Lecture 3:

Ray Tracing 1

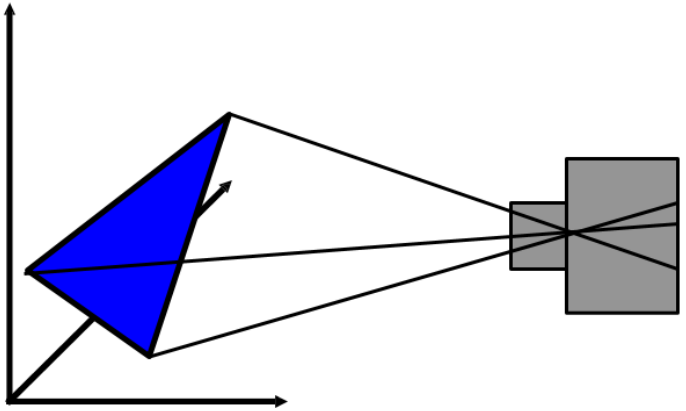
Contents

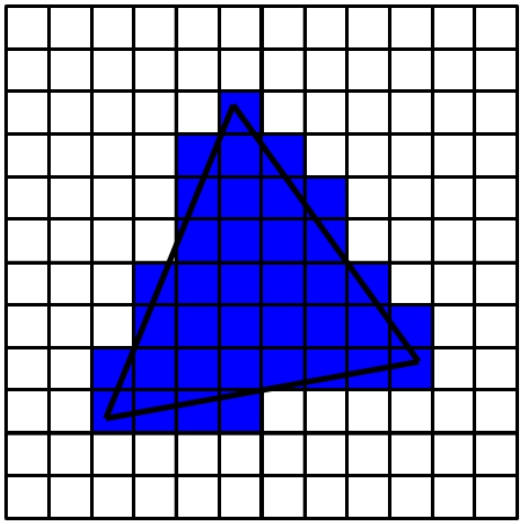
Rendering

- Given a camera and 3D scene as input, generate a 2D image as a view from the camera of the 3D scene

Algorithms: *Rasterization*

- **Primitive operation of all interactive graphics!**
  - Scan convert a single triangle at a time
- **Sequentially processes every triangle individually**
  - Can never access more than one triangle
- But most effects need access to the world: shadows, reflection, global illumination, *etc.*



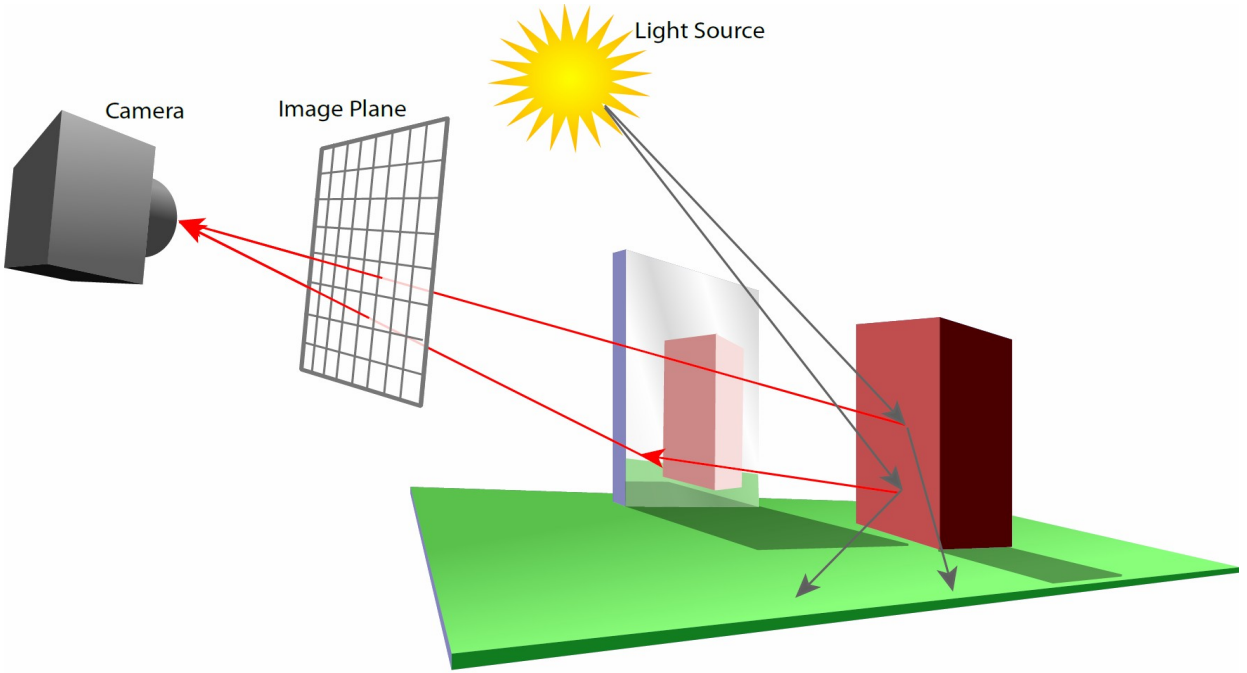


Rendering

- Given a camera and 3D scene as input, generate a 2D image as a view from the camera of the 3D scene

Algorithms: *Ray Tracing*

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



Light Distribution in a Scene

- Dynamic equilibrium
- Newly created, scattered, and absorbed photons

Forward Light Transport

- Shoot photons from the light sources 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 might transport light towards the camera
  - May try to connect to occluded light sources
- Ray Tracing

Surfaces

- 3D geometry of objects in a scene

Surface reflectance characteristics

- Color, texture, absorption, reflection, refraction, subsurface scattering
- Local property may vary over surface
- Mirror, glass, glossy, diffuse, …

Illumination

- Position and emission characteristics of light sources
- Repeatedly reflected light / indirect illumination

Assumption: air/empty space is totally transparent

- Simplification that excludes scattering effects in participating media volumes
- Later also volume objects, *e.g.* smoke, solid object (CT scan), …

Camera

- View point, viewing direction, field of view, resolution, …

One of the two fundamental rendering algorithms

Automatic, simple and intuitive

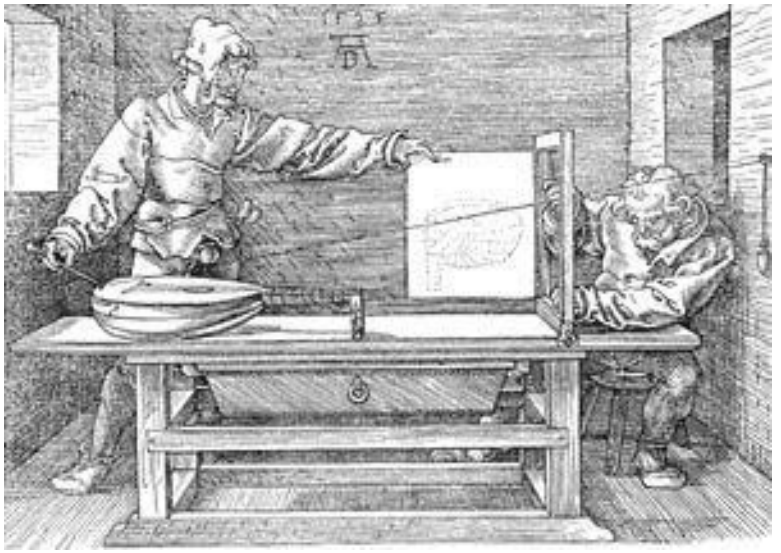
- Easy to understand and implement
- Delivers “correct“ images by default

Powerful and efficient

- Many optical global effects
- Shadows, reflections, refractions, …
- Efficient real-time implementation in SW and HW
- Can work 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), …
- Uses in lens design, geometric optics, …



Perspective Machine, Albrecht Dürer

Produce Highly Realistic Images

- Ray tracing enables *correct* simulation of light transport

Ray generation

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

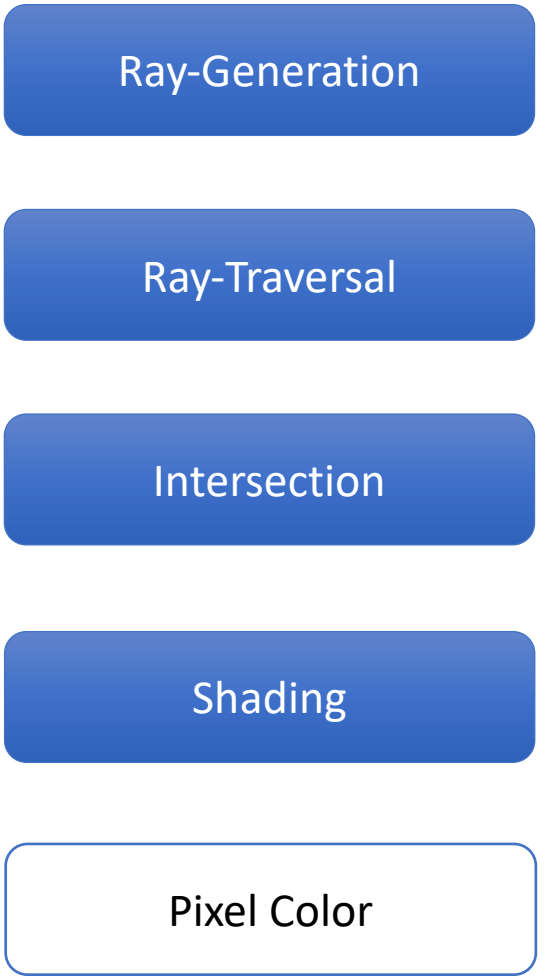
Ray traversal

- Traversal of spatial index structures

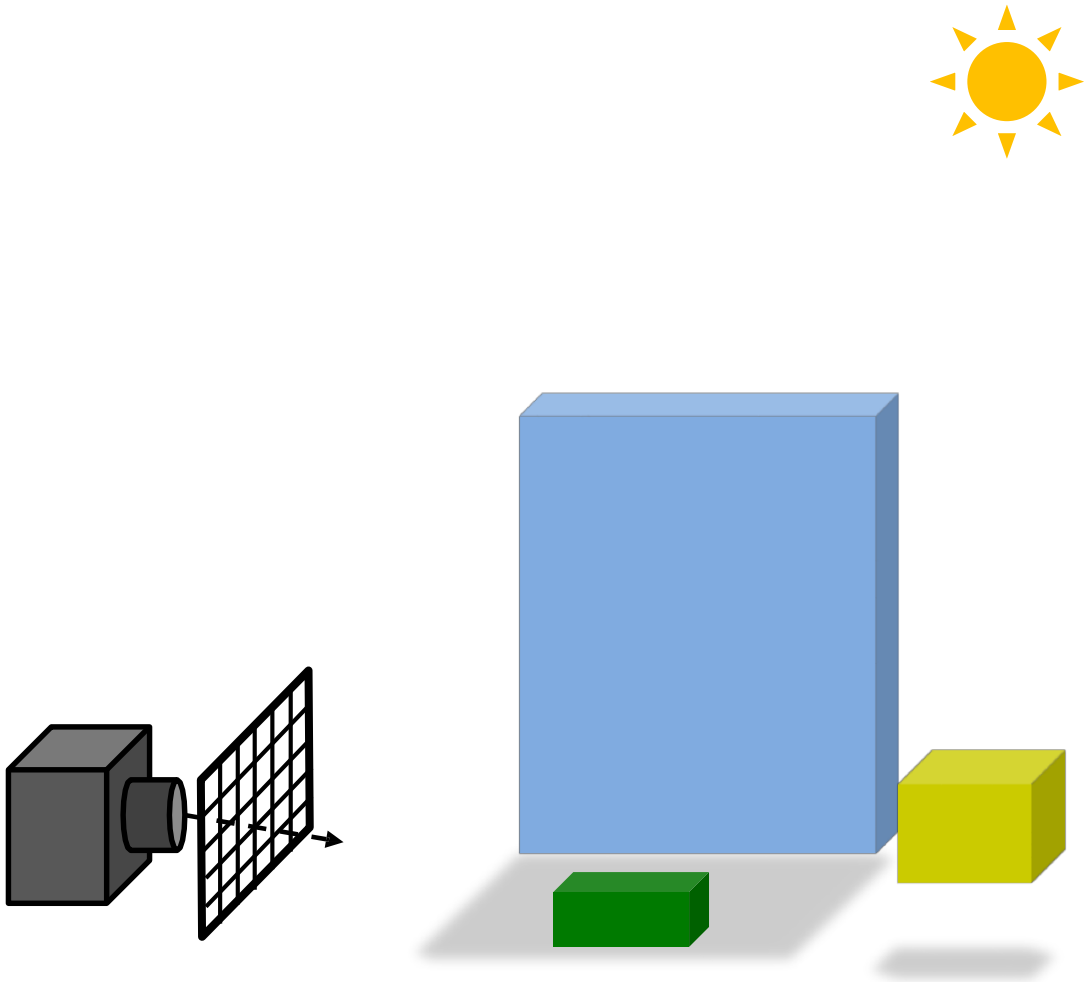
Ray-primitive intersection

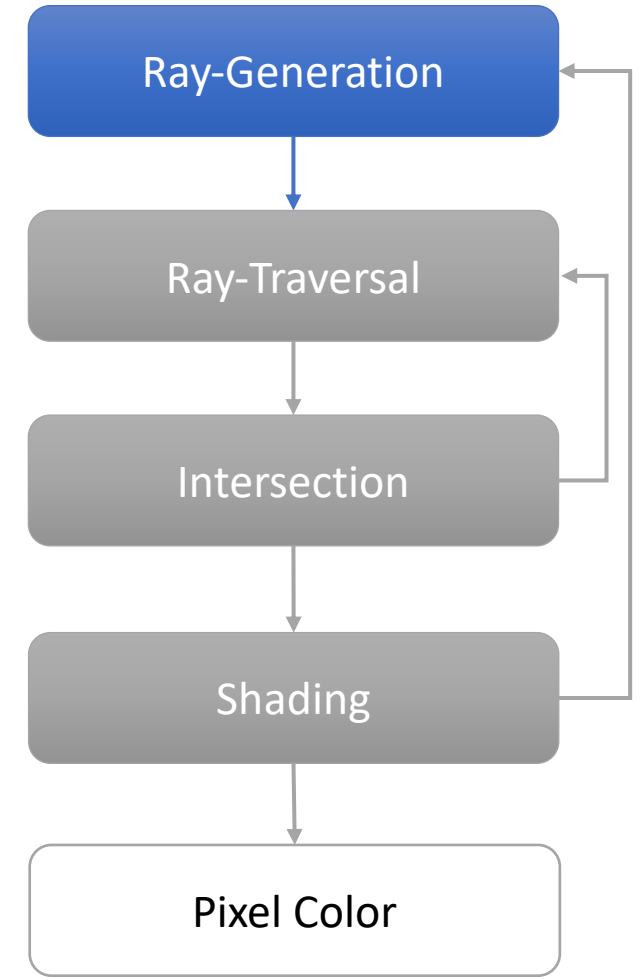
Shading the hit point

- Determine pixel color
  - Energy (color) travelling along primary ray
- Needed
  - Local material color, object texture and reflection properties
  - Local illumination at intersection point
    - Compute through recursive tracing of rays
    - Can be hard to determine correctly

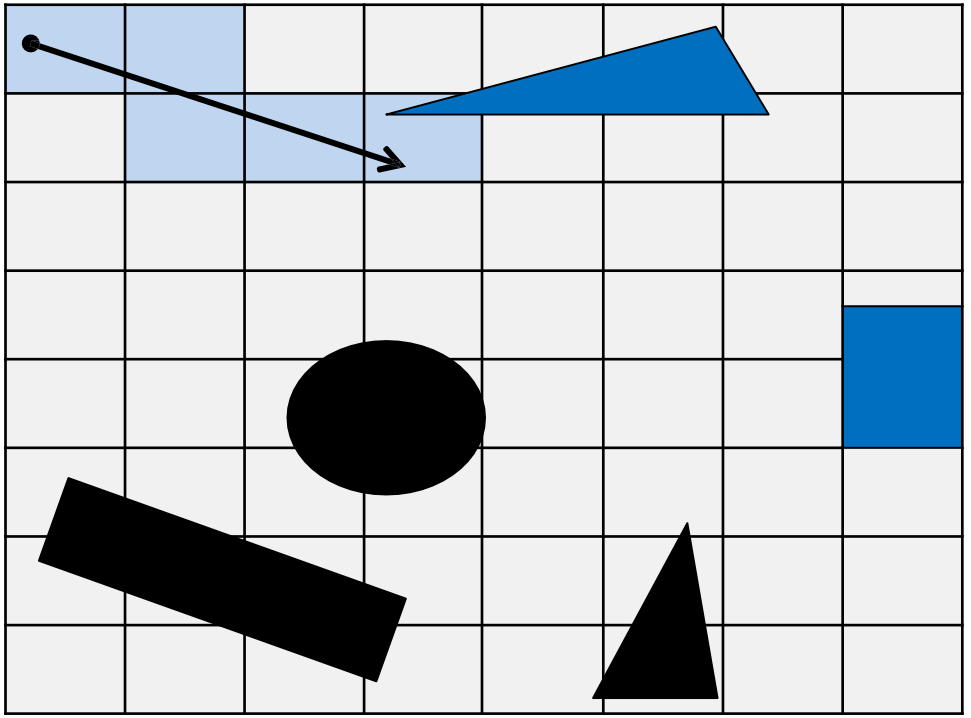


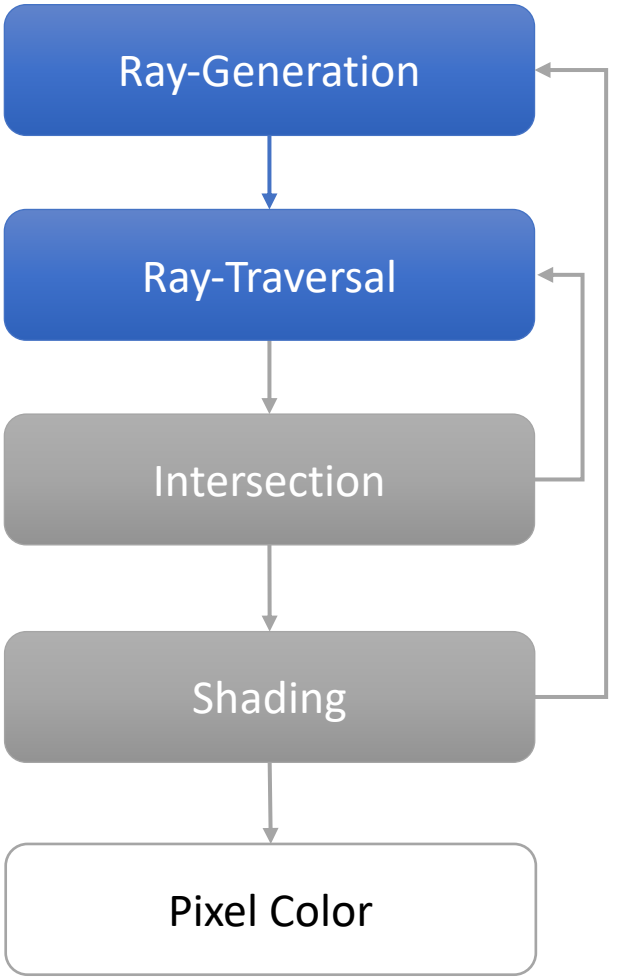
Ray Tracing Pipeline



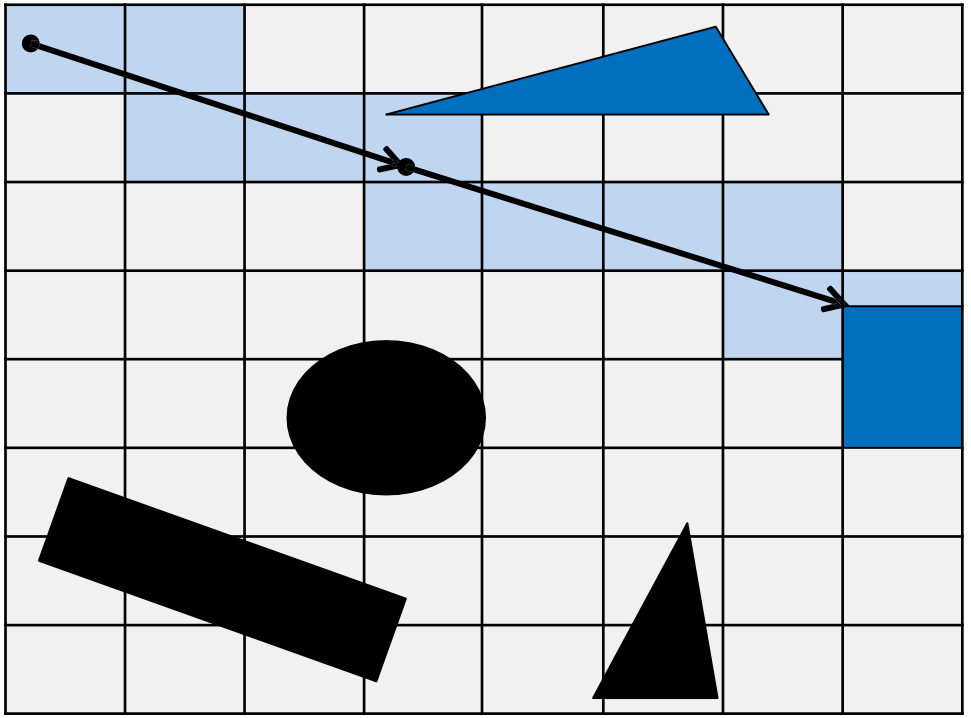


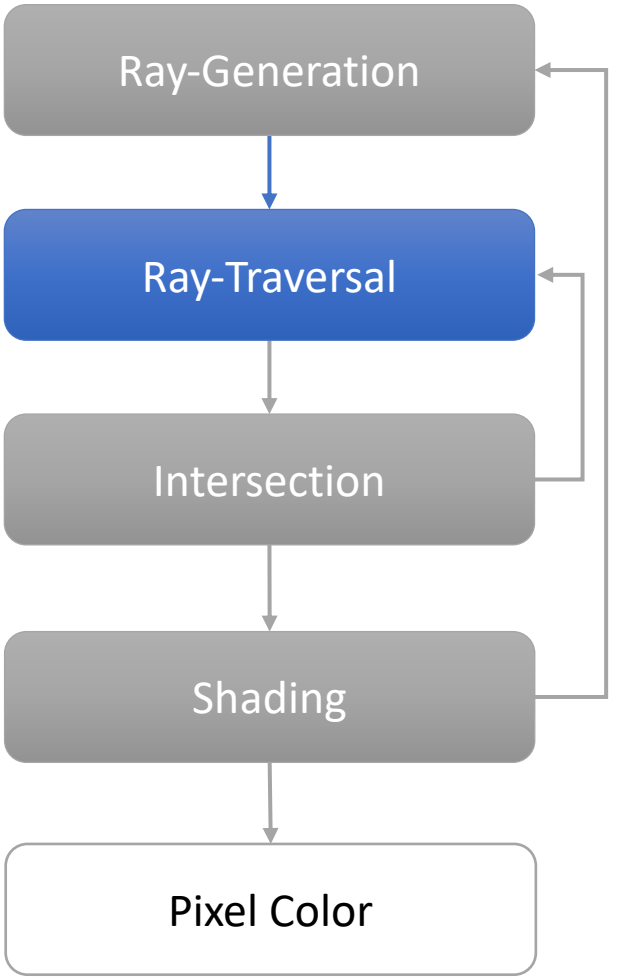
Ray Tracing Pipeline



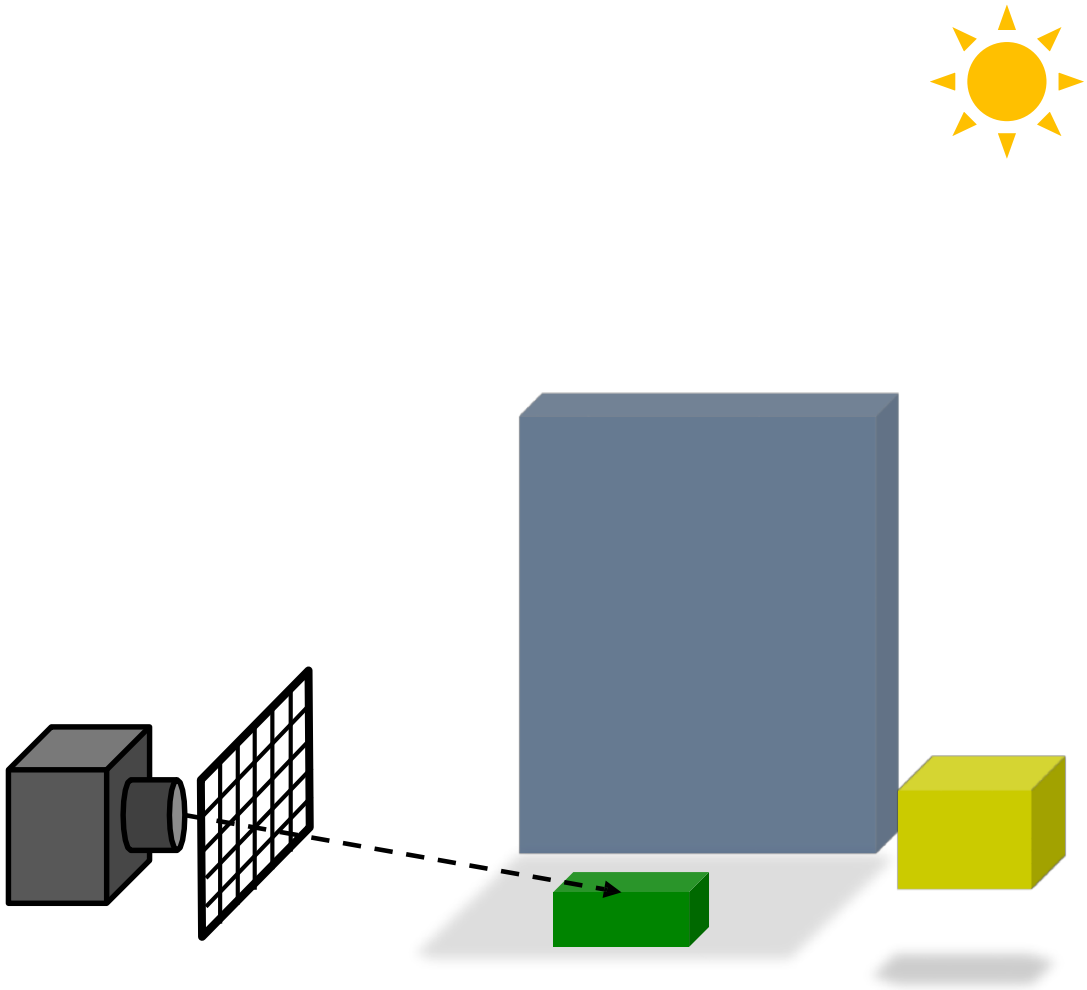


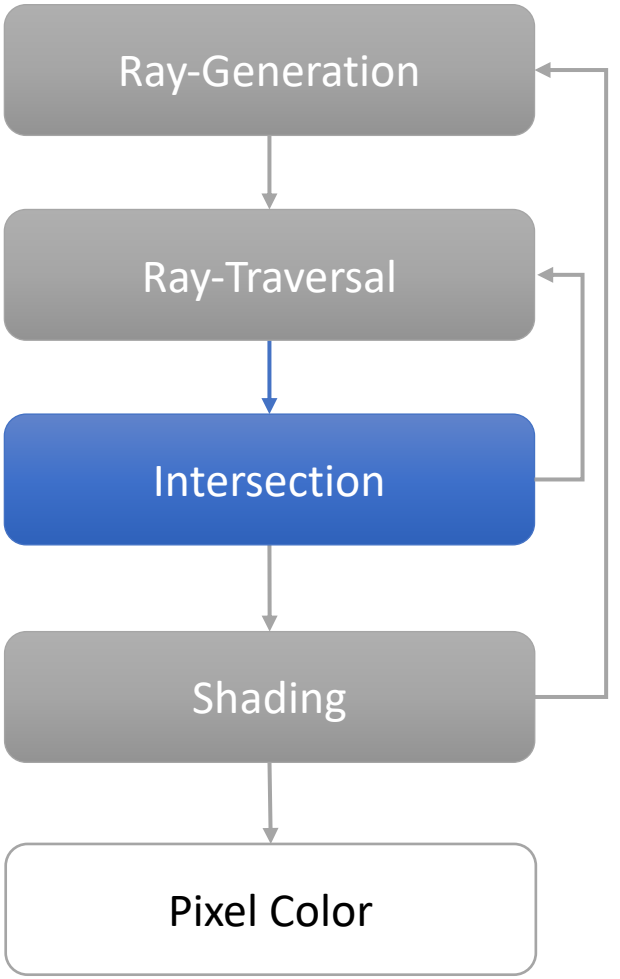
Ray Tracing Pipeline



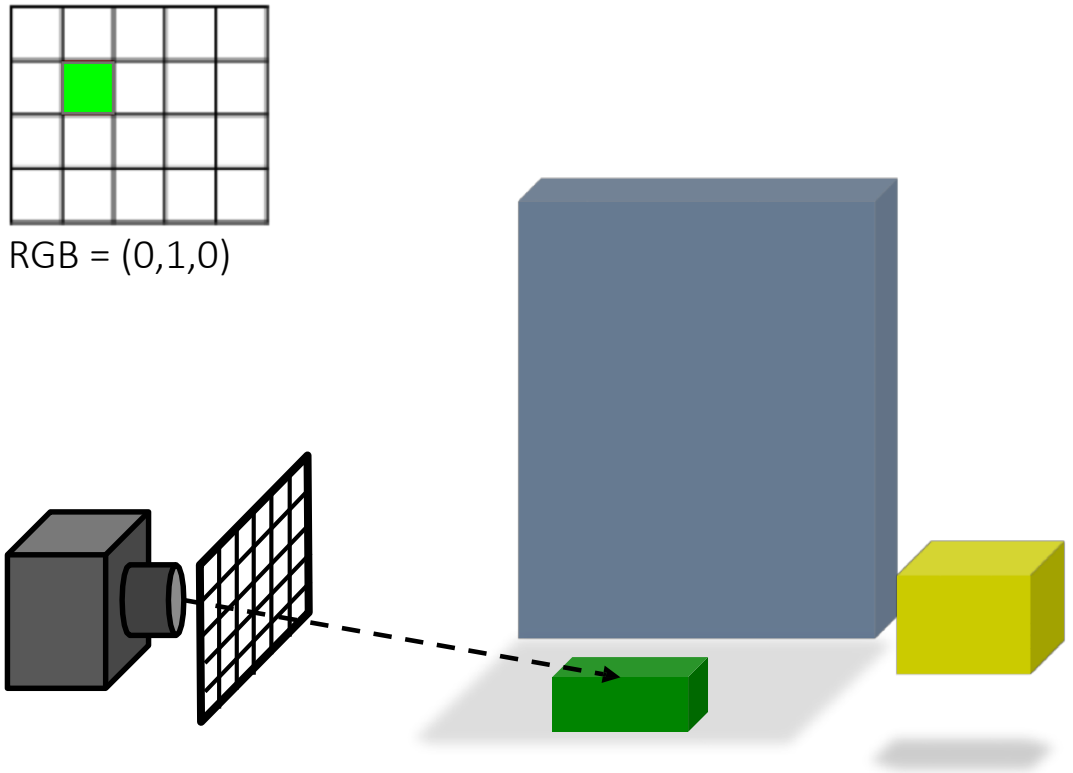


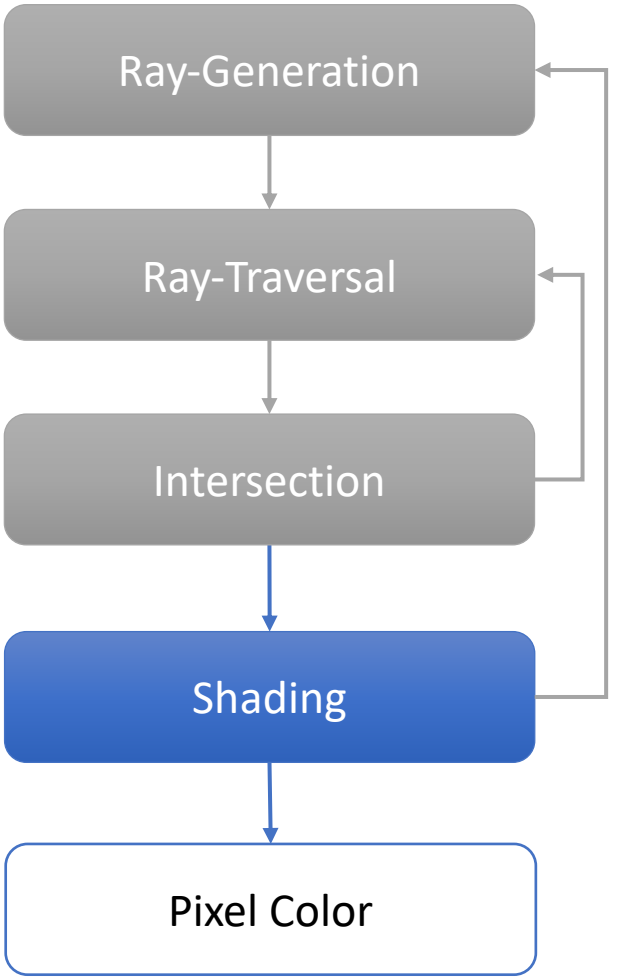
Ray Tracing Pipeline



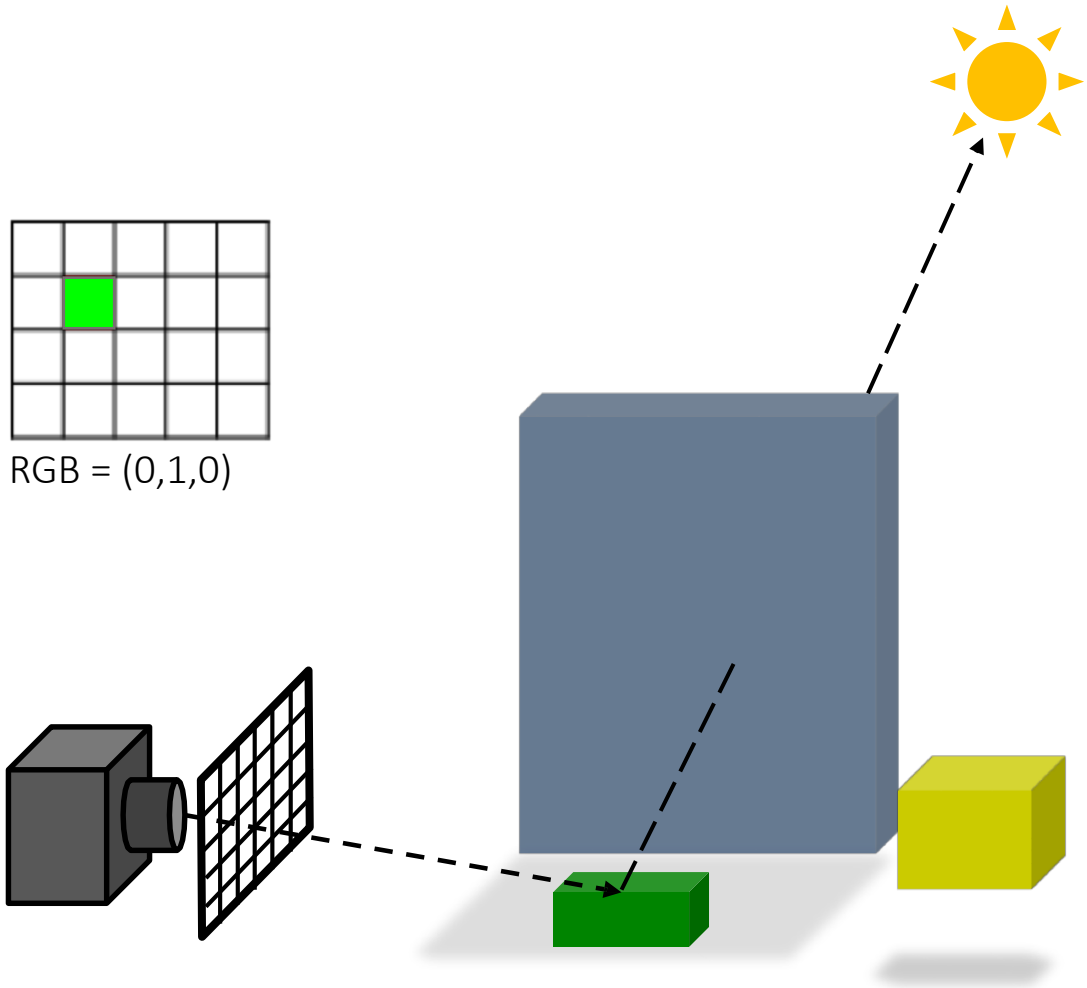


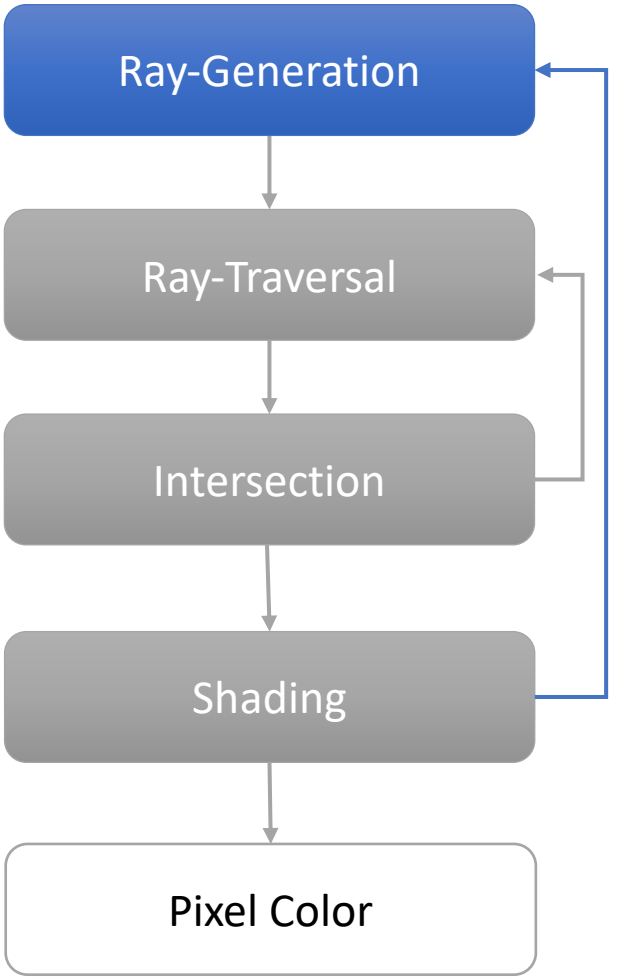
Ray Tracing Pipeline



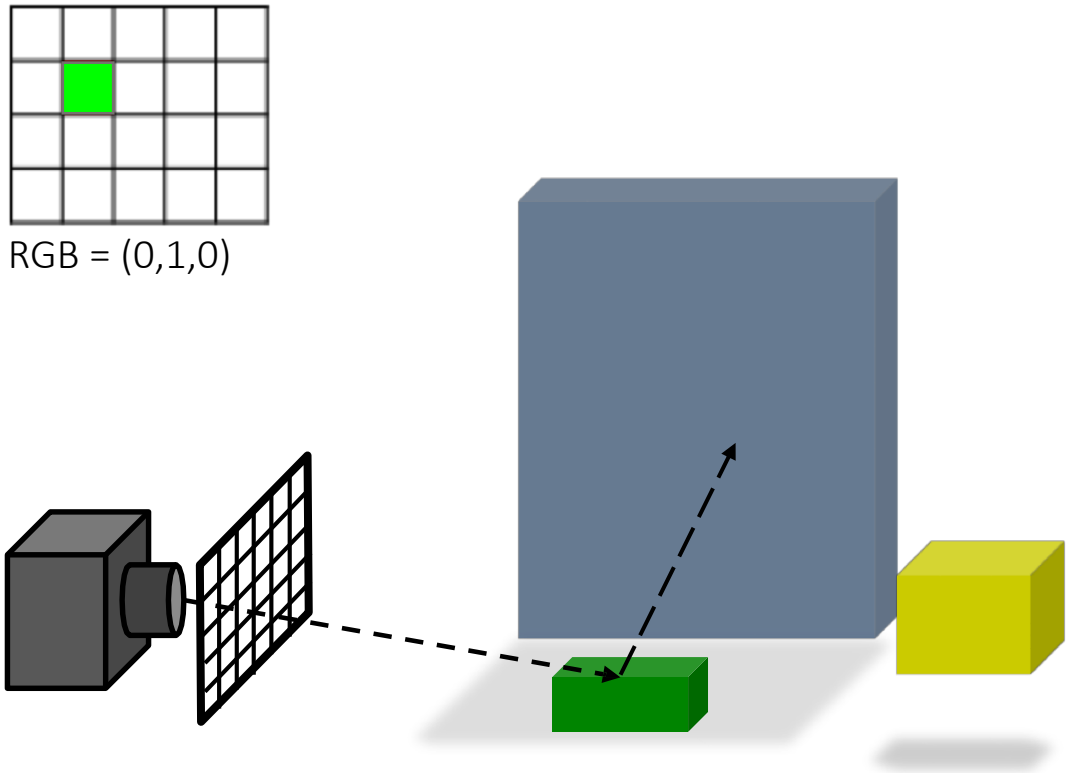


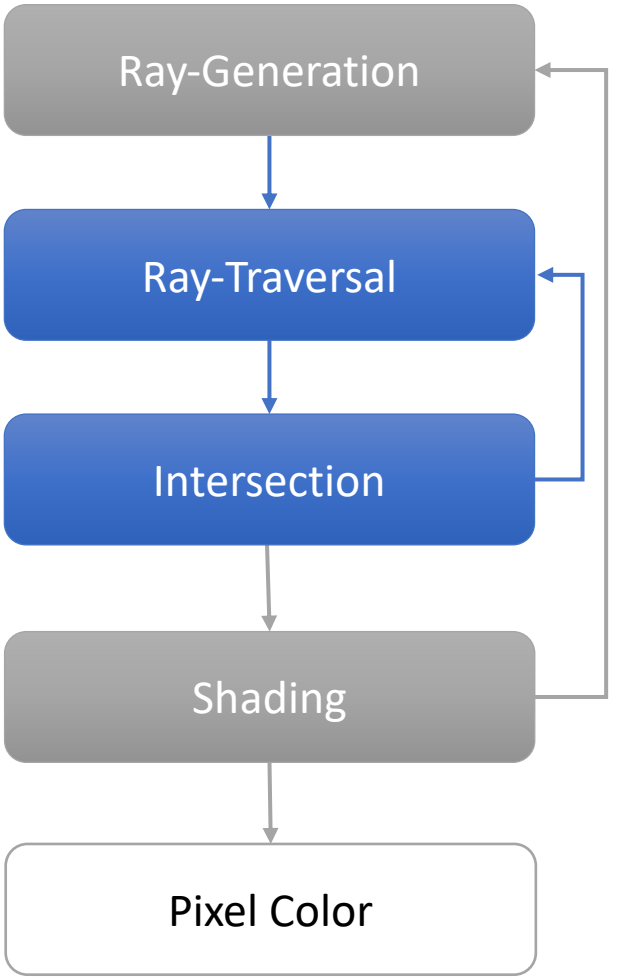
Ray Tracing Pipeline



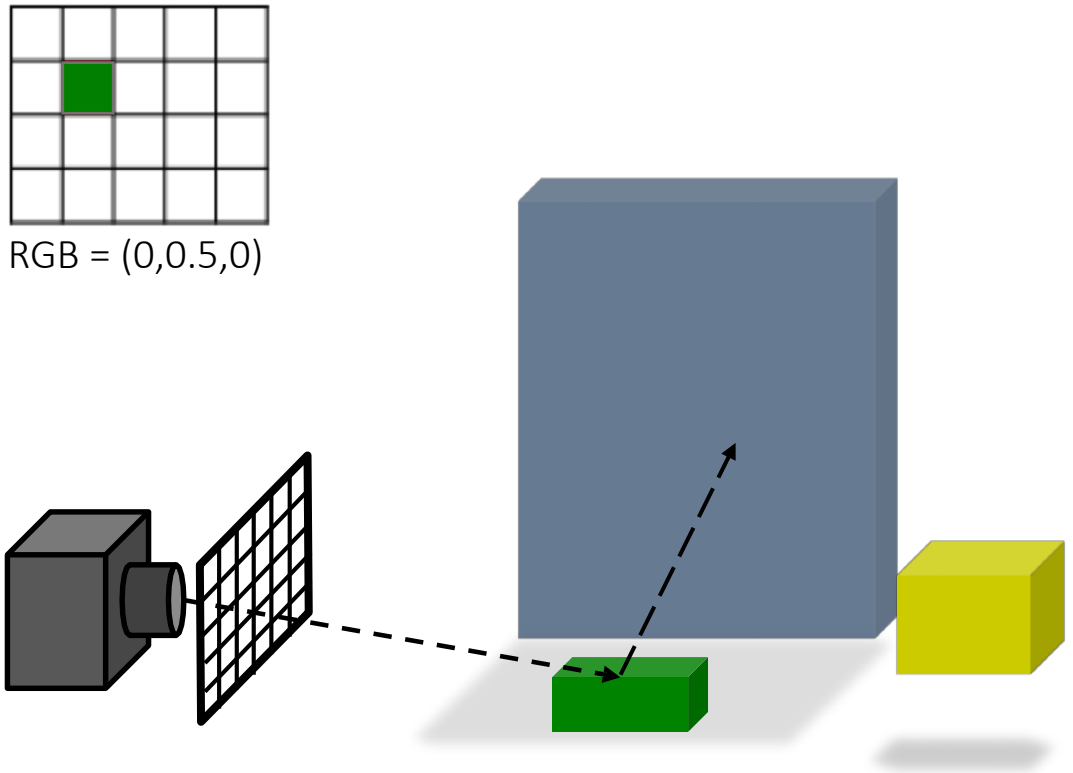


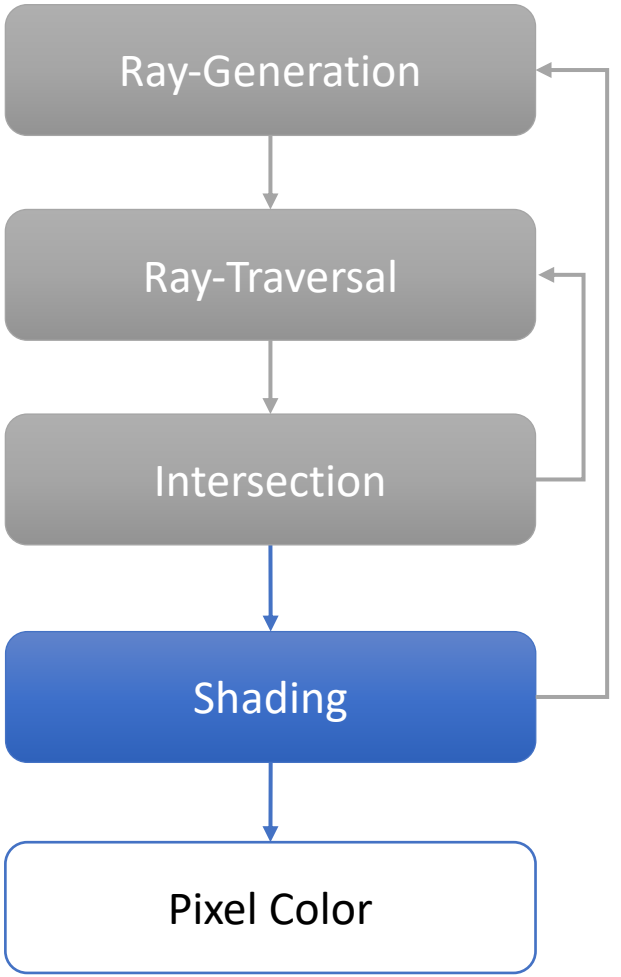
Ray Tracing Pipeline



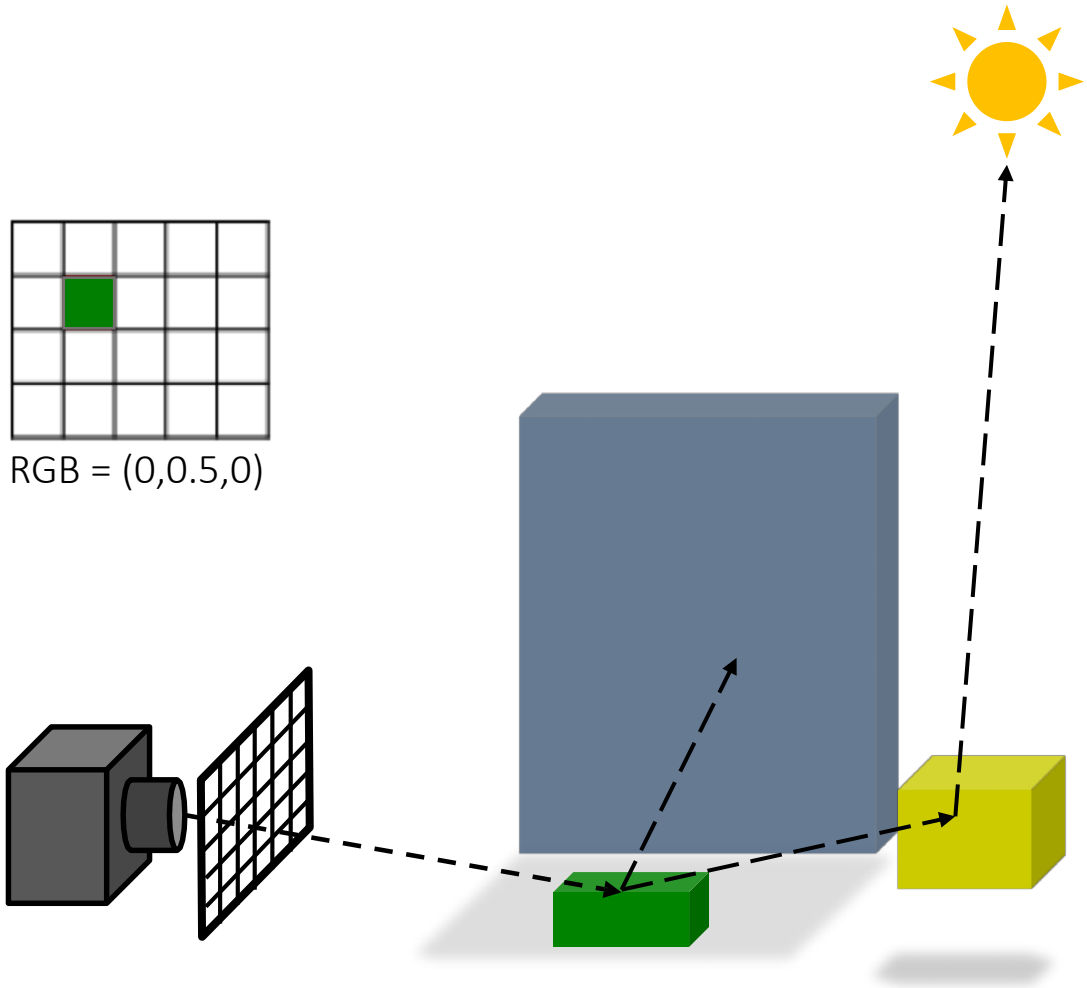


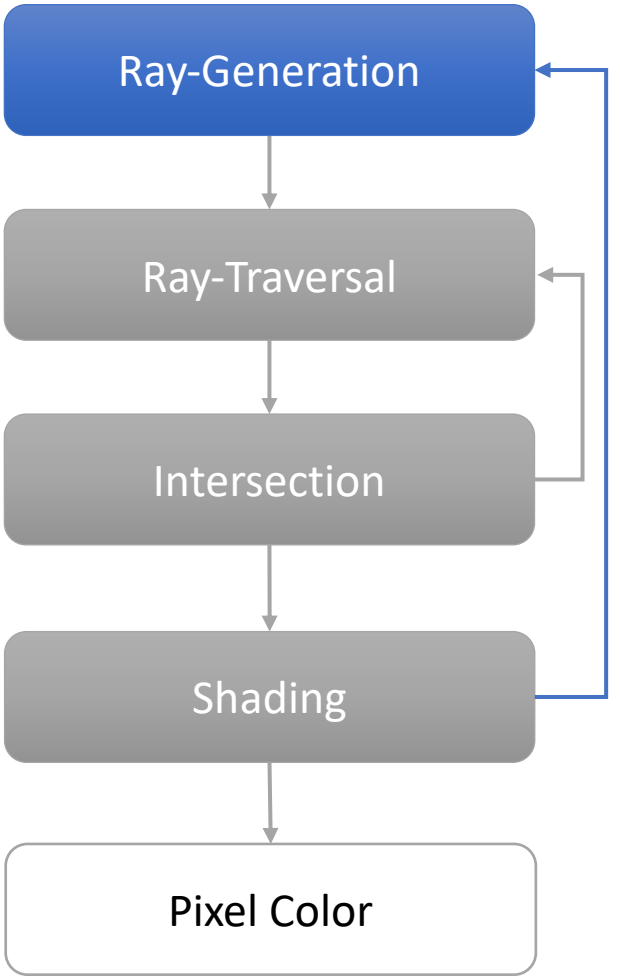
Ray Tracing Pipeline



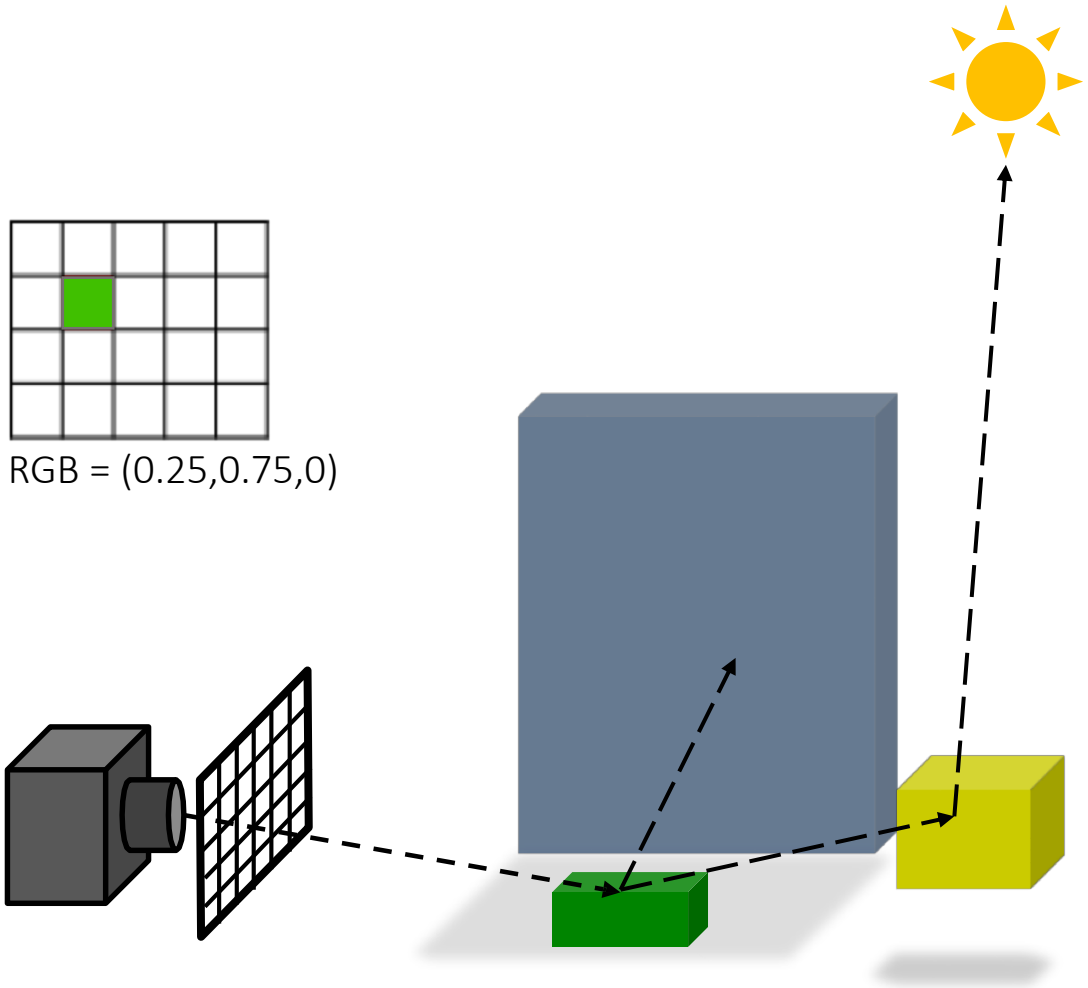


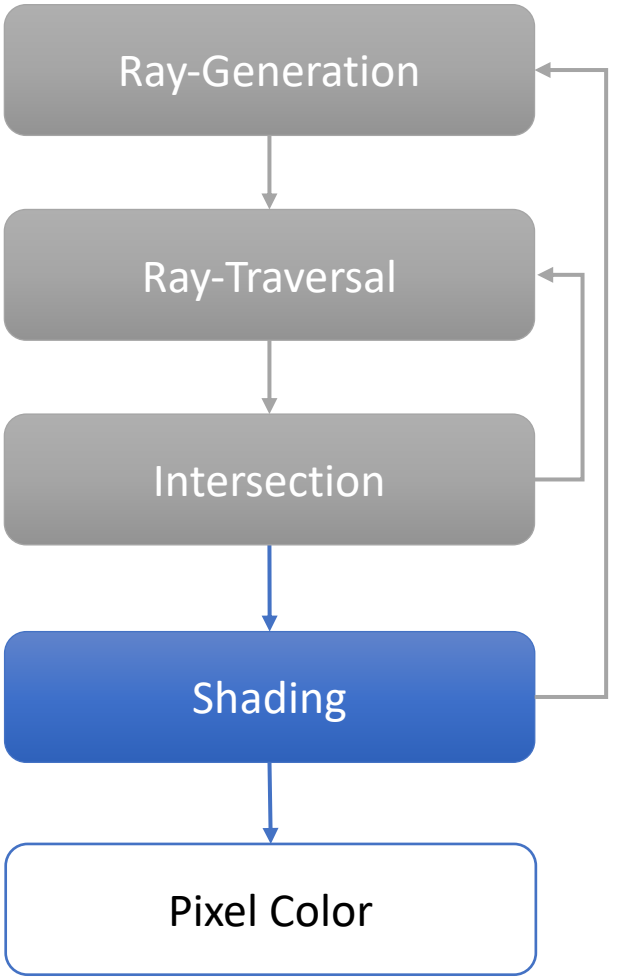
Ray Tracing Pipeline



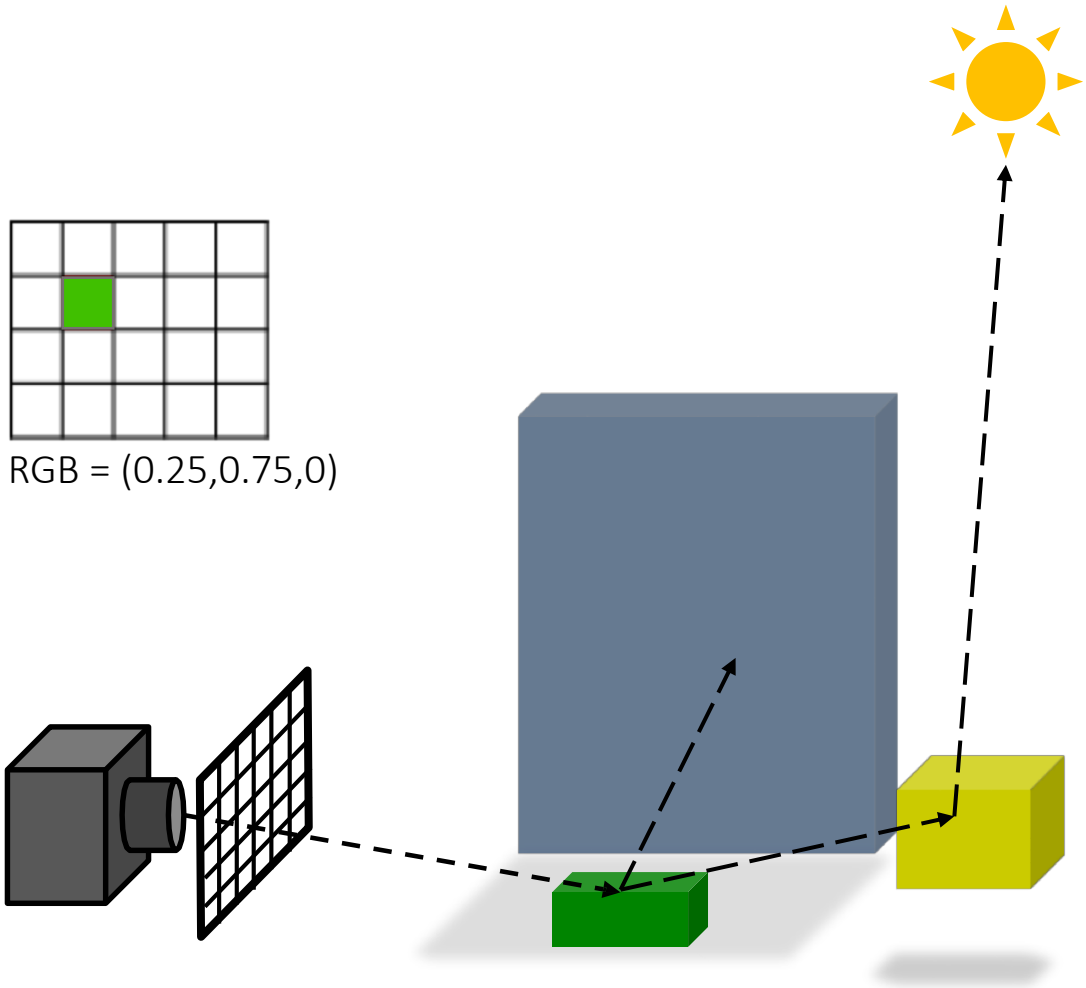


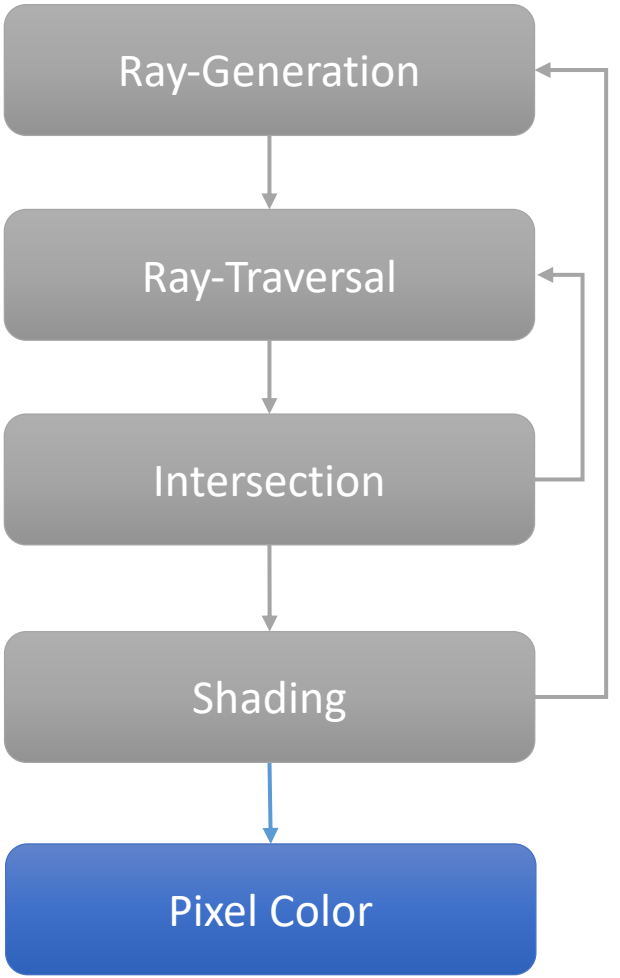
Ray Tracing Pipeline

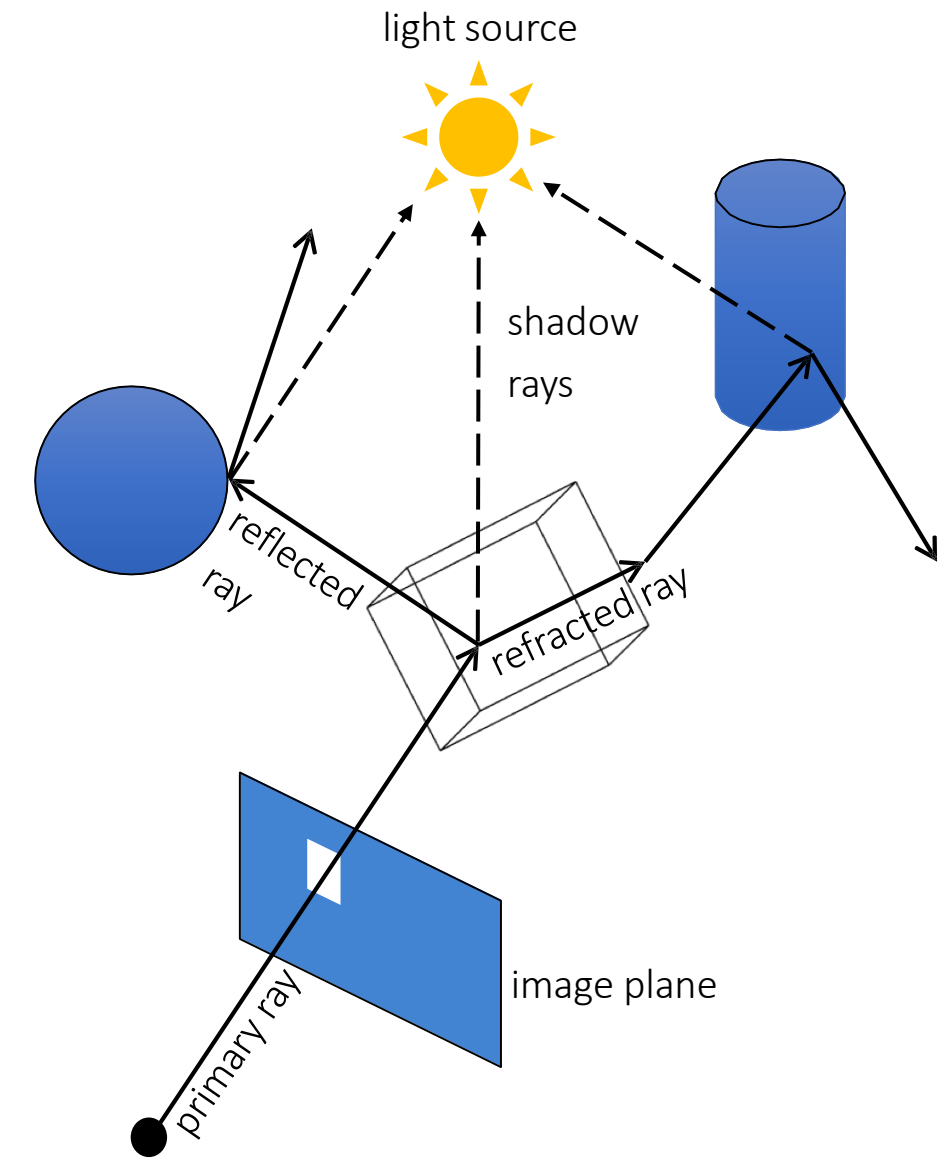




Ray Tracing Pipeline

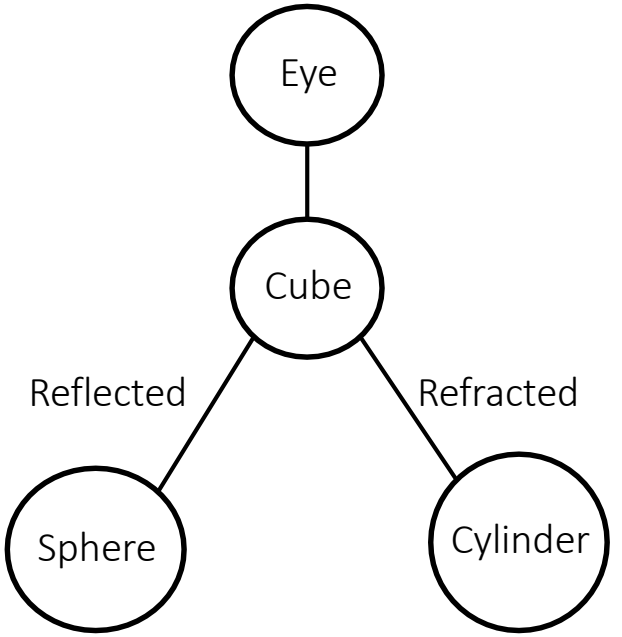






Searching recursively for paths to light sources

- Interaction of light & material at intersections
- Recursively trace new rays in reflection, refraction, and light direction



`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 radiance (*i.e.* Phong material model)
- Adding to the reflected radiance

`if (mirroring material)`

- Calculate radiance in reflected direction: `trace(R(ray, hit))`
- Adding mirroring part 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

Intersection point determines primary ray's "color"

- Diffuse object: color at intersection point
  - 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)

Illumination

- Point / directional light sources
- Area light sources
  - Approximate with multiple samples / shadow rays
- Indirect illumination
  - See Realistic Image Synthesis (RIS) course in next semester

More details later

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**: reflected uniformly in all directions perfect
- **Specular**: reflection, refraction

Used reflection models are often empirical

- Better physically accurate models are available (*e.g.* Ward model)

Ray Tracing Incorporates into a single framework

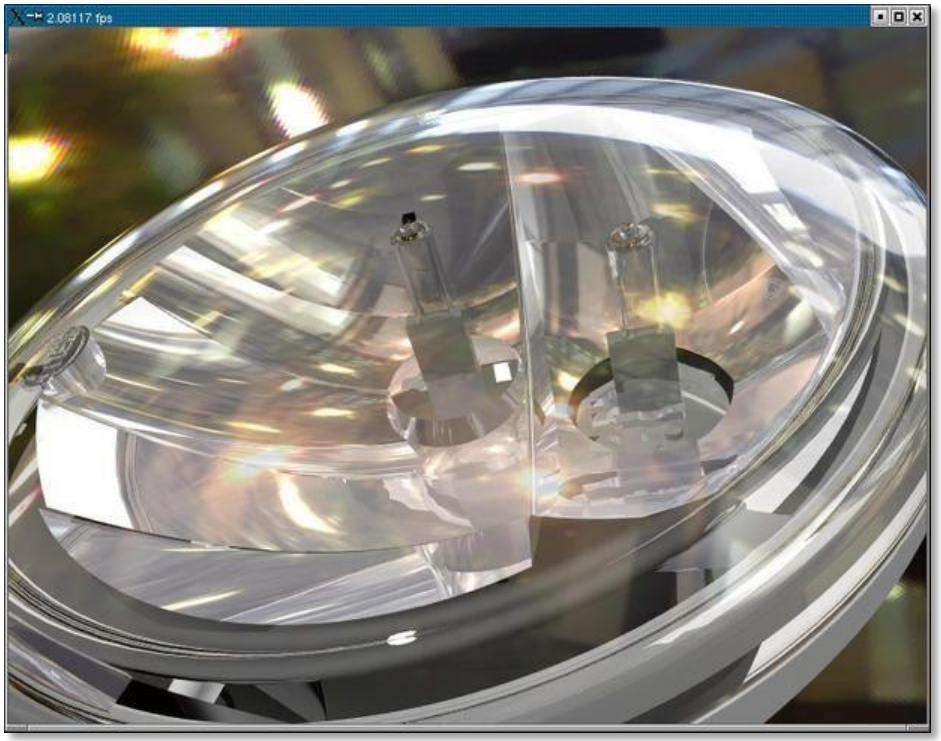
- Hidden surface removal
  - Front to back traversal
  - Early termination once first hit point is found
- Shadow computation
  - Shadow rays / shadow feelers are traced between a point on a surface and a light sources
- Exact simulation of some light paths
  - Reflection (reflected rays at a mirror 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

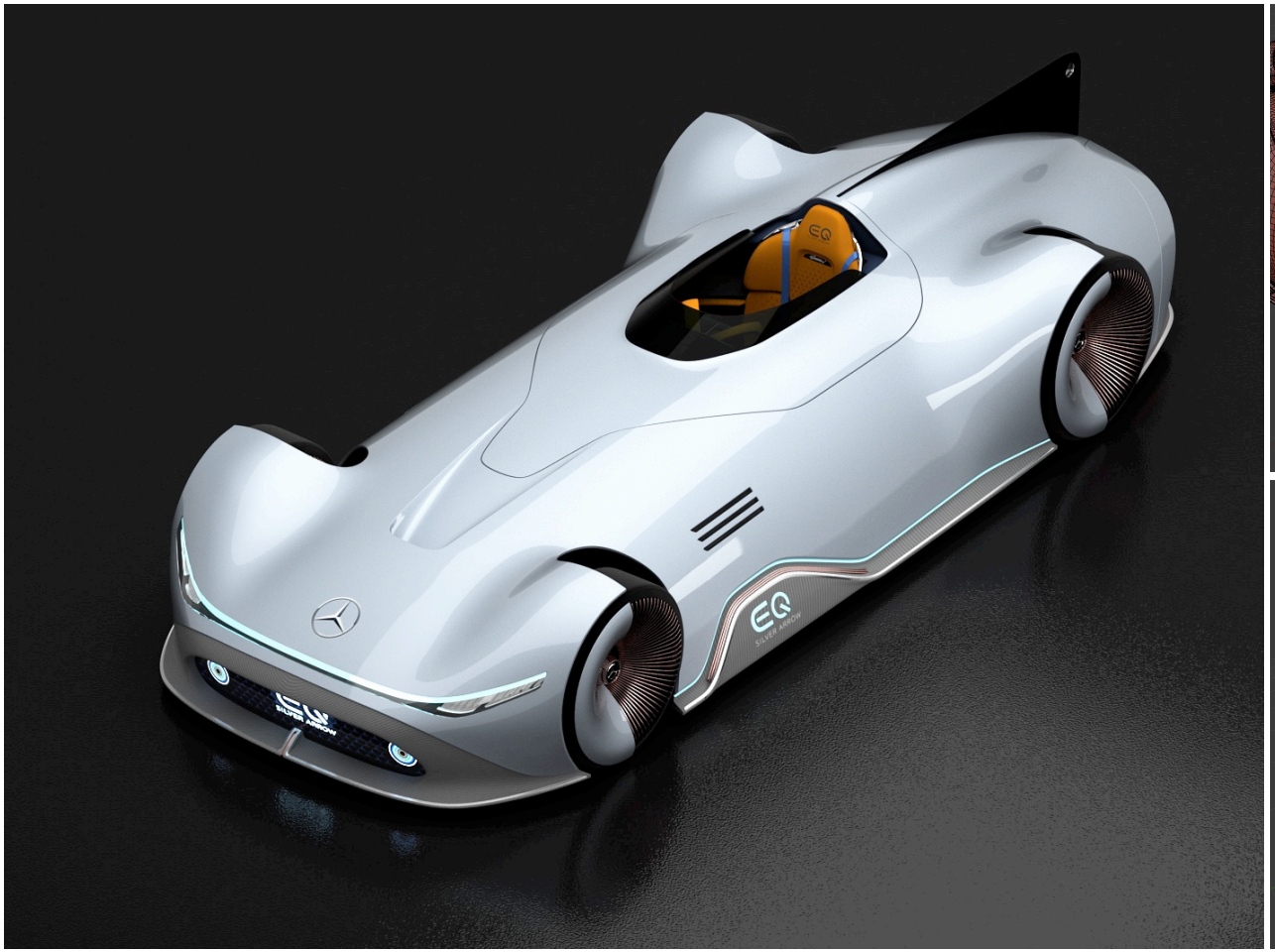
Models Physics of Global Light Transport

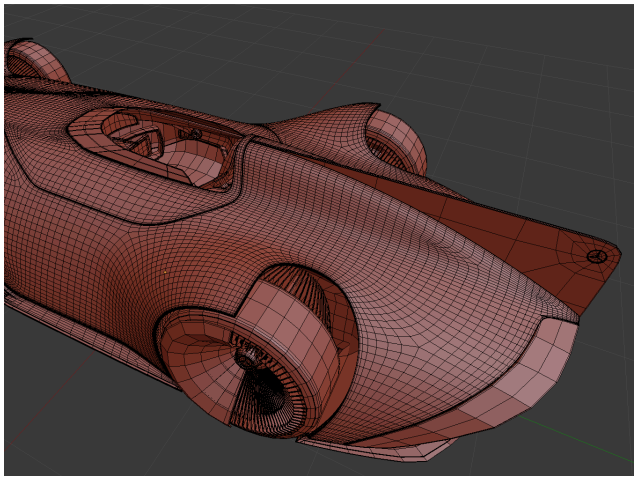
- Dependable, physically-correct visualization

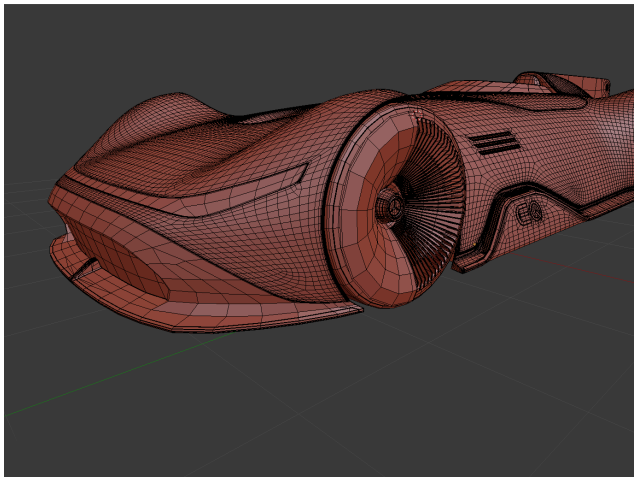


VW Visualization Center

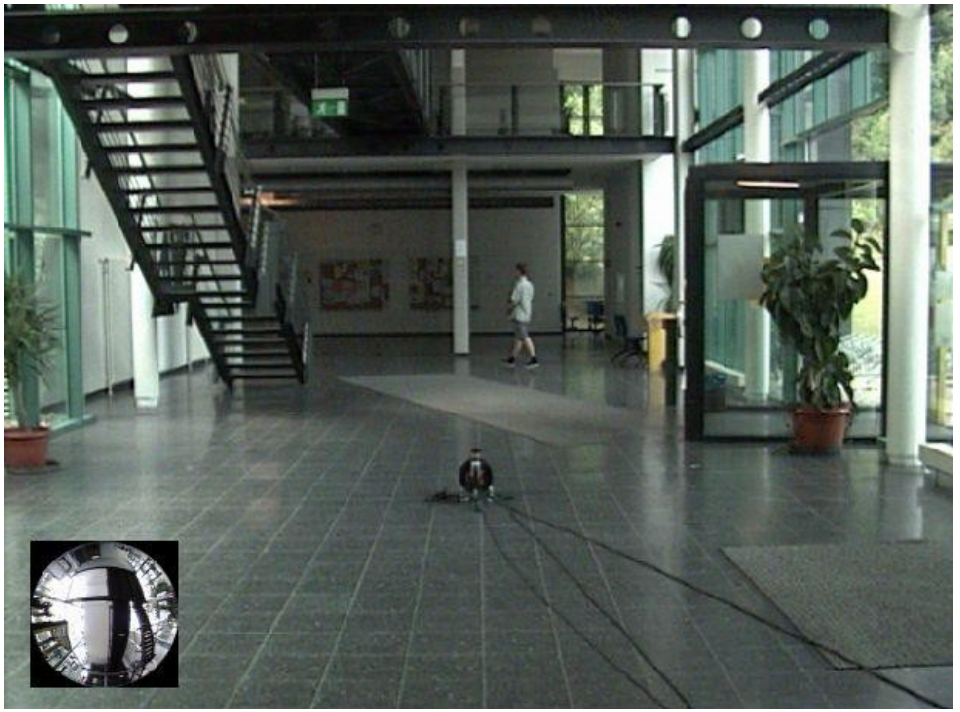
Realistic Visualization: CAD







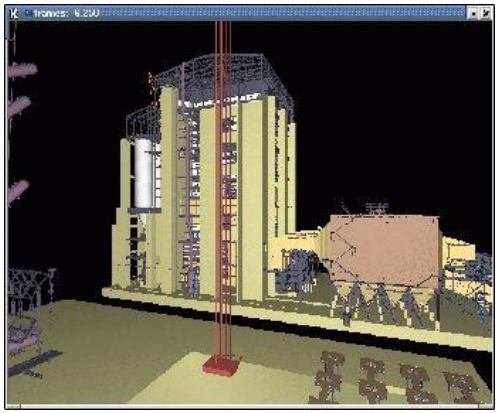
Realistic Visualization: VR / AR

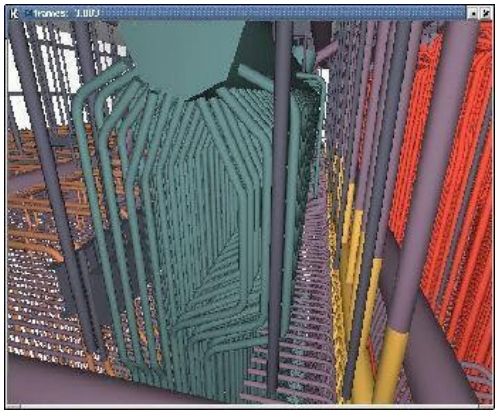


Lighting Simulation

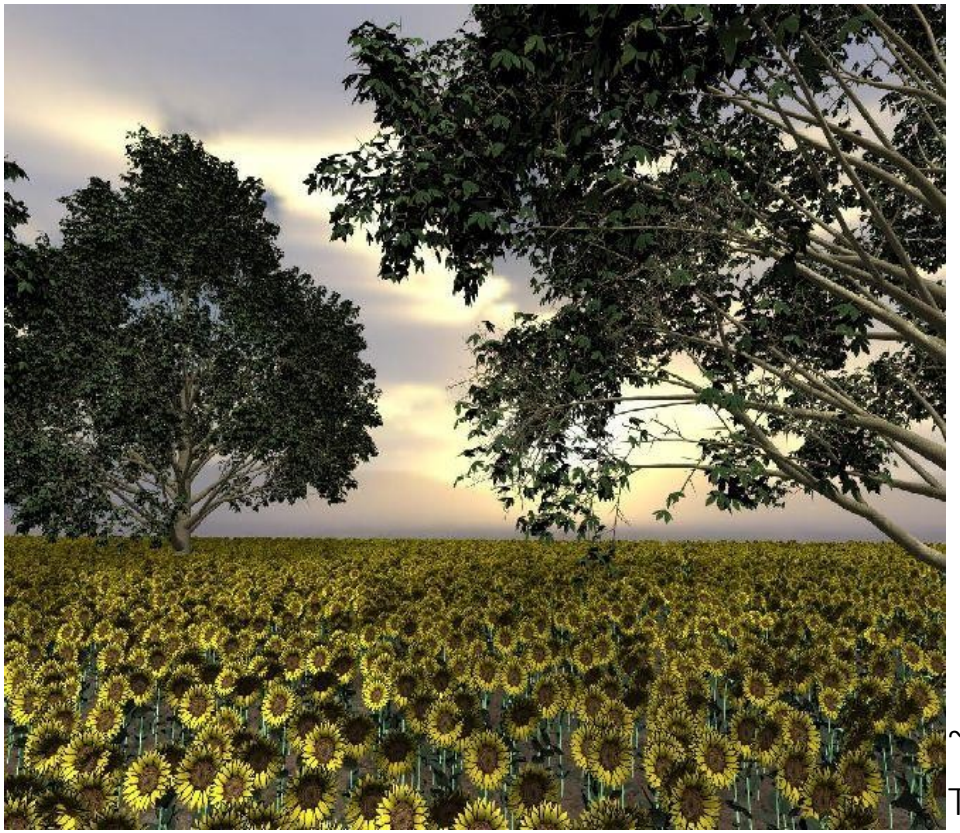
Huge Models

- Logarithmic scaling in scene size





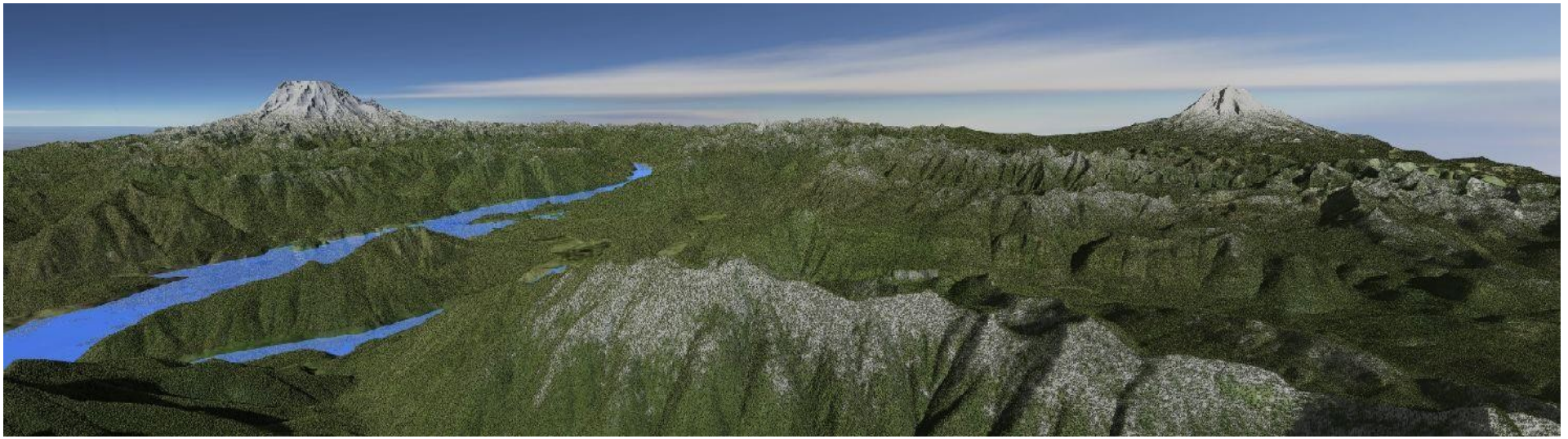
12.5 Million Triangles

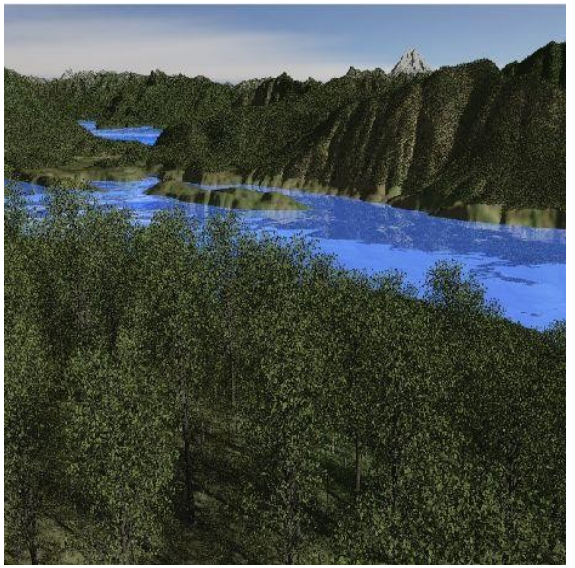


~1 Billion Triangles

Outdoor Environments

- 90 x 1012 (trillion) triangles





Games

In the Past

- Only used as an off-line technique
- Was computationally far too demanding (minutes to hours per frame)
- Believed to not be suitable for a HW implementation

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 / OSPRey (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, OSS),Mitsuba renderer (EPFL), imbatracer (SB), …
- Commercial: V-Ray (Chaos Group), Corona (Render Legion), VRED (Autodesk), MentalRay/iRay (MI), …

Tracing/Casting a ray

- Type of query
  - “Is there a primitive along a ray”
  - “How far is the closest primitive”

Other uses than rendering

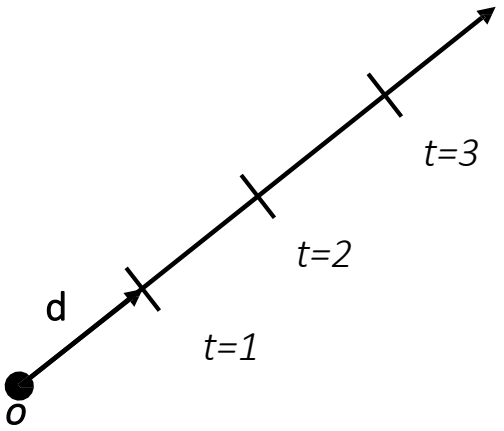
- Volume computation
- Sound waves tracing
- Collision detection
- …

Ray parameterization

- $\vec{r}(t) = \vec{o} + t\vec{d}$
  - Origin: $\vec{o} \in \mathbb{R}^3$
  - Direction: $\vec{d} \in \mathbb{R}^3$

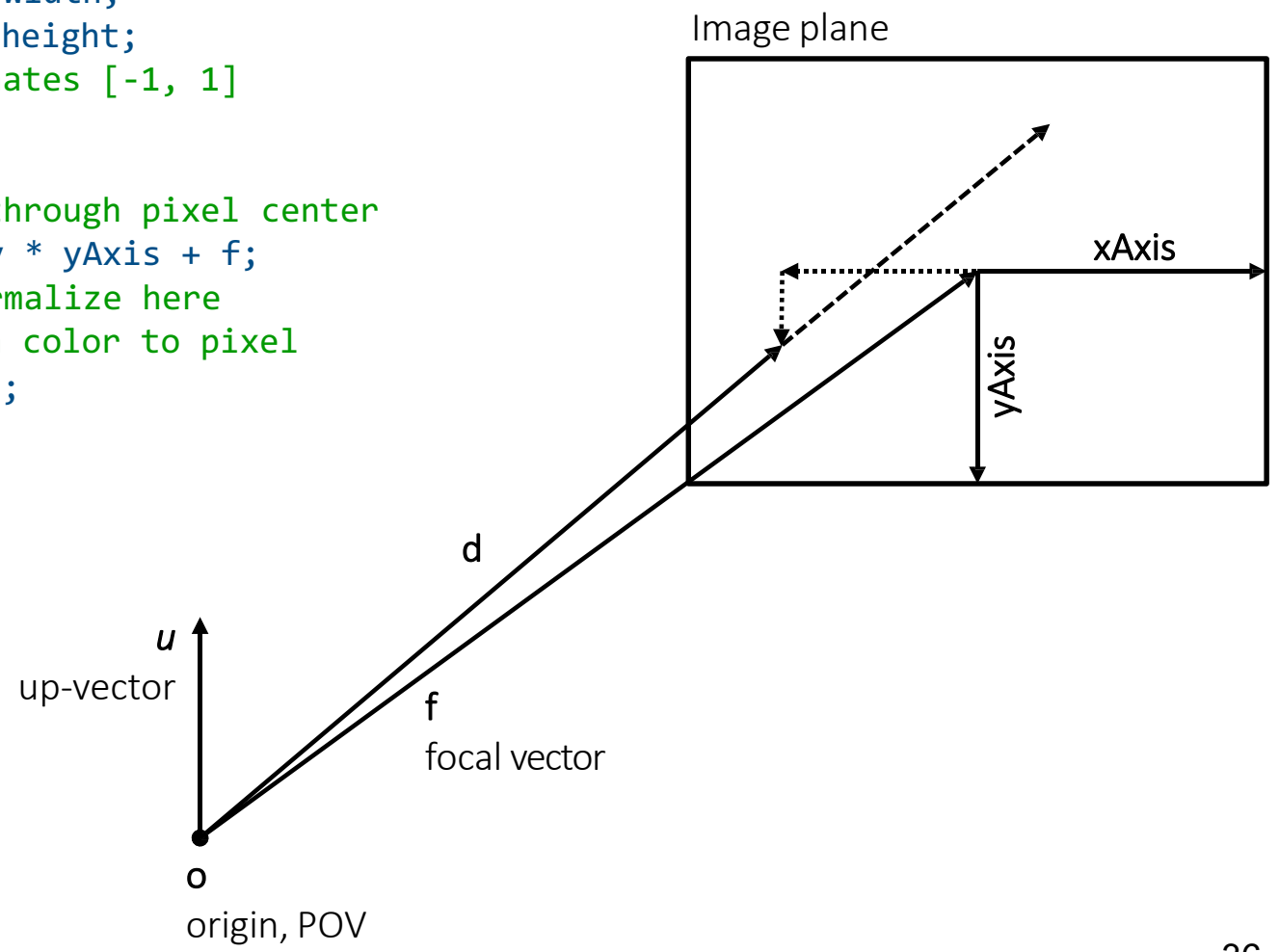
Ray

- All points on the graph of $\vec{r}(t)$, with $t \in \mathbb{R}$;



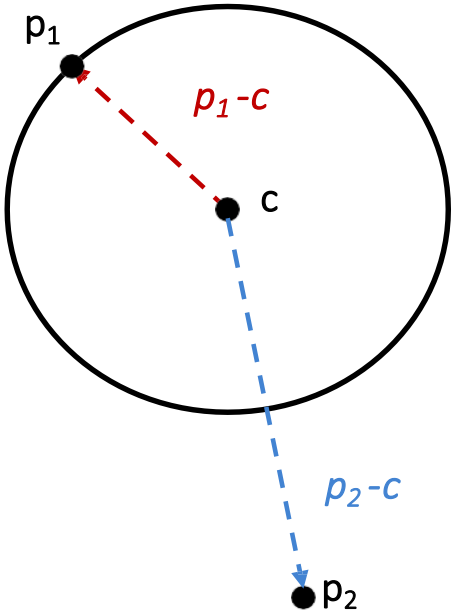
Perspective Camera Model

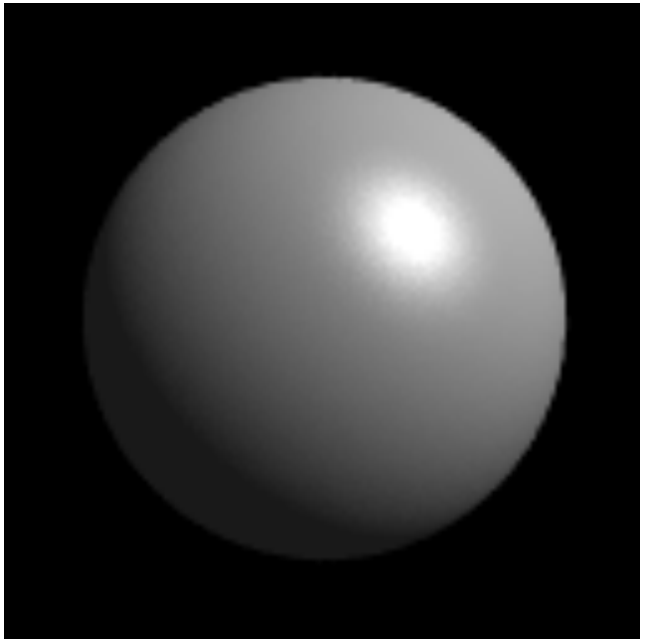
```
// For given image resolution res
// Loop over pixel raster coordinates
for(int y = 0; y < res.height; y++)
   for(int x = 0; x < res.width; x++)
   {
      // Normalized device coordinates [0, 1]
      ndcx = (x + 0.5) / res.width;
      ndcy = (y + 0.5) / res.height;
      // Screen space coordinates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



Sphere S

- $\forall \vec{p} \in \mathbb{R}^3$: $\vec{p} \in S \Leftrightarrow (\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) = r^2$
- Sphere center: $\vec{c} \in \mathbb{R}^3$
- Sphere radius: $r \in \mathbb{R}$





Given

- Ray: $\vec{r}(t) = \vec{o} + t\vec{d}$
- Sphere: $\forall \vec{p} \in \mathbb{R}^3: \ \vec{p} \in S \Leftrightarrow (\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) = r^2$

Find closest intersection point

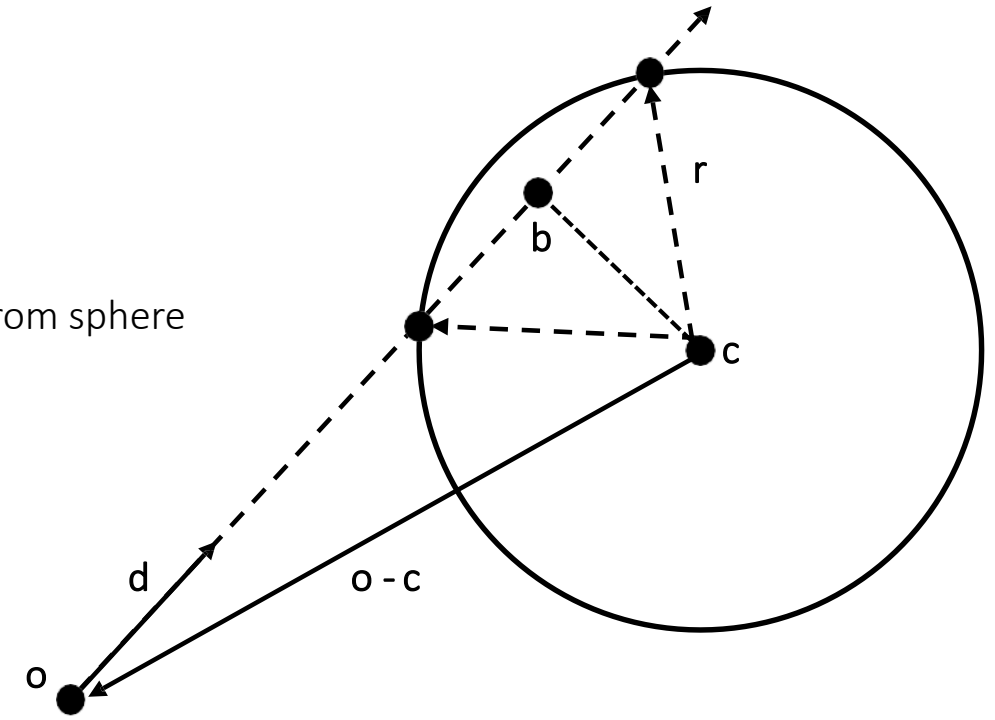
- **Algebraic approach**: substitute ray equation
  - $(\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) - r^2 = 0$ with $\vec{p} = \vec{o} + t\vec{d}$
  - $t^2\vec{d} \cdot \vec{d} + 2t\vec{d} \cdot (\vec{o} - \vec{c}) + (\vec{o} - \vec{c}) \cdot (\vec{o} - \vec{c}) - r^2 = 0$
  - Solve for t

Given

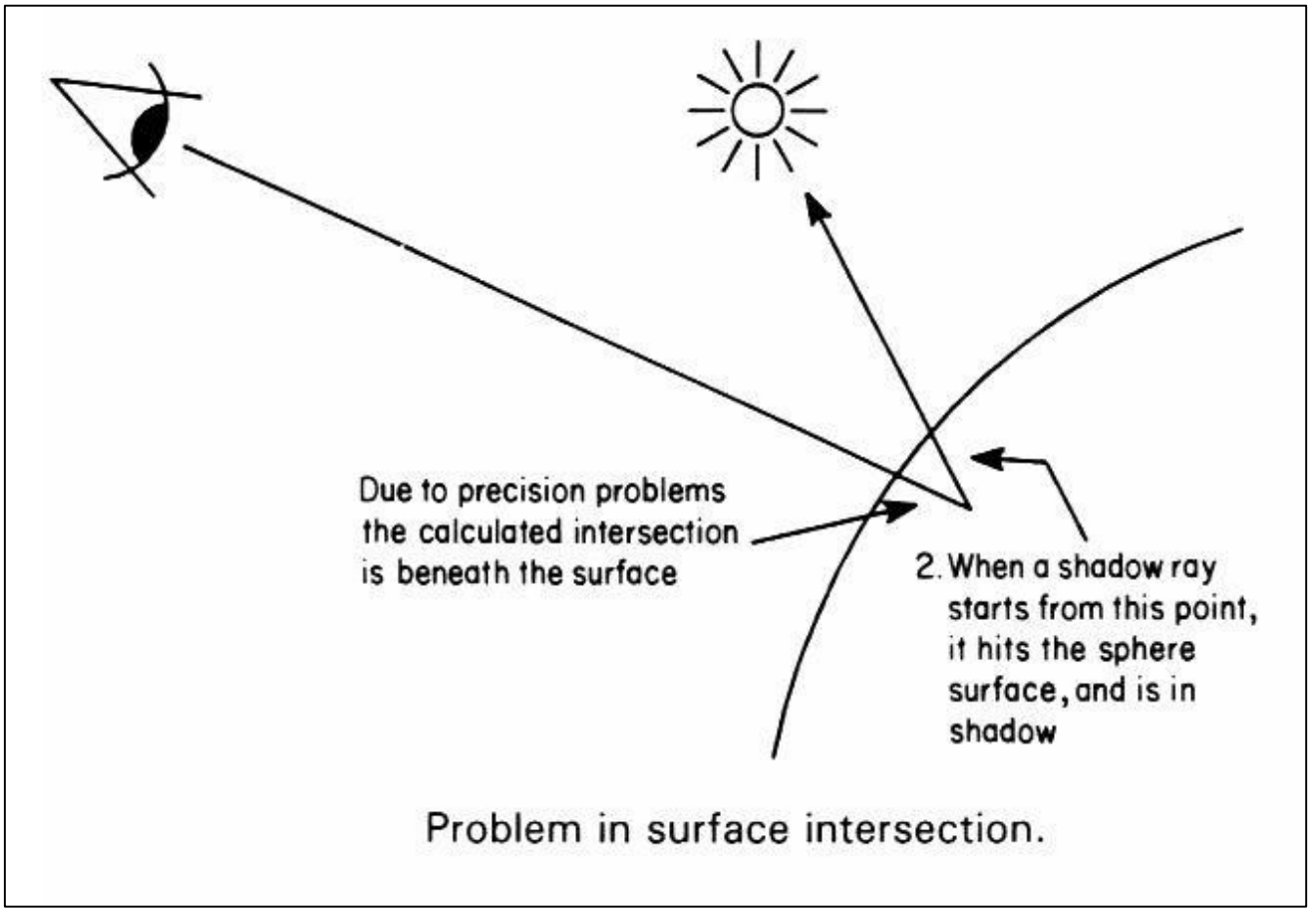
- Ray: $\vec{r}(t) = \vec{o} + t\vec{d}$
- Sphere: $\forall \vec{p} \in \mathbb{R}^3: \ \vec{p} \in S \Leftrightarrow (\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) = r^2$

Find closest intersection point

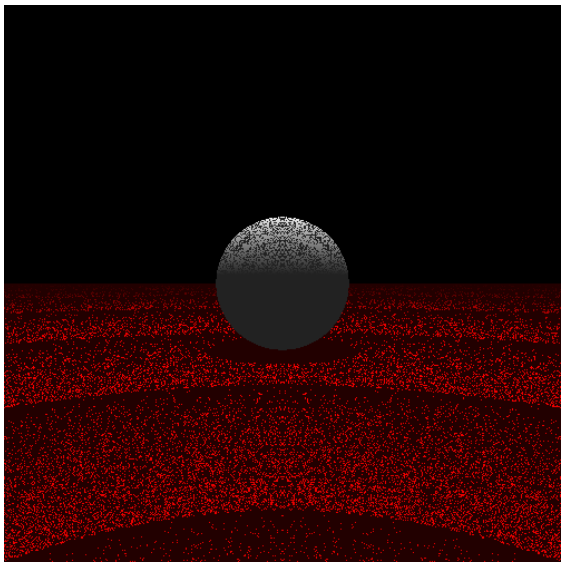
- Geometric approach:
  - Ray and center span a plane
  - Solve in 2D
  - Compute $|\vec{b} - \vec{c}|$, $|\vec{b} - \vec{c}|$
    - $\measuredangle obc = 90°$
  - Intersection(s) if $|\vec{b} - \vec{c}| \leq r$
- Be aware of floating point issue if o is far from sphere



Cause of „surface acne“

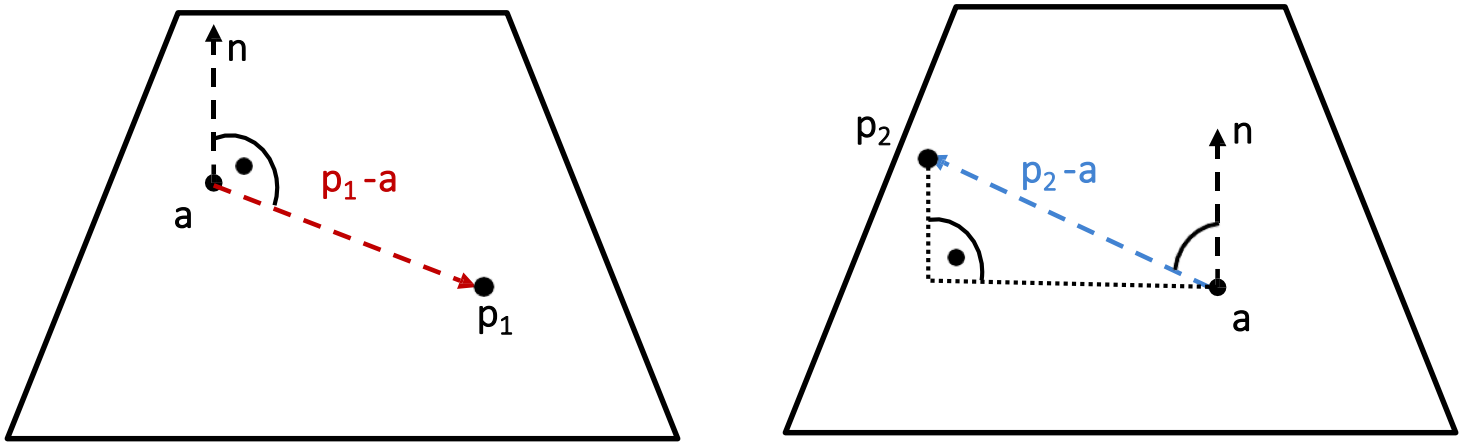


Problem in surface intersection.



Plane P

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



Given

- Ray: $\vec{r}(t) = \vec{o} + t\vec{d}$
- Plane: $\forall \vec{p} \in \mathbb{R}^3: \quad \vec{p} \in P \Leftrightarrow (\vec{p} - \vec{a}) \cdot \vec{n} = 0$

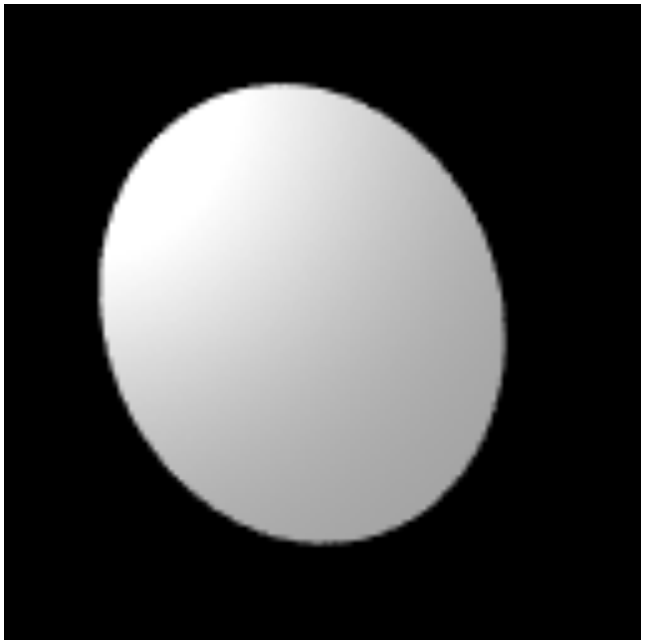
Compute intersection point

- Plane equation: $\vec{p} \in P \Leftrightarrow (\vec{p} - \vec{a}) \cdot \vec{n} = 0$

  $\Leftrightarrow \vec{p} \cdot \vec{n} - D = 0$, with $D = \vec{a} \cdot \vec{n}$
- Substitute ray parameterization: $\left(\vec{o} + t\vec{d}\right) \cdot \vec{n} - D = 0$
- Solve for t
  - 0, 1 or infinitely many solutions

Intersect ray with plane

- Discard intersection if $\|\vec{p} - \vec{a}\| > r$

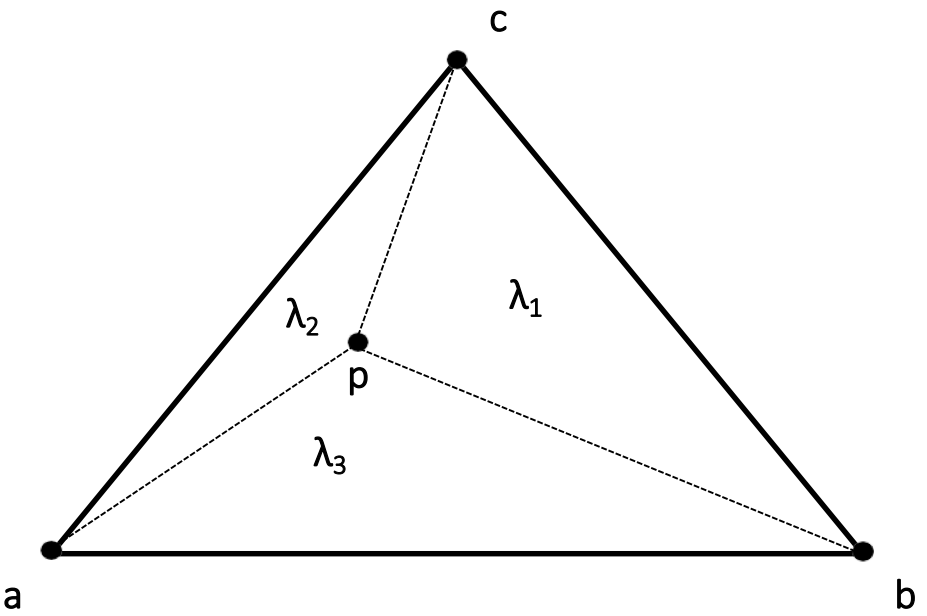


Triangle T

- Vertices: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$
- Affine combinations of $\vec{a}, \vec{b}, \vec{c}$ → points in the plane
  - Non-negative coefficients that sum up to 1 → points in the triangle
- $\forall \vec{p} \in \mathbb{R}^3$: $\vec{p} \in T \Leftrightarrow \exists \lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}, \lambda_1 + \lambda_2 + \lambda_3 = 1$ and
$$\vec{p} = \lambda_1 \vec{a} + \lambda_2 \vec{b} + \lambda_3 \vec{c}$$

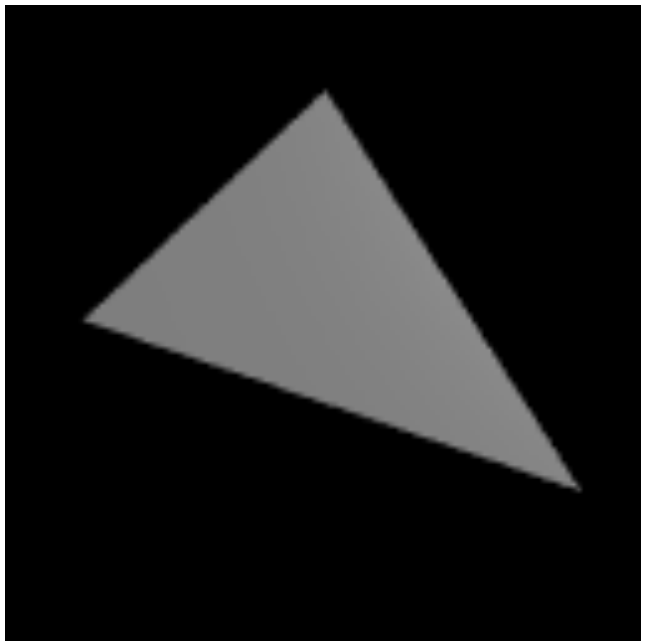
Barycentric coordinates

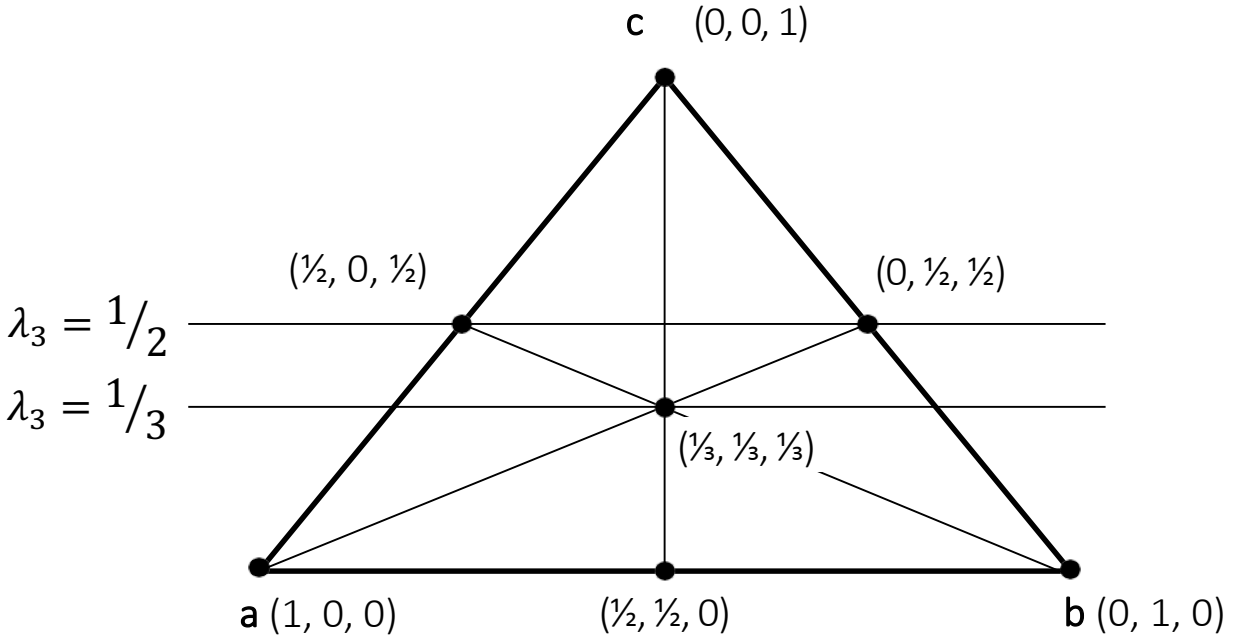
- $\lambda_1, \lambda_2, \lambda_3$
- $\lambda_1 = {}^{S_{pbc}}/_{S_{abc}}$
- S: signed area of triangles



Triangle T

- Vertices: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$
- Barycentric coordinates: $\lambda_1, \lambda_2, \lambda_3$
- $\lambda_1 + \lambda_2 + \lambda_3 = 1$
- $\lambda_1 = {}^{S_{pbc}}/_{S_{abc}}$, *etc*



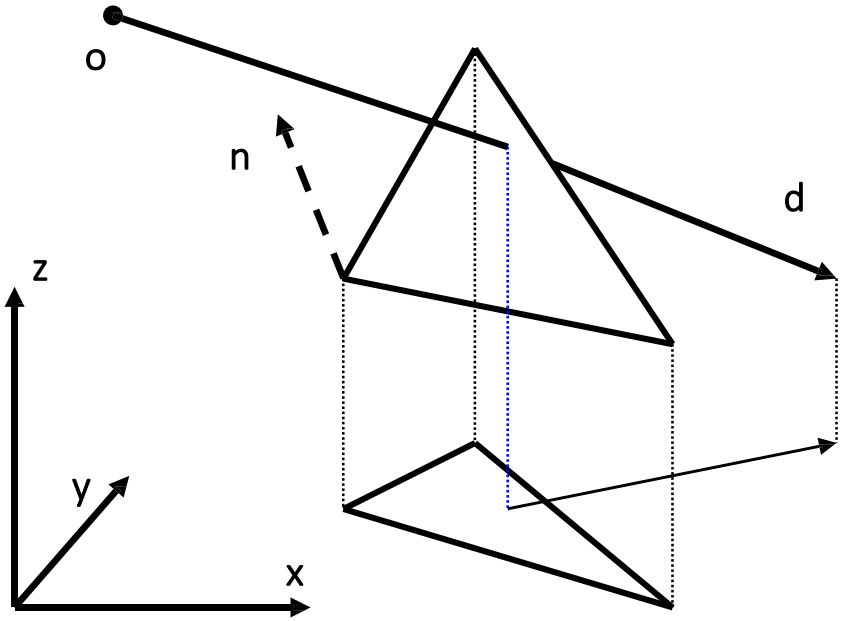


Compute intersection with triangle plane

Compute barycentric coordinates

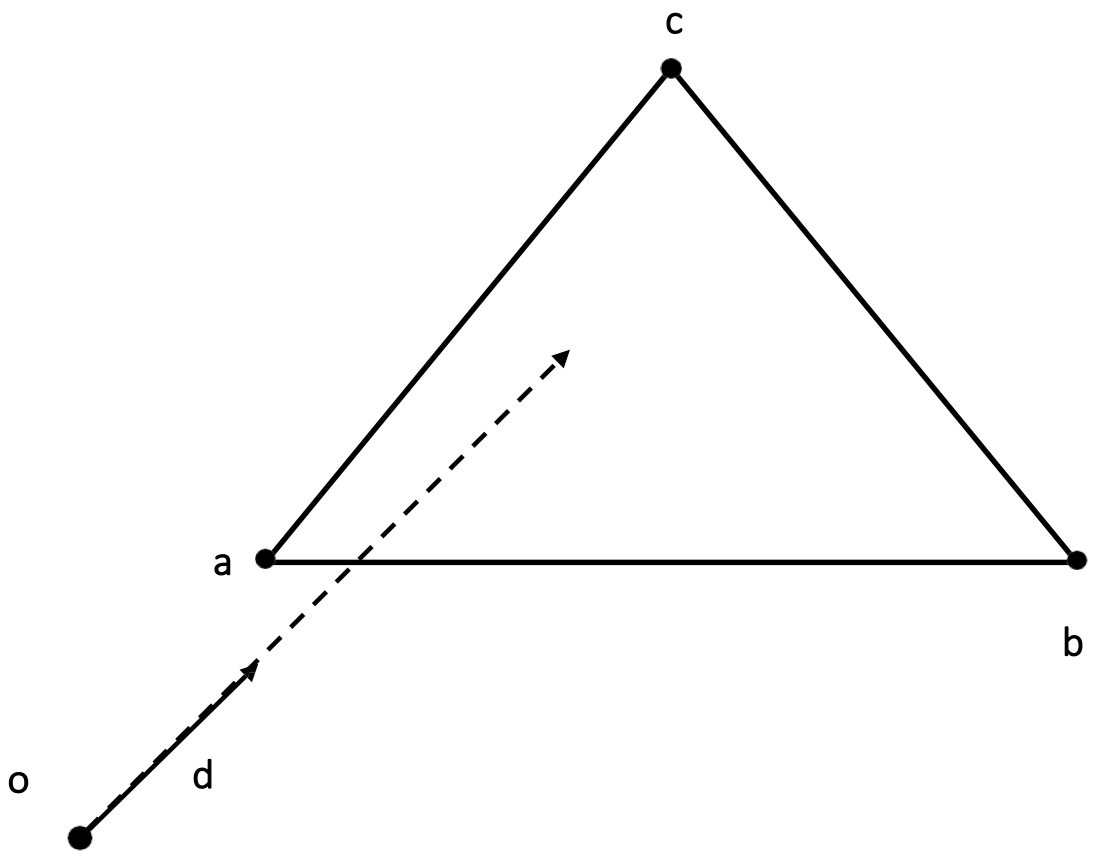
- Signed areas of sub-triangles
- Can be done in 2D, after “projection” onto major plane, depending on largest normal vector component

Test for positive BCs



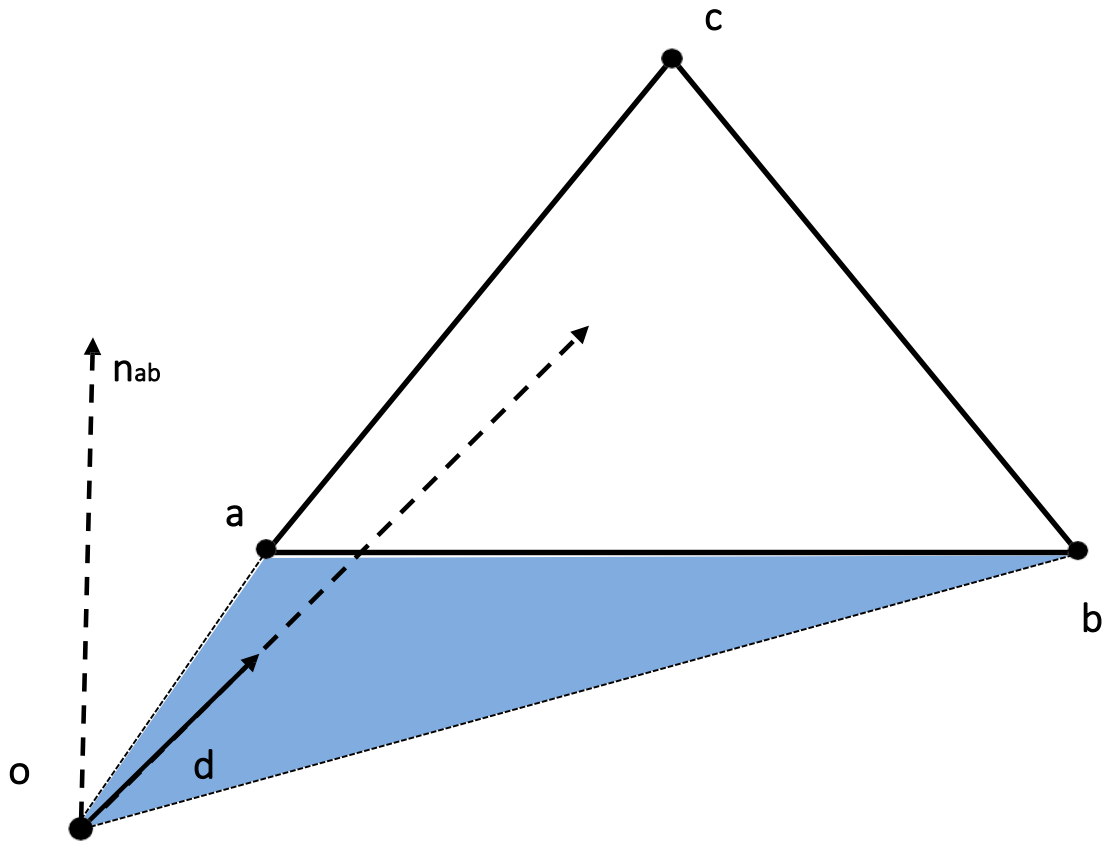
3D linear function across triangle (3D edge functions)

- Ray: $\vec{r}(t) = \vec{o} + t\vec{d}$
- Triangle: $\vec{a}, \vec{b}, \vec{c} \in \mathbb{R}^3$



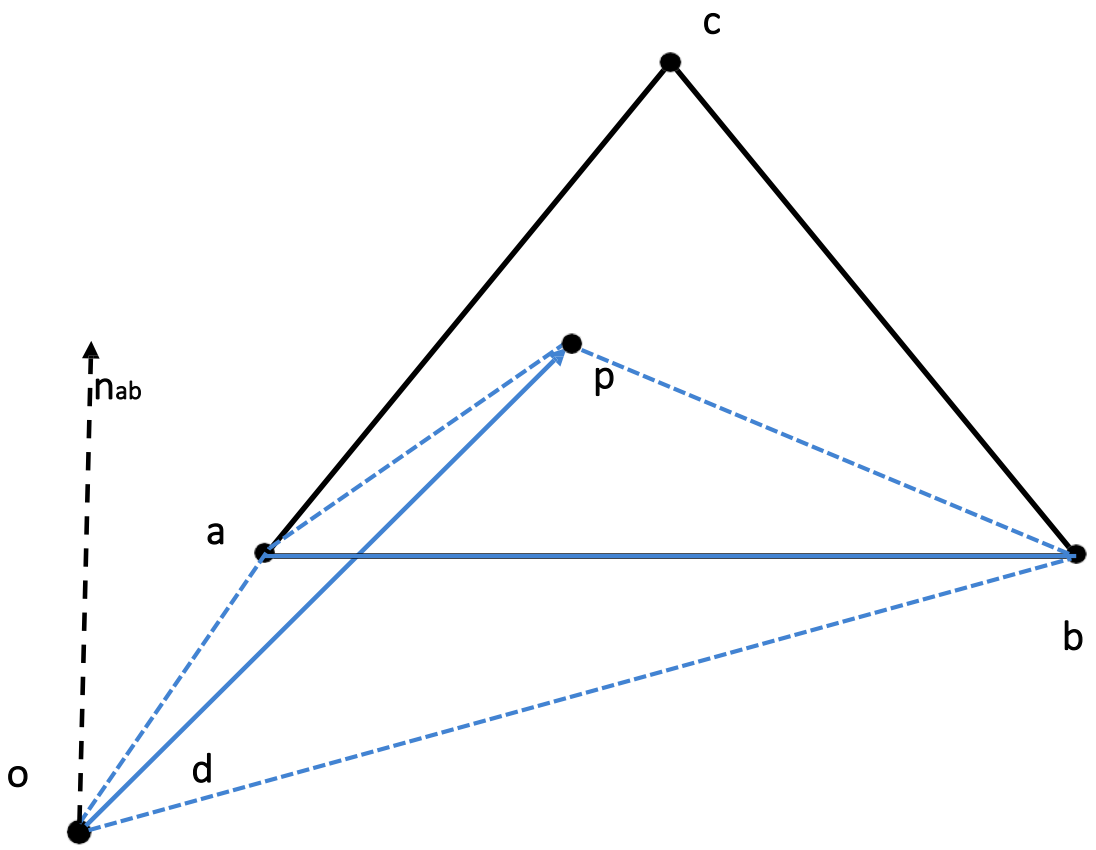
3D linear function across triangle (3D edge functions)

- Ray: $\vec{r}(t) = \vec{o} + t\vec{d}$
- 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 OAB (2 times)



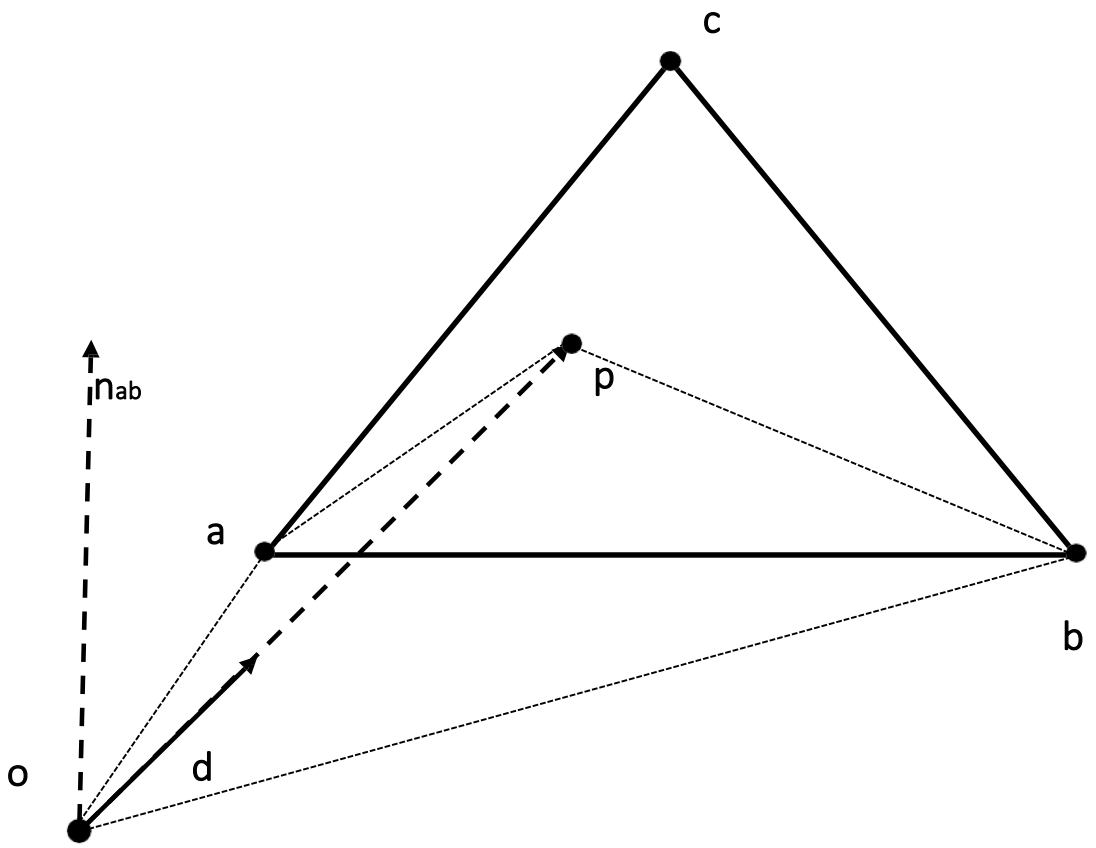
3D linear function across triangle (3D edge functions)

- Ray: $\vec{r}(t) = \vec{o} + t\vec{d}$
- 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 OAB (2 times)
- $\lambda_3^*(t) = \overrightarrow{n_{ab}} \cdot t\vec{d}$
  - Volume of OABP (6 times)
  - For $t = t_{hit}$



3D linear function across triangle (3D edge functions)

- Ray: $\vec{r}(t) = \vec{o} + t\vec{d}$
- 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 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
  - $\lambda_i = \frac{\lambda_i^*(t)}{\lambda_1^*(t) + \lambda_2^*(t) + \lambda_3^*(t)}, \quad i = 1, 2, 3$
  - Length of $t\vec{d}$ cancels out

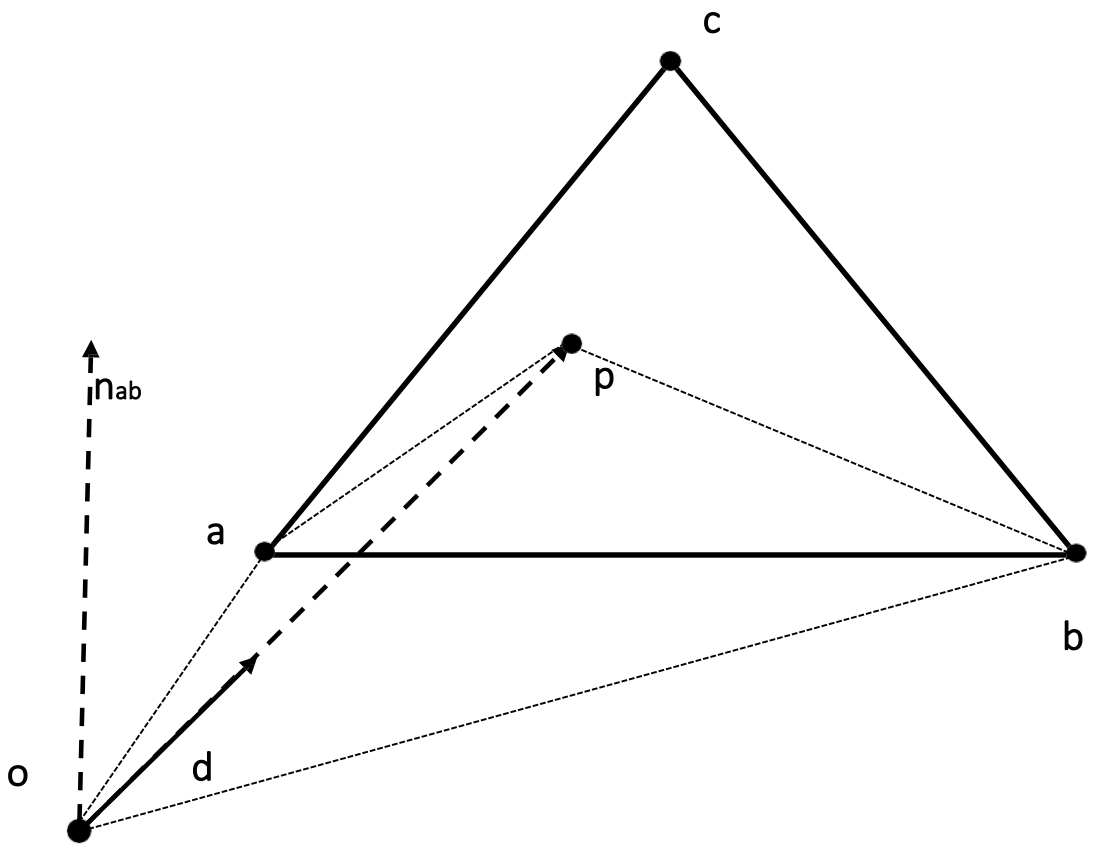


3D linear function across triangle (3D edge functions)

- Ray: $\vec{r}(t) = \vec{o} + t\vec{d}$
- 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 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
  - $\lambda_i = \frac{\lambda_i^*(t)}{\lambda_1^*(t) + \lambda_2^*(t) + \lambda_3^*(t)}, \quad i = 1, 2, 3$

For positive BCs

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



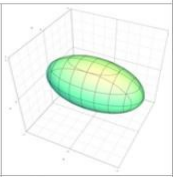
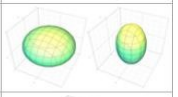
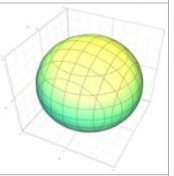
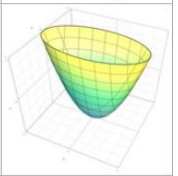
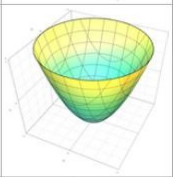
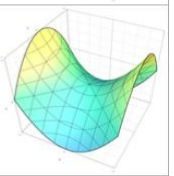
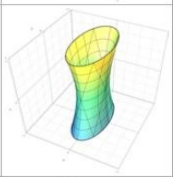
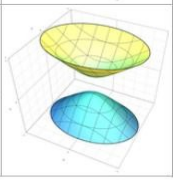
Quadrics

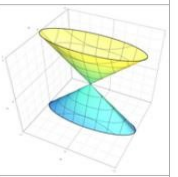
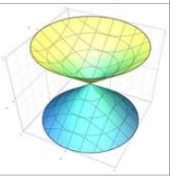
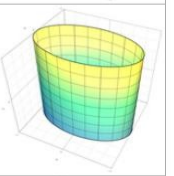
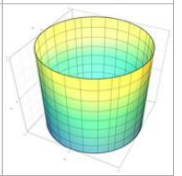
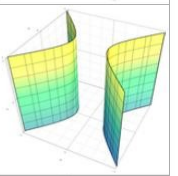
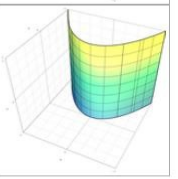
Implicit

- $f(x, y, z) = v$

Ray equation

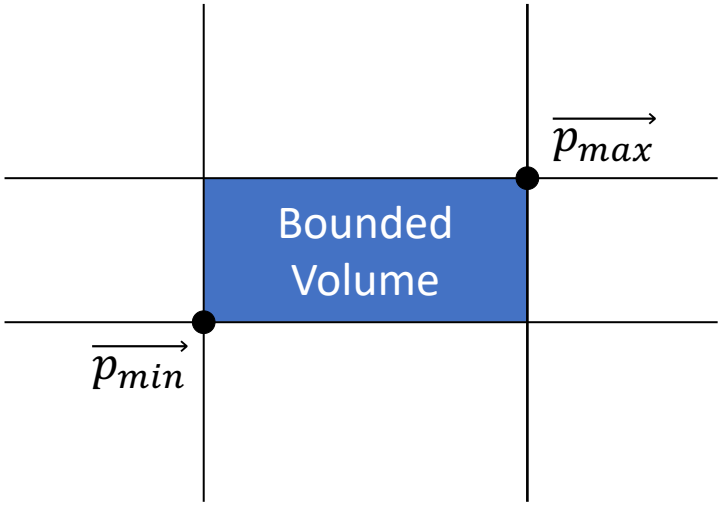
- $x = o_x + td_x$
- $y = o_y + td_y$
- $z = o_z + td_z$
- Solve for t

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	$x^2 + 2ay = 0$	

Given

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

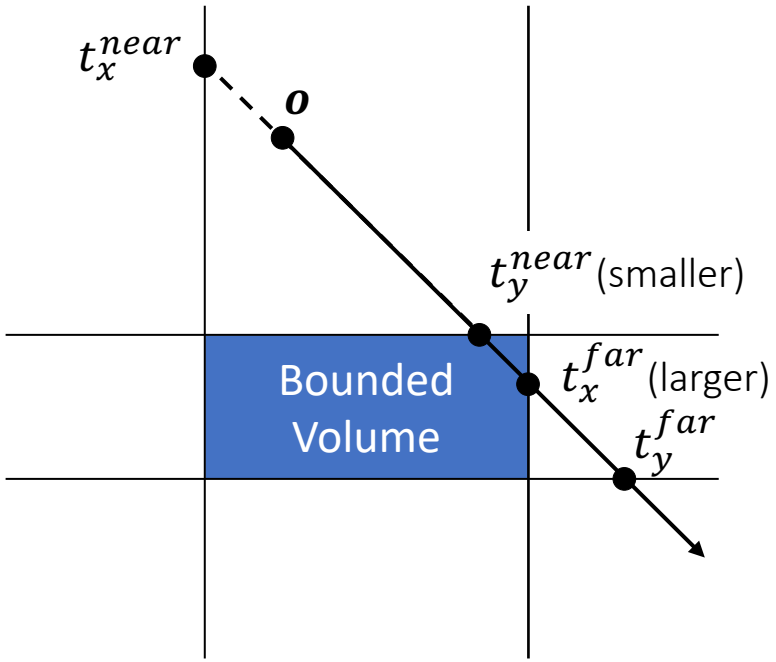


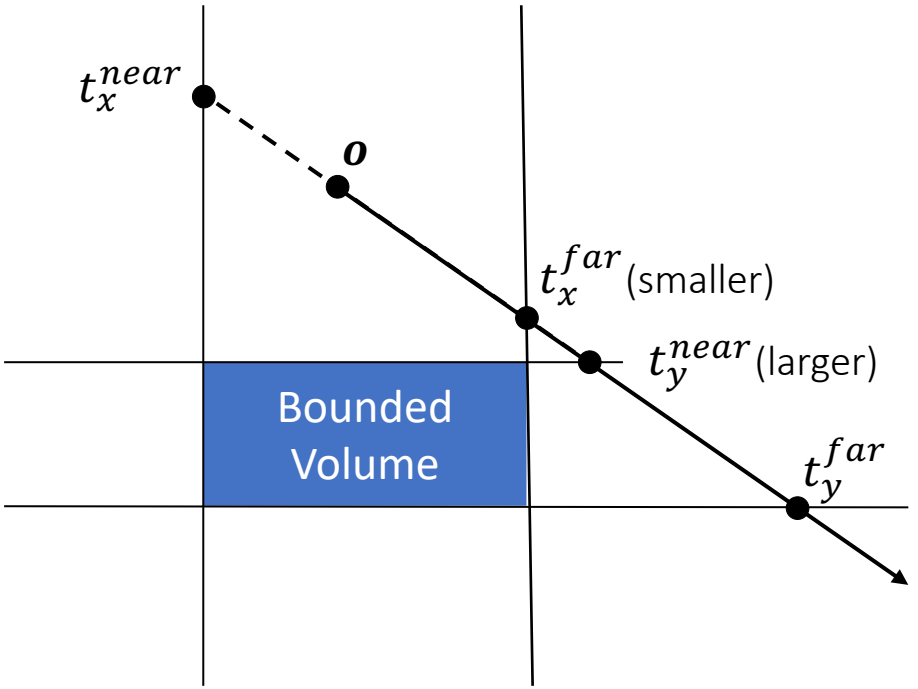
Given

- Ray: $\vec{r}(t) = \vec{o} + t\vec{d}$
- 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
- $max(\{t_i^{near} | i = x, y, z\}) < min(\{t_i^{far} | i = x, y, z\})$





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]