}{|b \times c|}$
  - $n_{ca} = \frac{c \times a}{|c \times a|}$

- for $i \in (ab, bc, ca)$:
  - $s_i = o - n_i$
  - if $P \cdot n_i + s_i < 0$:
    - return false



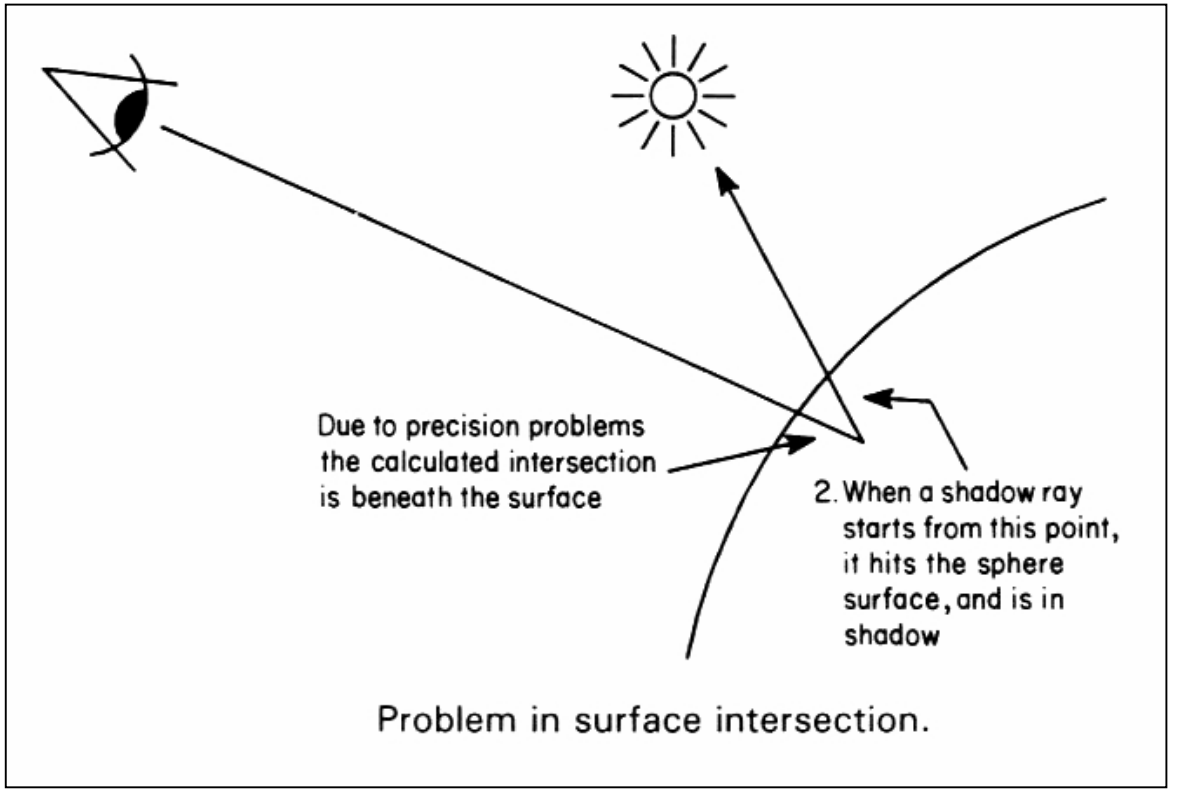
- P is in the triangle if n_{ab}, n_{bc}, n_{ca} all point towards the inside of the triangle

Precision Problems

- Due to floating point arithmetic
- Shadow rays might start just below the surface, reporting the surface itself as occlude
- Solution:
  - check if the value of parameter t is within some tolerance

    $abs(t) < \varepsilon$ with $\varepsilon = 0.000001$

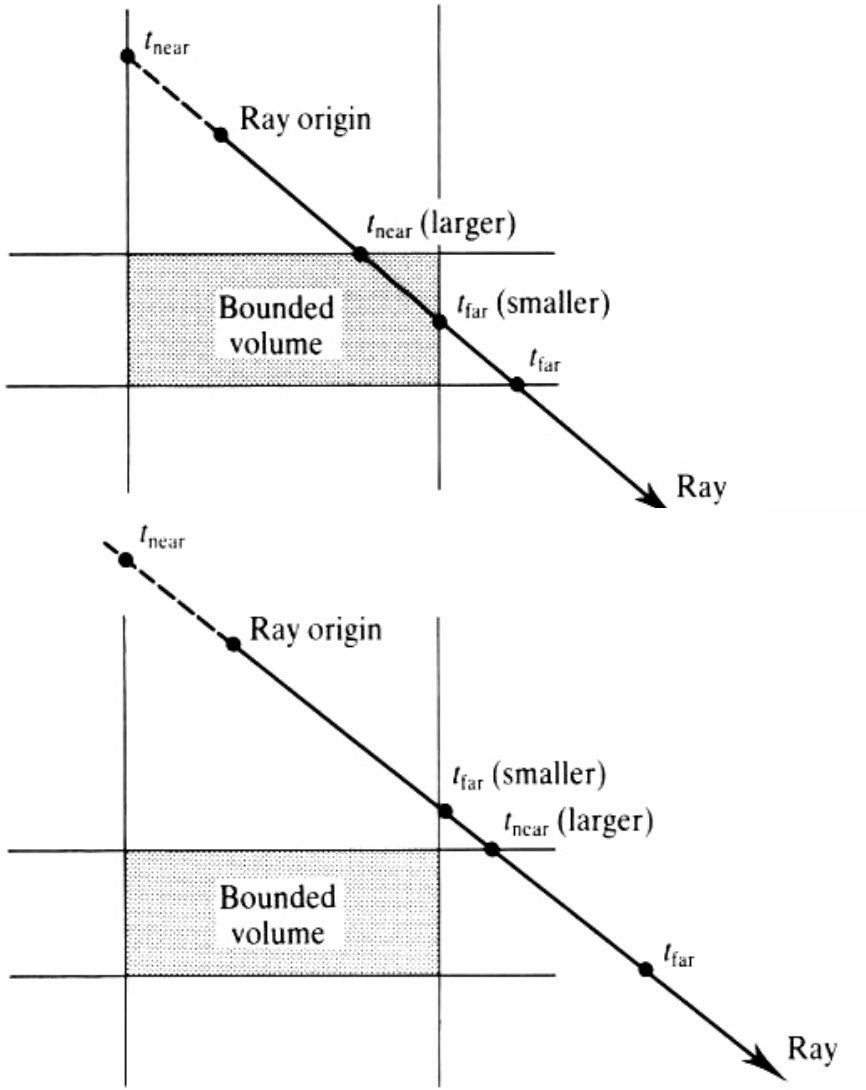
    (Pseudo intersection)



Problem in surface intersection.

Intersection Ray – Box

- Boxes are important for
  - Bounding volumes
  - Hierarchical structures
- Intersection test
  - test pairs of parallel planes in turn
  - calculate intersection distances
    - t_{near} first plane
    - t_{far} second plane
- The ray does not intersect the box if
  - $t_{near} > t_{far}$ for one pair of planes



History of Intersection Algorithms

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Ray-geometry intersection algorithms
  - Polygons: [Appel ’68]
  - Quadrics, CSG: [Goldstein & Nagel ’71]
  - Recursive Ray Tracing: [Whitted ’79]
  - Tori: [Roth ’82]
  - Bicubic patches: [Whitted ’80, Kajiya ’82]
  - Algebraic surfaces: [Hanrahan ’82]
  - Swept surfaces: [Kajiya ’83, van Wijk ’84]
  - Fractals: [Kajiya ’83]
  - Deformations: [Barr ’86]
  - NURBS: [Stürzlinger ’98]
  - Subdivision surfaces: [Kobbelt et al ’98]
  - …

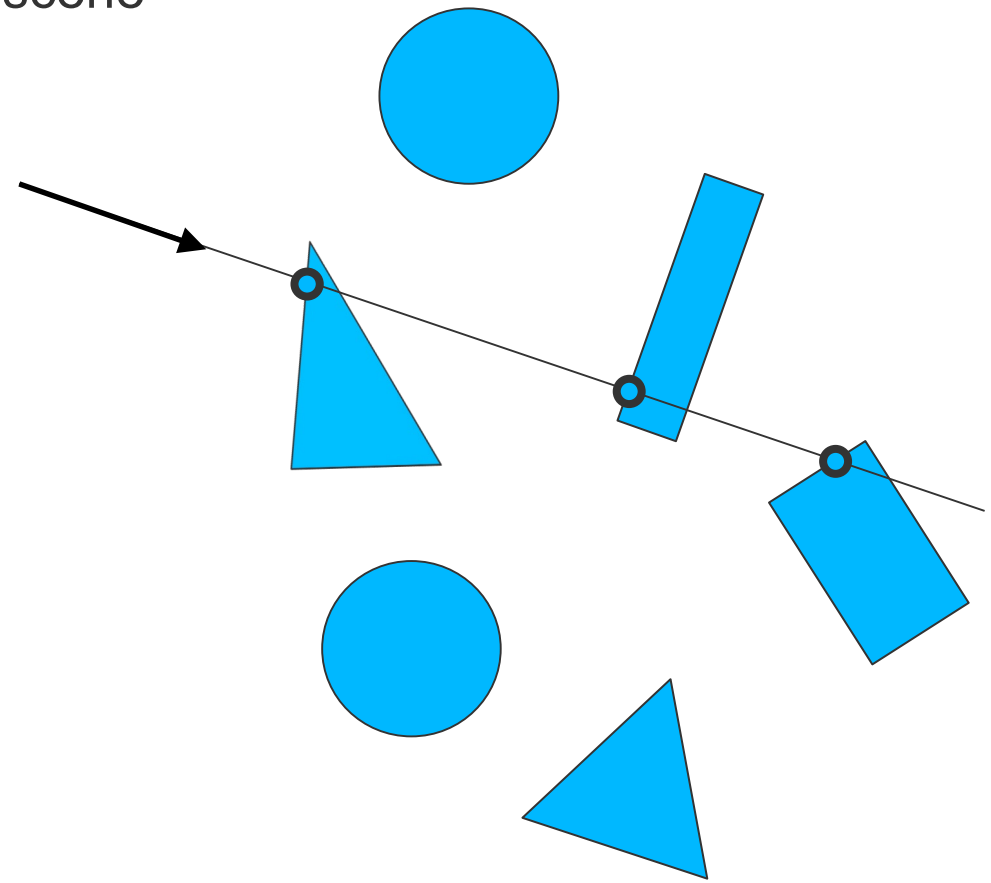
Intersection Ray – Scene

Which object is hit first?

Intersection Ray – Scene

- Find intersection with front-most surface in the scene
- Naïve algorithm:

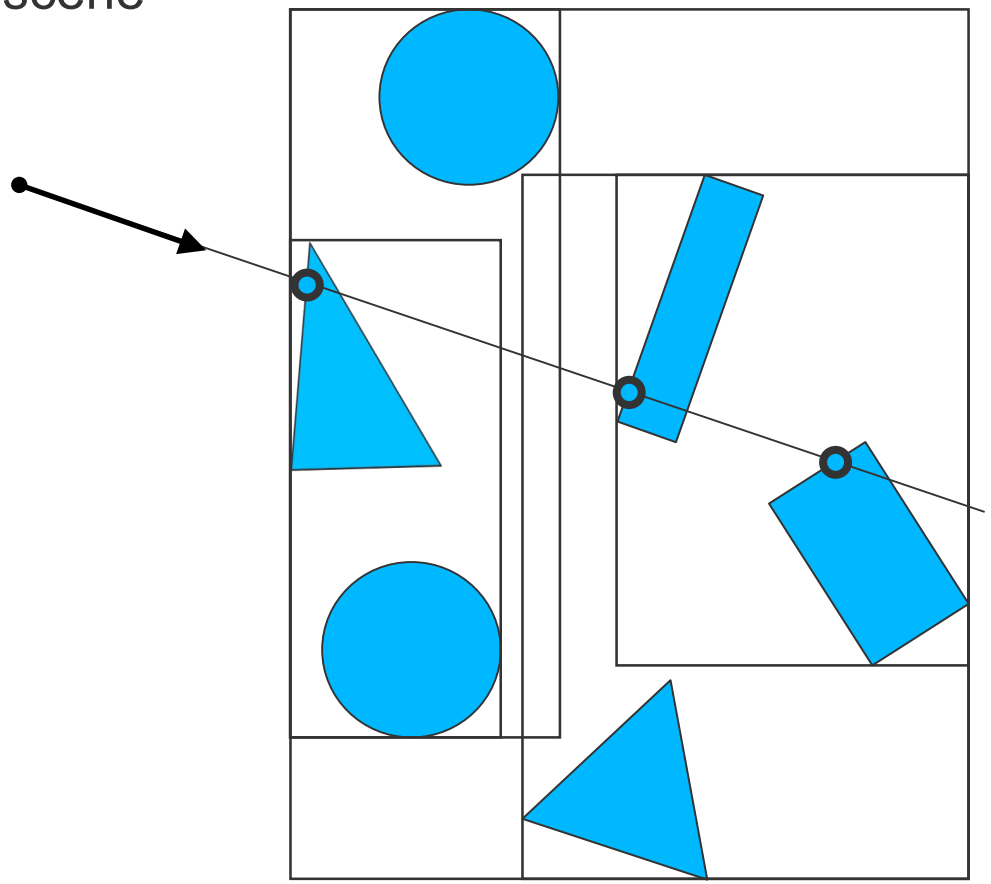
```
def findIntersection(ray, scene):
    min_t = infininty
    min_primitive = None
    for primitive in scene.primitives:
        t = intersect(ray, primitive)
        if t < min_t:
            min_t = t
            min_primitive = primitive
    return min_t, min_primitive
```



- More on acceleration follows later

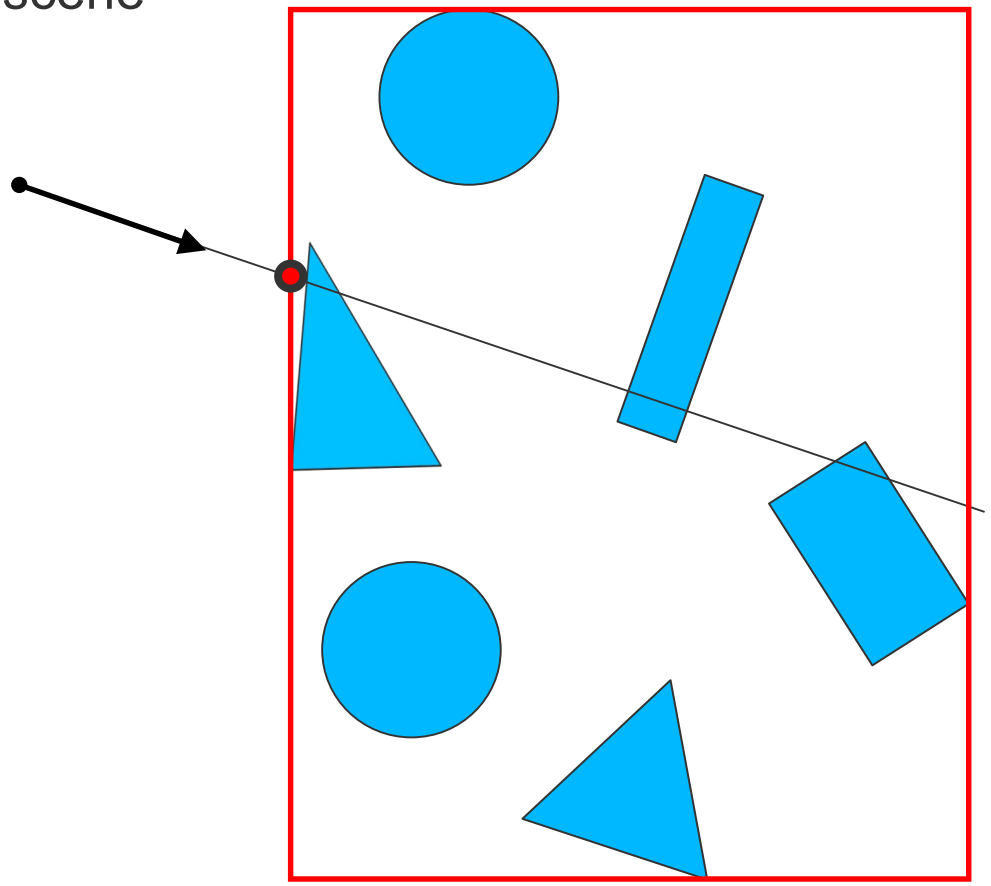
Intersection Ray – Scene BBox

- Find intersection with front-most surface in the scene
- Cluster multiple primitives in bounding box
- More on acceleration follows later



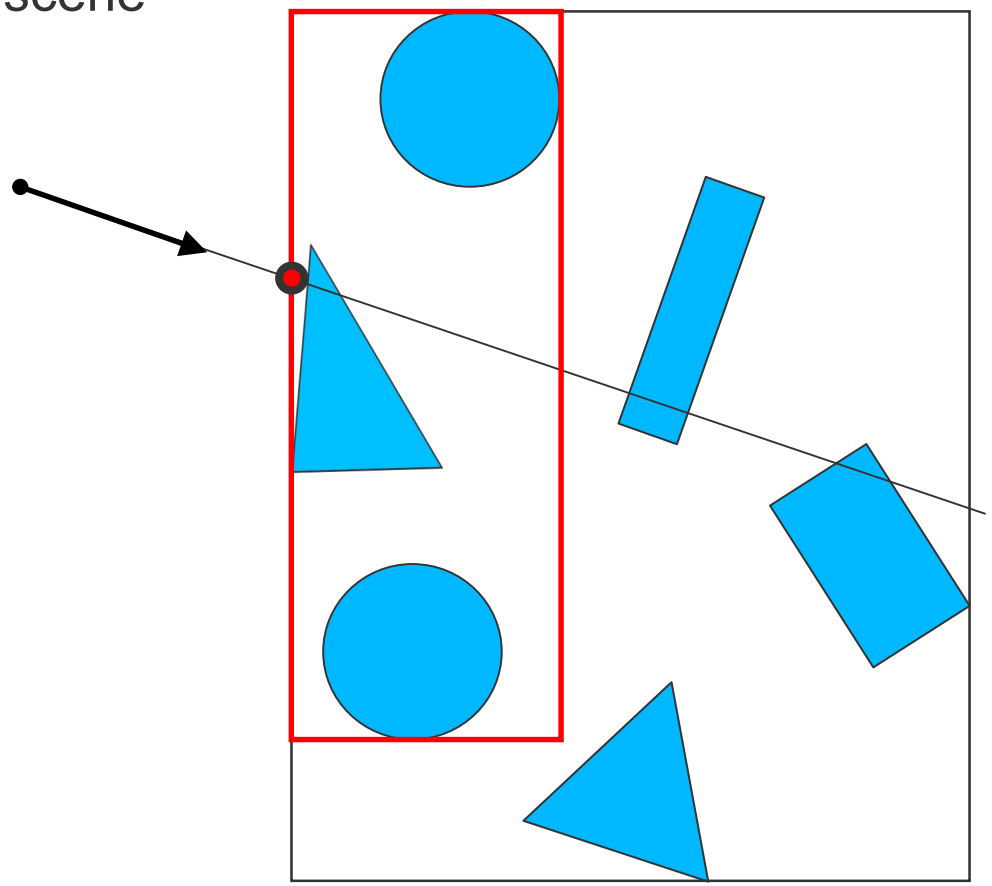
Intersection Ray – Scene BBox

- Find intersection with front-most surface in the scene
- Cluster multiple primitives in bounding box
- More on acceleration follows later



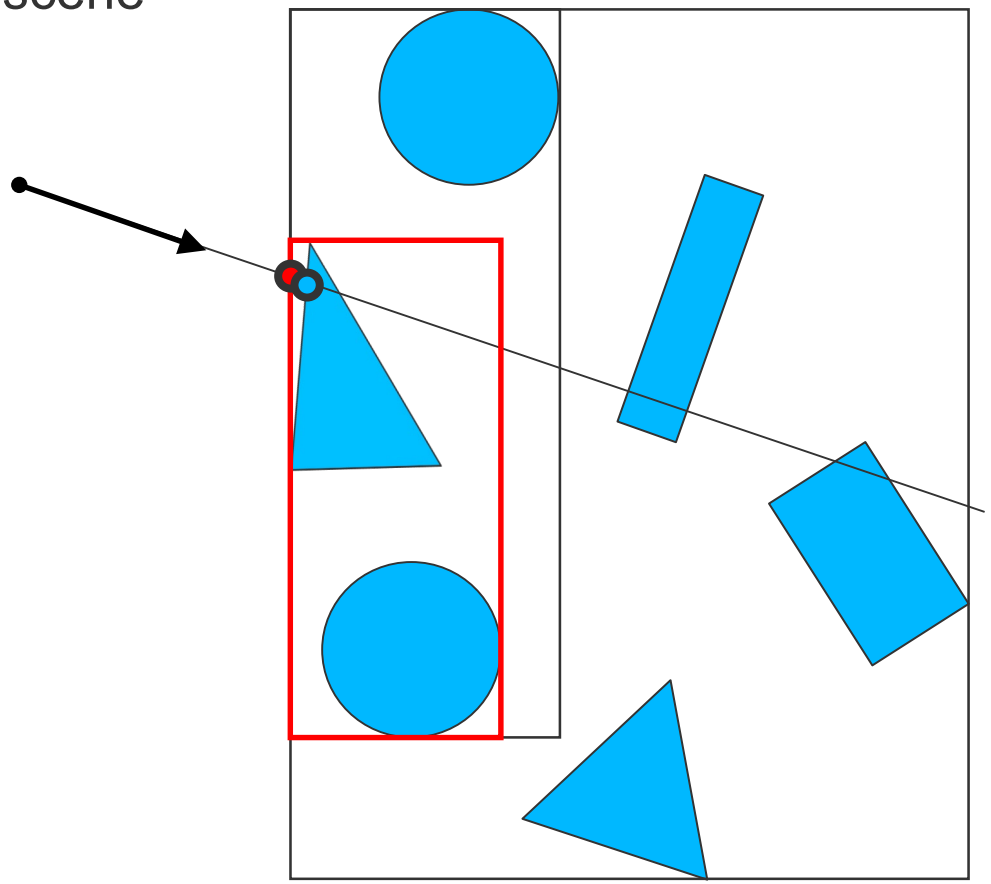
Intersection Ray – Scene BBox

- Find intersection with front-most surface in the scene
- Cluster multiple primitives in bounding box
- More on acceleration follows later



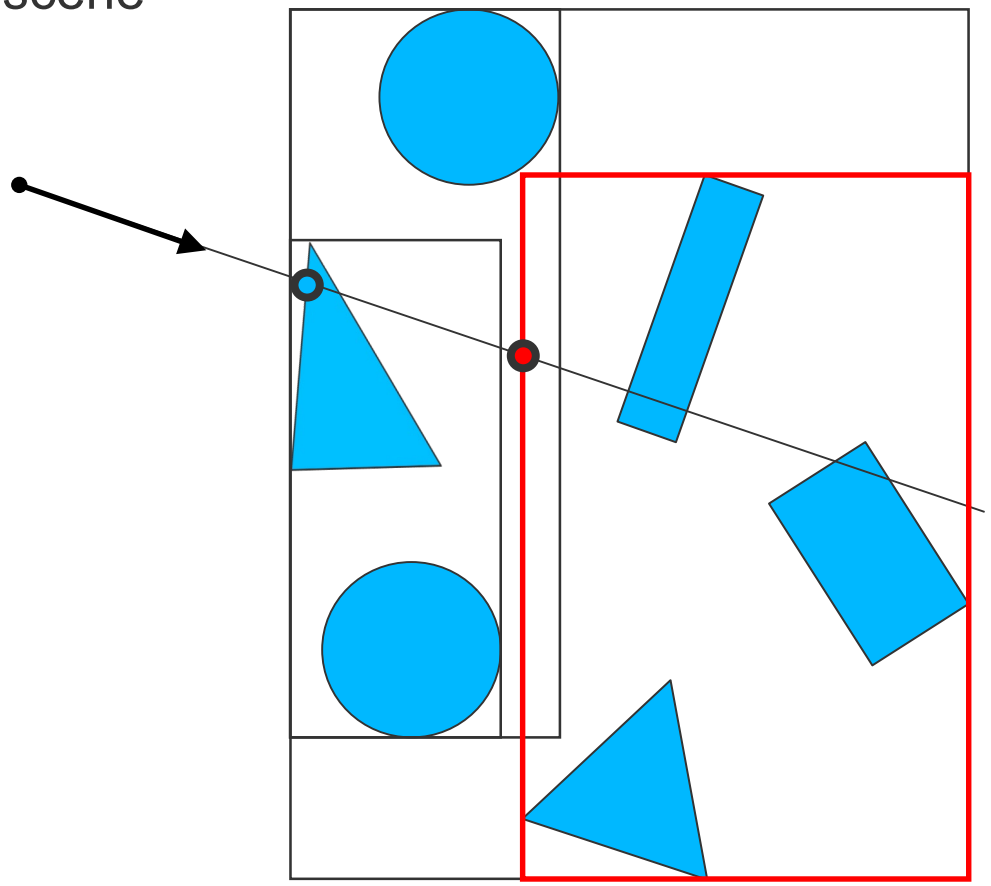
Intersection Ray – Scene BBox

- Find intersection with front-most surface in the scene
- Cluster multiple primitives in bounding box
- More on acceleration follows later



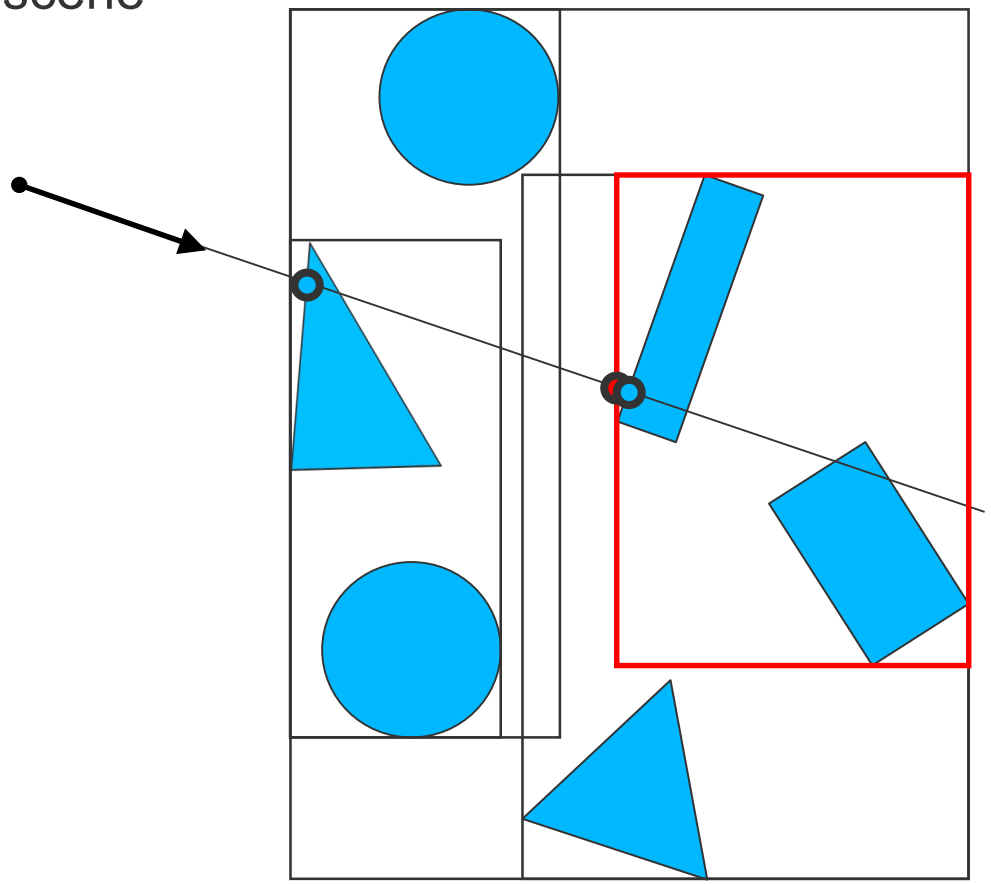
Intersection Ray – Scene BBox

- Find intersection with front-most surface in the scene
- Cluster multiple primitives in bounding box
- More on acceleration follows later



Intersection Ray – Scene BBox

- Find intersection with front-most surface in the scene
- Cluster multiple primitives in bounding box
- More on acceleration follows later



Shading

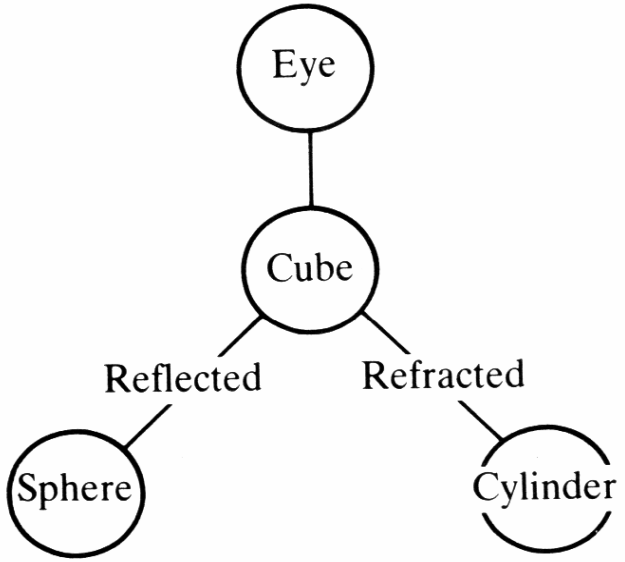
Which *color* do we observe along a ray?

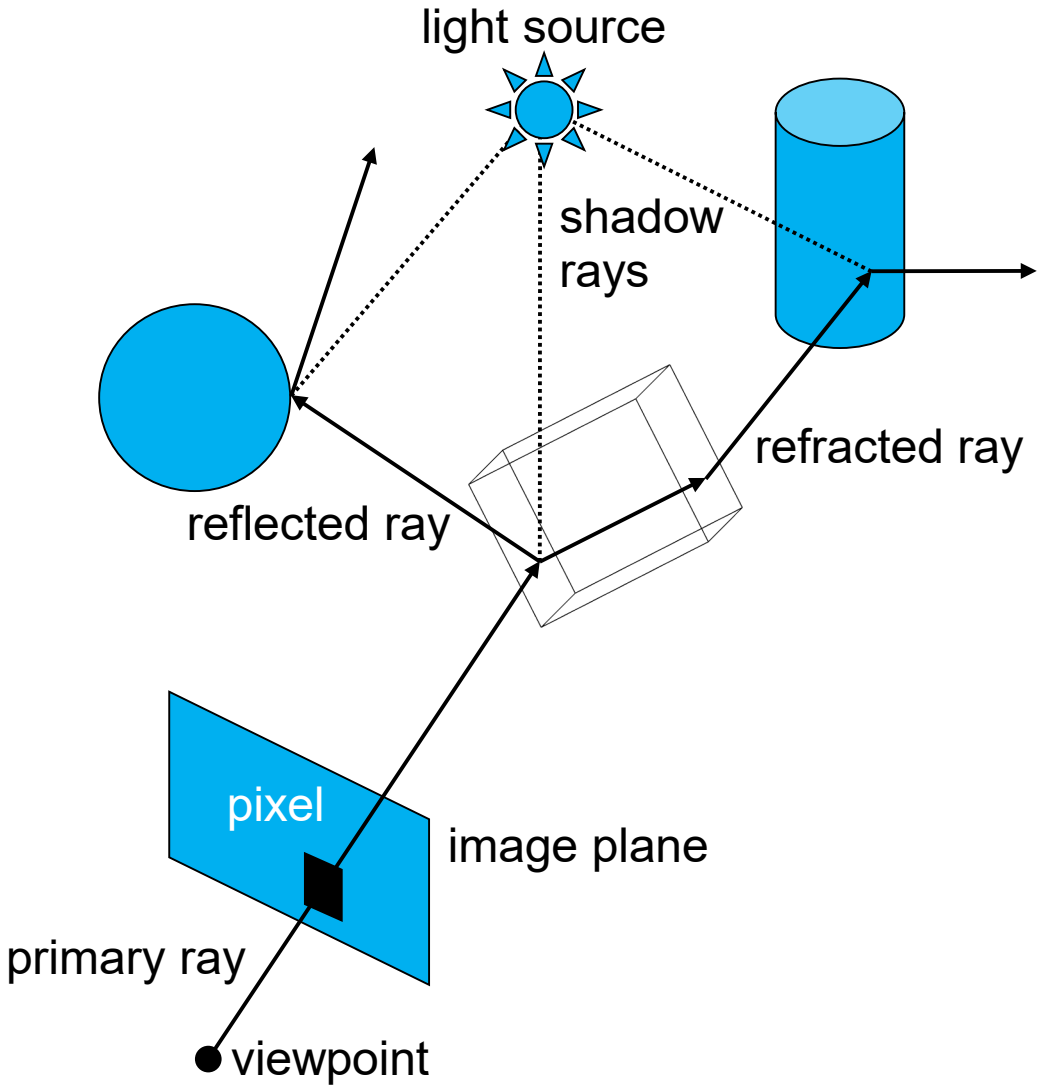
Shading

- Intersection point determines primary ray's *color*
- Diffuse object:
  - Color at intersection point
  - No variation with viewing angle (Lambertian)
  - Must still be illuminated
    - Point light source: shadow ray
    - Scales linearly with received light (Irradiance)
    - No illumination: in shadow = black
- Non-Lambertian Reflectance
  - Appearance depends on illumination and viewing direction
    - Local Bi-directional Reflectance Distribution Function (BRDF)
  - Simple cases:
    - Mirror, glass: secondary rays
- Area light sources and indirect illumination can be difficult

Recursive Ray Tracing

- Search recursively for paths to light sources
  - Interaction of light and material at each intersection point
  - Recursively trace new rays in reflection, refraction, and light direction





Ray Tracing Algorithm


```
def trace(ray, scene):
    hitpoint, material = intersect(ray, scene)
    return shade(ray, hitpoint, material, scene)

def shade(ray, hitpoint, material, scene):
    radiance = 0
    for light in scene.lights:
        rayToLight, distanceToLight = toLight(hitpoint, light)
        if shadowTrace(rayToLight, distanceToLight, scene):
            # add reflected radiance e.g. phong or diffuse
            radiance += reflectedRadiance(material, rayToLight, distanceToLight)
    if material.type == 'mirror':
        radiance += trace(reflect(ray, hitpoint), scene)
    if material.type == 'transparent':
        radiance += trace(refract(ray, hitpoint), scene)
    return radiance

def shadowTrace(ray, dist, scene):
    t, primitive = findIntersection(ray, scene)
    return dist < t
```

Ray Tracing

- Incorporates in a single framework:
  - Hidden surface removal
    - Front to back traversal
    - Early termination once first hit point is found
  - Shadow computation
    - Shadow rays are traced between a point on a surface and a light sources
  - Exact simulation of some light paths
    - Reflection (reflected rays at a mirror surface)
    - Refraction (refracted rays at a transparent surface, Snell's law)

- Limitations
  - Easily gets inefficient for full global illumination computations
    - Many reflections (exponential increase in number of rays)
    - Indirect illumination requires many rays to sample all incoming directions

Ray Tracing: Approximations

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Usually RGB color model instead of full spectrum
- Finite number of point lights instead of full indirect light
- Approximate material reflectance properties
  - Ambient: constant, non-directional background light
  - Diffuse: light reflected uniformly in all directions,
  - Specular: perfect reflection, refraction
- All are based on purely empirical foundation

Questions

- Write down and explain the principle steps of a recursive ray tracer.

- How do you compute the ray intersections with:
  - Plane
  - Box
  - Triangle
  - Sphere
  - Cylinder

- How do you evaluate the shading for a diffuse surface?

Wrap-Up

- Background
  - Forward light transport vs. backward search in RT
- Ray tracer
  - Ray generation, ray-object intersection, shading
- Ray-geometry intersection calculation
  - Sphere, plane, triangle, box
- Recursive ray tracing algorithm
  - Primary, secondary, shadow rays
- Next lecture
  - Acceleration techniques