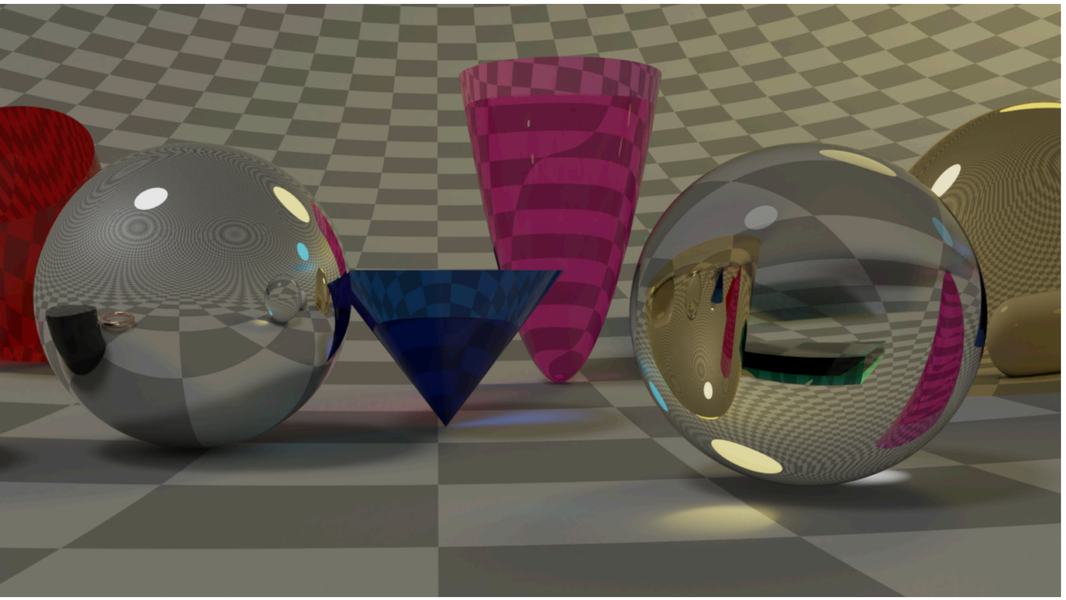
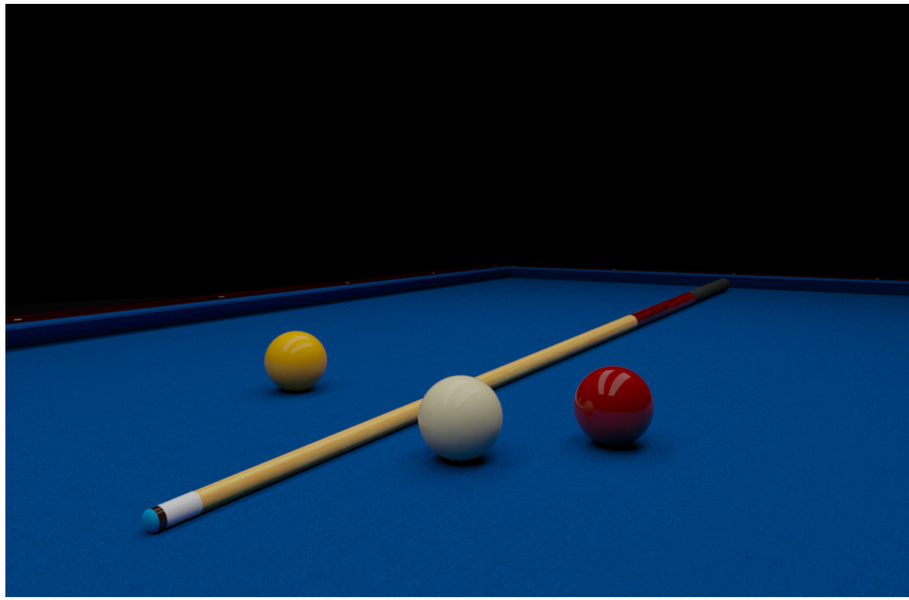
Visual Computing I:

Interactive Computer Graphics and Vision

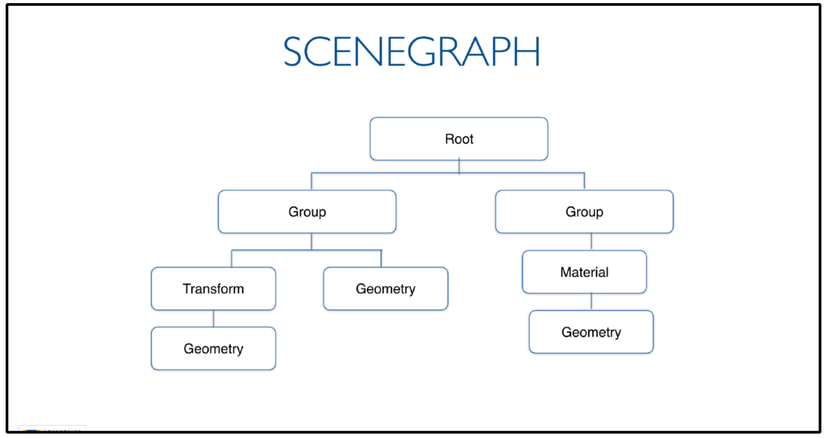


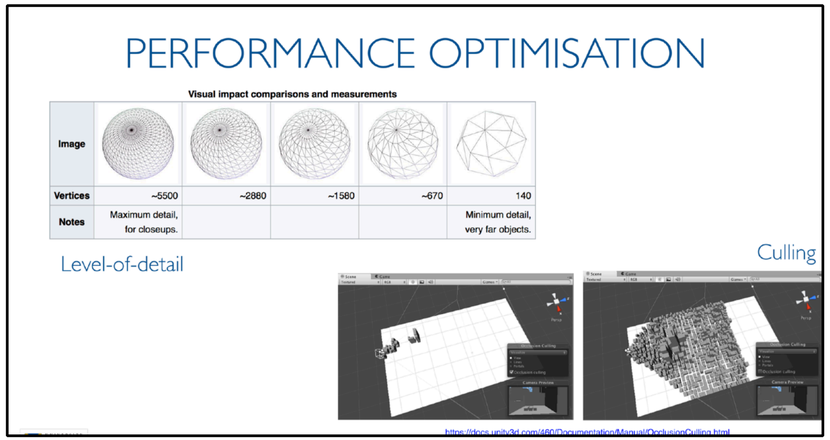


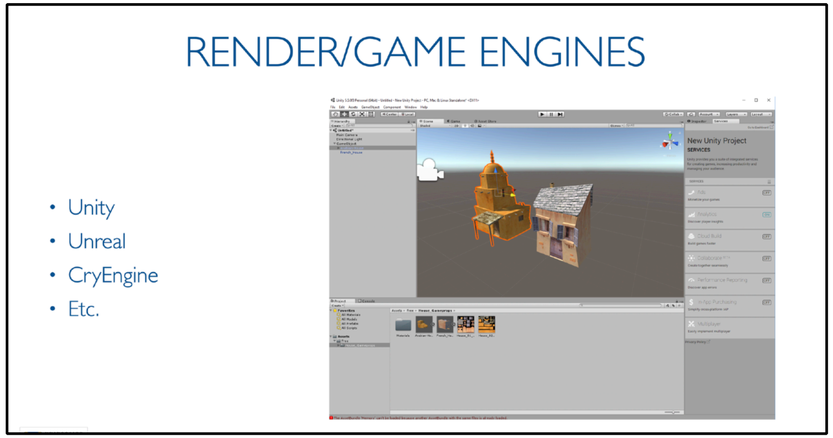
Raytracing

Stefanie Zollmann and Tobias Langlotz

Last time





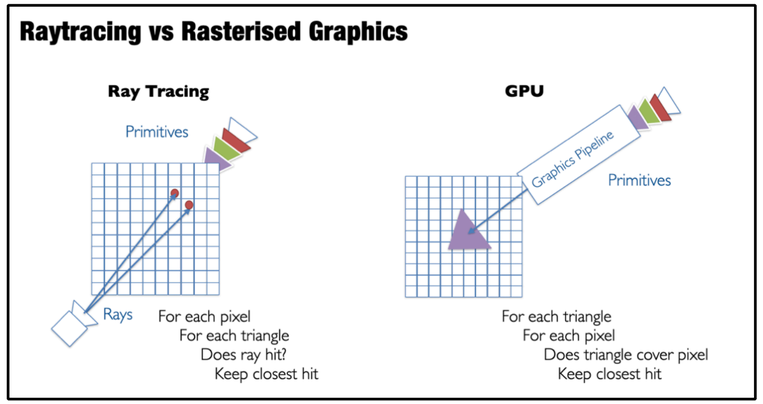


SCENEGRAPH

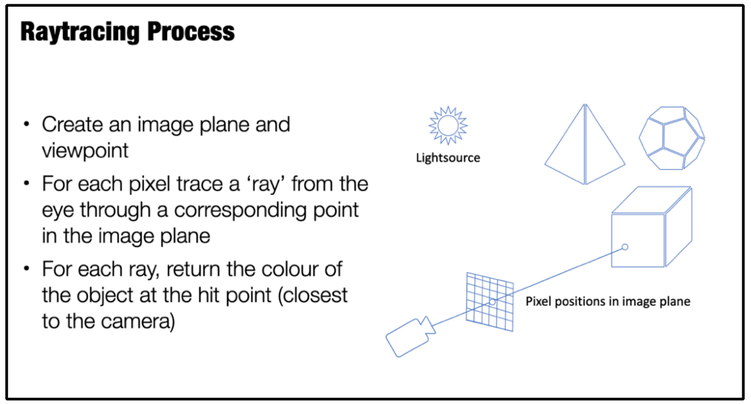
OPTIMISATIONS

GAME ENGINES

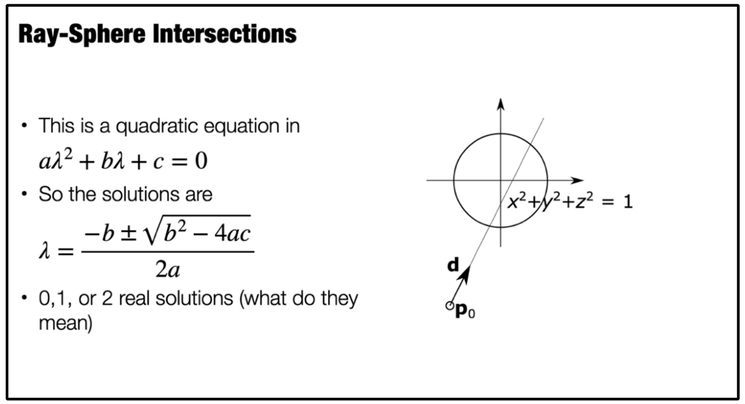
Today



Raytracing/Rasterised



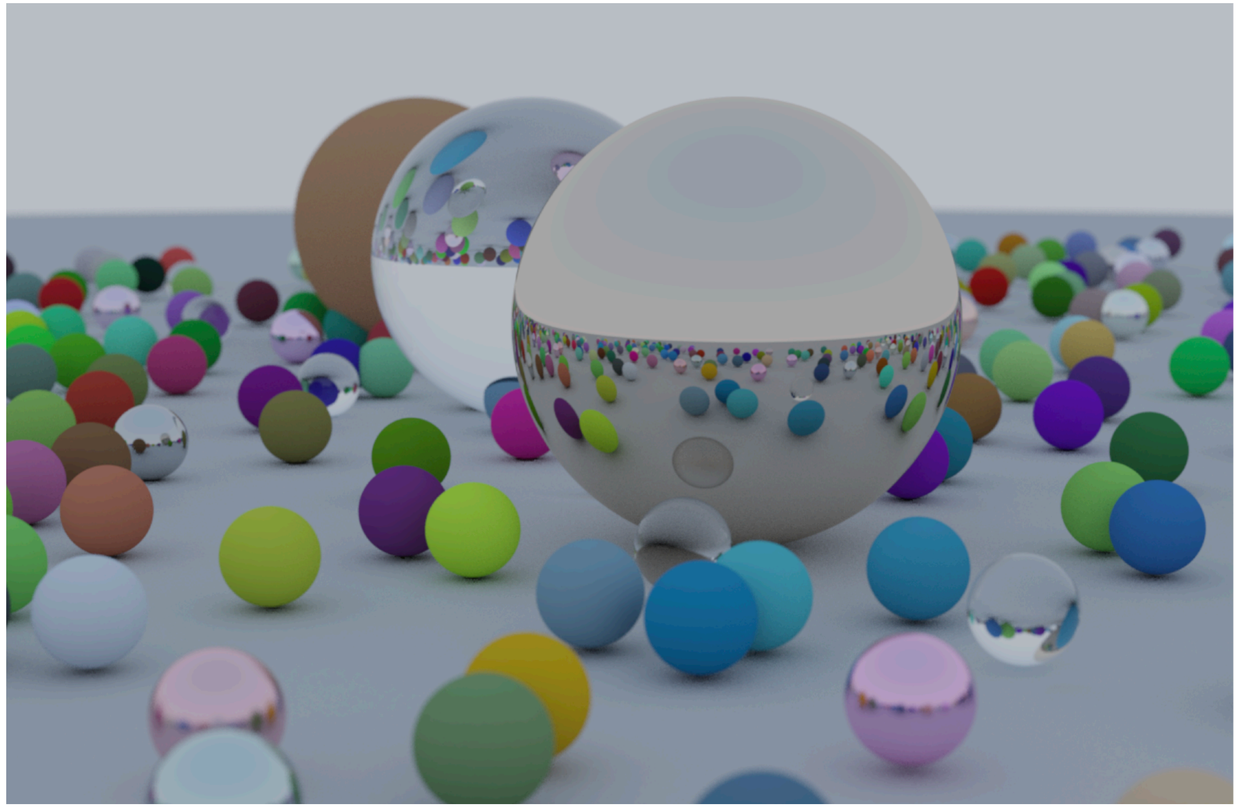
Raytracing Process



Intersections

Motivation

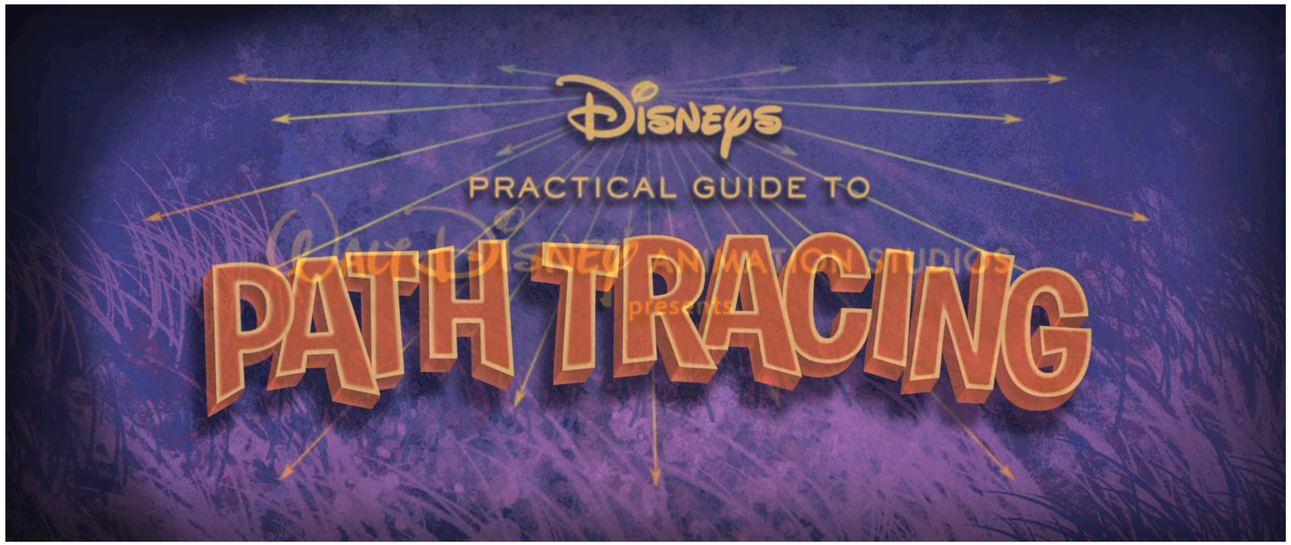
- Rasterisation provides high-performance image synthesis but is limited to local lighting models
- Global illumination phenomena (e.g., indirect lighting, soft shadows, caustics) are difficult to approximate in rasterisation pipelines
- Ray tracing approximates physical light transport, enabling more accurate optical effects



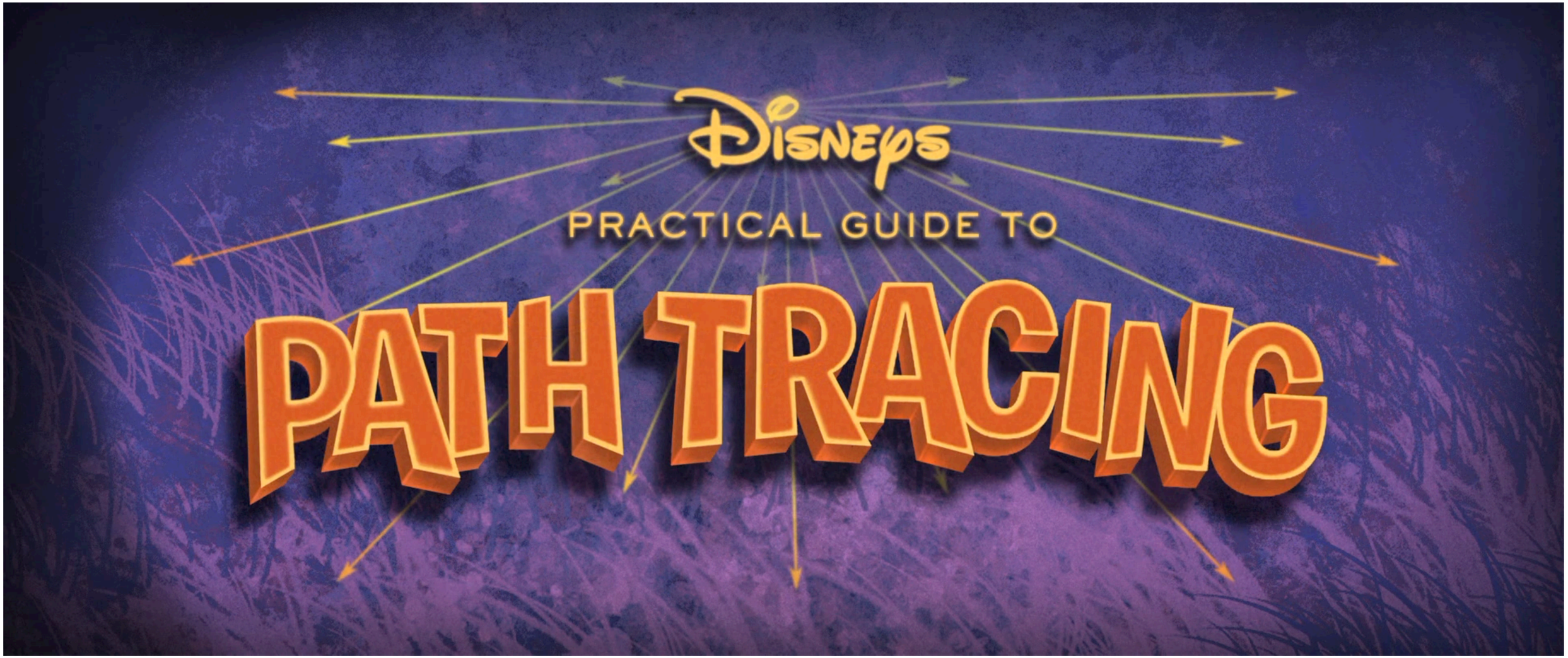
https://erichlof.github.io/THREE.js-PathTracing-Renderer/RayTracing_In_One_Weekend.html

Fundamentals

- Treat light propagation as geometric rays
- For each pixel, generate one or more camera rays
- Determine the nearest surface intersection for each ray
- Evaluate material response at the intersection
- Emit secondary rays as required to account for light interactions

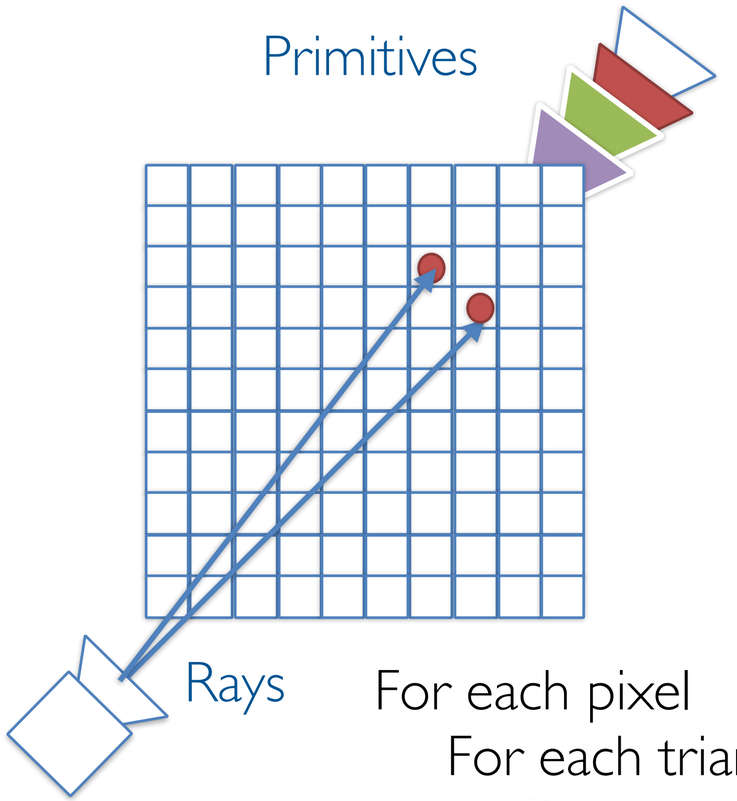


Fundamentals



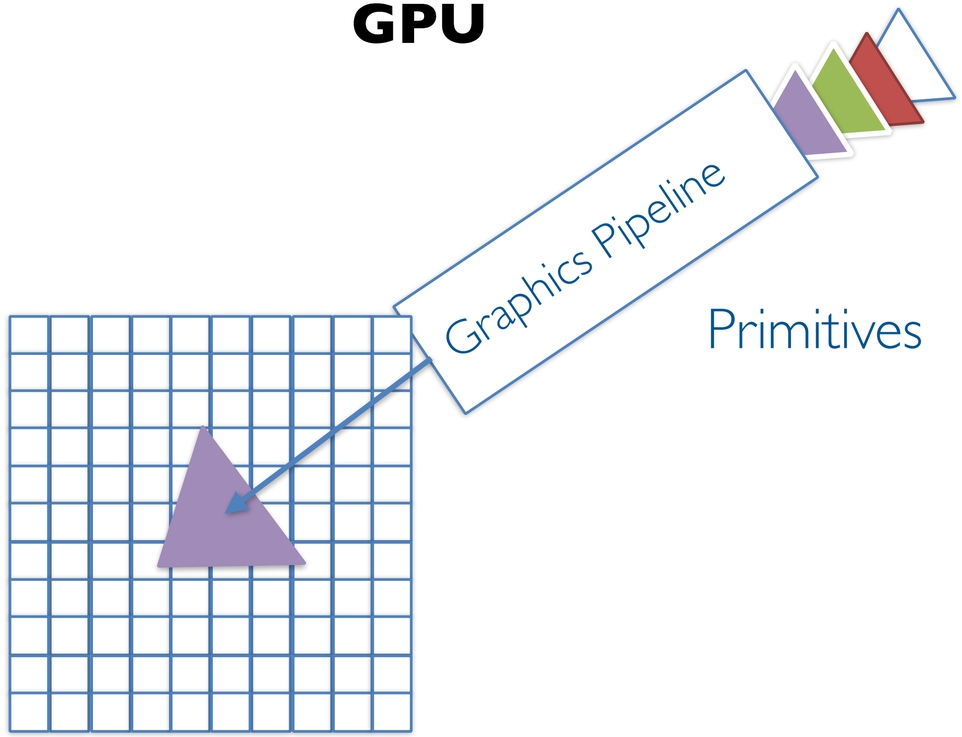
Raytracing vs Rasterised Graphics

Ray Tracing



For each pixel
  For each triangle
    Does ray hit?
      Keep closest hit

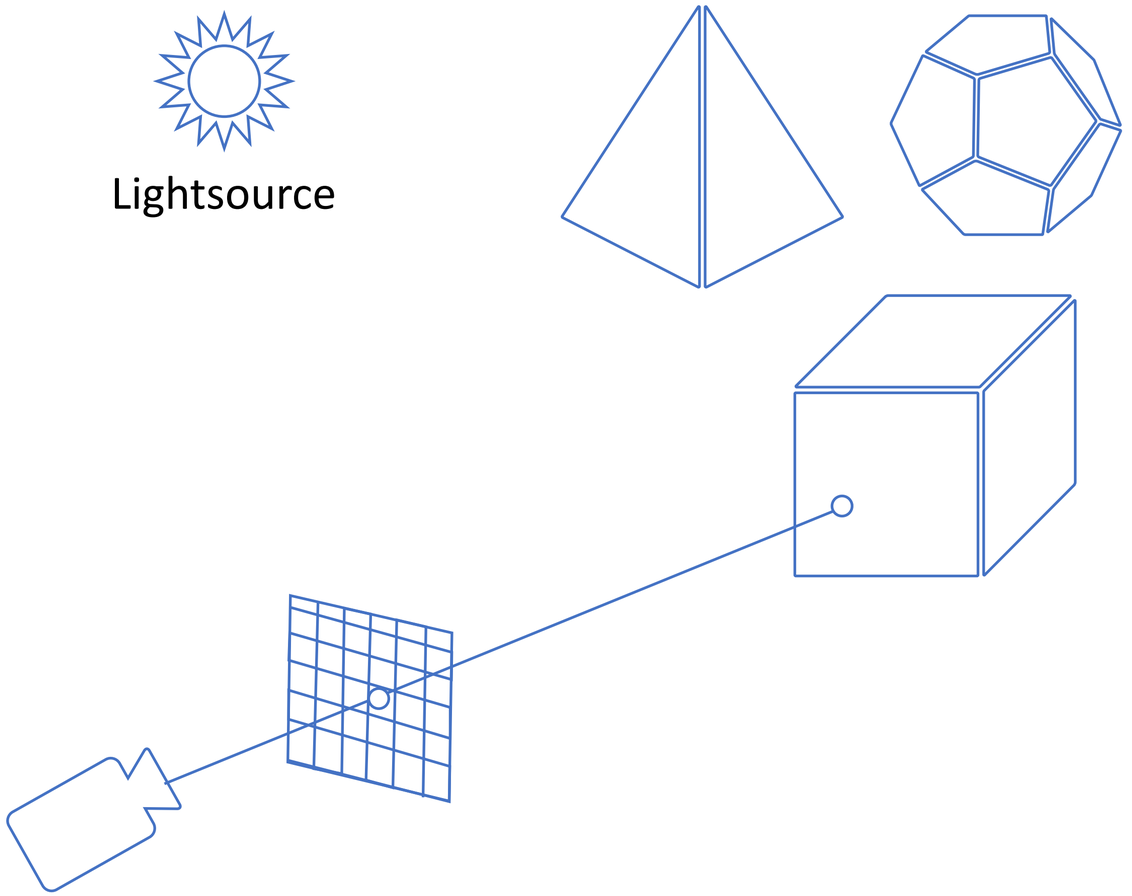
GPU



For each triangle
  For each pixel
    Does triangle cover pixel
      Keep closest hit

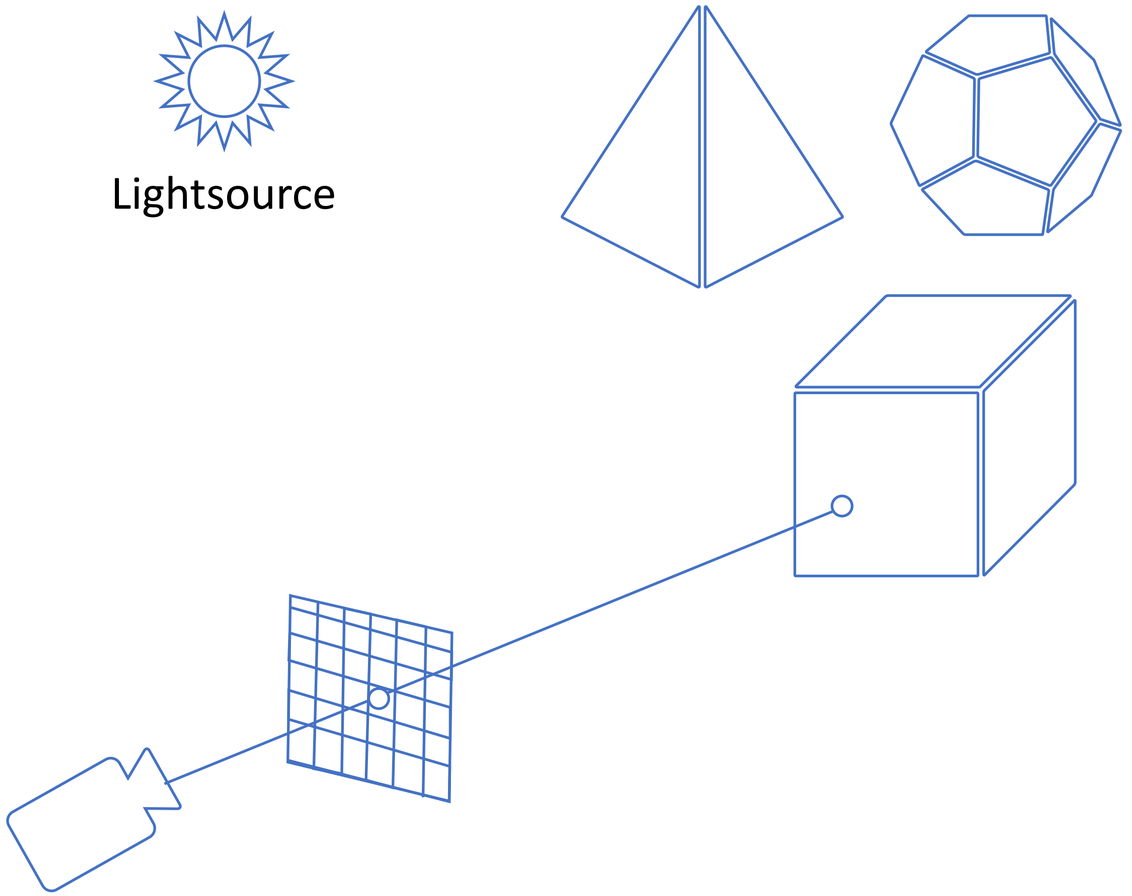
A Simple Raytracer

- Supports:
  - Spotlights and Shadows
  - Reflections
  - Refraction



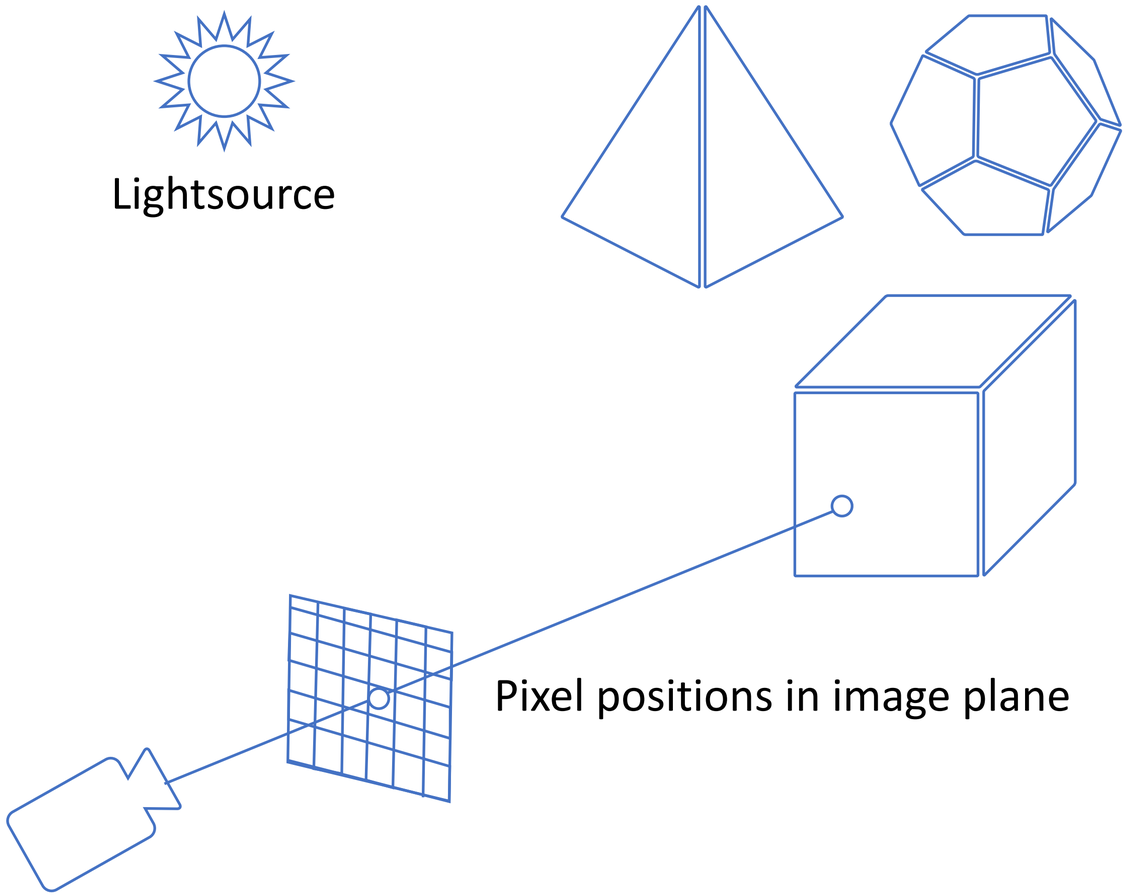
Local Illumination Models

- Compute outgoing radiance at an intersection
- Classic models:
  - Lambertian diffuse
  - Phong / Blinn–Phong specular components
- Requires shadow rays to determine direct-light visibility



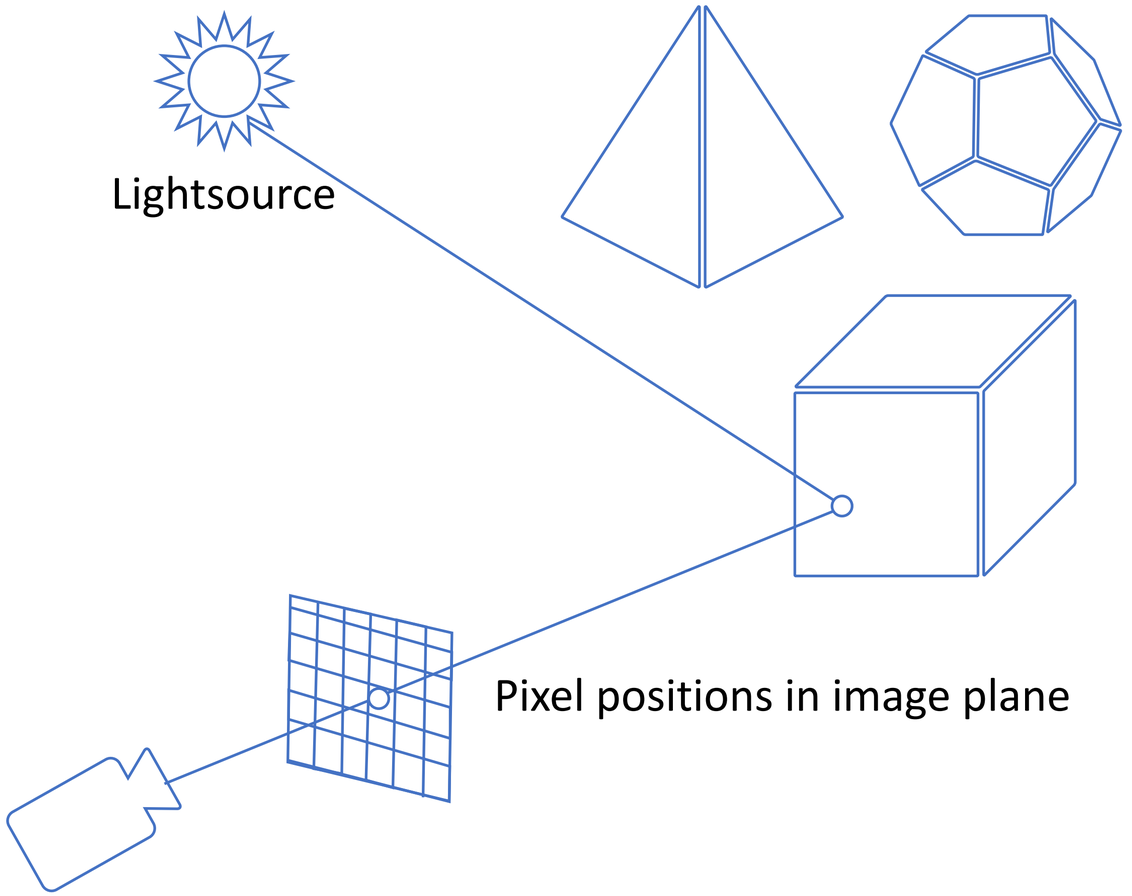
Raytracing Process

- Create an image plane and viewpoint
- For each pixel trace a ‘ray’ from the eye through a corresponding point in the image plane
- For each ray, return the colour of the object at the hit point (closest to the camera)



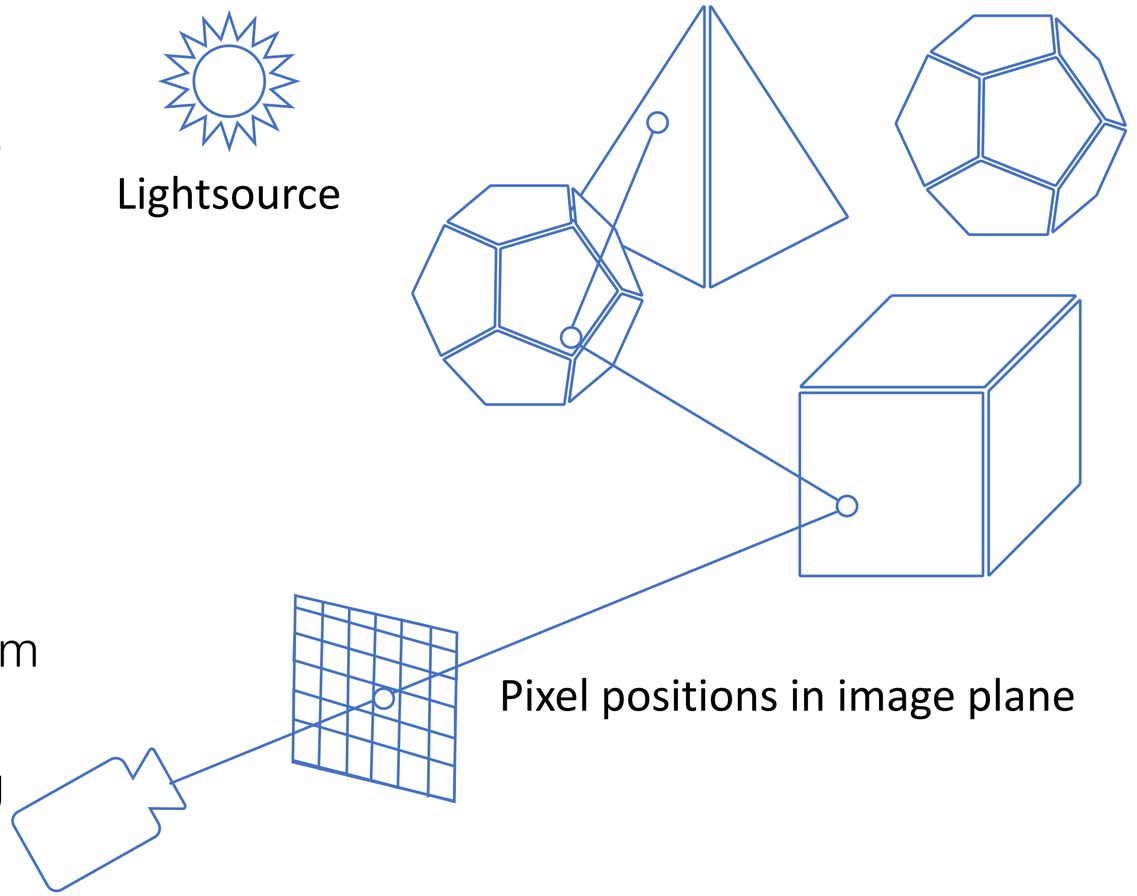
Raytracing Process - Shadows

- Shadows:
  - Ray is casted towards light source
  - If an object is located between hit point (surface point) and light source then the hit point is in shadow
  - If not, there is no shadow
  - Point lights: hard binary shadows
  - Area lights: sample positions on the emitter to produce soft penumbrae



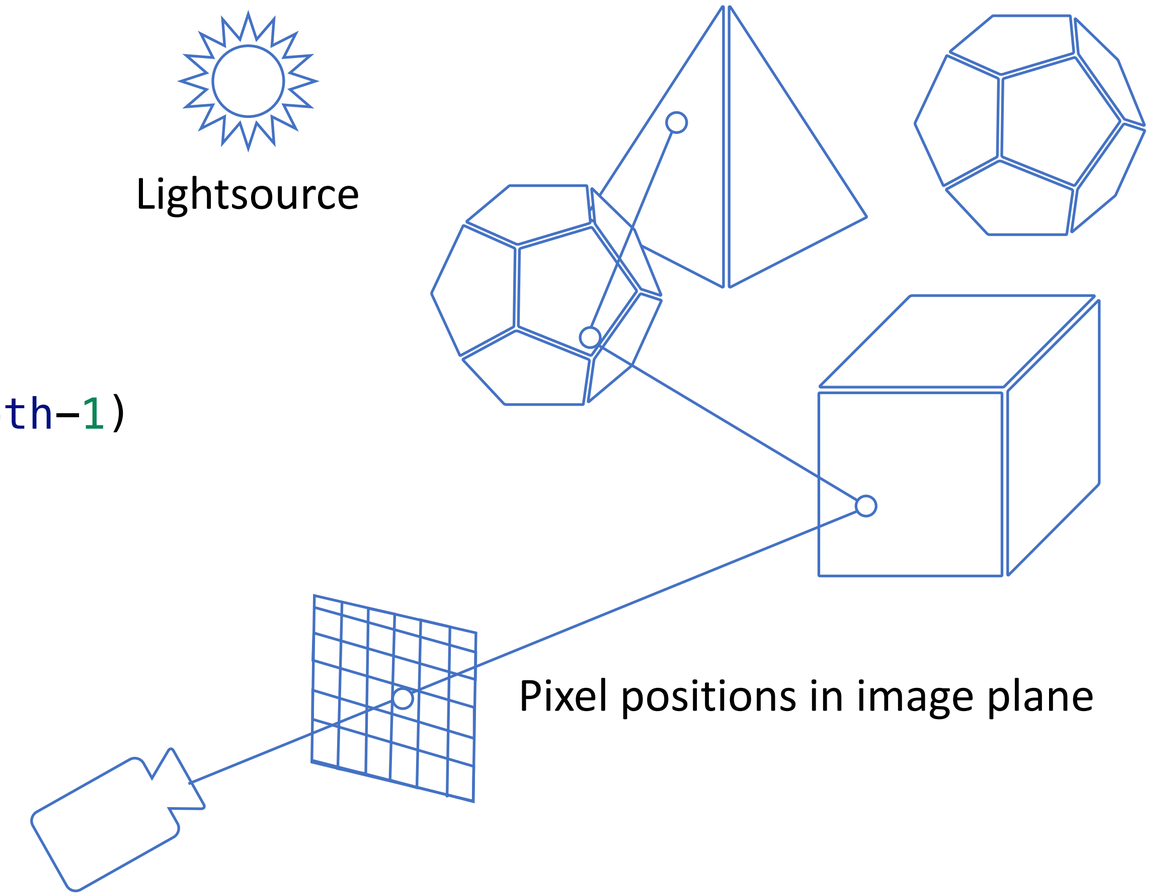
Raytracing Process - Reflection

- Method:
  - If ray is reflected on the surface (reflective material), a new (secondary) ray is created and cast in the reflective direction
  - Compute reflected direction using the law of reflection
    $\mathbf{r} = \mathbf{v} - 2(\mathbf{v} \cdot \mathbf{n})\mathbf{n}$
  - Spawn secondary ray from hit point
  - Modulate the reflected contribution by the material's reflectivity ρ_r
  - Trace the reflected ray recursively, respecting a maximum depth or probabilistic termination
  - Finally: Add returned reflected radiance to local shading
    $L = L_{\text{local}} + \rho_r L_{\text{reflected}}$



Raytracing Process - Reflection

```
colour render(ray, scene, fraction, depth)
   hit = intersect(ray, scene)
   if (depth == 0 || hit.kr * fraction < TINY)
      thisColour = hit.colour
   else
      thisColour = (1 - hit.kr) * hit.colour +
            hit.kr * render(reflection, scene,
                            hit.kr*fraction, depth-1)
   return fraction * thisColour
```



*hit.kr stands for “coefficient of reflection” (sometimes called reflectivity).
It is a value between 0 and 1:
0: the surface is non-reflective (all light is absorbed or diffused)
1: the surface is a perfect mirror (all light is reflected)
Values in between → partial reflection (some light is reflected, some is diffuse).

Raytracing Process - Refraction

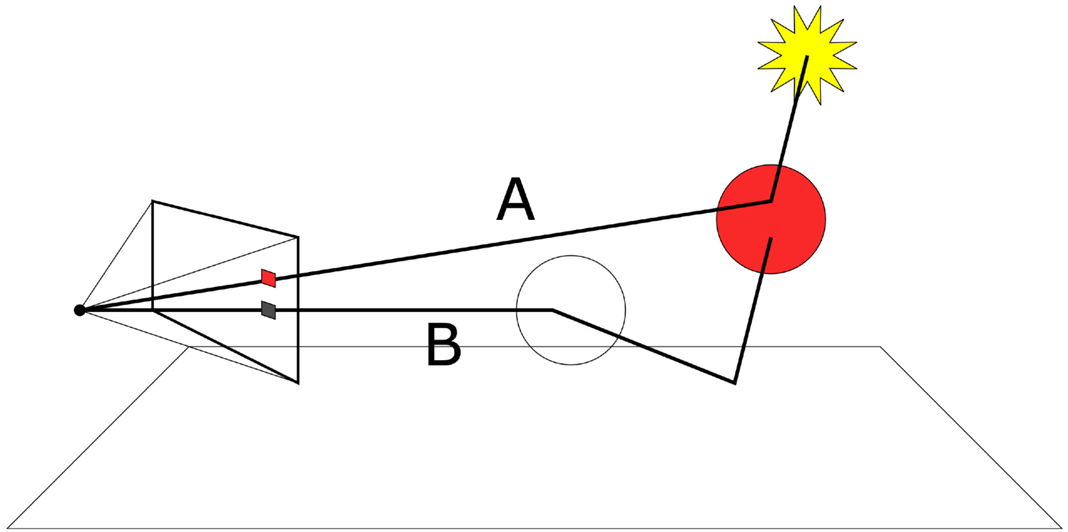
- Transparent objects
- Light bends at the surface
- Light ‘bends’ between materials
- Material has a refractive index
- Ratio of refractive indices determines the bending (Snell’s law): $n_1 \sin\theta_1 = n_2 \sin\theta_2$
  - Vacum: $\eta = 1$
  - Air: $\eta = 1.0003$
  - Water : $\eta = 1.333$
- Refracted ray:

$$\mathbf{T} = \frac{n_1}{n_2}\mathbf{I} + \left(\frac{n_1}{n_2}\cos\theta_i - \cos\theta_t\right)\mathbf{N}$$

By Ulflund - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=73684026

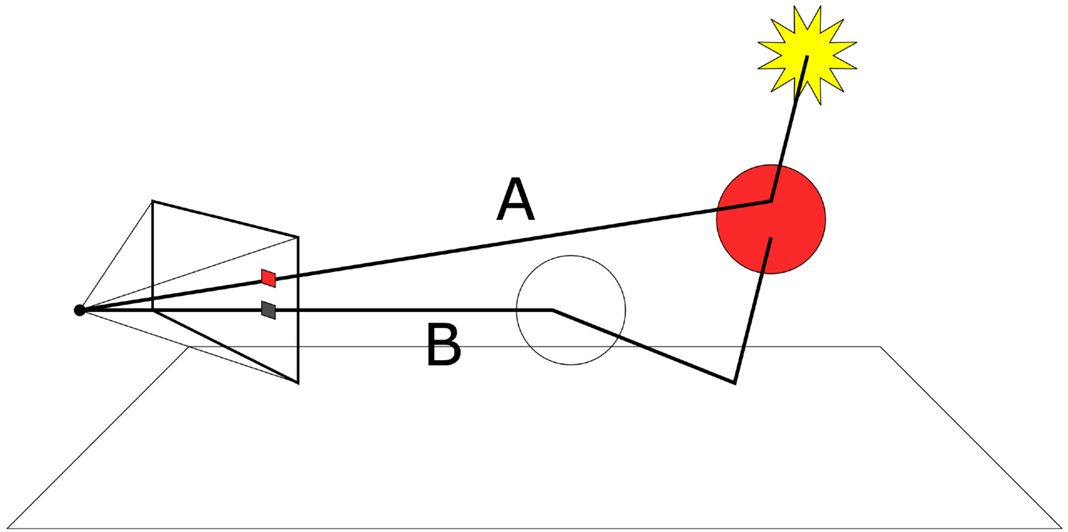
Raytracing Process - Examples

- Ray A is the ‘typical’ case
  - Ray hits the coloured sphere
  - A secondary ray is cast towards the light
  - That secondary ray doesn’t hit anything, so there is no shadow
  - The colour of the hit point on the sphere is used for the pixel
- Ray B is more complex
  - The ray hits a glass sphere which bends the ray, so a new ray is cast
  - The ray bends again as it leaves the sphere, then hits the ground plane
  - A ray is then cast to check for shadows
  - This hits the coloured sphere so the point is in shadow
  - The pixel is, therefore, black



Summary Raytracing Process

- The basic ray tracing algorithm:
  - Set up a camera – a projection point and an image plane
  - For each pixel in the plane:
    - Cast a ray from the projection point through the pixel
    - Determine the first object hit by that ray
    - Cast additional ray(s) to determine lighting
    - Additional rays can be used for reflection, refraction, etc.

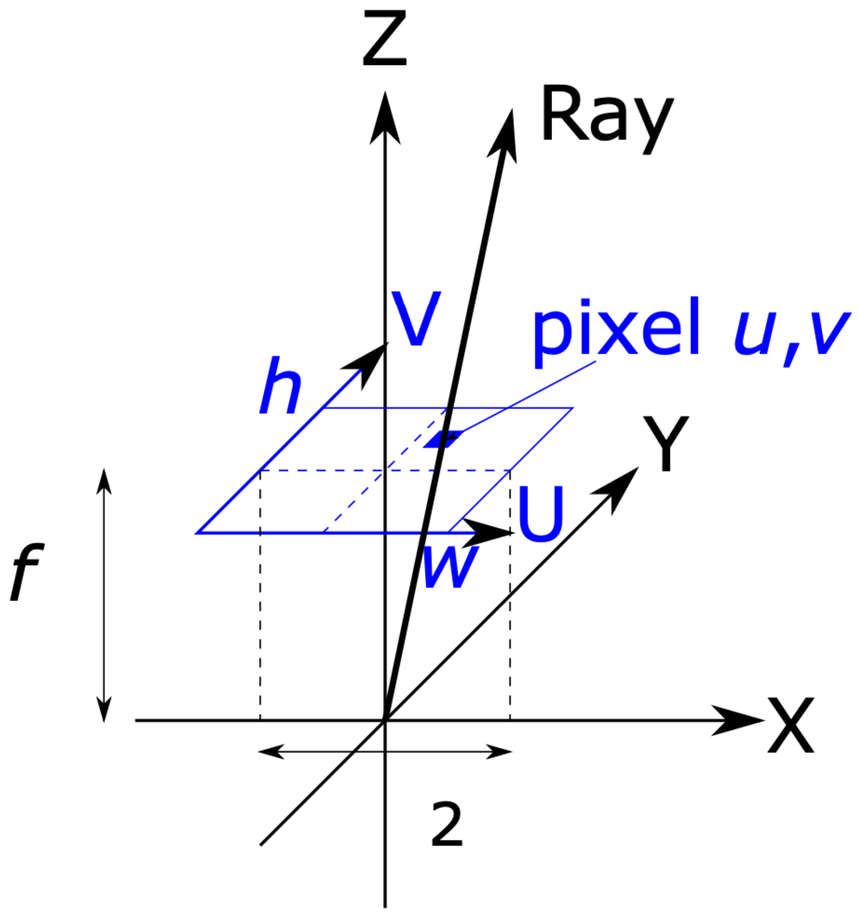


Raytracing Process (Pseudo Code)

```
function raycast():
  for all pixels (x,y):
    ray = compute_eye_ray(x, y)
    image(x,y) = trace(ray)


function trace(ray r):
  closest_t = ∞
  closest_o = null

  for all objects o:
    t = compute_intersection(r, o)
    if t > 0 and t < closest_t:
      closest_t = t
      closest_o = o

  if closest_o != null:
    return shade(closest_o, r, closest_t)
  else
    return background_color


function shade(object o, ray r, t):
  x = r(t)         // intersection point
  // evaluate Phong illumination model at x
  return compute_phong(o, x, r)
```

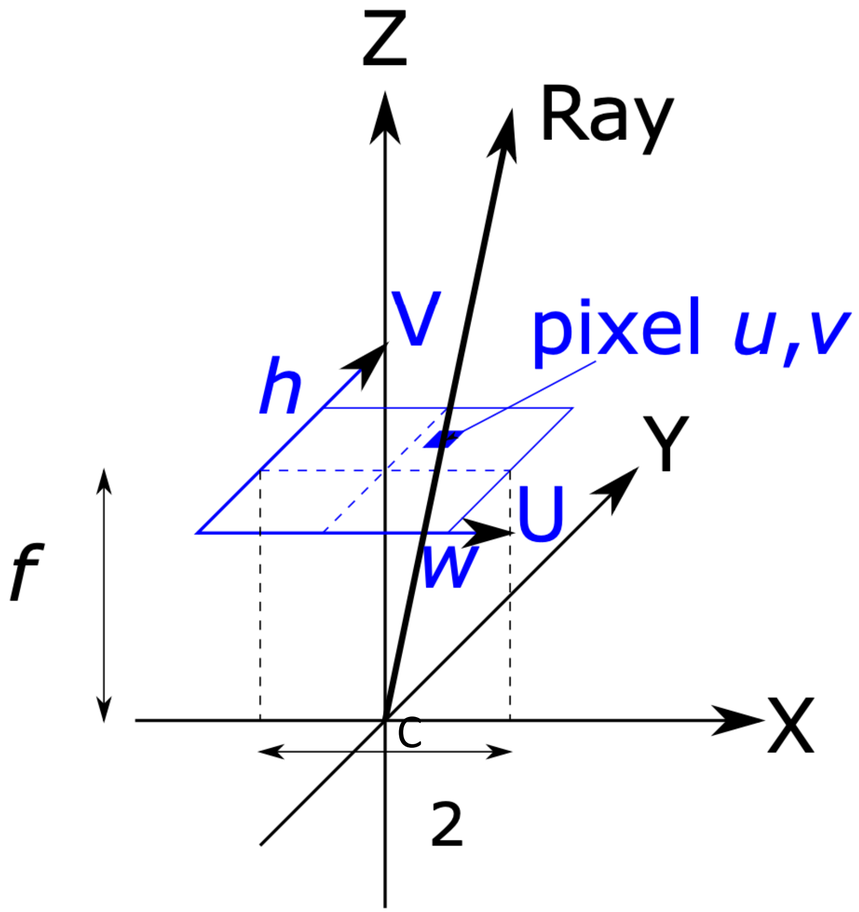
Primary Ray and Camera Model

- Primary Ray
  - Defined by:
    - Origin at the camera center
    - Direction through a pixel on the image plane
    - Intersection tests determine visible surfaces
- Camera models:
  - Pinhole (standard for educational ray tracers)
  - Thin-lens (focus, depth of field)



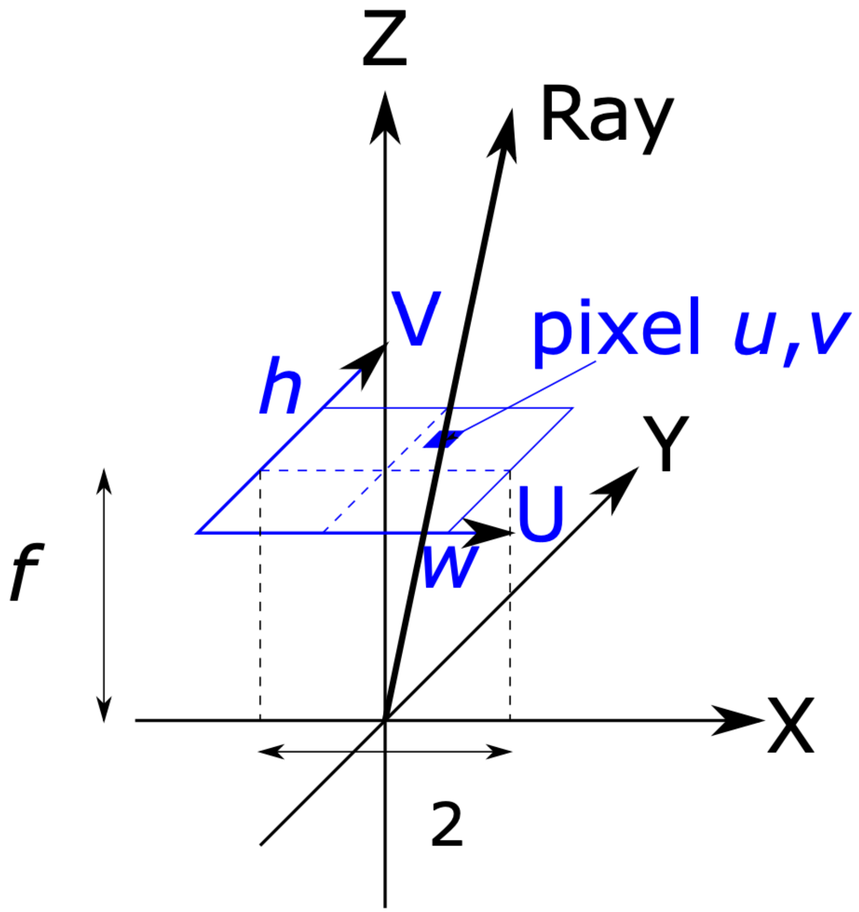
Primary Ray and Camera Model

- The primary ray for each pixel
  - Starts at the camera center, c
  - Goes through the middle of the pixel
- We make some simplifying assumptions
  - The camera is at the origin, looking along +Z
  - The (image) U and V axes are aligned with the (world) X and Y axes
  - The image plane is two ‘world units’ wide
  - We are given a focal length, f , and a rendered image size, w × h
  - Our pixels are given integer coordinates (u,v) starting from (0,0)



Primary Ray and Camera Model

- The ray is defined by two points
  - 1) Camera centre [0,0,0]
  - 2) The centre of the pixel (u,v)
- Pixel/image coordinates are (U,V)
- Need world (X,Y,Z) coordinates
- Where is the image origin?
- How big is each pixel?



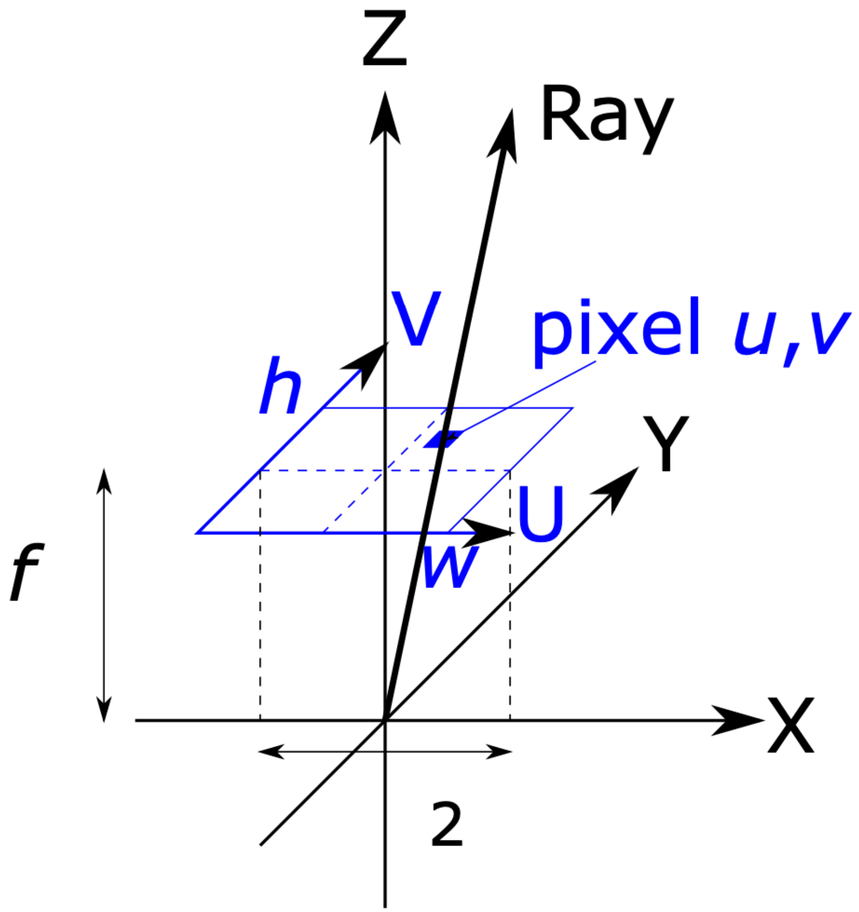
Primary Ray and Camera Model

- We can define the image origin:
  - Z value is focal length, f
  - X value is -1 (x values range from -1 to 1)
  - Y value is $\frac{h}{w}$ (y values range from $-\frac{h}{w}$ to $\frac{h}{w}$)
  - Size of each pixel in world units:
    - Width of image is 2
    - So pixels are $\frac{2}{w}$ on a side
    - Coordinates of pixel (u,v):

$$[-1 + (u + \frac{1}{2})\frac{2}{w}, \frac{-h}{w} + (v + \frac{1}{2})\frac{2}{w}, f]^T$$

- Our ray: $\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d}(u, v)$:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \lambda \begin{bmatrix} -1 + \left(u + \frac{1}{2}\right)\frac{2}{w} \\ -\frac{h}{w} + \left(v + \frac{1}{2}\right)\frac{2}{w} \\ f \\ 0 \end{bmatrix}$$



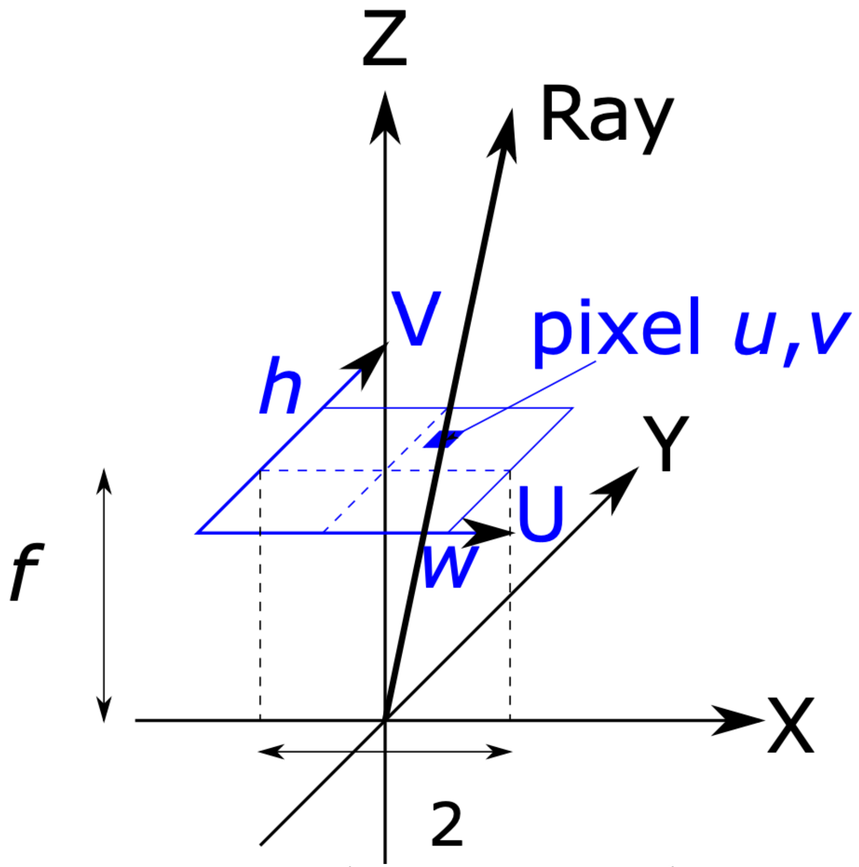
Primary Ray and Camera Model

- We can define the image origin:
  - Z value is focal length, f
  - X value is -1 (x values range from -1 to 1)
  - Y value is $\frac{h}{w}$ (y values range from $-\frac{h}{w}$ to $\frac{h}{w}$)
  - Size of each pixel in world units:
    - Width of image is 2
    - So pixels are $\frac{2}{w}$ on a side
    - Coordinates of pixel (u,v):

$$[-1 + (u + \frac{1}{2})\frac{2}{w}, \frac{-h}{w} + (v + \frac{1}{2})\frac{2}{w}, f]^T$$

- Our ray: $\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d}(u, v)$:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \lambda \begin{bmatrix} -1 + \left(u + \frac{1}{2}\right)\frac{2}{w} \\ -\frac{h}{w} + \left(v + \frac{1}{2}\right)\frac{2}{w} \\ f \\ 0 \end{bmatrix}$$

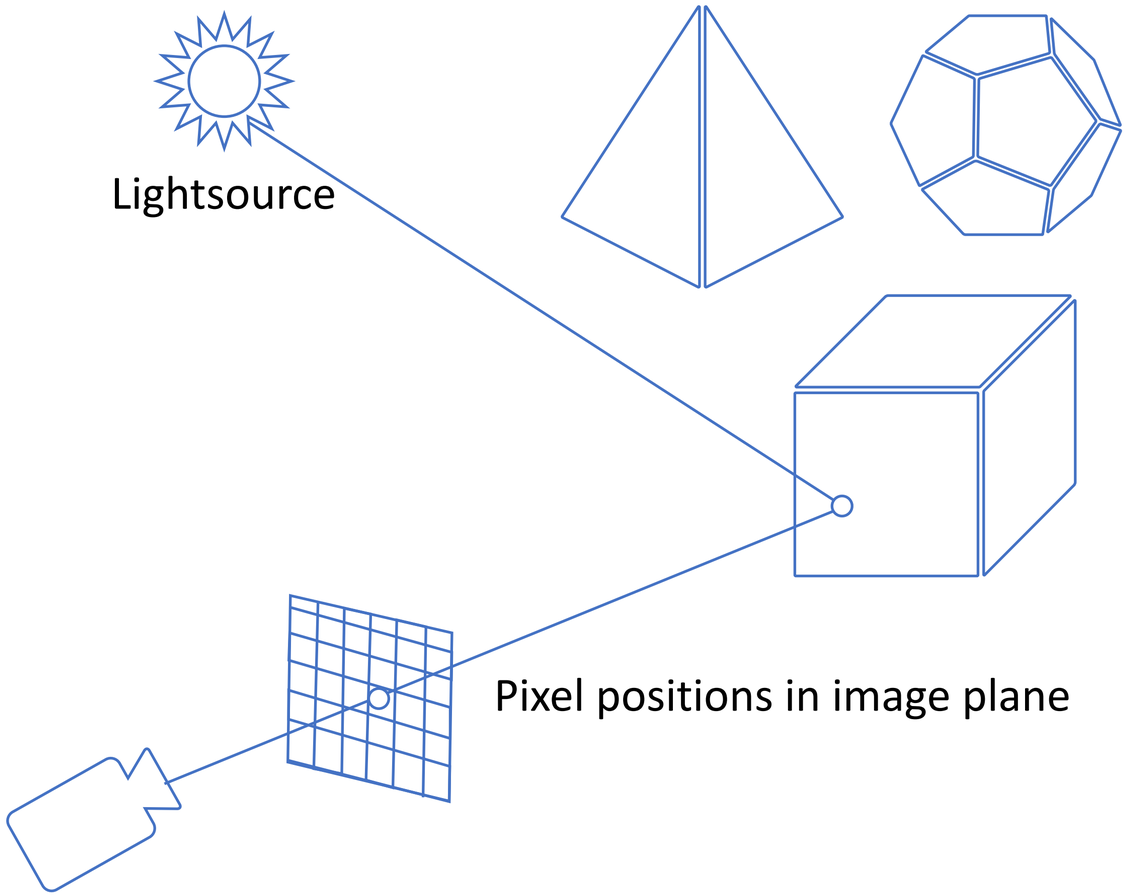


$$\mathbf{p}(0,0) = \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ f \end{bmatrix}, \qquad \mathbf{p}(1,0) = \begin{bmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ f \end{bmatrix},$$

$$\mathbf{p}(0,1) = \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ f \end{bmatrix}, \qquad \mathbf{p}(1,1) = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ f \end{bmatrix}.$$

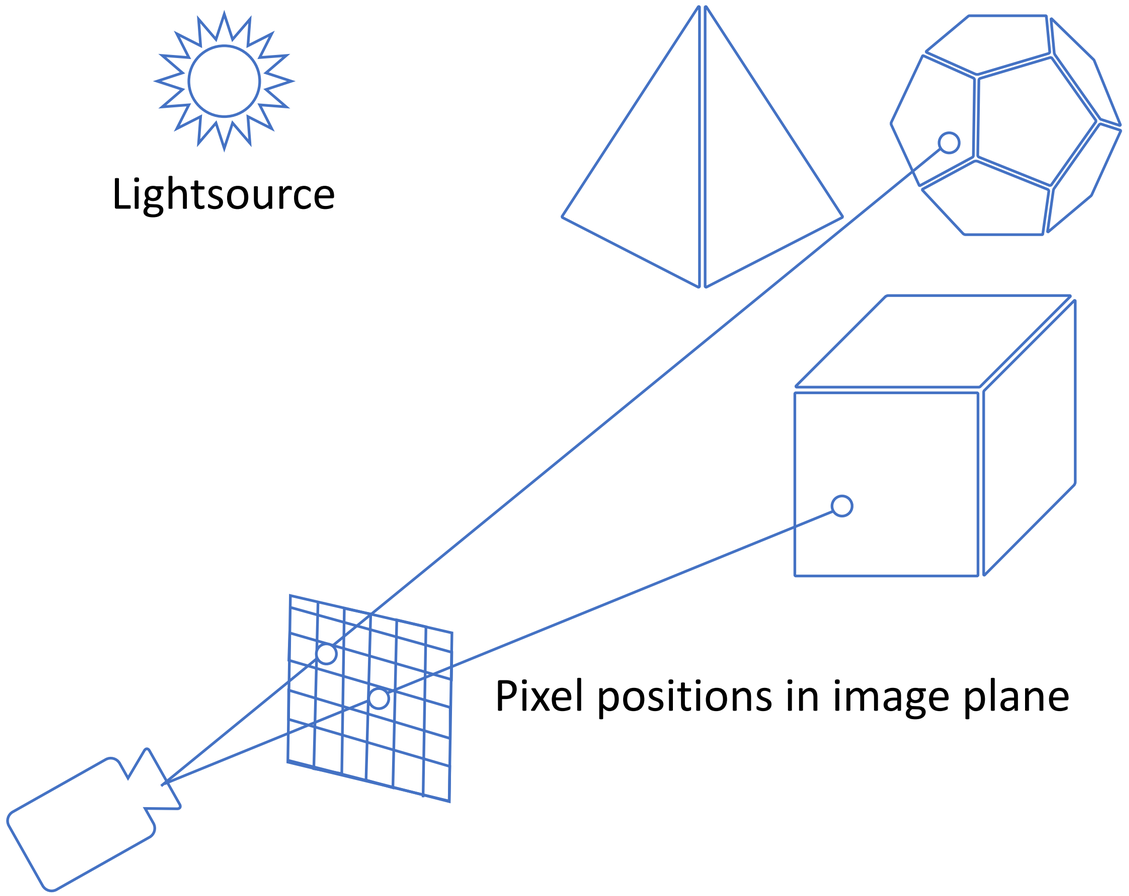
Secondary Rays

- Reflection rays to model specular and mirror-like materials
- Refraction rays for transparent materials:
  - Snell's law and Fresnel terms
  - Recursion depth controls complexity and runtime
- Enables physically plausible mirror reflections, transparency, and glossy effects



Geometric Intersection Tests

- Ray–sphere: closed-form quadratic equation
- Ray–box: testing each side
- Intersection cost dominates rendering -> acceleration required



Ray-Sphere Intersections

- Spheres have a simple mathematical form
  - Given a point $p = (x, y, z)$
  - Its squared distance from the origin is
    $$||p||^2 = p \cdot p = x^2 + y^2 + z^2$$
  - So a unit sphere at the origin is defined by
    $$p^2 = x^2 + y^2 + z^2 = 1$$

Ray-Sphere Intersections

- We have the equations:

$$p = p_0 + \lambda d$$

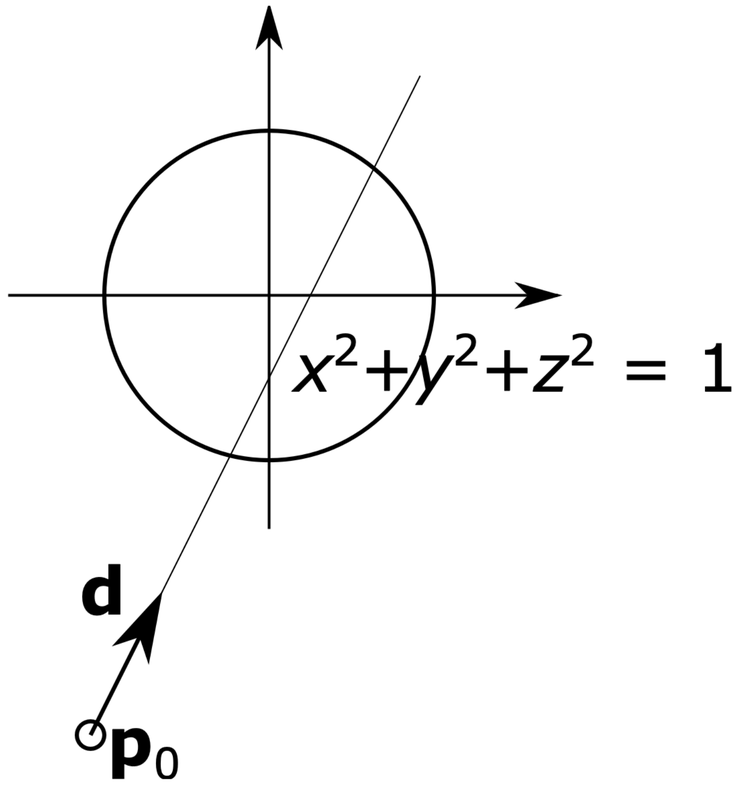
and

$$p \cdot p = 1$$

So

$$(p_0 + \lambda d) \cdot (p_0 + \lambda d) = 1$$

$$p_0 \cdot p_0 + 2\lambda p_0 \cdot d + \lambda^2 d \cdot d = 1$$



Ray-Sphere Intersections

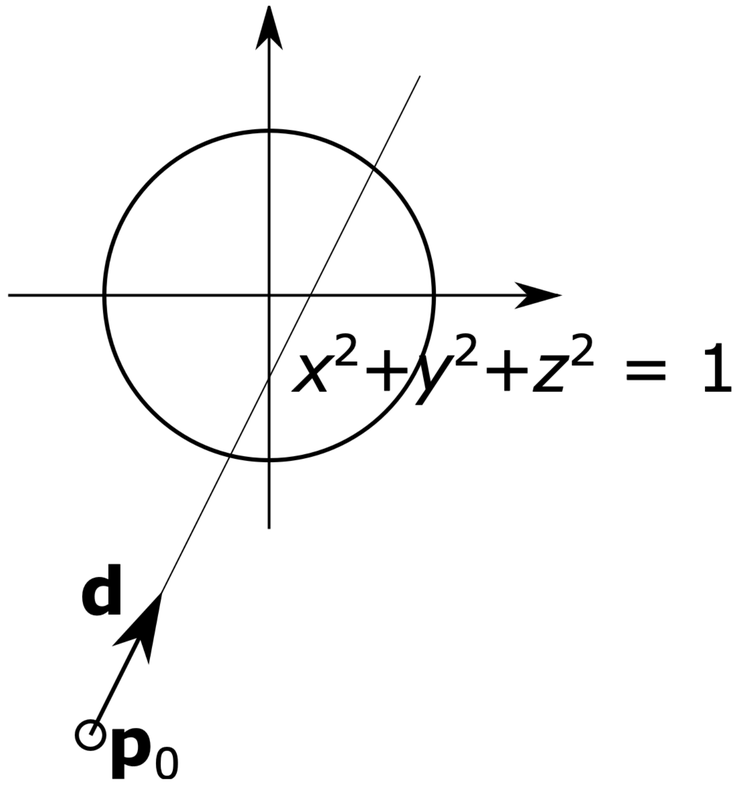
- This is a quadratic equation in

$$a\lambda^2 + b\lambda + c = 0$$

- So the solutions are

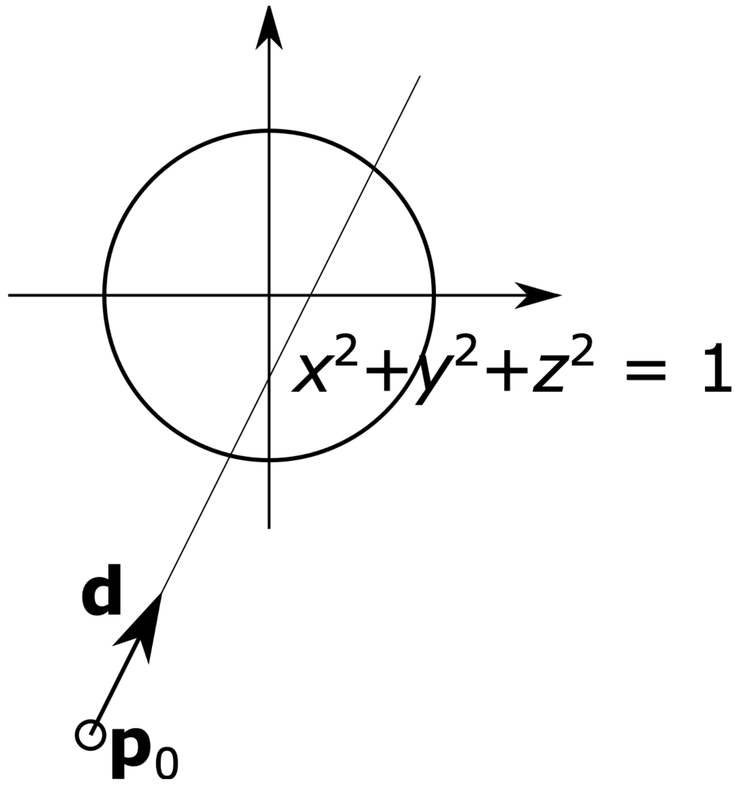
$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- 0,1, or 2 real solutions (what do they mean)



Ray-Sphere Intersections

- 0 solutions:
  - The ray passes next to the sphere
  - No intersection
  - No shading at this pixel
  - This is the no-hit case
- 1 solution:
  - The ray just touches the sphere at exactly one point
  - This point is called a tangent point
- 2 solutions:
  - The ray hits the sphere on the way in → first intersection: t1
  - The ray hits again on the way out → second intersection: t2
  - These two points are where the ray cuts through the sphere



Ray-Sphere Intersections

- What if the sphere is not at the origin?
  - We can represent this by a translation
  - More generally we can have any transformation, T
  - Represented as a 4×4 homogeneous matrix
- Easier to apply the transform to the ray
  - Apply the inverse transform, T^{-1} to the ray

Ray-Cube Intersections

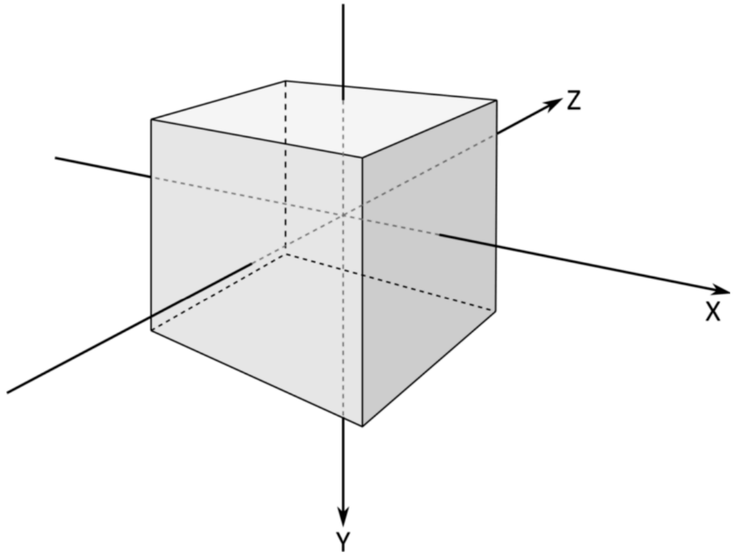
- A unit cube
  - From -1 to +1 on each axis
  - Intersect ray with each face
- X coordinate of a ray is

  $$x_0 + \lambda \Delta x$$

  - So at the plane $X = 1$ we have

    $$1 = x_0 + \lambda \Delta x$$

    -> Solve for $\lambda = \frac{1 - x_0}{\Delta x}$

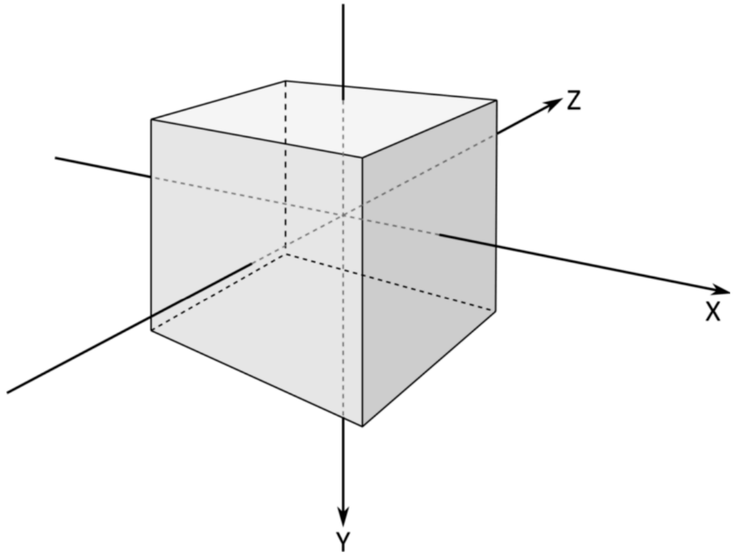


Ray-Cube Intersections

- Once we have λ, check whether we are in the face:

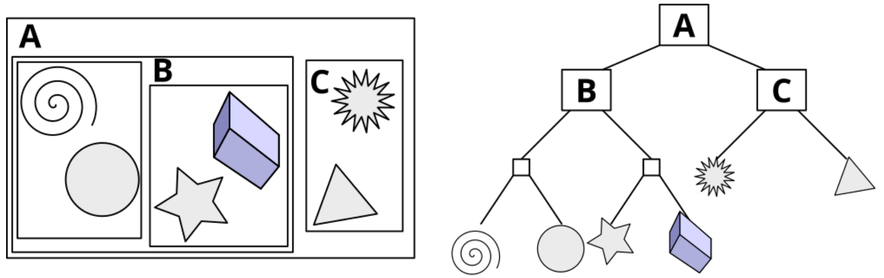
$$y_0 + \lambda \Delta y \in [-1,1]$$
$$z_0 + \lambda \Delta z \in [-1,1]$$

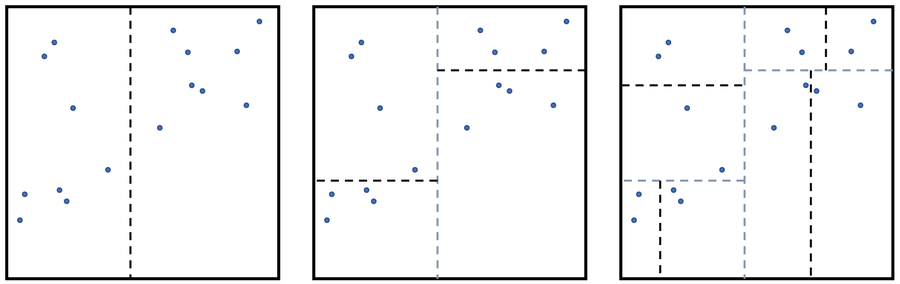
- Problems if Δx = 0:
  -> parallel
- Repeat for all 6 faces



Acceleration Structures

- Reduce the number of ray–primitive tests
- Common structures:
  - BVH (Bounding Volume Hierarchy): widely used due to efficient construction and traversal.
  - KD-tree: effective for static scenes requiring high traversal performance
  - Uniform grids / octrees: beneficial for scenes with known spatial regularity
- Key trade-off: build time vs. traversal efficiency





Real-time Raytracing

- Hardware acceleration (e.g., dedicated RT cores) -> significantly improved ray-tracing performance, enabling more efficient execution of complex lighting and visibility calculations
- AI-driven techniques are increasingly integrated into the ray-tracing pipeline, supporting tasks such as shading, denoising, and reconstruction with higher quality and stability
- Machine-learning-based denoisers have begun to replace traditional handcrafted approaches, providing cleaner images with fewer artefacts at lower sample counts
- Contemporary rendering systems now adopt hybrid pipelines, combining rasterisation, ray tracing, neural rendering, and volumetric methods to balance visual fidelity and performance:
  - Ray-traced shadows
  - Ray-traced reflections
  - Ray-traced ambient occlusion

RadFoam: Volumetric mesh ray tracing algorithm

The end!