Computer Graphics

- Introduction to Ray Tracing -

Philipp Slusallek

Rendering Algorithms

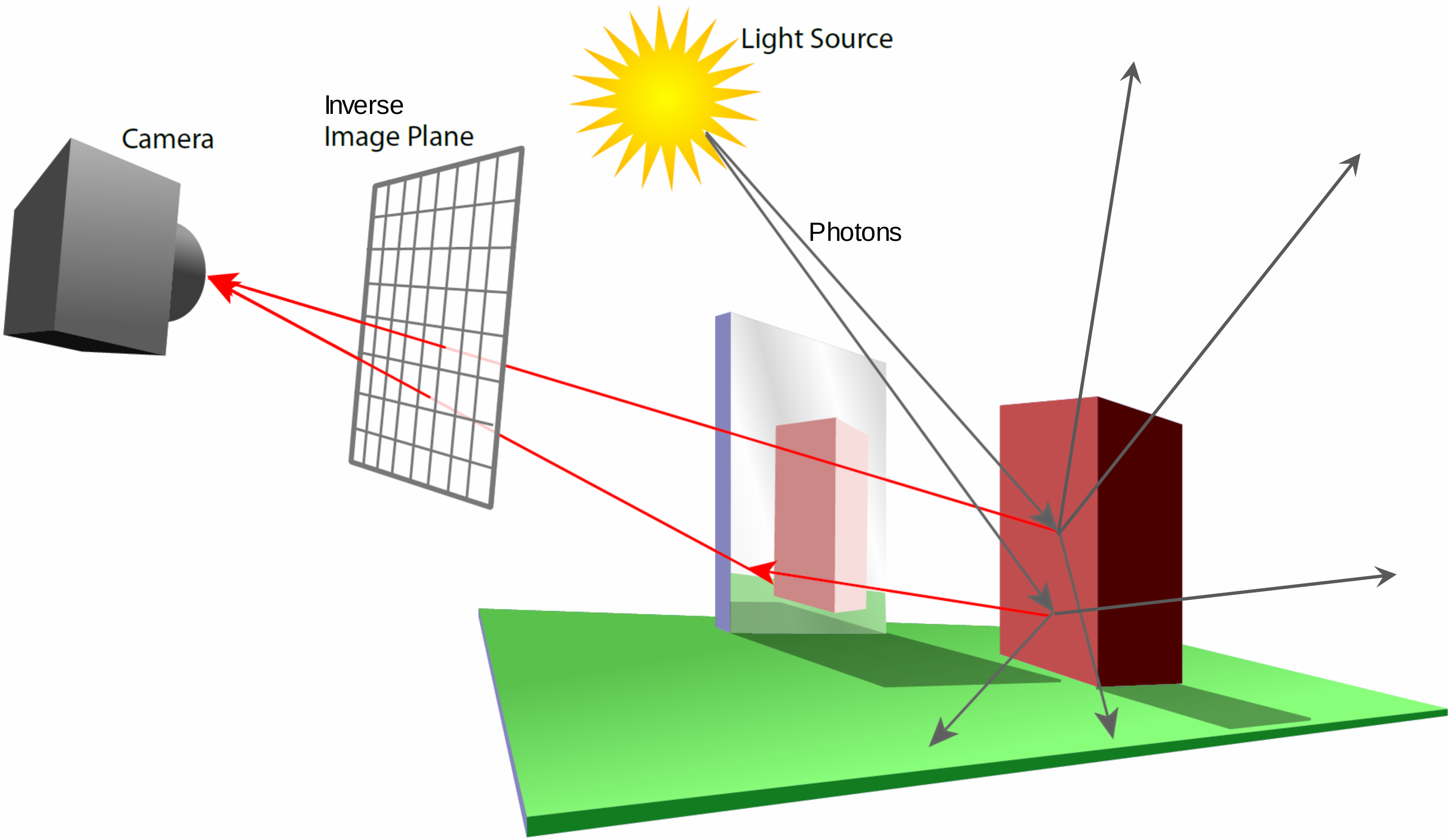
- **Rendering**
  - Definition: Given a 3D scene description as input and a camera, generate a 2D image as a view from the camera of the 3D scene
- **Algorithms**
  - Ray Tracing
    - Declarative scene description
    - Physically-based simulation of light transport
    - Throughout the scene from light sources to the camera
  - Rasterization
    - Traditional procedural/imperative drawing of scene content
      - One triangle at a time (conceptually)
    - See later in the course!

Scene Description in General

- **Surface Geometry**
  - 3D geometry of objects in a scene
  - Geometric primitives – triangles, polygons, spheres, …
- **Surface Appearance**
  - Color, texture, absorption, reflection, refraction, subsurface scattering
  - Types of materials: Diffuse, glossy, mirror, glass, …
- **Illumination**
  - Position and emission characteristics of light sources
  - Note: Light also reflects off of surfaces!
    - Secondary/indirect/global illumination
  - Assumption: air/empty space is totally transparent
    - Simplification that excludes scattering effects in *participating media* or *volumes,* e.g. smoke, solid object (CT scan), …
    - See later in course
- **Camera**
  - View point, viewing direction, field of view, resolution, …

OVERVIEW OF RAY-TRACING

Light Transport (1)

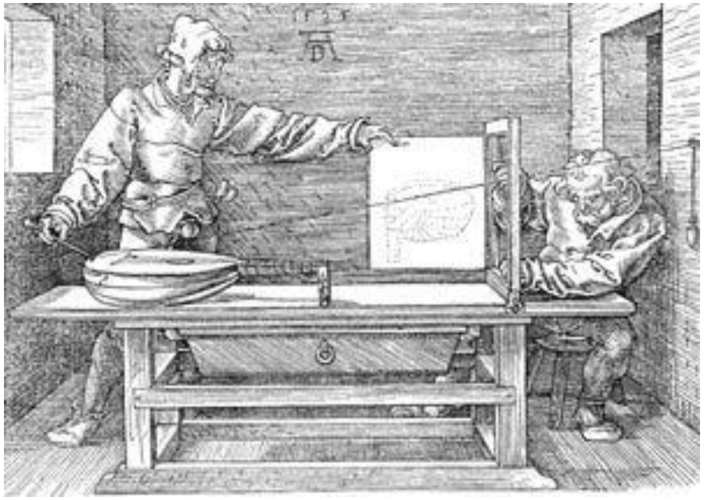


Light Transport (2)

- **Light Distribution in a Scene**
  - Dynamic equilibrium: As much light is absorbed as is emitted
- **Forward Light Transport**
  - Shoot photons from the light sources into scene
  - Scatter 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 or Light Tracing
- **Backward Light Transport**
  - Start at the detector (camera)
  - Trace only paths that might transport light towards camera
    - May be hard to find and connect to light sources
  - Ray Tracing

Ray Tracing Is…

- **Fundamental rendering algorithm**



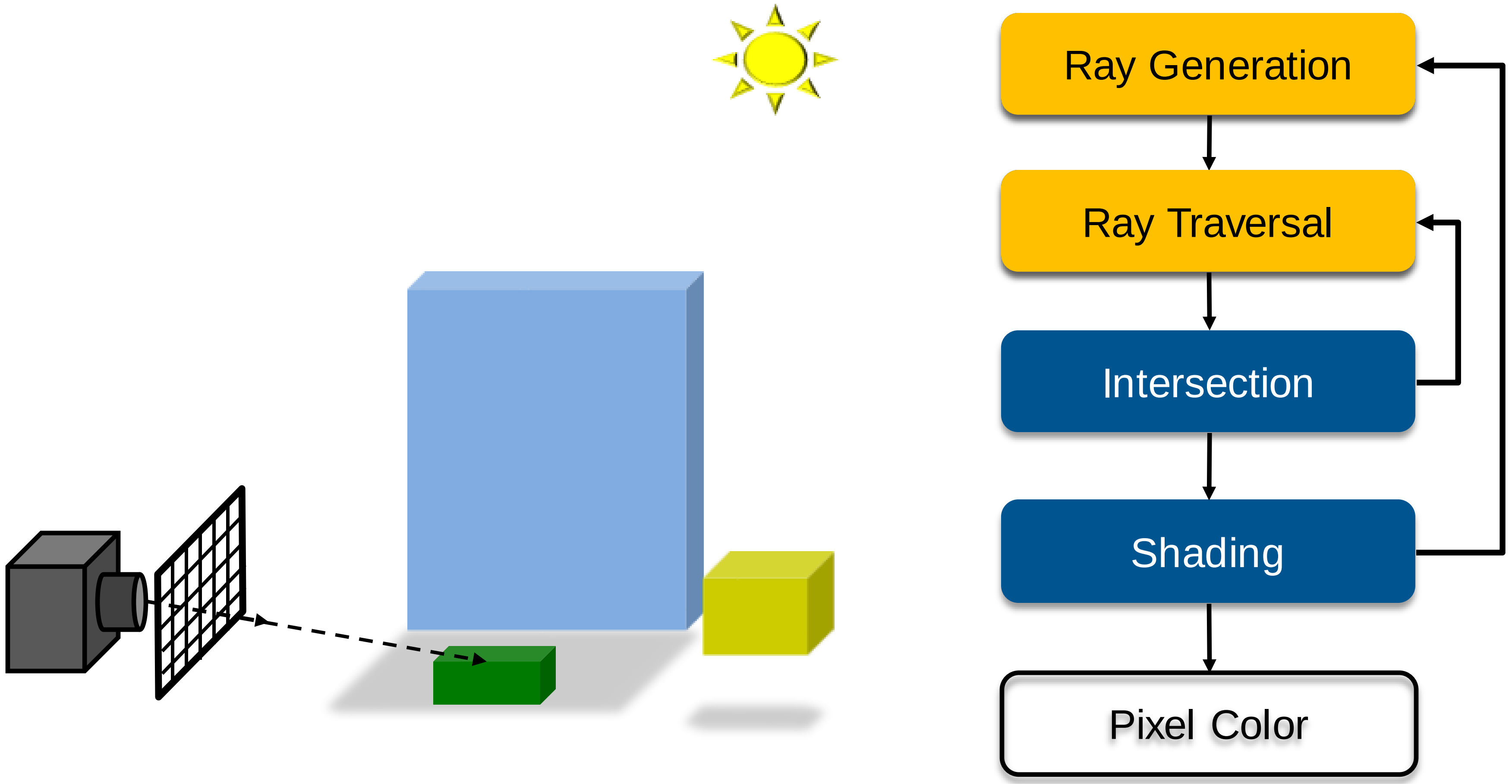
Perspective Machine, Albrecht Dürer

- **Automatic, simple and intuitive**
  - Easy to understand and implement
  - Delivers “correct“ images by default
- **Powerful and efficient**
  - Covers many optical global effects
    - Shadows, reflections, refractions, …
  - Efficient real-time implementation in SW – and now also in HW!
  - Can work in parallel and distributed environments
  - Logarithmic scalability with scene size: O(log n) vs. O(n)
  - Output sensitive and demand-driven approach
- **Concept of light rays is not new**
  - Empedocles (492-432 BC), Renaissance (Dürer, 1525), …
  - Used in lens design, geometric optics, neutron transport, …

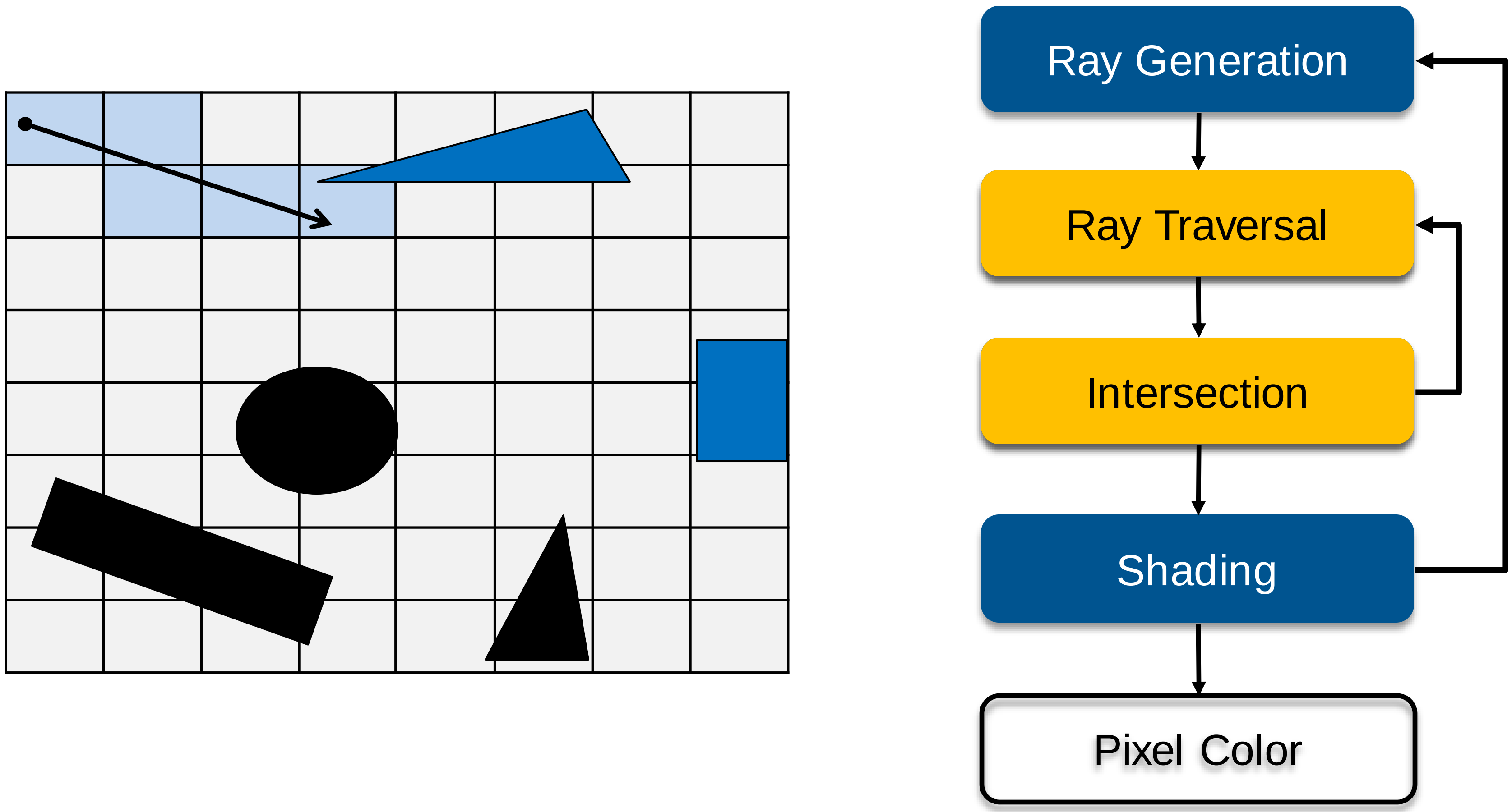
Fundamental Ray Tracing Steps

- **Generation of primary rays**
  - Rays from viewpoint along viewing directions into 3D scene
  - (At least) one ray per picture element (pixel) in image plane
- **Ray *casting***
  - Traversal of spatial index structures (acceleration structures)
    - For avoiding costly but unnecessary intersection computations
  - Ray-primitive intersection → hit point
- **Shading the hit point**
  - Compute light towards camera → pixel color
    - Light power (really “radiance”) travelling along primary ray
  - Needed for computation
    - Local reflection/scattering properties: material color, texture, …
    - Local illumination at intersection point
      - Can be hard to determine correctly (light could come from anywhere)
      - Simple: Test direct connection to lights (“shadow rays”)
      - Compute transparency/mirror effects through *recursive tracing of rays*

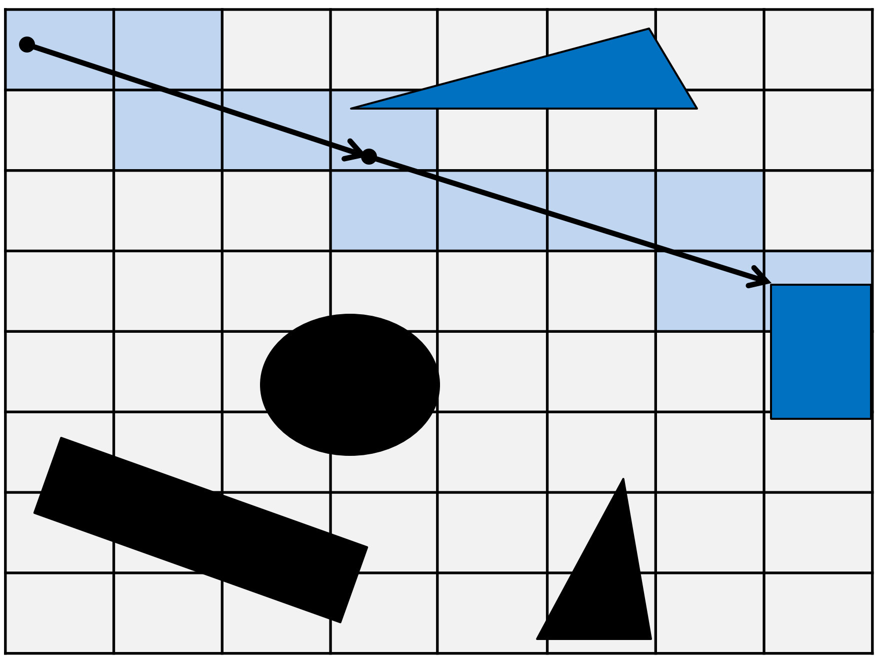
Ray Tracing Pipeline (1)

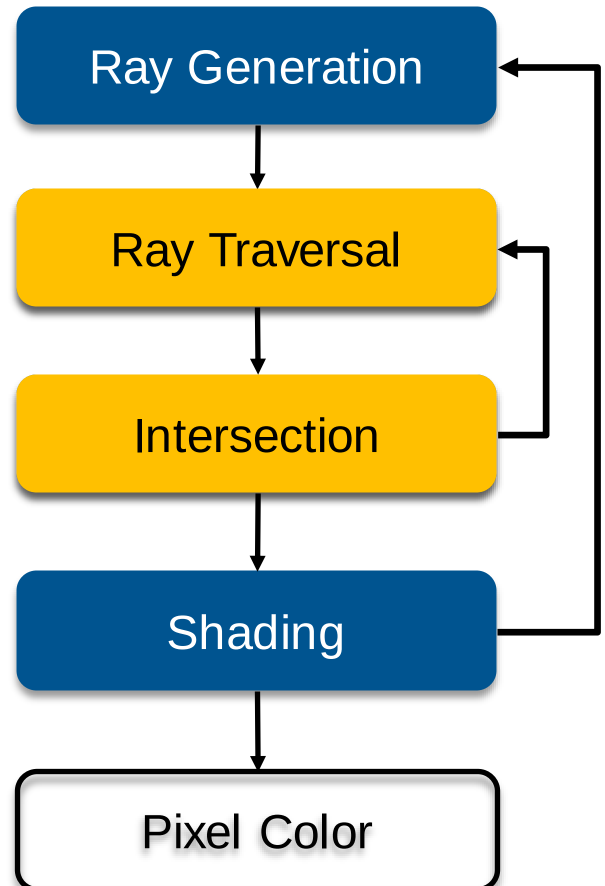


Ray Tracing Pipeline (2)

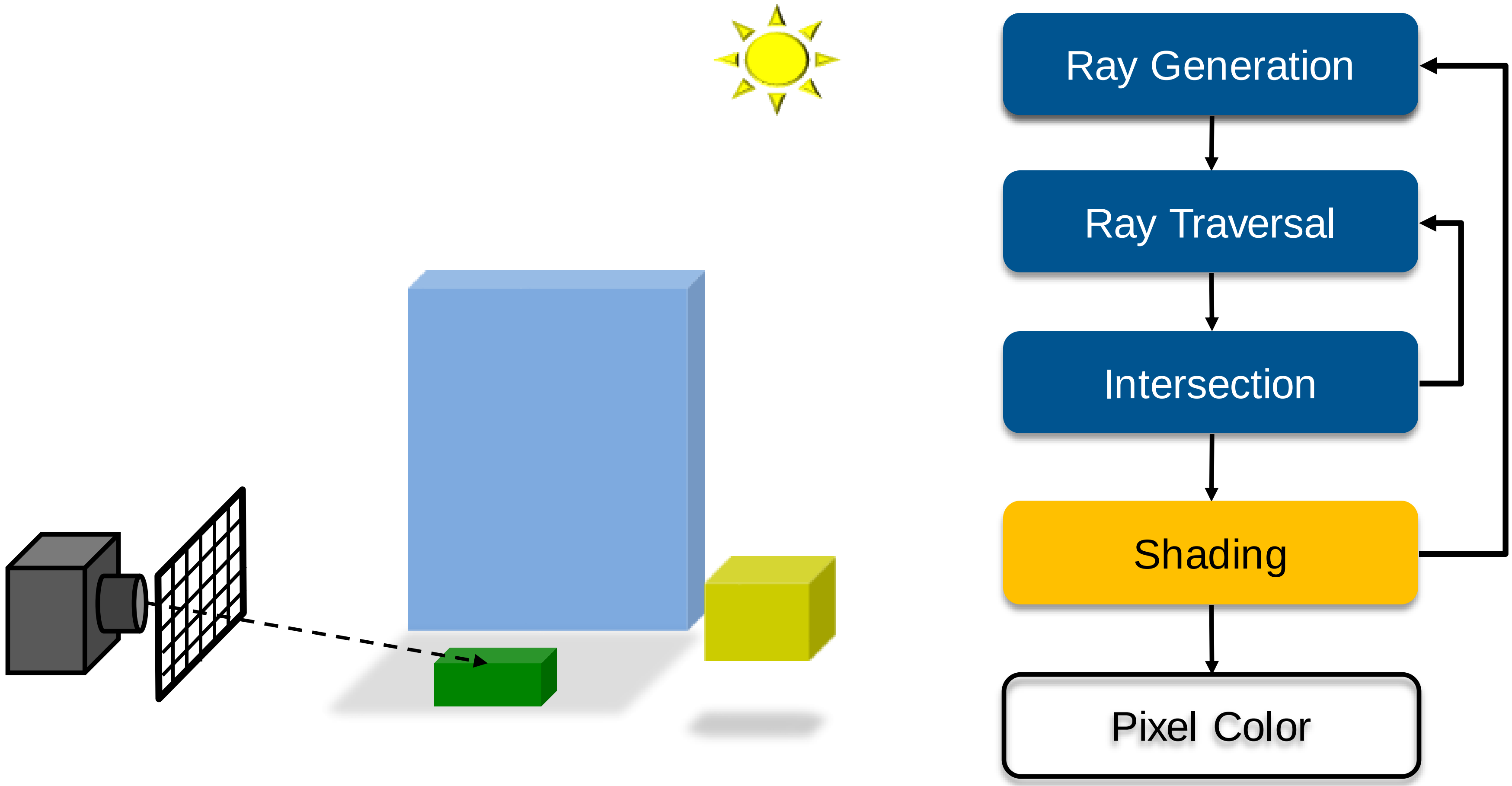


Ray Tracing Pipeline (3)

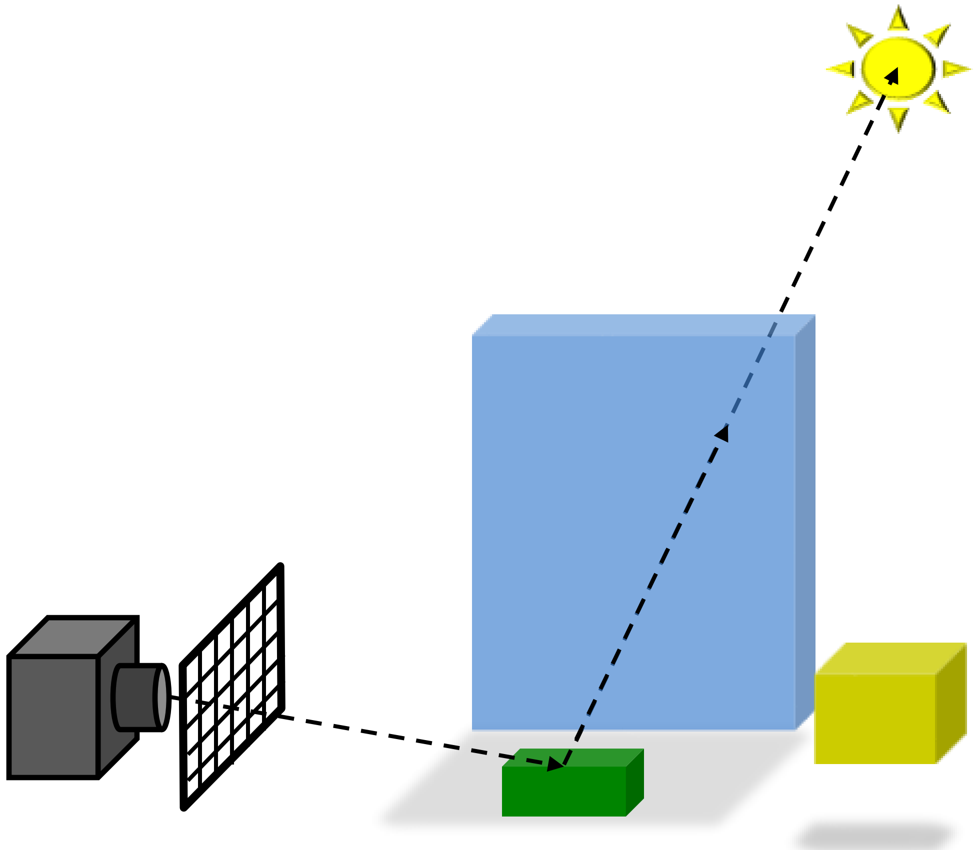


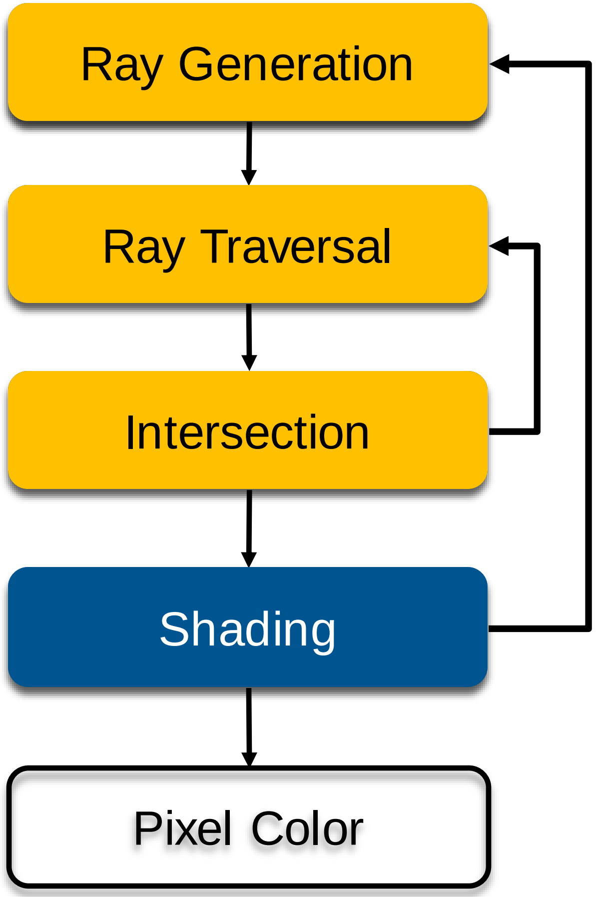


Ray Tracing Pipeline (4)

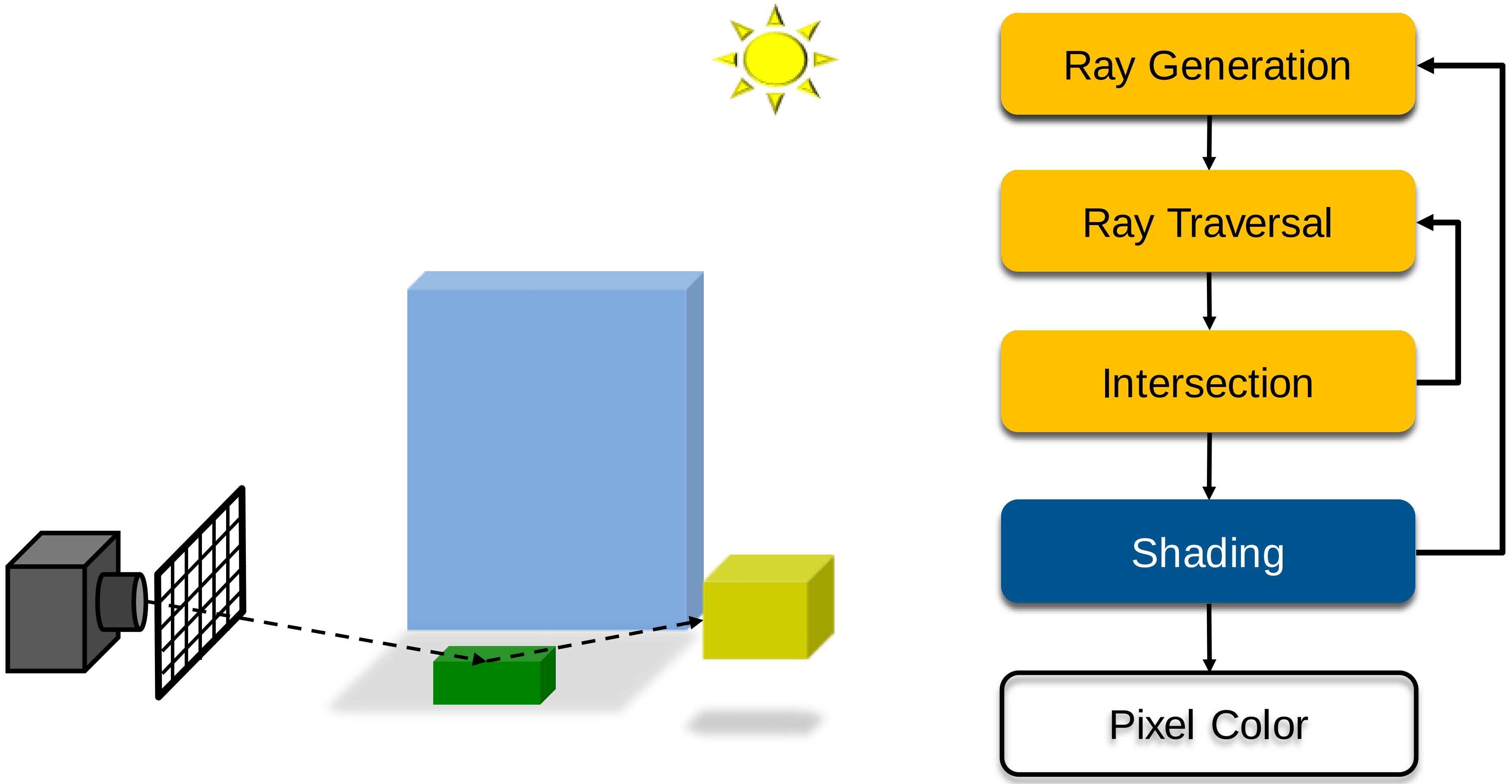


Ray Tracing Pipeline (5)

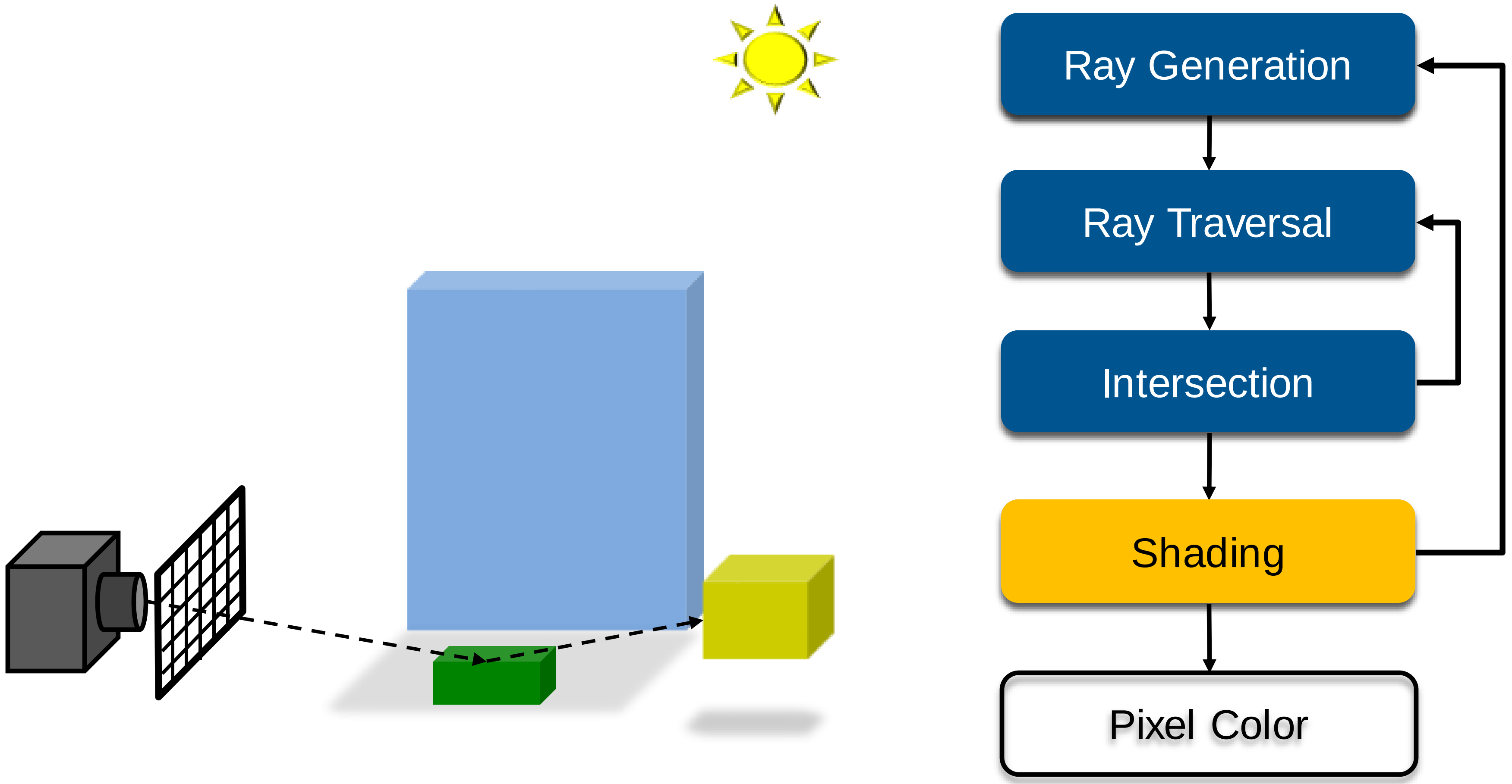




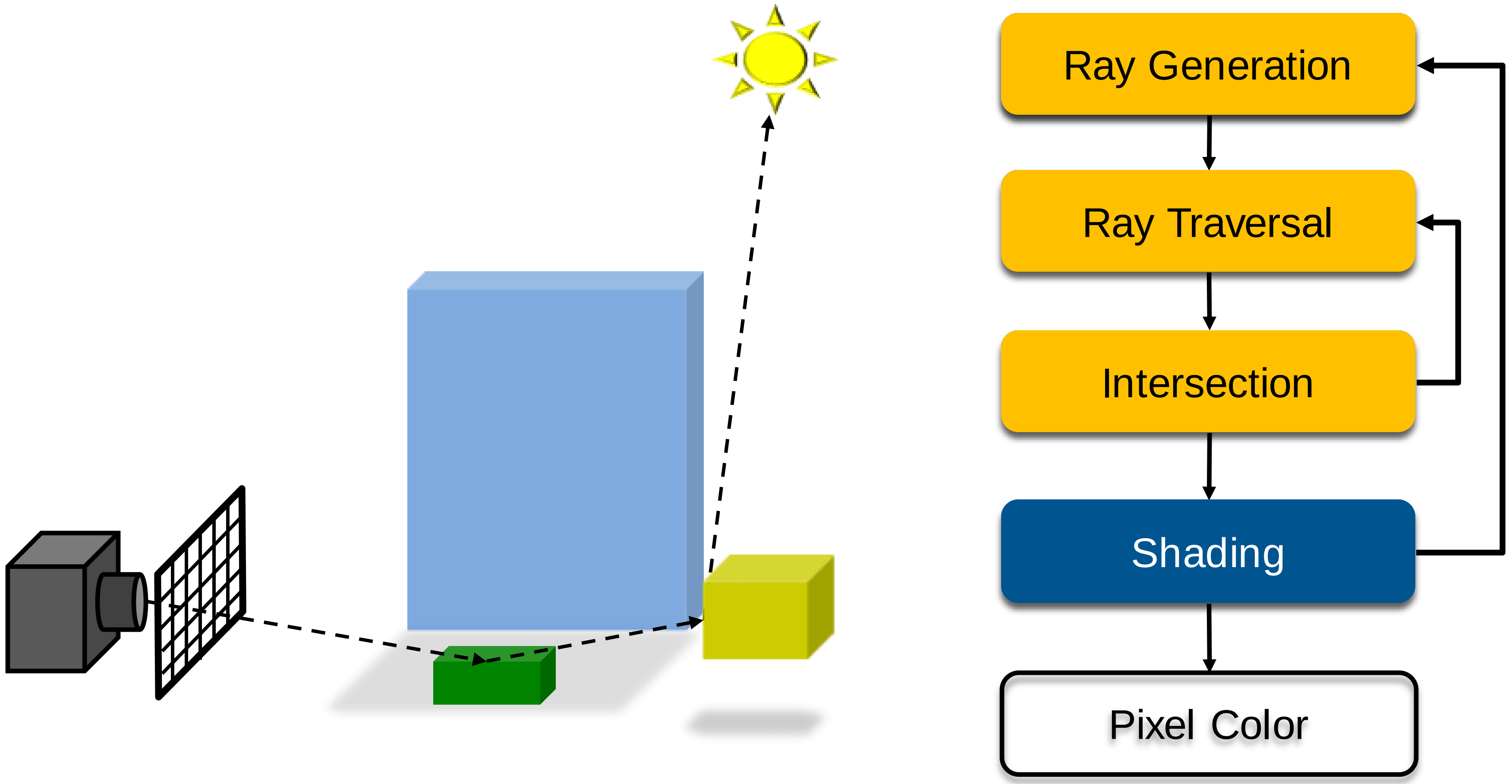
Recursive Ray Tracing Pipeline (6)



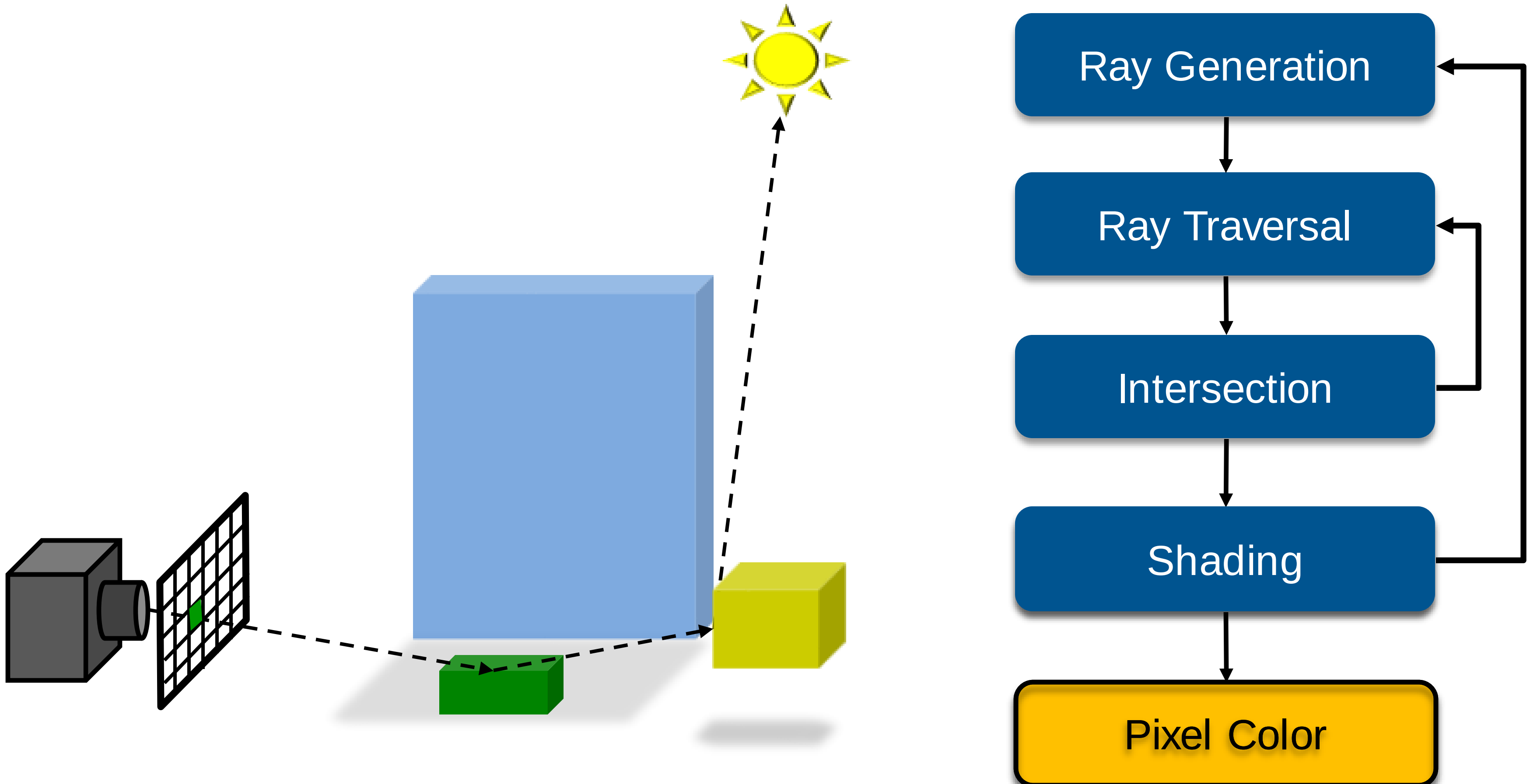
Recursive Ray Tracing Pipeline (7)



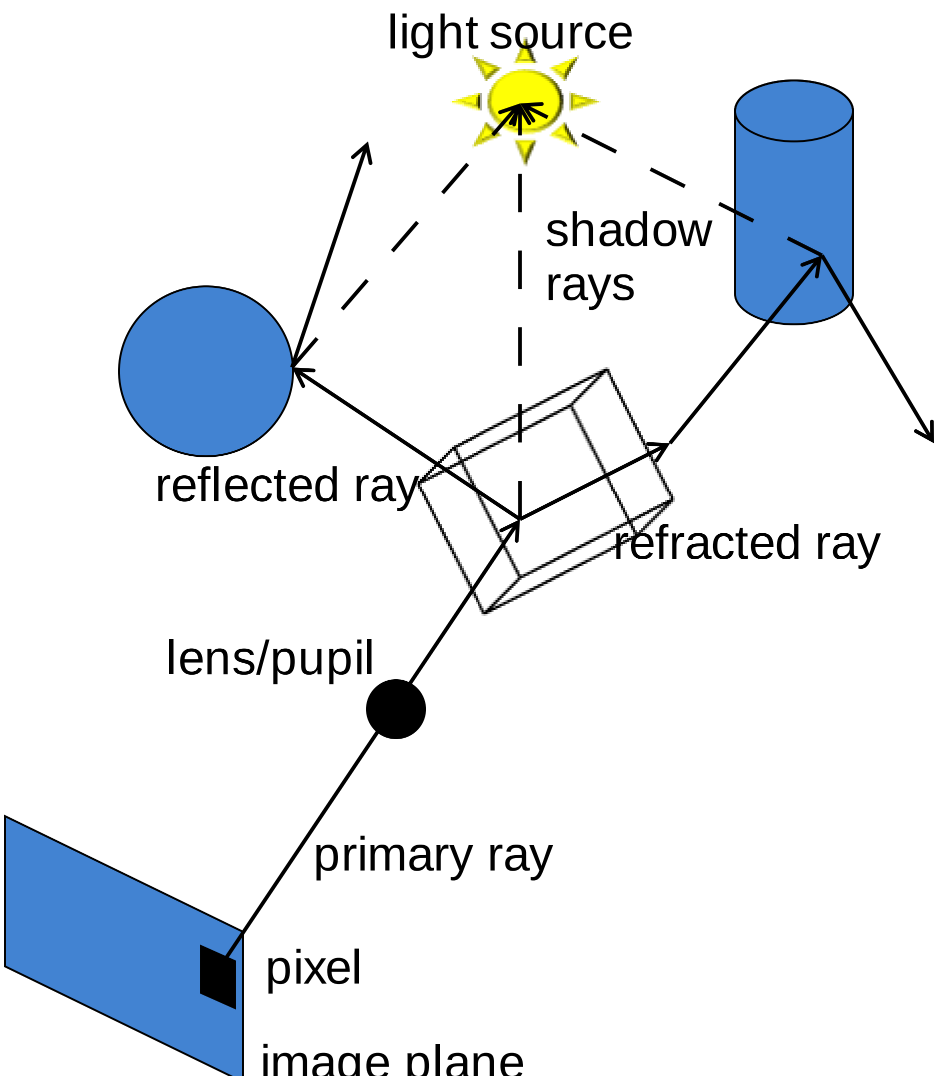
Recursive Ray Tracing Pipeline (8)



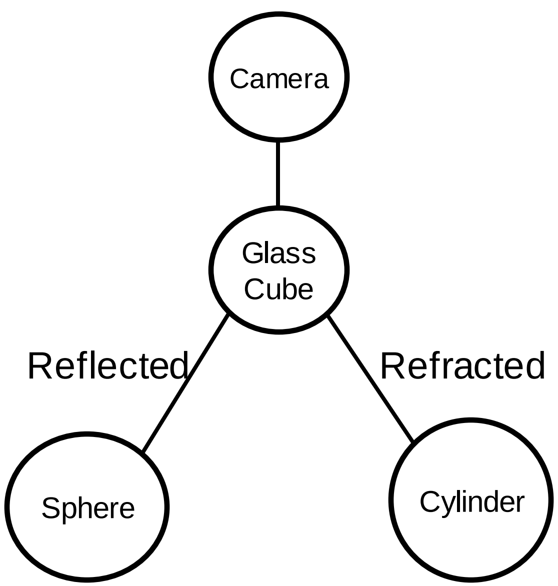
Recursive Ray Tracing Pipeline (9)



Recursive Ray Tracing



- **Searching recursively for paths to light sources**
  - Interaction of light & material at intersections
  - Trace rays to light sources
  - Recursively trace new ray paths in reflection & refraction directions



Ray Tracing Algorithm

- **Trace(ray)**
  - Search the next intersection point (hit, material)
  - Return Shade(ray, hit, material) → radiance/color
- **Shade(ray, hit, material)**
  - If object is emissive (i.e. light source)
    - Add radiance emitted towards ray to the reflected radiance
  - For each light source
    - if ShadowTrace(ray towards light source, distance to light)
      - Compute radiance emitted from light source towards shadow ray
      - Calculate radiance reflected at hit point towards incoming ray
      - Adding radiance to the reflected radiance
  - If mirroring material
    - Recursively calculate radiance from reflected direction:
      - Trace(ReflectRay(ray, hit))
    - Adding mirrored radiance to the reflected radiance
  - Similar for transmissive materials
  - 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 – WHY?

Ray Tracing Algorithm

- **Trace(ray)**
  - Search the next intersection point (hit, material)
  - Return Shade(ray, hit, material) → radiance/color
- **Shade(ray, hit, material)**
  - If object is emissive (i.e. light source)
    - Add radiance emitted towards ray to the reflected radiance
  - For each light source
    - if ShadowTrace(ray towards light source!!, distance to light)
      - Compute radiance emitted from light source towards shadow ray
      - Calculate radiance reflected at hit point towards incoming ray
      - Adding radiance to the reflected radiance
  - If mirroring material
    - Recursively calculate radiance from reflected direction:
      - Trace(ReflectRay(ray, hit))
    - Adding mirrored radiance to the reflected radiance
  - Similar for transmissive materials
  - 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 – WHY?

Shading (Material)

- **Intersection point determines primary ray's "color"**
  - Diffuse object: isotropic reflection of illumination at hit point
    - No variation with viewing angle: diffuse (or Lambertian)
  - Specular: Perfect reflection/refraction (mirror, glass)
    - Only one outgoing direction each → Trace secondary ray path(s)
  - More general reflectance models
    - Appearance depends on illumination and viewing direction
    - Local ***Bi-directional Reflectance Distribution Function*** (BRDF)
- **Illumination**
  - Point/directional light sources
  - Slight generalization: Area light sources
    - Approximate with multiple samples / shadow rays
  - Global illumination (computes also indirect illumination)
    - See Realistic Image Synthesis (RIS) course in next semester
- **More details later**

Common Approximations

- **Usually RGB color model (red, green, blue)**
  - Instead of full spectrum → later
- **Light only from finite # of light sources**
  - Instead of full indirect light from all directions
- **Approximate material reflectance properties**
  - Diffuse: light reflected uniformly in all directions
  - Specular: perfect reflection, refraction
  - Or mix of these two
- **Reflection models are often empirical**
  - Often using Phong/Blinn shading model (or variation thereof)
  - But physically-based models are available as well

  → later

Ray Tracing Features

- **Incorporates into 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 & 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**
  - Many reflections or refractions
    - Exponential increase in number of rays
  - Indirect illumination requires many rays to sample all incoming directions
    - Easily gets inefficient for full global illumination computations
  - Solved with Path Tracing (→ RIS course)

Ray Tracing Can…

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

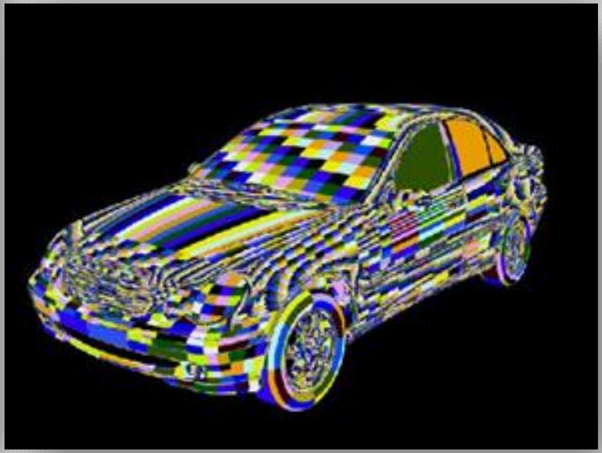
What is Possible?

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

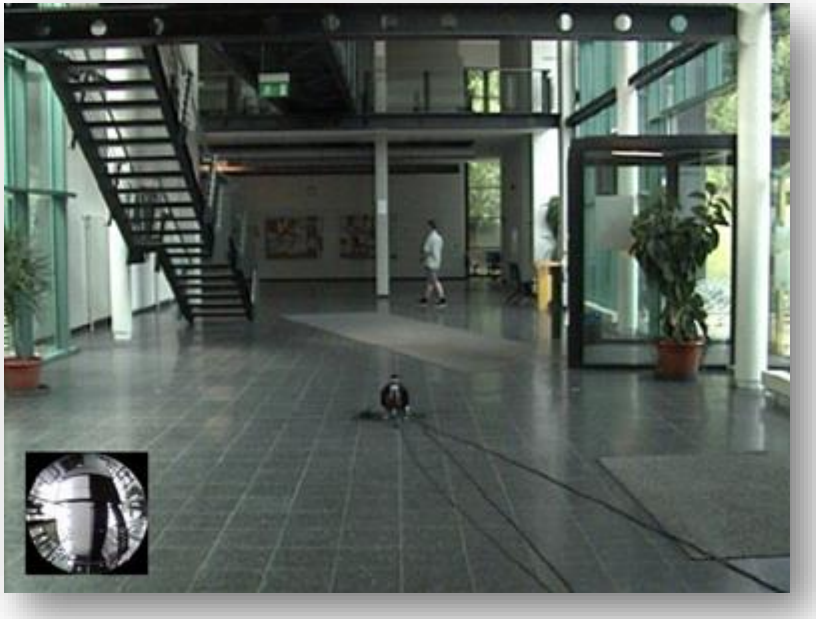
VW Visualization Center

Realistic Visualization: CAD





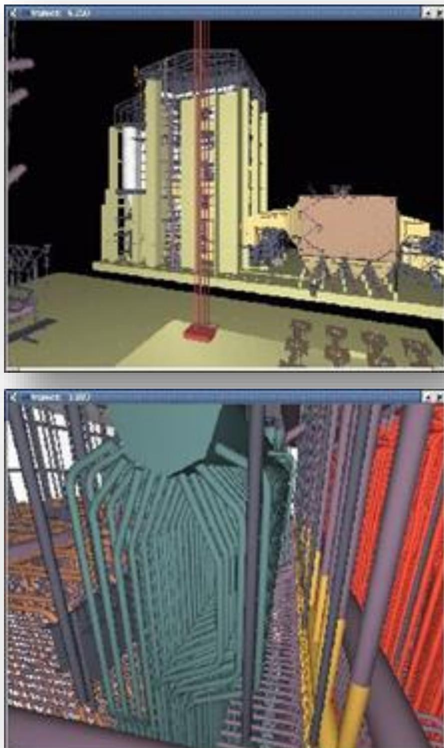
Realistic Visualization: VR/AR



Lighting Simulation

What is Possible?

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

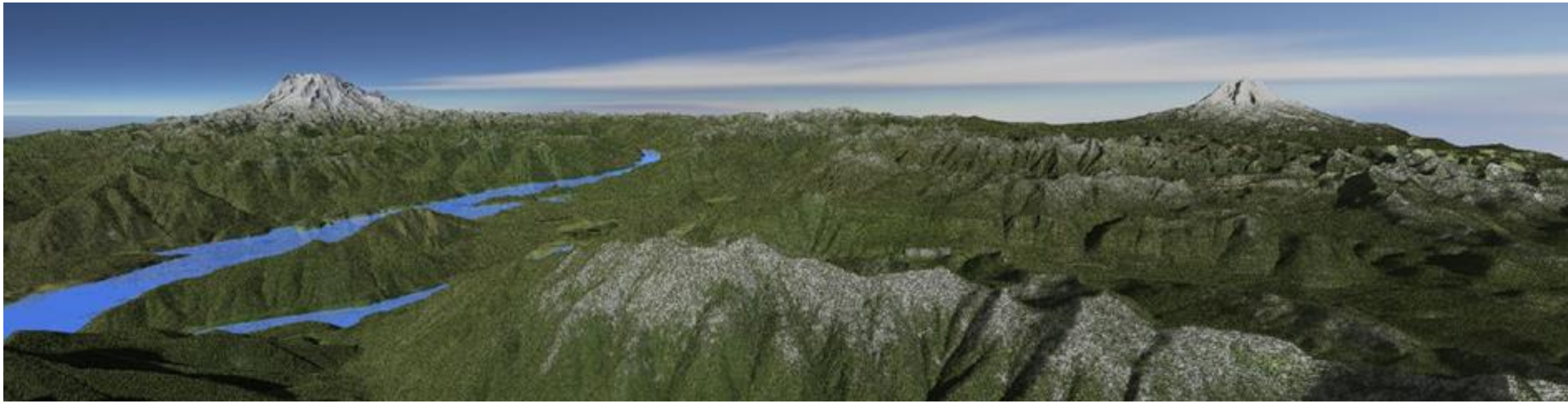


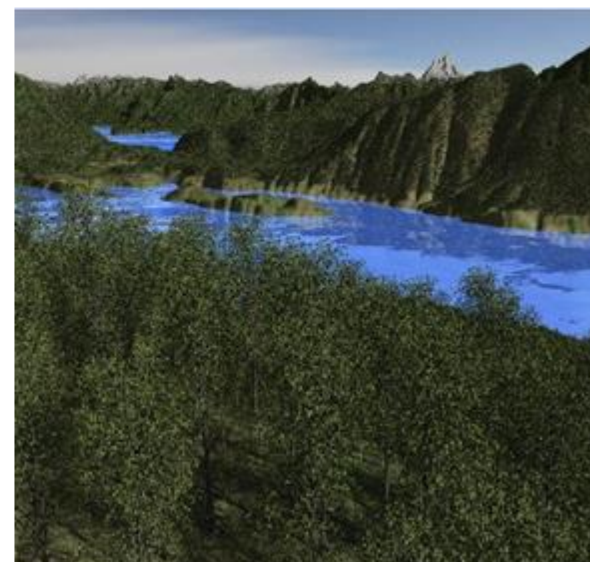
12.5 Million Triangles

~1 Billion Triangles

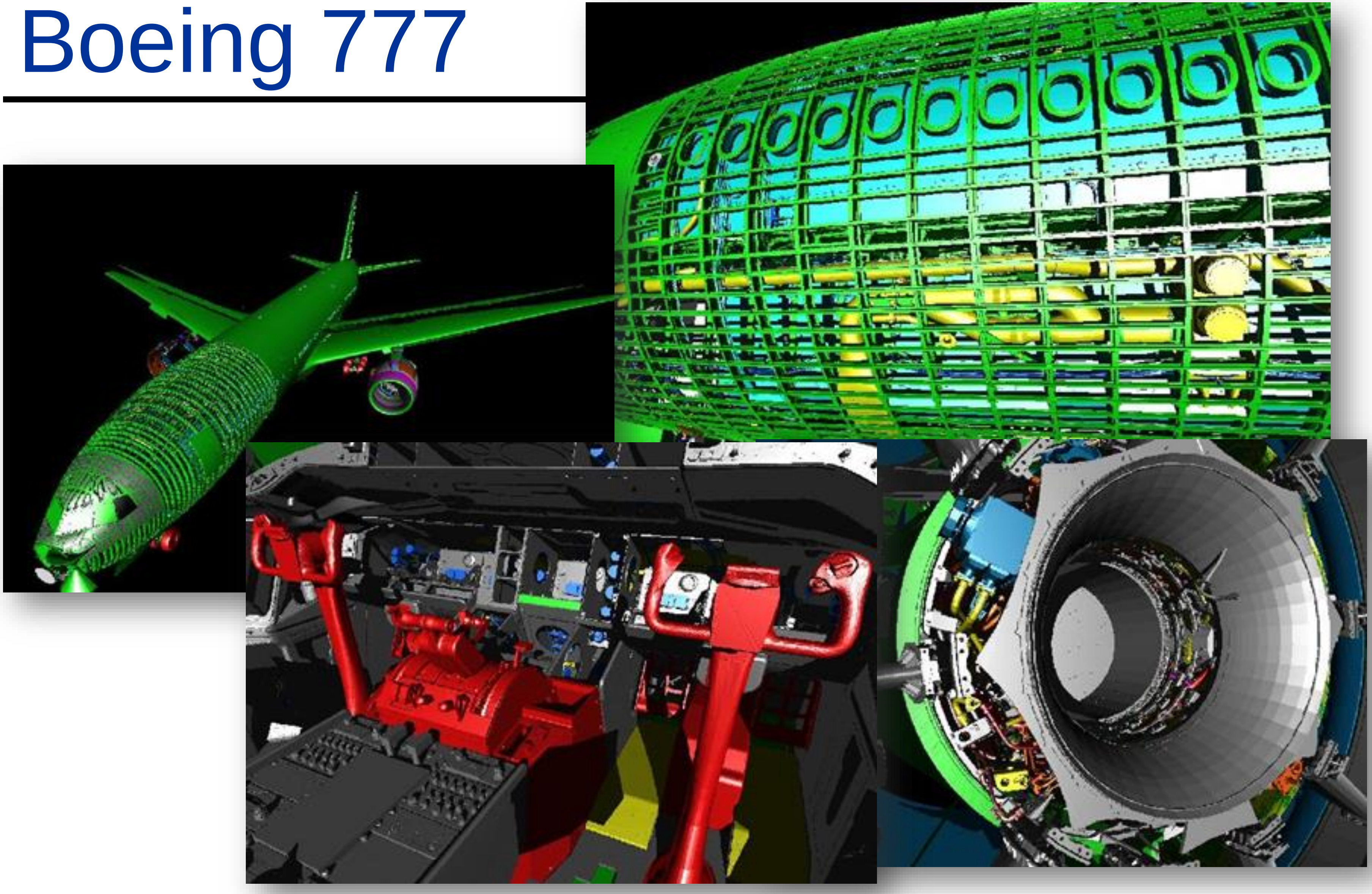
Outdoor Environments

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





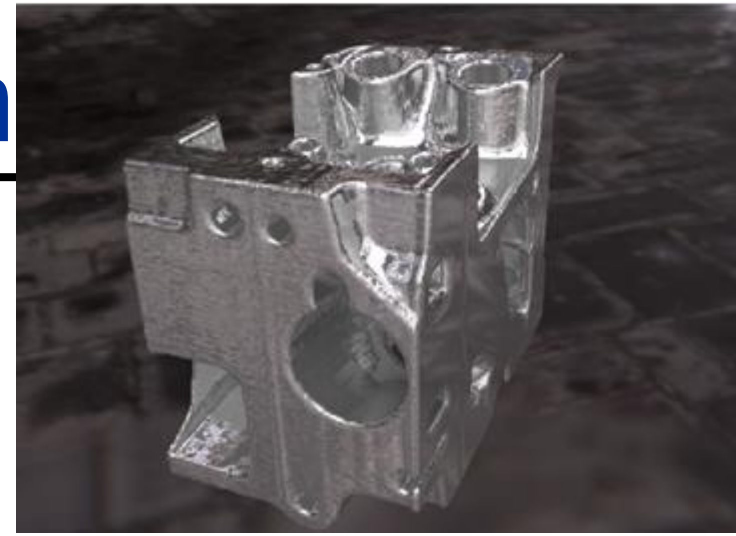
Boeing 777

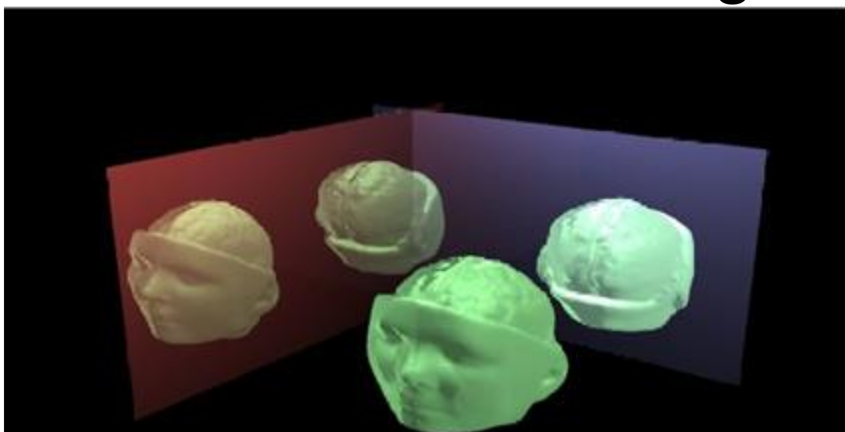


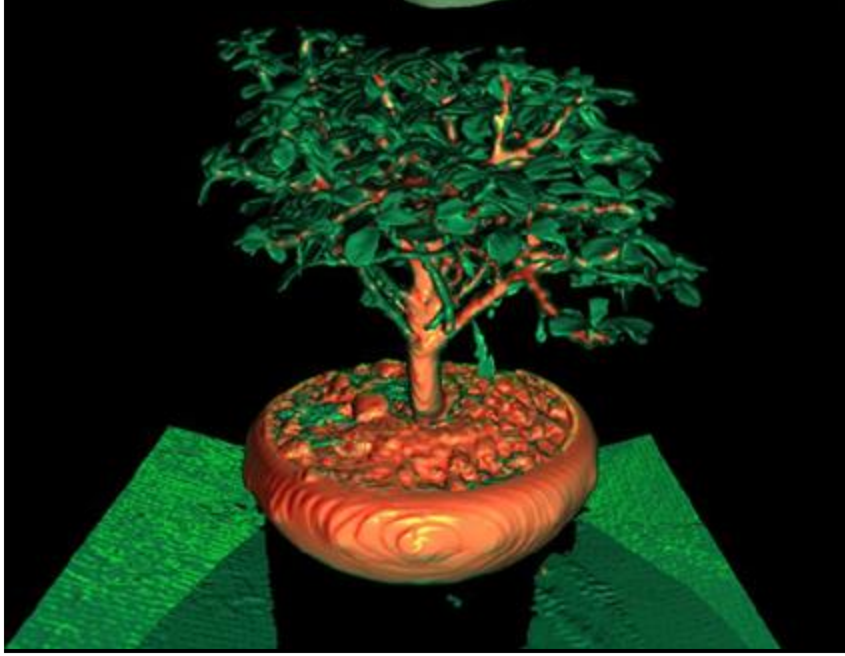
Boeing 777: ~350 million individual polygons, ~30 GB on disk

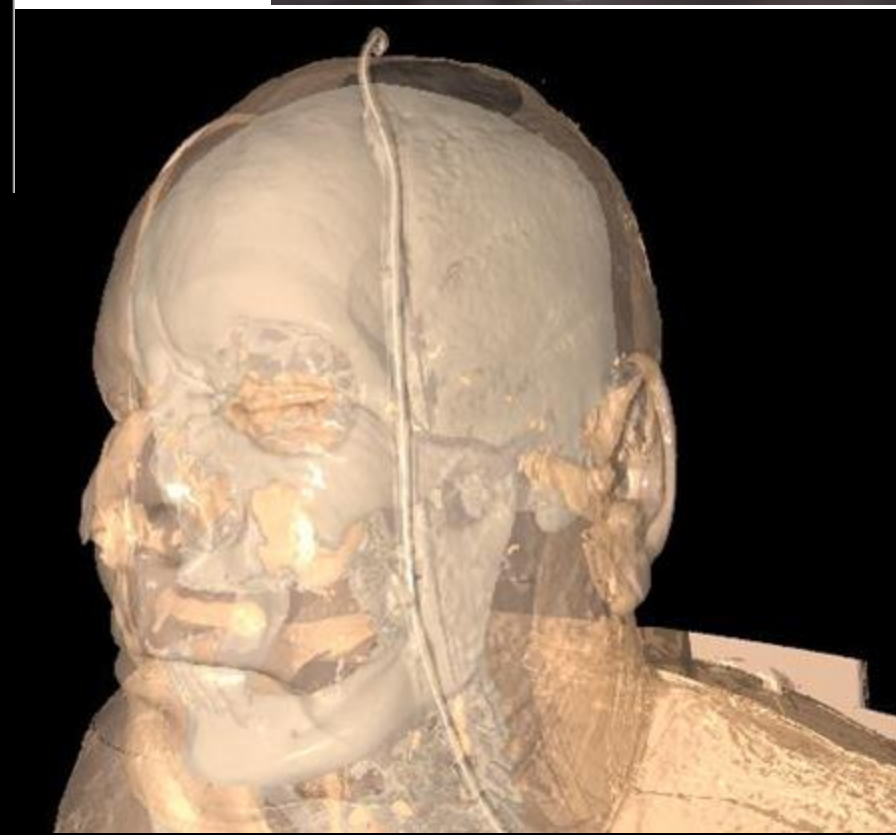
Volume Visualization

- **Iso-surface rendering**



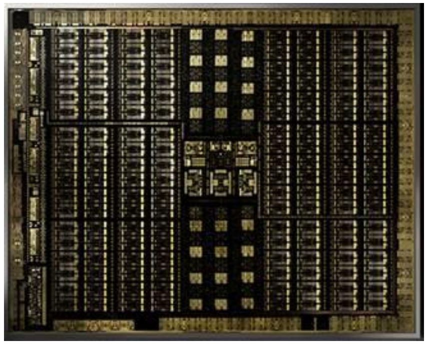






Games? (in 2006)

Games!



Nvidia RTX (Turing)
(up to 10 Grays/s)

Ray Tracing in CG

- **In the Past (until end of 80ies)**
  - Was computationally very demanding (minutes to hours per frame)
  - Tried hard to speed it up, but always too slow → only off-line use
- **“Lost generation” (1990ies)**
  - Believed ray tracing would not be suitable for HW implementations
  - Believed ray tracing would always be slower than rasterization
- **More Recently**
  - Interactive ray tracing on supercomputers [Parker, U. Utah‘98]
  - Interactive ray tracing on PCs [Wald‘01]
  - Distributed real-time ray tracing on PC clusters [Wald’01]
  - RPU: First full HW implementation [Siggraph 2005]
  - Commercial tools: Embree (Intel/CPU), OptiX (Nvidia/GPU)
  - Complete film industry has switched to ray tracing (Monte-Carlo)
- **Own conference**
  - Symposium on Interactive RT, now High-Performance Graphics (HPG)
- **Ray tracing systems**
  - Research: PBRT (offline, physically-based, based on book, OSS), Mitsuba-2 renderer (EPFL), Rodent (SB), …
  - Products: Blender (OSS), V-Ray (Chaos Group), Arnold & VRED (Autodesk), Corona (Render Legion), MentalRay/iRay (MI), …

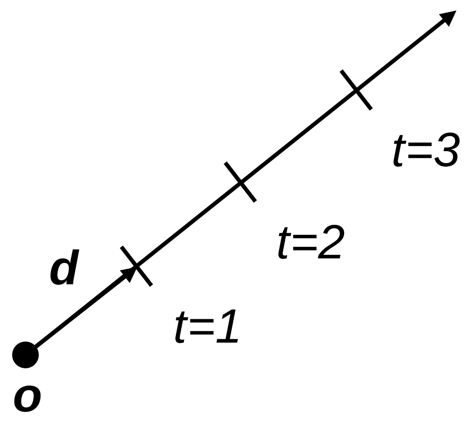
Ray Casting Outside CG

- **Tracing/Casting a ray**
  - Special type of query
    - “Is there a primitive along a ray”
    - “How far is the closest primitive”
- **Other uses than rendering**
  - Visibility computation
  - Volume computation
  - Collision detection
  - Acoustics
  - Radar
  - …

RAY-PRIMITIVE INTERSECTIONS

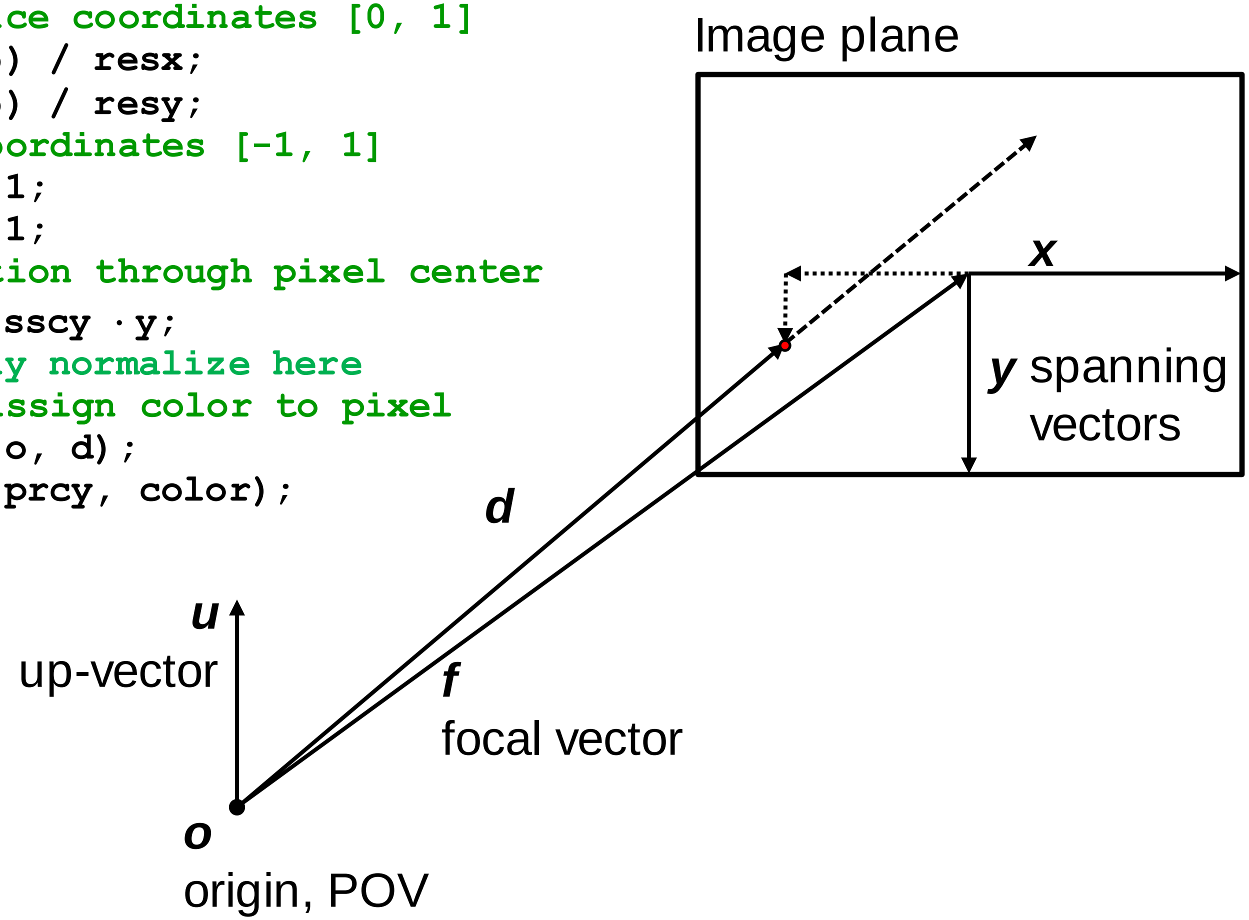
Basic Math - Ray

- **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 $t \in \mathbb{R}_{0+}$



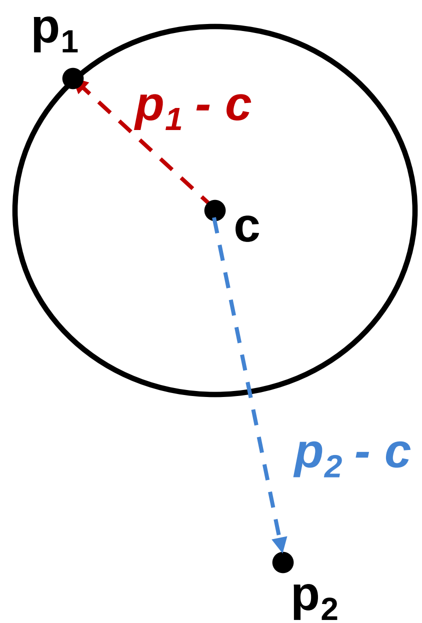
Pinhole Camera Model

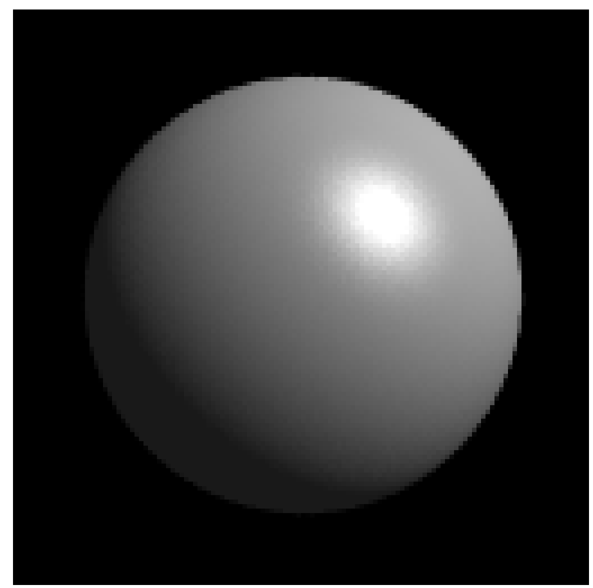
```
// For given image resolution {resx, resy}
// Loop over pixel raster coordinates [0, res-1]
for(prcx = 0; prcx < resx; prcx++)
  for(prcy = 0; prcy < resy; prcy++)
  {
    // Normalized device coordinates [0, 1]
    ndcx = (prcx + 0.5) / resx;
    ndcy = (prcy + 0.5) / resy;
    // Screen space coordinates [-1, 1]
    sscx = ndcx * 2 - 1;
    sscy = ndcy * 2 - 1;
    // 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
  - $\forall \vec{p} \in \mathbb{R}^3 : \vec{p} \in S \Leftrightarrow (\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) - r^2 = 0$
    - The distance between points on the sphere and its center equals the radius



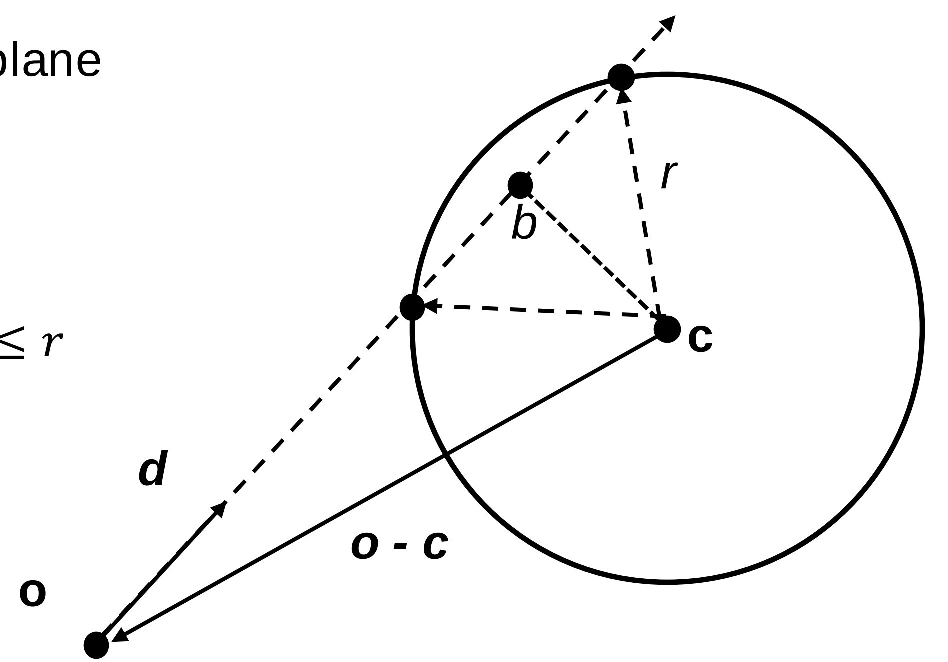


Ray-Sphere Intersection

- **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*

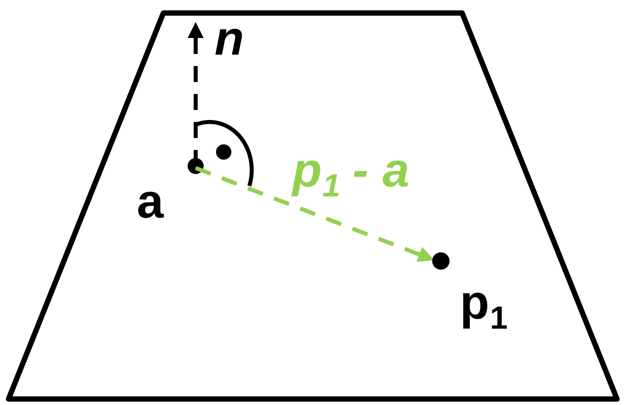
Ray-Sphere Intersection (2)

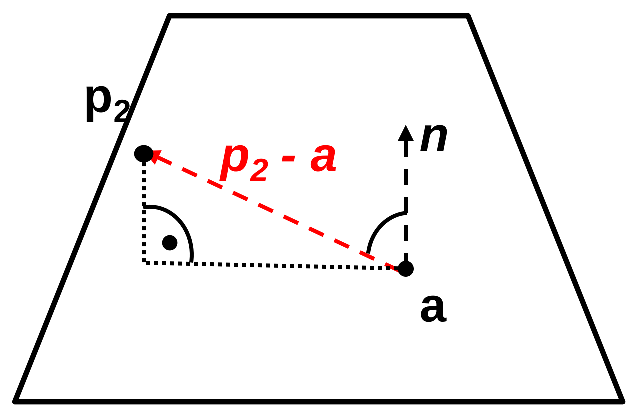
- **Given**
  - Ray: $r(t) = \vec{o} + t\vec{d}$, t $\in \mathbb{R}$; $\vec{o}, \vec{d} \in \mathbb{R}^3$
  - Sphere: $\vec{c} \in \mathbb{R}^3, r \in \mathbb{R}$:
    - $\forall \vec{p} \in \mathbb{R}^3$: $\vec{p} \in S \Leftrightarrow (\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) - r^2 = 0$
- **Find closest intersection point**
  - Geometric approach
    - Ray and center span a plane
    - Solve in 2D
    - Compute $|\vec{b} - \vec{o}|, |\vec{b} - \vec{c}|$
      - Such that $\measuredangle obc = 90°$
    - Intersection(s) if $|\vec{b} - \vec{c}| \leq r$
  - Be aware of floating point issues if o is far from sphere



Basic Math - Plane

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





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
    - **How many intersections could there be?**

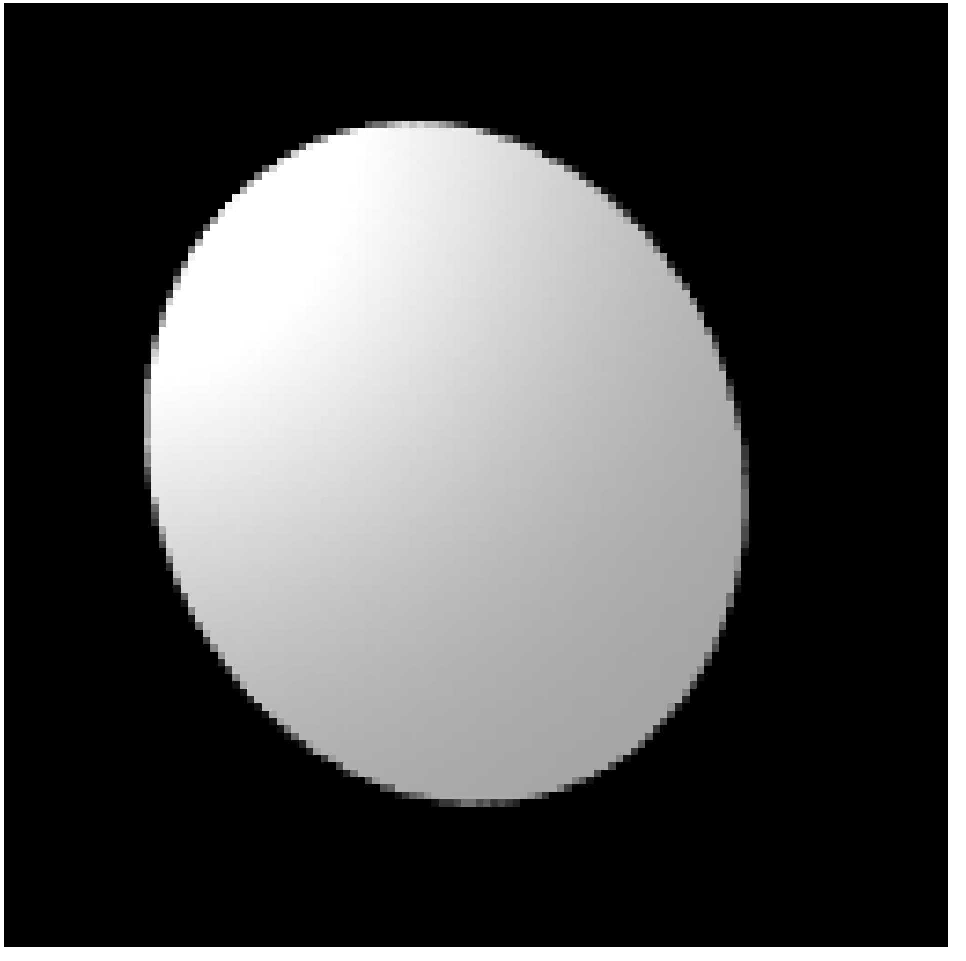
Ray-Plane Intersection

- **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
    - 1: General case
    - 0: Ray is parallel to but offset from plane
    - ∞: Ray lies within plane

Ray-Disc Intersection

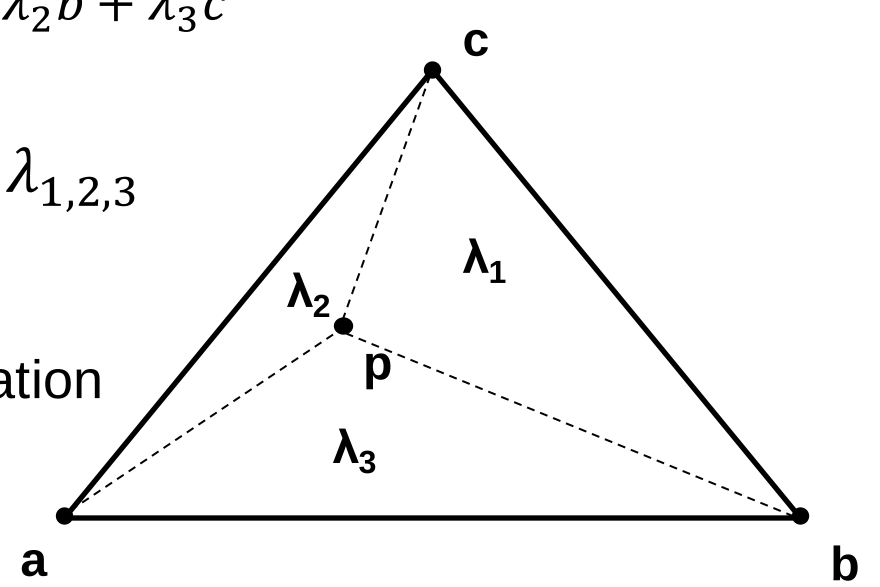
- **Intersect ray with plane**
- **Discard intersection if ||p – a|| > r**



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

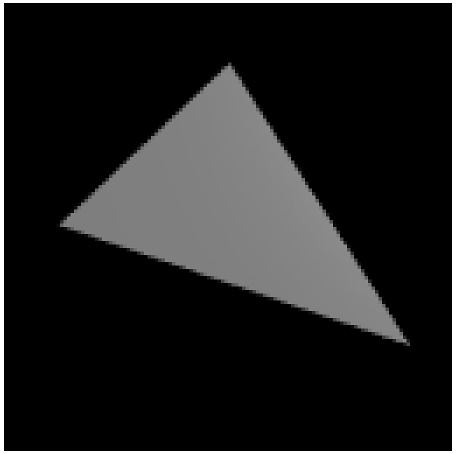
- **Barycentric coordinates** $\lambda_{1,2,3}$
  - $\lambda_1 = A_{pbc} / A_{abc}$, etc.
  - A: signed area of triangle,
    based on CLW/CCW orientation

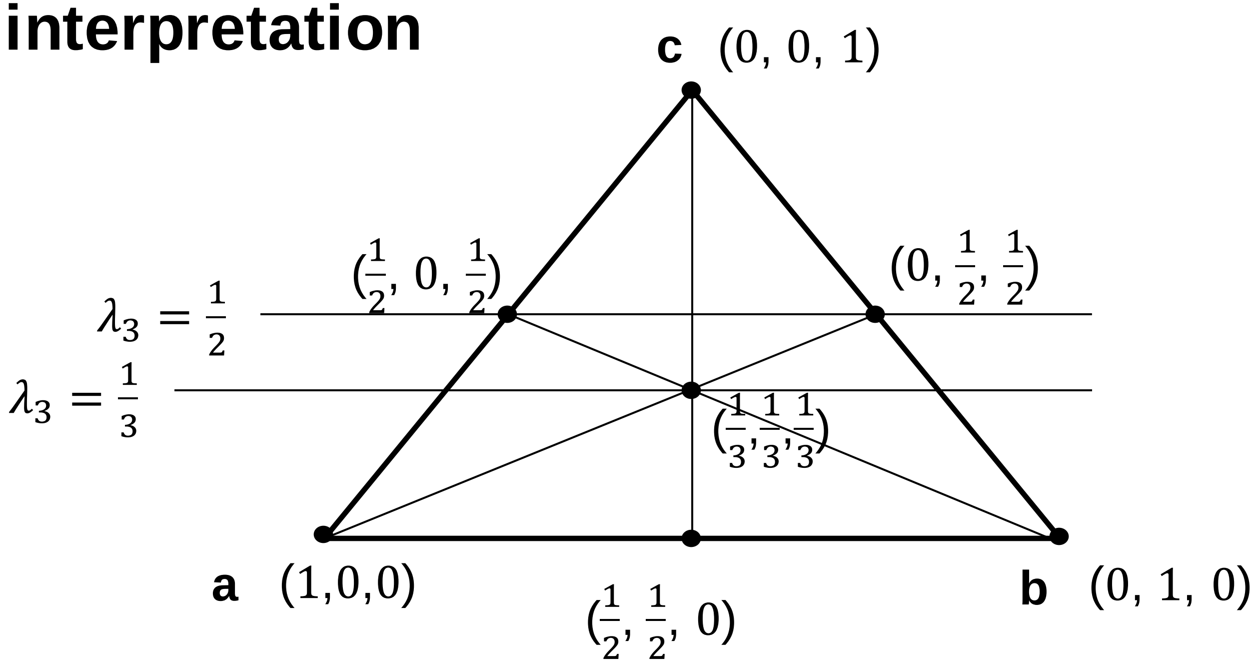


Barycentric Coordinates (BCs)

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

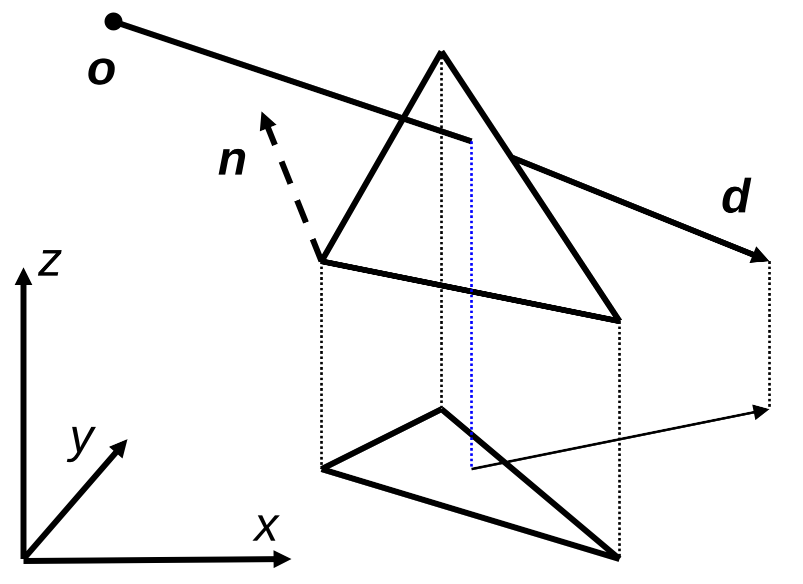
- **Easy geometric interpretation**





Triangle Intersection: Plane-Based

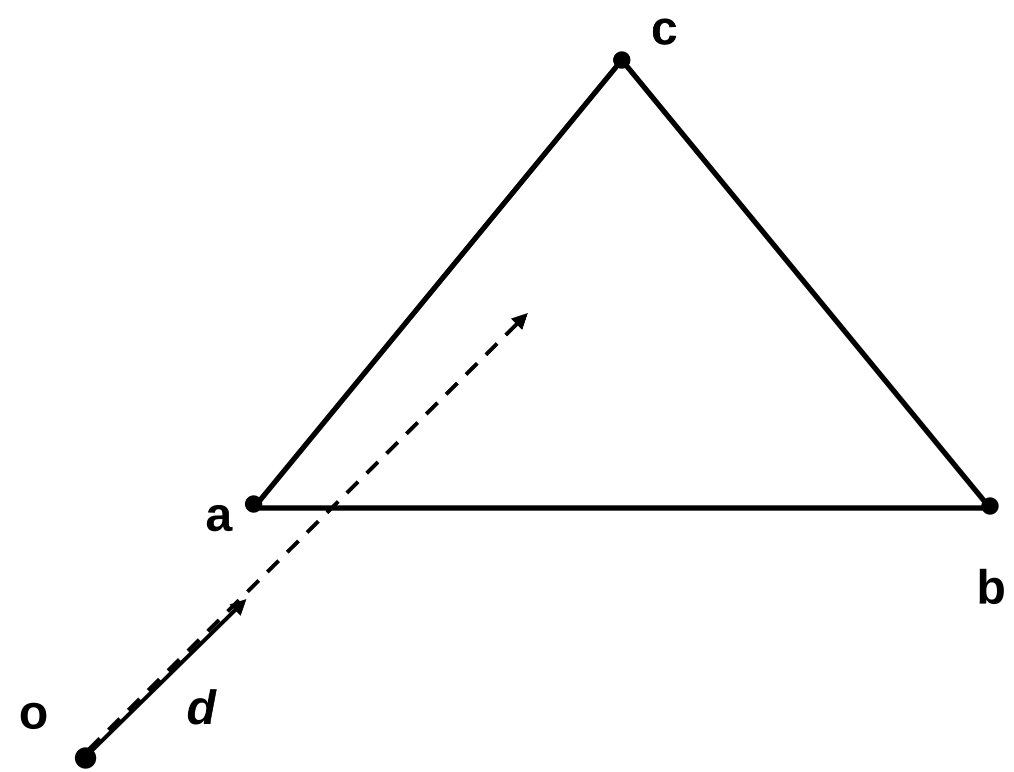
- **Compute intersection with triangle's plane**
  - Plane equation easily computable from vertices via cross product
- **Compute barycentric coordinates**
  - Signed areas of subtriangles
  - Can be done in 2D, after "projection" onto major plane, depending on largest component of normal vector
    - Maximizes area and numerical stability
- **Test for positive BCs**



- **Issues:**
  - Edges of neighboring triangles might not be identical
  - Due to inaccuracies of floats
  - Need a better method!

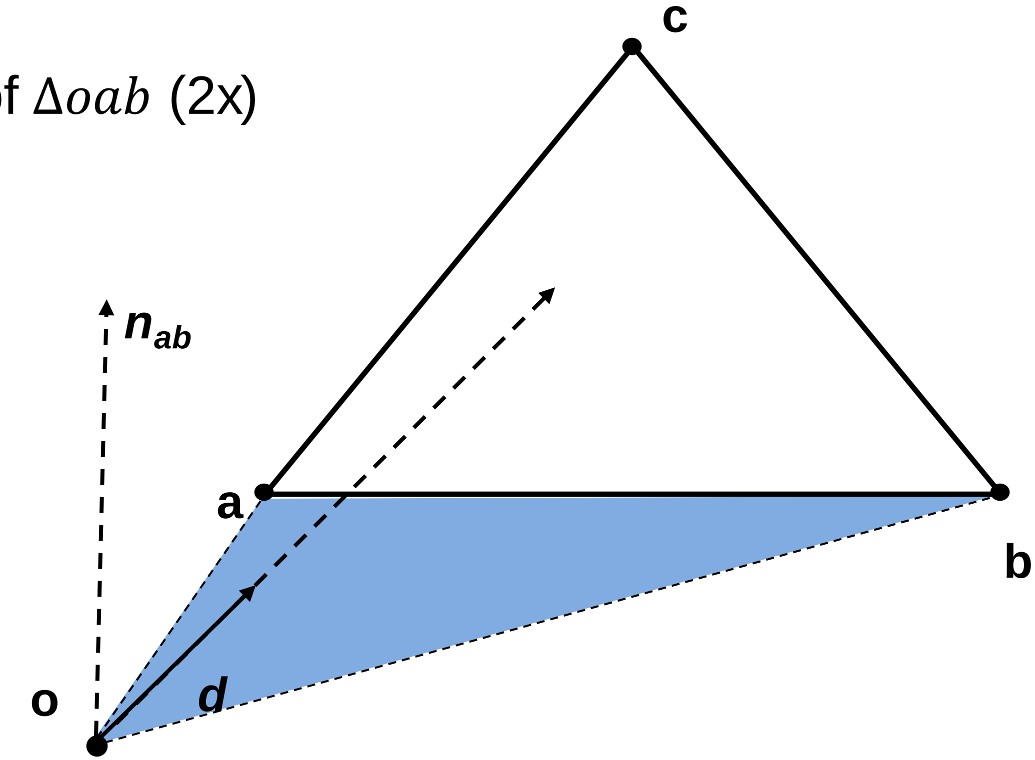
Triangle Intersection: Edge-Based

- **3D linear function across triangle (3D edge functions)**
  - Ray: $\vec{o} + t\vec{d}$, $t \in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$



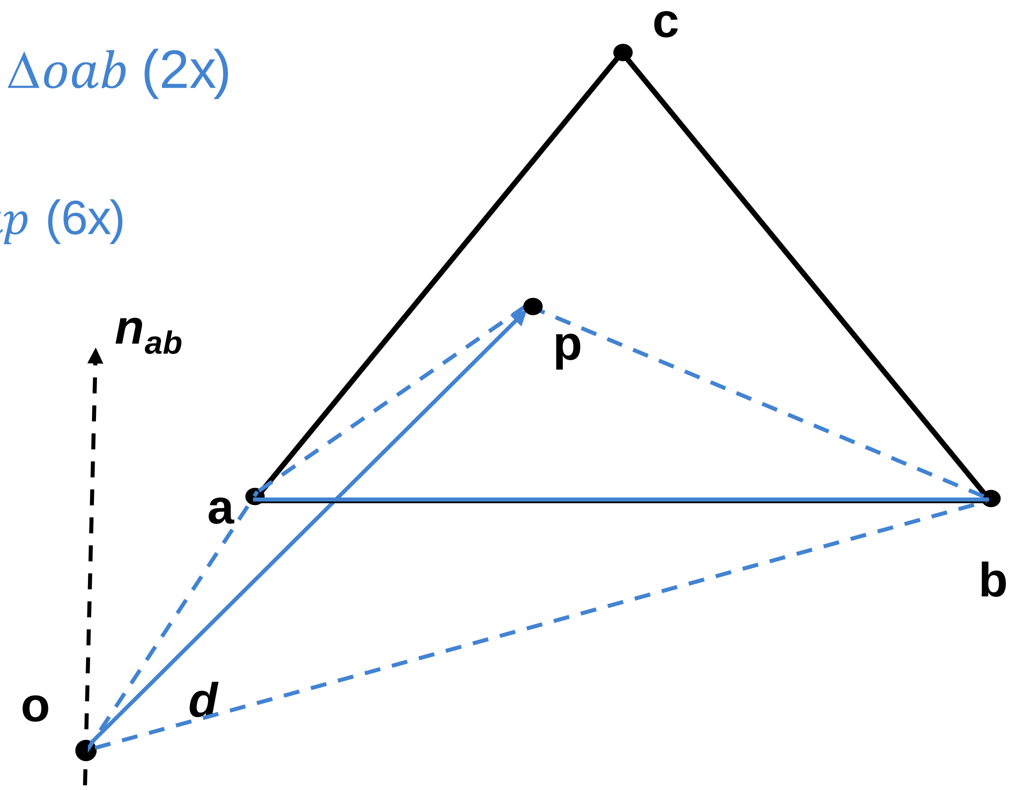
Triangle Intersection: Edge-Based

- **3D linear function across triangle (3D edge functions)**
  - Ray: $\vec{o} + t\vec{d}$,   t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$
  - $\overrightarrow{n_{ab}} = (\vec{b} - \vec{o}) \times (\overrightarrow{a} - \vec{o})$
  - $|\overrightarrow{n_{ab}}|$ is the signed area of Δoab (2x)



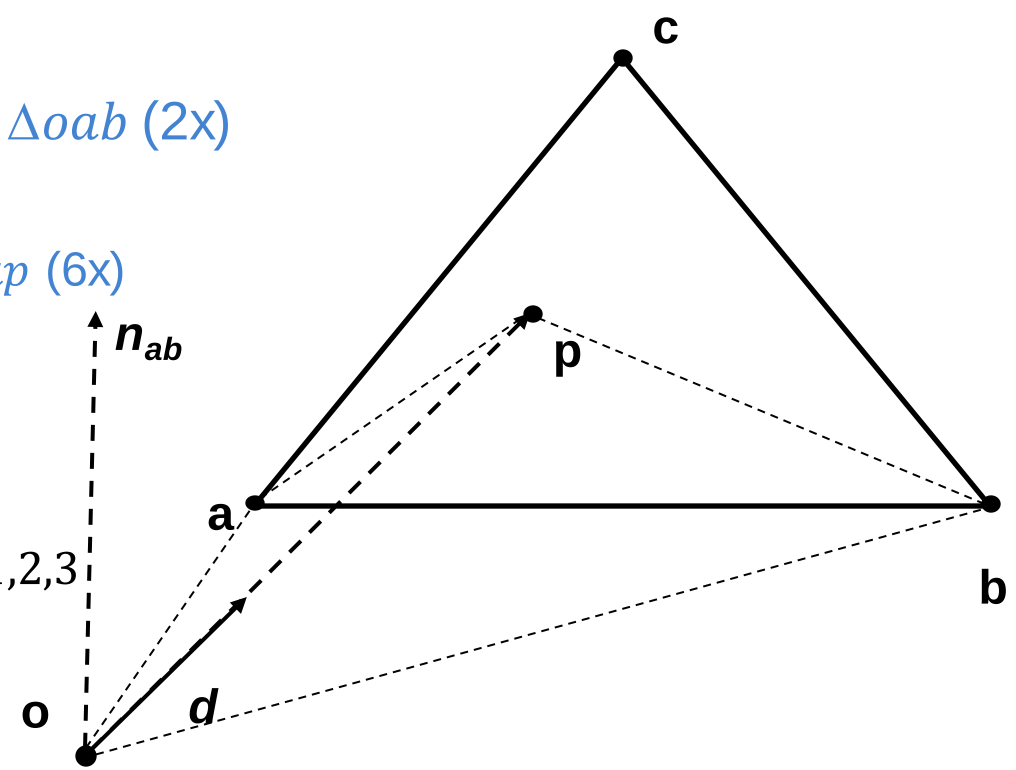
Triangle Intersection: Edge-Based

- **3D linear function across triangle (3D edge functions)**
  - Ray: $\vec{o} + t\vec{d}$, $t \in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$
  - $\overrightarrow{n_{ab}} = (\vec{b} - \vec{o}) \times (\overrightarrow{a} - \vec{o})$
  - $|\overrightarrow{n_{ab}}|$ is the signed area of Δoab (2x)
  - $\lambda_3^*(t) = \overrightarrow{n_{ab}} \cdot t\vec{d}$
    - Volume of tetrahedra $obap$ (6x)
    - For $t = t_{hit}$



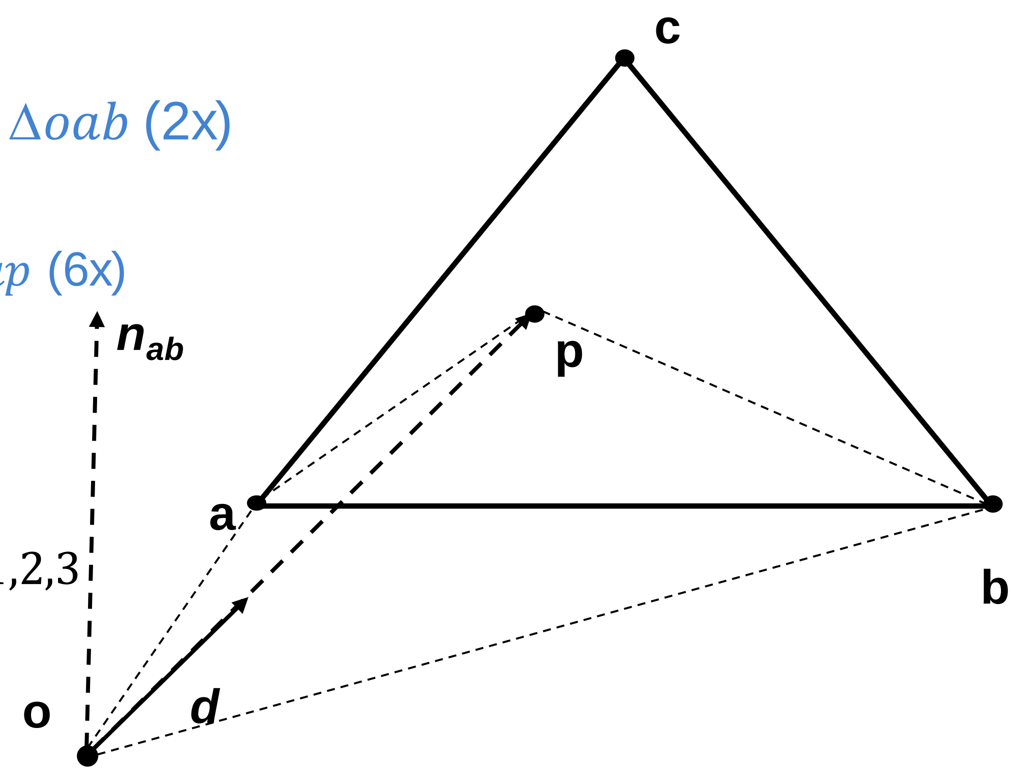
Triangle Intersection: Edge-Based

- **3D linear function across triangle (3D edge functions)**
  - Ray: $\vec{o} + t\vec{d}$, $\quad$ t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$
  - $\overrightarrow{n_{ab}} = (\vec{b} - \vec{o}) \times (\vec{a} - \vec{o})$
  - $|\overrightarrow{n_{ab}}|$ is the signed area of Δoab (2x)
  - $\lambda_3^*(t) = \overrightarrow{n_{ab}} \cdot t\vec{d}$
    - Volume of tetrahedra $obap$ (6x)
    - 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



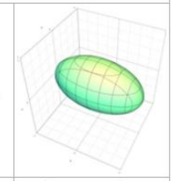
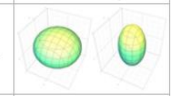
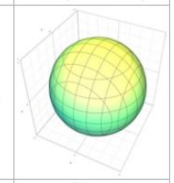
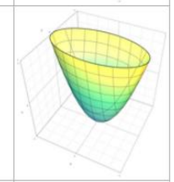
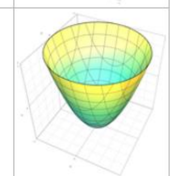
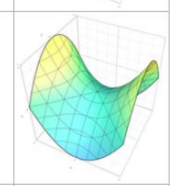
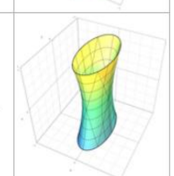
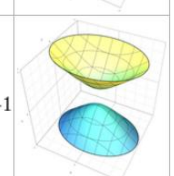
Triangle Intersection: Edge-Based

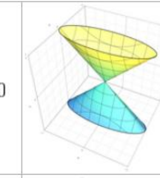
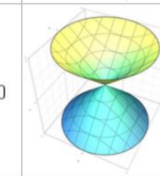
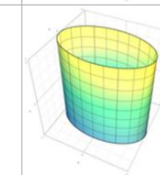
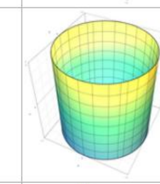
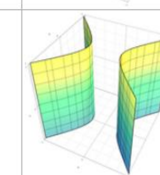
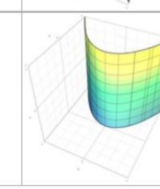
- **3D linear function across triangle (3D edge functions)**
  - Ray: $\vec{o} + t\vec{d}$, t $\in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$
  - $\overrightarrow{n_{ab}} = (\vec{b} - \vec{o}) \times (\vec{a} - \vec{o})$
  - $|\overrightarrow{n_{ab}}|$ is the signed area of Δoab (2x)
  - $\lambda_3^*(t) = \overrightarrow{n_{ab}} \cdot t\vec{d}$
    - Volume of tetrahedra $obap$ (6x)
    - For $t = t_{hit}$
  - $\lambda_{1,2}^*(t) = \overrightarrow{n_{bc,ac}} \cdot t\vec{d}$
  - Normalize
    - $\lambda_i = \frac{\lambda_i^*(t)}{\lambda_1^*(t)+\lambda_2^*(t)+\lambda_3^*(t)}, i = 1,2,3$
- **Hit, if all BCs positive:**
  - Compute $\vec{p} = \lambda_1\vec{a} + \lambda_2\vec{b} + \lambda_3\vec{c}$

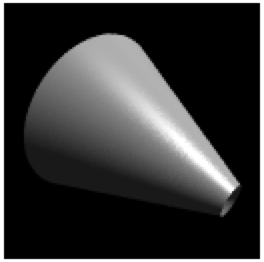


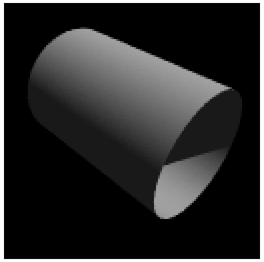
Quadrics

- **Implicit**
  - $f(x, y, z) = v$
- **Ray equation**
  - $x = x_o + t\, x_d$
  - $y = y_o + t\, y_d$
  - $z = z_o + t\, z_d$
- **Solve for t**

Non-degenerate real quadric surfaces		
Ellipsoid	$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$	
Spheroid (special case of ellipsoid)	$\frac{x^2}{a^2}+\frac{y^2}{a^2}+\frac{z^2}{b^2}=1$	
Sphere (special case of spheroid)	$\frac{x^2}{a^2}+\frac{y^2}{a^2}+\frac{z^2}{a^2}=1$	
Elliptic paraboloid	$\frac{x^2}{a^2}+\frac{y^2}{b^2}-z=0$	
Circular paraboloid(special case of elliptic paraboloid)	$\frac{x^2}{a^2}+\frac{y^2}{a^2}-z=0$	
Hyperbolic paraboloid	$\frac{x^2}{a^2}-\frac{y^2}{b^2}-z=0$	
Hyperboloid of one sheet	$\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=1$	
Hyperboloid of two sheets	$\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=-1$	

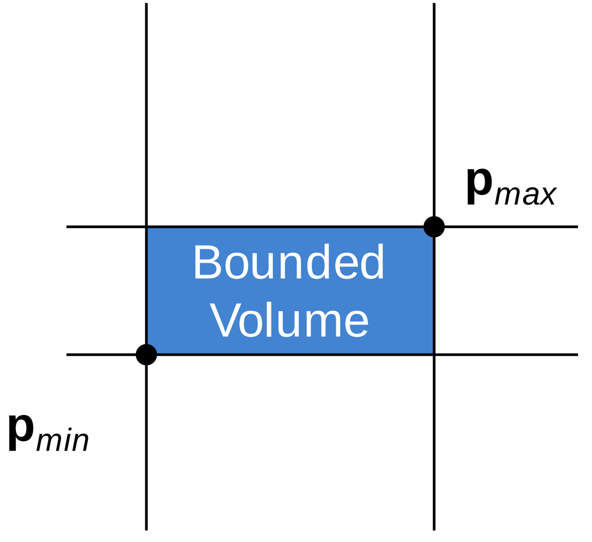
Degenerate quadric surfaces		
Cone	$\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=0$	
Circular Cone (special case of cone)	$\frac{x^2}{a^2}+\frac{y^2}{a^2}-\frac{z^2}{b^2}=0$	
Elliptic cylinder	$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$	
Circular cylinder (special case of elliptic cylinder)	$\frac{x^2}{a^2}+\frac{y^2}{a^2}=1$	
Hyperbolic cylinder	$\frac{x^2}{a^2}-\frac{y^2}{b^2}=1$	
Parabolic cylinder	$x^2+2ay=0$	

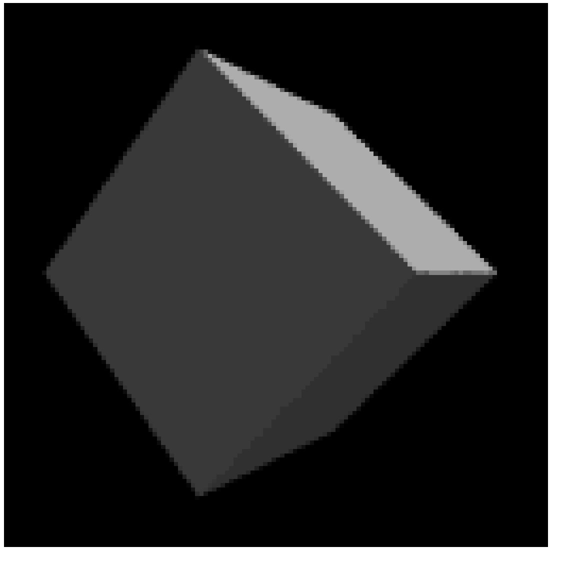




Axis Aligned Bounding Box

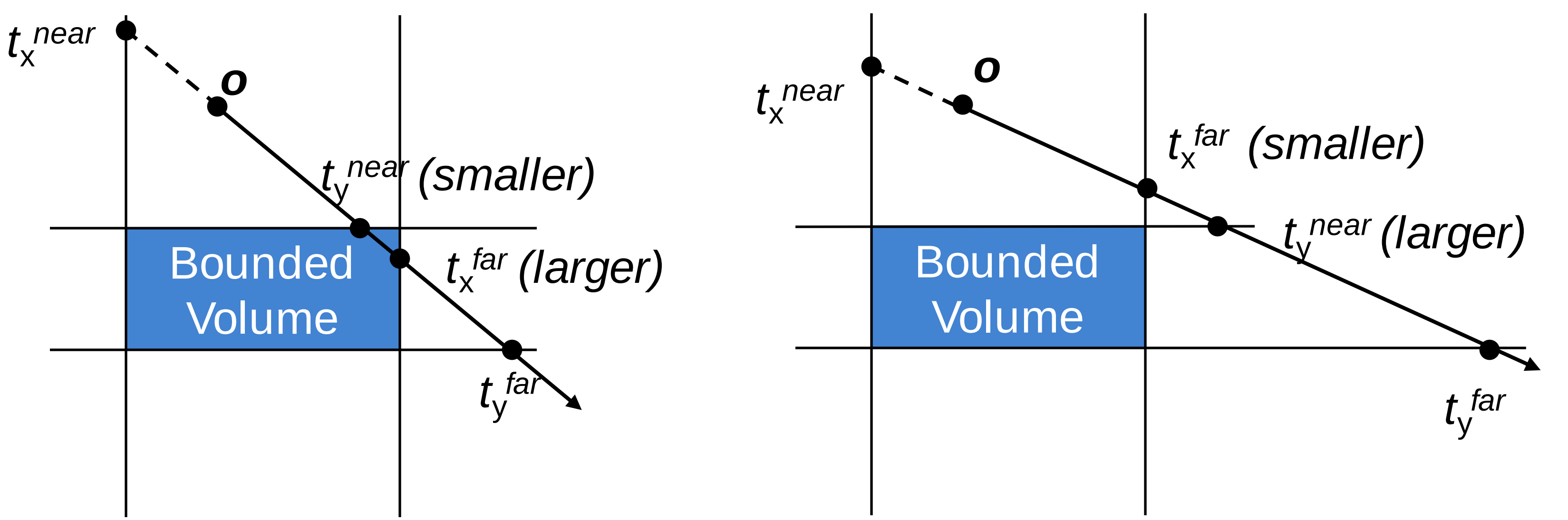
- **Given**
  - Ray: $\vec{o} + t\vec{d}$, $t \in \mathbb{R}; \vec{o}, \vec{d} \in \mathbb{R}^3$
  - Axis aligned bounding box (AABB): $\overrightarrow{p_{min}}, \overrightarrow{p_{max}} \in \mathbb{R}^3$





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
  - $\max(\{t_i^{near} | i = x, y, z\}) < \min(\{t_i^{far} | i = x, y, z\})$

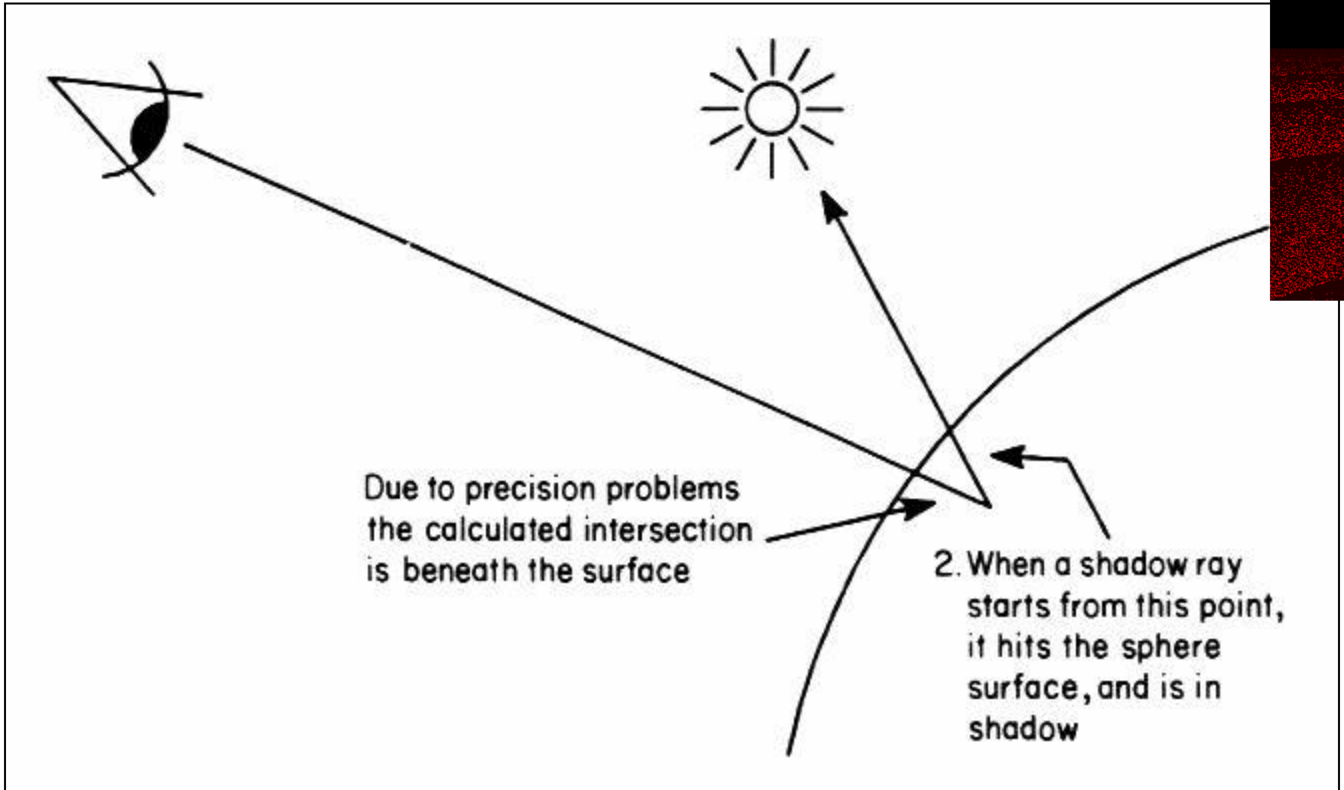


History of Intersection Algorithms

- **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]

Precision Problems

- **E.g., cause of „surface acne“**



Problem in surface intersection.