EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

FACULTY OF
SCIENCE
Computer Graphics

Computer Graphics (Graphische Datenverarbeitung)

- Rasterization & Ray Tracing -

WS 2022/23

Hendrik Lensch

Overview

- Last lecture
  - Camera transformations

- Today
  - Advanced acceleration structures
    - Theoretical Background
    - Hierarchical Grids, kd-Trees, Octrees
    - Bounding Volume Hierarchies
  - Ray bundles

- Next lecture
  - Dynamically changing scenes

Overview

- Last Lecture
  - Camera Transfomrations

- Today
  - Rasterization
    - Z-Buffer
  - Ray Tracing I
    - Background
    - Basic ray tracing
    - Recursive ray tracing algorithm

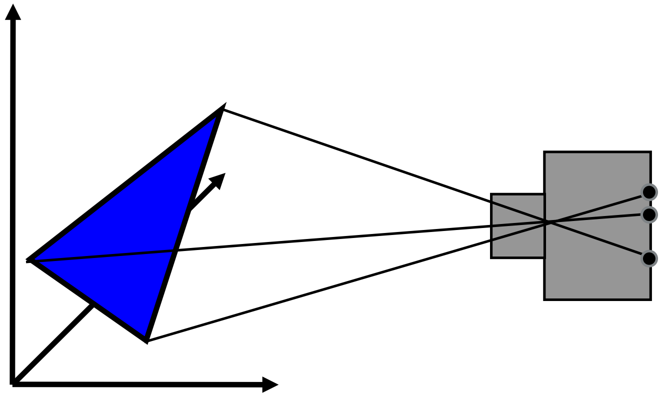
- Next Lecture
  - Ray Tracing II: Spatial indices

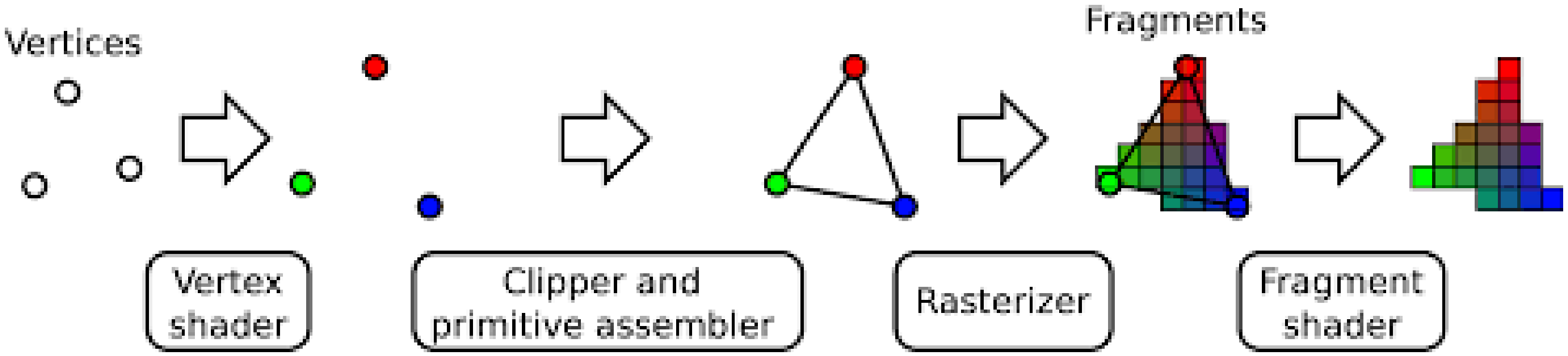
Rasterization

Image Formation: Rasterization

- Primitive operation of all interactive graphics
- Scan convert a single triangle at a time

1. Project the vertices into the image plane
2. Scanline by scanline enumerate and fill the pixels covered by the triangle





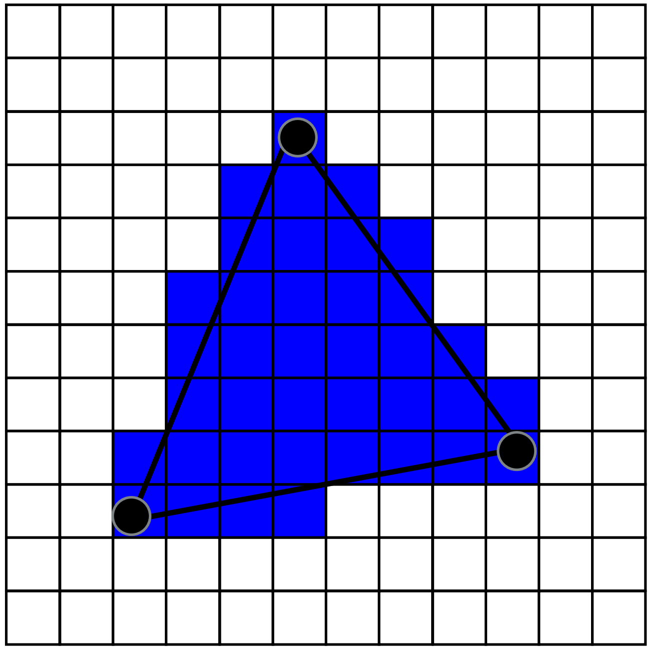
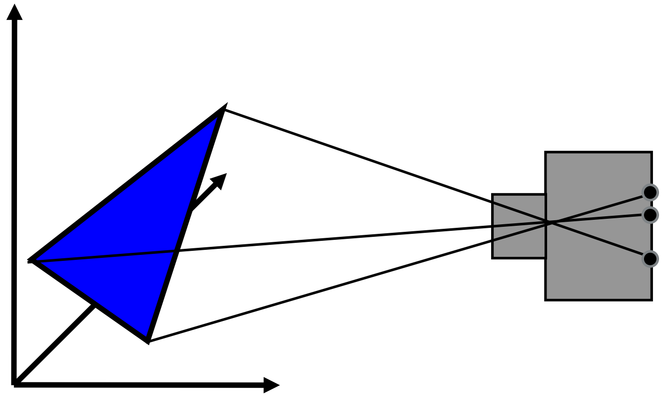
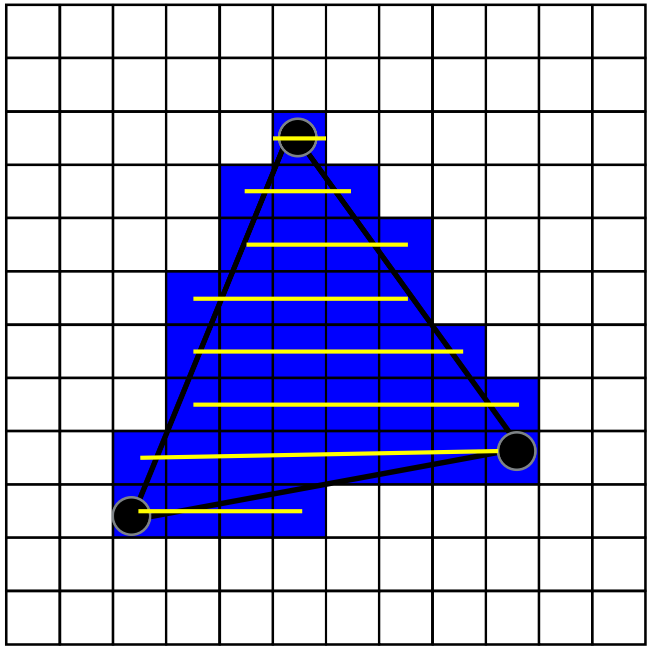


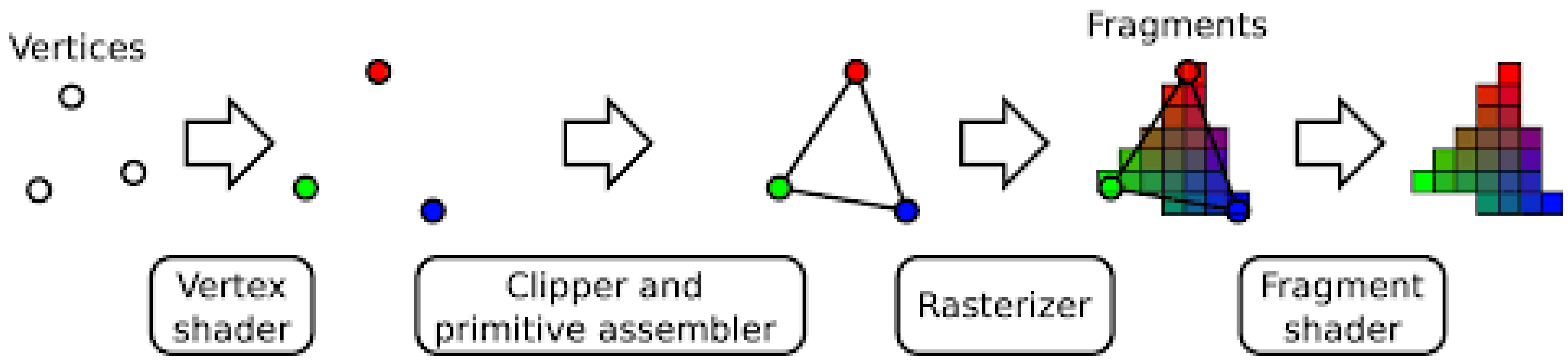
Image Formation: Rasterization



- Primitive operation of all interactive graphics
- Scan convert a single triangle at a time

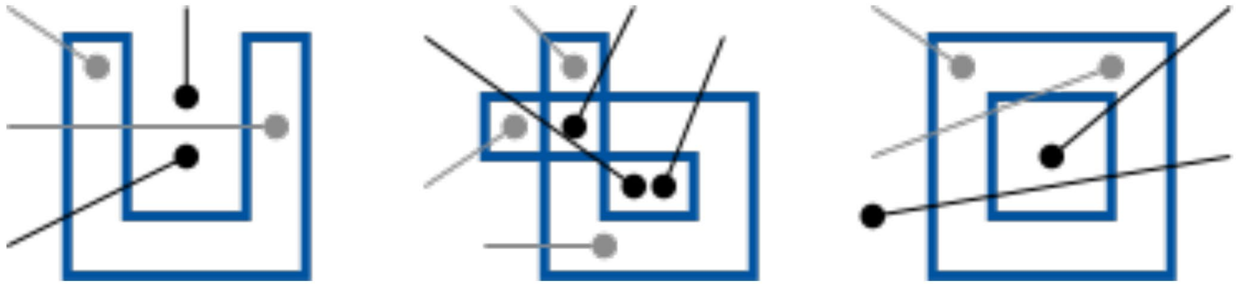
1. Project the vertices into the image plane
2. Scanline by scanline enumerate and fill the pixels covered by the triangle



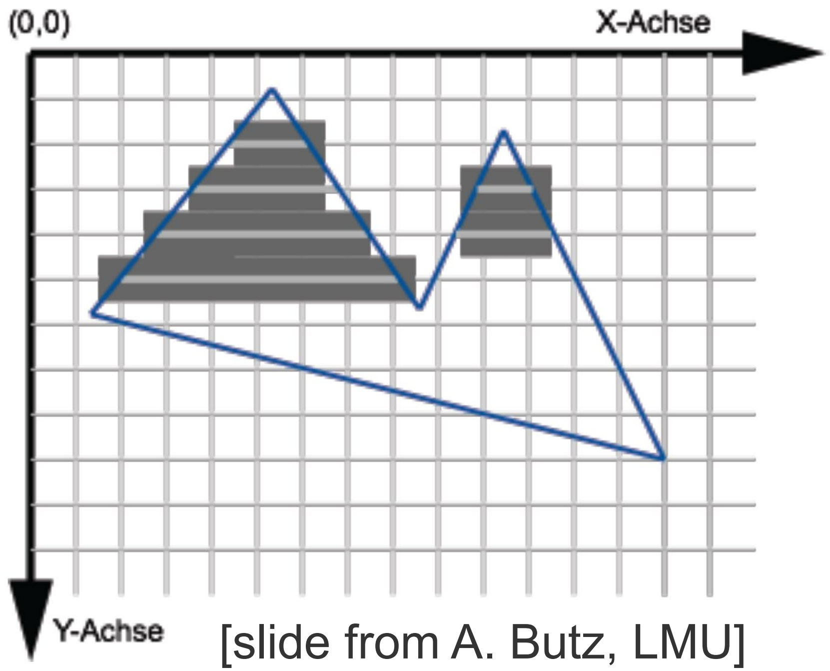


Polygon – Inside/Outside Test

- Define parity of a point in 2D:
  - Send a ray from this point to infinity
  - Direction irrelevant (!)
  - Count number of lines it crosses
    - If 0 or even: even parity (outside)
    - If odd: odd parity (inside)



- Determine polygon area $(x_{min}, x_{max}, y_{min}, y_{max})$
- Scan the polygon area line by line
- Within each line, scan pixels from left to right
  - Start with parity = 0 (even)
  - Switch parity each time we cross a line
  - Set all pixels with odd parity



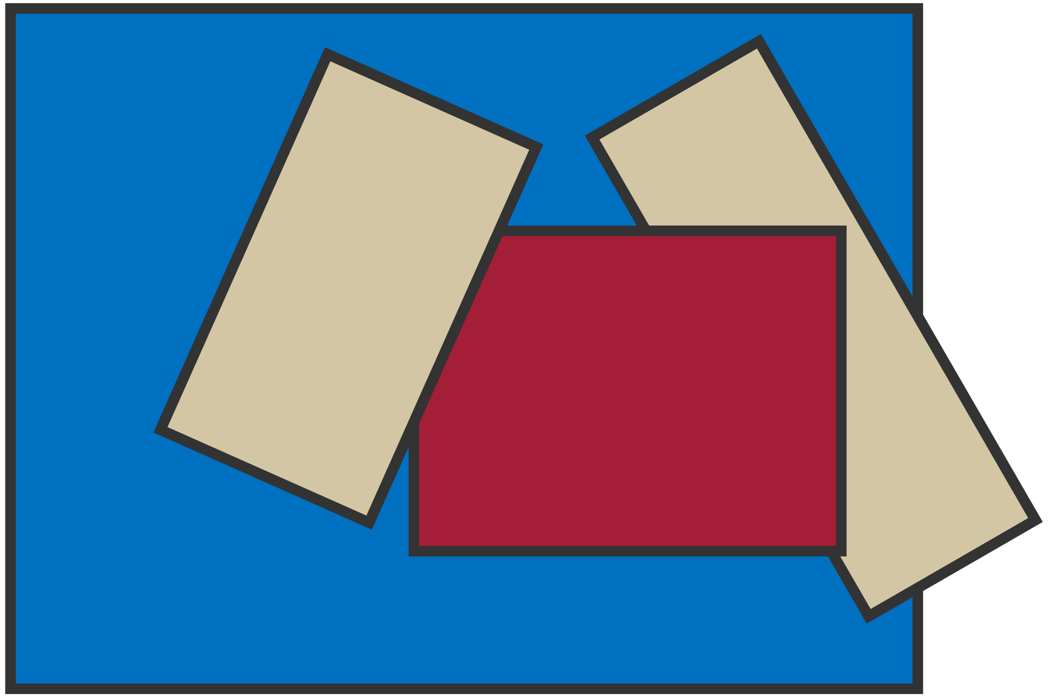
[slide from A. Butz, LMU]

Occlusions and how to deal with them

- Need to determine which objects occlude which others objects
  - Want to draw only the frontmost (parts of) objects

- More general: draw the frontmost polygons…
  - …or maybe parts of polygons?

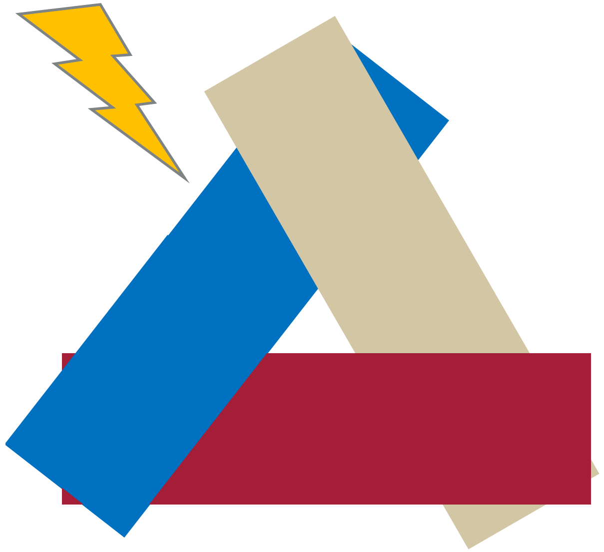
- Occlusion is an important depth cue for humans
  - Need to get this really correct!



Occlusion – Simple: Depth-sort + ordered rendering

- Regularly used in 2D vector graphics
  - Sort polygons according to their z position in view coordinates
- Draw all polygons from back to front
  - Back polygons will be overdrawn
  - Front polygons will remain visible

→ "Painter’s algorithm”



- Problem 1: Self-occlusion
  - Not a problem with triangles

- Problem 2: Circular occlusion
  - Think of a pin wheel!

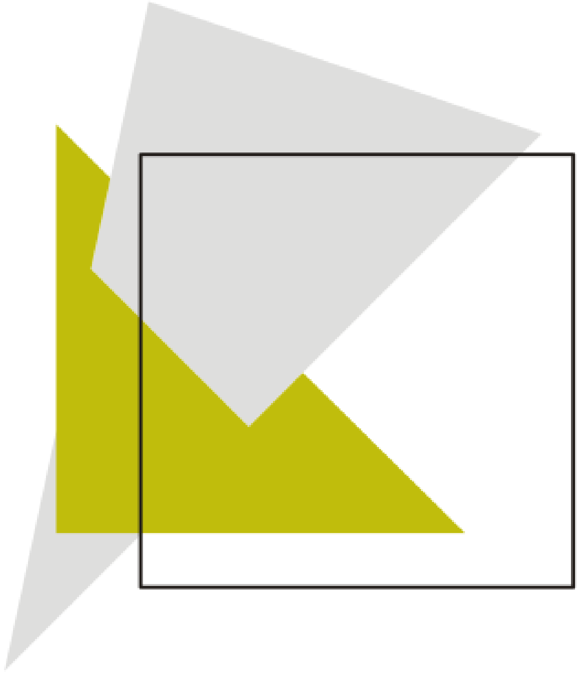
COPPENRATH
DIE SPIEGELBURG

Z-Buffer Algorithm – Better Occlusion Handling

- ***Idea:*** Compute depth not per polygon, but per pixel!
  - Approach: for each pixel of the rendered image (frame buffer)
  - keep also a depth value (Z-buffer)

- Initialize the Z-buffer with z_{far}, which is the far clipping plane and
  - hence the furthest distance we need to care about

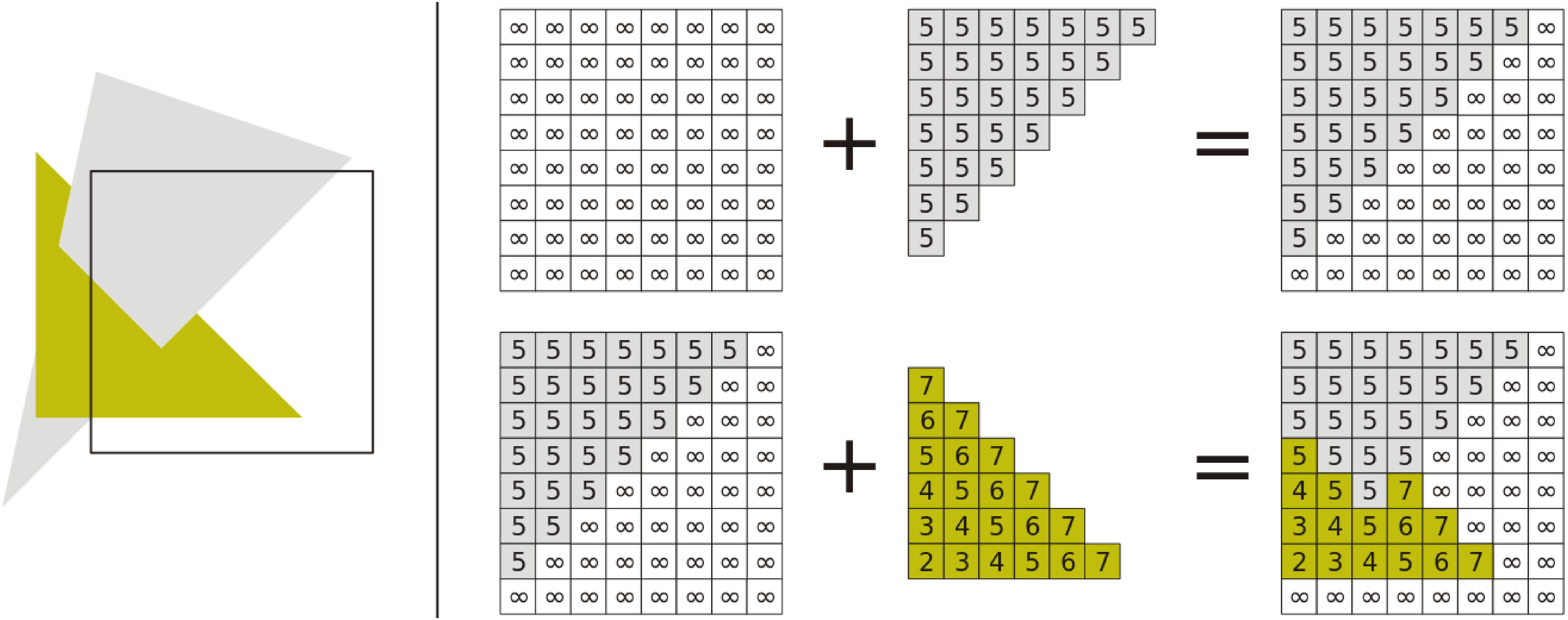
- Loop over all polygons
  - Determine which pixels are filled by the polygon
  - For each pixel
    - Compute the z value (depth) at that position
    - If z > value stored in Z-buffer (remember: negative z!)
      - Draw the pixel in the image
      - Set Z-buffer value to z



[slide from A. Butz, LMU]

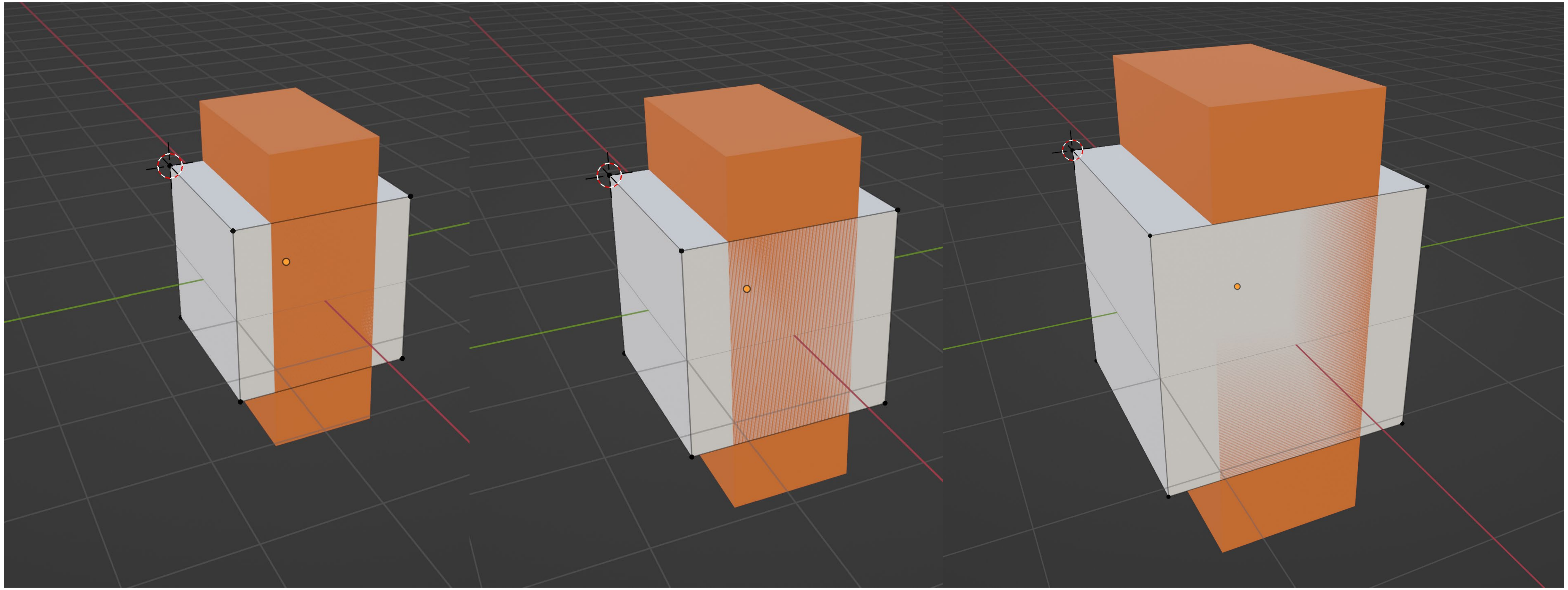
Z-Buffer Example

- In contrast to OpenGL, this example uses positive z-values (and thus tests for z <= Z-buffer-value)!



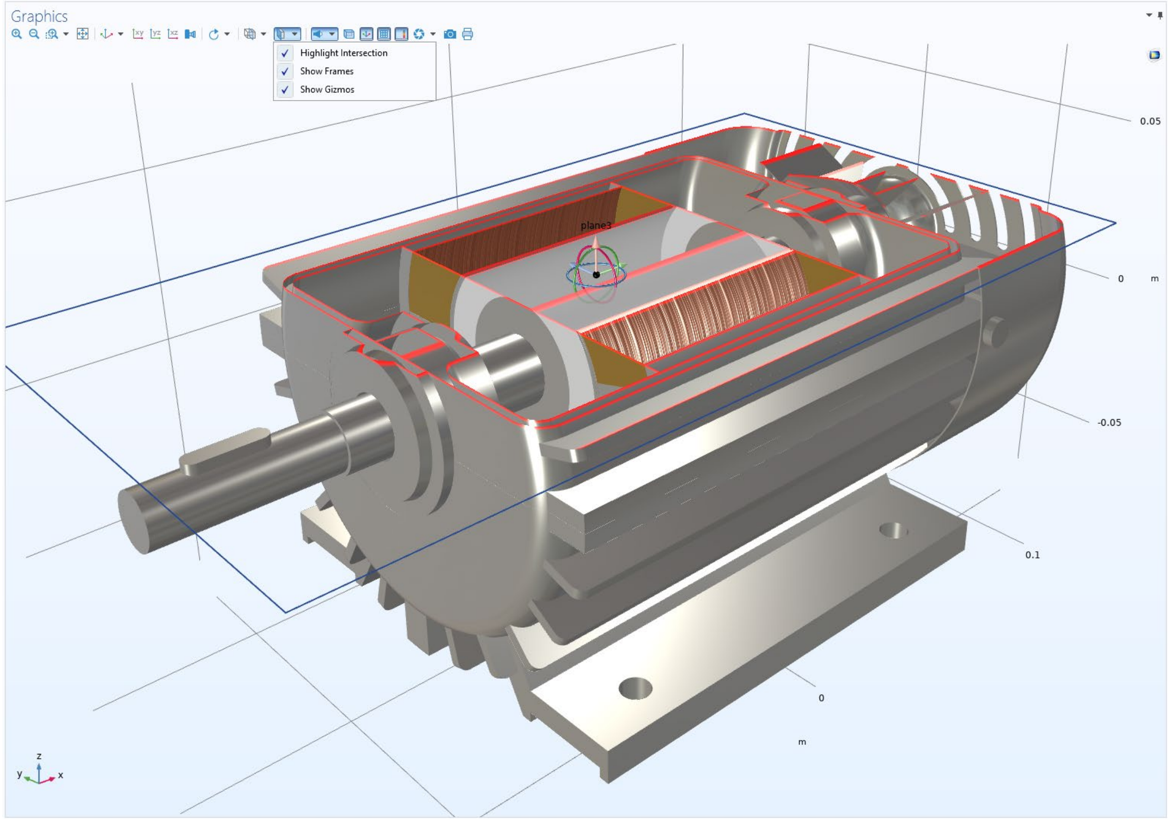
Z-Buffer: Tips and Tricks

- Z-Buffer normally built into graphics hardware
- Limited precision (e.g., 16 bit)
  - Potential problems with large models
  - Set clipping planes wisely!
  - Never have 2 polygons in the exact same place (z-fighting)



Z-Buffer: Tips and Tricks

- Z-Buffer can be initialized partially to something else than z_{far}
  - At pixels initialized to z_{near} no polygons will be drawn
  - Use to cut out holes in objects – clip planes
  - Then re-render the objects you want to see through these holes



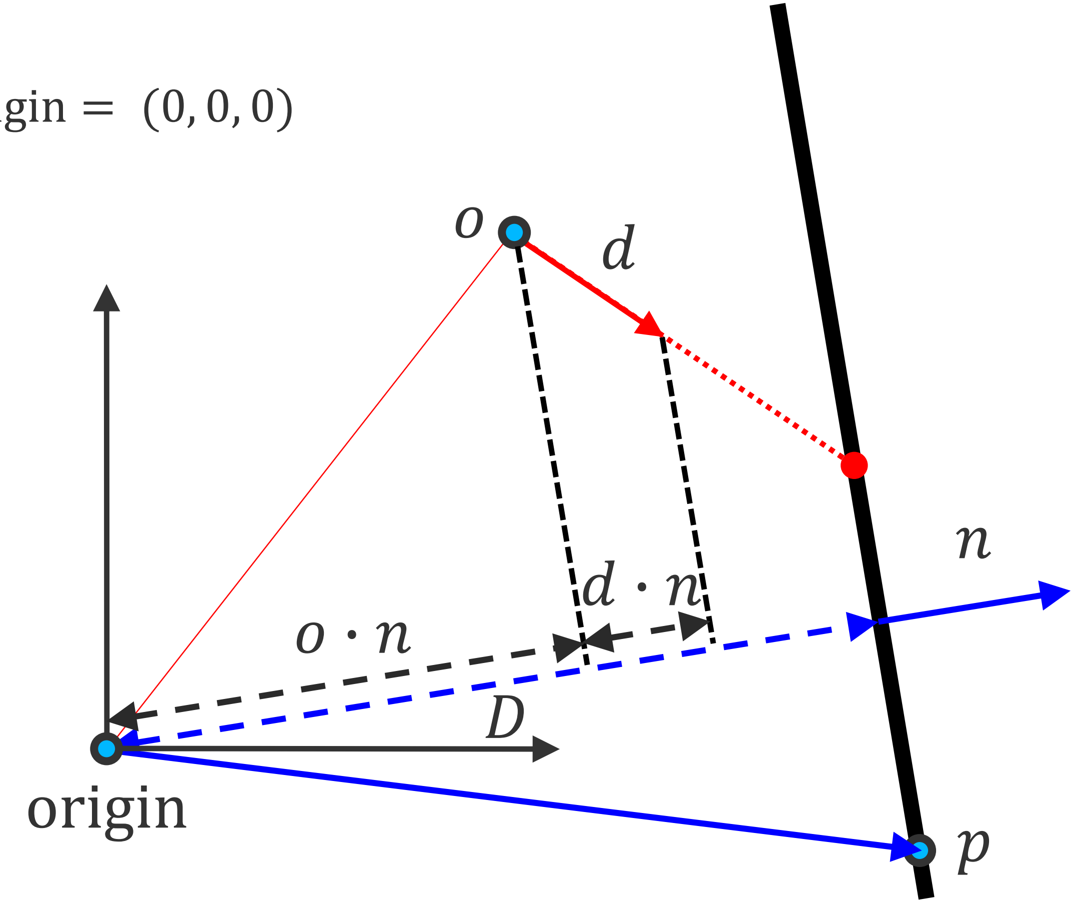
 [https://www.comsol.de/press/gallery/?filter=COMSOL%20Version%205.6]

Ray Tracing (Recap)

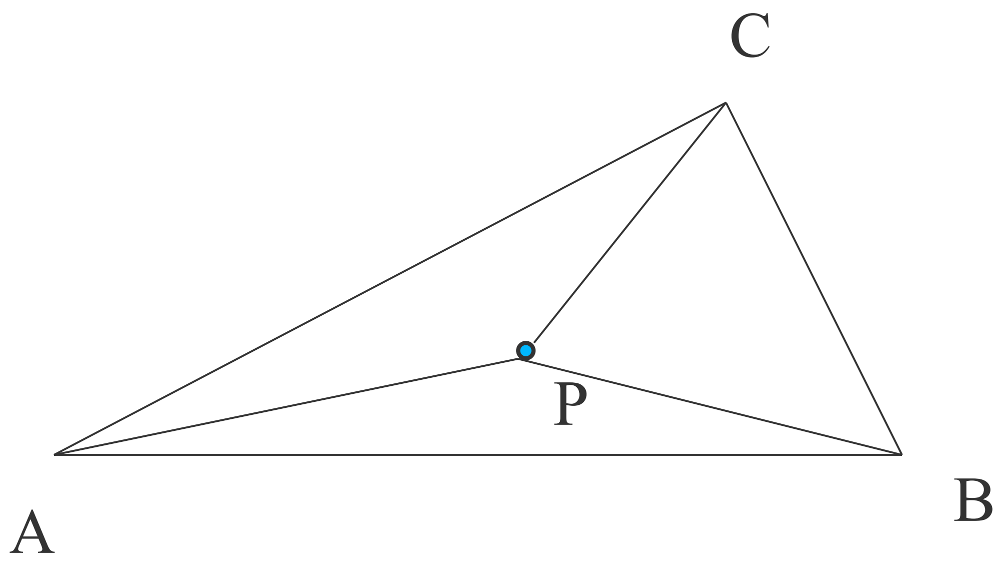
Intersection Ray – Plane

- 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 $\text{origin} = (0,0,0)$

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

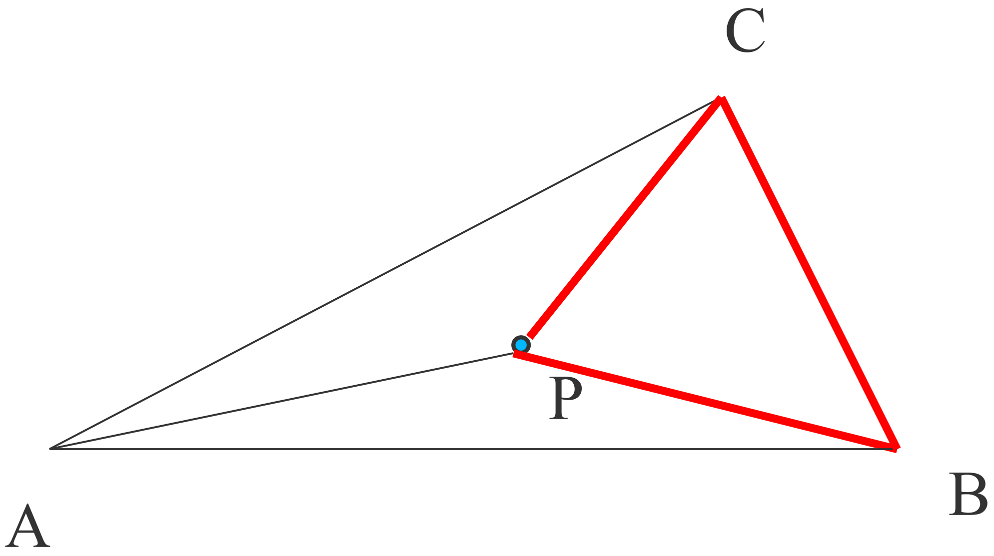


Barycentric Coordinates



$$P = \lambda_1 A + \lambda_2 B + \lambda_3 C$$

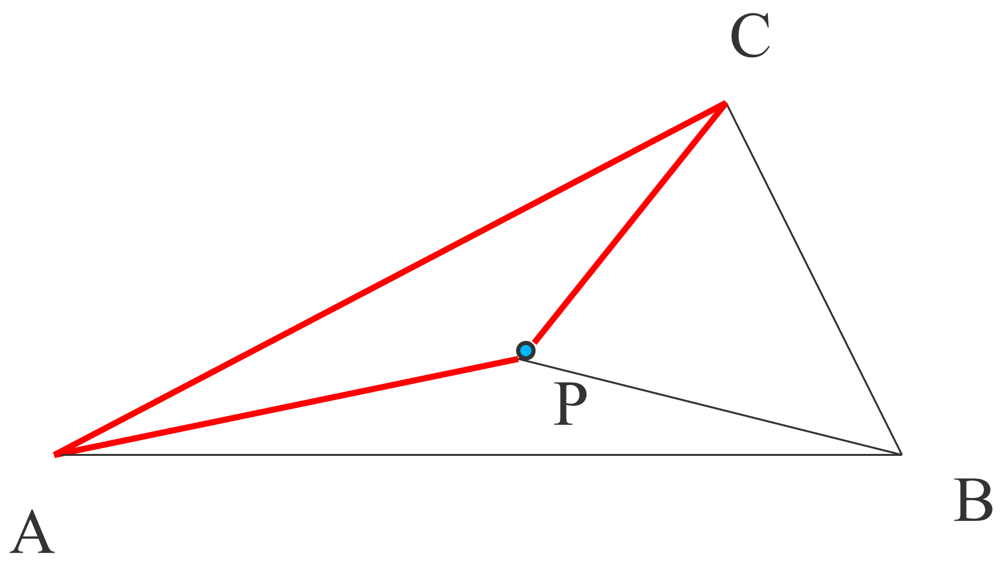
Barycentric Coordinates



$$P = \lambda_1 A + \lambda_2 B + \lambda_3 C$$

$$\lambda_1 = \frac{S_A}{S_\Delta}$$

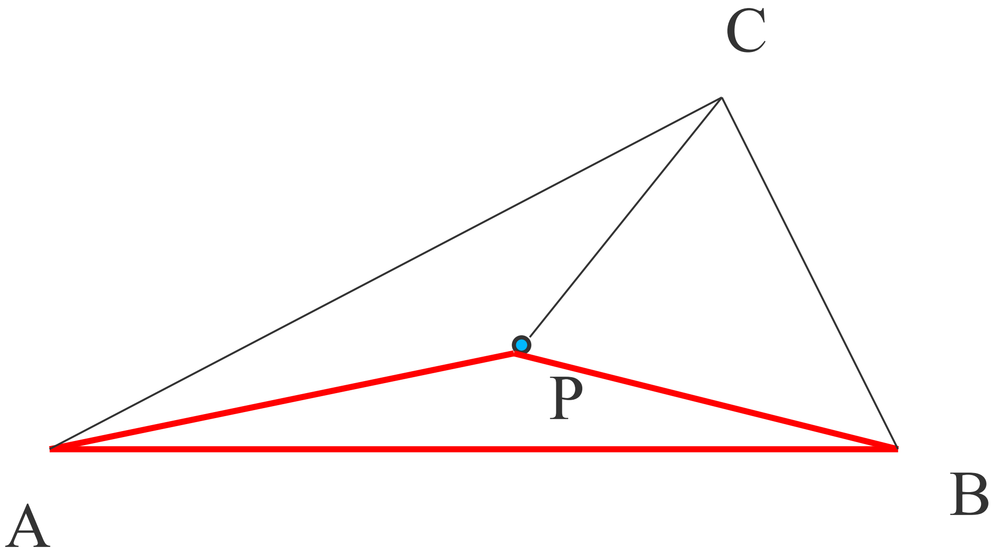
Barycentric Coordinates



$$P = \lambda_1 A + \lambda_2 B + \lambda_3 C$$

$$\lambda_2 = \frac{S_B}{S_\Delta}$$

Barycentric Coordinates

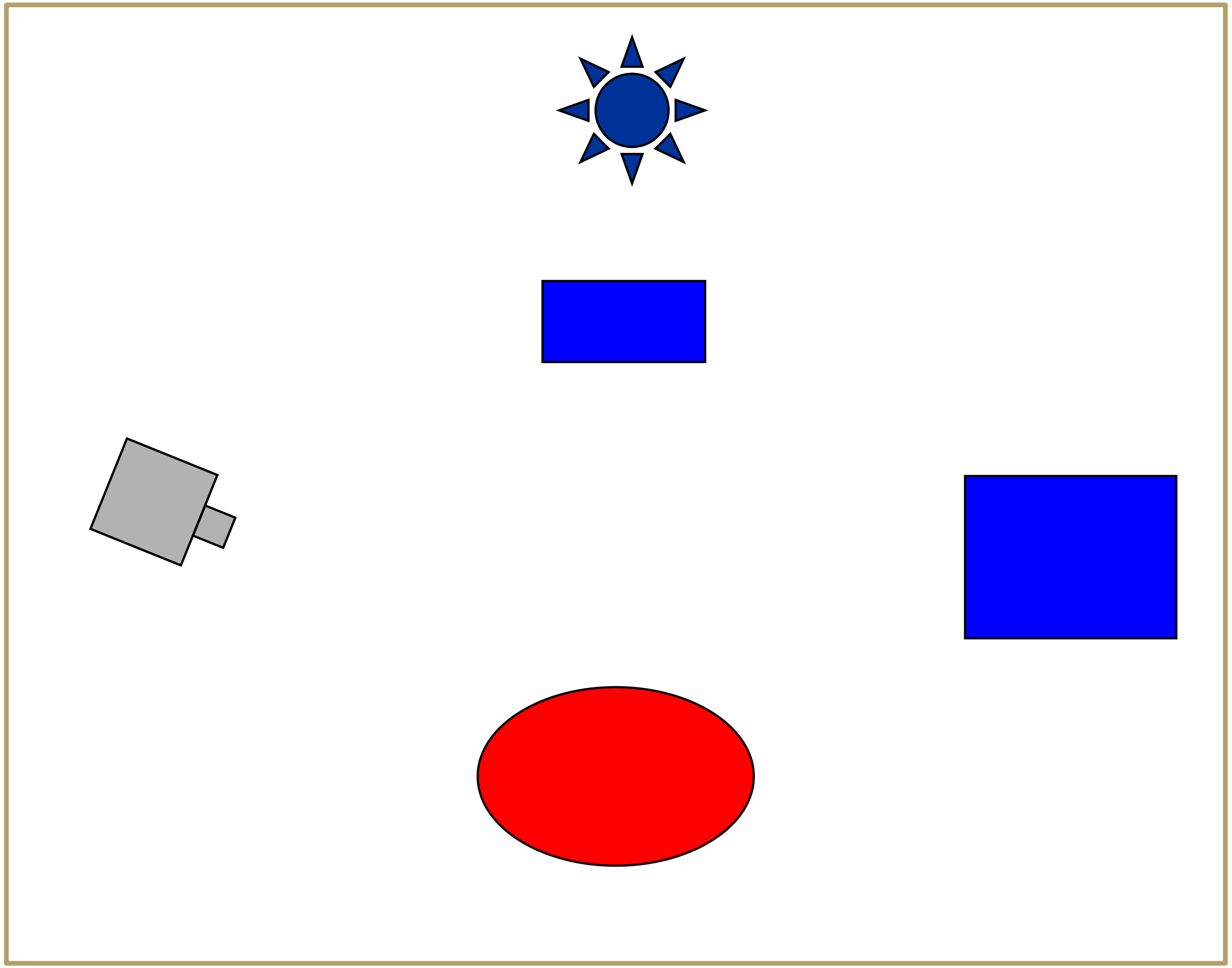


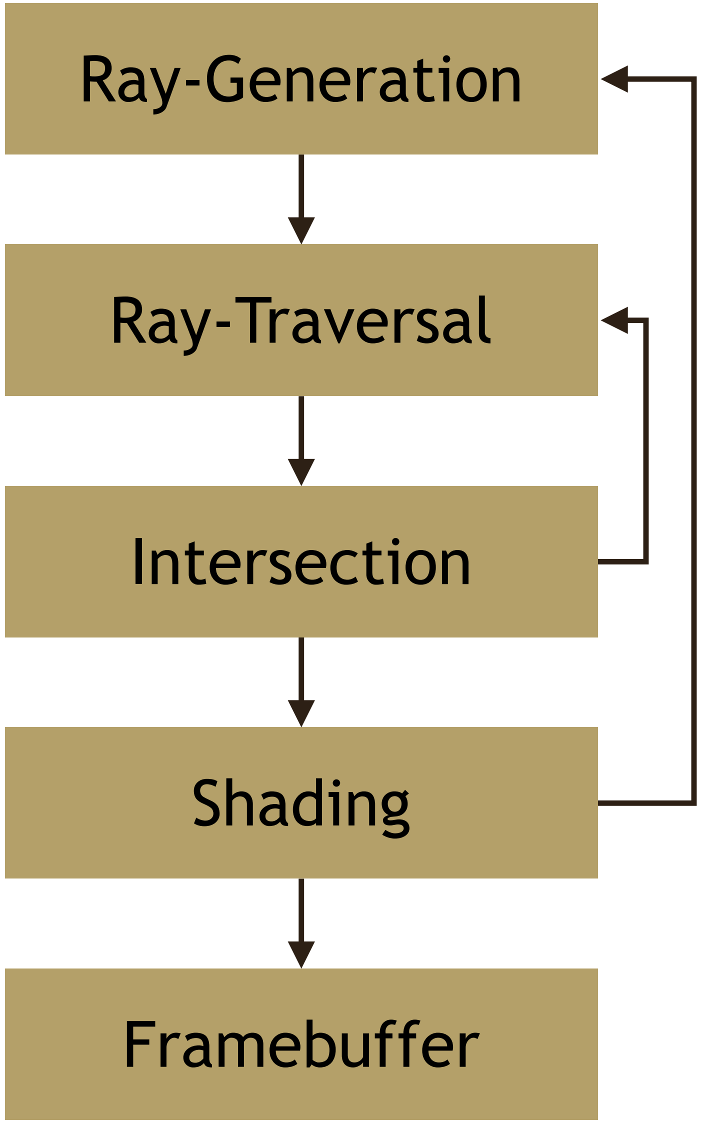
$$P = \lambda_1 A + \lambda_2 B + \lambda_3 C$$

$$\lambda_3 = \frac{S_C}{S_\Delta}$$

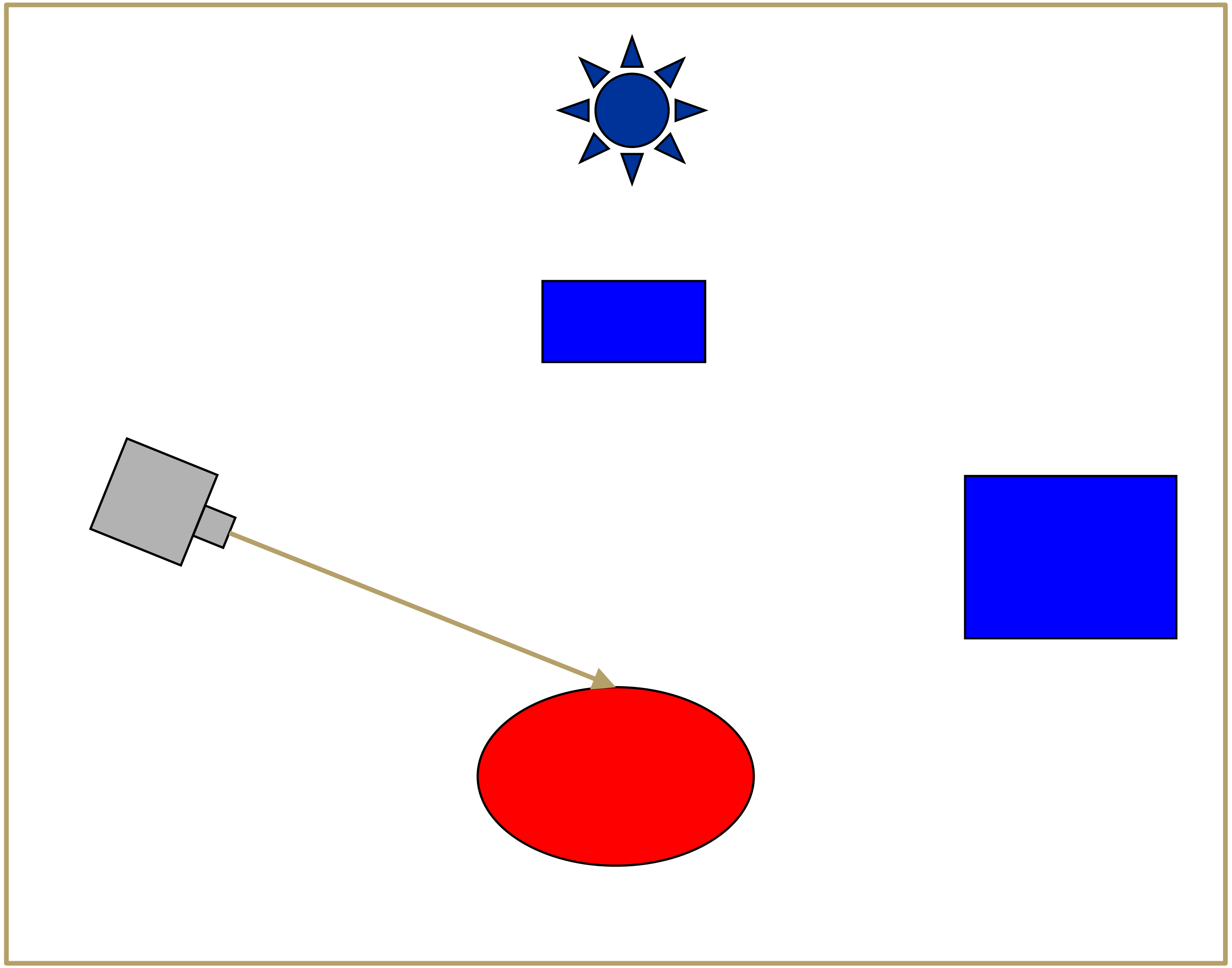
Recursive Ray Tracing

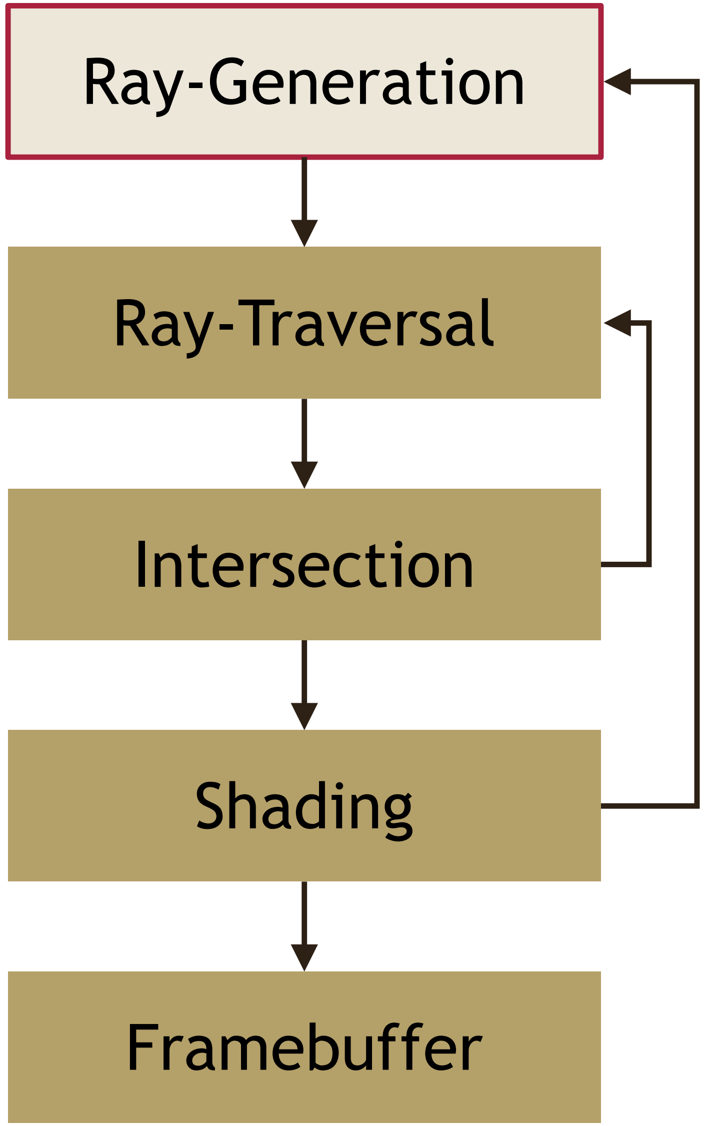
Ray Tracing Pipeline



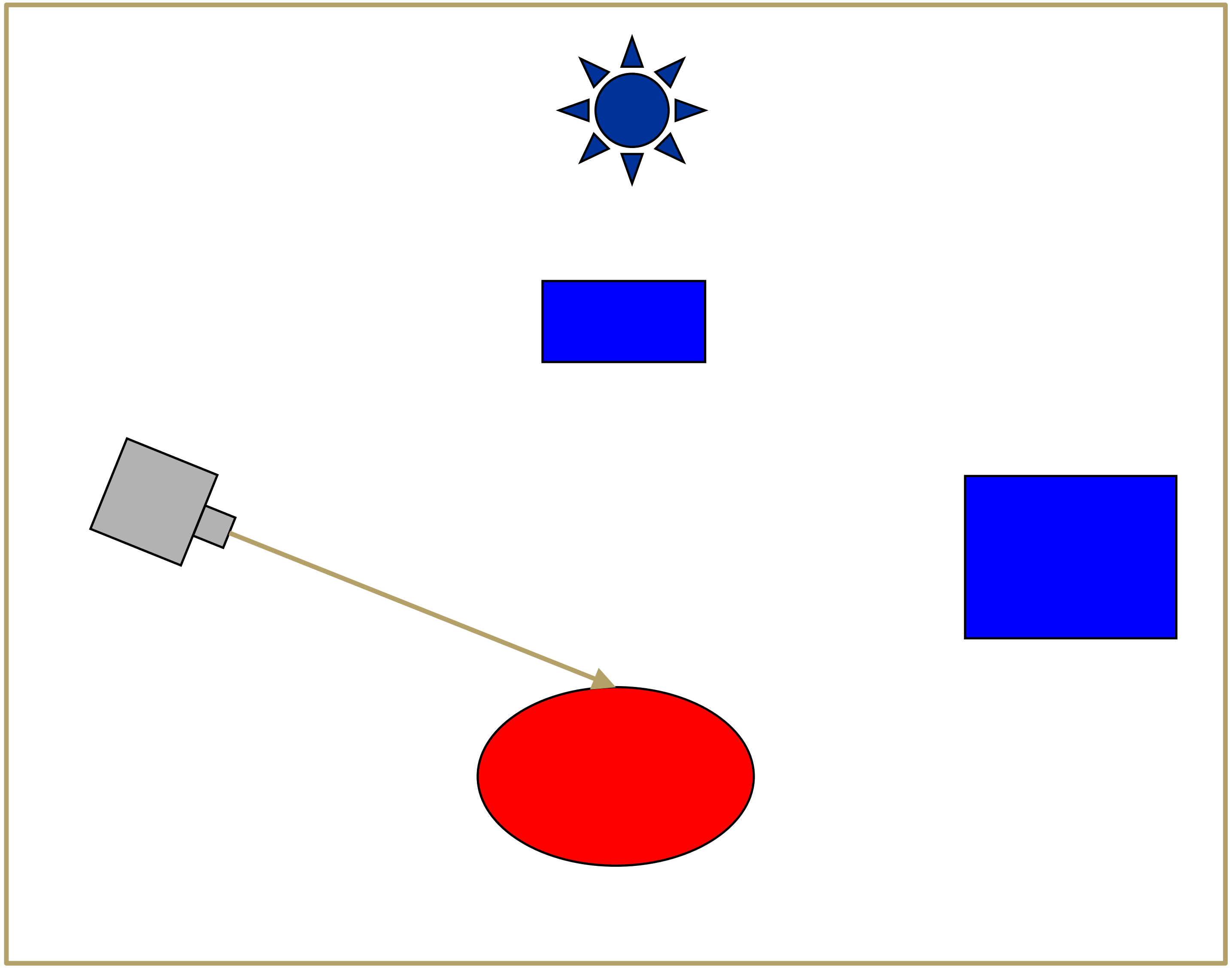


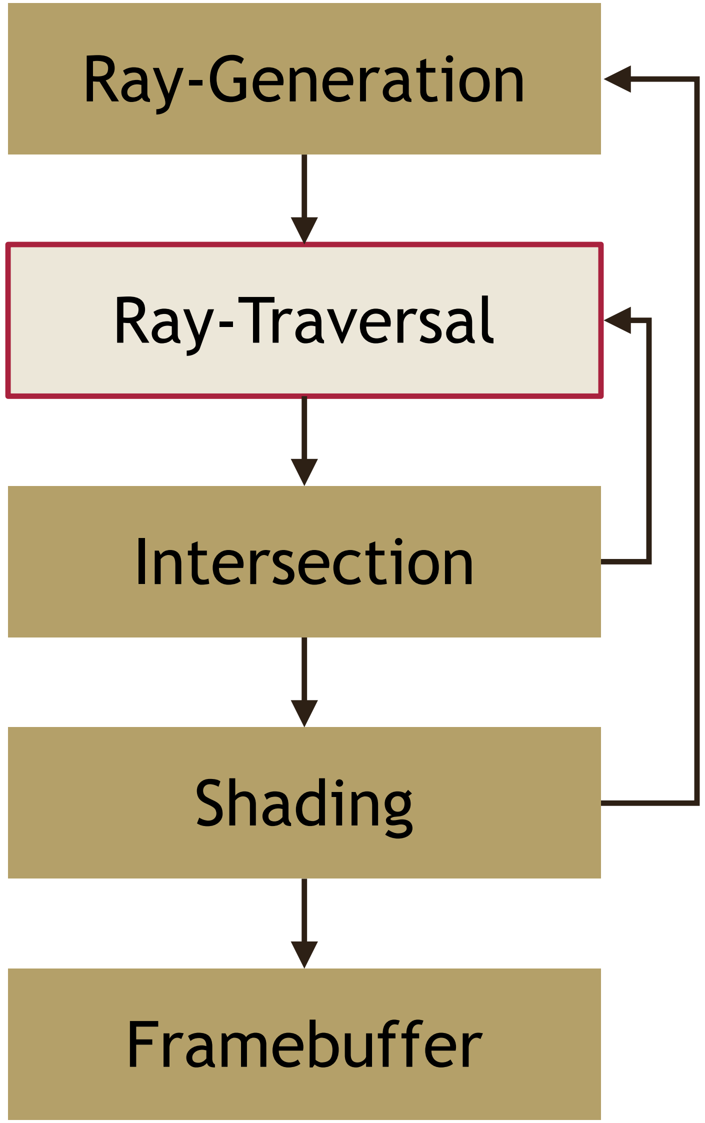
Ray Tracing Pipeline



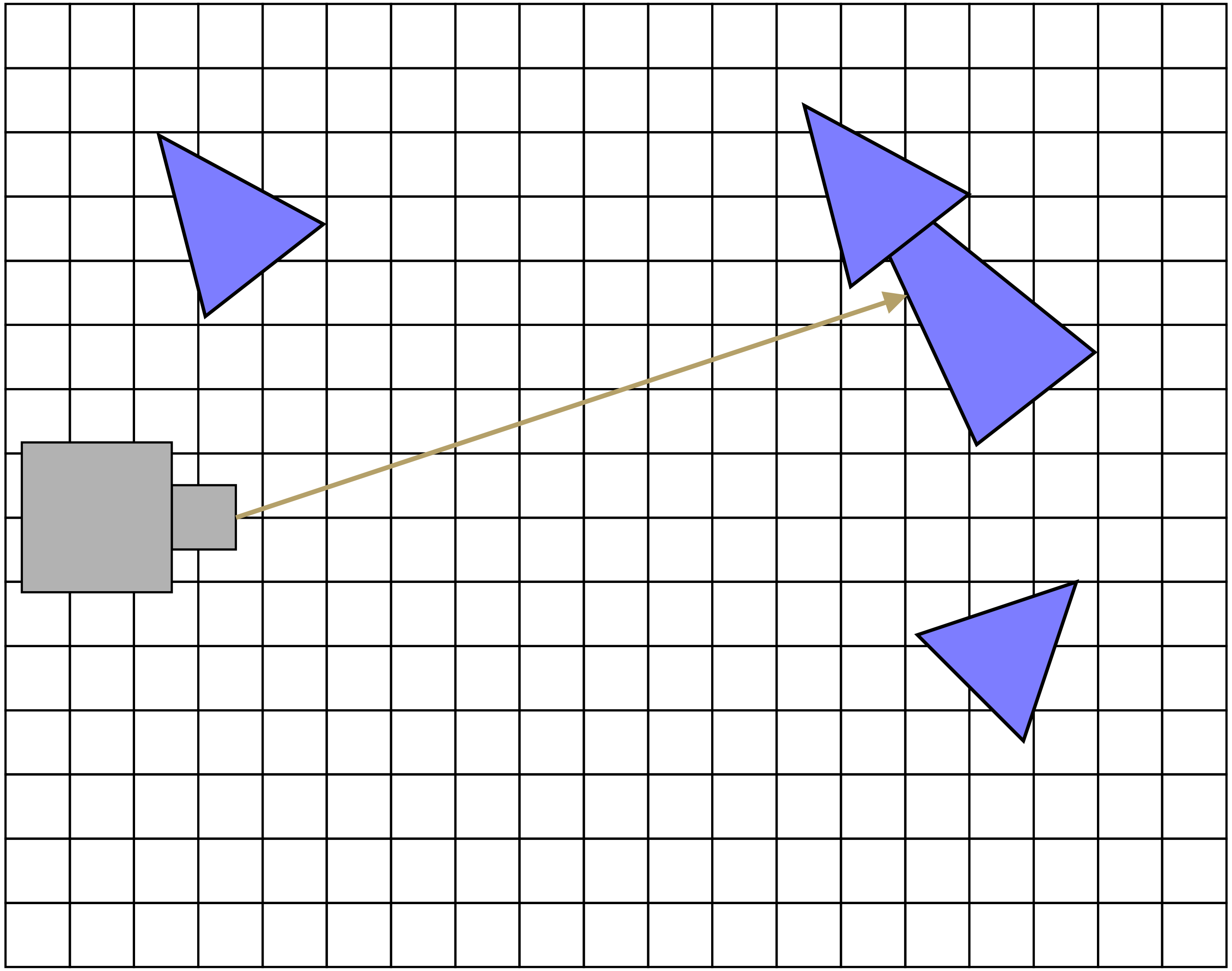


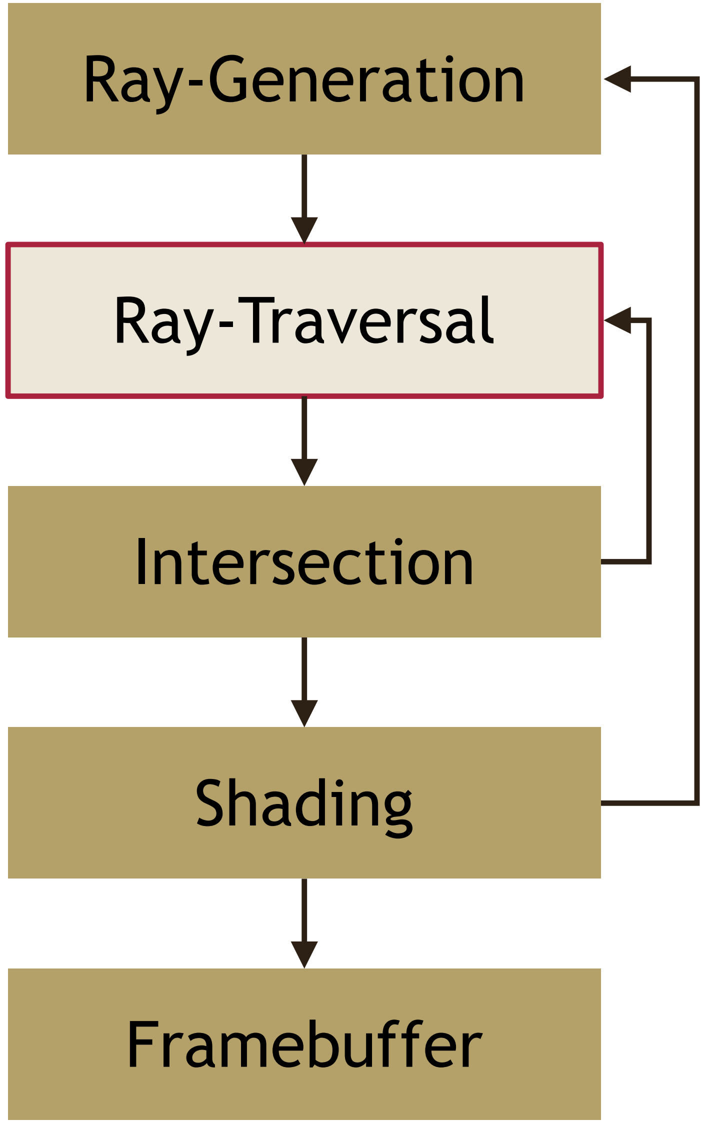
Ray Tracing Pipeline



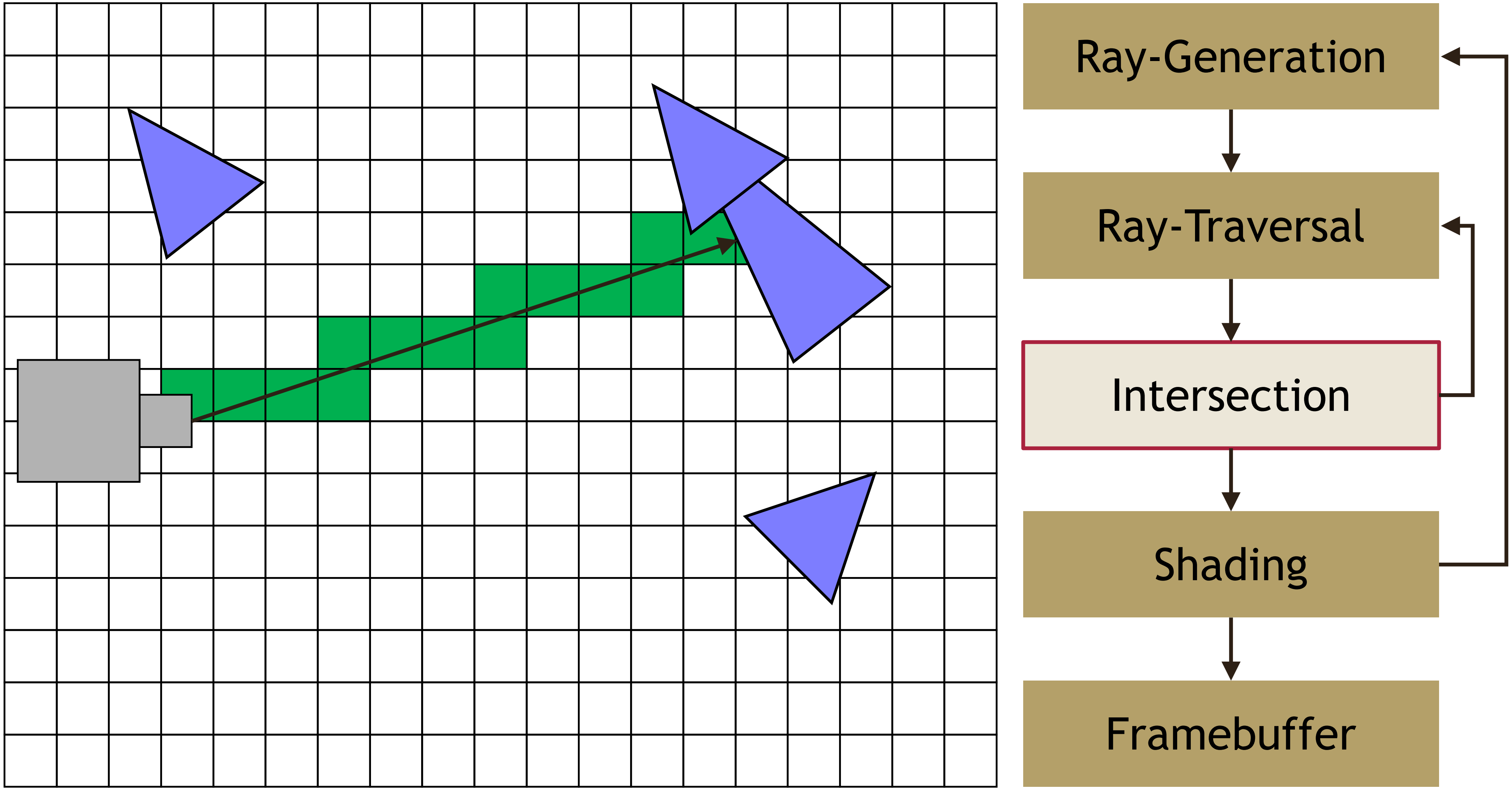


Ray Tracing Pipeline

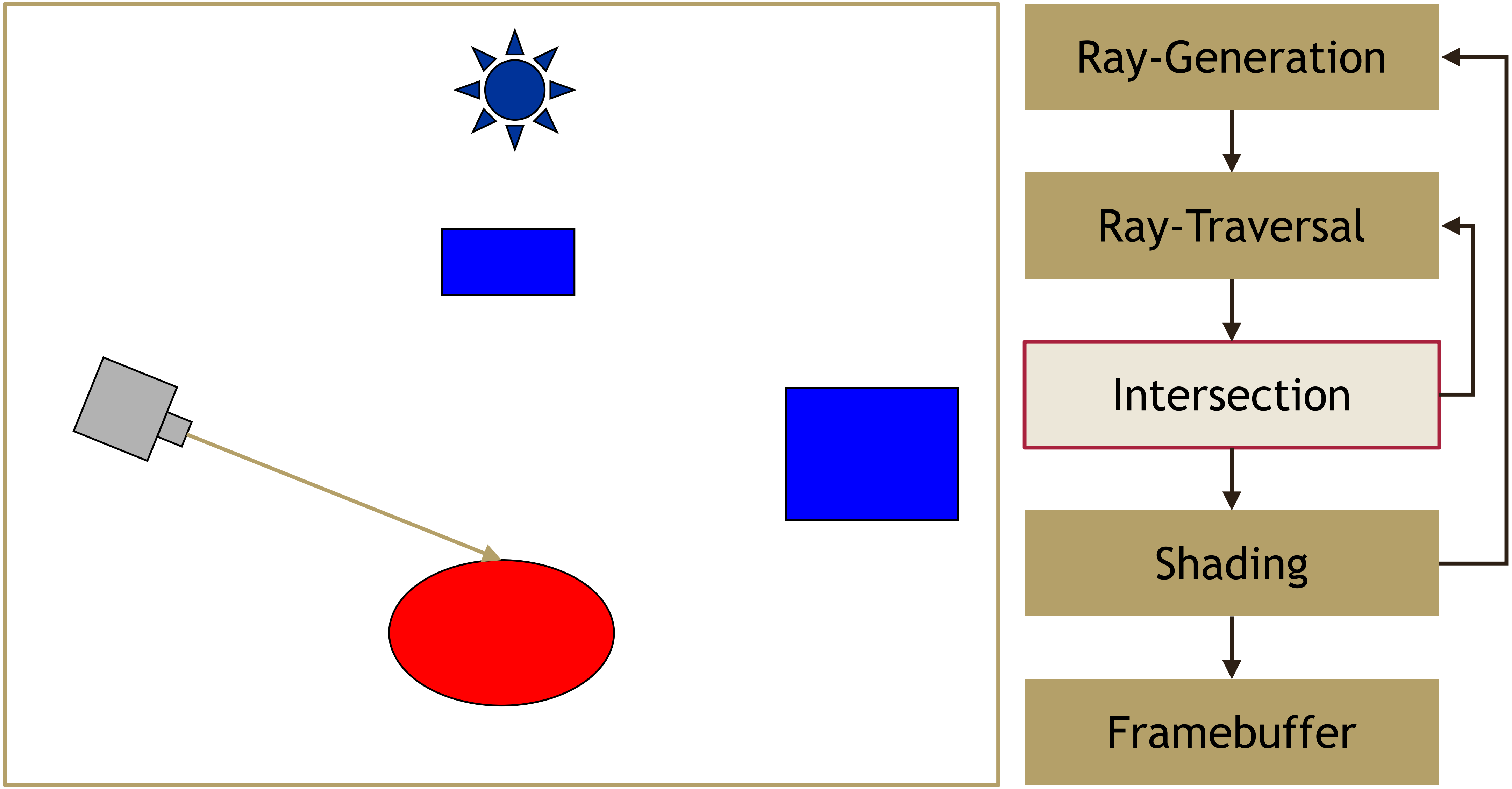




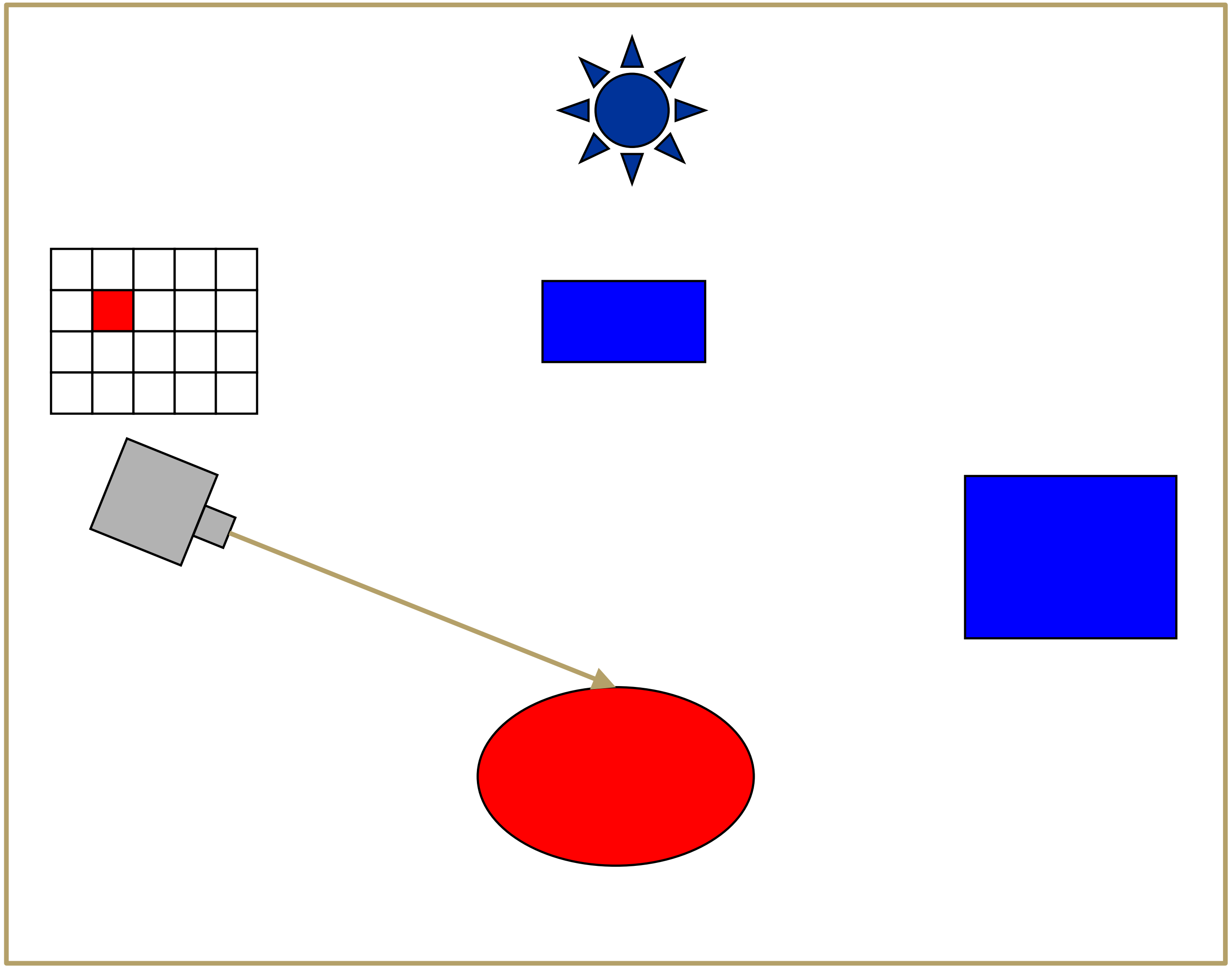
Ray Tracing Pipeline



Ray Tracing Pipeline



Ray Tracing Pipeline



Ray-Generation

Ray-Traversal

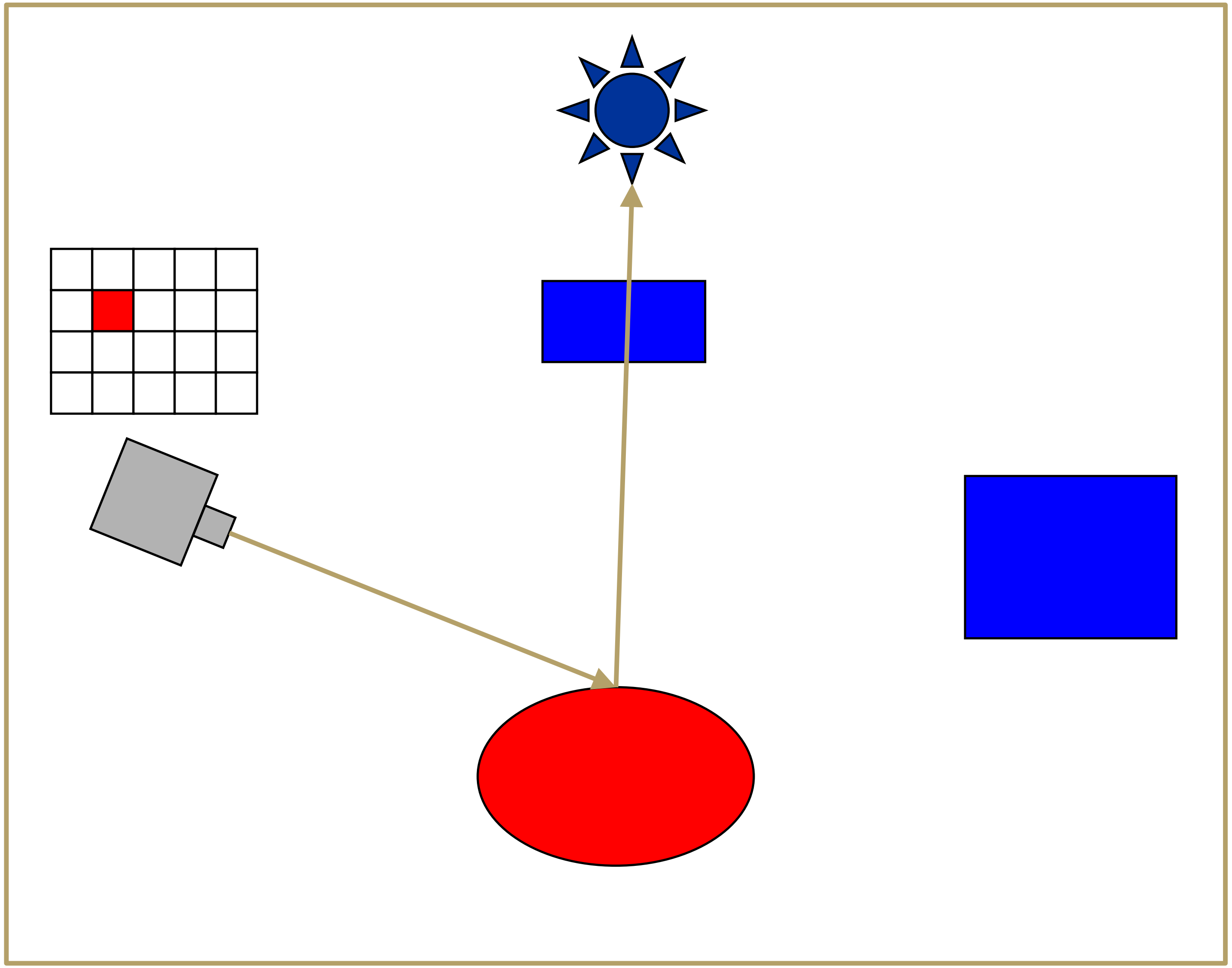
Intersection

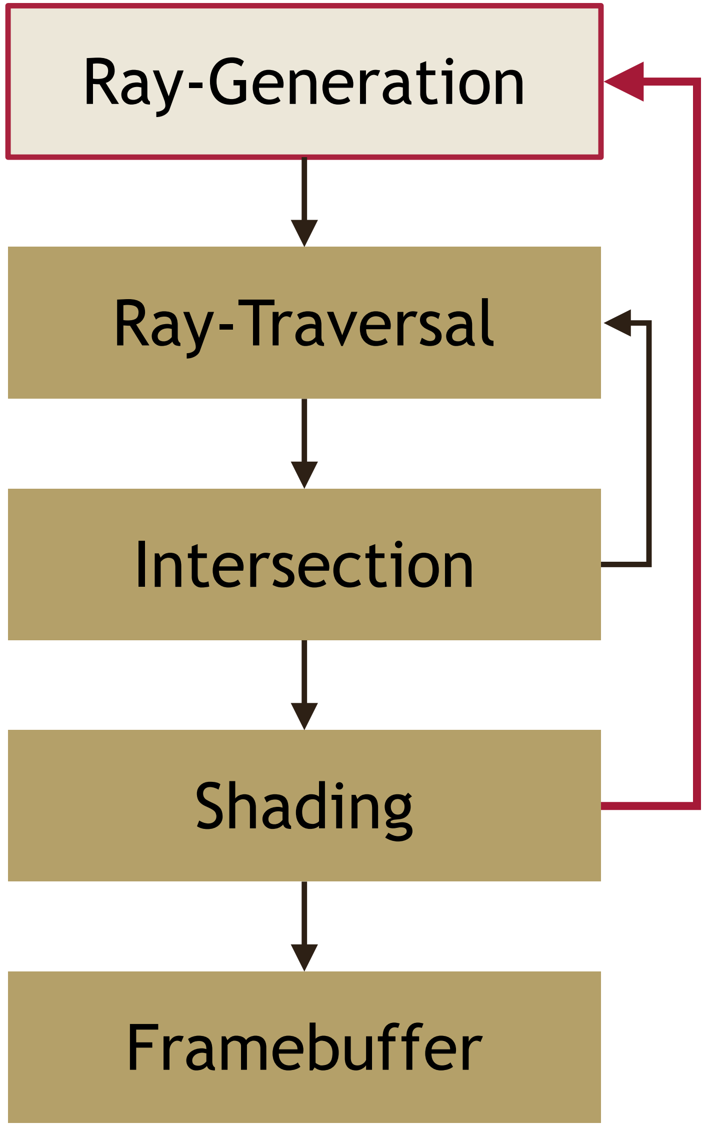
Shading

Framebuffer

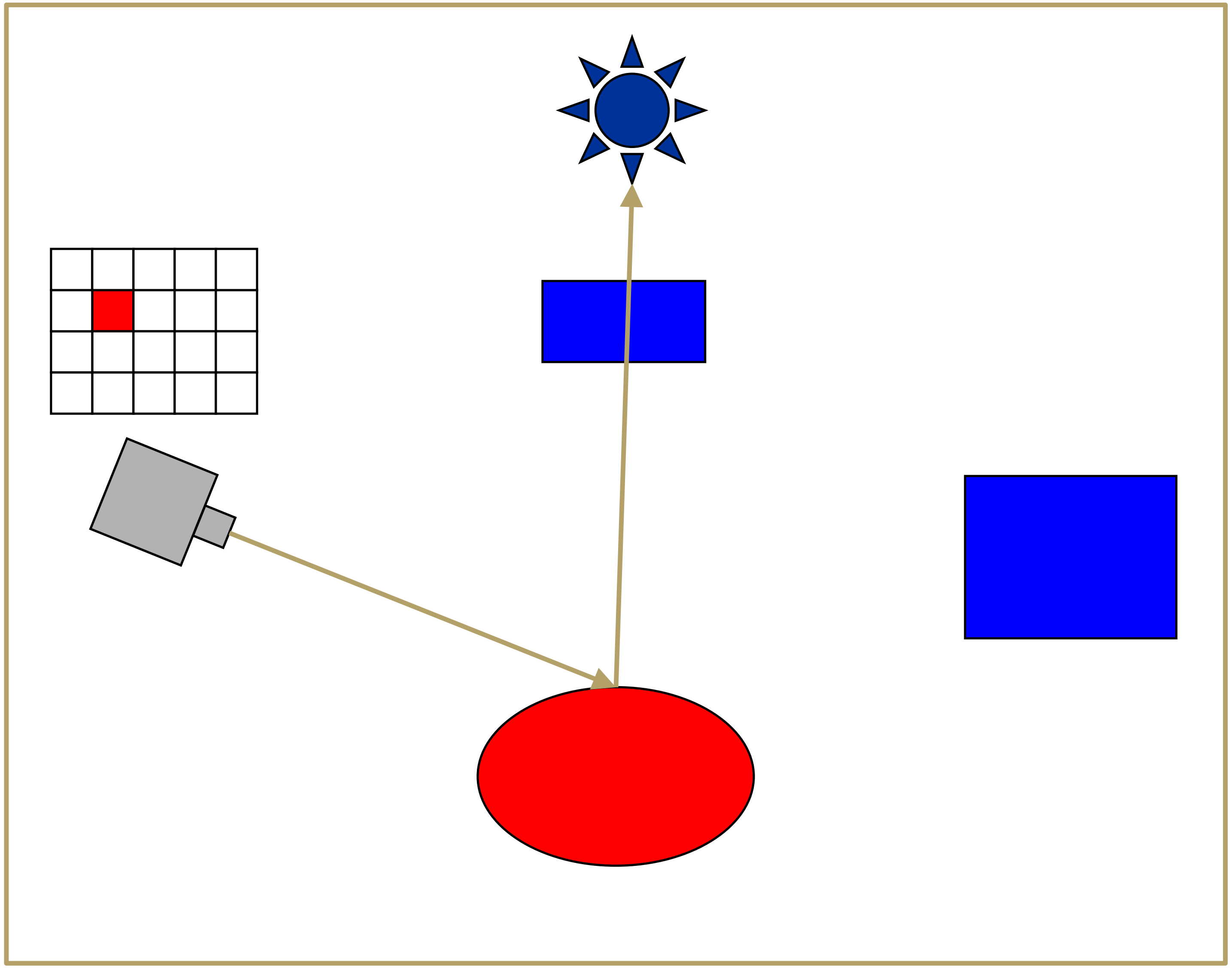
Ray Tracing Pipeline

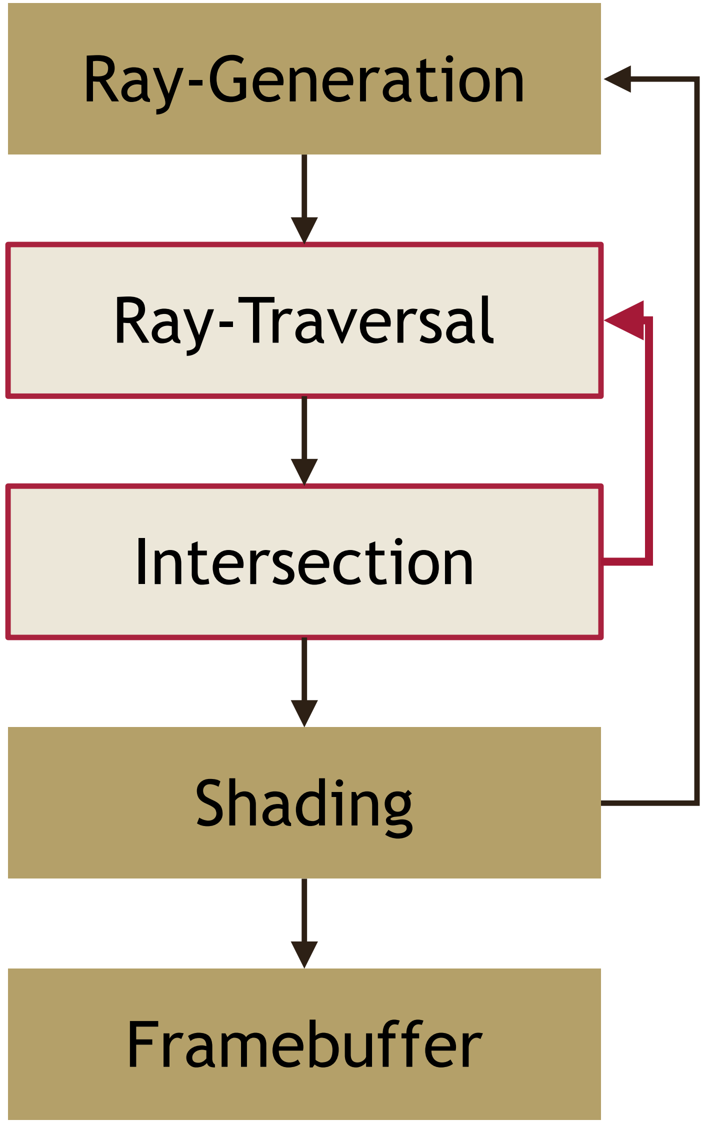
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN



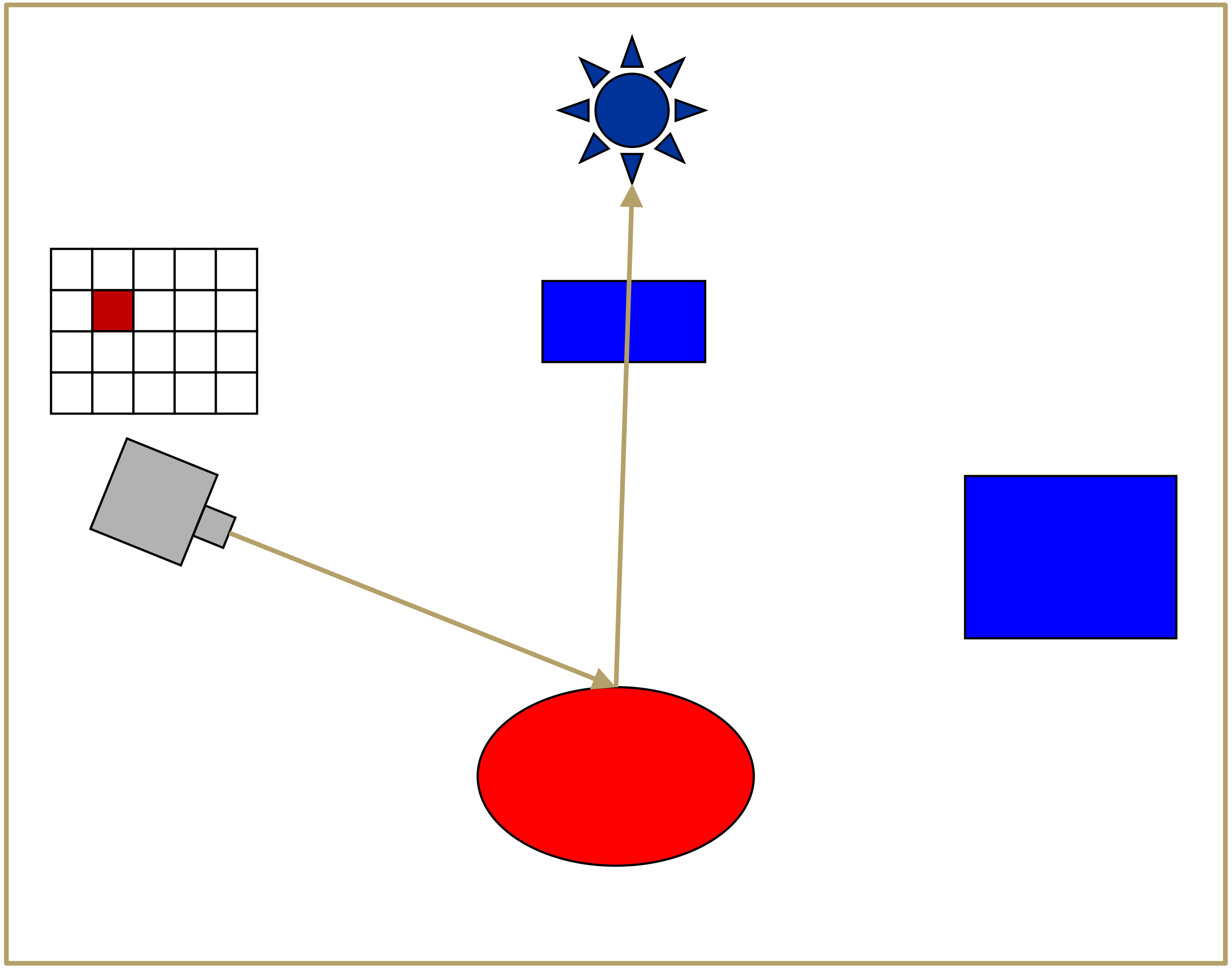


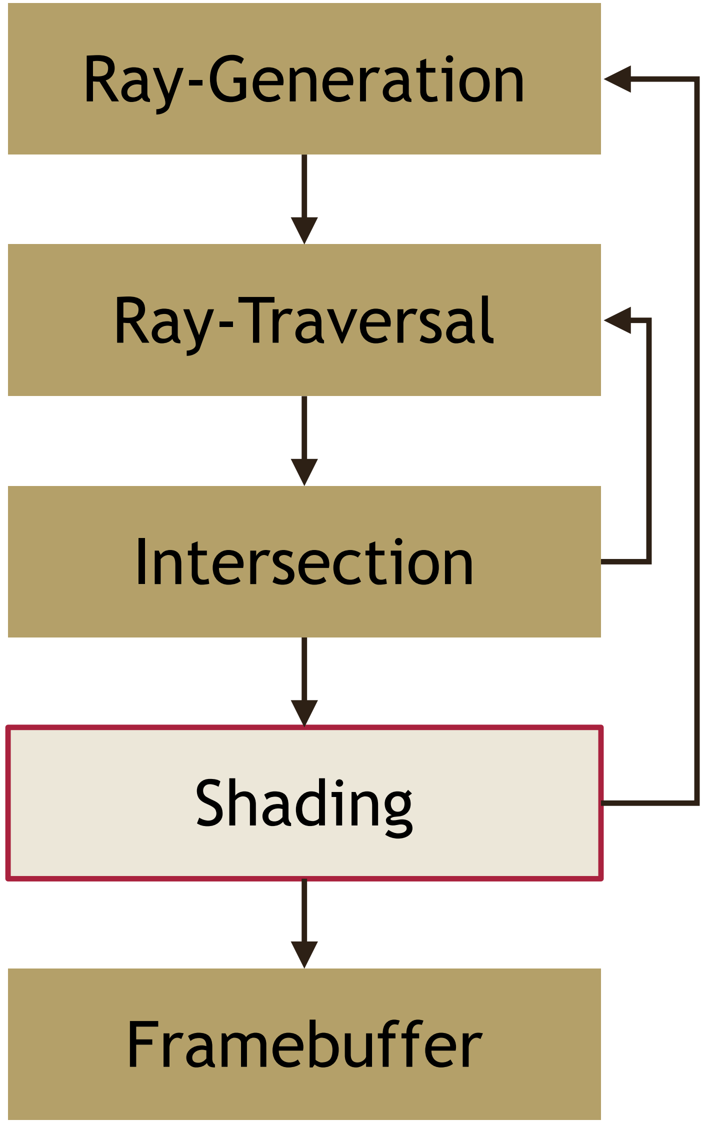
Ray Tracing Pipeline



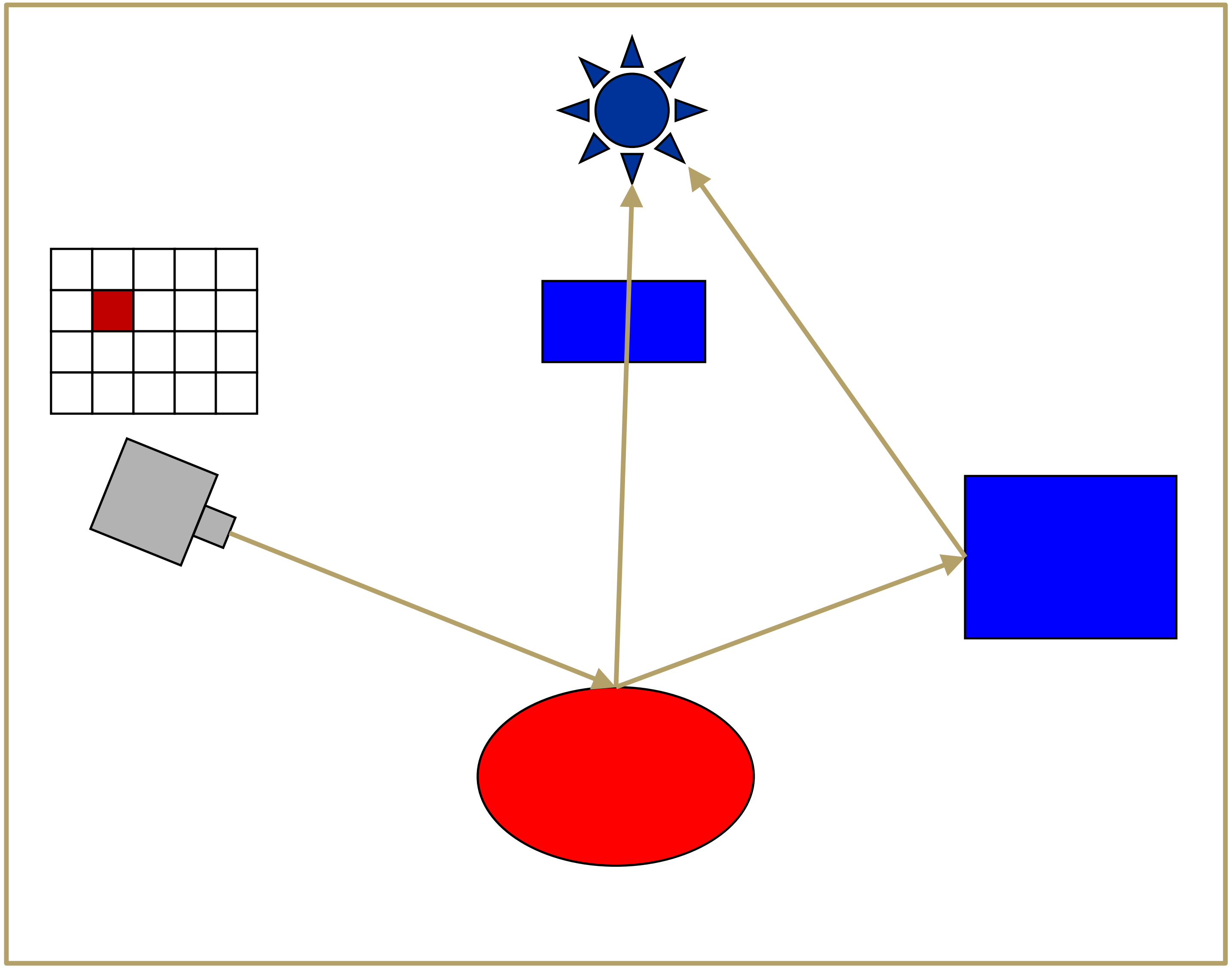


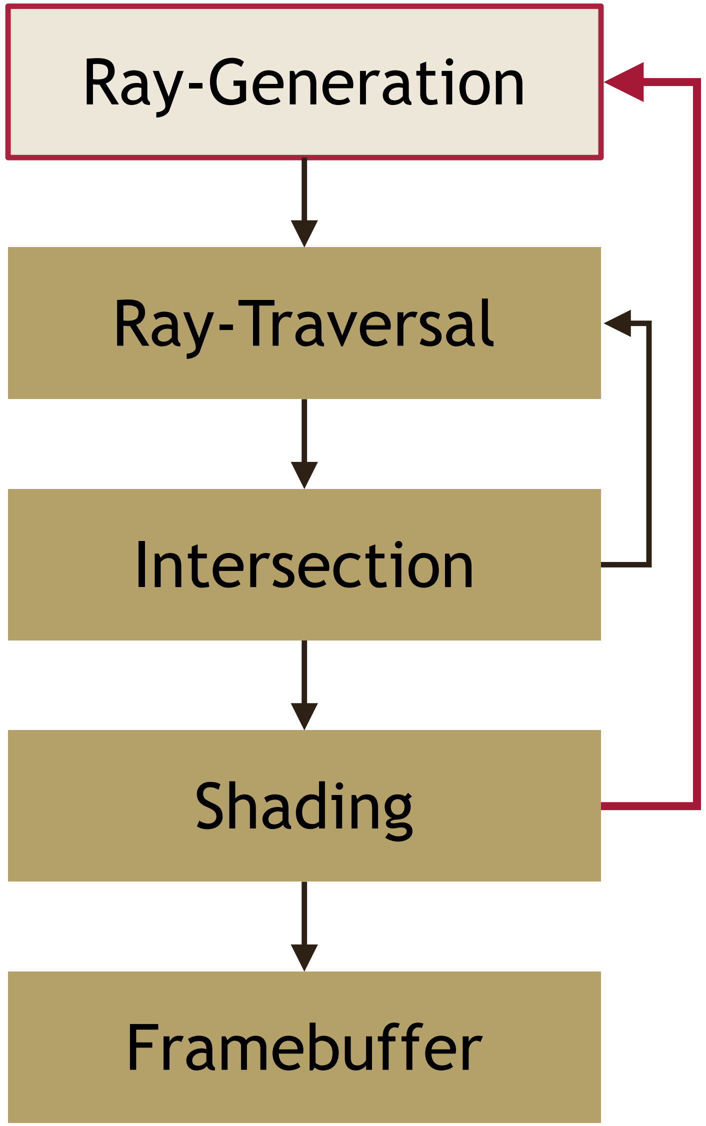
Ray Tracing Pipeline



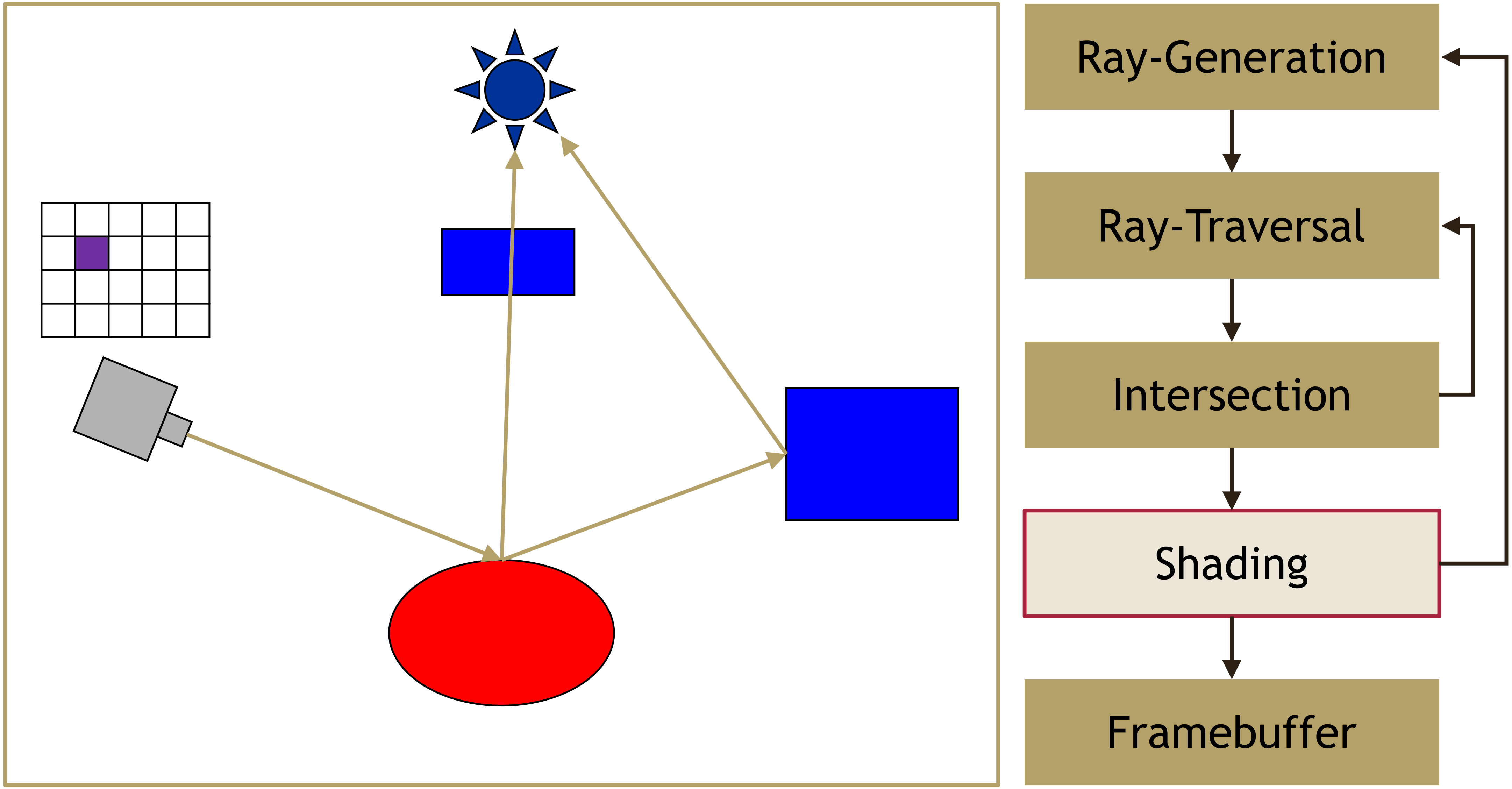


Ray Tracing Pipeline

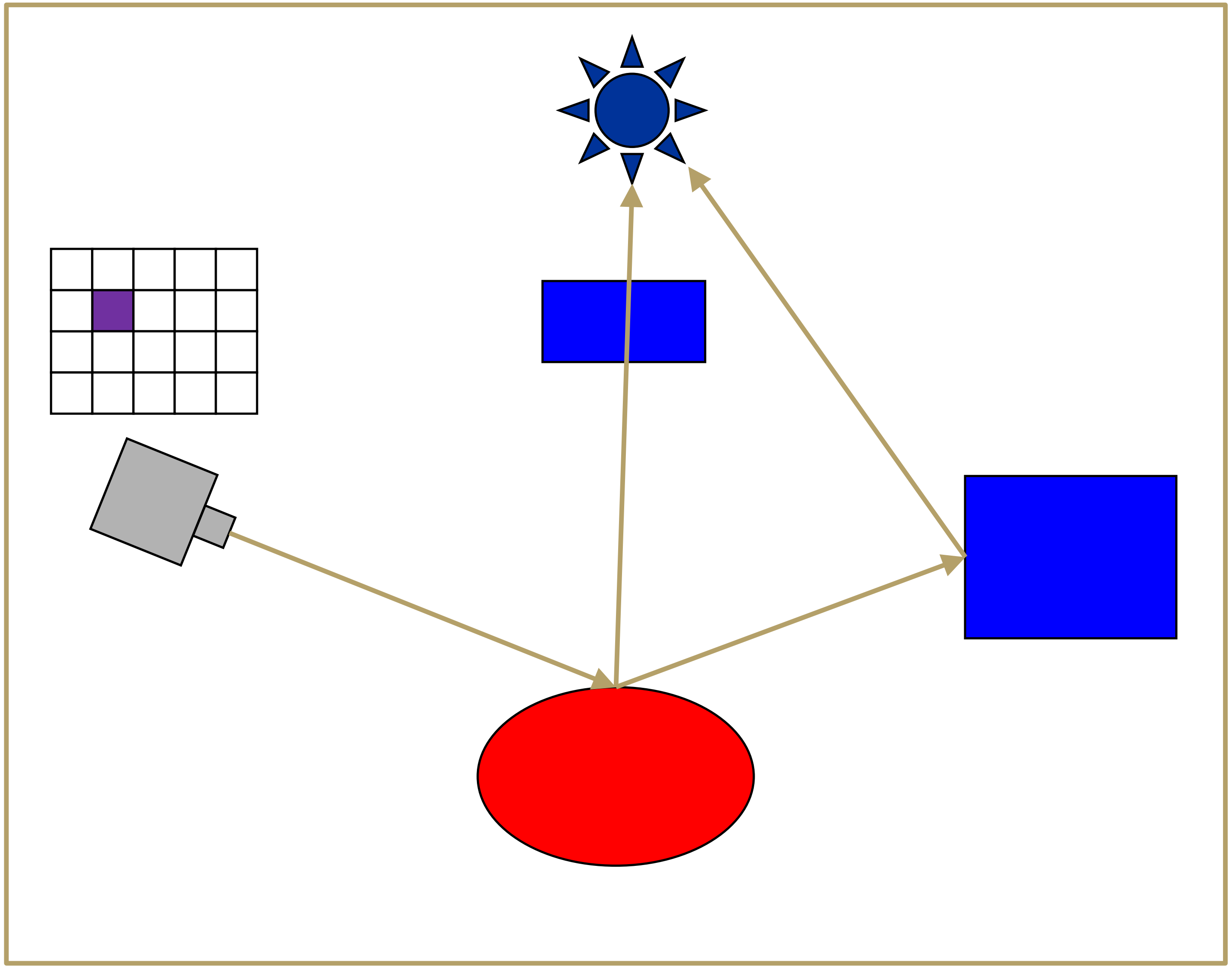


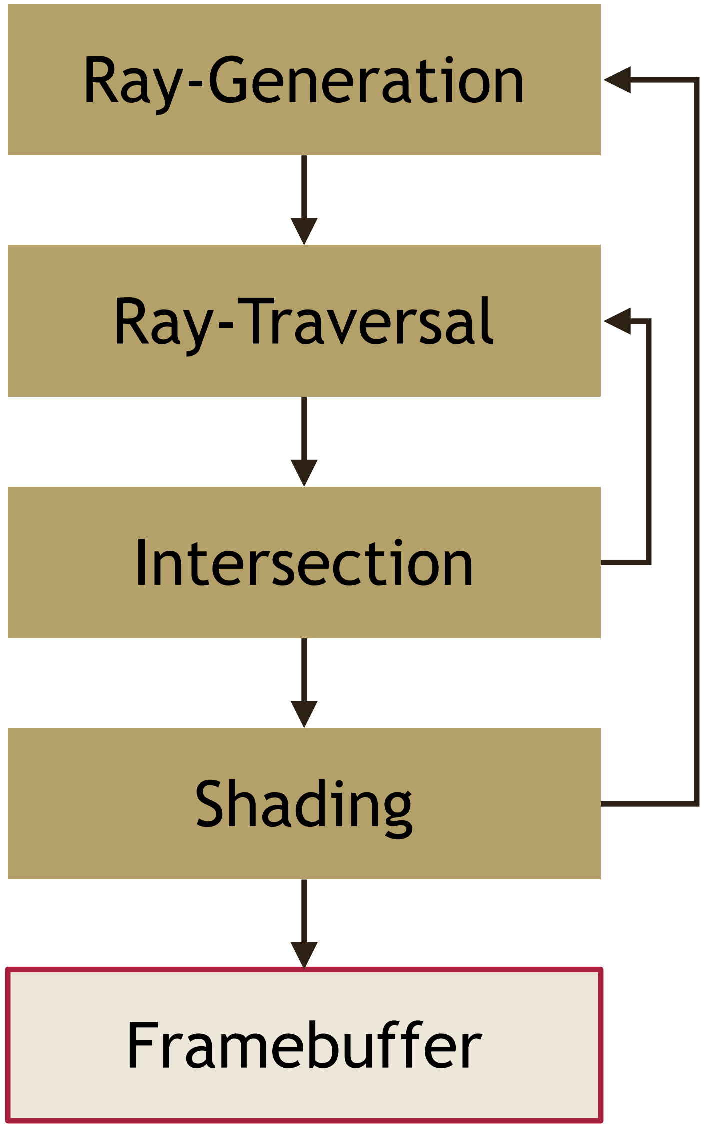


Ray Tracing Pipeline



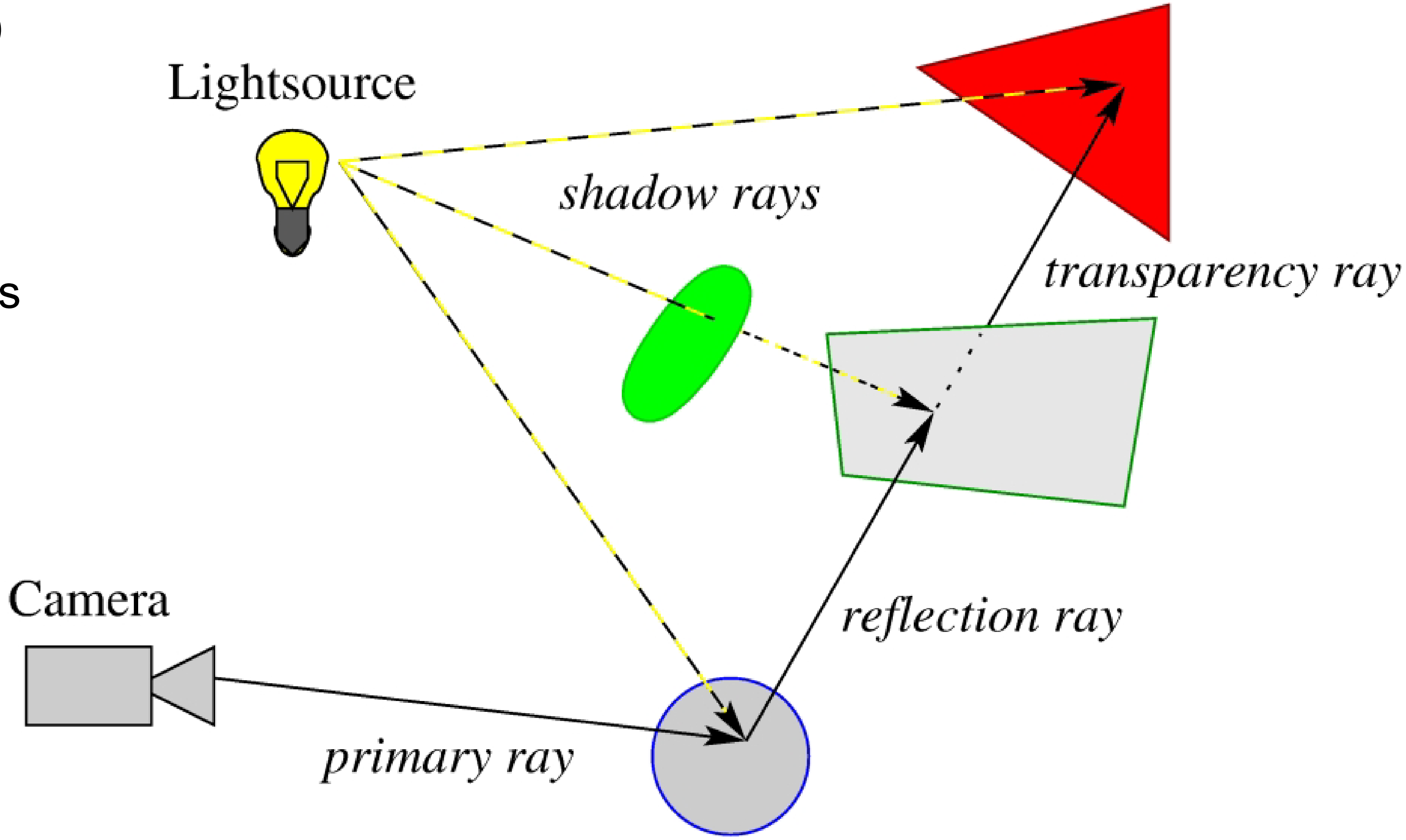
Ray Tracing Pipeline





Ray Tracing

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



Intersection Ray – Scene

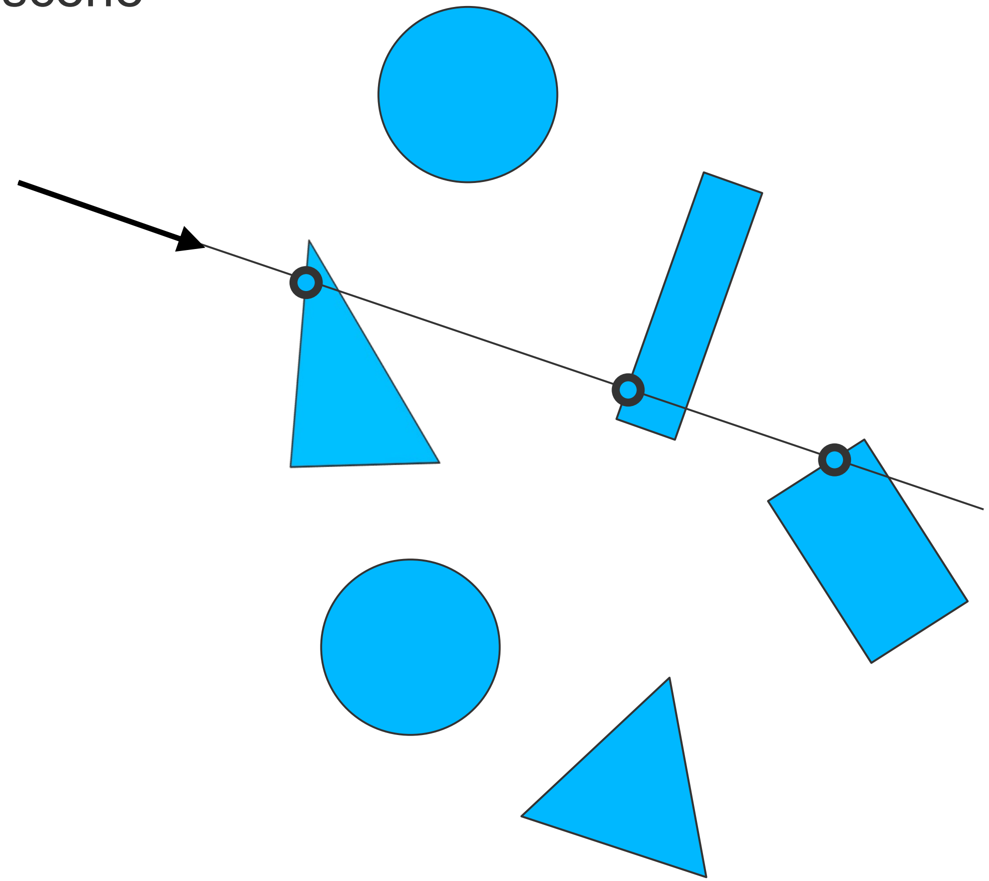
Which object is hit first?

Intersection Ray – Scene

- Find intersection with front-most surface in the scene

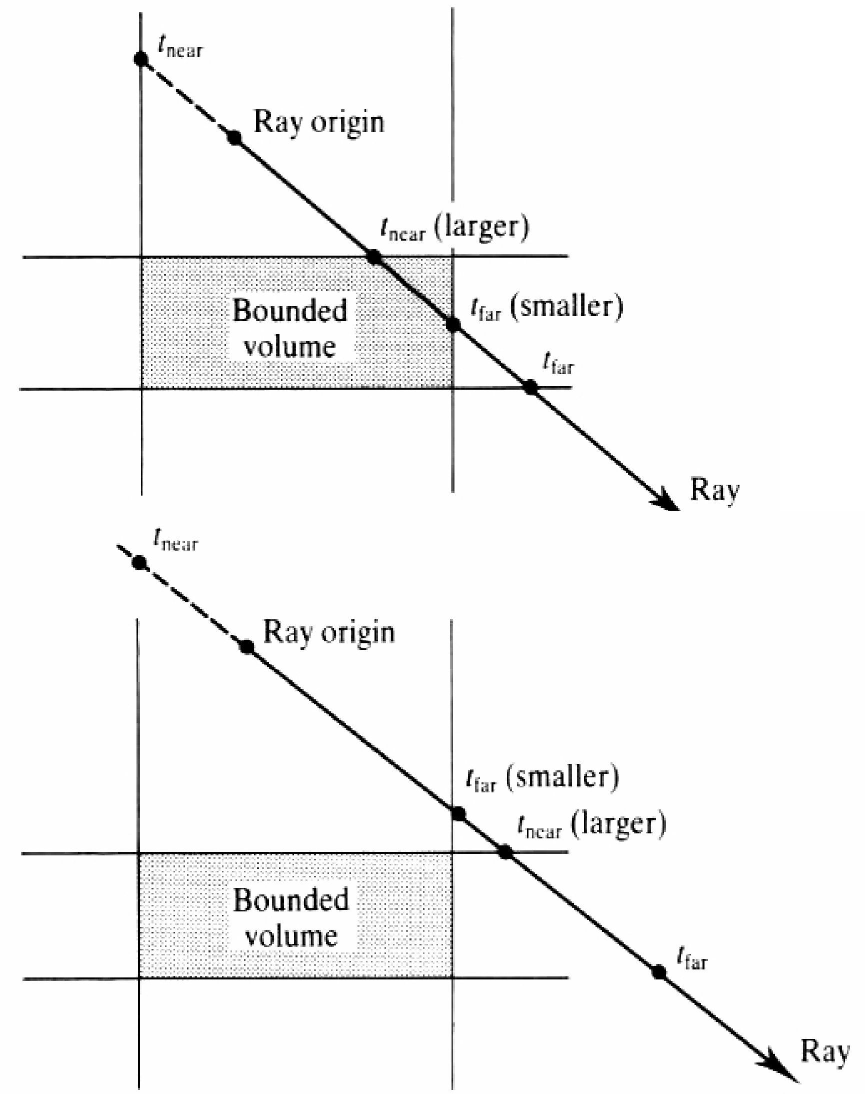
- Naïve algorithm:

```
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
```



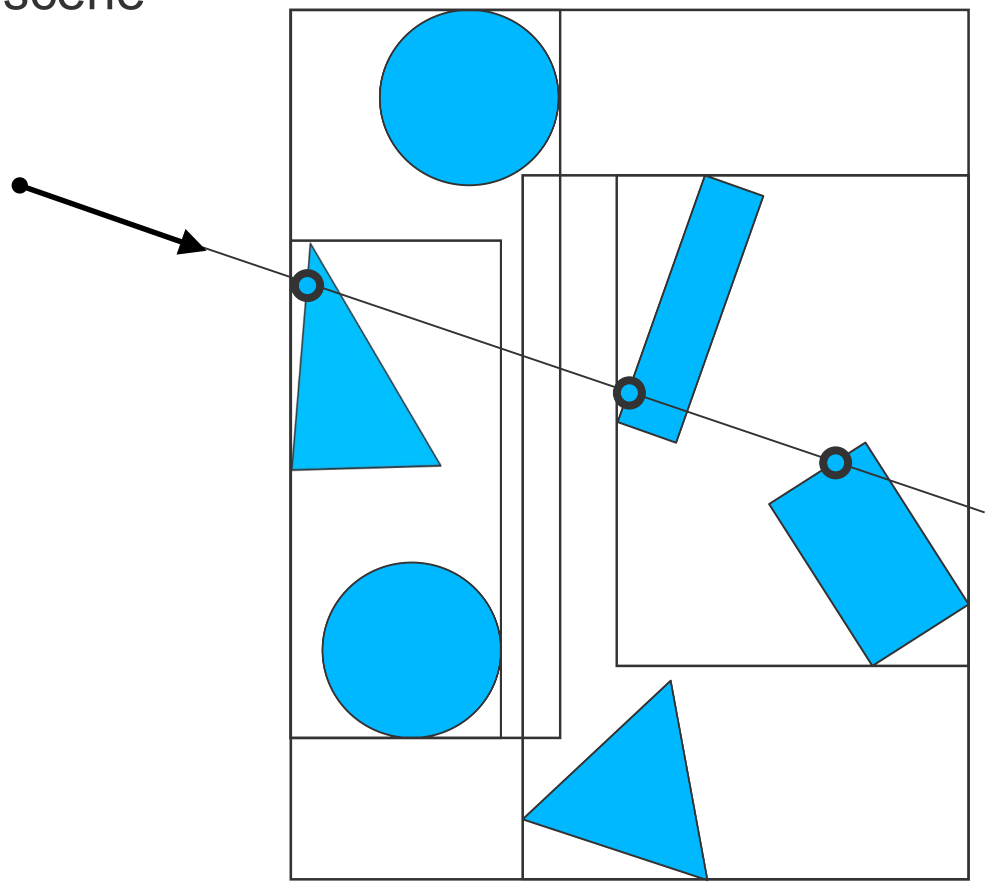
Intersection Ray – Box

- 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



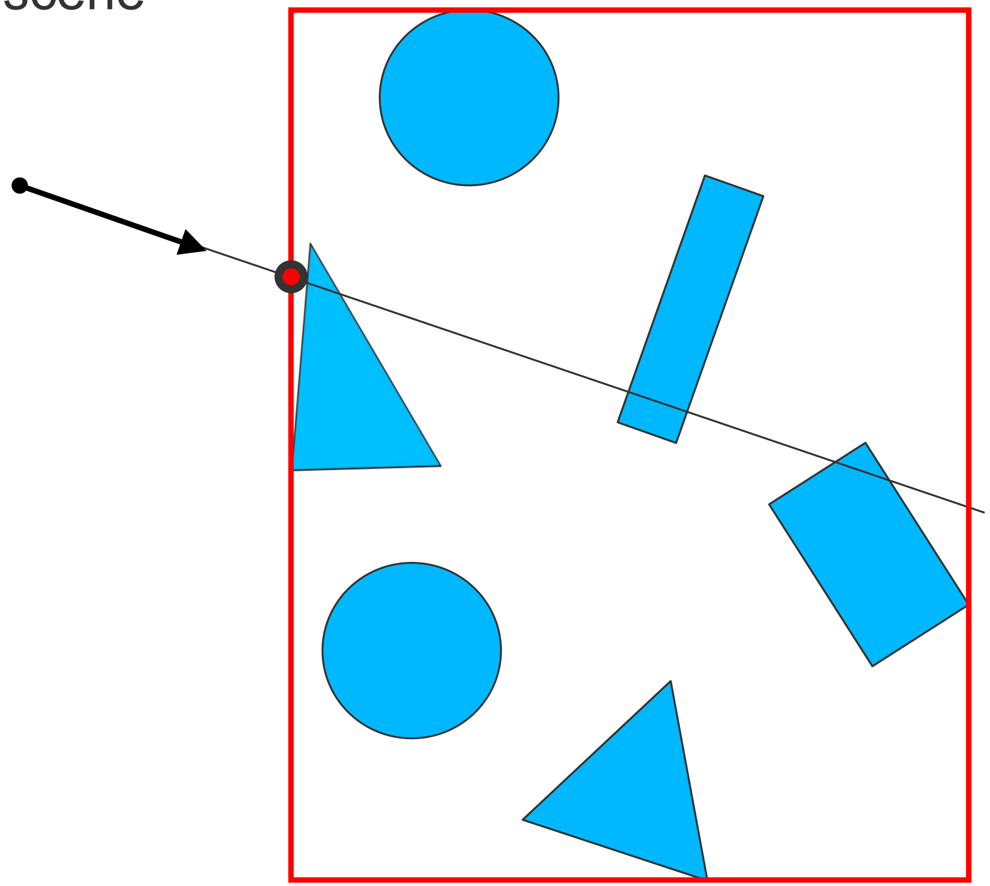
Intersection Ray – Scene BBox

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



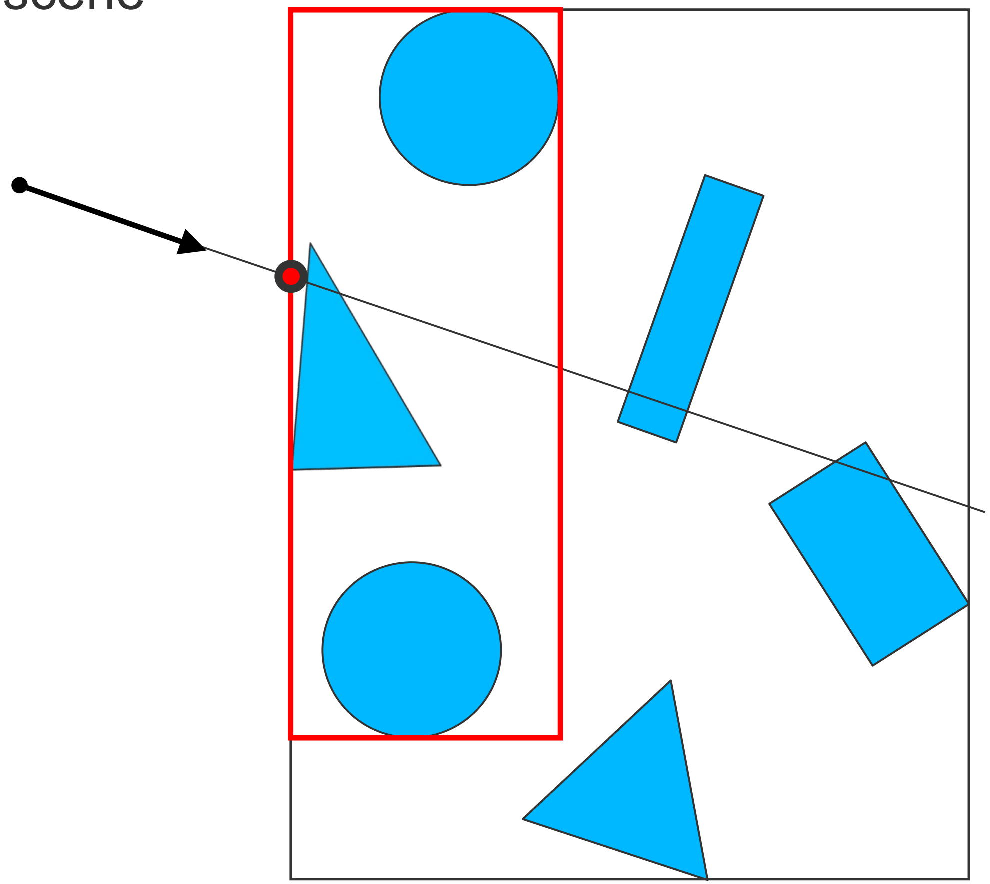
Intersection Ray – Scene BBox

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



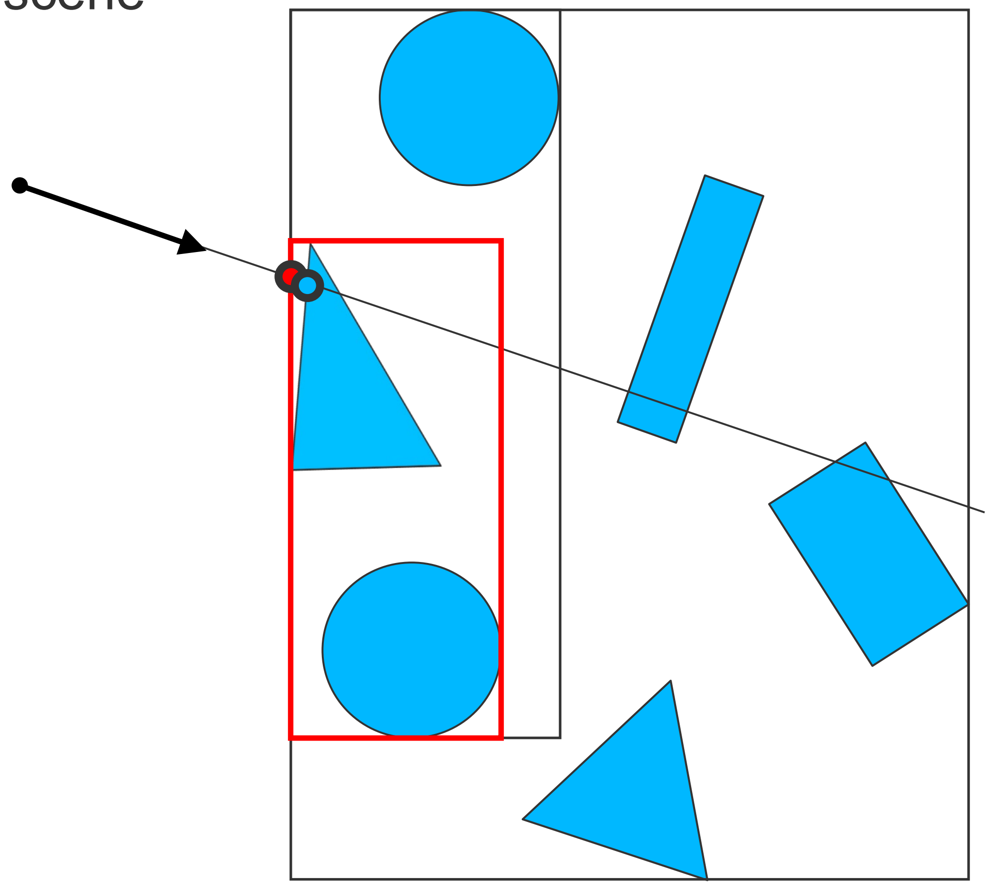
Intersection Ray – Scene BBox

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



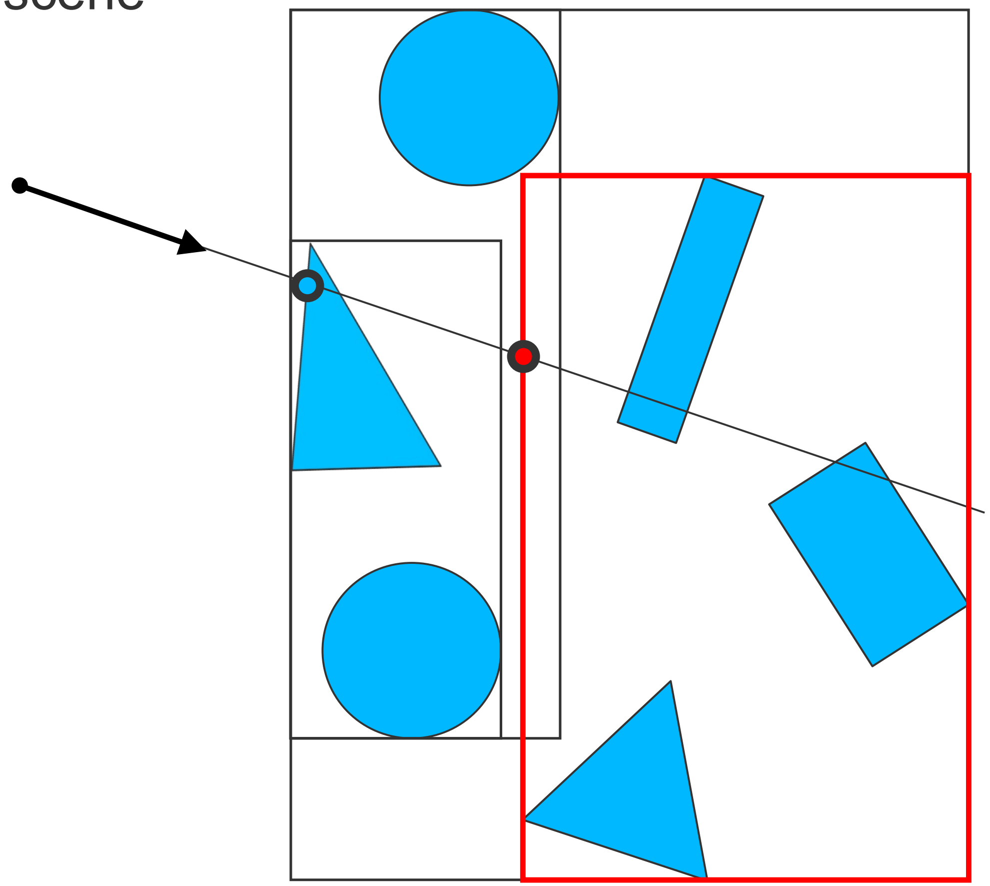
Intersection Ray – Scene BBox

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



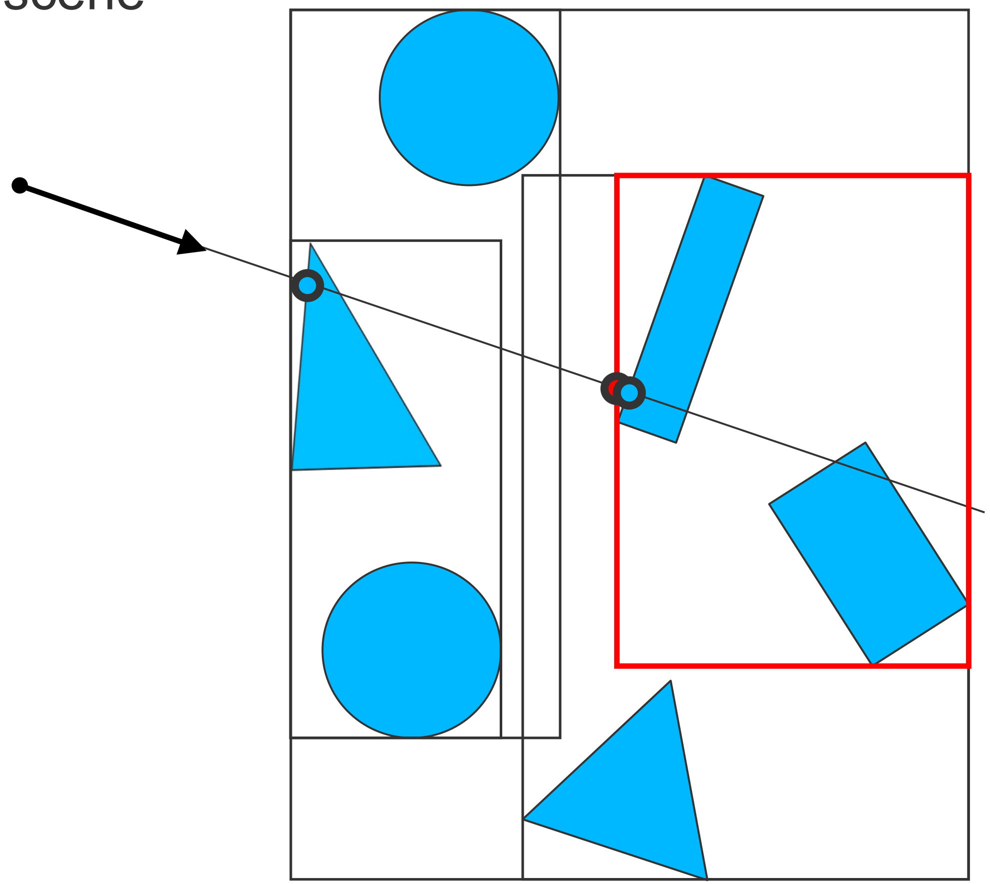
Intersection Ray – Scene BBox

- 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

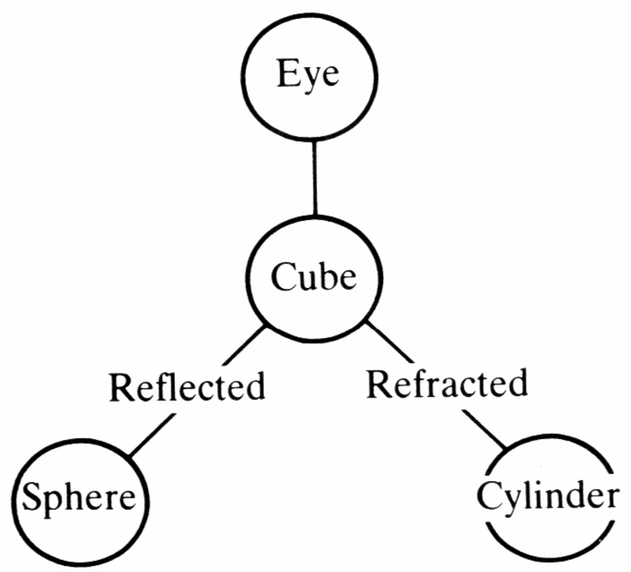
Which *color* do we observe along a ray?

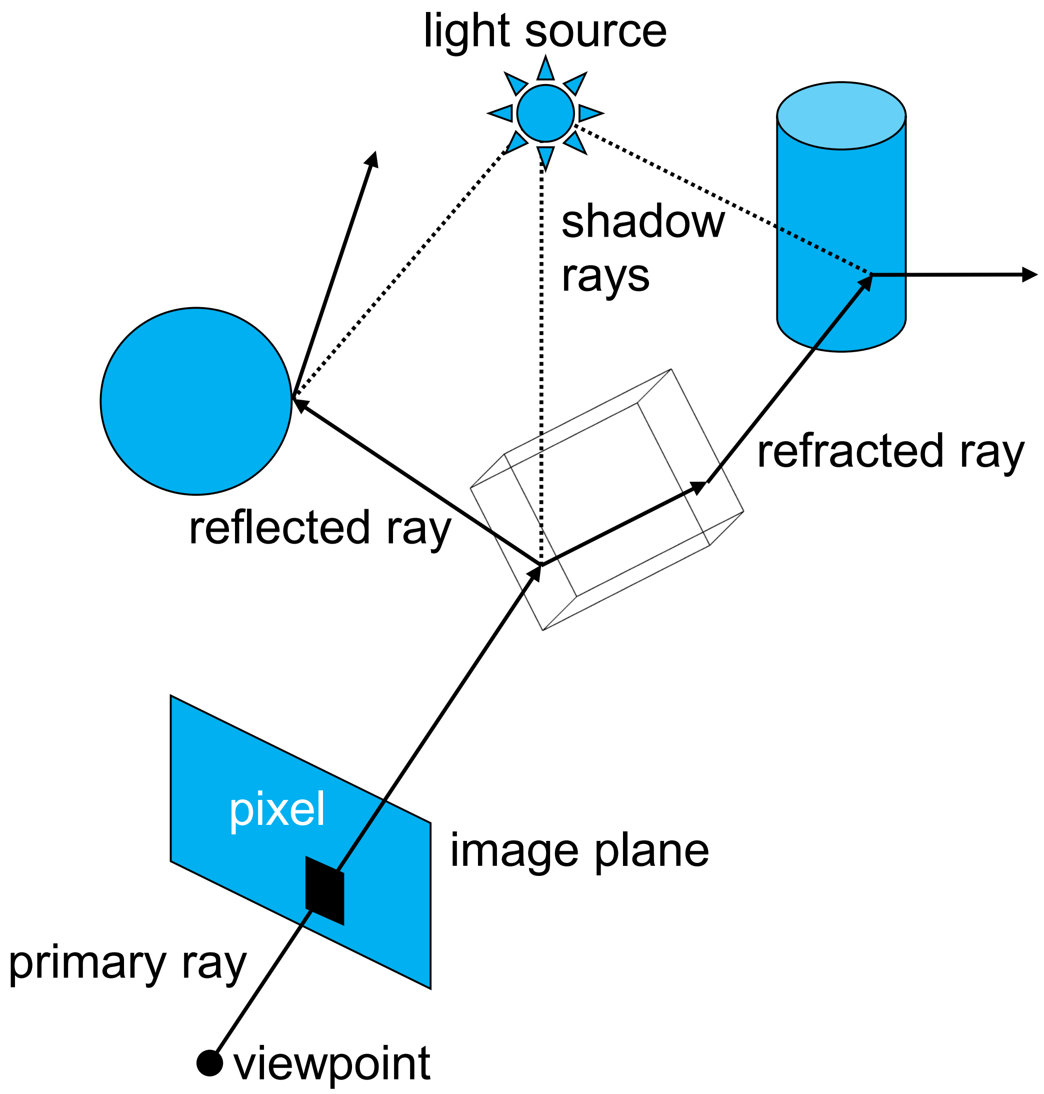
Shading

- Intersection point determines primary ray's *color*
- Diffuse object: $L = \rho(n \cdot l)L_i = \rho \cos\theta \, L_i$
  - 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

- 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

- How does the Z-buffer Algorithm solve the occlusion problem?
- How is the first surface extracted in a ray tracer?
- Write down and explain the principle steps of a recursive ray tracer.
- How do you evaluate the shading for a diffuse surface?

Wrap-Up

- 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 structures