Lecture #16

Ray Tracing - Basics

Computer Graphics
Winter Term 2016/17

Marc Stamminger / Roberto Grosso

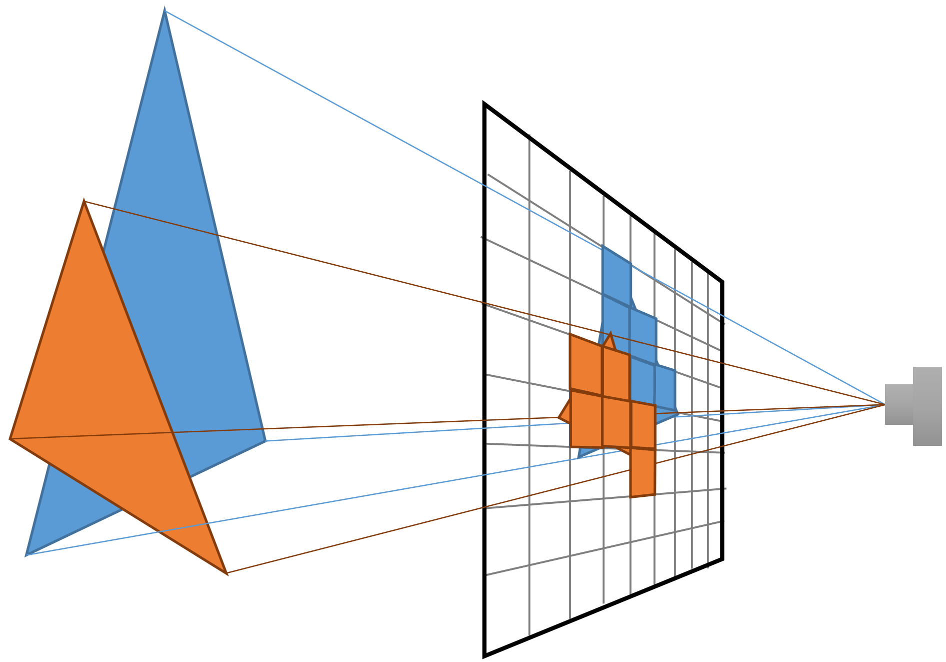
Introduction

- Up to now: **Rasterization**
  - Scanline algorithm
  - Local illumination using Phong lighting

- **Ray Tracing:**
  A different rendering paradigm
  - More illumination effects → Global Illumination
  - Physically motivated illumination computations

Up to now: Rasterization

```
for each triangle t
    find pixels inside t
        shade pixel
```

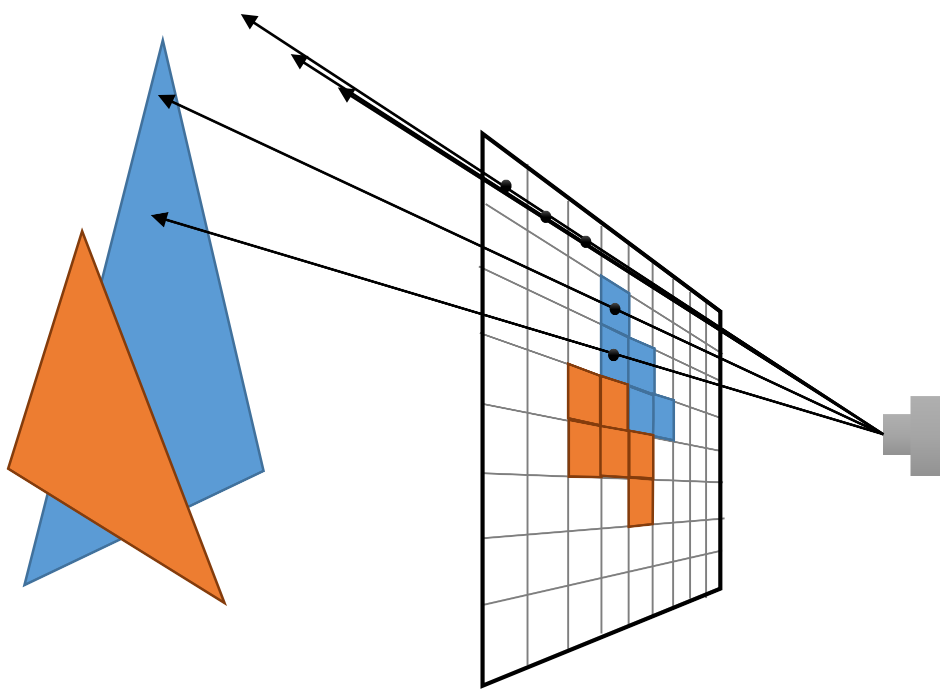


Ray Casting

- Ray Casting ⊂ Ray Tracing

```
for each pixel p
    cast a ray through pixel p
        shade p
```

= "find scene point visible in p"

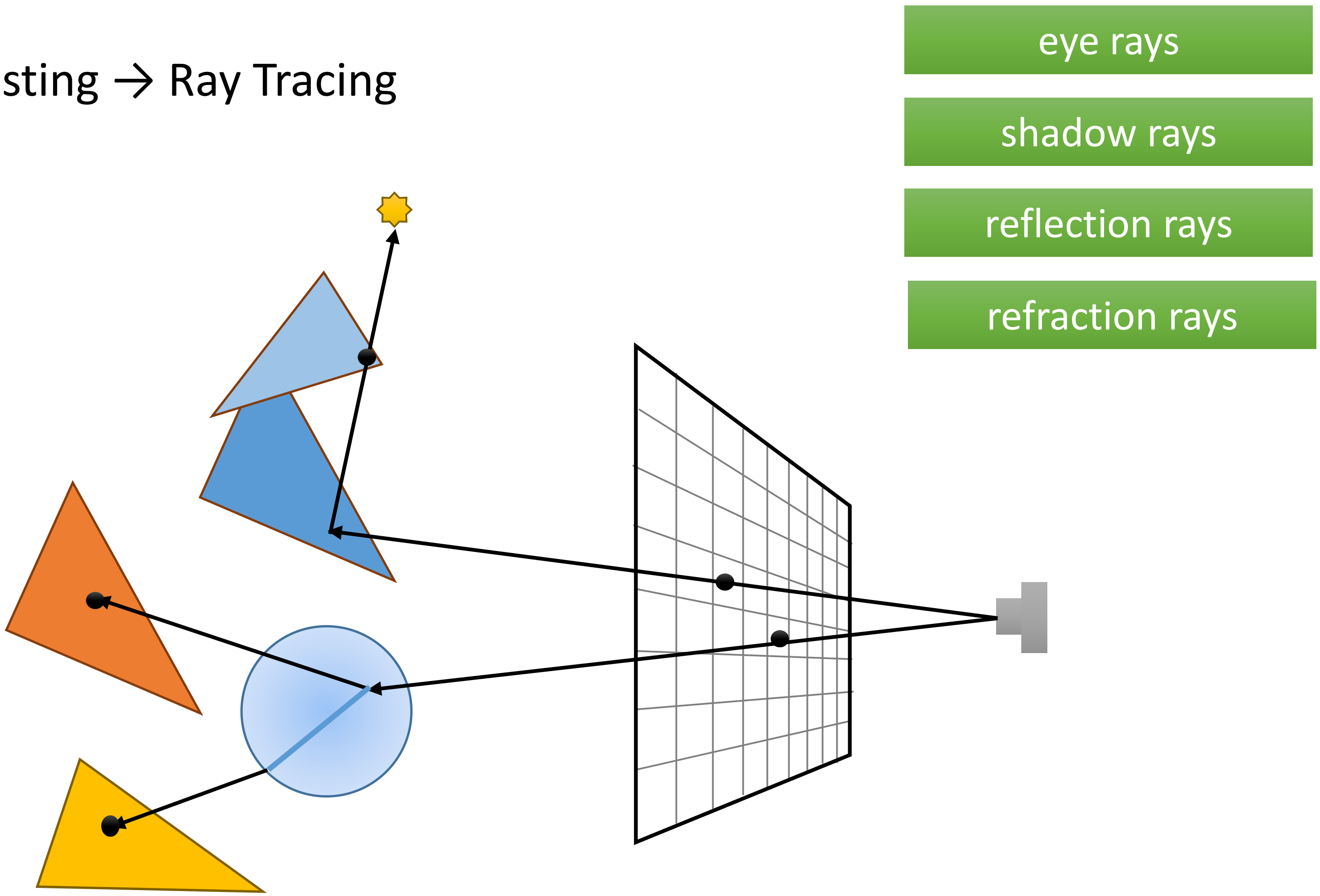


Ray Casting → Ray Tracing

- Having a method at hand that intersects a ray with our scene, we use this to generate new lighting effects that are not directly possible with rasterization
  → reflections
  → refractions
  → shadows
  → indirect illumination (later)
  …

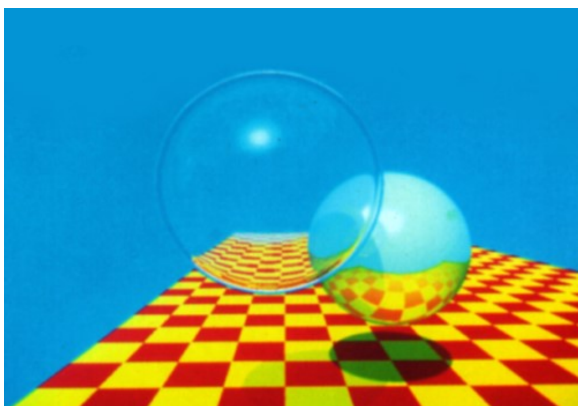
Ray Tracing

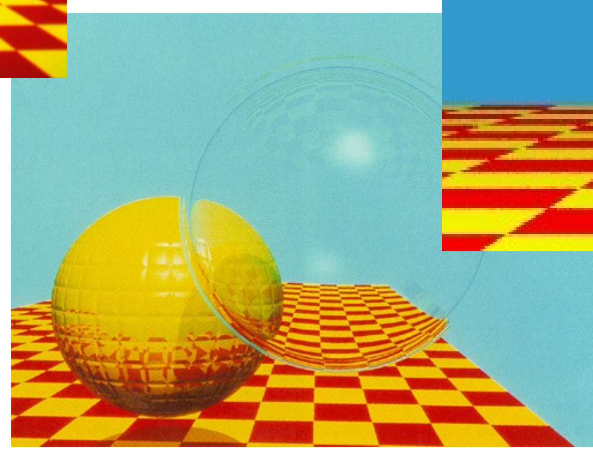
- Ray Casting → Ray Tracing

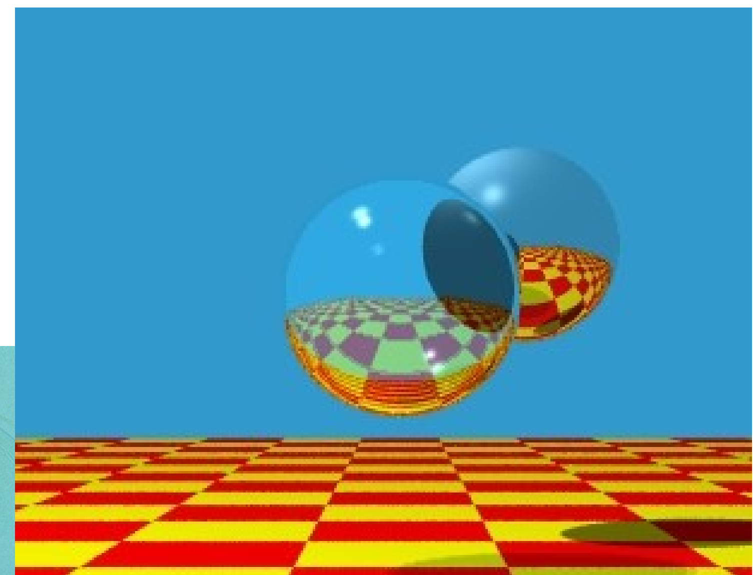


Introduction

- 1968: Ray Casting: Arthur Appel
- 1979: Recursive ray tracing: Turner Whitted







reflection
and
refraction

Images by Turner Whitted

Introduction

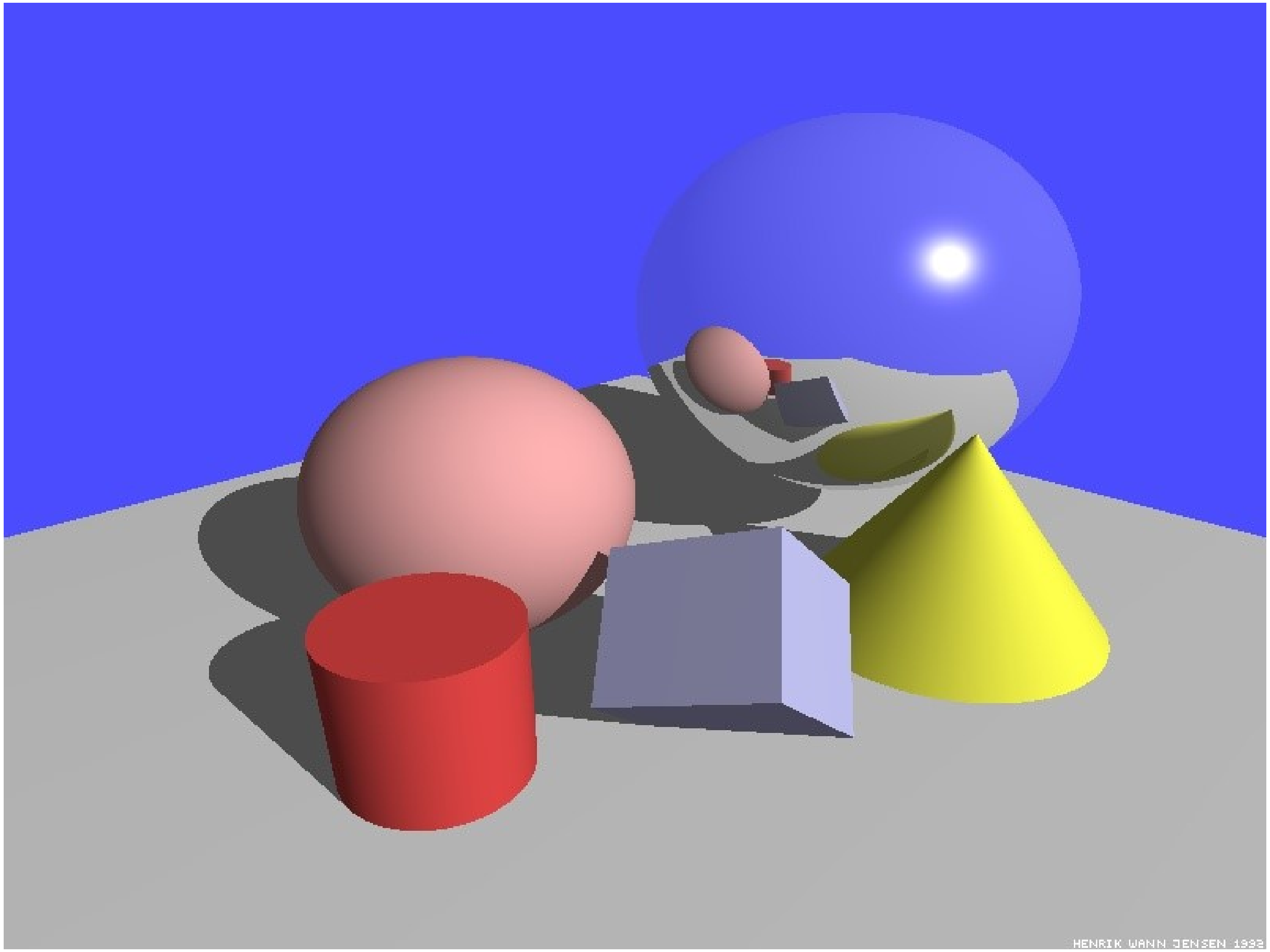


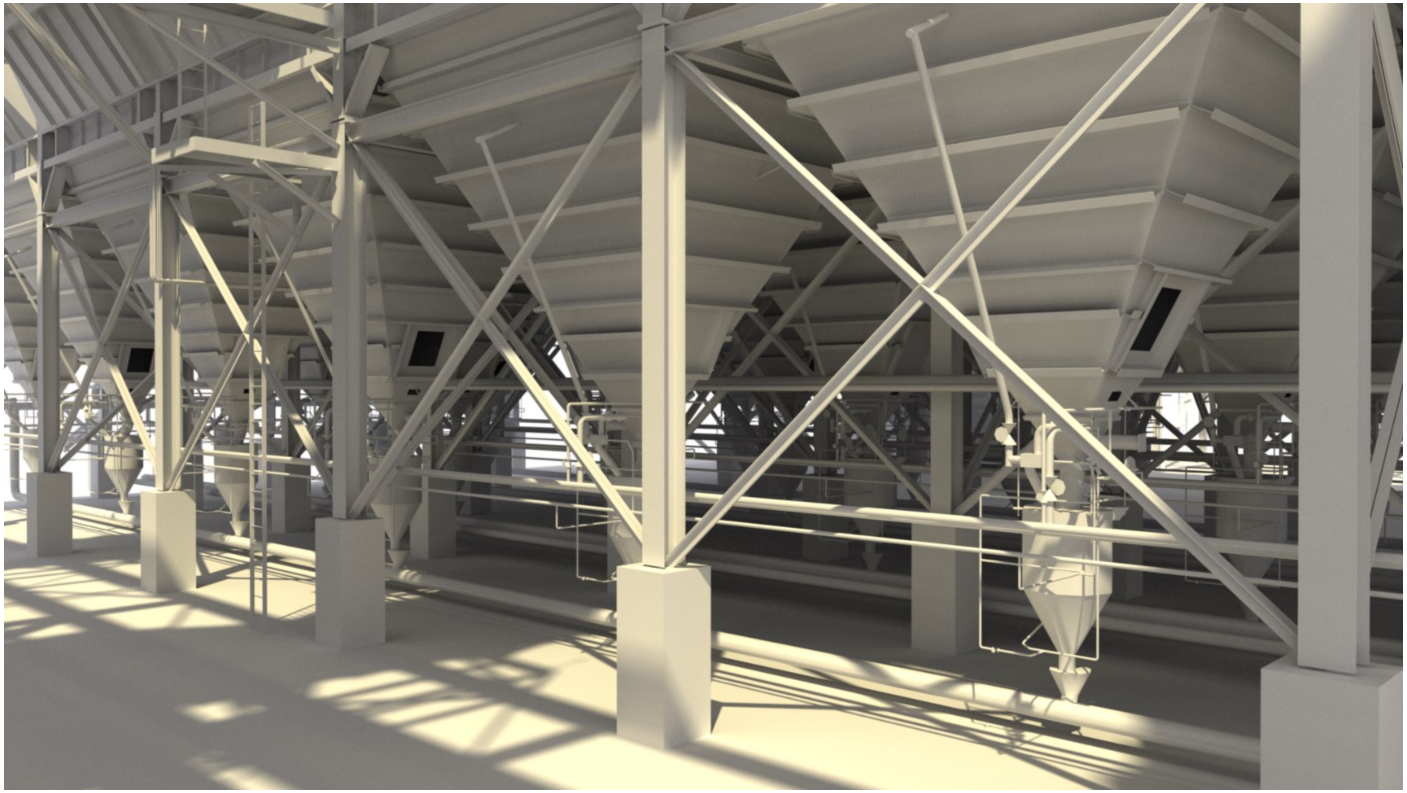
Image by Henrik Wann Jensen. He writes: One of my first ray tracing images (1990-1991).
Rendered first time on an Amiga in HAM mode (the good old days).

Introduction

- Car with reflections, refractions, environment lighting

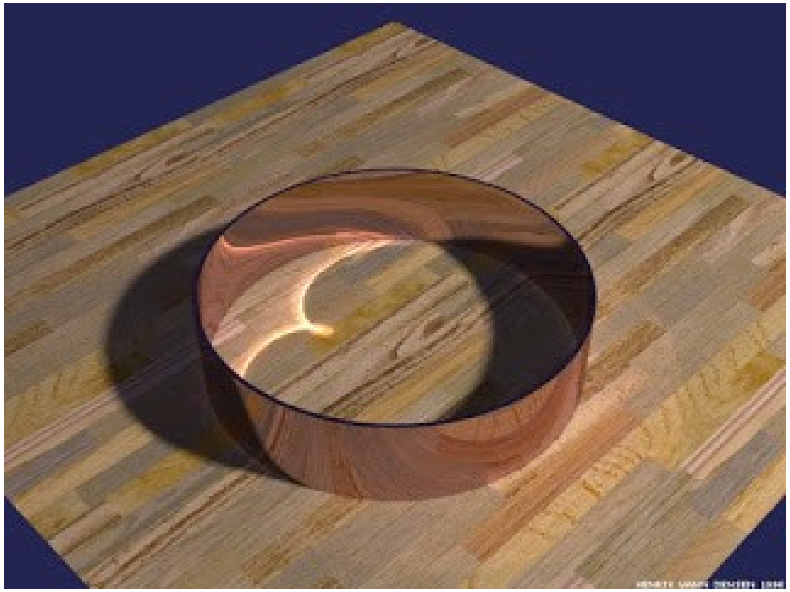
Introduction

- Indirect Illumination → not possible with simple ray tracing

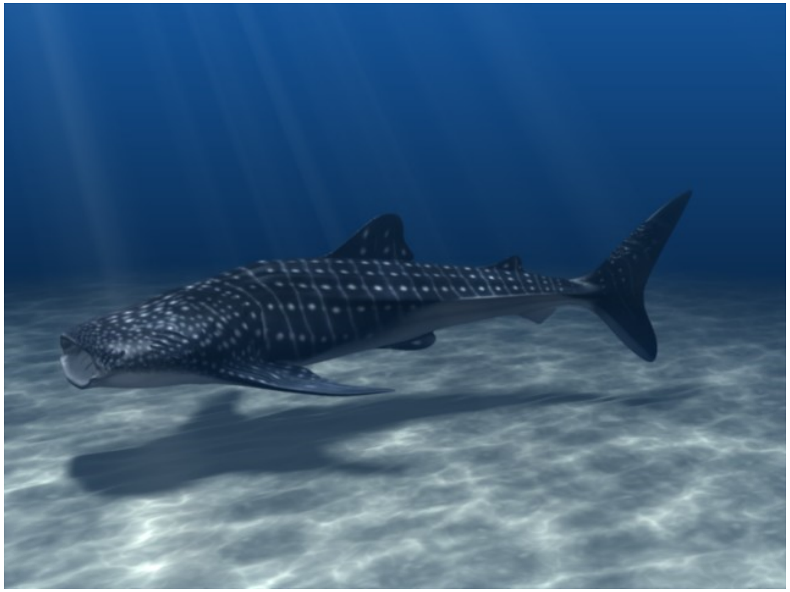


Introduction

- Caustics: light patterns generated by reflections off specular surfaces
  → not possible with simple ray tracing



graphics.ucsd.edu



https://blenderartists.org/forum/showthread.php?116585-Realistic-underwater-lighting-and-caustics-added-tutorial-link

Ray Tracing

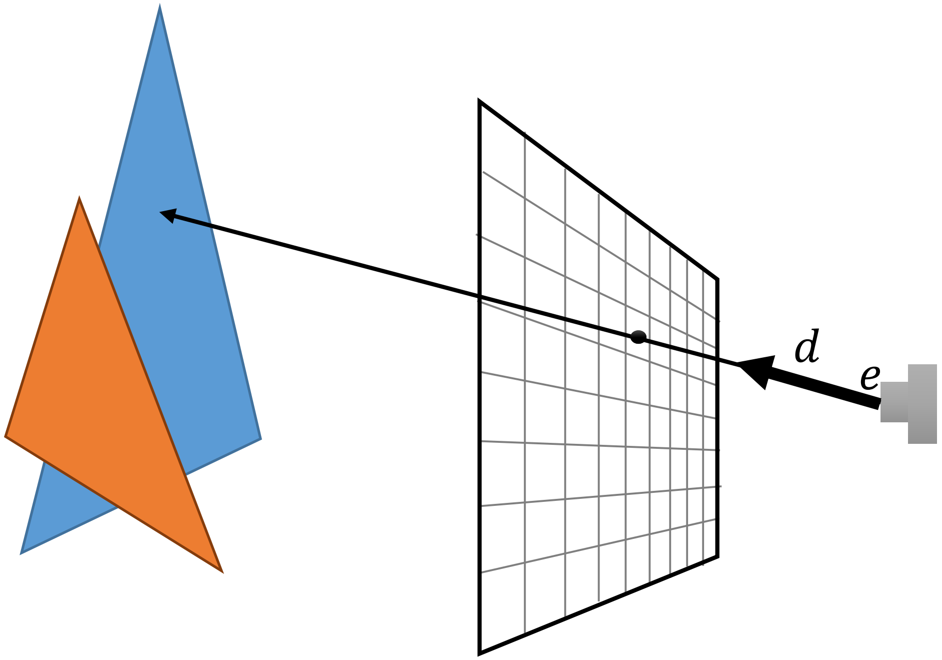
- Today: Basics of Ray Tracing
  - how to generate eye rays
  - how to intersect a ray with scene geometry
  - a first ray caster

- Tomorrow:
  - how to generate secondary rays
  - recursive ray tracing procedure

- Next weeks:
  - accelerations structures for fast ray tracing
  - special effects possible with ray tracing
  - the rendering equation

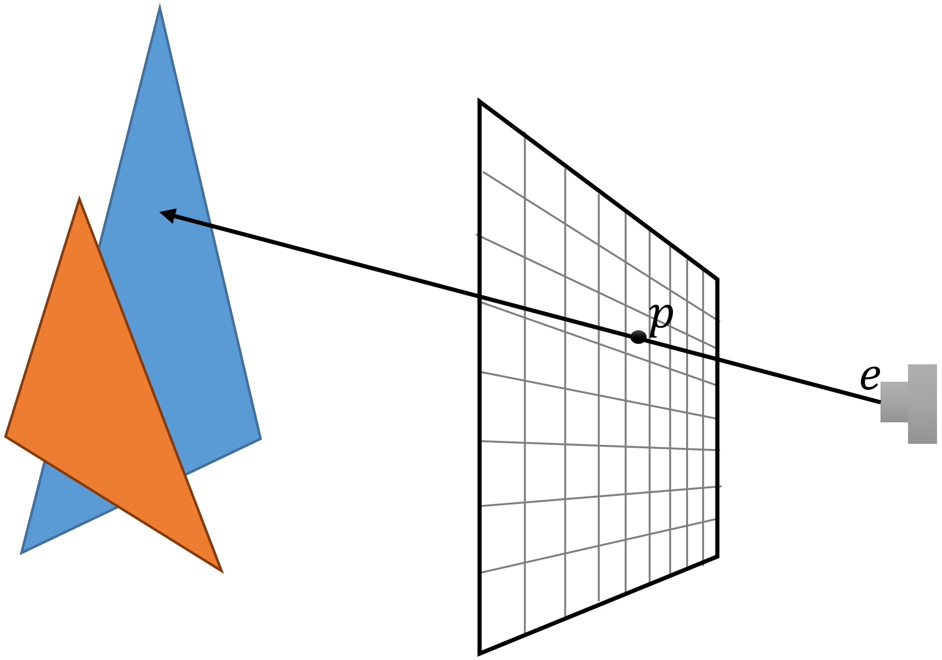
Rays

- Mathematical representation of an (eye) ray
  - Parametric line from ray origin (eye) e in direction d
    $$p(t) = e + td$$
  - $p(0) = e$
  - $t_1, t_2 > 0$ and $t_1 < t_2 \Rightarrow$ $p(t1)$ closer to the eye than $p(t2)$
  - $t < 0 \Rightarrow p(t)$ behind the eye



- Ray test:
  - find intersection of ray with scene with smallest $t > 0$

Eye Ray Generation



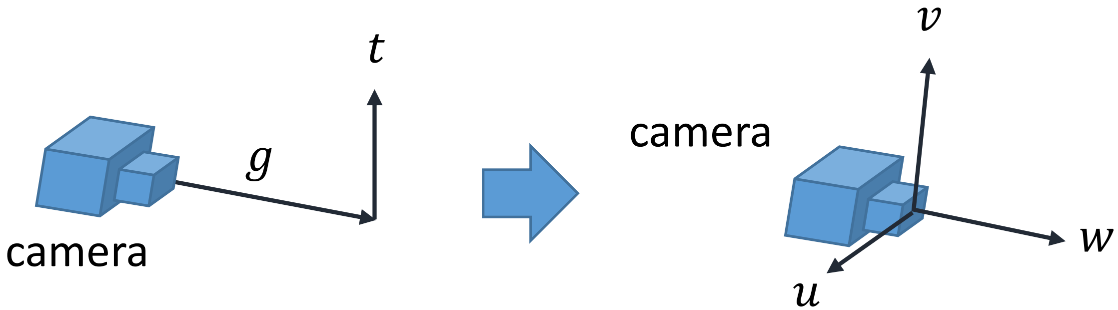
- Every eye ray belongs to one pixel
- Starting point of eye ray: camera
- Eye ray goes through pixel on image plane

- For a particular pixel, the eye ray is:
  $e =$ camera position
  $d = (p - e)/||p - e||$

- Intersection with objects:
  t-values ($t > 0$)
  - Smallest $t \Rightarrow$ first intersection $\Rightarrow$ visible object

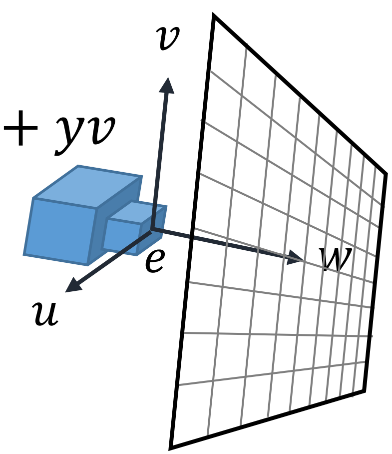
Eye Ray Generation

- Remember from Lecture #07: Viewing
  Given camera position, view direction and up-vector
  → compute camera basis vectors u, v, w



- Use this to generate an image plane:

$$e + w + xu + yv$$



Eye Ray Generation

- Point (x, y) on image plane:

$$e + w + xu + yv$$

- Using the field of view *fovy* and *aspect* ratio, a window is defined on the image plane:

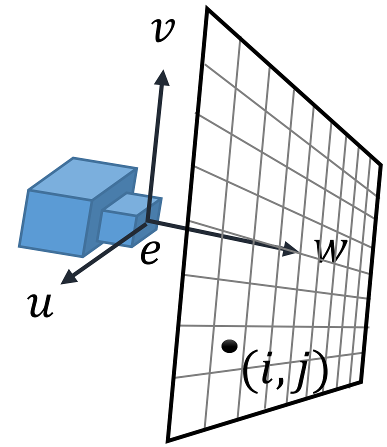
$$(x, y) \in [-x_m, x_m] \times [-y_m, y_m]$$

with $y_m = \tan\frac{fovy}{2}, x_m = aspect\ y_m$

- Finally, we have to map integer pixel coordinates (i, j) to this window:

$$x = \left(\underbrace{\frac{i+0.5}{n_x}}_{\text{relative coordinate}} \times 2 - 1\right) x_m, \qquad y \text{ analog}$$

number of pixels in x



Eye Ray Generation

- Eye ray computation:
  - compute (u, v, w) for camera frame
  - for pixel (i, j): eye ray is $e + td$ with

  e = camera position

  $x = \left(\frac{i+0.5}{n_x} \times 2 - 1\right) \times aspect \times \tan\frac{fovy}{2}$ and

$$y = \left(\frac{j + 0.5}{n_y} \times 2 - 1\right) \times \tan\frac{fovy}{2}$$

$$d = \frac{\mathrm{w} + \mathrm{xu} + \mathrm{yv}}{\|w + xu + yv\|}$$

Eye Ray Generation

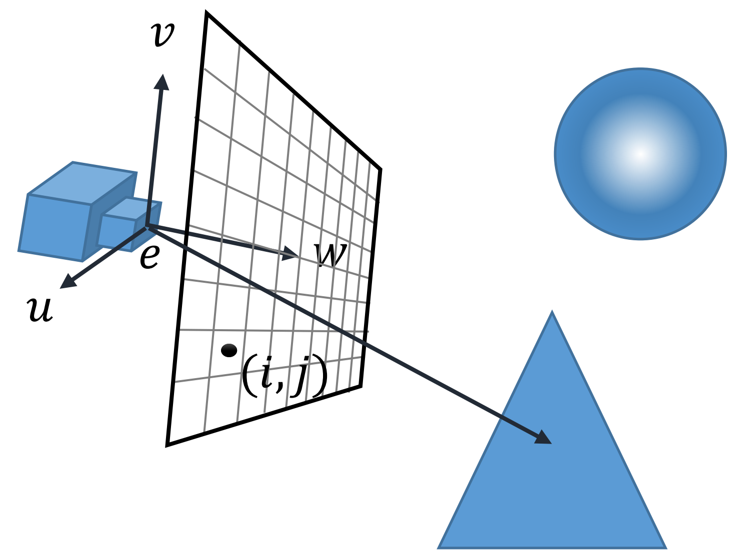
- With this model, we can handle other camera types than projective pin hole cameras, e.g. panorama cameras, fish eye lenses or similar

- This is not possible with rasterization and a pinhole camera (why?)

Ray – Object Intersection

- Does ray intersect a scene object ? And if so, at which t ?

- Today we look at triangles, polygons, and spheres



Ray – Object Intersection

- Planes and plane equations
  - Normal

$$n = (B - A) \times (C - A)/\|(B - A) \times (C - A)\|$$

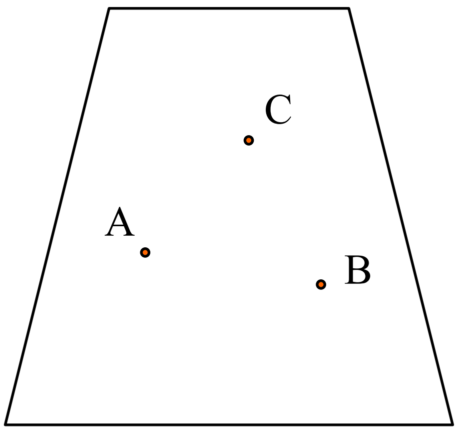
  - plane equation, point-normal form

$$n \circ (x - A) = 0$$

  - constant-normal form

$$n \circ x = d \qquad (= \mathrm{n} \circ A)$$

  - d is the distance to the origin of the coord. system



Ray – Object Intersection

- Halfspaces
  - positive half space $\{x \in \mathbb{R}^3 : n \circ x - d > 0\}$
  - negative half space $\{x \in \mathbb{R}^3 : n \circ x - d < 0\}$
  - the positive halfspace lies on the side in which the normal points, and the negative halfspace on the opposite side.
  - a point is in front of the plane, if it is in the positive halfspace, and a point is behind the plane, if it is in the negative halfspace
- Distance
  - $n \circ x - d$ is the distance of x to the plane

Ray - Plane intersection

- Parametric representation of a ray
  $p(t) = e + td, \|d\| = 1$ or
  $p(t) = e + t(s - e)$, where s is a point in space, e.g. pixel mid point

- Implicit equation of a plane
  $$\pi = (n, d) = \{x \in \mathbb{R}^3 : n \cdot x = d\}$$
  - n is the plane's normal, if $\|n\| = 1, d$ is the distance of the plane to the origin

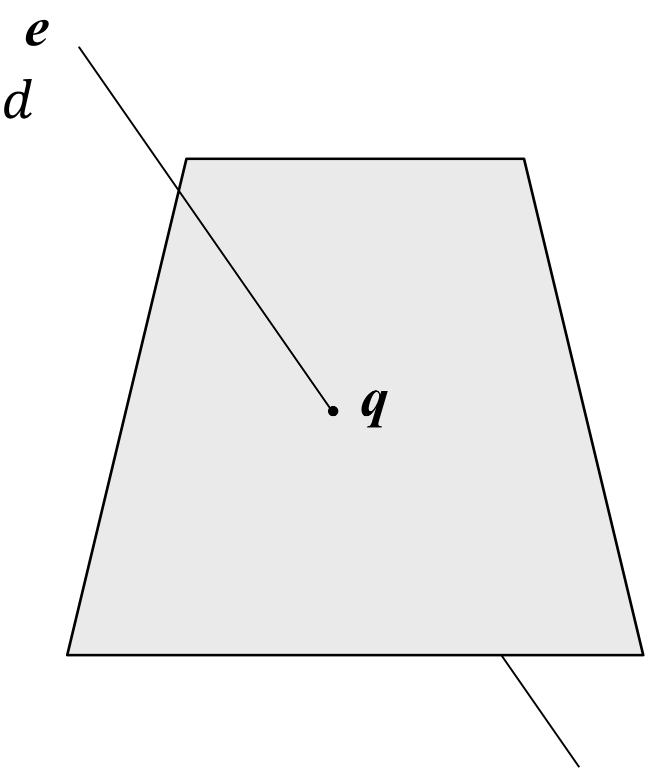
Ray - Plane intersection

- Set p(t) in the implicit equation for the plane π

$$n \circ p(t) = d$$

$$t = \frac{d - n \circ e}{n \circ d}$$

$$q = e + \frac{d - n \circ e}{n \circ d} d$$

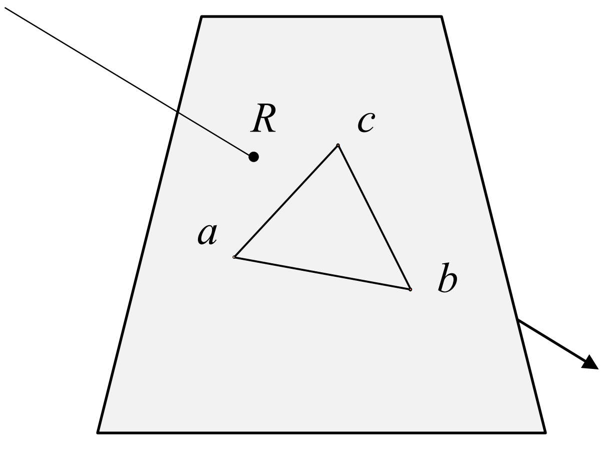


Ray - Triangle intersection

- Many algorithms exist for this problem
- Here barycentric coordinates are used for the parametric plane containing the triangle
- System of equations:

$$e + td = a + \beta(b - a) + \gamma(c - a)$$

- Unknowns: t, β, γ
- Intersection at $R = e + td$ if
  - $t > 0$ (R on positive part of ray)
  - $\beta > 0, \gamma > 0, \beta + \gamma < 1$ (R within triangle)



Ray - Triangle intersection

- Solve $e + td = a + \beta(b - a) + \gamma(c - a)$:

$$\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ d & a-b & a-c \\ \vdots & \vdots & \vdots \end{pmatrix}}_{M} \begin{pmatrix} t \\ \beta \\ \gamma \end{pmatrix} = a - e$$

Ray - Triangle intersection

- Solve using Cramer's rule:
  - $t = \det\begin{pmatrix} \vdots & \vdots & \vdots \\ a-e & a-b & a-c \\ \vdots & \vdots & \vdots \end{pmatrix} / \det A$
  - $\beta = \det\begin{pmatrix} \vdots & \vdots & \vdots \\ d & a-e & a-c \\ \vdots & \vdots & \vdots \end{pmatrix} / \det A$
  - $\gamma = \det\begin{pmatrix} \vdots & \vdots & \vdots \\ d & a-b & a-e \\ \vdots & \vdots & \vdots \end{pmatrix} / \det A$
- If $\det A = 0$, then
  - The triangle is degenerate (a line or a point) or
  - The ray is parallel to the triangle

Ray - Triangle intersection

```
Boolean raytri (ray r, vector3 a, vector3 b,
                vector3 c, interval[t0,t1])
       compute t
       if (t < t0) or (t > t1) then
              return false
       compute γ
       if (γ < 0) or (γ > 1) then
              return false
       compute β
       if (β < 0) or (β > 1) then
              return false
       if (β + γ >1)  then
      return false
   else
      return true
```

Ray - Triangle intersection

```
// Given segment pq and triangle abc, returns whether segment intersects
// triangle and if so, also returns the barycentric coordinates (u,v,w)
// of the intersection point
int IntersectSegmentTriangle(Point p, Point q, Point a, Point b, Point c,
                             float &u, float &v, float &w, float &t)
{
    Vector ab = b - a;
    Vector ac = c - a;
    Vector qp = p - q;

    // Compute triangle normal. Can be precalculated or cached if
    // intersecting multiple segments against the same triangle
    Vector n = Cross(ab, ac);

    // Compute denominator d. If d <= 0, segment is parallel to or points
    // away from triangle, so exit early
    float d = Dot(qp, n);

    if (d <= 0.0f) return 0;

    // Compute intersection t value of pq with plane of triangle. A ray
    // intersects iff 0 <= t. Segment intersects iff 0 <= t <= 1. Delay
    // dividing by d until intersection has been found to pierce triangle
    Vector ap = p - a;
```

Ray - Triangle intersection

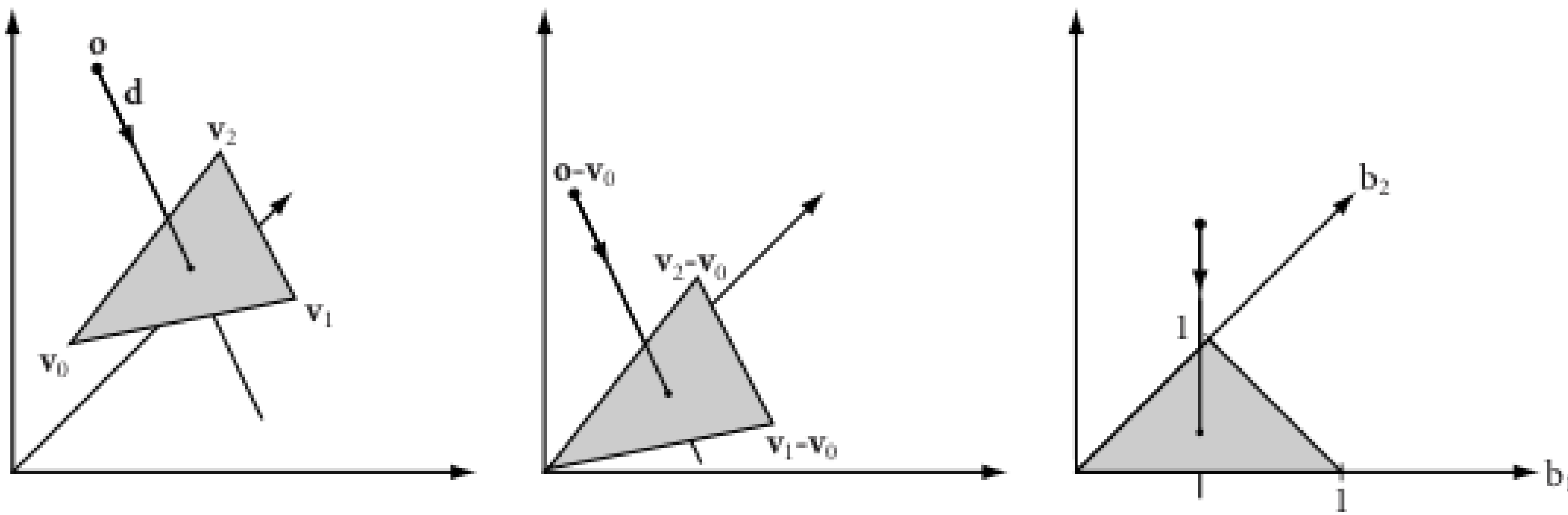
```
    t = Dot(ap, n);
    if (t < 0.0f) return 0;

    // Compute barycentric coordinate components and test if within bounds
    Vector e = Cross(qp, ap);
    v = Dot(ac, e);
    if (v < 0.0f || v > d) return 0;
    w = -Dot(ab, e);
    if (w < 0.0f || v + w > d) return 0;

    // Segment/ray intersects triangle. Perform delayed division and
    // compute the last barycentric coordinate component
    float ood = 1.0f / d;
    t *= ood;
    v *= ood;
    w *= ood;
    u = 1.0f - v - w;
    return 1;

}
```

Ray - Triangle intersection



Interpretation of ray-triangle intersection test, Möller Trumbore

Ray – Polygon intersection

- Ray Polygon Intersection
  - Given a planar polygon with
    - m vertices p_1 through p_m
    - All on a plane with normal n
  - Method
    - 1) compute intersection p of ray with the plane containing the polygon
    - 2) test if p is within the polygon

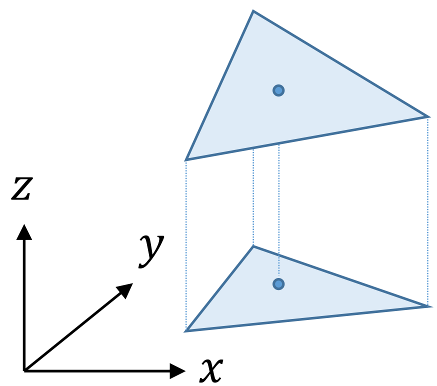
Ray – Polygon Intersection

- Ray Polygon Intersection
  - Ray $e + td$
  - Compute intersection with plane containing polygon

$$p = e + \frac{(p_1 - e) \circ n}{d \circ n} d$$

Ray - Polygon Intersection

- Location of p with respect to polygon
  - Project polygon and p onto the coordinate plane with the largest projection.
  - Then do 2D inside-outside test



In this example, we would project
the triangle to the xy-plane because
the normal has maximal component in z

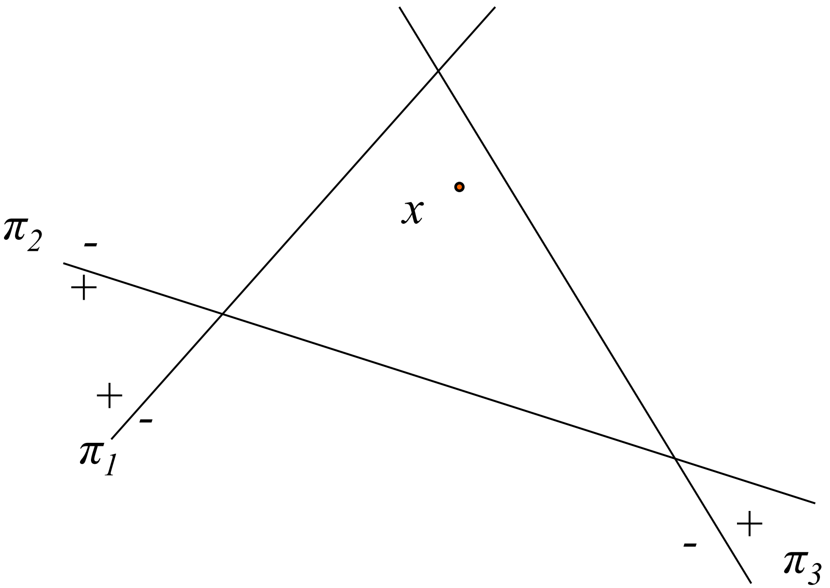
Ray - Polygon intersection

- Two different cases:
    - Polygon is convex → simple edge tests possible
    - Polygon can be non-convex → more complicated in/out test needed

Ray - Polygon intersection

- Convex polytopes (polyhedra)
  - A polyhedron (convex polygon in 2D) can be described as the intersection of a set of halfspaces
  - a point x is inside the polyhedron, if it is behind all the halfspaces

$$n_1 \circ x - d_1 < 0$$
$$n_2 \circ x - d_2 < 0$$
$$n_3 \circ x - d_3 < 0$$

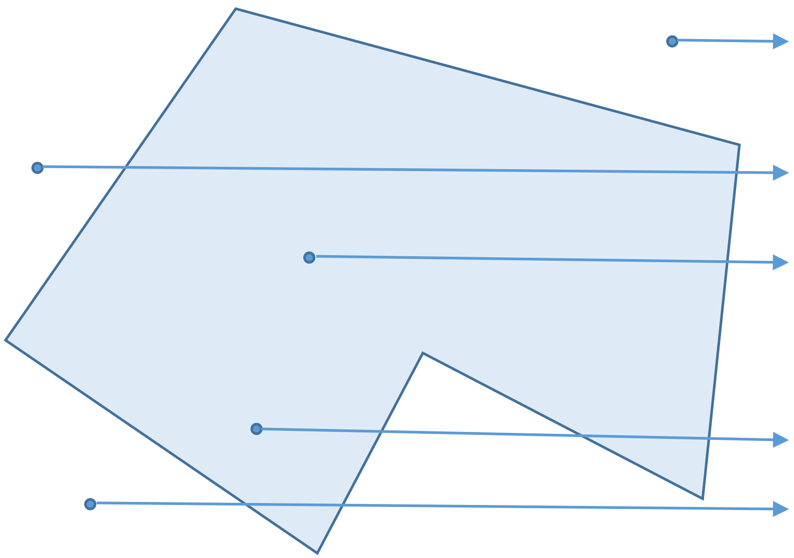


- Inside – Outside test for convex polygons
  - convert edges to half space representation
  - check point for all half spaces
    → as soon as one fails, point is outside
    → if none fails, point is inside
  - ideally: half space vectors stored with triangle
    → fast, but consumes memory

Ray - Polygon intersection

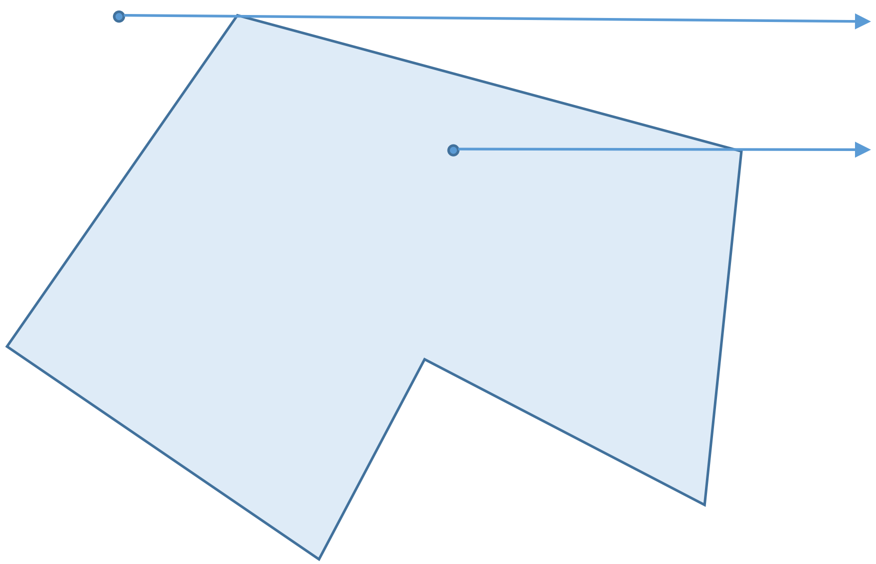
- For non-convex polygons, the previous test is not correct
  → example ?

- General polygon inside-outside test
  - Generate ray from point in arbitrary direction
  - Count intersections with polygon boundary
    - Even -> outside
    - Odd -> inside



Ray - Polygon intersection

- Polygon inside-outside test
  - Problem: ray hits one vertex
    → should it count twice or once?
  - in the example on the right, the upper ray should count two intersections, the lower one only one…
  - Simple robust solution:
    If such a boundary case is detected, use other ray with new direction



Ray – Sphere Intersection

- Implicit surface equation $f(x) = 0$
- Example: sphere with center c and radius r:

$$(x - c) \circ (x - c) - r^2 = 0$$

- Set the ray in the implicit equation and find t and the intersection point p, if possible
  $f\big(p(t)\big) = 0 \Rightarrow$ ray parameter t

Ray – Sphere Intersection

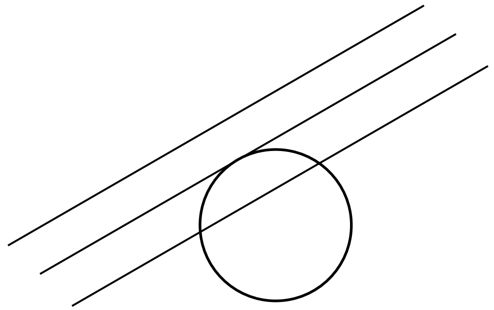
- Intersection with ray $p(t) = e + td$:
$$(e + td - c) \circ (e + td - c) - r^2 = 0$$
- Results in quadratic equation
$$(d \circ d)t^2 + 2\, d \circ (e - c)t + (e - c) \circ (e - c) - r^2 = 0$$
- Since $(d \circ d) = 1$:

$$t = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$$

with $b = 2\, d \circ (e - c)$ and $c = (e - c) \circ (e - c) - \mathrm{r}^2$

Ray – Sphere Intersection

- Meaning of the discriminant $b^2 - 4c$
  - If negative → no intersections
  - If positive → two solutions
    - where ray enters the sphere
    - where ray leaves the sphere
  - If zero → ray touches sphere at exactly one point



- Always check discriminant first!
- Remark: sphere-ray intersection important
- Using sphere as bounding object for more complex objects
- Provides sufficient information about existence of intersection

Ray – Quadric intersection

- Similar tests available for
  - cylinders
  - cones
  - tori
  - …

Ray Casting

- Up to now, we have the following:

```
for each pixel p in image plane
    generate eye ray (e,d) through pixel p
    tmin = infinity; omin = null;
    for each scene object o
        t = intersect ray (e,d) with object o
        if ray intersects object and t < tmin
            tmin = t;
            omin = o;
    if omin != null
        compute lighting at hit point on object omin
        set p to this color
    else
        set p to background color
```

just learned

just learned

???

Ray Casting - Lighting

- How to do the lighting at a found hit point?
- → we need the hit point, its surface normal, maybe texture coordinates etc.

- In a triangle mesh, these can be computed from barycentric coordinates of hit point
- Typically, this information is stored in a Hit-Object

```
struct Hit {
    float t; // ray parameter
    Obj *obj; // hit scene object
    float alpha,beta,gamma; // barycentric coordinates

    vec3 getPost() { … }
    vec3 getNormal() { … }
    vec2 getTexCoord() { … }
}

Hit Scene::intersect(Ray &ray) { … }
```

Ray Casting - Lighting

- new version

```
for each pixel p in image plane
    Ray ray = camera->getEyeRay(p);
    Hit closestHit = null;
    for each scene object o
        Hit hit = o.intersect(ray);
        if closestHit == null || hit.t < closestHit.t
            closestHit = hit;
    if closestHit != null
        c = closestHit.obj.shader.computeLighting(
                closestHit.getPos(),
                closestHit.getNormal(),
                …);
        setPixelColor(p,c);
    else
        setPixelColor(p,backgroundColor);
```

Next Lecture

- New effects using secondary rays
  - shadows, reflections, refractions
  - recursive ray tracing