Ray Casting II

Courtesy of James Arvo and David Kirk. Used with permission.

MIT EECS 6.837

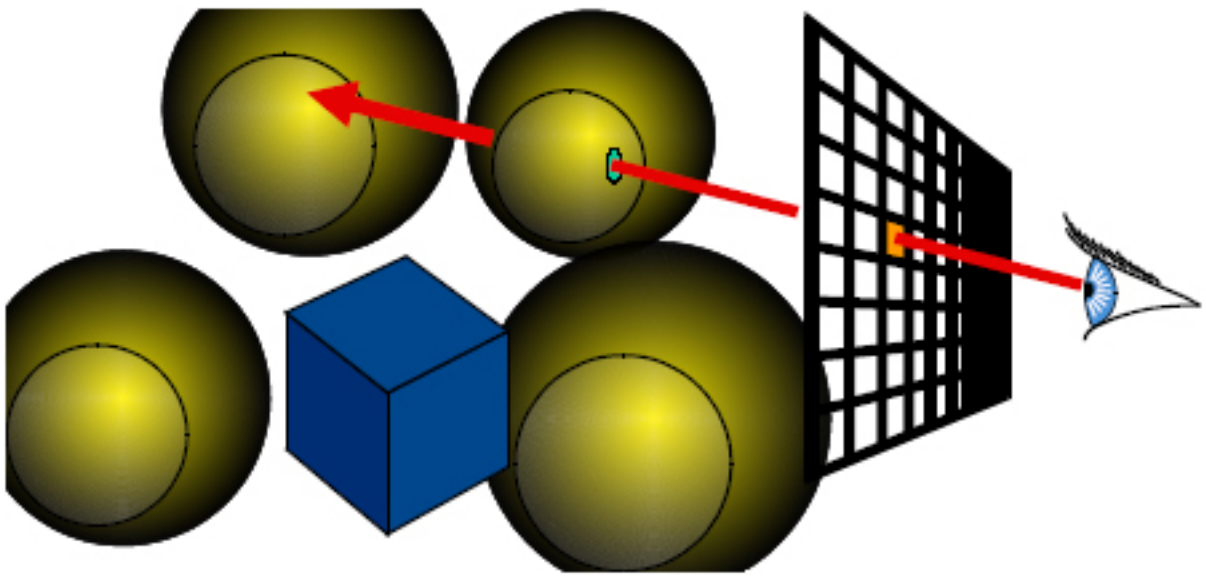
Frédo Durand and Barb Cutler

Some slides courtesy of Leonard McMillan

Review of Ray Casting

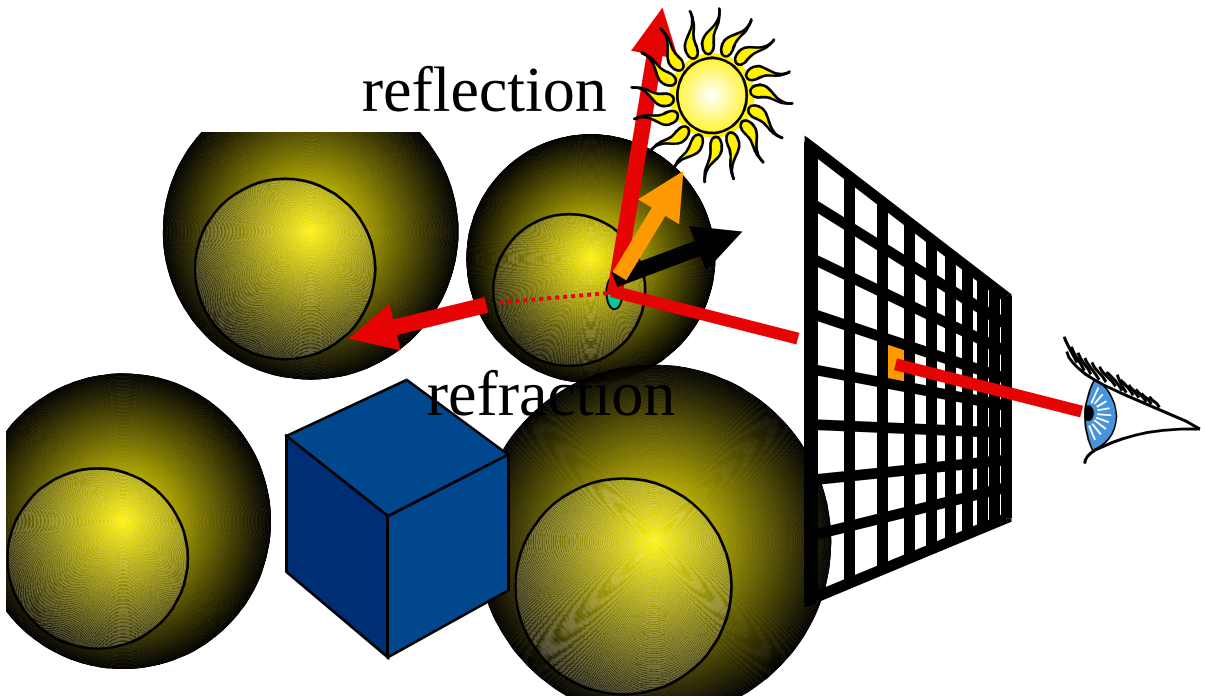
Ray Casting

```
For every pixel
   Construct a ray from the eye
   For every object in the scene
        Find intersection with the ray
        Keep if closest
```



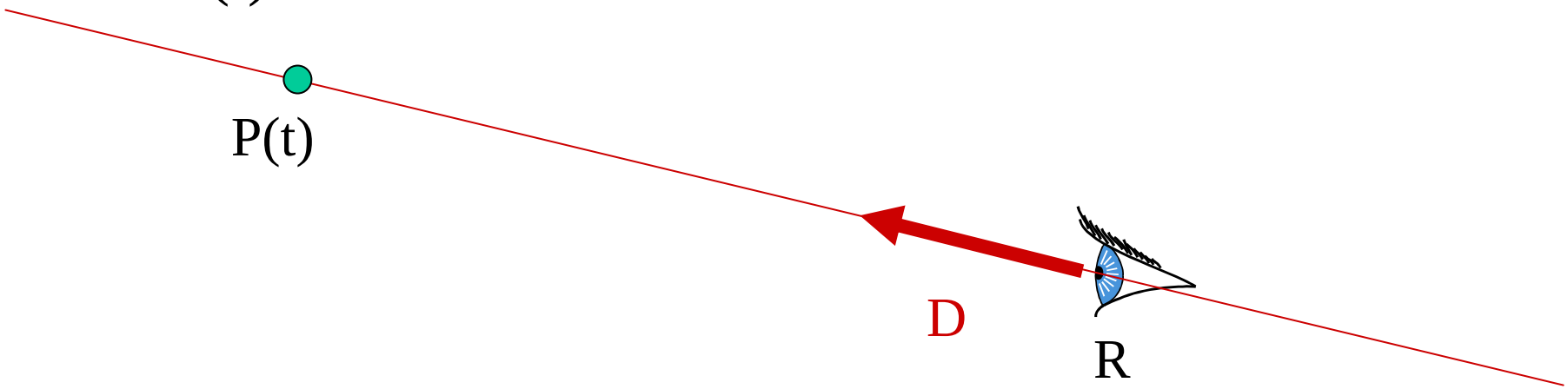
Ray Tracing

- Secondary rays (shadows, reflection, refraction)
- In a couple of weeks



Ray representation

- Two vectors:
  - Origin
  - Direction (normalized is better)
- Parametric line
  - P(t) = R + t * D

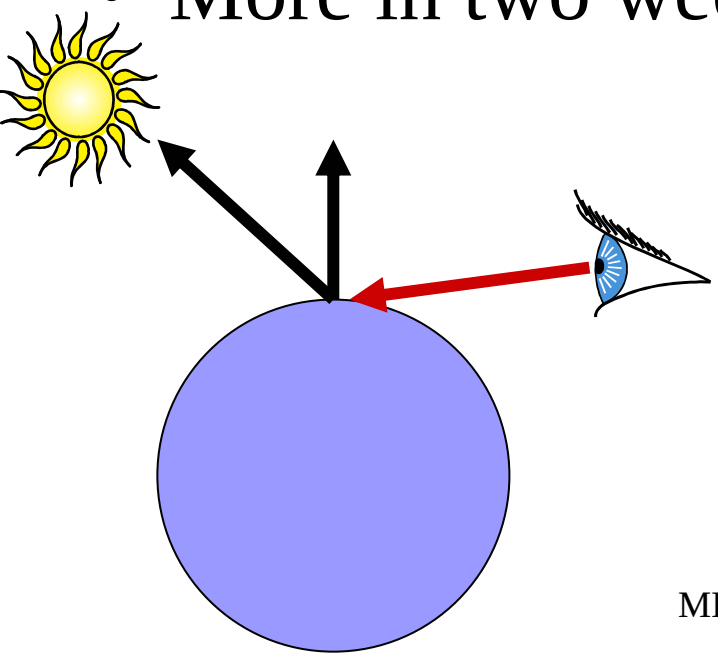


Explicit vs. implicit

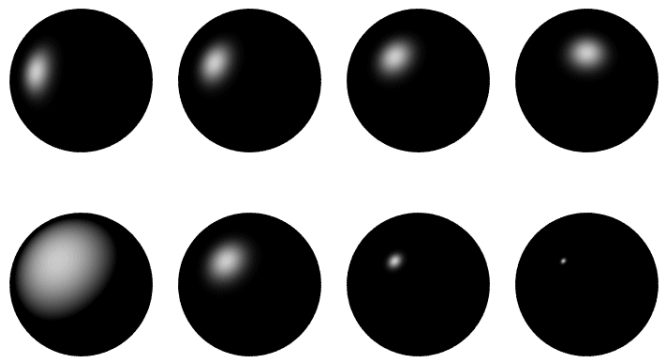
- Implicit
  - Solution of an equation
  - Does not tell us how to generate a point on the plane
  - Tells us how to check that a point is on the plane
- Explicit
  - Parametric
  - How to generate points
  - Harder to verify that a point is on the ray

A note on shading

- Normal direction, direction to light
- Diffuse component: dot product
- Specular component for shiny materials
  - Depends on viewpoint
- More in two weeks



Diffuse sphere



shiny spheres

Textbook

- Recommended, not required
- Peter Shirley
  Fundamentals of Computer Graphics
  AK Peters

References for ray casting/tracing

- Shirley Chapter 9
- Specialized books:
  - An Introduction to Ray Tracing, Edited by Andrew S. Glassner
  - Realistic Ray Tracing, Peter Shirley
  - Realistic Image Synthesis Using Photon Mapping, Henrik Wann Jensen
- Online resources

http://www.irtc.org/

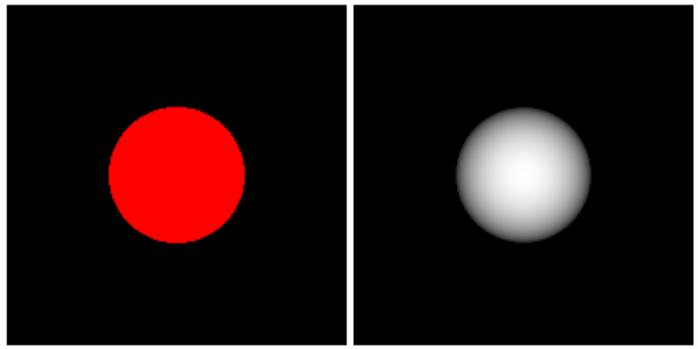
http://www.acm.org/tog/resources/RTNews/html/

http://www.povray.org/

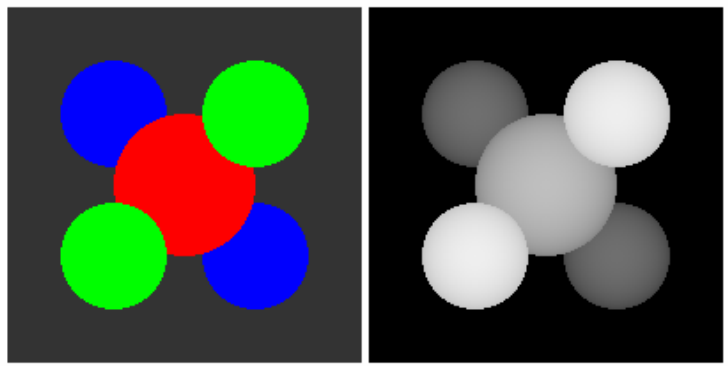
http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtrace0.htm

http://www.siggraph.org/education/materials/HyperGraph/raytrace/rt_java/raytrace.html

Assignment 1

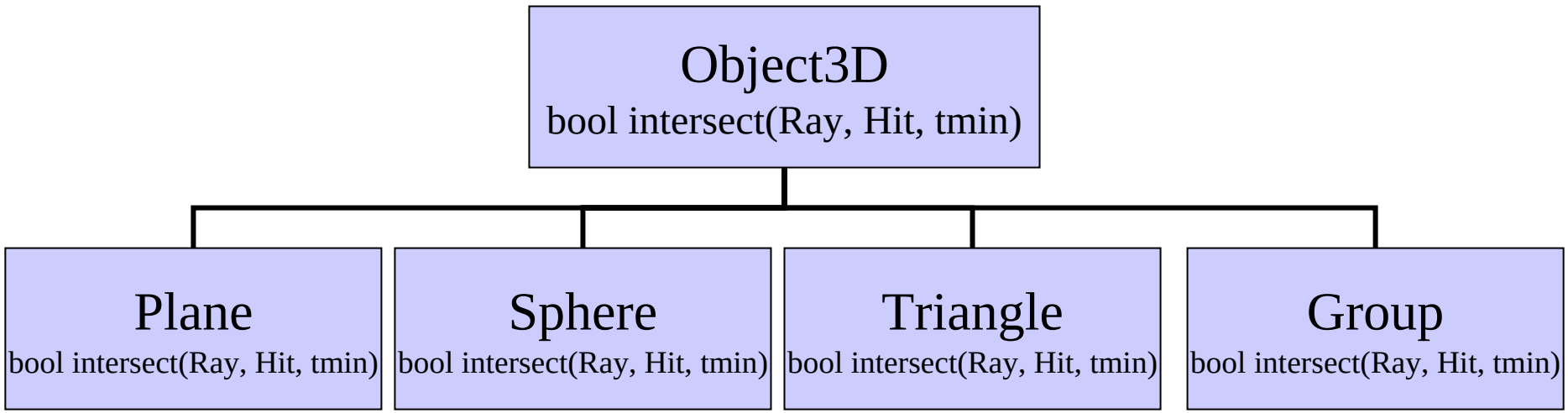


- Write a basic ray caster
  - Orthographic camera
  - Spheres
  - Display: constant color and distance
- We provide
  - Ray
  - Hit
  - Parsing
  - And linear algebra, image



Object-oriented design

- We want to be able to add primitives easily
  - Inheritance and virtual methods
- Even the scene is derived from Object3D!



Ray

```
class Ray {

public:

  // CONSTRUCTOR & DESTRUCTOR
  Ray () {}
  Ray (const Vec3f &dir, const Vec3f &orig) {
    direction = dir;
    origin = orig; }
  Ray (const Ray& r) {*this=r;}

  // ACCESSORS
  const Vec3f& getOrigin() const { return origin; }
  const Vec3f& getDirection() const { return direction; }
  Vec3f pointAtParameter(float t) const {
    return origin+direction*t; }

private:

  // REPRESENTATION
  Vec3f direction;
  Vec3f origin;

};
```

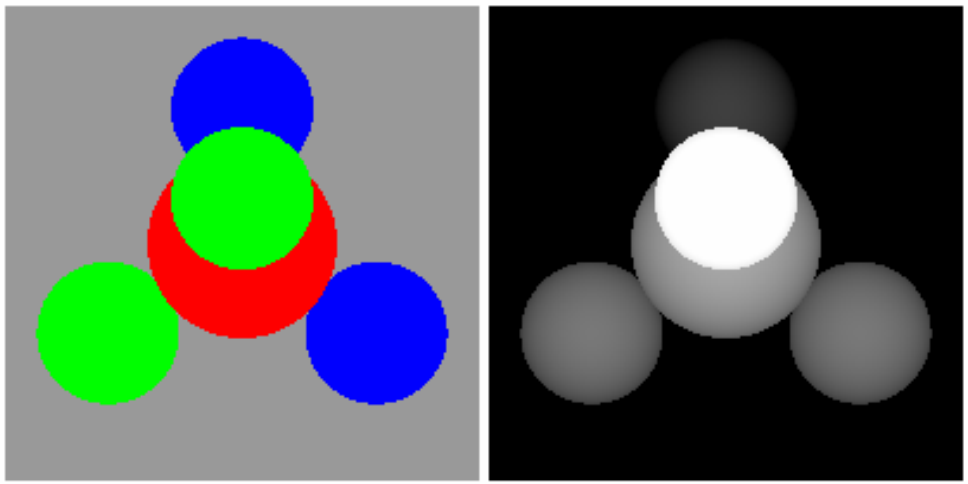
Hit

- Store intersection point & various information

```
class Hit {
public:
   // CONSTRUCTOR & DESTRUCTOR
   Hit(float _t, Vec3f c) { t = _t; color = c; }
   ~Hit() {}
   // ACCESSORS
   float getT() const { return t; }
   Vec3f getColor() const { return color; }
   // MODIFIER
   void set(float _t, Vec3f c) { t = _t; color = c; }
private:
   // REPRESENTATION
   float t;
   Vec3f color;
   //Material *material;
   //Vec3f normal;
};
```

Tasks

- Abstract Object3D
- Sphere and intersection
- Group class
- Abstract camera and derive Orthographic
- Main function

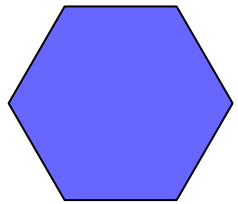


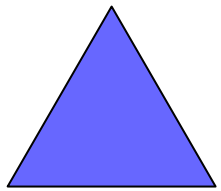
Questions?

Image removed due to copyright considerations.

Overview of today

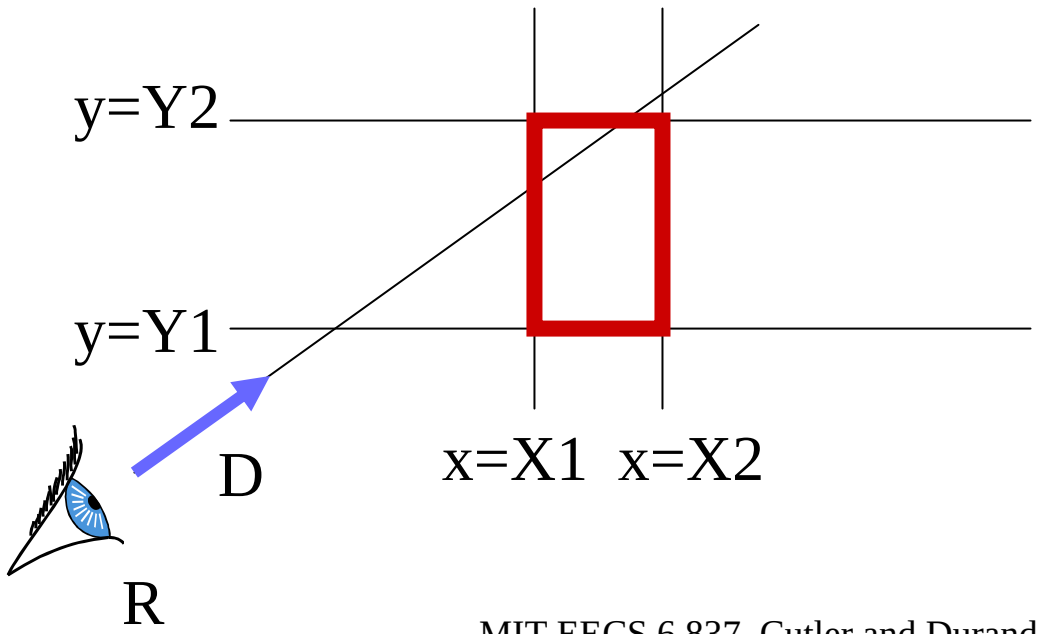
- Ray-box intersection
- Ray-polygon intersection
- Ray-triangle intersection





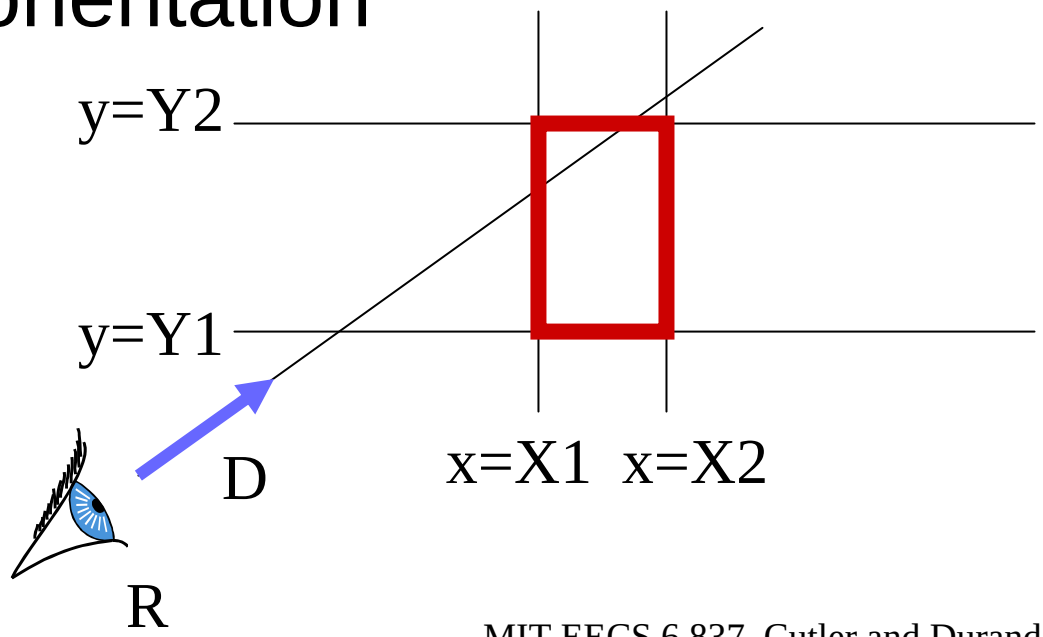
Ray-Parallelepiped Intersection

- Axis-aligned
- From (X1, Y1, Z1) to (X2, Y2, Z2)
- Ray P(t)=R+Dt



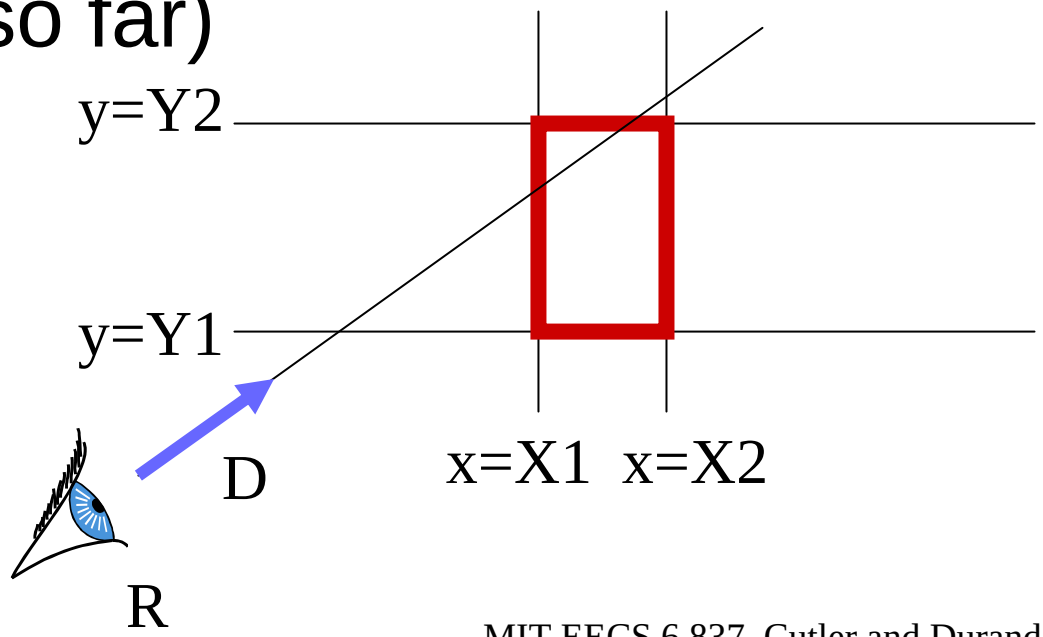
Naïve ray-box Intersection

- Use 6 plane equations
- Compute all 6 intersection
- Check that points are inside box Ax+by+Cz+D<0 with proper normal orientation



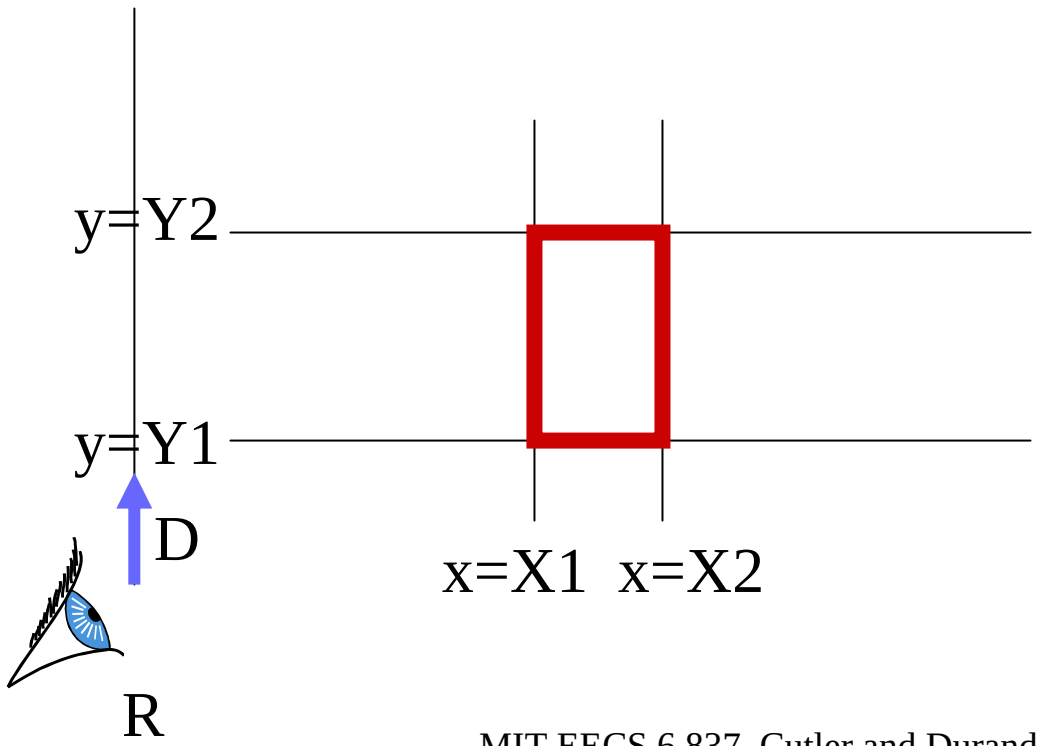
Factoring out computation

- Pairs of planes have the same normal
- Normals have only one non-0 component
- Do computations one dimension at a time
- Maintain tnear and tfar (closest and farthest so far)



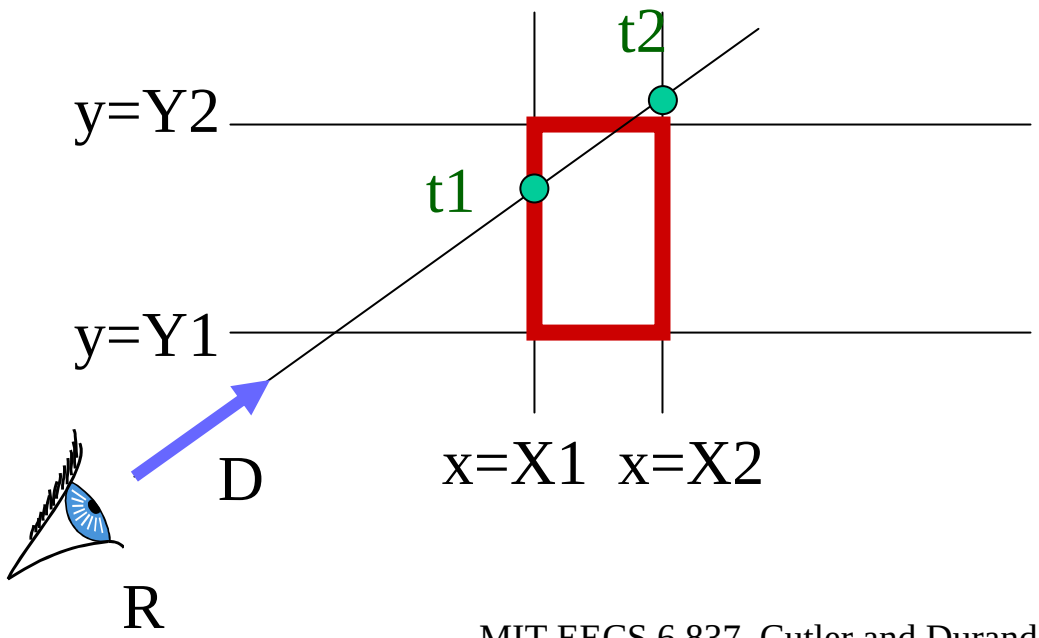
Test if parallel

- If Dx=0, then ray is parallel
  - If Rx<X1 or Rx>x2 return false



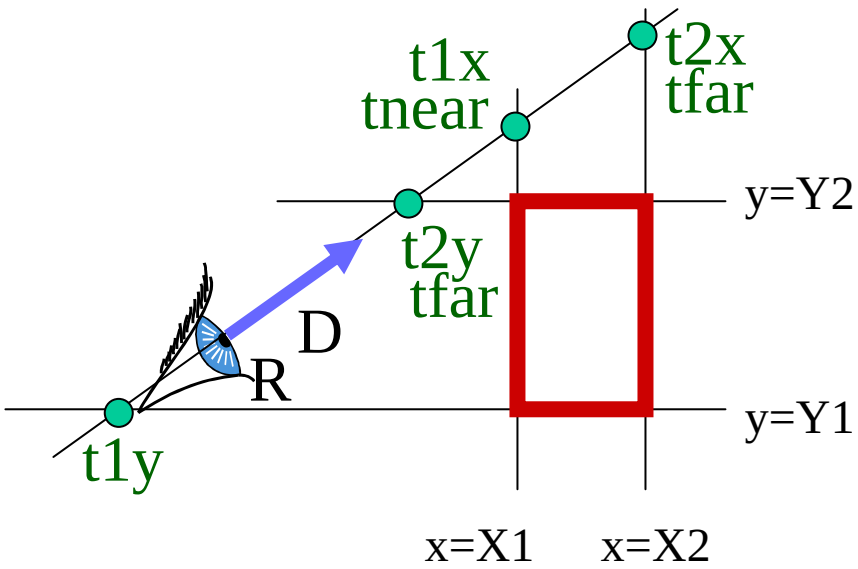
If not parallel

- Calculate intersection distance t1 and t2
  - t1=(X1-Rx)/Dx
  - t2=(X2-Rx)/Dx



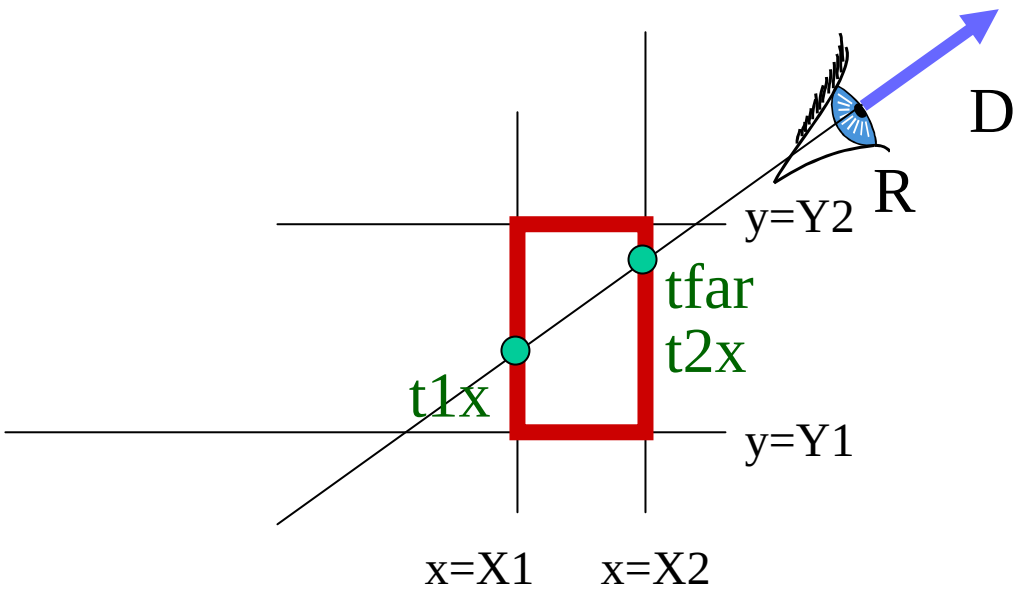
Test 1

- Maintain tnear and tfar
  - If t1>t2, swap
  - if t1>tnear, tnear =t1 if t2<tfar, tfar=t2
- If tnear>tfar, box is missed



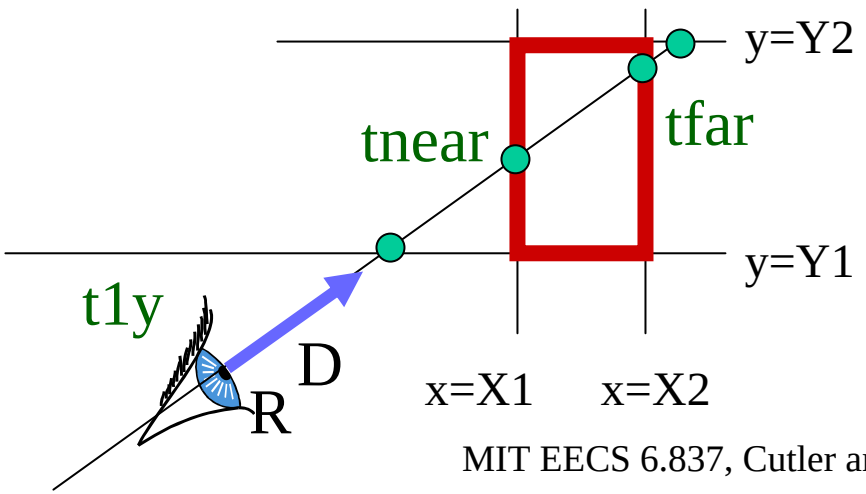
Test 2

- If tfar<0, box is behind



Algorithm recap

- Do for all 3 axis
  - Calculate intersection distance t1 and t2
  - Maintain tnear and tfar
  - If tnear>tfar, box is missed
  - If tfar<0, box is behind
- If box survived tests, report intersection at tnear



Efficiency issues

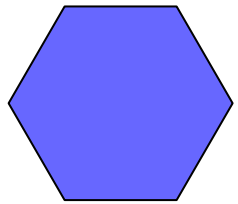
- Do for all 3 axis
  - Calculate intersection distance t1 and t2
  - Maintain tnear and tfar
  - If tnear>tfar, box is missed
  - If tfar<0, box is behind
- If box survived tests, report intersection at tnear
- 1/Dx, 1/Dy and 1/Dz can be precomputed and shared for many boxes
- Unroll the loop
  - Loops are costly (because of termination if)
  - Avoids the tnear tfar for X dimension

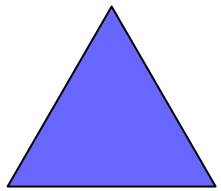
Questions?

Image removed due to copyright considerations.

Overview of today

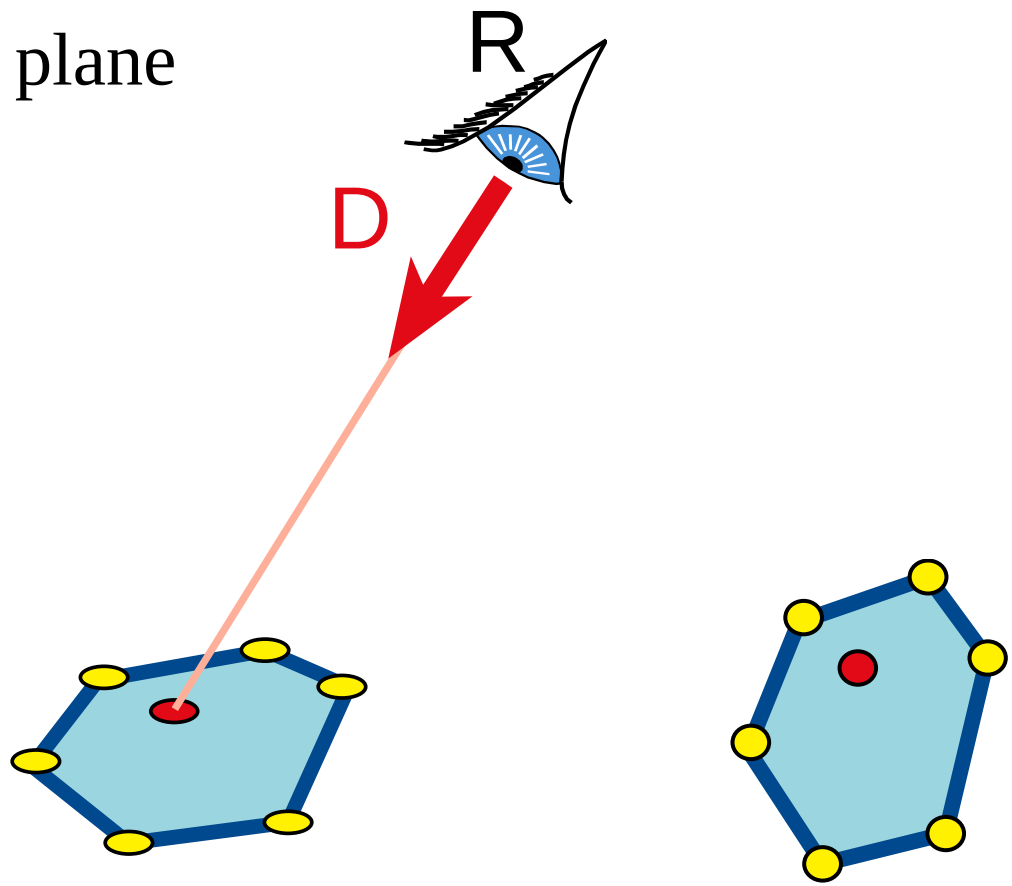
- Ray-box intersection
- Ray-polygon intersection
- Ray-triangle intersection





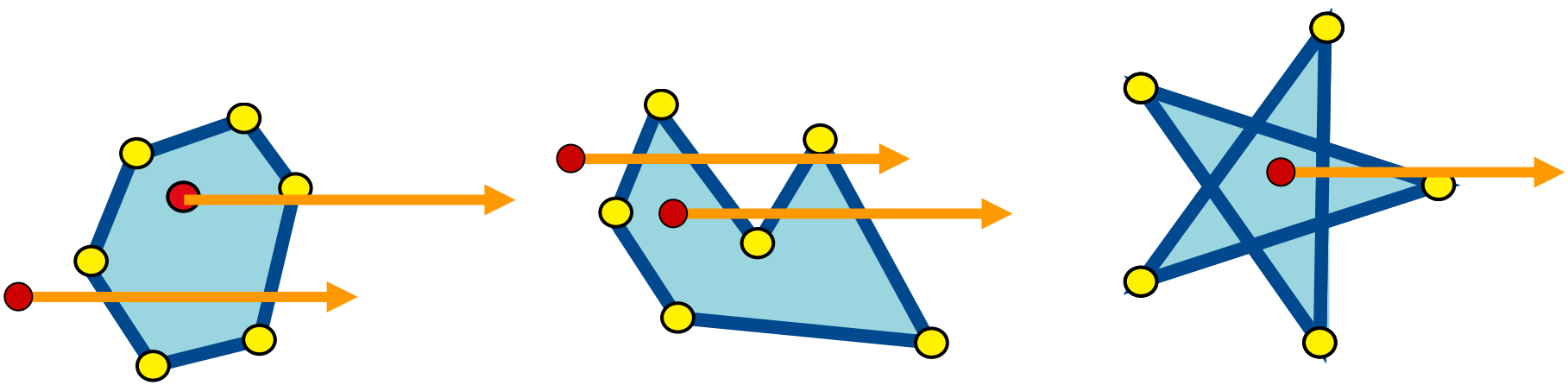
Ray-polygon intersection

- Ray-plane intersection
- Test if intersection is in the polygon
  - Solve in the 2D plane



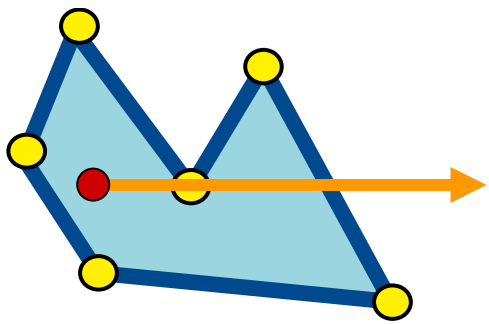
Point inside/outside polygon

- Ray intersection definition:
  - Cast a ray in any direction
    - (axis-aligned is smarter)
  - Count intersection
  - If odd number, point is inside
- Works for concave and star-shaped



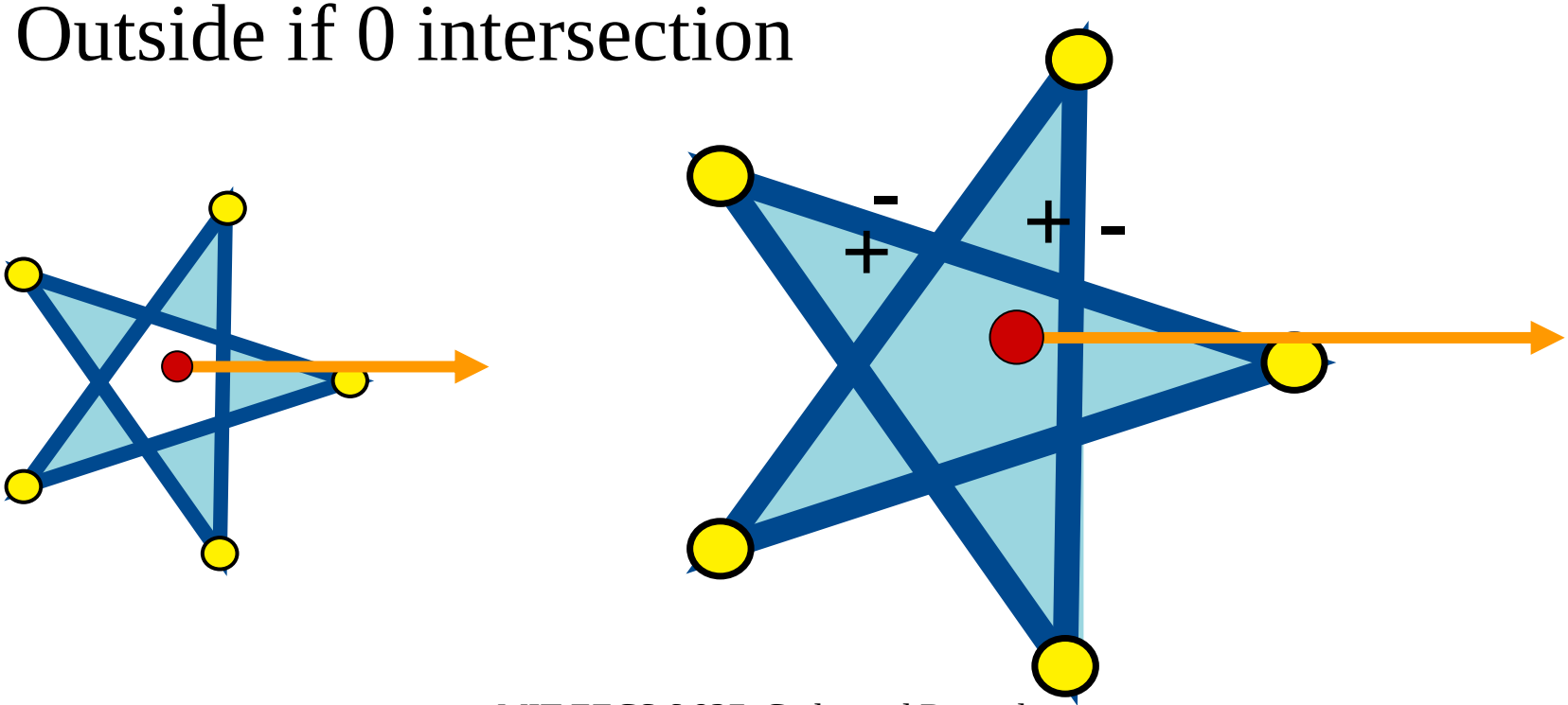
Precision issue

- What if we intersect a vertex?
  - We might wrongly count an intersection for each adjacent edge
- Decide that the vertex is always above the ray



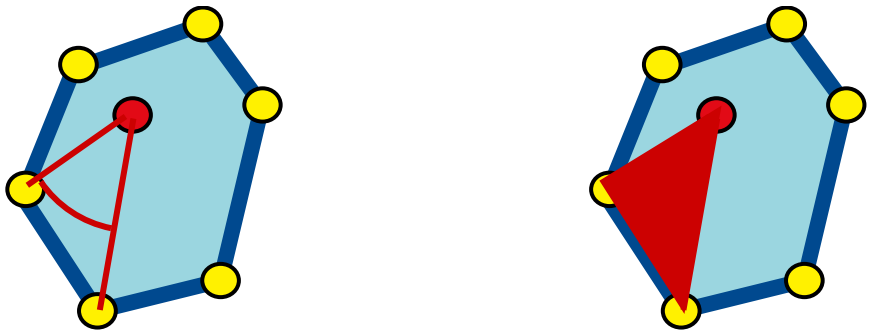
Winding number

- To solve problem with star pentagon
- Oriented edges
- Signed number of intersection
- Outside if 0 intersection



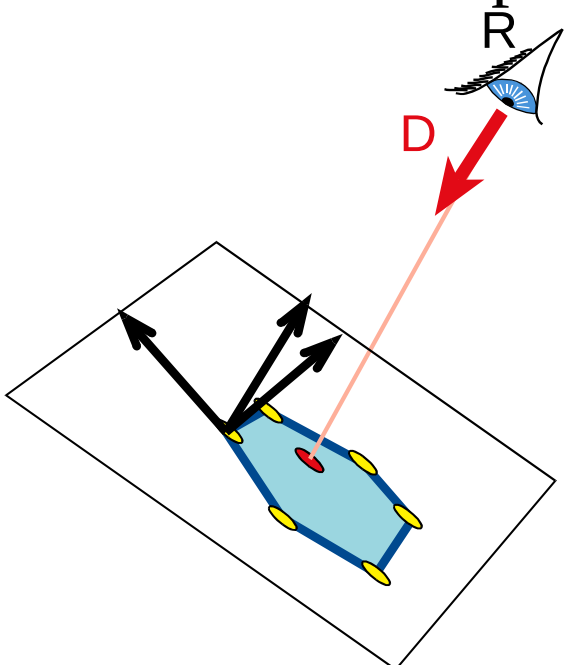
Alternative definitions

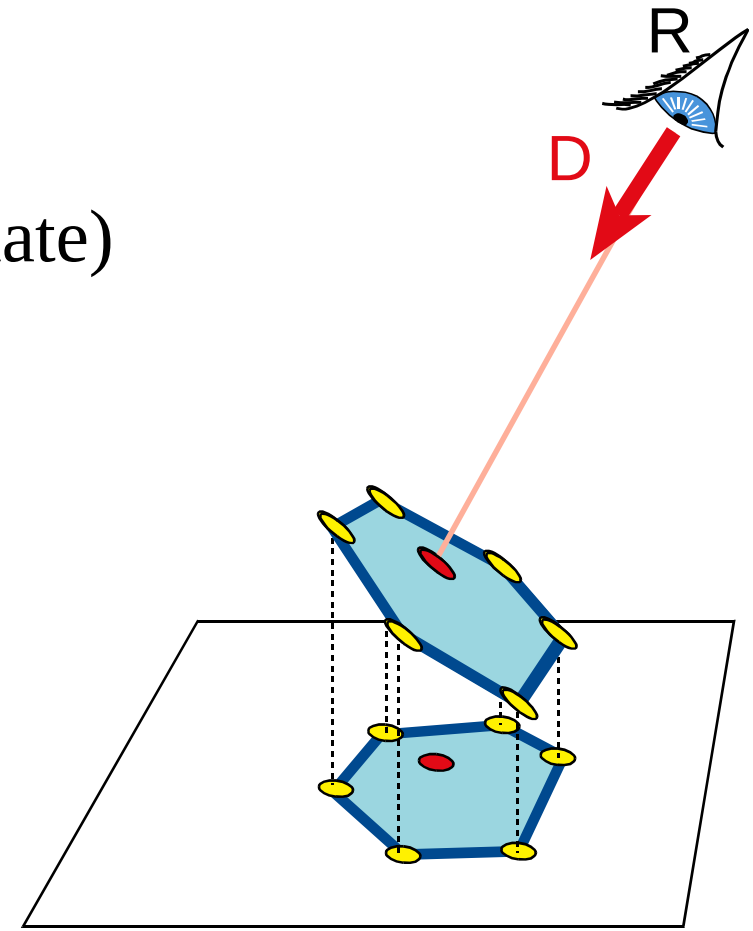
- Sum of the signed angles from point to vertices
  - 360 if inside, 0 if outside
- Sum of the signed areas of point-edge triangles
  - Area of polygon if inside, 0 if outside



How do we project into 2D?

- Along normal
  - Costly
- Along axis
  - Smarter (just drop 1 coordinate)
  - Beware of parallel plane



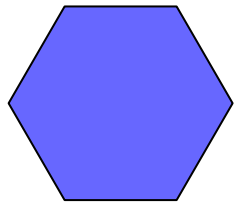


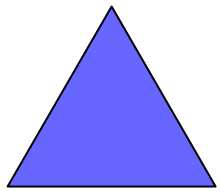
Questions?

Image removed due to copyright considerations.

Overview of today

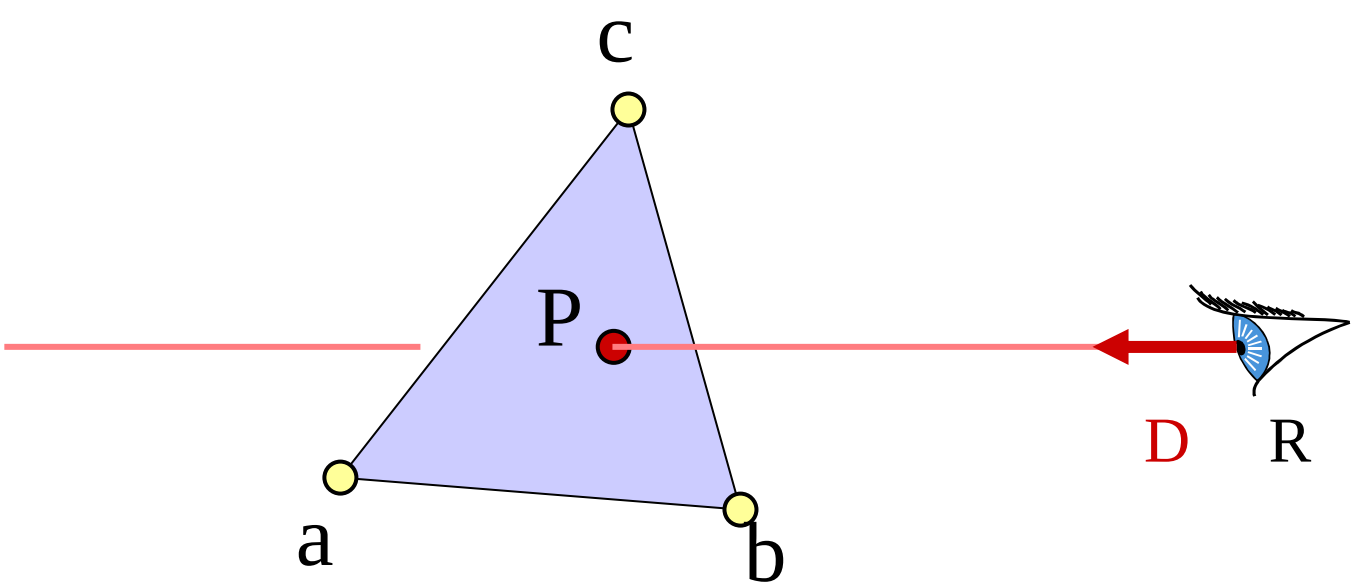
- Ray-box intersection
- Ray-polygon intersection
- Ray-triangle intersection





Ray triangle intersection

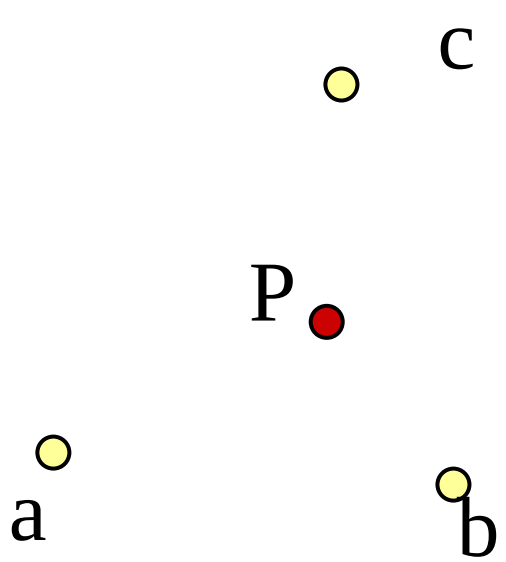
- Use ray-polygon
- Or try to be smarter
  - Use barycentric coordinates



Barycentric definition of a plane

- $P(\alpha, \beta, \gamma)=\alpha a+\beta b+\gamma c$ with $\alpha + \beta + \gamma =1$
- Is it explicit or implicit?

[Möbius, 1827]



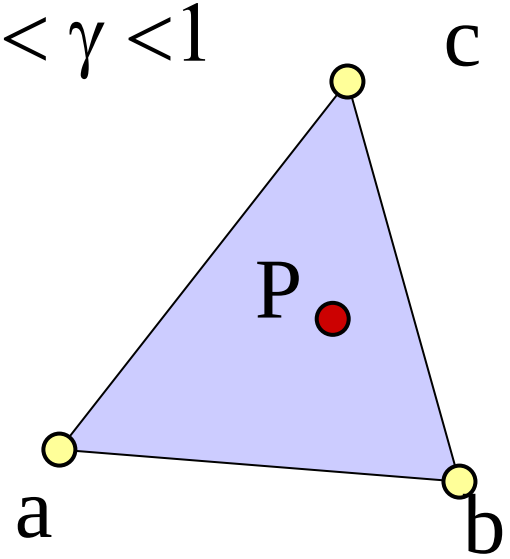
Barycentric definition of a triangle

- $P(\alpha, \beta, \gamma)=\alpha a+\beta b+\gamma c$
  with $\alpha + \beta + \gamma =1$

$0< \alpha <1$

$0< \beta <1$

$0< \gamma <1$

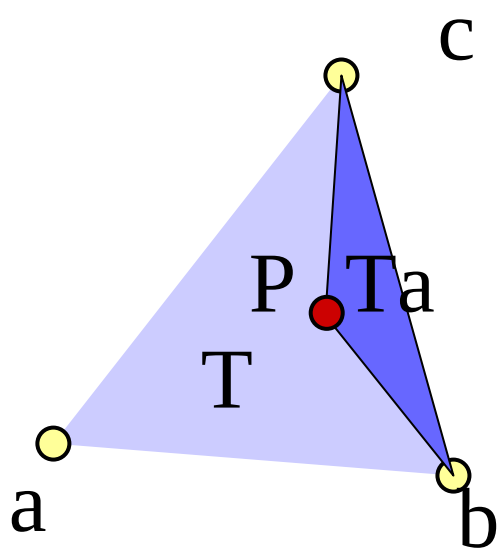


Given P, how can we compute α, β, γ ?

- Compute the areas of the opposite subtriangle
  - Ratio with complete area

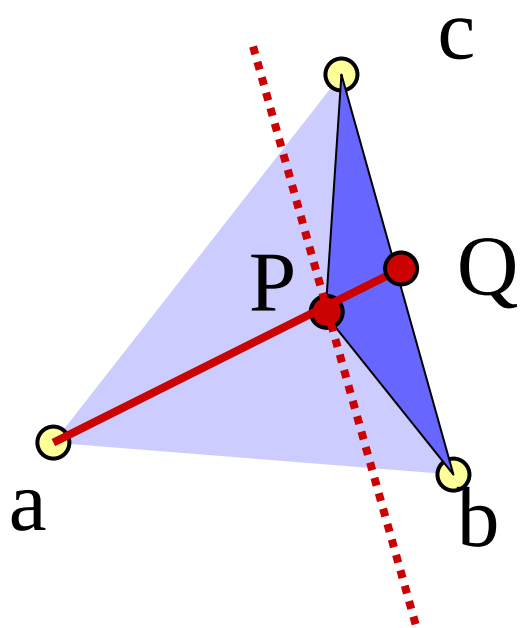
$$\alpha = A_a/A, \qquad \beta = A_b/A \qquad \gamma = A_c/A$$

Use signed areas for points outside the triangle



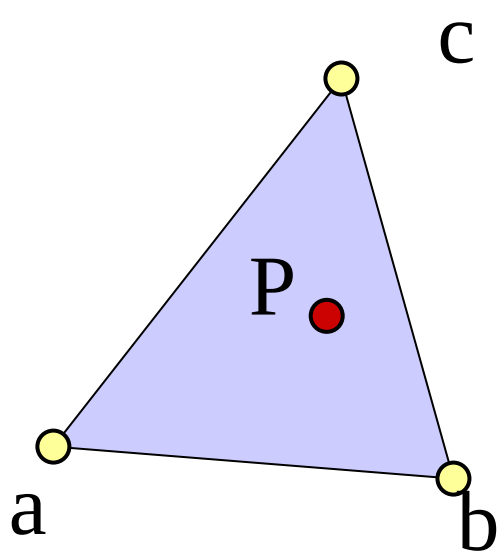
Intuition behind area formula

- P is barycenter of a and Q
- A is the interpolation coefficient on aQ
- All points on line parallel to bc have the same α
- All such Ta triangles have same height/area



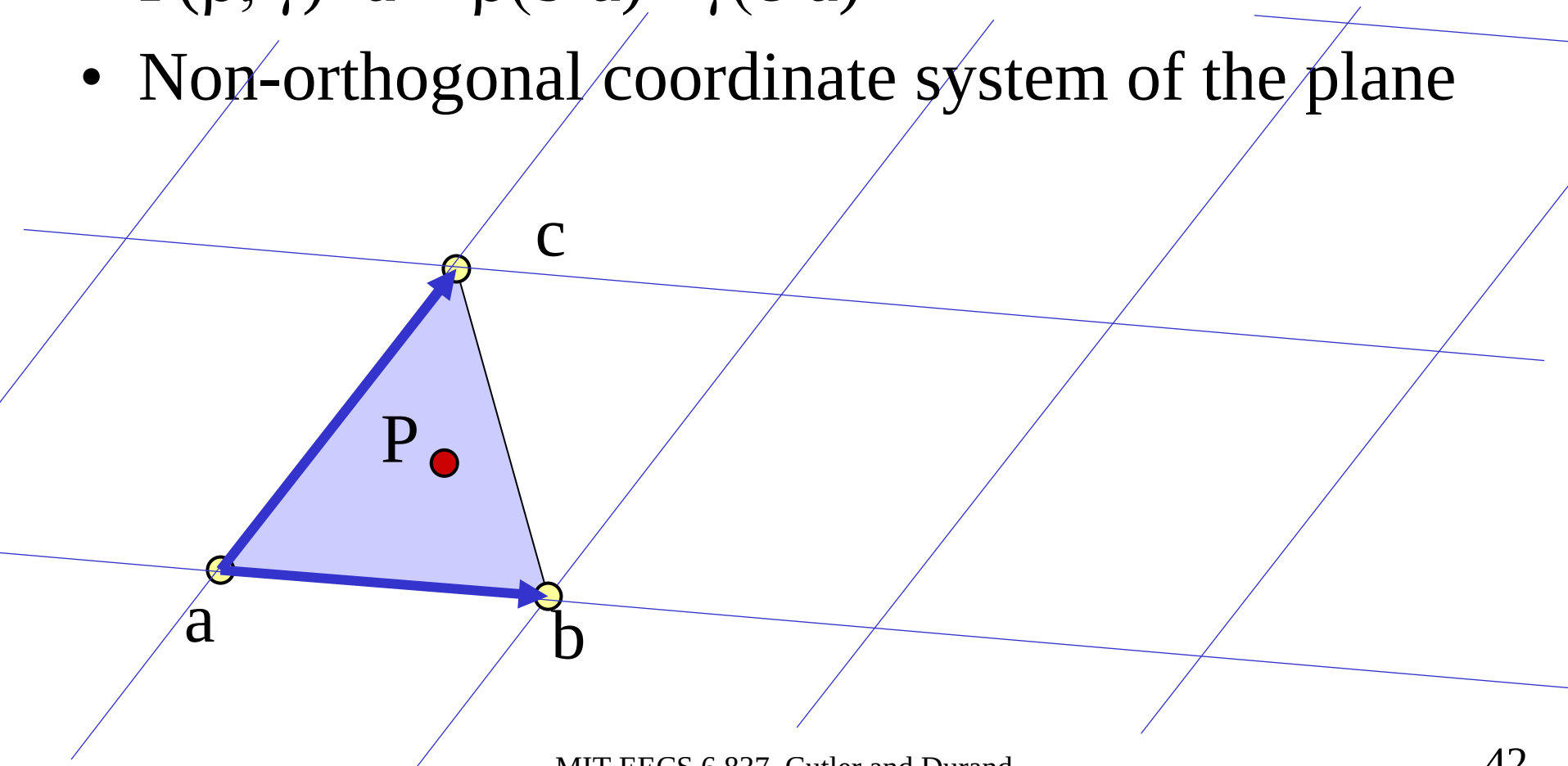
Simplify

- Since $\alpha + \beta + \gamma = 1$
  we can write $\alpha = 1 - \beta - \gamma$
- $P(\beta, \gamma) = (1-\beta-\gamma)\, a + \beta b + \gamma c$



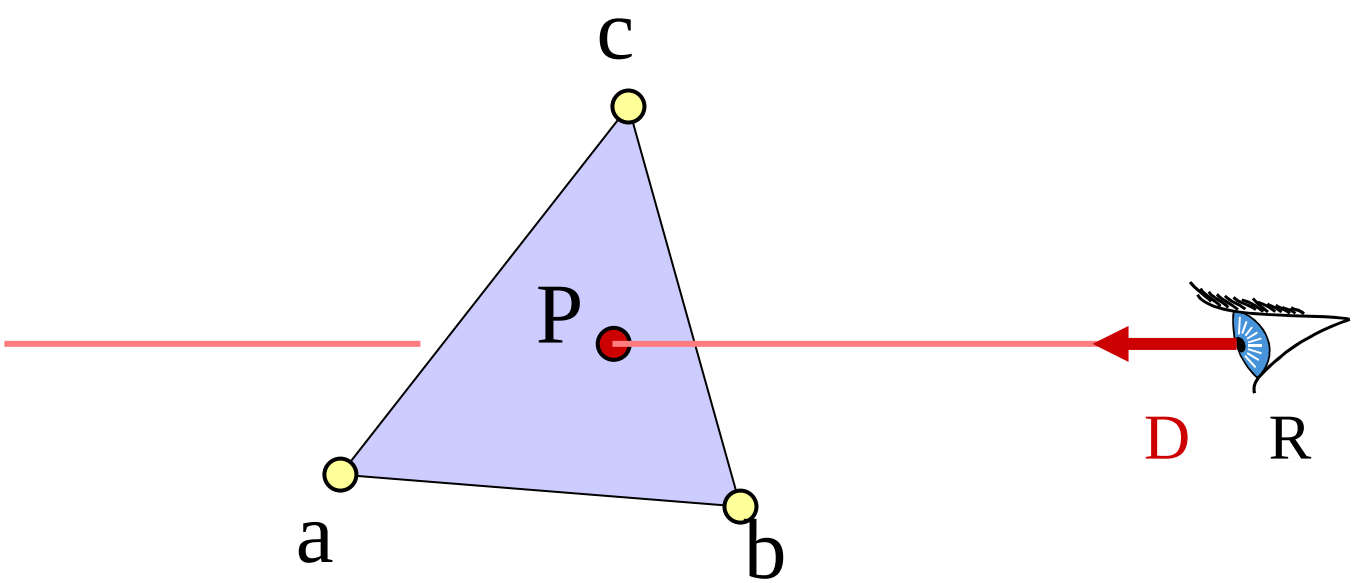
Simplify

- $P(\beta, \gamma)=(1-\beta-\gamma)\, a + \beta b + \gamma c$
- $P(\beta, \gamma)=a + \beta(b-a) + \gamma(c-a)$
- Non-orthogonal coordinate system of the plane



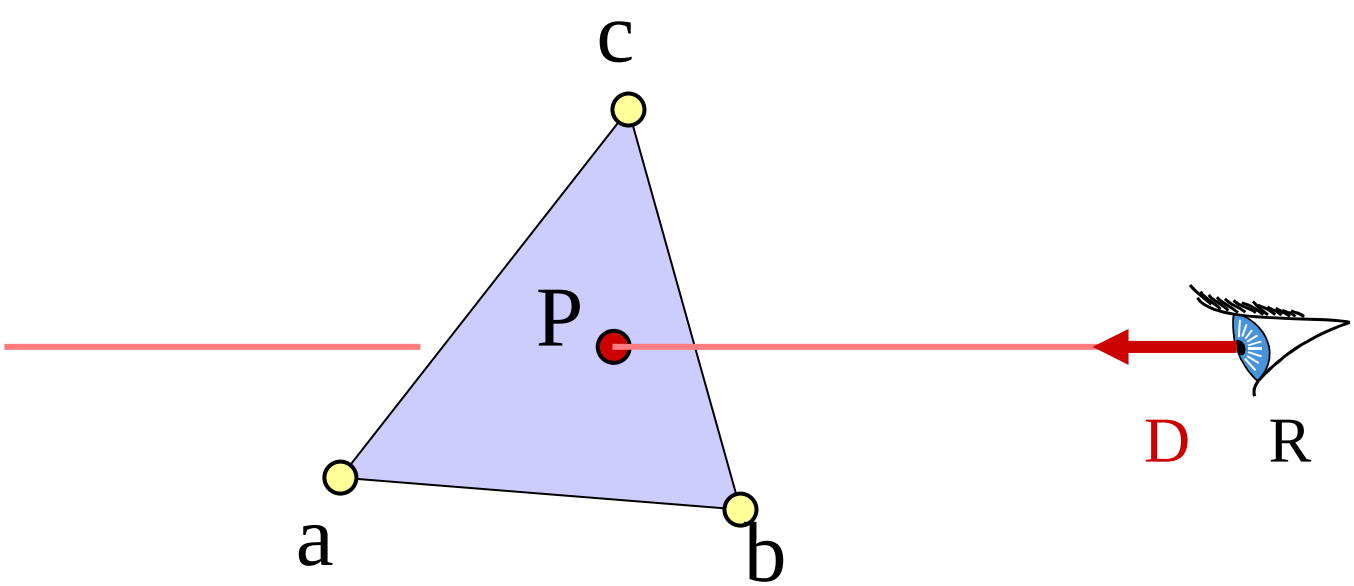
How do we use it for intersection?

- Insert ray equation into barycentric expression of triangle
- $P(t)= a+\beta (b-a)+\gamma (c-a)$
- Intersection if $\beta+\gamma<1$; $0<\beta$ and $0<\gamma$



Intersection

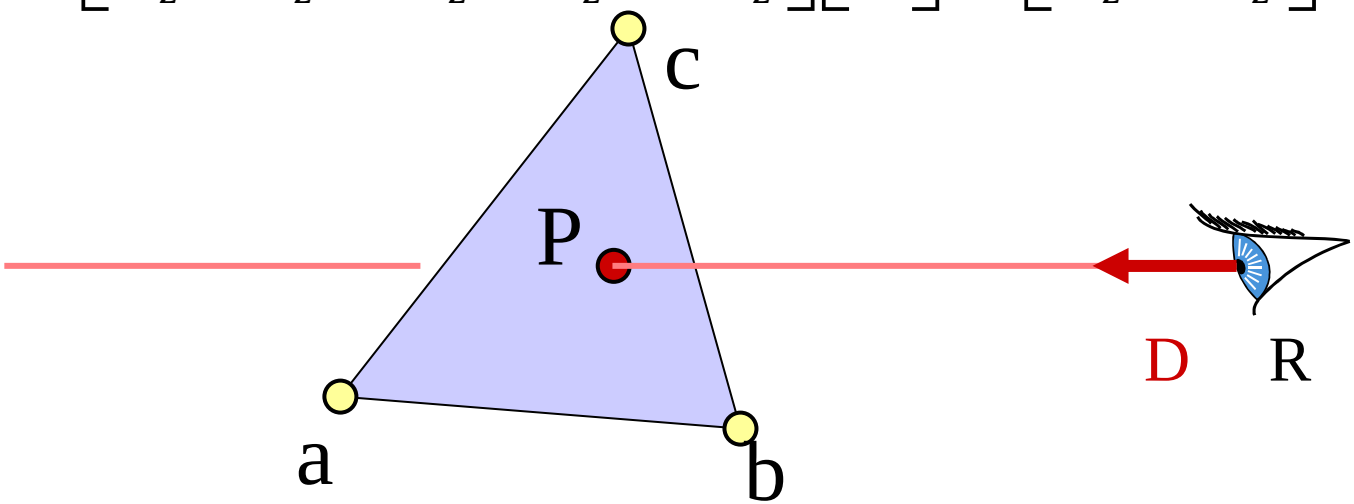
- $R_x+tD_x= a_x+\beta (b_x-a_x)+\gamma (c_x-a_x)$
- $R_y+tD_y= a_y+\beta (b_y-a_y)+\gamma (c_y-a_y)$
- $R_z+tD_z= a_z+\beta (b_z-a_z)+\gamma (c_z-a_z)$



Matrix form

- $R_x+tD_x= a_x+\beta (b_x-a_x)+\gamma (c_x-a_x)$
- $R_y+tD_y= a_y+\beta (b_y-a_y)+\gamma (c_y-a_y)$
- $R_z+tD_z= a_z+\beta (b_z-a_z)+\gamma (c_z-a_z)$

$$\begin{bmatrix} a_x-b_x & a_x-c_x & D_x \\ a_y-b_y & a_y-c_y & D_y \\ a_z-b_z & a_z-c_z & D_z \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x-R_x \\ a_y-R_y \\ a_z-R_z \end{bmatrix}$$

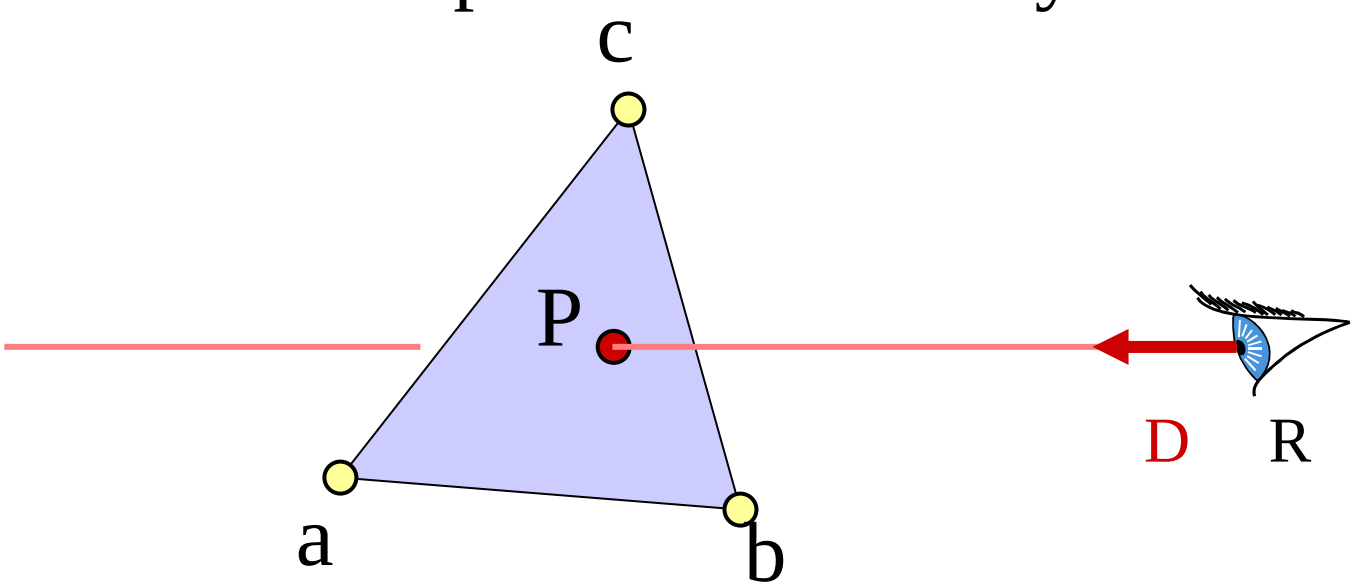


Cramer's rule

- | | denotes the determinant

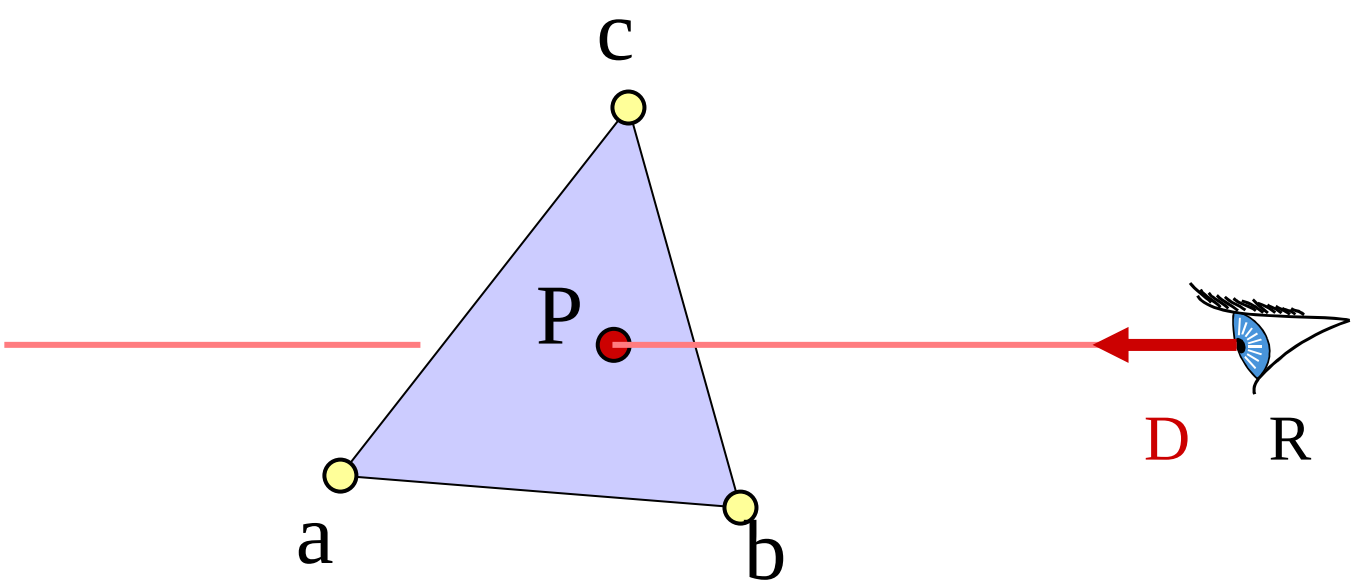
$$\beta = \frac{\begin{vmatrix} a_x - R_x & a_x - c_x & D_x \\ a_y - R_y & a_y - c_y & D_y \\ a_z - R_z & a_z - c_z & D_z \end{vmatrix}}{|A|} \qquad \gamma = \frac{\begin{vmatrix} a_x - b_x & a_x - R_x & D_x \\ a_y - b_y & a_y - R_y & D_y \\ a_z - b_z & a_z - R_z & D_z \end{vmatrix}}{|A|} \qquad t = \frac{\begin{vmatrix} a_x - b_x & a_x - c_x & a_x - R_x \\ a_y - b_y & a_y - c_y & a_y - R_y \\ a_z - b_z & a_z - c_z & a_z - R_z \end{vmatrix}}{|A|}$$

- Can be copied mechanically in the code



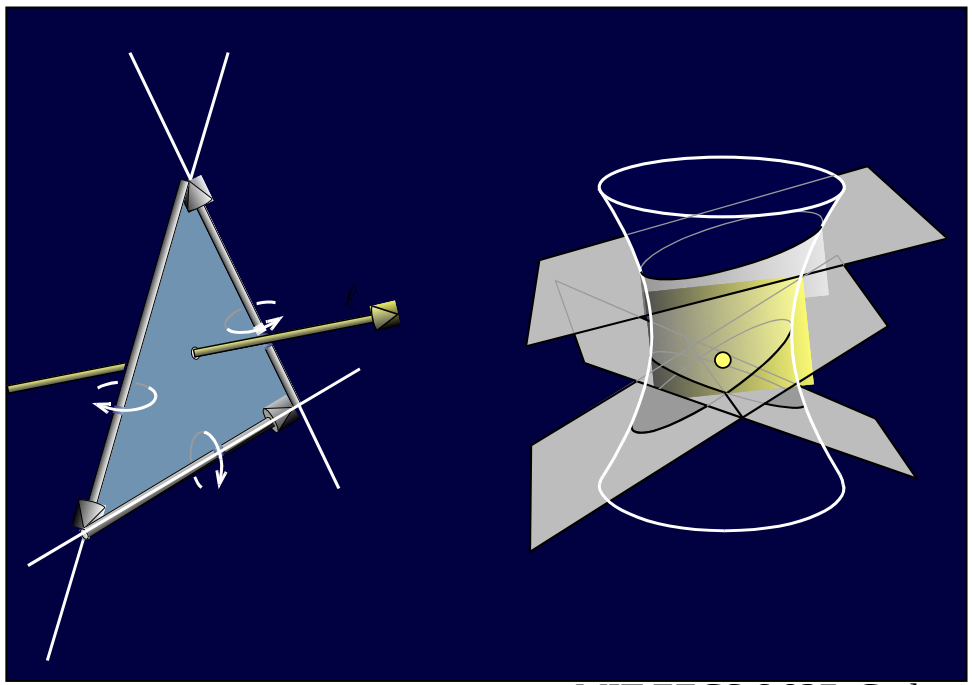
Advantage

- Efficient
- Store no plane equation
- Get the barycentric coordinates for free
  - Useful for interpolation, texture mapping



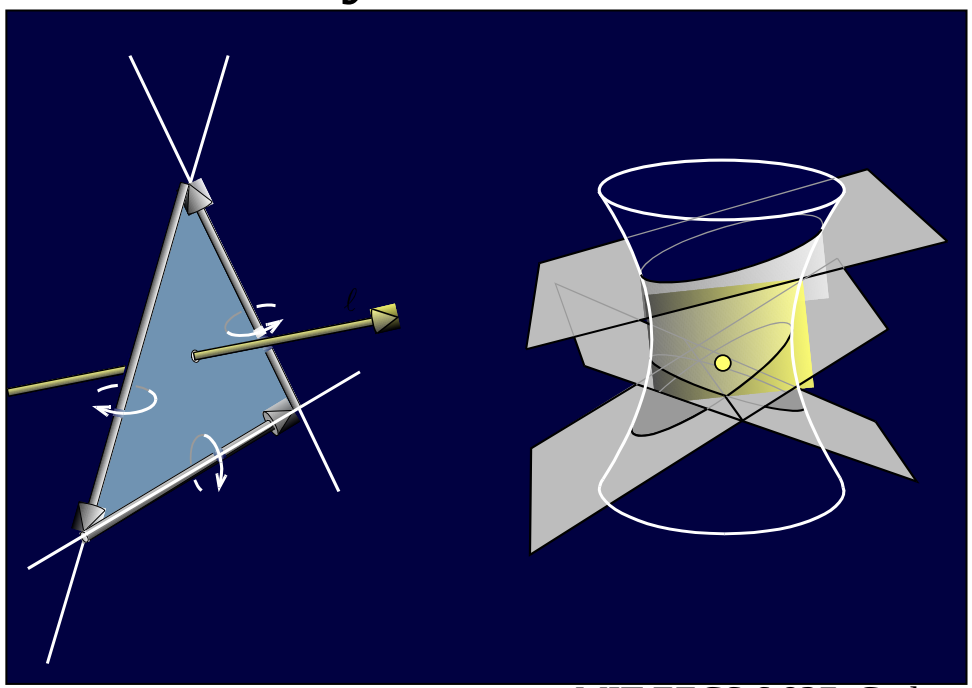
Plucker computation

- Plucker space:
  6 or 5 dimensional space describing 3D lines
- A line is a point in Plucker space



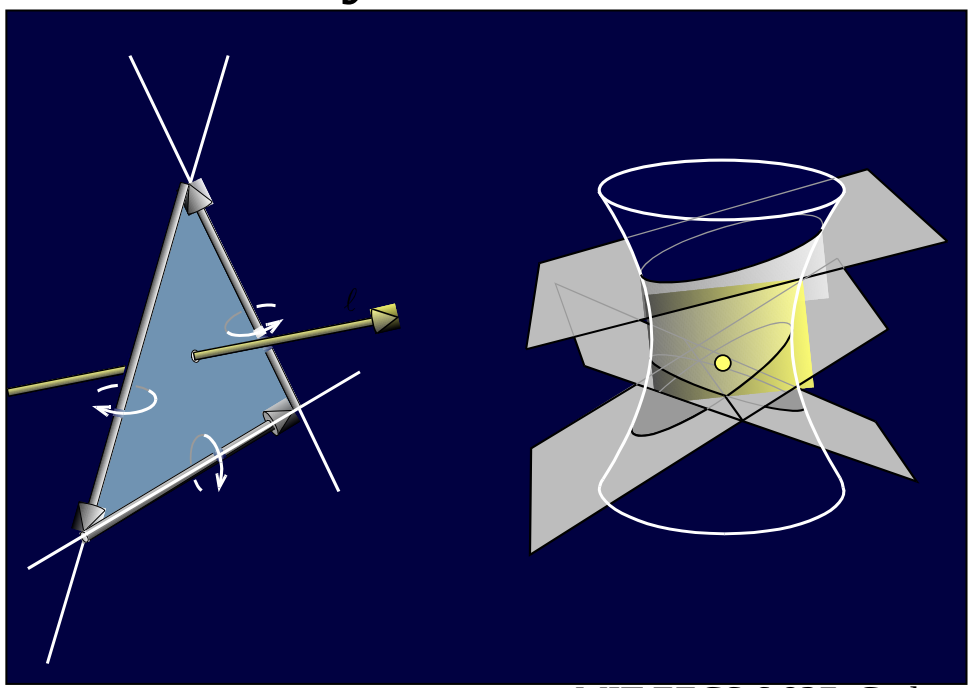
Plucker computation

- The rays intersecting a line are a hyperplane
- A triangle defines 3 hyperplanes
- The polytope defined by the hyperplanes is the set of rays that intersect the triangle



Plucker computation

- The rays intersecting a line are a hyperplane
- A triangle defines 3 hyperplanes
- The polytope defined by the hyperplanes is the set of rays that intersect the triangle



- Ray-triangle intersection becomes a polytope inclusion
- Couple of additional issues

Next week: Transformations

- Permits 3D IFS ;-)

Image removed due to copyright considerations.