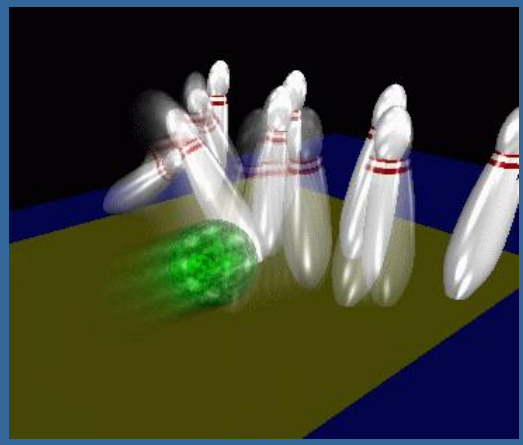
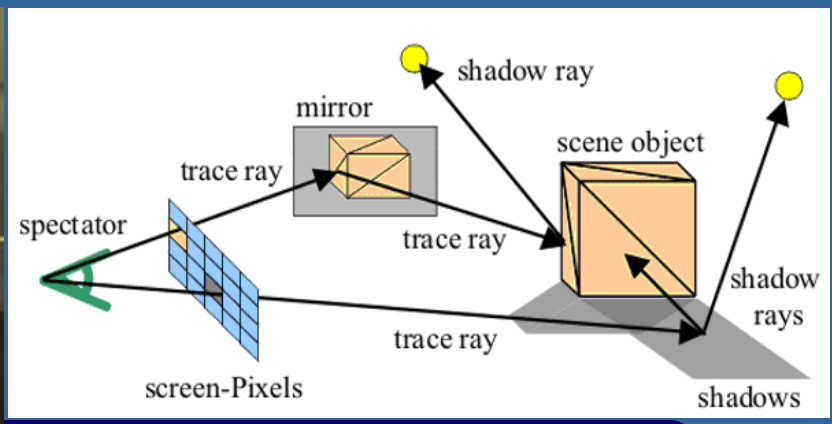
Intersection Testing
Chapter 16





Department of Computer Engineering

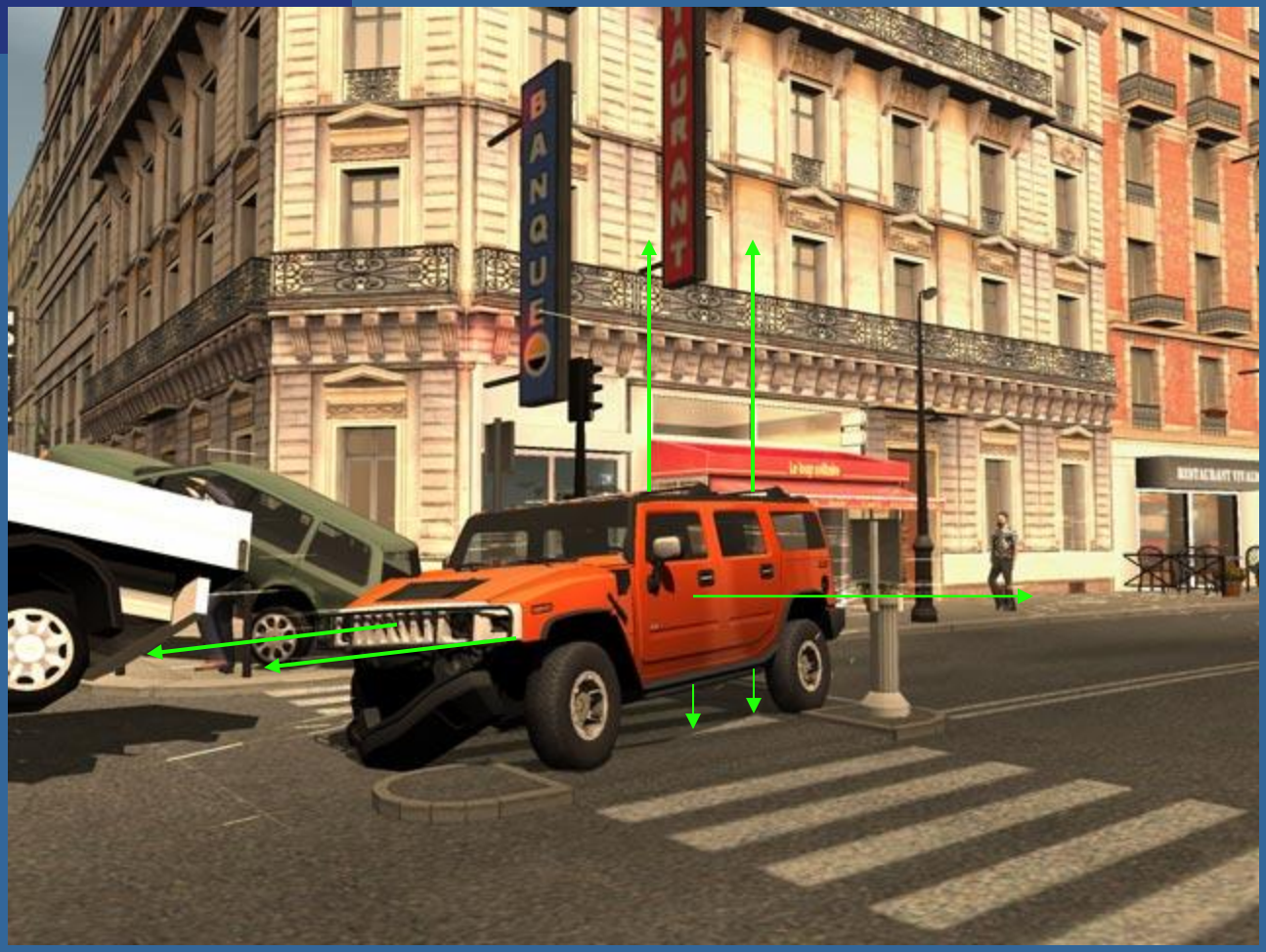
Chalmers University of Technology

What for?

- A tool needed for the graphics people all the time…
- Very important components:
  - Need to make them fast!

- Finding if (and where) a ray hits an object
  - Picking
  - Ray tracing and global illumination
- For speed-up techniques
- Collision detection (treated in a later lecture)
  - E.g., NVIDIA PhysX used by: Unreal Engine, Unity, …
    - Games e.g.: *The Witcher 3: Wild Hunt*, *Warframe*, *Killing Floor 2*, *Fallout 4*, *Batman: Arkham Knight*, *Planetside 2,* and *Borderlands 2*…

Example



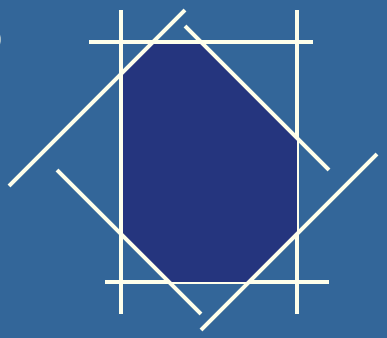
Midtown Madness 3, DICE

Some basic geometrical primitives

- Ray:
- Sphere:
- Box
  - Axis-aligned (AABB)
  - Oriented (OBB)
- *k*-DOP





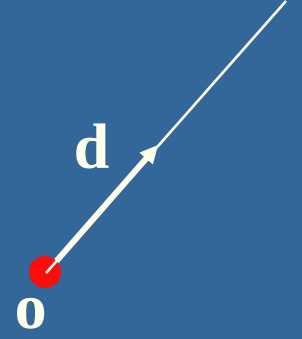


Some different techniques

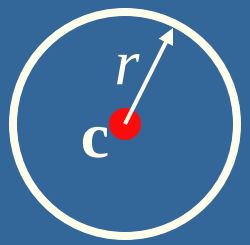
- Analytical
  - ”Solve an equation system”
  - E.g., ray/sphere, ray/plane, ray/triangle,
- Geometrical
  - ”Uses spatial reasoning”. Method operates on geometric entities like points, vectors, edges, surfaces, volumes…
  - Ray/box, ray/polygon
  - Separating axis theorem (SAT)
  - The Gilbert-Johnson-Keerthi (GJK) algorithm
    - Not treated in this course
- Dynamic tests (to find time of collision)

- Given these, one can derive many tests quite easily
  - However, often tricks are needed to make them fast

Analytical: Ray/sphere test



- Sphere center: **c**, and radius r
- Ray function: $\mathbf{r}(t)=\mathbf{o}+t\mathbf{d}$
- Sphere equation: $\|\mathbf{p}-\mathbf{c}\|=r$
- Replace **p** by $\mathbf{r}(t)$, and square it:



$$\|\mathbf{x}\| = \sqrt{(x^2 + y^2 + z^2)}$$

$$\mathbf{a} \cdot \mathbf{b} = (a_x b_x + a_y b_y + a_z b_z)$$

$$(\mathbf{r}(t)-\mathbf{c})\cdot(\mathbf{r}(t)-\mathbf{c})-r^2=0$$

$$(\mathbf{o}+t\mathbf{d}-\mathbf{c})\cdot(\mathbf{o}+t\mathbf{d}-\mathbf{c})-r^2=0$$

x•x is not $\mathbf{x}^2$

$$(t\mathbf{d}+(\mathbf{o}-\mathbf{c}))\times(t\mathbf{d}+(\mathbf{o}-\mathbf{c}))-r^2=0$$

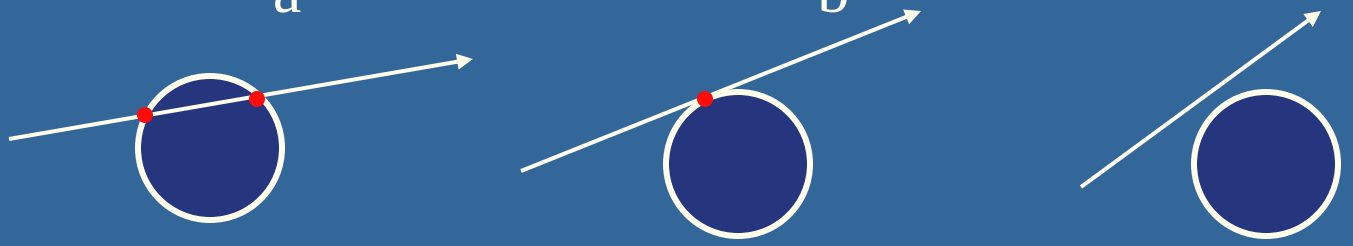
$$(\mathbf{d}\cdot\mathbf{d})t^2+2((\mathbf{o}-\mathbf{c})\cdot\mathbf{d})t+(\mathbf{o}-\mathbf{c})\cdot(\mathbf{o}-\mathbf{c})-r^2=0$$

$$t^2+2((\mathbf{o}-\mathbf{c})\cdot\mathbf{d})t+(\mathbf{o}-\mathbf{c})\cdot(\mathbf{o}-\mathbf{c})-r^2=0 \qquad \|\mathbf{d}\|=1$$

Analytical, continued

$$t^2 + 2((\mathbf{o}-\mathbf{c})\cdot\mathbf{d})t + (\mathbf{o}-\mathbf{c})\cdot(\mathbf{o}-\mathbf{c}) - r^2 = 0 \qquad t = \frac{-a}{2} + \sqrt{\left(\frac{a}{2}\right)^2 - b}$$

a b



- Check for quick rejection tests

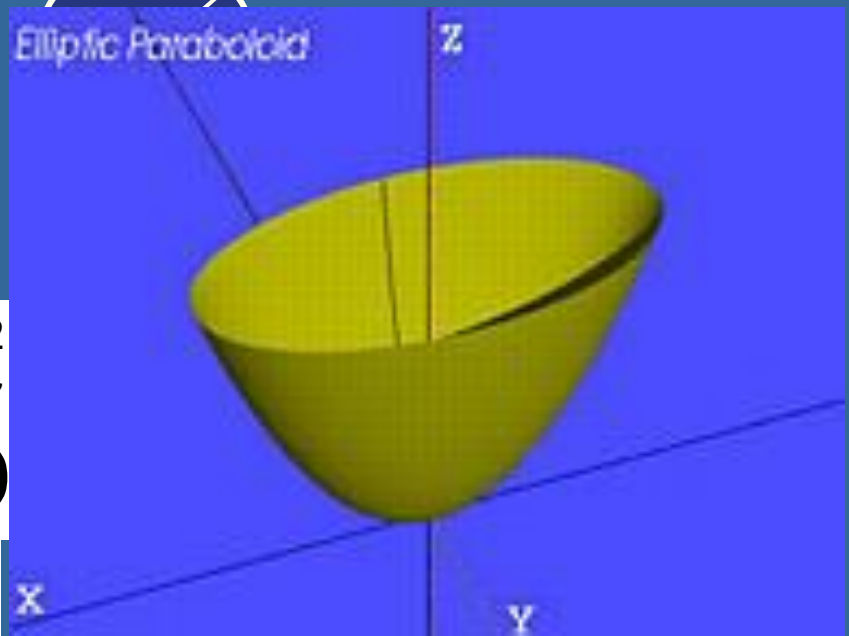
$$(\mathbf{o}-\mathbf{c})\cdot\mathbf{d} > 0 \ ?$$

$$(\mathbf{o}-\mathbf{c})\cdot(\mathbf{o}-\mathbf{c}) - r^2 < 0 \ ?$$

- to save unnessesary
- Other shapes: $p_x^2 + p_y^2$

$$(p_x/a)^2 + (p_y/b)^2 + (p_z/c)$$

$$(p_x/a)^2 + (p_y/b)^2 - p_z = 0$$



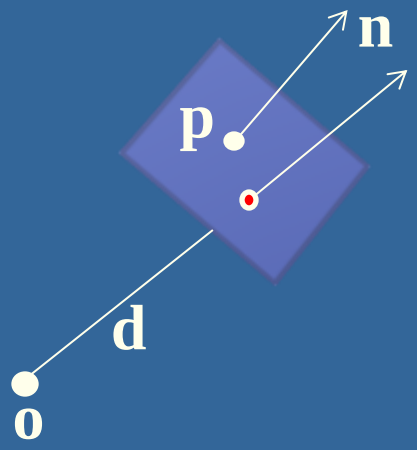
Ray/Plane Intersections

- Ray function: r(t)=**o**+t**d**
- Plane equation: **n**•**x** + d = 0;
- Replace **x** by r(t):

  n•(**o**+t**d**) + d = 0

  n•**o**+t(**n**•**d**) + d = 0

  t = (–d –**n**•**o**) / (**n**•**d**)




```
Vec3f rayPlaneIntersect(vec3f o,dir, n, d)
{
        float t=(-d-n.dot(o)) / (n.dot(dir));
        return o + dir*t;
}
```

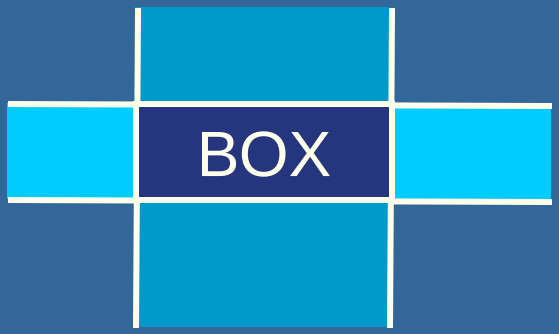
Analytical tests – quick guide

- If both objects are described as functions, set them equal and solve
  - See ray/triangle-test: tri(u,v) = r(t)
- If function and equation, replace:
  - See ray/sphere and ray/plane
    - E.g.,
      - plane eq: **n**•**x** + d = 0; // **x** is the variable
      - Ray function: r(t)=**o**+t**d** // t is the parameter (variable)
      - Replace **x** with r(t)
- If two equations, find their common solutions.

Geometrical: Ray/Box Intersection

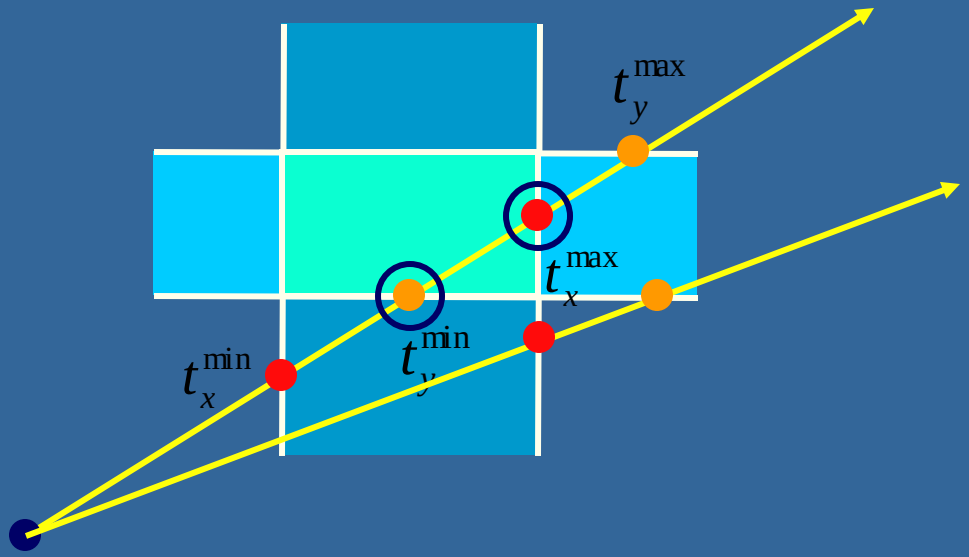
- Boxes and spheres often used as bounding volumes
- A slab is the volume between two parallell planes:

- A box is the logical intersection of three slabs (2 in 2D):



Geometrical: Ray/Box Intersection (2)

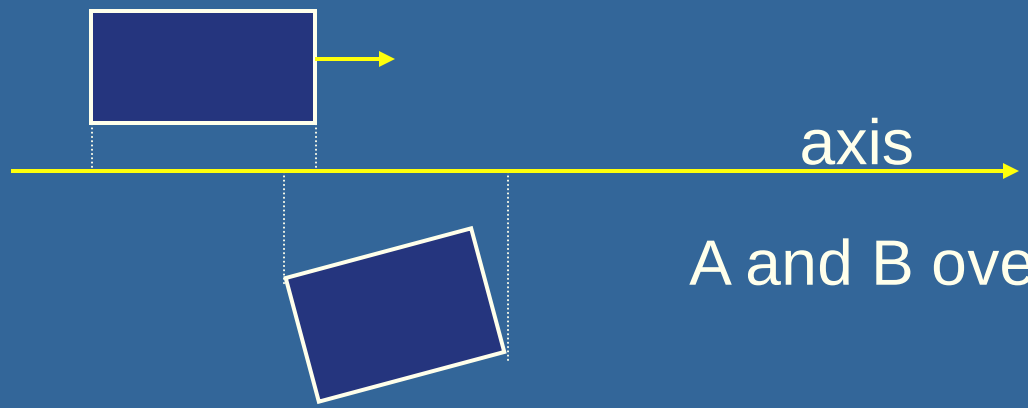
- Intersect the 2 planes of each slab with the ray



- Keep max of t^{min} and min of t^{max},
  - i.e., $t^{min} = max(t_x^{min}, t_y^{min}, t_z^{min})$, $t^{max} = min(t_x^{max}, t_y^{max}, t_z^{max})$
- If $t^{min} < t^{max}$ then we got an intersection
- Special case when ray parallell to slab

Separating Axis Theorem (SAT) Page 947 in book

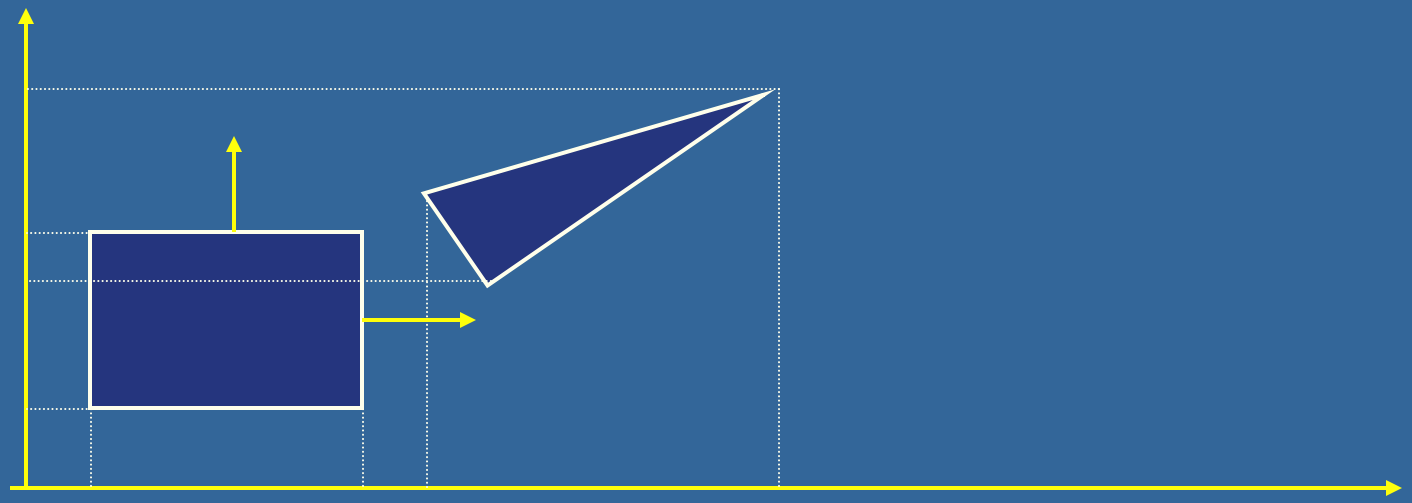
- Two convex polyhedrons, A and B, are disjoint if any of the following axes separate the objects' projections:
  - A face normal of A
  - A face normal of B
  - Any $edge_A \times edge_B$ ($\times$ is crossproduct)



A and B overlaps on this axis

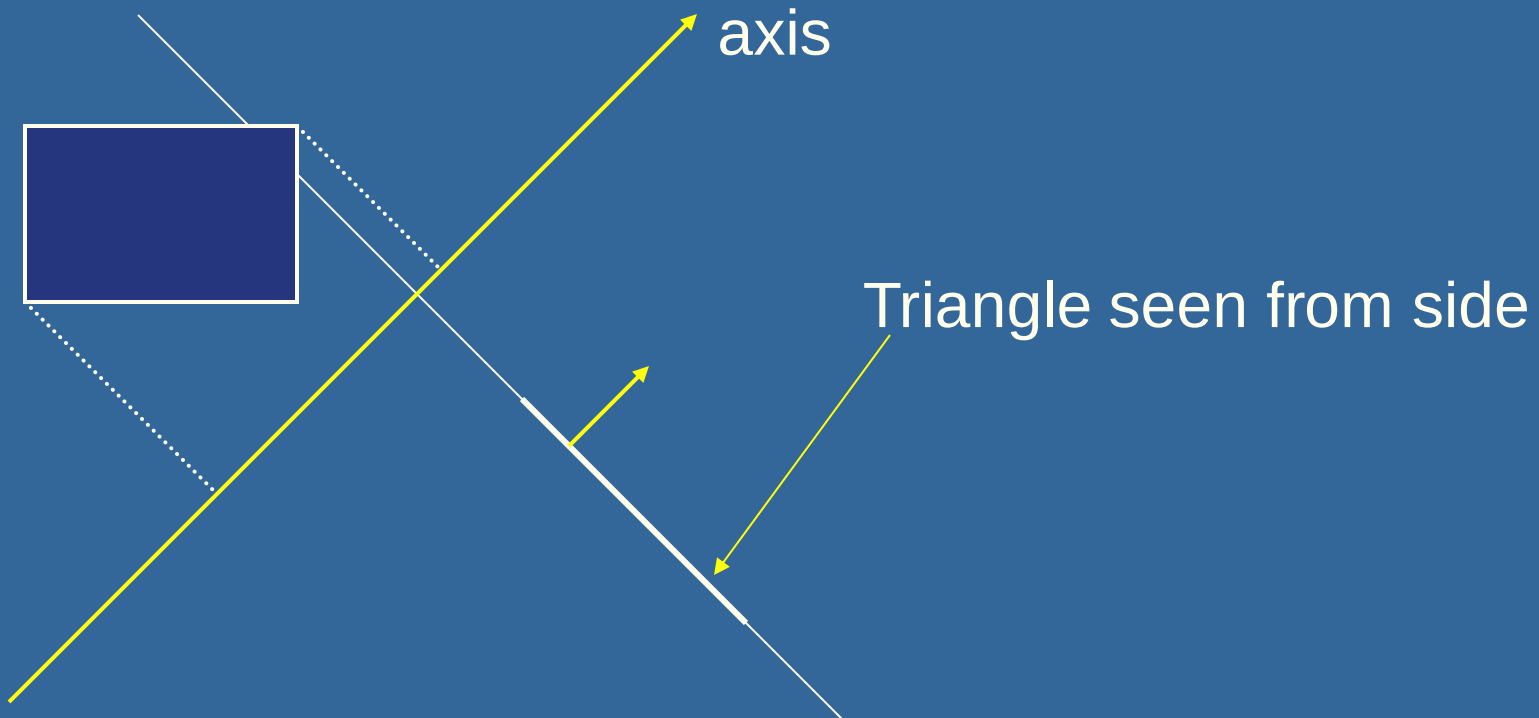
SAT example: Triangle/Box

- E.g an axis-aligned box and a triangle
- **1)** test the axes that are face normals of **the box:**
  - That is, x,y, and z



Triangle/Box with SAT (2)

- Assume that they overlapped on x,y,z
- Must continue testing
- **2)** the face normal of **the triangle**



Triangle/Box with SAT (3)

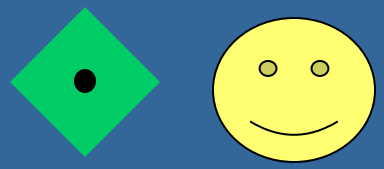
- If still no separating axis has been found…
- **3)** Test axes that are cross products of the edges: $\mathbf{t}=\mathbf{e}_{box} \times \mathbf{e}_{triangle}$
- Example:
  - x-axis from box: $\mathbf{e}_{box}=(1,0,0)$
  - $\mathbf{e}_{triangle}=\mathbf{v}_1-\mathbf{v}_0$
- Test up to all such combinations
  - If there is at least one separating axis, then the objects do not collide
  - Else they do overlap

Rules of Thumb for Intersection Testing

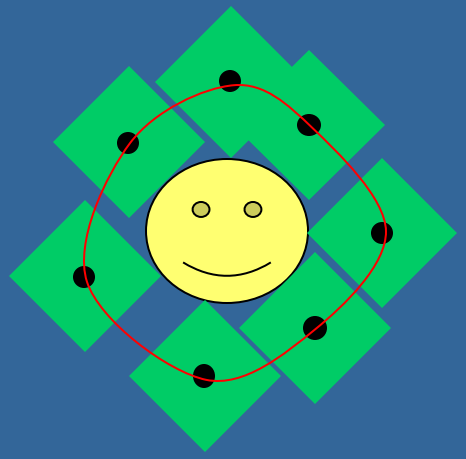
- Acceptance and rejection test
  - Try them early on to make a fast exit
- Postpone expensive calculations if possible
- Use dimension reduction
  - E.g. several one-dimensional tests instead of one complex 3D test, or 2D instead of 3D
- Share computations between objects if possible

An Intersection Trick

Create new shape by sweeping object 1 as close as possible outside object 2. The sweep creates a new 3:rd object (red).
Now, it is enough to test if the center point of object 1 is outside object 3.
Works for all objects and all dimensions.
May be computationally expensive to create the sweep, though.

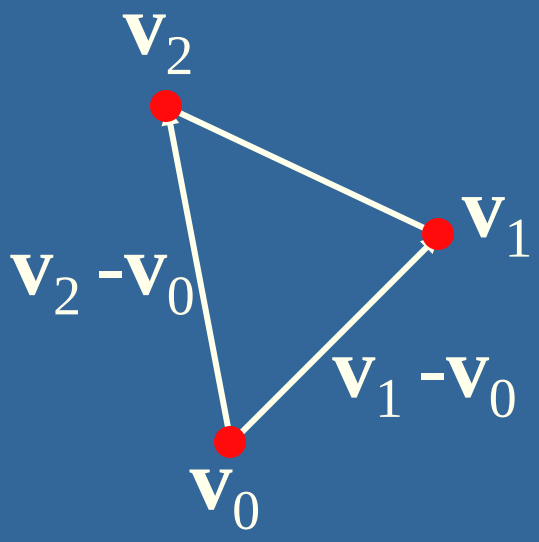


Obj 1 Obj 2



Another analytical example: Ray/Triangle in detail

Make 2 functions, for points along ray and inside triangle and set those 2 functions equal to each other.



- Ray: $\mathbf{r}(t)=\mathbf{o}+t\mathbf{d}$
- Triangle vertices: $\mathbf{v}_0$, $\mathbf{v}_1$, $\mathbf{v}_2$
- A point in the triangle:

$$\mathbf{t}(u,v) = \mathbf{v}_0 + u(\mathbf{v}_1 - \mathbf{v}_0) + v(\mathbf{v}_2 - \mathbf{v}_0)$$

where $[u,v>=0,\ u+v<=1]$ is inside triangle

- Set $\mathbf{t}(u,v)=\mathbf{r}(t)$, and solve for t, u, v:

$\mathbf{v}_0+u(\mathbf{v}_1 - \mathbf{v}_0) +v(\mathbf{v}_2 - \mathbf{v}_0) = \mathbf{o}+t\mathbf{d}$ (3 eq., one in each of x,y,z dim.)

$\Rightarrow -t\mathbf{d} + u(\mathbf{v}_1 - \mathbf{v}_0) + v(\mathbf{v}_2 - \mathbf{v}_0) = \mathbf{o}-\mathbf{v}_0$

$\Rightarrow [-\mathbf{d}, (\mathbf{v}_1 - \mathbf{v}_0), (\mathbf{v}_2 - \mathbf{v}_0)]\ [t, u, v]^T = \mathbf{o}-\mathbf{v}_0$

$$\begin{pmatrix} | & | & | \\ -\mathbf{d} & \mathbf{v}_1 - \mathbf{v}_0 & \mathbf{v}_2 - \mathbf{v}_0 \\ | & | & | \end{pmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix} = \begin{pmatrix} | \\ \mathbf{o} - \mathbf{v}_0 \\ | \end{pmatrix}$$

$\mathbf{Ax=b}$

$\mathbf{x=A^{-1}b}$

BONUS

Ray/Triangle (1)

$$\begin{pmatrix} | & | & | \\ -\mathbf{d} & \mathbf{v}_1 - \mathbf{v}_0 & \mathbf{v}_2 - \mathbf{v}_0 \\ | & | & | \end{pmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix} = \begin{pmatrix} | \\ \mathbf{o} - \mathbf{v}_0 \\ | \end{pmatrix}$$

- Solve for *t,u,v* using Cramer's rule for a system of *n* linear equations with *n* unknowns: $\boldsymbol{A\,x = b}$

Cramer's rule:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix} \Rightarrow x = \frac{\begin{vmatrix} j & b & c \\ k & e & f \\ l & h & i \end{vmatrix}}{\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}}, \quad y = \frac{\begin{vmatrix} a & j & c \\ d & k & f \\ g & l & i \end{vmatrix}}{\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}}, \text{ and } z = \frac{\begin{vmatrix} a & b & j \\ d & e & k \\ g & h & l \end{vmatrix}}{\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}}$$

Simplify our equation system by setting:

$$\mathbf{e}_1 = \mathbf{v}_1 - \mathbf{v}_0 \qquad \mathbf{e}_2 = \mathbf{v}_2 - \mathbf{v}_0 \qquad \mathbf{s} = \mathbf{o} - \mathbf{v}_0$$

$$\Rightarrow \begin{pmatrix} | & | & | \\ -\mathbf{d} & \mathbf{e}_1 & \mathbf{e}_2 \\ | & | & | \end{pmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix} = \begin{pmatrix} | \\ \mathbf{s} \\ | \end{pmatrix}$$

Cramer's rule gives:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{\det(-\mathbf{d}, \mathbf{e}_1, \mathbf{e}_2)} \begin{pmatrix} \det(\mathbf{s}, \mathbf{e}_1, \mathbf{e}_2) \\ \det(-\mathbf{d}, \mathbf{s}, \mathbf{e}_2) \\ \det(-\mathbf{d}, \mathbf{e}_1, \mathbf{s}) \end{pmatrix}$$

BONUS

Ray/Triangle (2)

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{\det(-\mathbf{d}, \mathbf{e}_1, \mathbf{e}_2)} \begin{pmatrix} \det(\mathbf{s}, \mathbf{e}_1, \mathbf{e}_2) \\ \det(-\mathbf{d}, \mathbf{s}, \mathbf{e}_2) \\ \det(-\mathbf{d}, \mathbf{e}_1, \mathbf{s}) \end{pmatrix}$$

- To compute determinant

Use this fact : $\det(\mathbf{a},\mathbf{b},\mathbf{c}) = (\mathbf{a}\times\mathbf{b})\cdot\mathbf{c} = -(\mathbf{a}\times\mathbf{c})\cdot\mathbf{b}$

This gives:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(\mathbf{d}\times\mathbf{e}_2)\cdot\mathbf{e}_1} \begin{pmatrix} (\mathbf{s}\times\mathbf{e}_1)\cdot\mathbf{e}_2 \\ (\mathbf{d}\times\mathbf{e}_2)\cdot\mathbf{s} \\ (\mathbf{s}\times\mathbf{e}_1)\cdot\mathbf{d} \end{pmatrix}$$

- Share factors to speed up computations:

$$\mathbf{p} = \mathbf{d}\times\mathbf{e}_2$$
$$a = \mathbf{p}\cdot\mathbf{e}_1$$
$$f = 1/a$$

- Compute as little as possible. Then test.

Compute $u = f(\mathbf{p}\cdot\mathbf{s})$

Then test valid bounds:

```
if (u<0 or u>1) exit;
```

Then also test v and u+v <=1

Misc

- There is an Advanced Computer Graphics Seminar Course in sp 4, 7.5p
  - 1-2 seminars every week
    - Advanced CG techniques
  - Do a project of your choice.

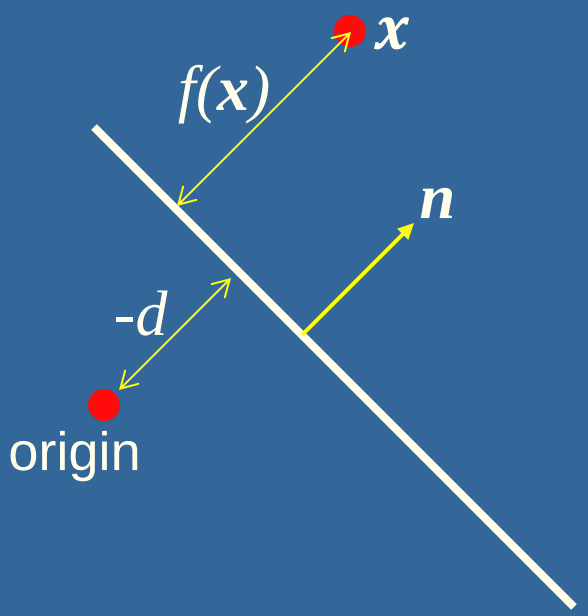
The Plane Equation

Plane : $\pi : \mathbf{n} \cdot \mathbf{p} + d = 0$

If $\boldsymbol{n} \cdot \boldsymbol{x} + d = 0$, then $\boldsymbol{x}$ lies in the plane.

The function $f(\mathbf{x}) = \mathbf{n} \cdot \mathbf{x} + d$ gives the signed distance of $\mathbf{x}$ from the plane. ($\mathbf{n}$ should be normalized.)

- $f(x)>0$ means above the plane
- $f(x)<0$ means below the plane



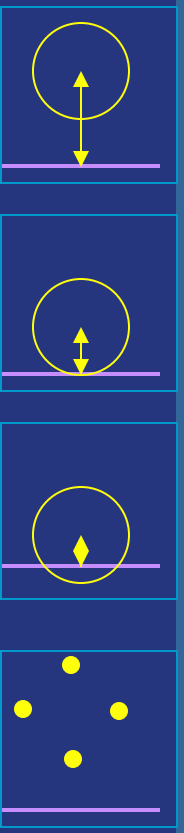
$-d$ is signed distance of origin to plane

Sphere/Plane Box/Plane

Plane : $\pi : \mathbf{n} \cdot \mathbf{p} + d = 0$

Sphere : $\mathbf{c}$ r

AABB: $\mathbf{b}^{\min}$ $\mathbf{b}^{\max}$



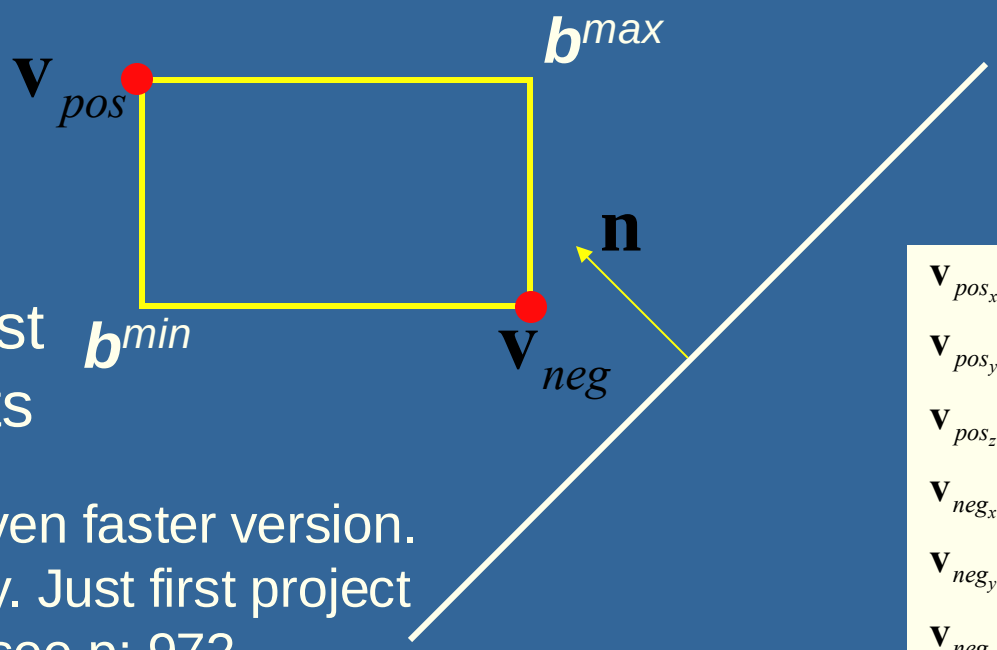
- Sphere: compute $f(\mathbf{c}) = \mathbf{n} \cdot \mathbf{c} + d$
- $f(\mathbf{c})$ is the signed distance
- abs($f(\mathbf{c})$) > r no collision
- abs($f(\mathbf{c})$) = r sphere touches the plane
- abs($f(\mathbf{c})$) < r sphere intersects plane

- Box: insert all 8 corners
- If all f's have the same sign, then all points are on the same side, and no collision

$$\text{Plane}: \quad \pi : \mathbf{n} \cdot \mathbf{p} + d = 0$$
$$\text{Sphere}: \quad \mathbf{c} \quad r$$
$$\text{Box}: \quad \mathbf{b}^{\min} \quad \mathbf{b}^{\max}$$

AABB/plane

- The smart way (shown in 2D)
- Find the two vertices that have the most positive and most negative value when tested againt the plane



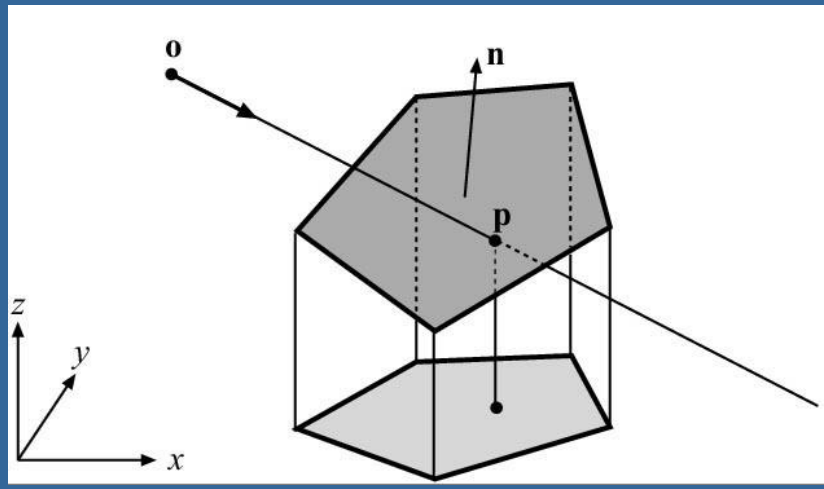
Need only test the red points

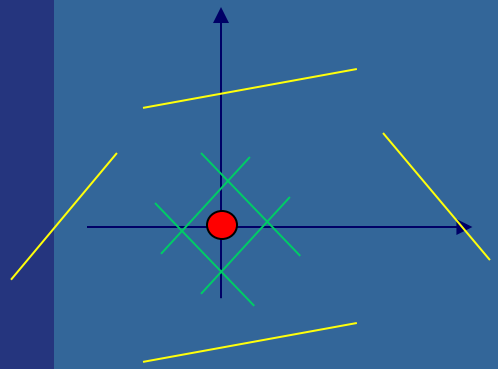
$$\mathbf{v}_{pos_x} = (\mathbf{n}_x > 0) ? \mathbf{b}_{\max_x} : \mathbf{b}_{\min_x}$$
$$\mathbf{v}_{pos_y} = (\mathbf{n}_y > 0) ? \mathbf{b}_{\max_y} : \mathbf{b}_{\min_y}$$
$$\mathbf{v}_{pos_z} = (\mathbf{n}_z > 0) ? \mathbf{b}_{\max z} : \mathbf{b}_{\min_z}$$
$$\mathbf{v}_{neg_x} = (\mathbf{n}_x < 0) ? \mathbf{b}_{\max_x} : \mathbf{b}_{\min_x}$$
$$\mathbf{v}_{neg_y} = (\mathbf{n}_y < 0) ? \mathbf{b}_{\max_y} : \mathbf{b}_{\min_y}$$
$$\mathbf{v}_{neg_z} = (\mathbf{n}_z < 0) ? \mathbf{b}_{\max z} : \mathbf{b}_{\min_z}$$

See page 970 for even faster version. OBB almost as easy. Just first project **n** on OBB's axes – see p: 972

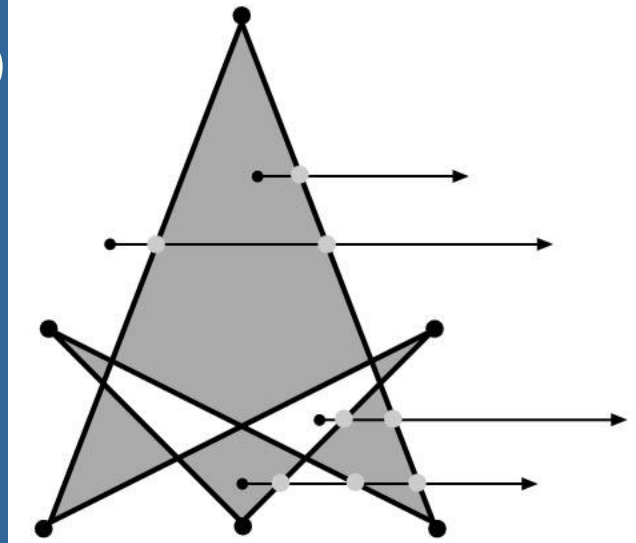
Ray/Polygon: The Crossing Test

- Intersect ray with polygon plane
- Project from 3D to 2D
- How?
- Find $\max(|n_x|,|n_y|,|n_z|)$
- Skip that coordinate!
- Then, count crossing in 2D





The number of times this ray intersects the polygon edges is counted. If the number of crossings is odd, the point is inside the polygon. If even, the point is outside.



Volume/Volume tests

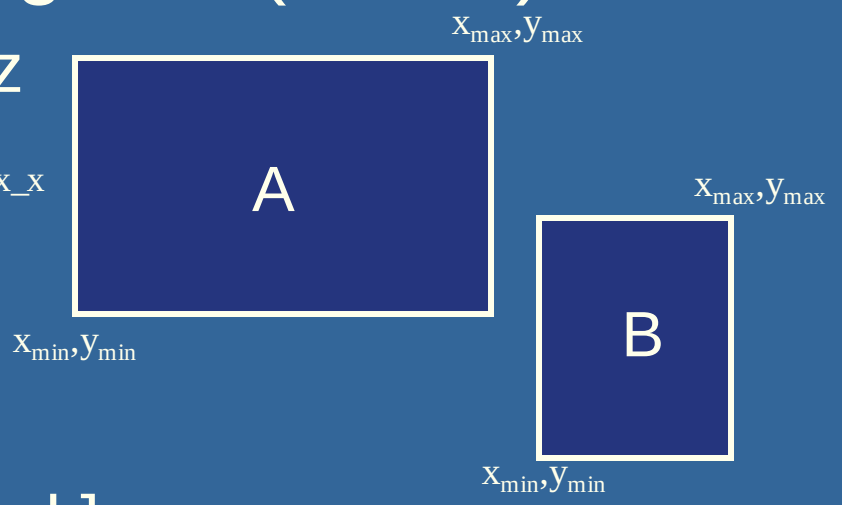
If $A_{min\_x} > B_{max\_x}$ or
$A_{min\_y} > B_{max\_y}$ or
$A_{min\_z} > B_{max\_z}$ or
$B_{min\_x} > A_{max\_x}$ or
$B_{min\_y} > A_{max\_y}$ or
$B_{min\_z} > A_{max\_z}$
return no_intersection
Else
return intersection.

- Used in collision detection
- Sphere/sphere
  - Compute squared distance between sphere centers, and compare to $(r_1+r_2)^2$:

$\|\mathbf{c}_1-\mathbf{c}_2\| \le (r_1+r_2) \Rightarrow$
$(\mathbf{c}_1-\mathbf{c}_2)\bullet(\mathbf{c}_1-\mathbf{c}_2) \le (r_1+r_2)^2$

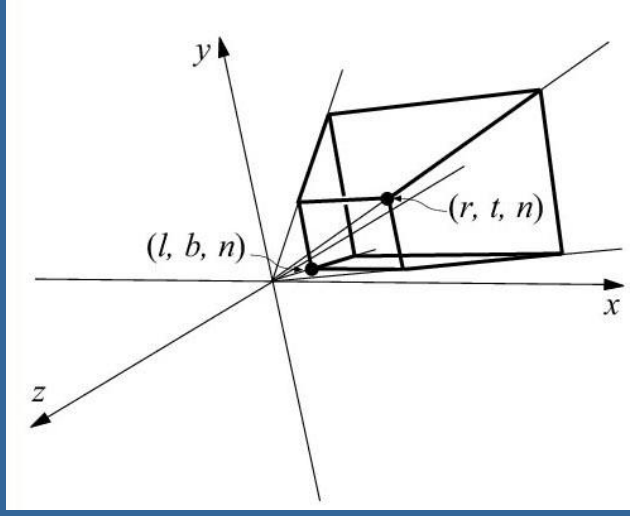
- Axis-Aligned Bounding Box (AABB)
  - Test in 1D for x,y, and z

If $B_{min\_x} > A_{max\_x}$ or $A_{min\_x} > B_{max\_x}$
=> no intersection along x.
… same with y,z …



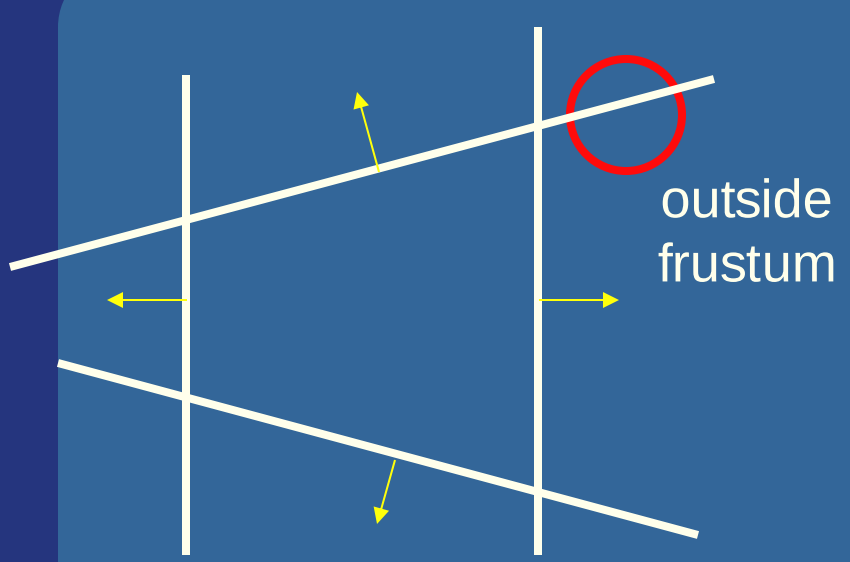
- Oriented Bounding boxes
  - Use SAT [details in book]

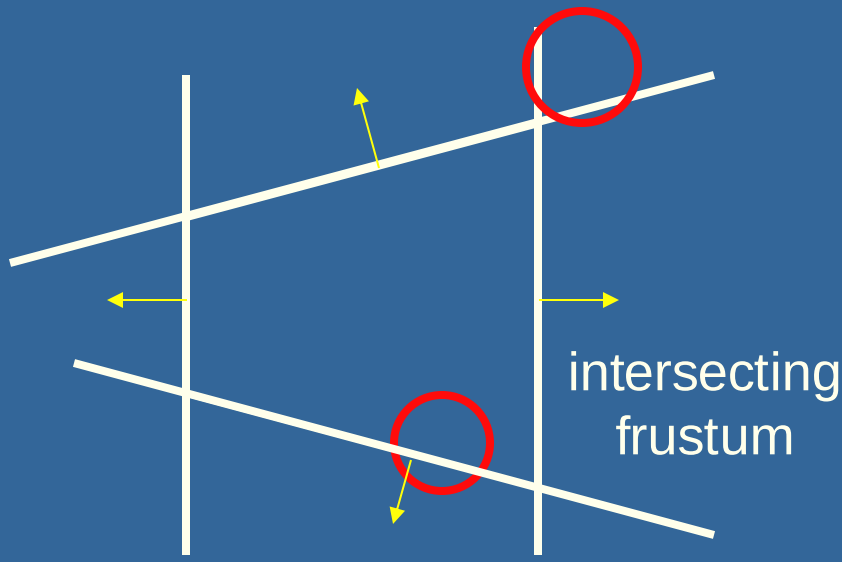
View frustum testing



- View frustum is 6 planes:
- Near, far, right, left, top, bottom
- Get their plane equations from projection matrix
  - Let all positive half spaces be outside frustum
  - p. 983-984.
- Sphere/frustum common approach:
  - Test sphere against each of the 6 frustum planes:
    - If outside the plane => no intersection
    - If intersecting the plane or inside, continue
  - If not outside after all six planes, then conservatively concider sphere as inside or intersecting
- Example follows…

View frustum testing example





- Not exact test, but not incorrect
  - A sphere that is reported to be inside, can actually be outside
  - Not vice versa
- Similar frustum test for boxes

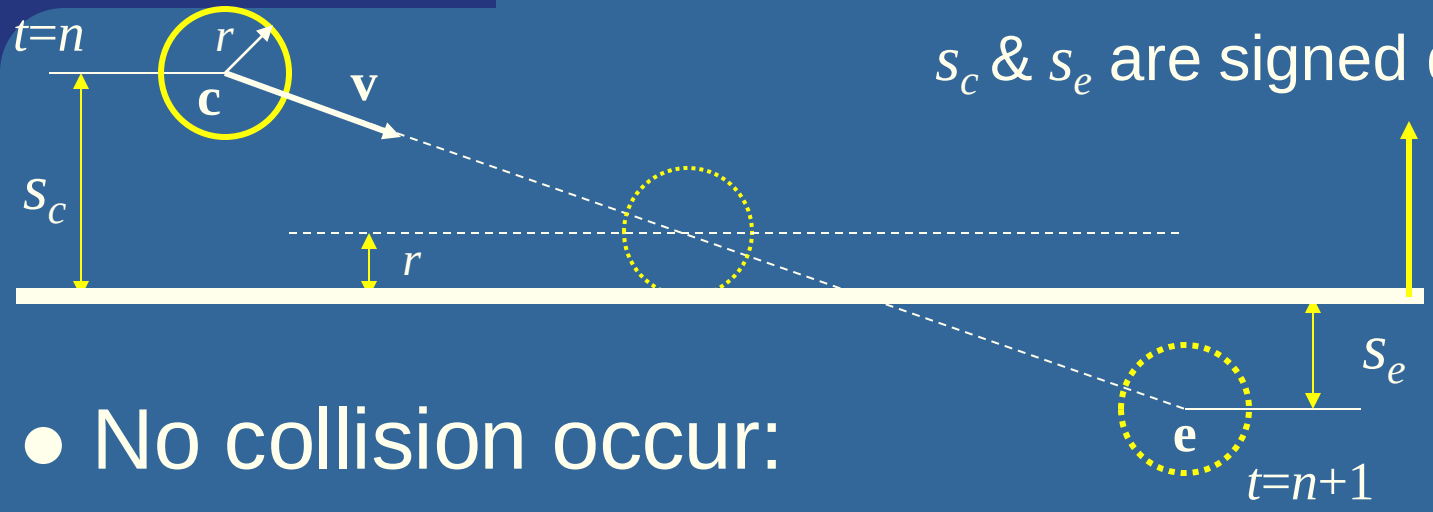
Dynamic Intersection Testing
[In book: 620-628]

- Testing is often done every rendered frame, i.e., at discrete time intervals
- Therefore, you can get ”quantum effects”

- Dynamic testing deals with this
- Is more expensive
- Deals with a time interval: time between two frames

Dynamic intersection testing Sphere/Plane

BONUS



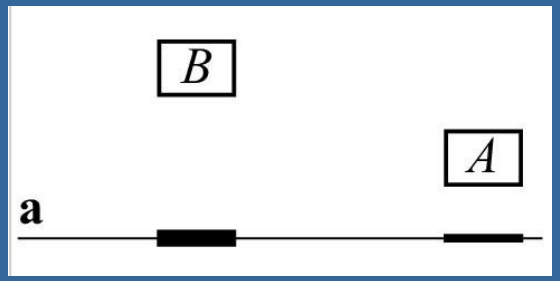
s_c & s_e are signed distances

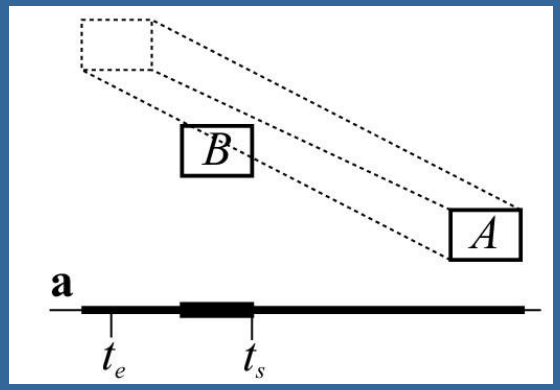
- No collision occur:
  - If they are on the same side of the plane ($s_c s_e > 0$)
    - and: $|s_c| > r$ and $|s_e| > r$
- Otherwise, sphere can move $|s_c| - r$
- Time of collision: $t_{cd} = n + \frac{s_c - r}{s_c - s_e}$ s_e is signed distance
- Response: reflect **v** around **n**, and move $(1-t_{cd})\mathbf{r}$ (**r**=refl vector)

BONUS

Dynamic Separating Axis Theorem

- SAT: tests one axis at a time for overlap





- Same with DSAT, but:
  - Use a relative system where B is fixed
    - i.e., compute A's relative motion to B.
  - Need to adjust A's projection on the axis so that the interval moves on the axis as well
- Need to test same axes as with SAT
- Same criteria for overlap/disjoint:
  - If no overlap on axis => disjoint
  - If overlap on all axes => objects overlap

Exercises

- Create a function (by writing code on paper) that tests for intersection between:
  - two spheres
  - a ray and a sphere
  - view frustum and a sphere

What you need to know

- Analytic test:
  - Be able to compute ray vs sphere or other similar formula
  - Ray/triangle, ray/plane
  - Point/plane, Sphere/plane,
  - Know expressions for ray, sphere, cylinder, plane, triangle

- Geometrical tests
  - Ray/box with slab-test
  - Ray/polygon (3D->2D)
  - box/plane
  - AABB/AABB
  - View frustum vs spheres/AABB:s/BVHs.
  - Separating Axis Theorem (SAT)

- Know what a dynamic test is

BONUS

Scan Line Fill

Set active edges to AB and AC

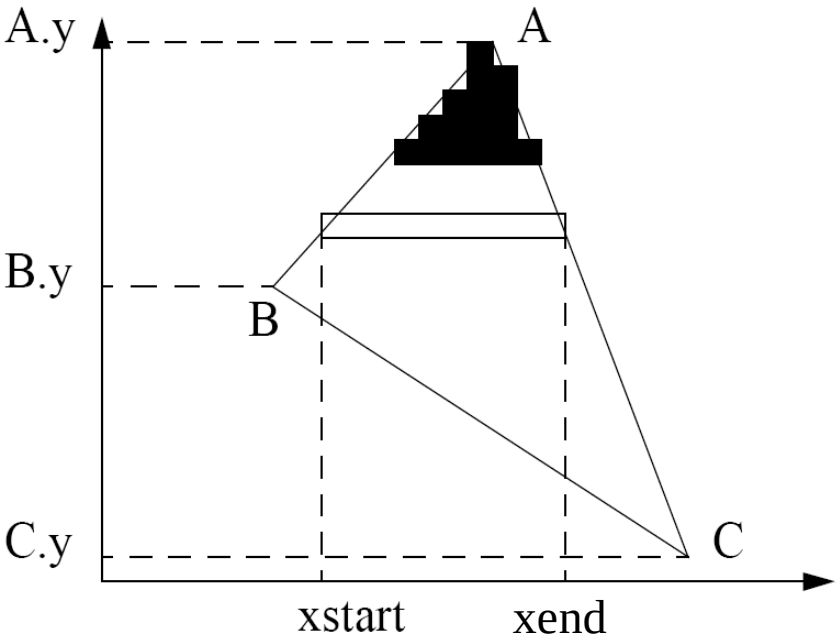
For y = A.y, A.y-1,...,C.y

If y=B.y → exchange AB with BC

Compute xstart and xend.
Interpolate color, depth, texcoords etc for points (xstart,y) and (xend,y)

For x = xstart, xstart+1, ...,xend

Compute color, depth etc for (x,y) using interpolation.

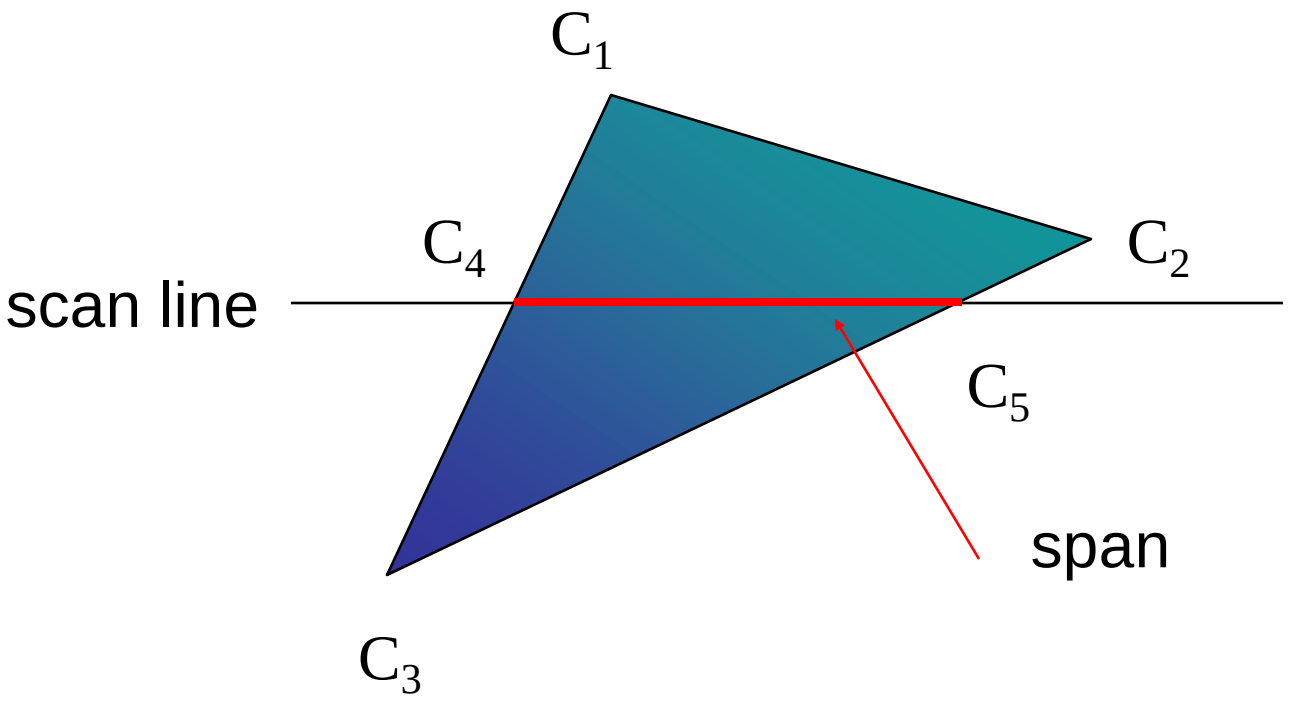


This is one way to rasterize a triangle. (Nowadays, stamp rasterizion is typically used - see web if you are interested)

BONUS

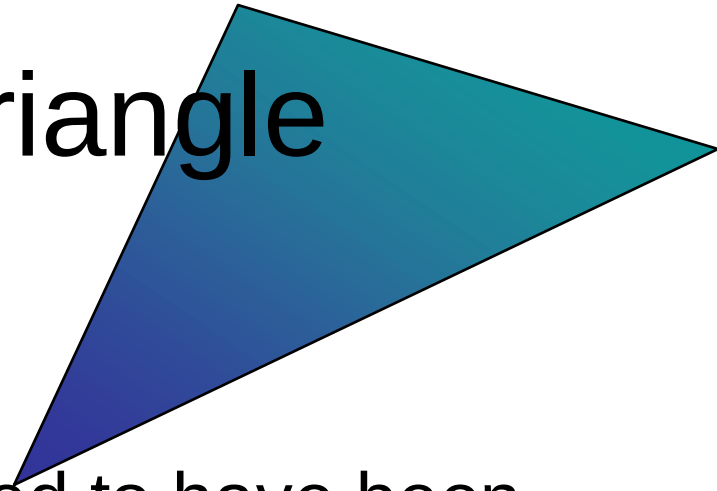
Using Interpolation

C_1 C_2 C_3 output by the vertex shader.
C_4 determined by interpolating between C_1 and C_3
C_5 determined by interpolating between C_2 and C_3
interpolate between C_4 and C_5 along span



BONUS

Rasterizing a Triangle

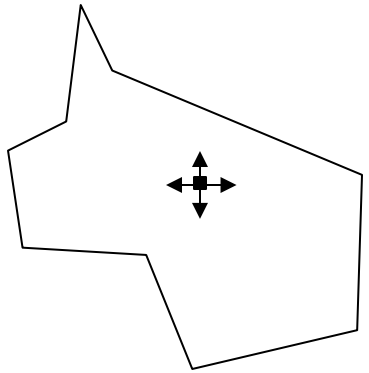


- Convex Polygons only
- Nonconvex polygons assumed to have been tessellated
- Shader results (e.g. colors) have been computed for the vertices. Depth occlusion resolved with z-buffer.
  - March across scan lines interpolating vertex shader output parameters, as input to the fragment shader.
  - Incremental work small

Flood Fill

- Fill can be done recursively if we know a seed point located inside (WHITE)
- Scan convert edges into buffer in edge/inside color (BLACK)

```
flood_fill(int x, int y) {
     if(read_pixel(x,y)= = WHITE) {
        write_pixel(x,y,BLACK);
        flood_fill(x-1, y);
        flood_fill(x+1, y);
        flood_fill(x, y+1);
        flood_fill(x, y-1);
}    }
```



- DOT product