Computer Graphics

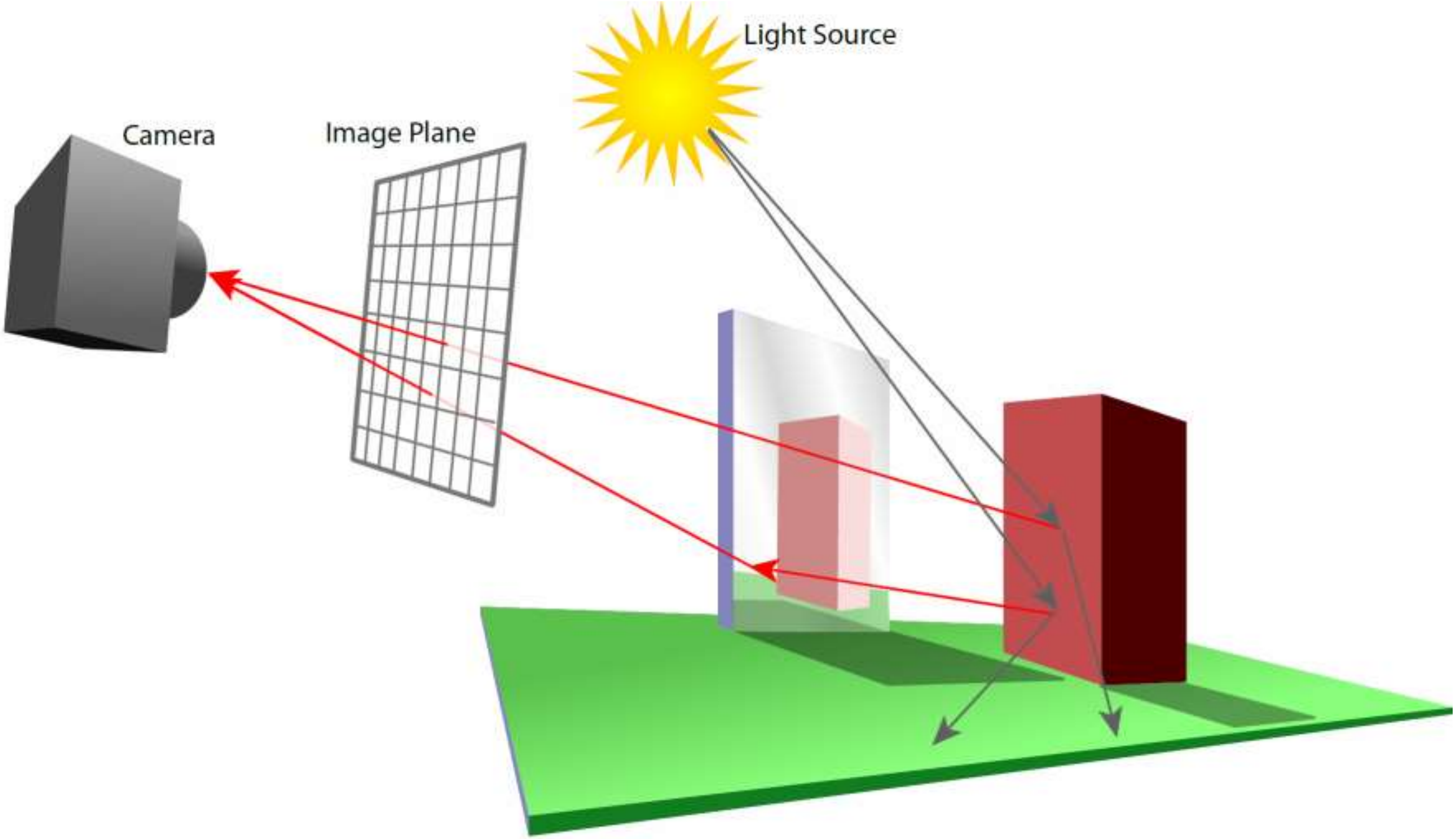
- Introduction to Ray Tracing -

Philipp Slusallek

Rendering Algorithms

- **Def.: Rendering**
  - Given a 3D scene as input and a camera, generate a 2D image as a view from the camera of the 3D scene
- **In this course**
  - Scene
    - Set of surfaces in R^3 described by
      - Geometric primitives, e.g. spheres, polygons, triangles, …
      - Appearance, e.g. reflectance color, light emission, texture, …
      - Later also volume objects, e.g. smoke, solid object (CT scan), …
    - Camera
      - View point, viewing direction, field of view, resolution, …
  - Algorithms
    - Rasterization
      - Traditional procedural/imperative drawing of a scene content
    - Ray Tracing
      - Declarative scene description
      - Physically-based simulation of light transport

Light Transport (1)



Light Transport (2)

- **Light Distribution in a Scene**
  - Dynamic equilibrium of light emitting light sources and light absorbing surfaces (or volumes)
- **Forward Light Transport**
  - Shoot photons from the light sources into scene
  - Reflect at surfaces and record when a detector is hit
    - Photons that hit the camera produce the final image
    - Most photons will not reach the camera
  - Particle Tracing
- **Backward Light Transport**
  - Start at the detector (camera)
  - Trace only paths that might transport light towards it
    - Try to connect to light sources
  - Ray Tracing

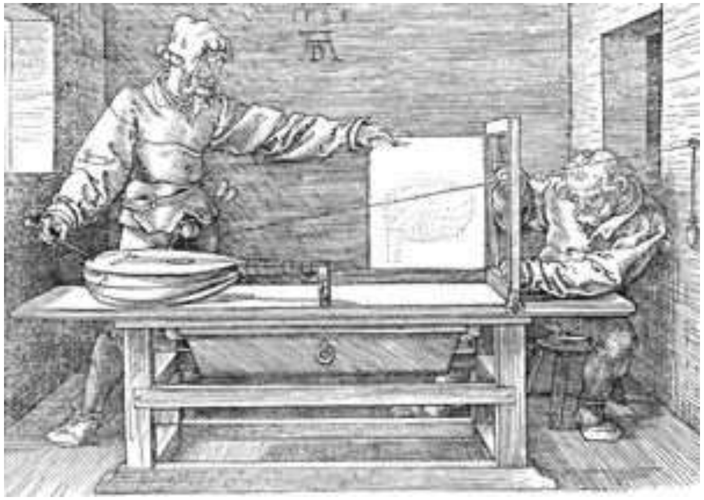
Ingredients

- **Surface Geometry**
  - 3D geometry of objects in a scene
  - Geometric primitives – triangles, spheres, “points”, …
- **Surface Appearance**
  - Color, absorption, reflection, refraction, subsurface scattering
  - Typical material types: Diffuse, glossy, mirror, glass, …
- **Illumination**
  - Position, characteristics of light emitters
  - Note: Light is reflected off of surfaces!
    - Secondary/indirect/global illumination
  - Assumption: air/empty space is totally transparent
    - Simplification that excludes scattering effects in participating media volumes (for now)

OVERVIEW OF RAY-TRACING

Ray Tracing Is…

- **Fundamental rendering algorithm**
- **Automatic, simple and intuitive**
  - Easy to understand and implement
  - Delivers “correct“ images by default
- **Powerful and efficient**
  - Many optical global effects
    - Shadows, reflections, refractions, …
  - Efficient “real-time” implementation in SW and HW
  - Works well in parallel and distributed environments
  - Logarithmic scalability with scene size: O(log n) vs. O(n)
  - Output sensitive and demand driven
- **Concept of light rays is not new**
  - Empedocles (492-432 BC), Renaissance (Dürer, 1525), …
  - Used in lens design, lighting design, heat simulation, radar, …



Perspective Machine, Albrecht Dürer

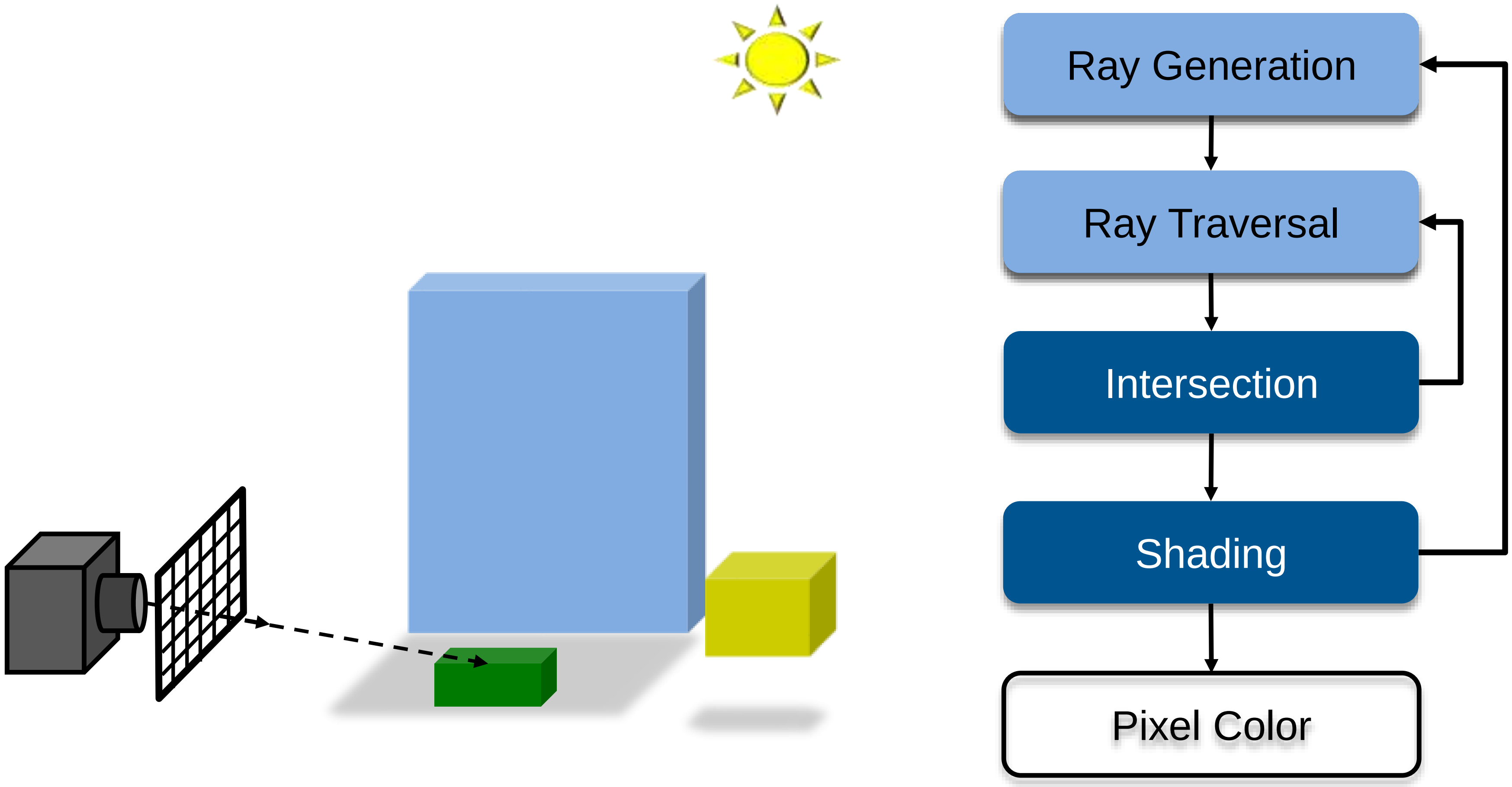
Ray Tracing Can…

- **Produce Realistic Images**
  - By simulating light transport

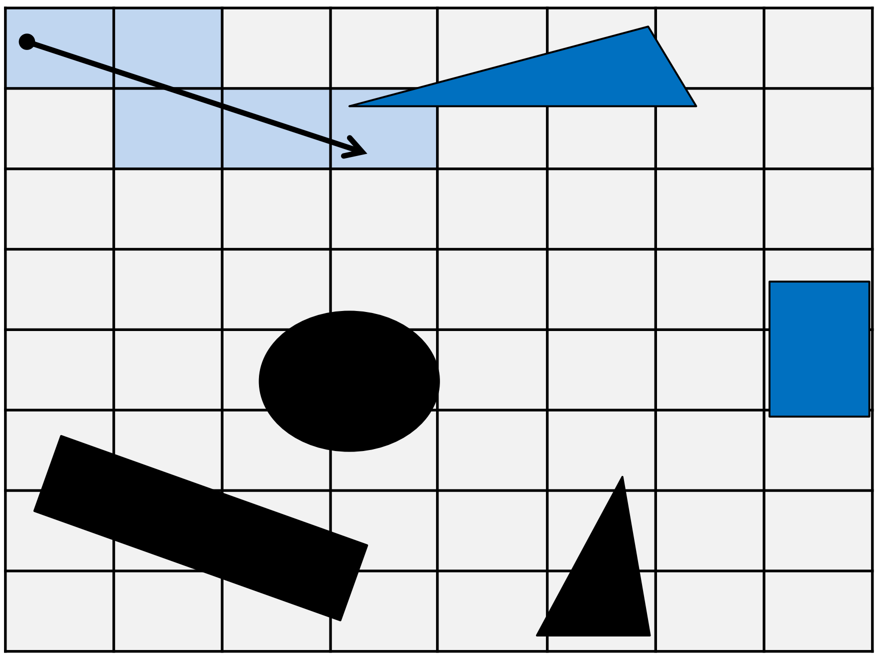
Fundamental Ray Tracing Steps

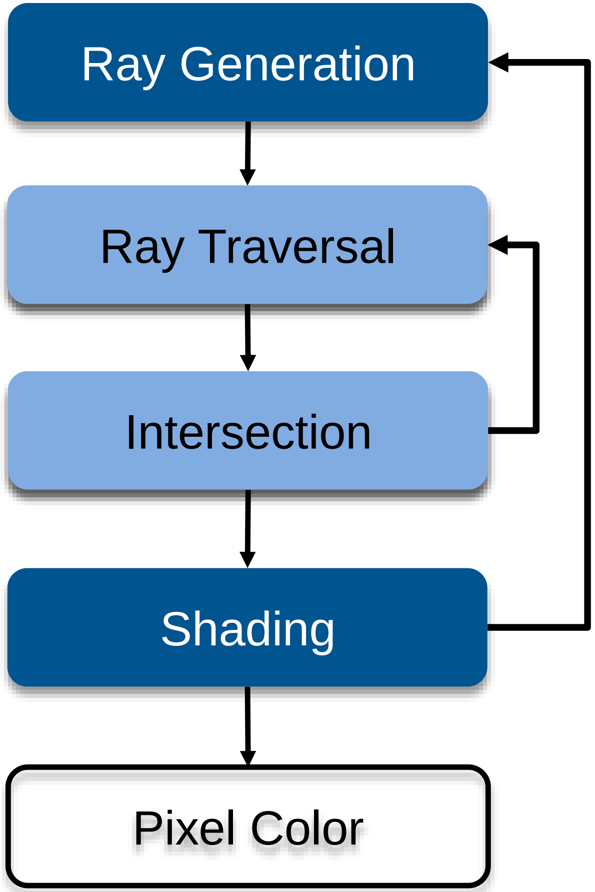
- **Generation of primary rays**
  - Rays from viewpoint along viewing directions into 3D scene
  - (At least) one ray per picture element (pixel)
- **Ray Casting: Finding the first hit point**
  - Traversal of spatial index structures
  - Ray-primitive intersection
- **Shading the hit point**
  - Determine pixel color
    - Energy (color) travelling along primary ray
  - Needed
    - Local material color and reflection properties
    - Object texture
    - Illumination at the intersection point
      - Computed through (recursive) tracing of rays
      - Can be hard to determine correctly

Ray Tracing Pipeline (1)

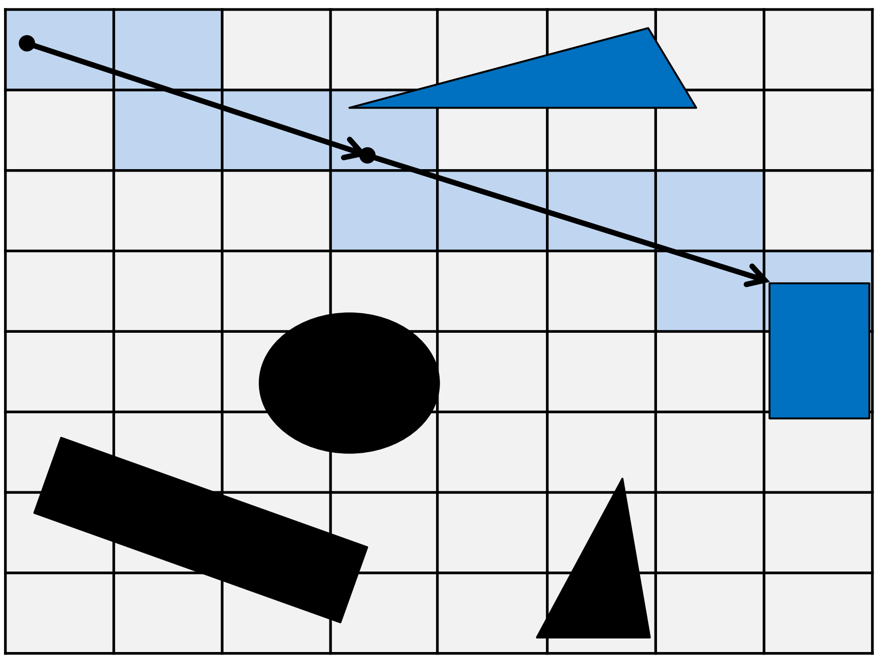


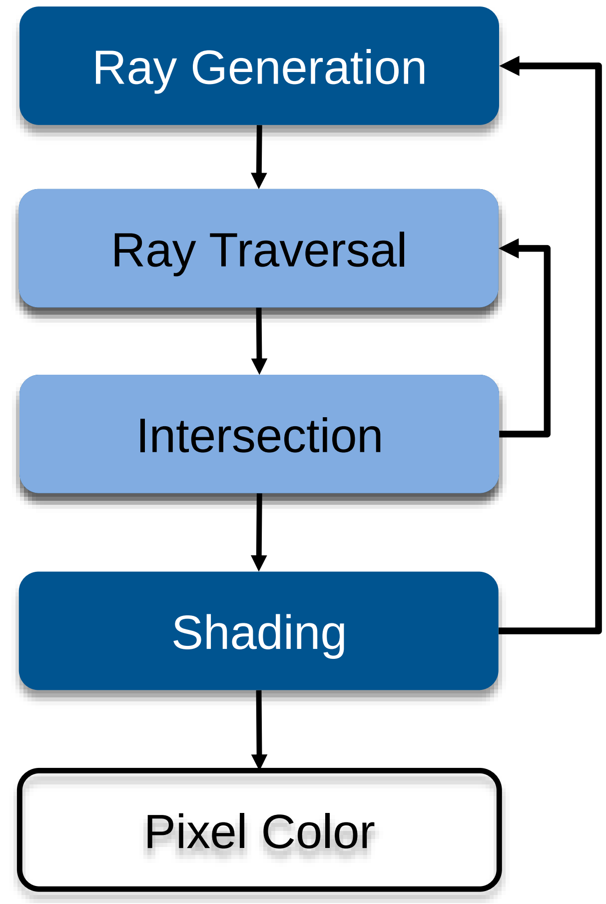
Ray Tracing Pipeline (2)



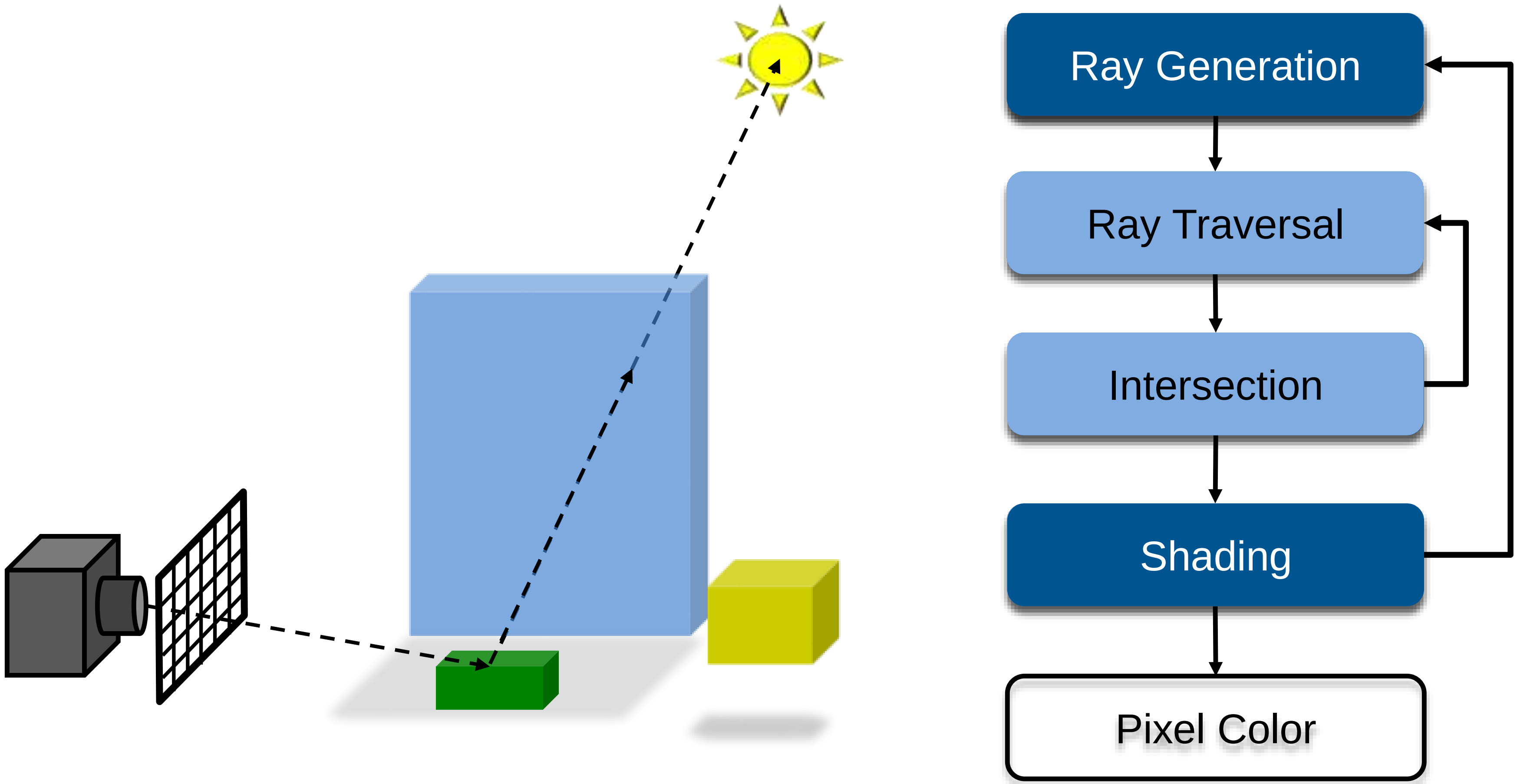


Ray Tracing Pipeline (3)

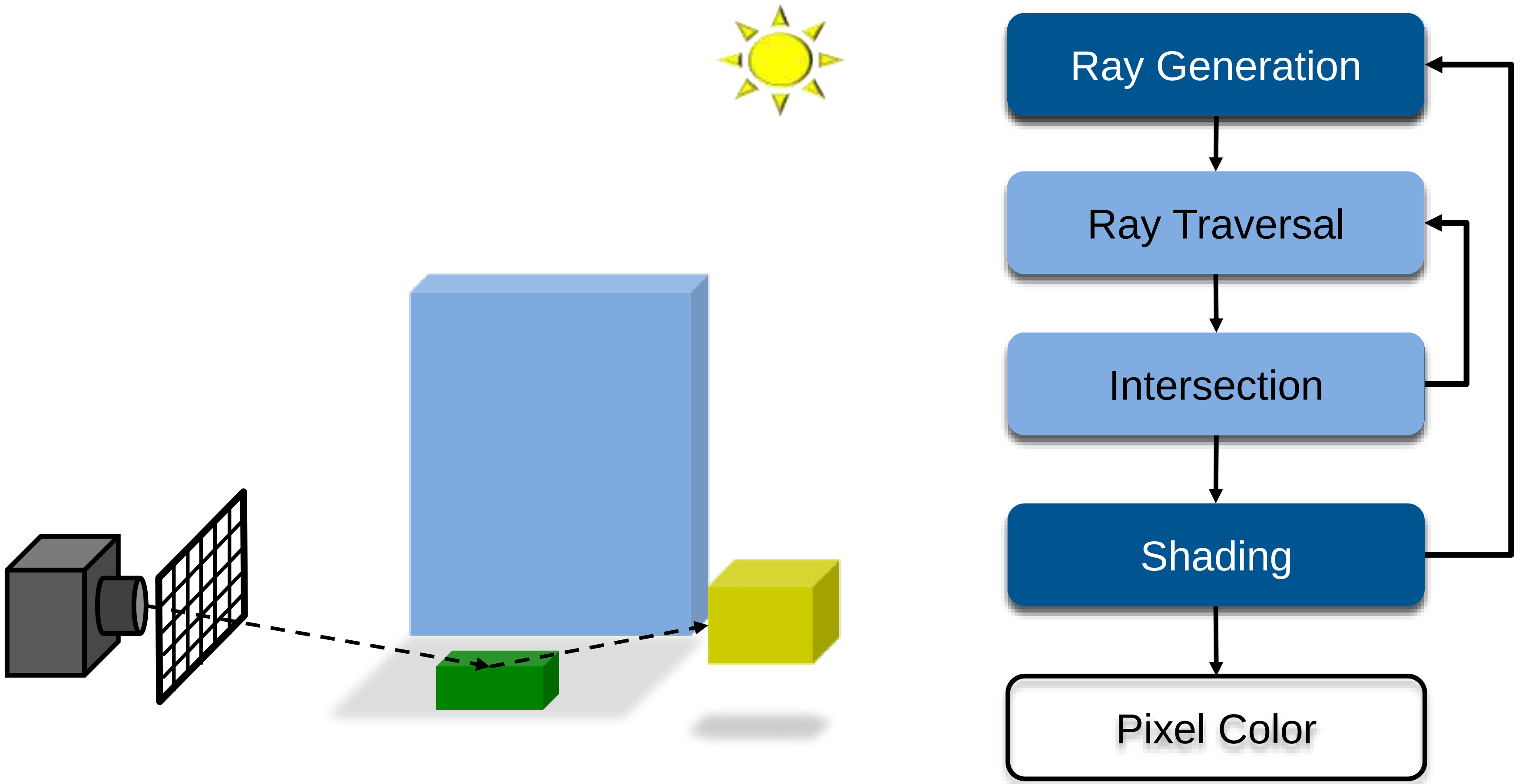




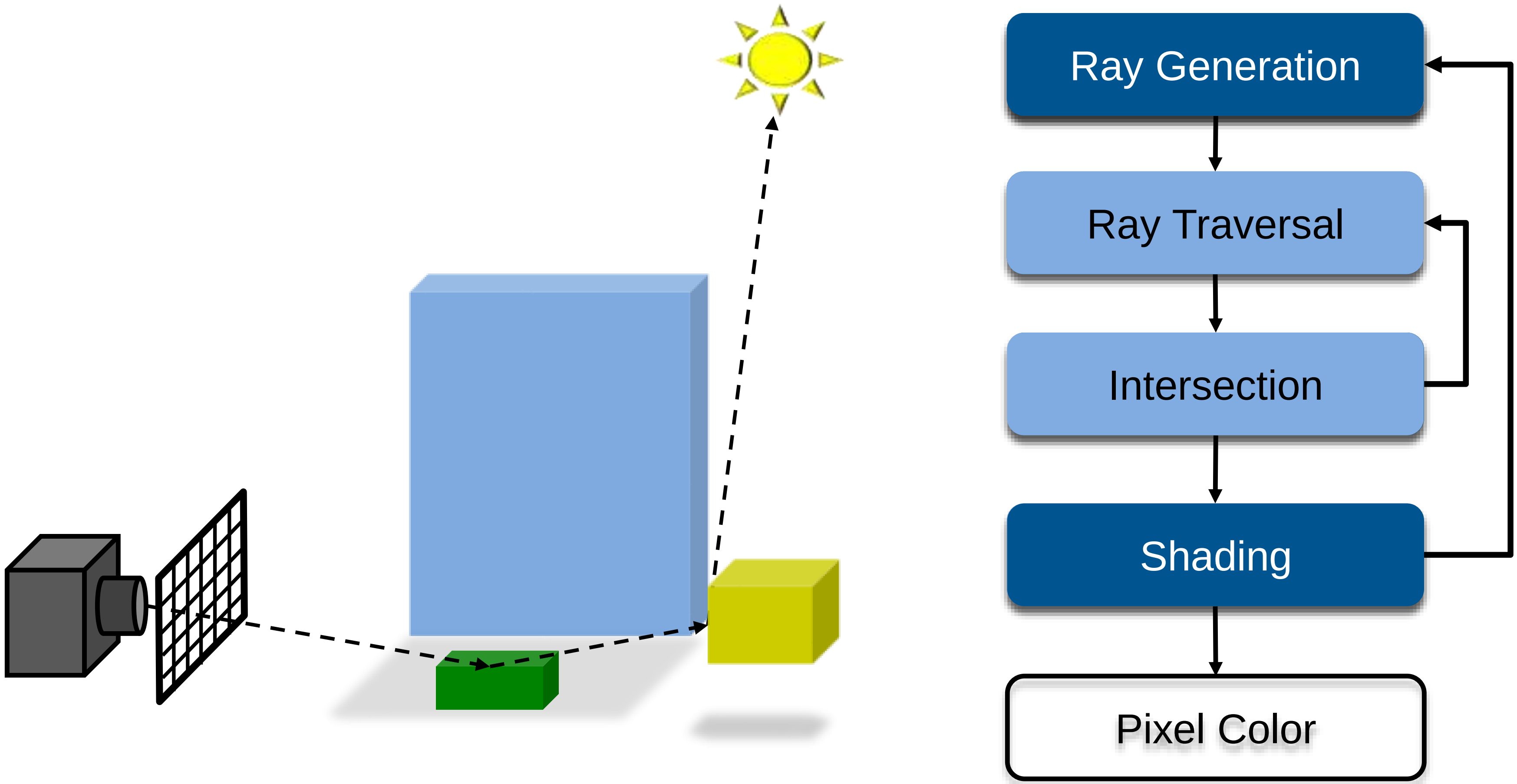
Ray Tracing Pipeline (4)



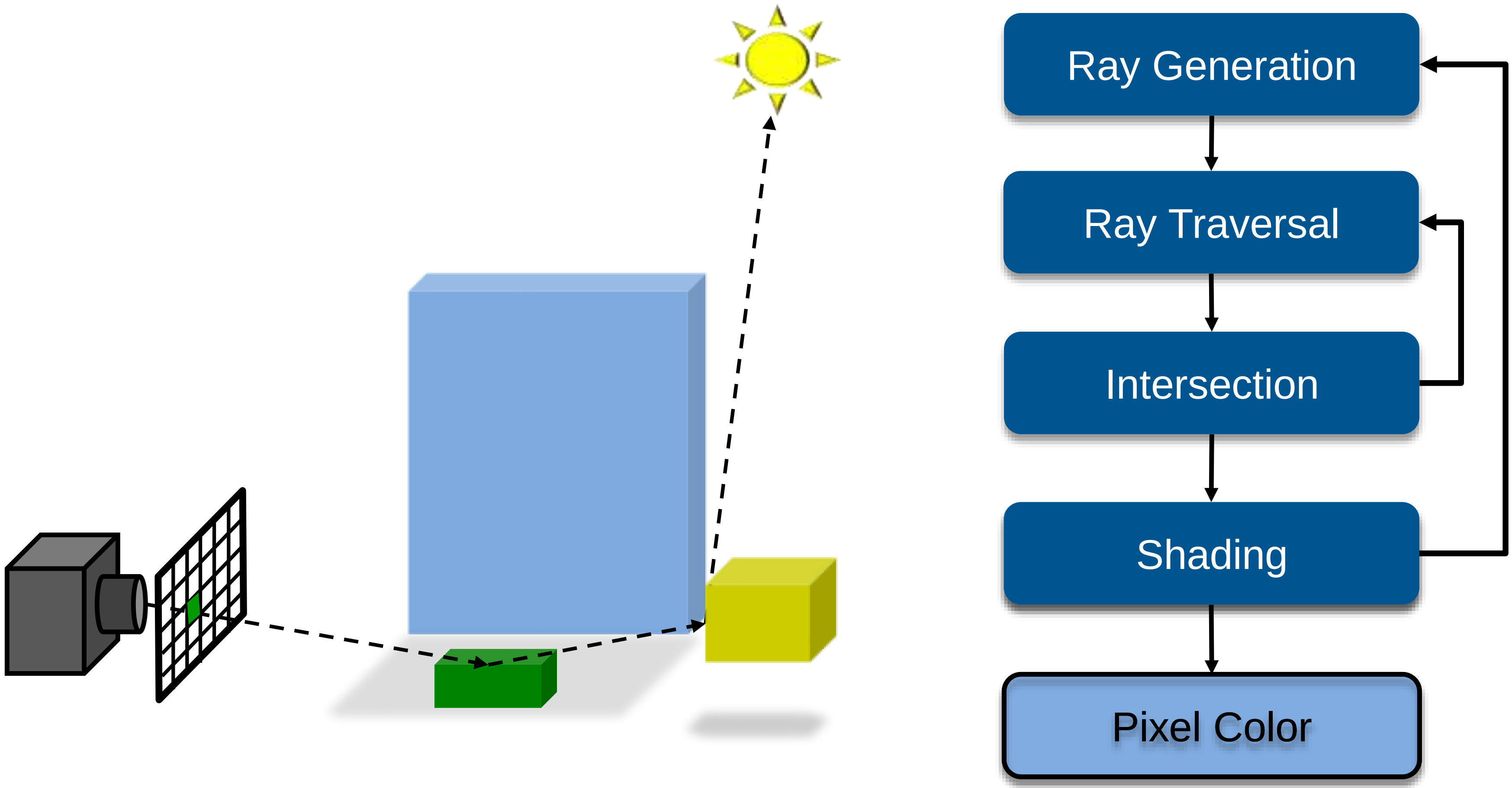
Ray Tracing Pipeline (5)



Ray Tracing Pipeline (6)



Ray Tracing Pipeline (7)



Ray Tracing in CG

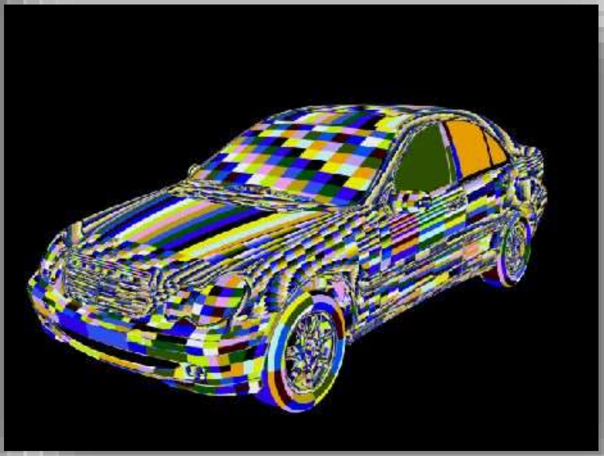
- **In the Past**
  - Only used as an off-line technique
  - Was computationally far too demanding
  - Rendering times of minutes and hours
- **More Recently**
  - Interactive ray tracing on supercomputers [Parker, U. Utah‘98]
  - Interactive ray tracing on PCs [Wald‘01]
  - Distributed ray tracing on PC clusters [Wald’01]
  - Complete film industry has switched to ray tracing (Monte-Carlo)
- **Own conference**
  - Symposium on Interactive Ray Tracing, now High-Performance Graphics
- **Ray tracing systems**
  - Teaching: PBRT (offline, physically-based, based on book, OSS), …
  - Tools: Embree/OSPRey (Intel), OptiX (NVidia),
  - Commercial: MentalRay/iRay (MI), V-Ray (Chaos Group), …
  - Research: Mitsuba renderer (EPFL), imbatracer (SB)

What is Possible?

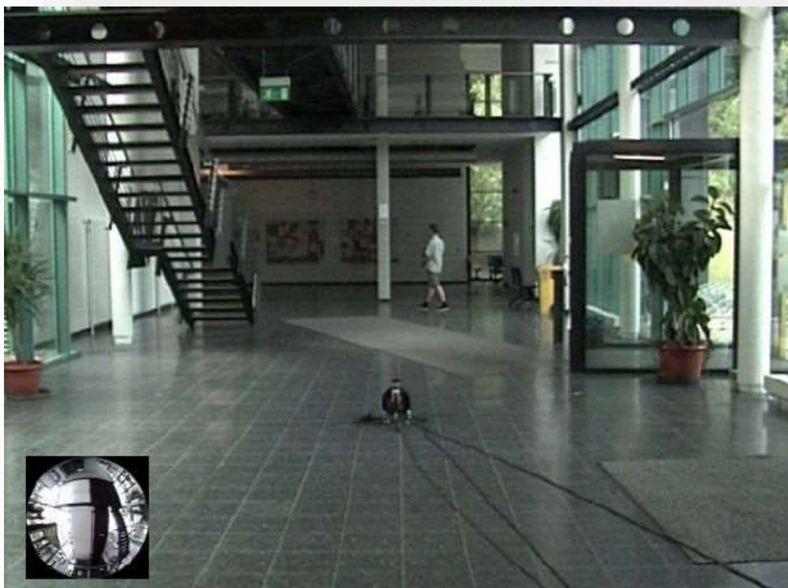
- **Models Physics of Global Light Transport**
  - Dependable, physically-correct visualization

VW Visualization Center

Realistic Visualization: CAD

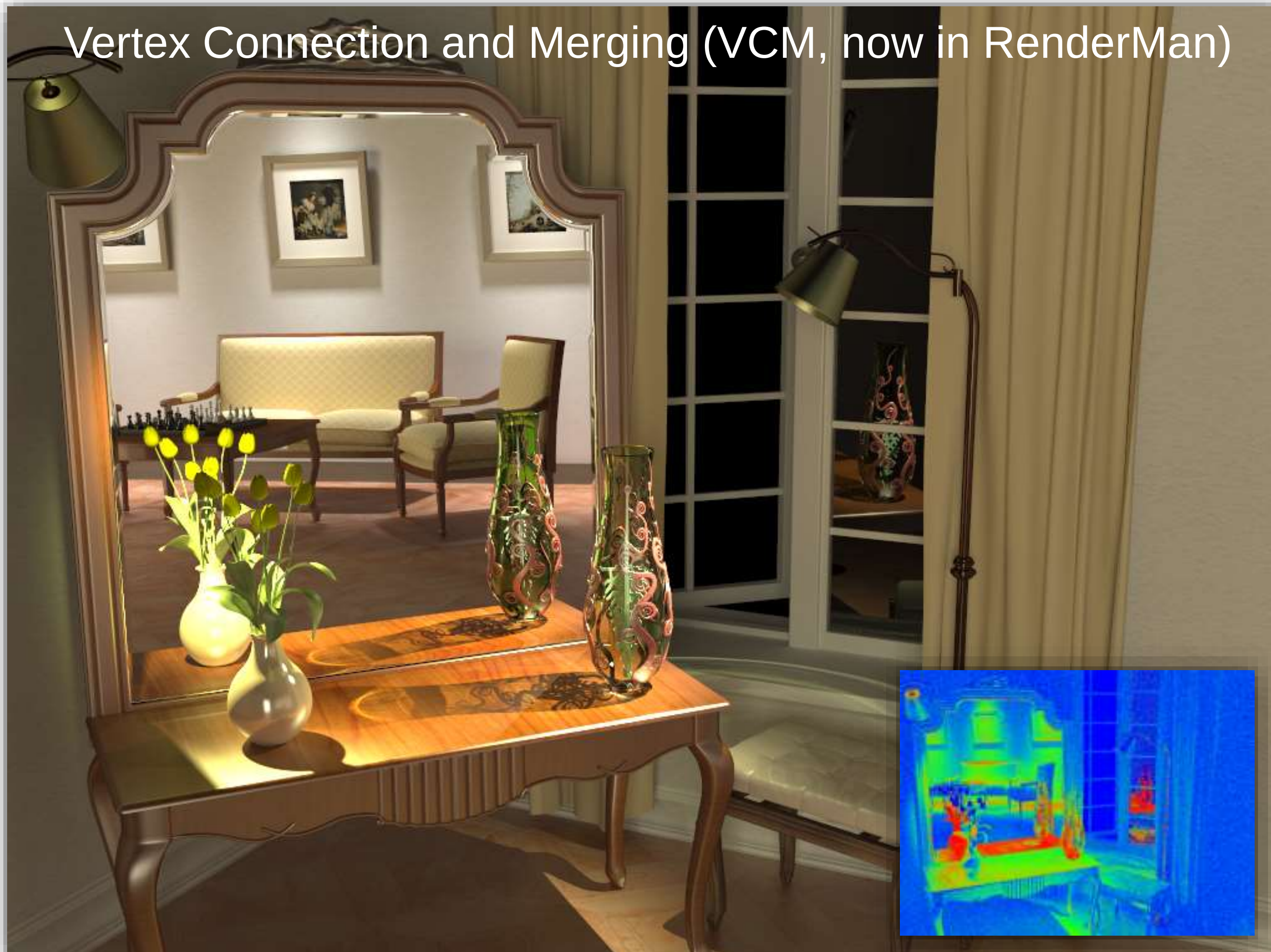


Realistic Visualization: VR/AR



Lighting Simulation

Lighting Simulation

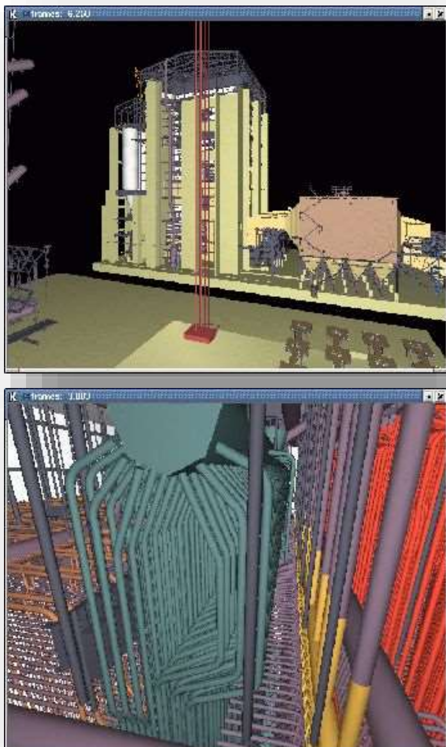


Vertex Connection and Merging (VCM, now in RenderMan)

What is Possible?

- **Huge Models**
  - Logarithmic scaling in scene size

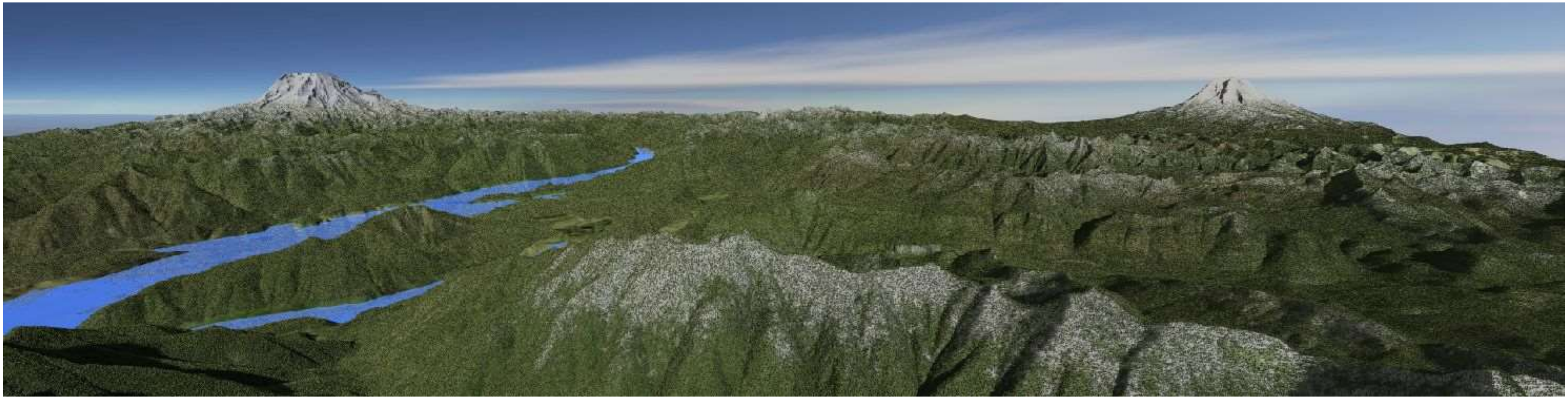
12.5 Million Triangles



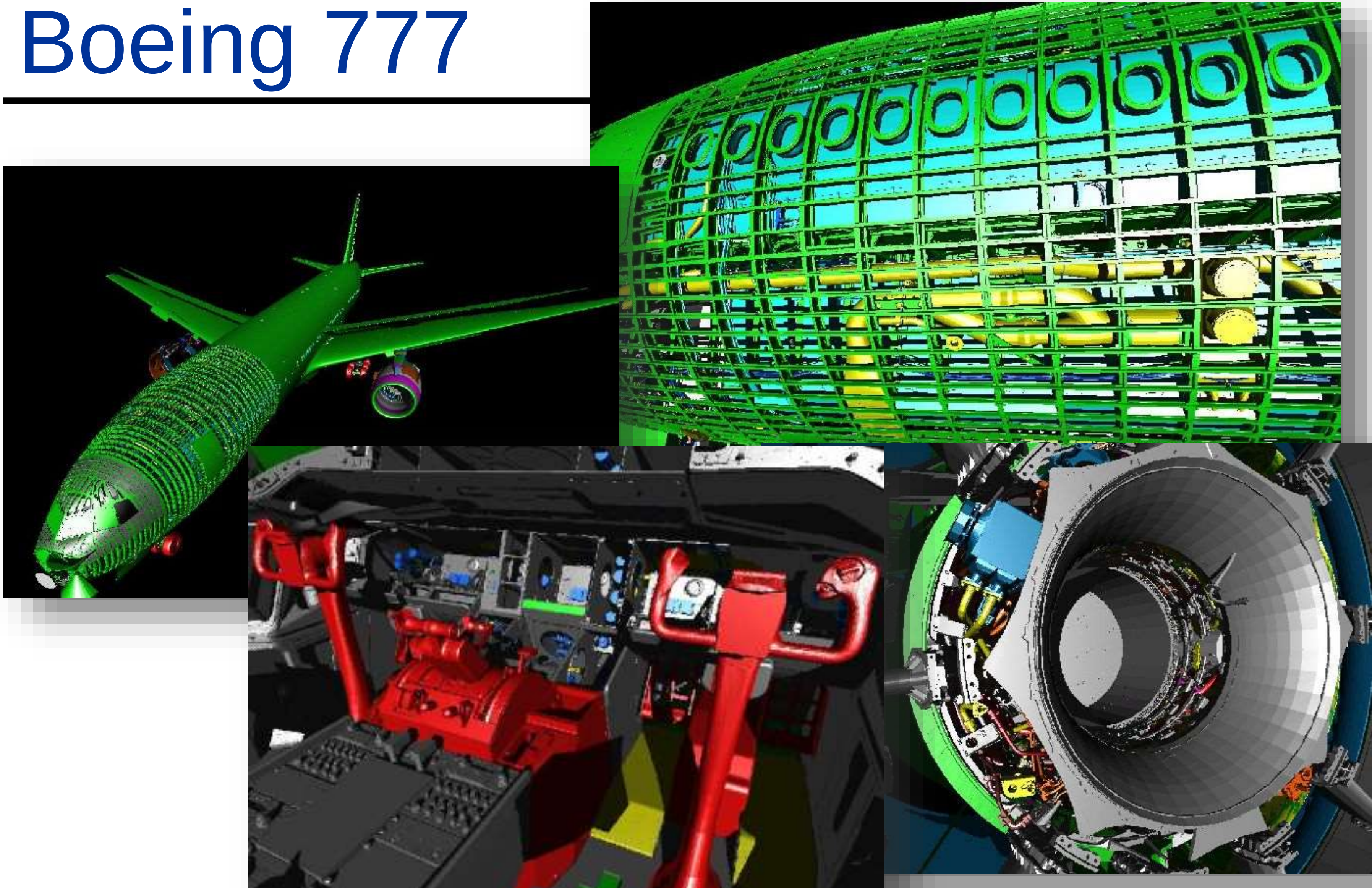
~1 Billion Triangles

Outdoor Environments

- **90 x 10^12 (trillion) triangles**



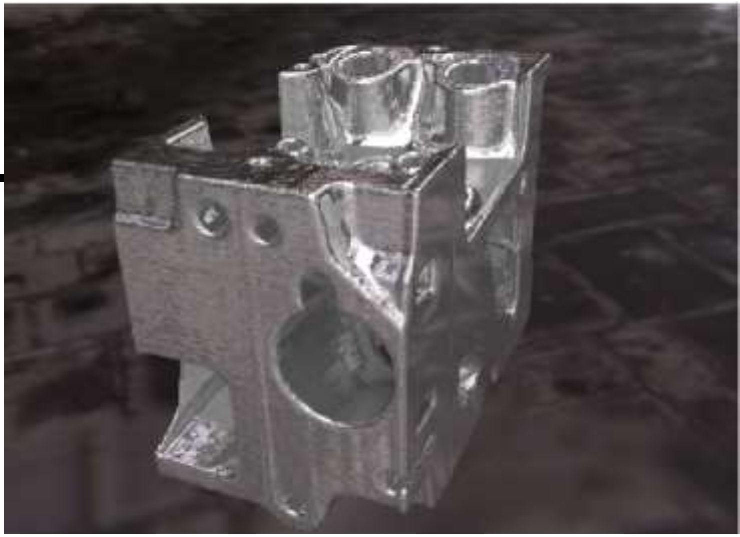

Boeing 777

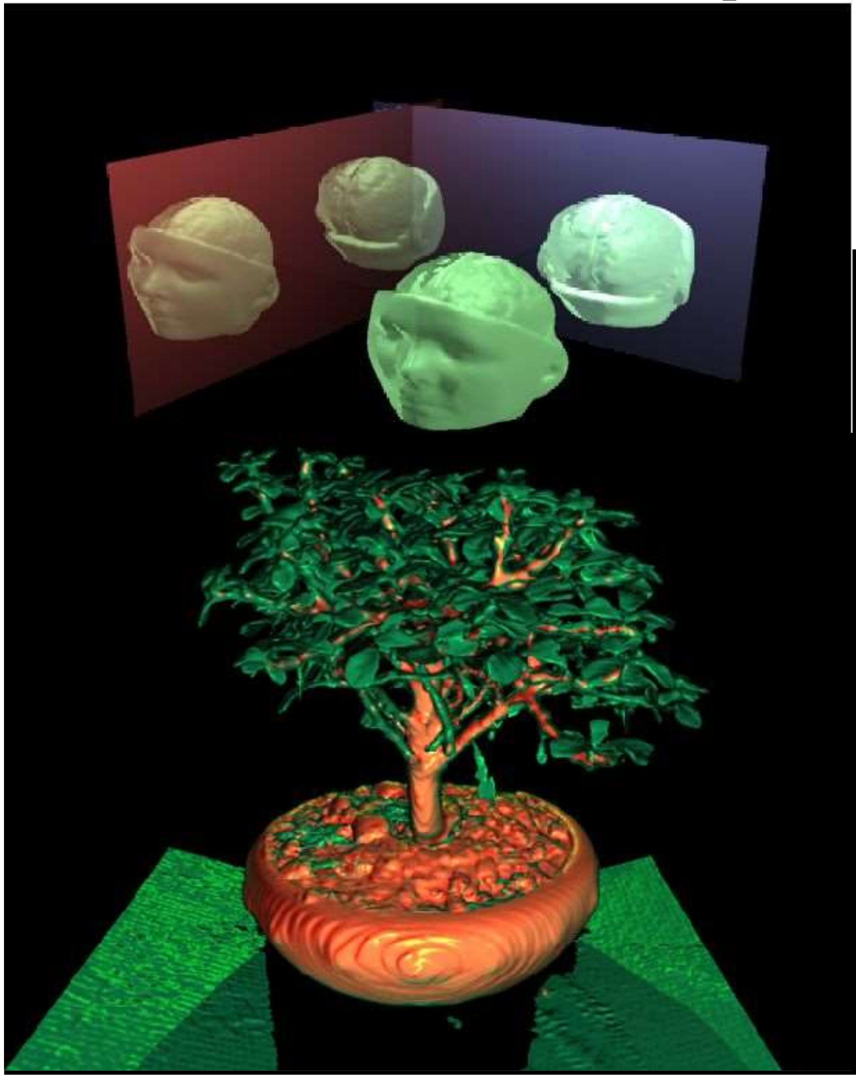


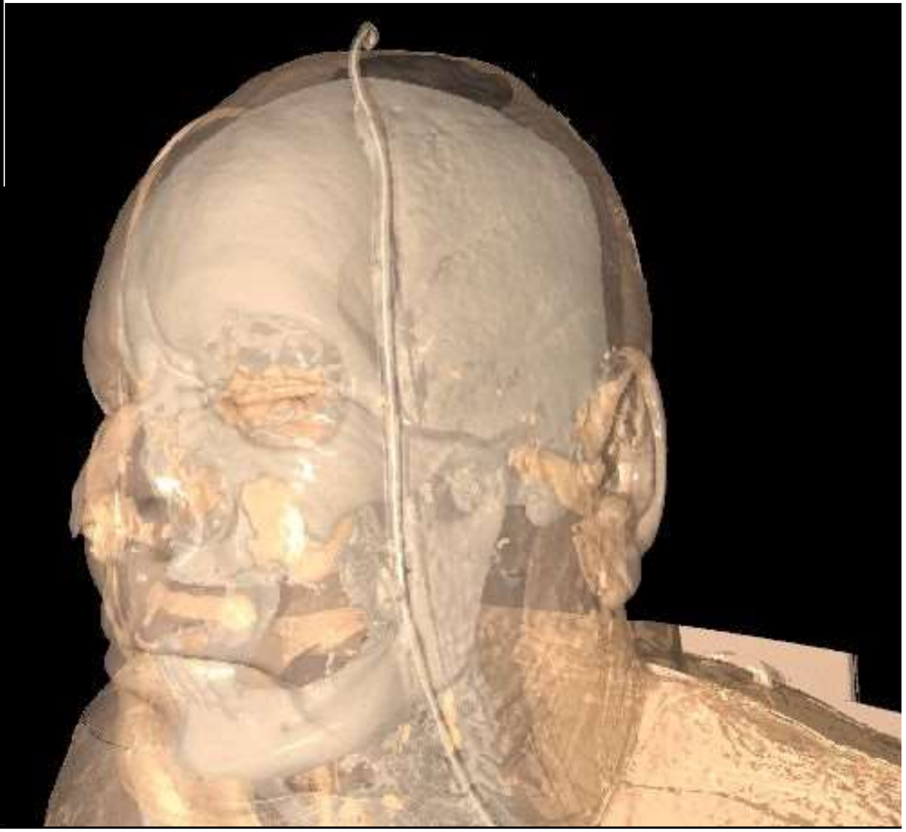
Boeing 777: ~350 million individual polygons, ~30 GB on disk

Volume Visualization

- **Iso-surface rendering**



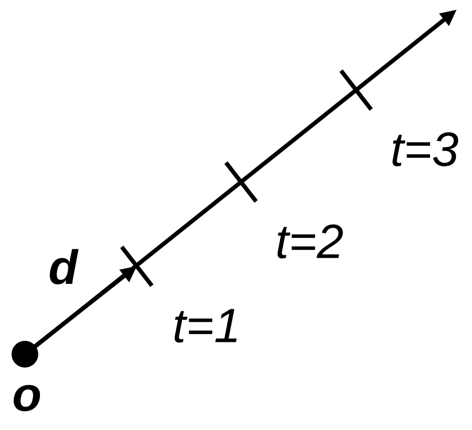




RAY-PRIMITIVE INTERSECTIONS

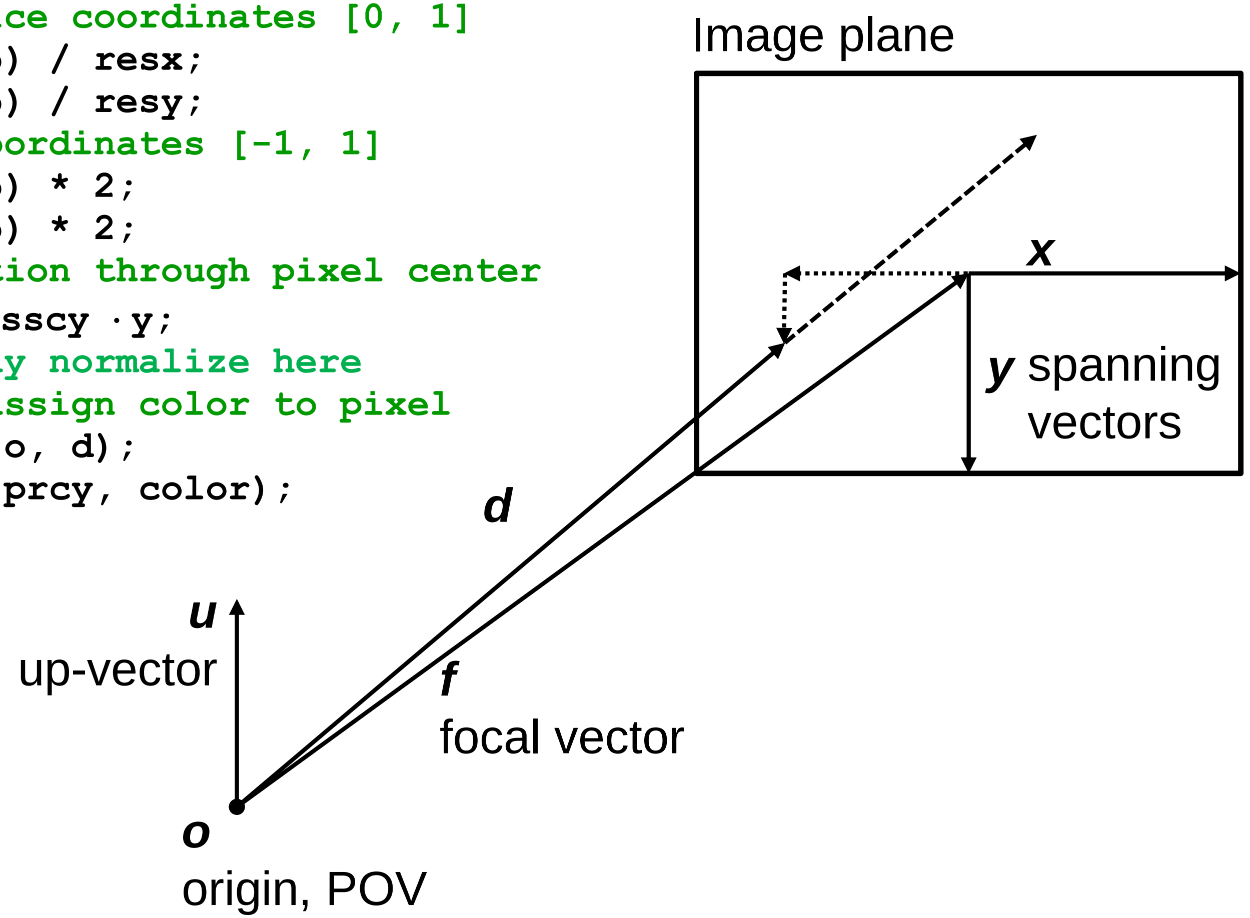
Basic Math - Ray

- **Ray parameterization**
  - $r(t) = \vec{o} + t\vec{d}$,   t $\in \mathbb{R}$; $\vec{o}, \vec{d} \in \mathbb{R}^3$: origin and direction
- **Ray**
  - All points on the graph of $r(t)$, with t $\in \mathbb{R}_{0+}$



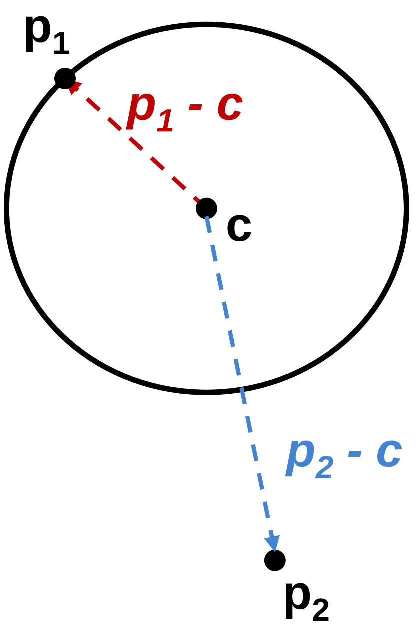
Pinhole Camera Model

```
// For given image resolution {resx, resy}
// Loop over pixel raster coordinates [0, res-1]
for(prcx = 0; prcx < resx; prcx++)
  for(prcy = 0; prcy < resy; prcy++)
  {
    // Normalized device coordinates [0, 1]
    ndcx = (prcx + 0.5) / resx;
    ndcy = (prcy + 0.5) / resy;
    // Screen space coordinates [-1, 1]
    sscx = (ndcx - 0.5) * 2;
    sscy = (ndcy - 0.5) * 2;
    // Generate direction through pixel center
    d = f + sscx ·x + sscy ·y;
    d = d / |d|; // May normalize here
    // Trace ray and assign color to pixel
    color = trace_ray(o, d);
    write_pixel(prcx, prcy, color);
  }
```



Basic Math - Sphere

- **Sphere** S
  - $\vec{c} \in \mathbb{R}^3, r \in \mathbb{R}$: center and radius
  - Sphere: $\forall \vec{p} \in \mathbb{R}^3 : \vec{p} \in S \Leftrightarrow (\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) - r^2 = 0$
    - The distance between the points on the sphere and its center equals the radius

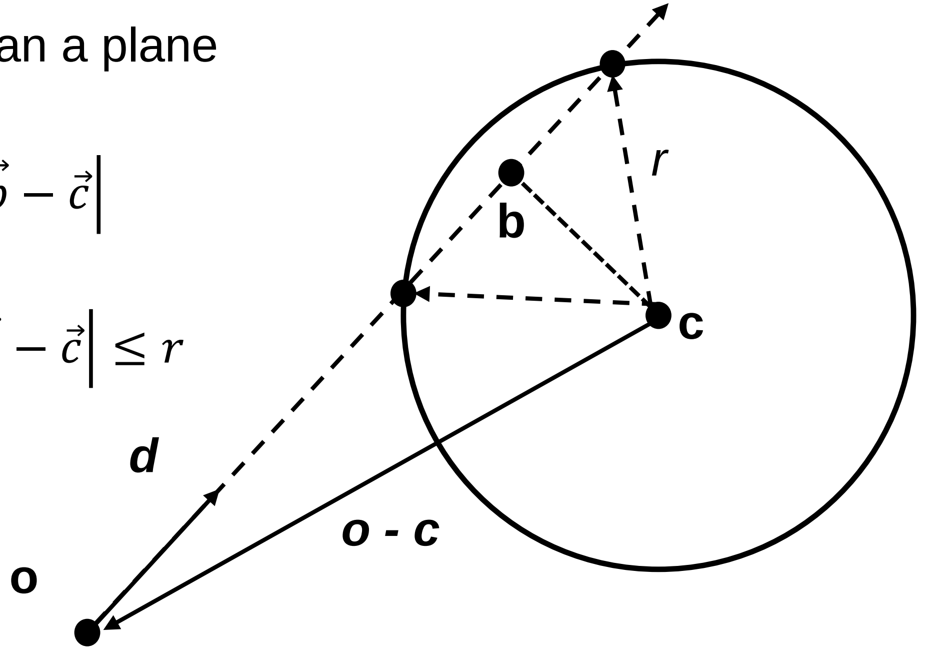


Ray-Sphere Intersection

- **Given**
  - Ray: $r(t) = \vec{o} + t\vec{d}$,   t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Sphere: $\vec{c} \in \mathbb{R}^3, r \in \mathbb{R}$:
    - $\forall \vec{p} \in \mathbb{R}^3 : \vec{p} \in S \Leftrightarrow (\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) - r^2 = 0$
- **Find closest intersection point**
  - Algebraic approach: substitute ray equation
    - $(\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) - r^2 = 0$ with $\vec{p} = \vec{o} + t\vec{d}$
    - $t^2\vec{d} \cdot \vec{d} + 2t\vec{d} \cdot (\vec{o} - \vec{c}) + (\vec{o} - \vec{c}) \cdot (\vec{o} - \vec{c}) - r^2 = 0$
    - Solve for *t*

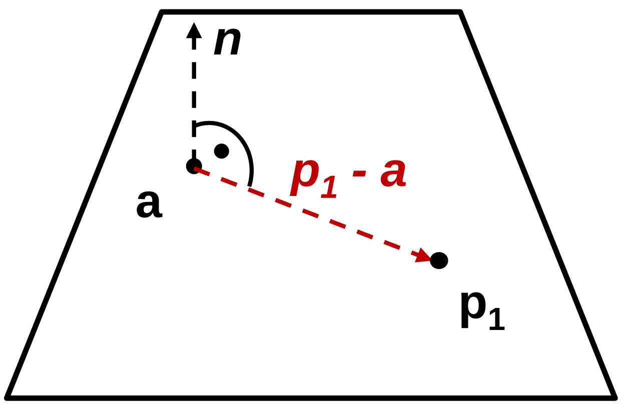
Ray-Sphere Intersection (2)

- **Given**
  - Ray: $r(t) = \vec{o} + t\vec{d}$, t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Sphere: $\vec{c} \in \mathbb{R}^3, r \in \mathbb{R}$:
    - $\forall \vec{p} \in \mathbb{R}^3 : \vec{p} \in S \Leftrightarrow (\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) - r^2 = 0$
- **Find closest intersection point**
  - Geometric approach
    - Ray and center span a plane
    - Solve in 2D
    - Compute $\left|\vec{b} - \vec{o}\right|, \left|\vec{b} - \vec{c}\right|$
      - $\measuredangle OBC = 90°$
    - Intersection(s) if $\left|\vec{b} - \vec{c}\right| \leq r$

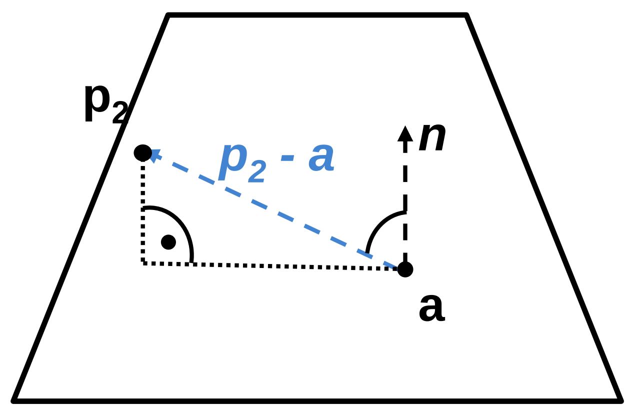


Basic Math - Plane

- **Plane** P
  - $\vec{n}, \vec{a} \in \mathbb{R}^3$: normal and point in P
  - $\forall \vec{p} \in \mathbb{R}^3 : \vec{p} \in P \Leftrightarrow (\vec{p} - \vec{a}) \cdot \vec{n} = 0$
    - The difference vector between any two points on the plane is either 0 or orthogonal to the plane normal



Points in the plane



Points off the plane

Ray-Plane Intersection

- **Given**
  - Ray: $r(t) = \vec{o} + t\vec{d}$, t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Plane: $\vec{n}, \vec{a} \in \mathbb{R}^3$: normal and point in P
- **Compute intersection point**
  - Plane equation: $\vec{p} \in P \Leftrightarrow (\vec{p} - \vec{a}) \cdot \vec{n} = 0$

    $\Leftrightarrow \quad \vec{p} \cdot \vec{n} - D = 0, \; with \;\; D = \vec{a} \cdot \vec{n}$
  - Substitute ray parameterization: $(\vec{o} + t\vec{d}) \cdot \vec{n} - D = 0$
  - Solve for *t*, giving n solutions:
    - ????

Ray-Plane Intersection

- **Given**
  - Ray: $r(t) = \vec{o} + t\vec{d}$, t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Plane: $\vec{n}, \vec{a} \in \mathbb{R}^3$: normal and point in P
- **Compute intersection point**
  - Plane equation: $\vec{p} \in P \Leftrightarrow (\vec{p} - \vec{a}) \cdot \vec{n} = 0$

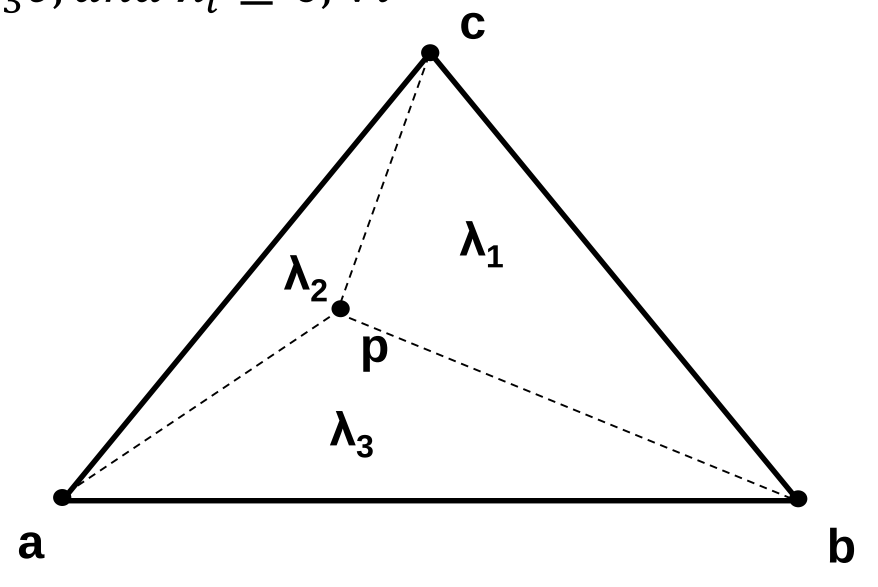
    $\Leftrightarrow \quad \vec{p} \cdot \vec{n} - D = 0, \; with \;\; D = \vec{a} \cdot \vec{n}$
  - Substitute ray parameterization: $(\vec{o} + t\vec{d}) \cdot \vec{n} - D = 0$
  - Solve for *t,* giving n solutions:
    - 0: Ray is parallel to plane
    - 1: Ray intersects the plane
    - ∞: Ray is *in* the plane

Basic Math - Triangle

- **Triangle** T
  - $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$: vertices
  - Affine combinations of $\vec{a}, \vec{b}, \vec{c} \rightarrow$ points defining the triangle plane
    - Non-negative coefficients that sum up to 1 $\rightarrow$ points in the triangle
  - $\vec{p} \in \mathbb{R}^3 : \vec{p} \in T \Leftrightarrow \exists \lambda_{1,2,3} \in \mathbb{R}_{0+},\ \lambda_1 + \lambda_2 + \lambda_3 = 1\ and$

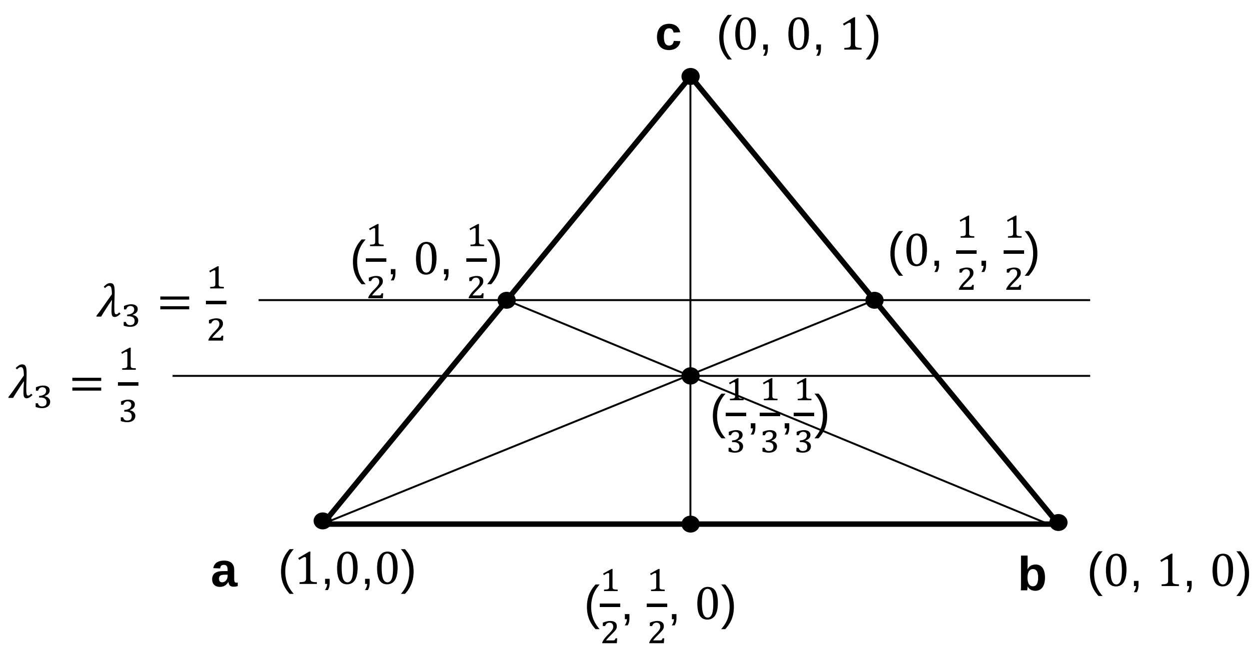
$$\vec{p} = \lambda_1 \vec{a} + \lambda_2 \vec{b} + \lambda_3 \vec{c}, and\ \lambda_i \geq 0, \forall i$$

- **Barycentric Coordinates**
  - $\lambda_{1,2,3}$
  - $\lambda_1 = S_{pbc}/S_{abc}$



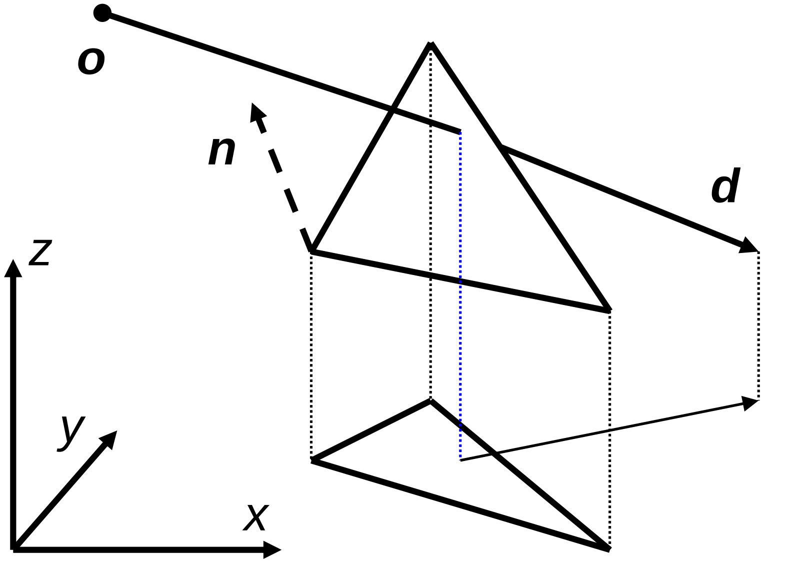
Barycentric Coordinates

- **Triangle** T
  - $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$: vertices
  - $\lambda_{1,2,3}$: barycentric coordinates, area coordinates
  - $\lambda_1 + \lambda_2 + \lambda_3 = 1$
  - $\lambda_1 = S_{pbc}/S_{abc}$



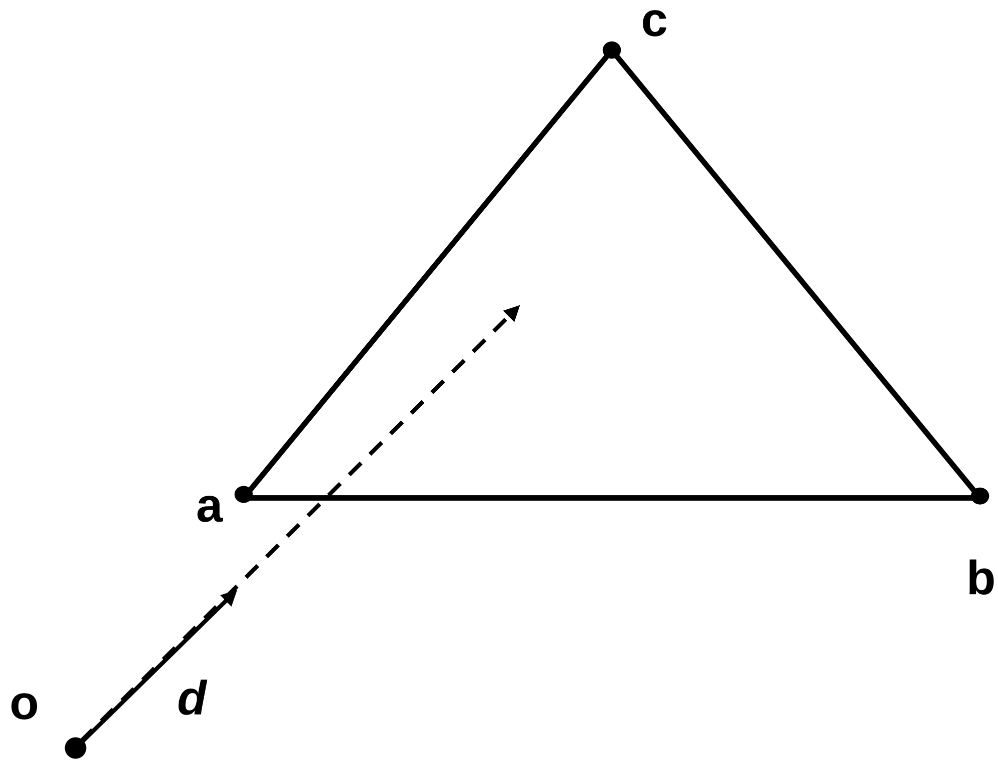
Triangle Intersection: Plane-Based

- **Compute intersection with triangle plane**
- **Project onto a coordinate plane**
  - Use the most aligned coordinate plane
    - xy: if n_z is maximal, etc.
  - Coordinate plane and 2D vertices can be pre-computed
- **Compute barycentric coordinates**
  - Signed areas of subtriangles
  - Test for positive BCs



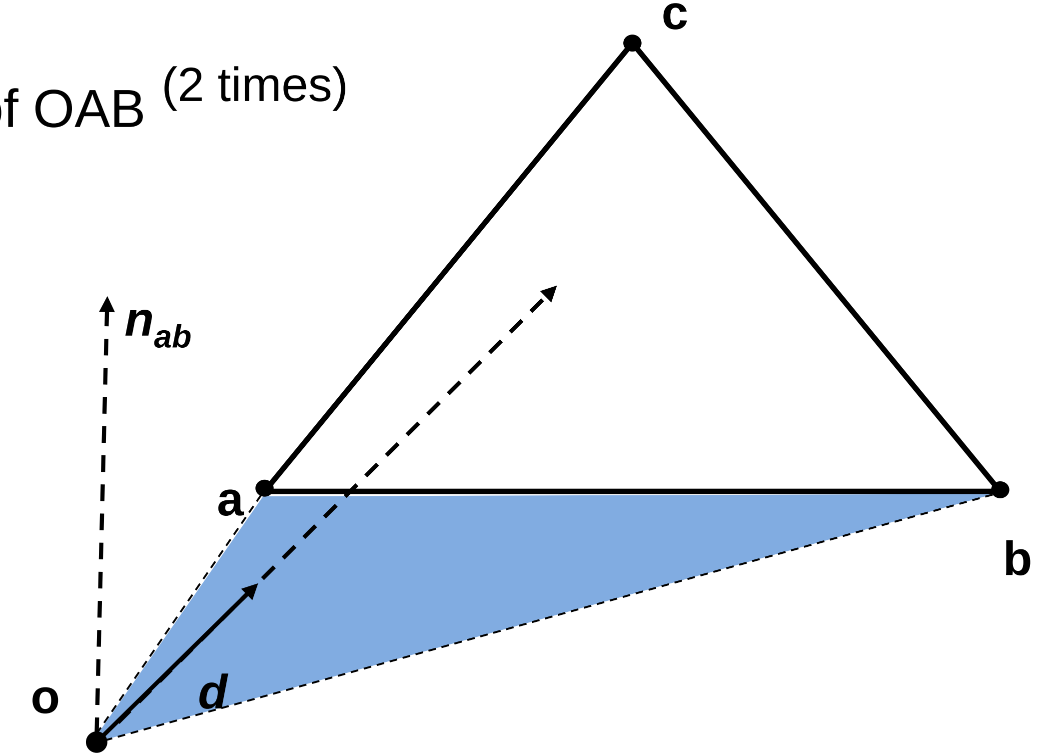
Triangle Intersection Edge-Based (1)

- **3D Linear Function across triangle**
  - Ray: $\vec{o} + t\vec{d}$,   t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$



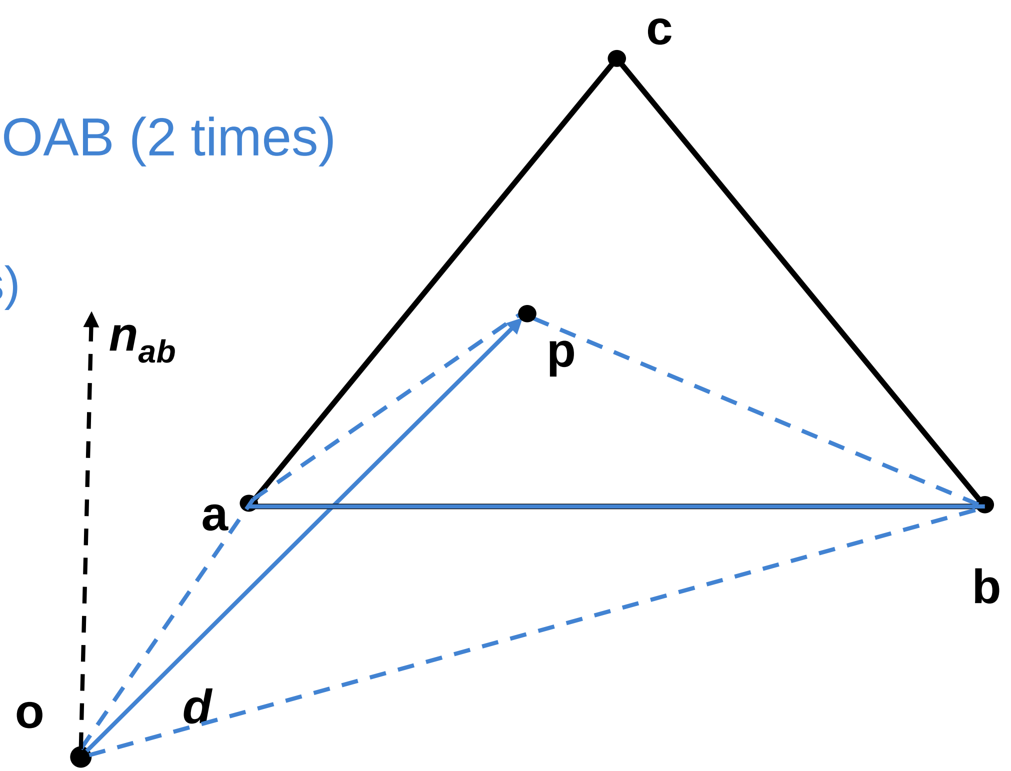
Triangle Intersection Edge-Based (2)

- **3D Linear Function across triangle**
  - Ray: $\vec{o} + t\vec{d}$, t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$
  - $\overrightarrow{n_{ab}} = \left(\vec{b} - \vec{o}\right) \times (\vec{a} - \vec{o})$
  - $|\overrightarrow{n_{ab}}|$ is the signed area of OAB (2 times)



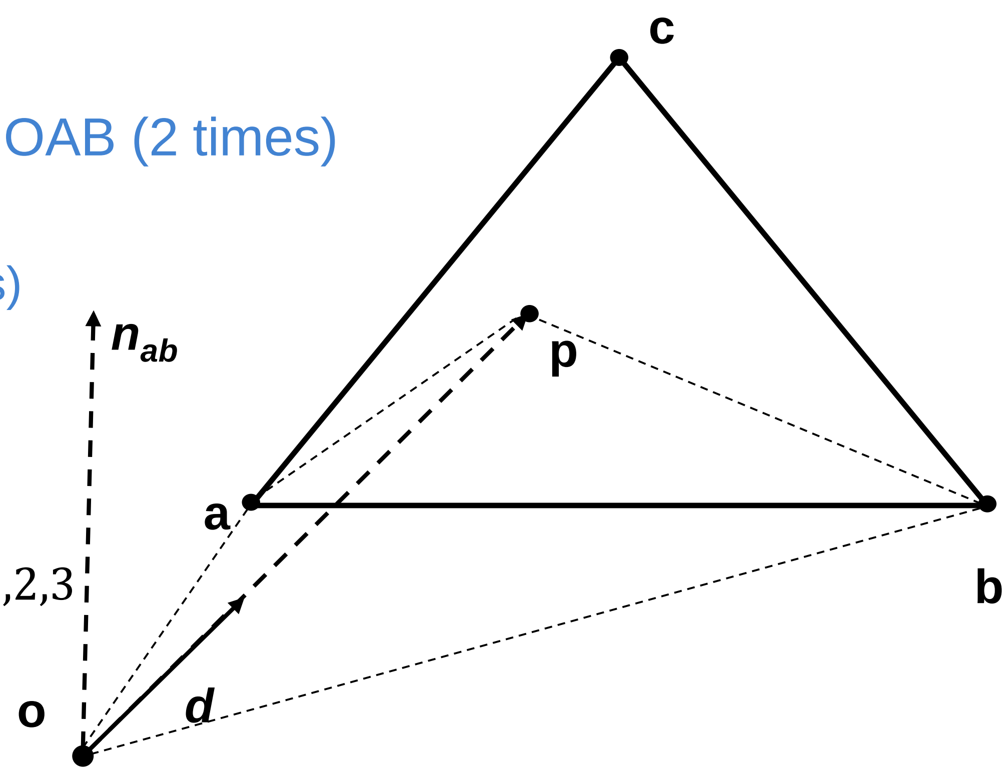
Triangle Intersection Edge-Based (3)

- **3D Linear Function across triangle**
  - Ray: $\vec{o} + t\vec{d}$, t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$
  - $\overrightarrow{n_{ab}} = \left(\vec{b} - \vec{o}\right) \times (\overrightarrow{a} - \vec{o})$
  - $|\overrightarrow{n_{ab}}|$ is the signed area of OAB (2 times)
  - $\lambda_3^*(t) = \overrightarrow{n_{ab}} \cdot t\vec{d}$
    - Volume of OABP (6 times)
    - For $t = t_{hit}$



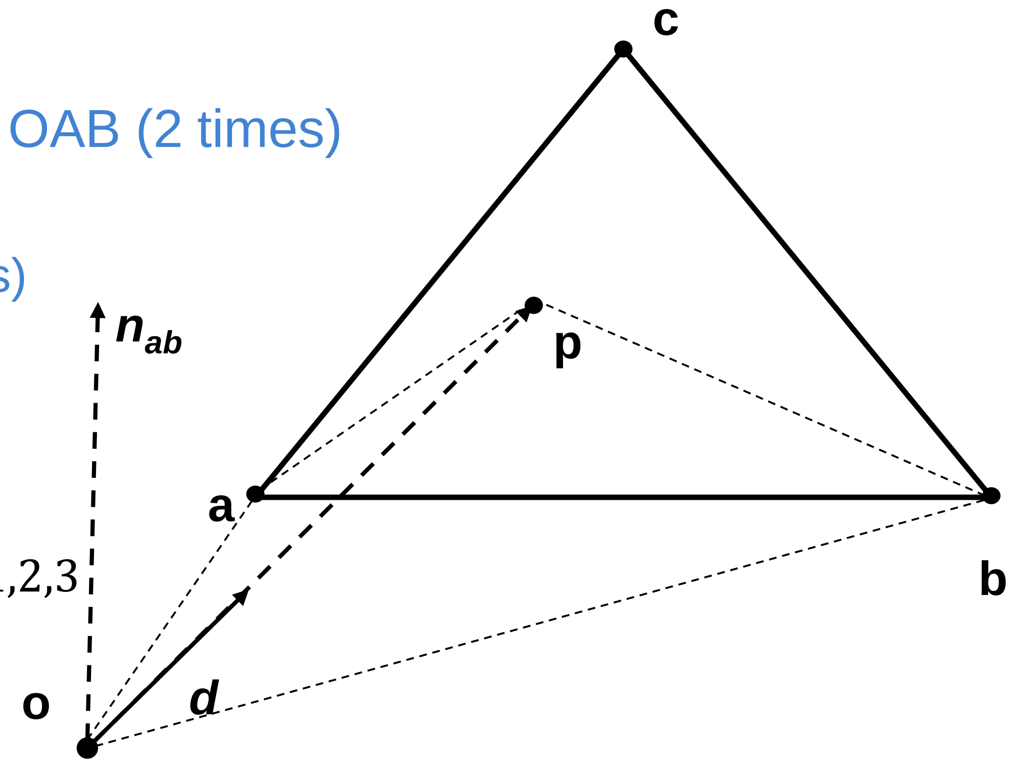
Triangle Intersection Edge-Based (4)

- **3D Linear Function across triangle**
  - Ray: $\vec{o} + t\vec{d}$, $\quad$ t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$
  - $\overrightarrow{n_{ab}} = \left(\vec{b} - \vec{o}\right) \times (\vec{a} - \vec{o})$
  - $|\overrightarrow{n_{ab}}|$ is the signed area of OAB (2 times)
  - $\lambda_3^*(t) = \overrightarrow{n_{ab}} \cdot t\vec{d}$
    - Volume of OABP (6 times)
    - For $t = t_{hit}$
  - $\lambda_{1,2}^*(t) = \overrightarrow{n_{bc,ac}} \cdot t\vec{d}$
  - Normalize
    - $\lambda_i = \frac{\lambda_i^*(t)}{\lambda_1^*(t)+\lambda_2^*(t)+\lambda_3^*(t)}, i = 1,2,3$
    - Length of $t\vec{d}$ cancels out



Triangle Intersection Edge-Based (5)

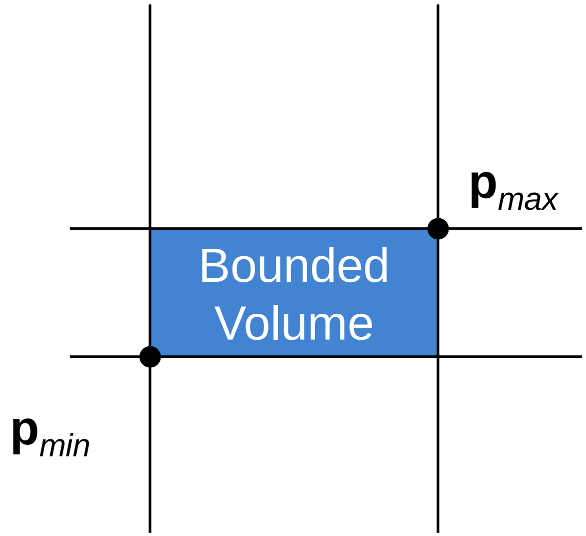
- **3D Linear Function across triangle**
  - Ray: $\vec{o} + t\vec{d}$, t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$
  - $\overrightarrow{n_{ab}} = \left(\vec{b} - \vec{o}\right) \times (\overrightarrow{a} - \vec{o})$
  - $|\overrightarrow{n_{ab}}|$ is the signed area of OAB (2 times)
  - $\lambda_3^*(t) = \overrightarrow{n_{ab}} \cdot t\vec{d}$
    - Volume of OABP (6 times)
    - For $t = t_{hit}$
  - $\lambda_{1,2}^*(t) = \overrightarrow{n_{bc,ac}} \cdot t\vec{d}$
  - Normalize
    - $\lambda_i = \frac{\lambda_i^*(t)}{\lambda_1^*(t)+\lambda_2^*(t)+\lambda_3^*(t)}, i = 1,2,3$



- **For positive BCs**
  - Compute $\vec{p} = \lambda_1\vec{a} + \lambda_2\vec{b} + \lambda_3\vec{c}$

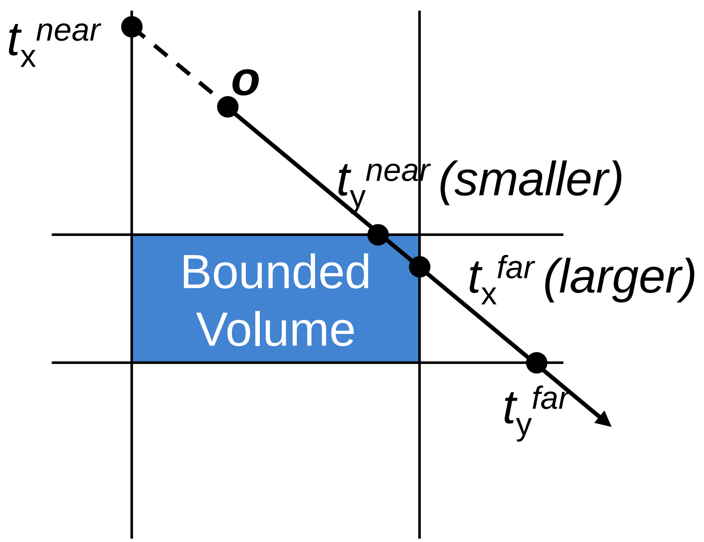
Axis Aligned Bounding Box

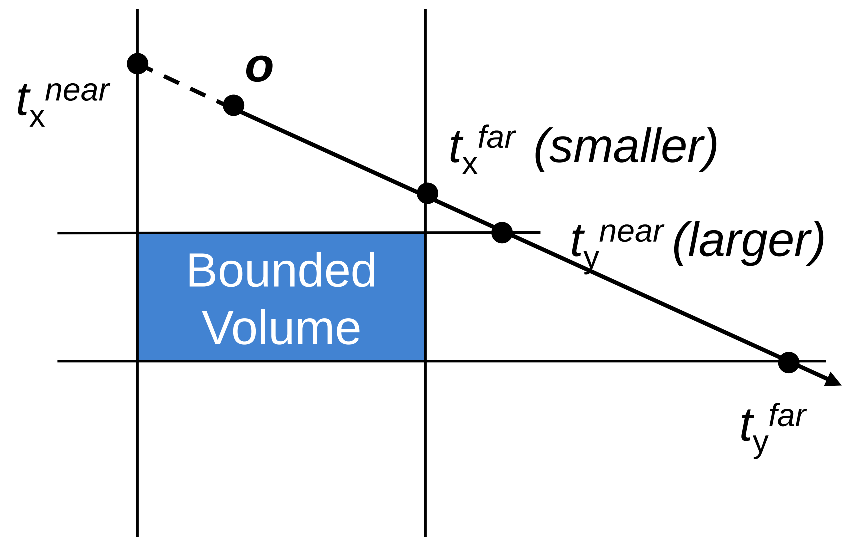
- **Given**
  - Ray: $\vec{o} + t\vec{d}$, t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Axis aligned bounding box (AABB): $\overrightarrow{p_{min}}, \overrightarrow{p_{max}} \in \mathbb{R}^3$



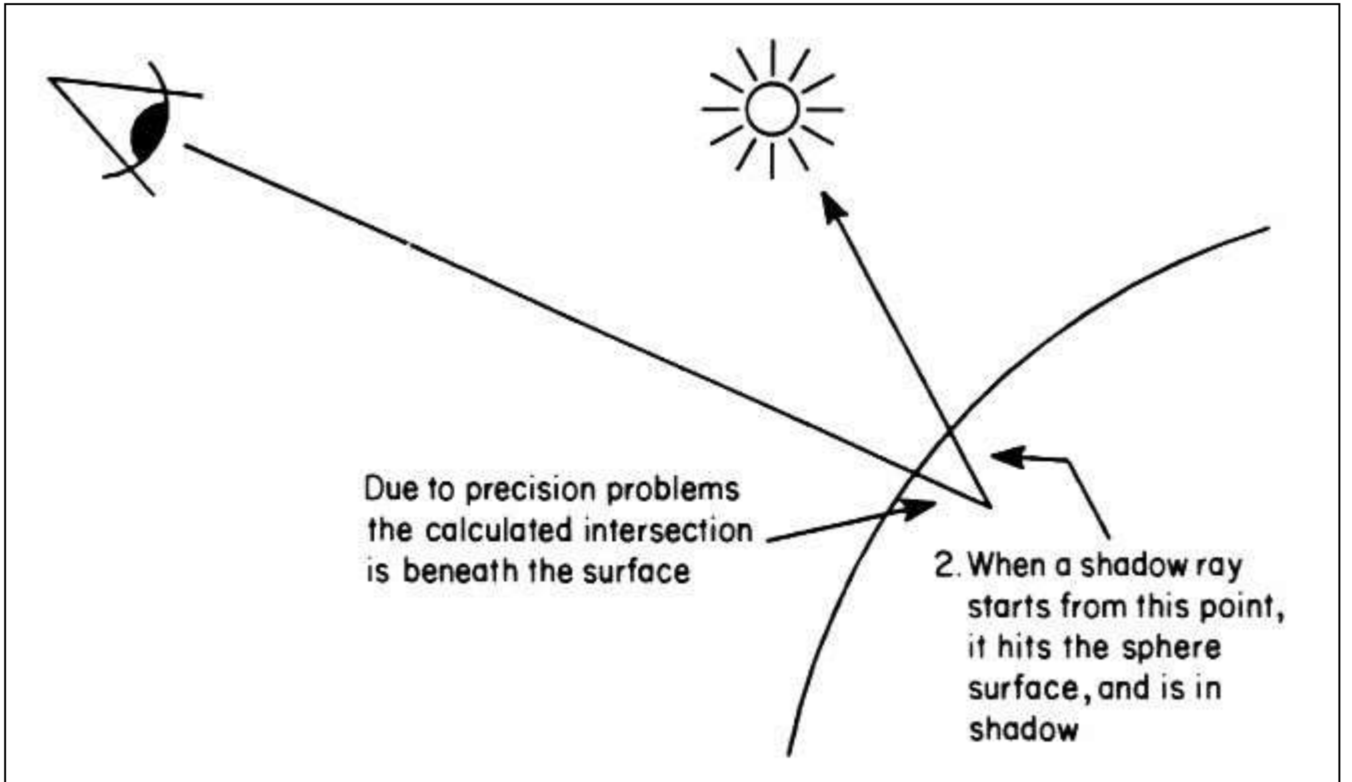
Ray-Box Intersection

- **Given**
  - Ray: $\vec{o} + t\vec{d}$, t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Axis aligned bounding box (AABB): $\overrightarrow{p_{min}}, \overrightarrow{p_{max}} \in \mathbb{R}^3$
- **"Slabs test" for ray-box intersection**
  - Ray enters the box in all dimensions before exiting in any
  - $\max(\{t_i^{near} | i = x, y, z\}) < \min(\{t_i^{far} | i = x, y, z\})$





Precision Problems



Problem in surface intersection.

History of Intersection Algorithms

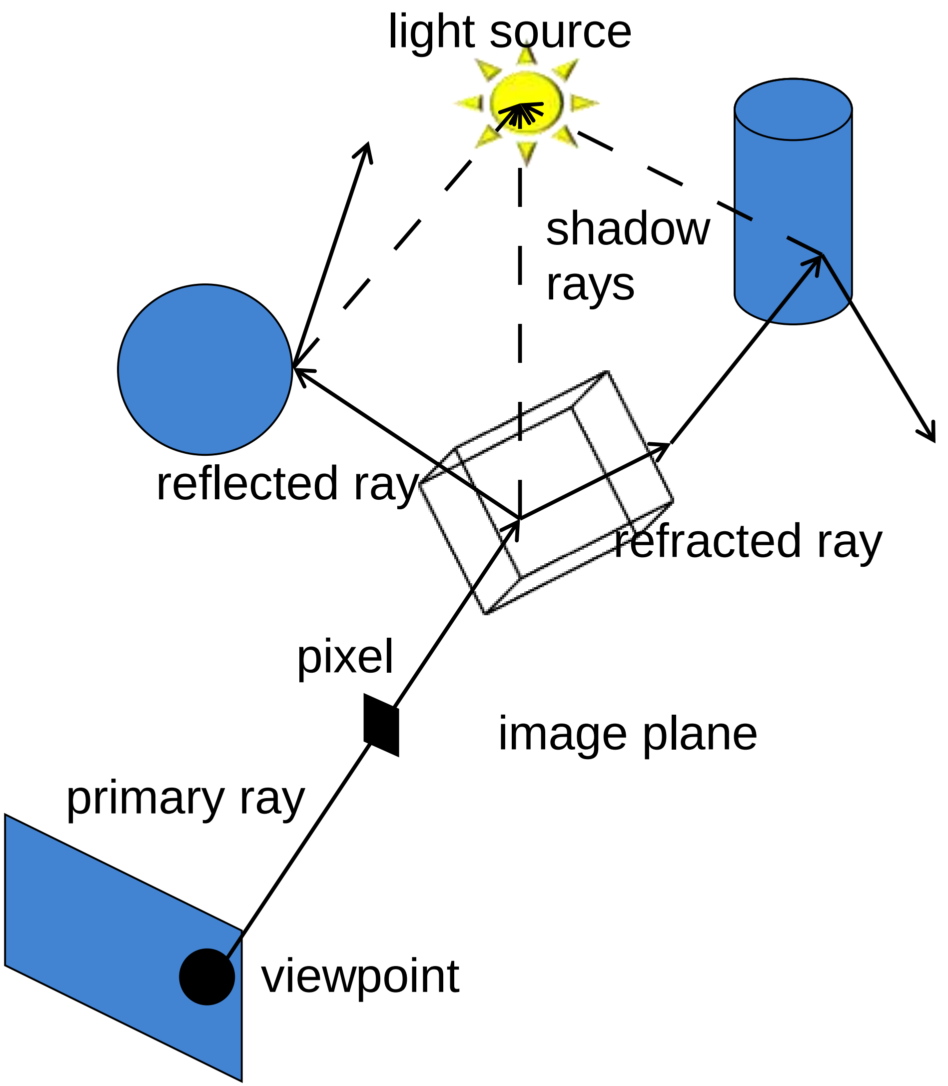
- **Ray-geometry intersection algorithms**
  - Polygons: [Appel ’68]
  - Quadrics, CSG: [Goldstein & Nagel ’71]
  - Recursive Ray Tracing: [Whitted ’79]
  - Tori: [Roth ’82]
  - Bicubic patches: [Whitted ’80, Kajiya ’82]
  - Algebraic surfaces: [Hanrahan ’82]
  - Swept surfaces: [Kajiya ’83, van Wijk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

Shading

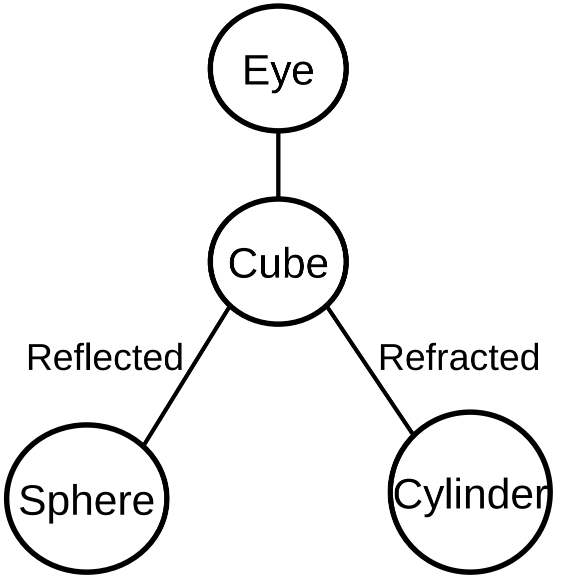
- **Intersection point determines primary ray's "color"**
- **Diffuse object: Color at point and incoming light**
  - No variation with viewing angle: diffuse (Lambertian)
- **Perfect reflection/refraction (mirror, glass)**
  - Only one outgoing direction → Trace one secondary ray
- **Non-Lambertian Reflectance**
  - Appearance depends on illumination and viewing direction
  - Local Bi-directional Reflectance Distribution Function (BRDF, later)
- **Area light sources**
  - Approximate with multiple samples / shadow rays
- **Indirect illumination**
  - See course in next semester (RIS)

- **More details later**

Recursive Ray Tracing



- **Searching recursively for paths to light sources**
  - Interaction of light & material at intersections
  - Recursively trace new rays in reflection, refraction, and light direction



Ray Tracing Algorithm

- **Trace(ray)**
  - Search the next intersection point (hit, material)
  - Return Shade(ray, hit, material)
- **Shade(ray, hit, material)**
  - For each light source
    - if ShadowTrace(ray to light source, distance to light)
      - Calculate reflected light/radiance (i.e. Phong material model)
      - Adding to the reflected radiance
  - If mirroring material
    - Calculate radiance in reflected direction: Trace(R(ray, hit))
    - Adding mirrored light to the reflected radiance
  - Same for transmission
  - Return reflected radiance
- **ShadowTrace(ray, dist)**
  - Return false, if intersection with distance < dist has been found
  - Can be changed to handle transparent objects as well
    - But not with refraction

Ray Tracing Features

- **Ray Tracing incorporates into a single framework**
  - Hidden surface removal
    - Front to back traversal
    - Early termination once first hit point is found
  - Shadow computation
    - Shadow rays/ shadow feelers 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

Common Approximations

- **Usually RGB color model instead of full spectrum**
- **Finite # 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
- **Used reflection models are often empirical**
  - Better physically accurate models are available (e.g. Ward model)

Ray Casting Outside CG

- **Tracing/Casting a ray**
  - Type of query
    - “Is there a primitive along a ray”
    - “How far is the closest primitive”
- **Other uses than rendering**
  - Volume computation
  - Sound waves tracing
  - Collision detection
  - …