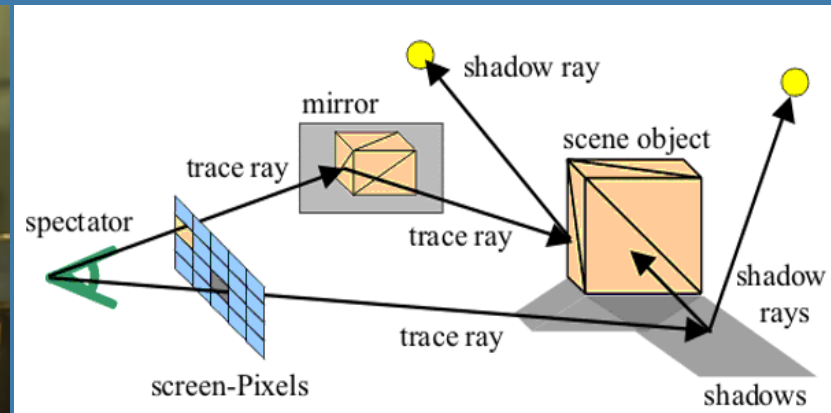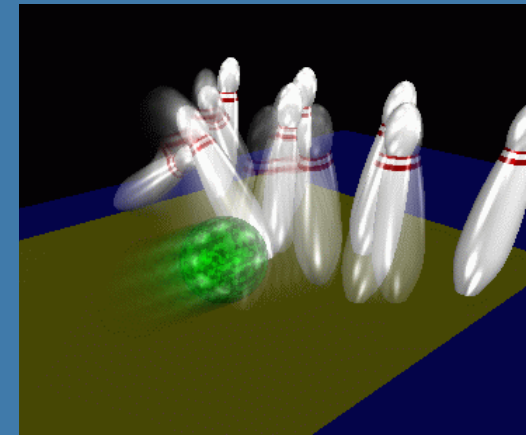# Intersection Testing
Chapter 16

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)

Example

Midtown Madness 3, DICE

Some basic geometrical primitives
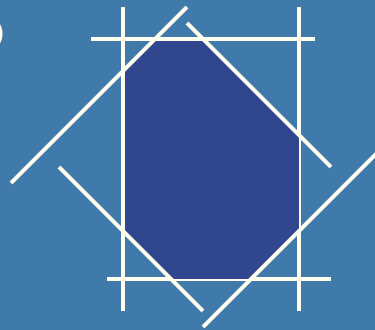
- Ray:

- Sphere:

- Box
  - Axis-aligned (AABB)
  - Oriented (OBB)

- *k*-DOP

Four different techniques

- Analytical

- Geometrical

- Separating axis theorem (SAT)

- Dynamic tests

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

Analytical: Ray/sphere test

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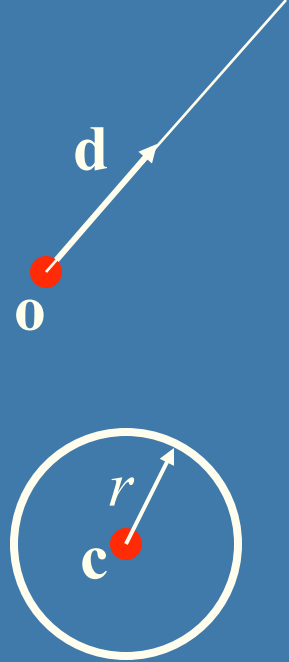- Replace $\mathbf{p}$ by $\mathbf{r}(t)$, and square it:

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

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

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

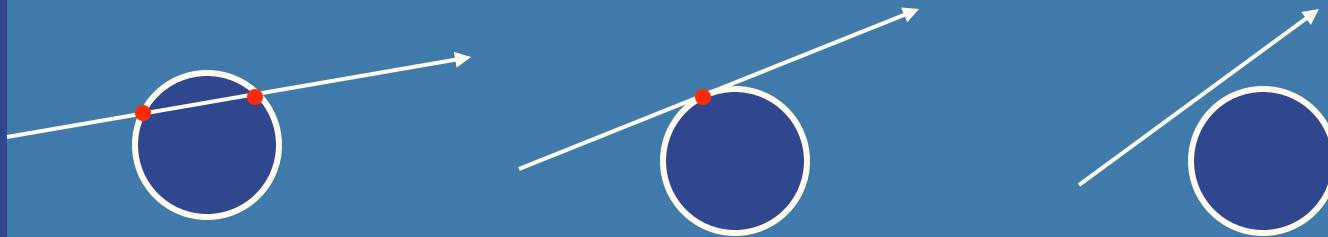$$(\mathbf{d}\cdot\mathbf{d})t^2+2((\mathbf{o}-\mathbf{c})\cdot\mathbf{d})t+(\mathbf{o}-\mathbf{c})\cdot(\mathbf{o}-\mathbf{c})-r^2=0$$

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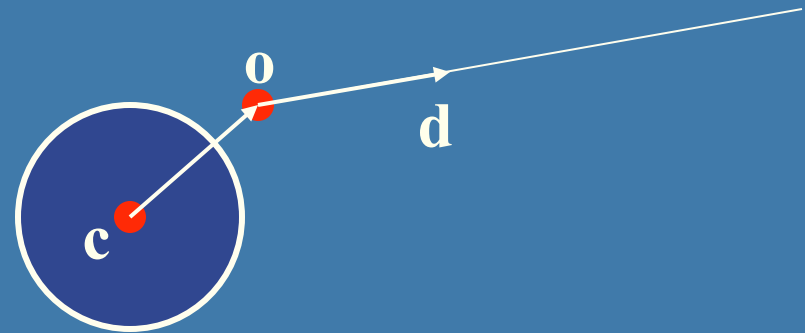
Analytical, continued

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

- Be a little smart…

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

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

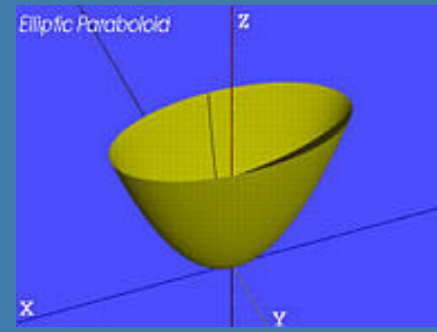- Such tests are called "rejection tests"

- Other shapes: $p_x^2 + p_v^2 = r^2$

$$(p_x / a)^2 + (p_v / b)^2 + (p_z / c)^2 = 1$$

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

Elliptic Paraboloid

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):

BOX
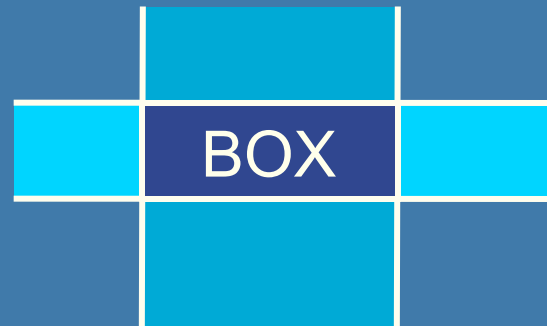
Geometrical: Ray/Box Intersection (2)

- Intersect the 2 planes of each slab with the ray
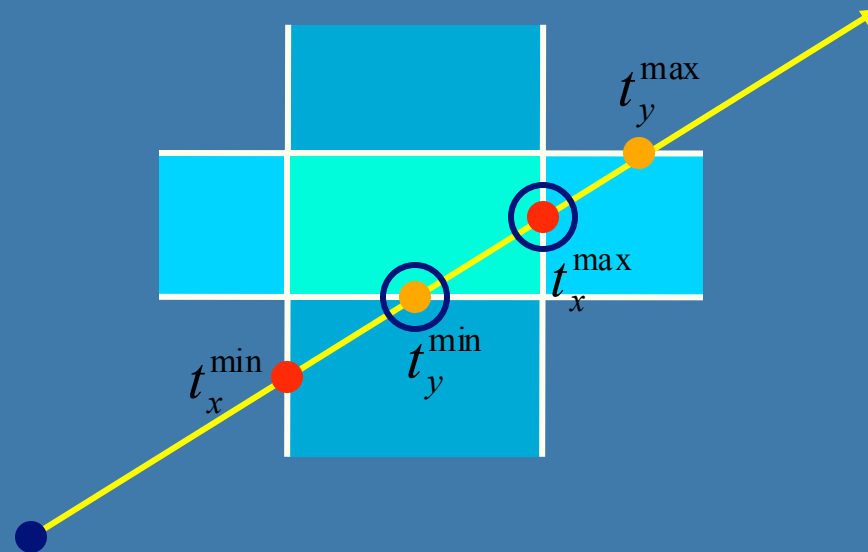
- Keep max of t^{min} and min of t^{max}
- If $t^{min} < t^{max}$ then we got an intersection
- Special case when ray parallell to slab

Separating Axis Theorem (SAT) Page 563 in book

- Two convex polyhedrons, A and B, are disjoint if any of the following axes separate the objects:
  - An axis orthogonal to a face of A
  - An axis orthogonal to a face of B
  - An axis formed from the cross product of one edge from each of A and B
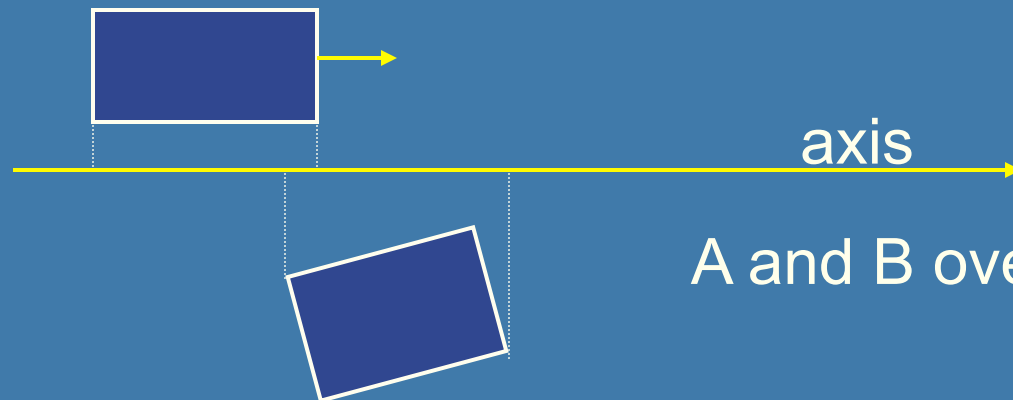
axis

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 orthogonal to the faces of the box
- That is, x,y, and z
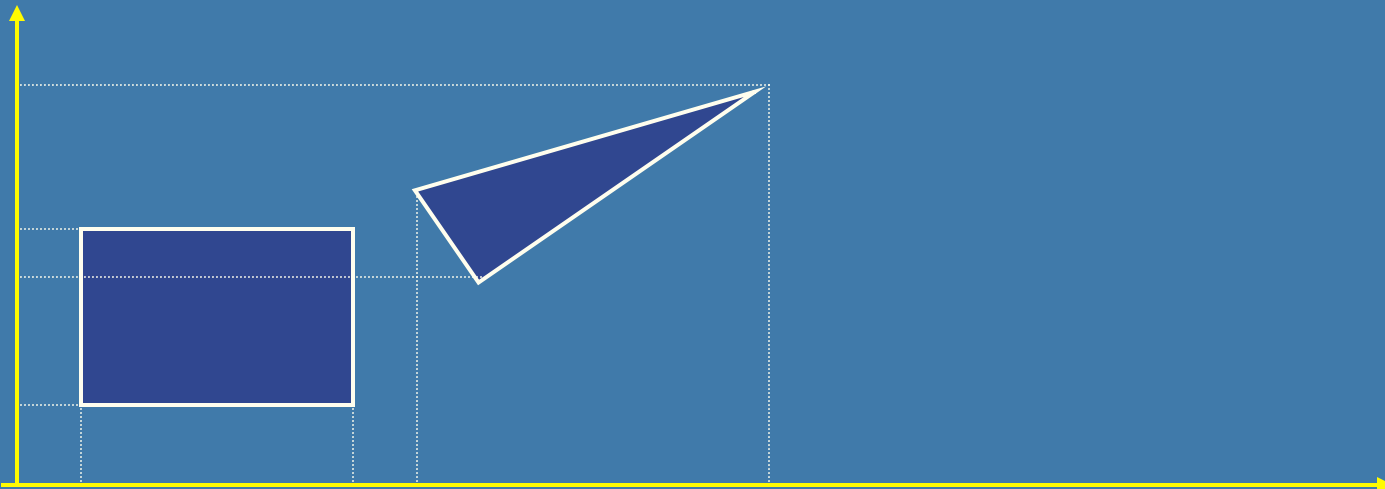
Triangle/Box with SAT (2)

- Assume that they overlapped on x,y,z
- Must continue testing
- **2)** Axis orthogonal to face of triangle

axis

Triangle seen from side
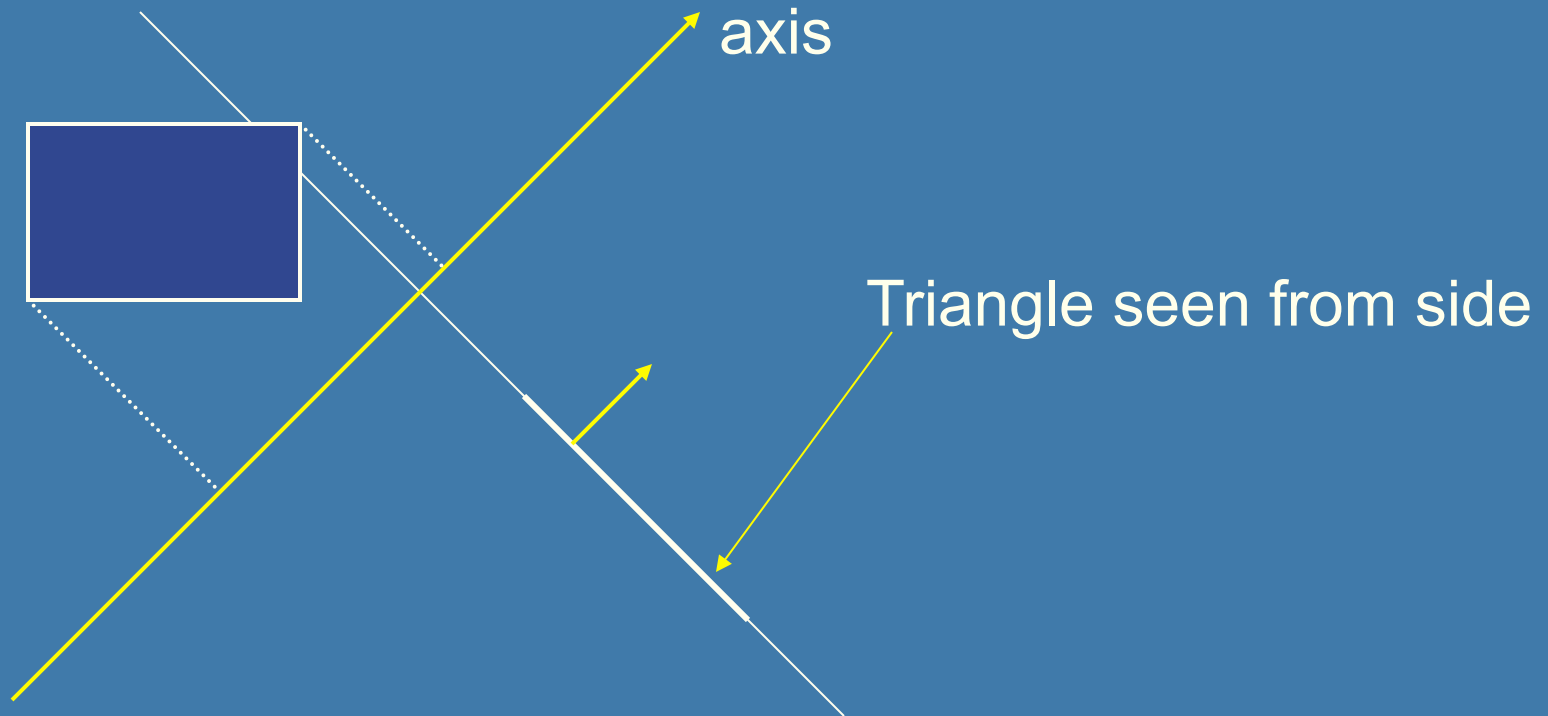
Triangle/Box with SAT (3)

- If still no separating axis has been found…
- **3)** Test axis: $\mathbf{t}=\mathbf{e}_{box}$ x $\mathbf{e}_{triangle}$
- Example:
  - x-axis from box: $\mathbf{e}_{box}=(1,0,0)$
  - $\mathbf{e}_{triangle}=\mathbf{v}_1-\mathbf{v}_0$
- Test 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. 3 one-dimensional tests instead of one complex 3D test, or 2D instead of 3D
- Share computations between objects if possible
- Timing!

Another analytical example: Ray/Triangle in detail

- 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) = $
  $= (1-u-v)\mathbf{v}_0 + u\mathbf{v}_1 + v\mathbf{v}_2 \quad [u,v >= 0,\ u+v <= 1]$
- Set $\mathbf{t}(u,v) = \mathbf{r}(t)$, and solve!

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

$\mathbf{v}_2$

$\mathbf{v}_1$

$\mathbf{v}_2 - \mathbf{v}_0$

$\mathbf{v}_1 - \mathbf{v}_0$

$\mathbf{v}_0$

Ray/Triangle (2)

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

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

- Solve with Cramer's rule:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix}$$

$$A\,x = b$$

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

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

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

Ray/Triangle (2)

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

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

- Solve with Cramer's rule:

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

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

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

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

- Share factors to speed up computations

Ray/Triangle (3) Implementation

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

- Be smart!
  - Compute as little as possible. Then test
- Examples:
$$\mathbf{p} = \mathbf{d} \times \mathbf{e}_2$$
$$a = \mathbf{p} \cdot \mathbf{e}_1$$
$$f = 1/a$$

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

- Then test valid bounds

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

Point/Plane

● Insert a point $\mathbf{x}$ into plane equation:

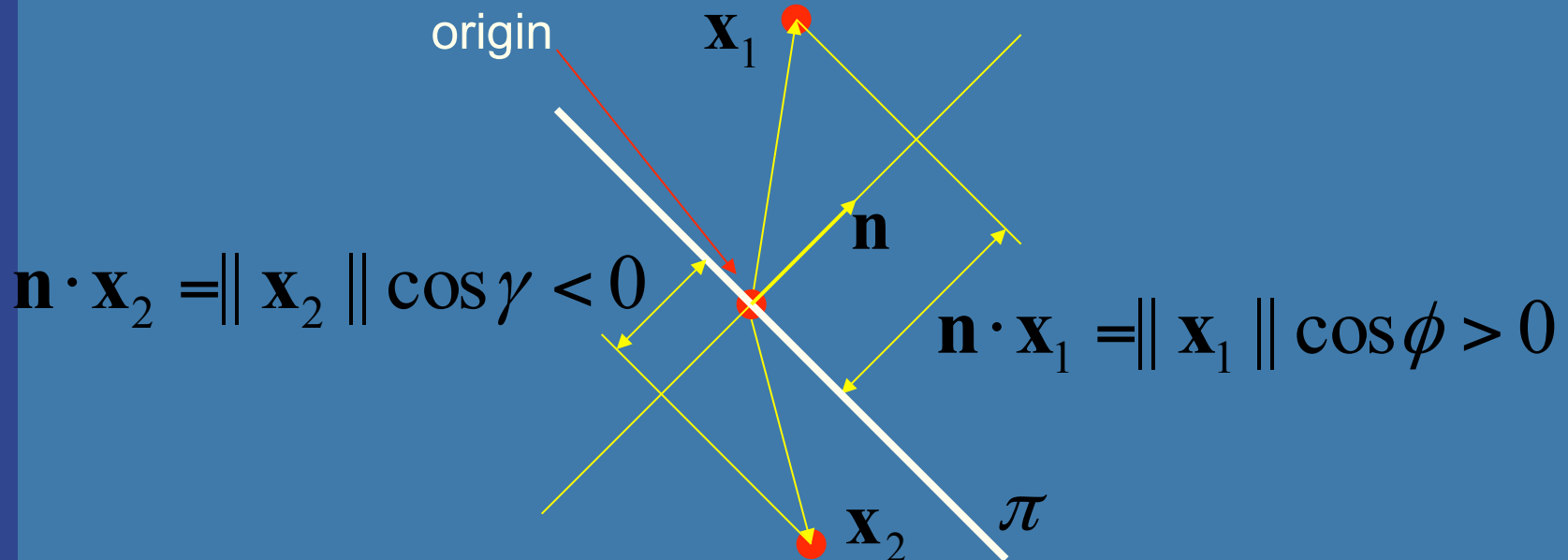$$f(\mathbf{x}) = \mathbf{n} \cdot \mathbf{x} + d$$

$$f(\mathbf{x}) = \mathbf{n} \cdot \mathbf{x} + d = 0 \quad \text{for } \mathbf{x}\text{'s on the plane}$$

$$f(\mathbf{x}) = \mathbf{n} \cdot \mathbf{x} + d < 0 \quad \text{for } \mathbf{x}\text{'s on one side of the plane}$$

$$f(\mathbf{x}) = \mathbf{n} \cdot \mathbf{x} + d > 0 \quad \text{for } \mathbf{x}\text{'s on the other side}$$

Negative
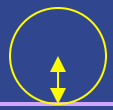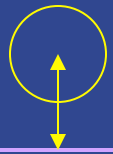half space

Positive
half space

origin

$\mathbf{x}_1$

$\mathbf{n}$

$$\mathbf{n} \cdot \mathbf{x}_2 = \| \mathbf{x}_2 \| \cos \gamma < 0$$

$$\mathbf{n} \cdot \mathbf{x}_1 = \| \mathbf{x}_1 \| \cos \phi > 0$$

$\mathbf{x}_2$

π

Sphere/Plane Box/Plane

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

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

AABB/plane

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

- The smart way (shown in 2D)
- Find the two vertices that have the most positive and most negative value when tested againt the plane
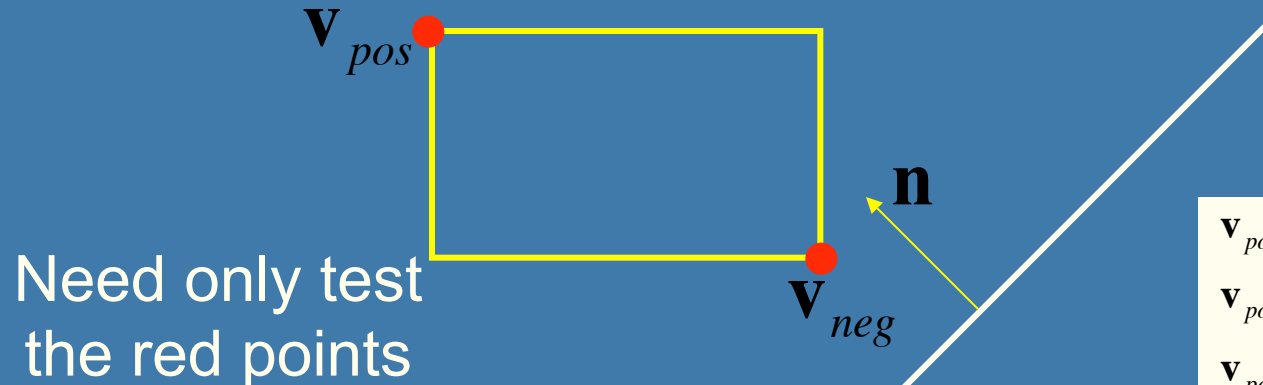
$\mathbf{v}_{pos}$

$\mathbf{n}$

Need only test the red points

$\mathbf{v}_{neg}$

OBB almost as easy. Just first project $\mathbf{n}$ on OBB's axes – see p: 757

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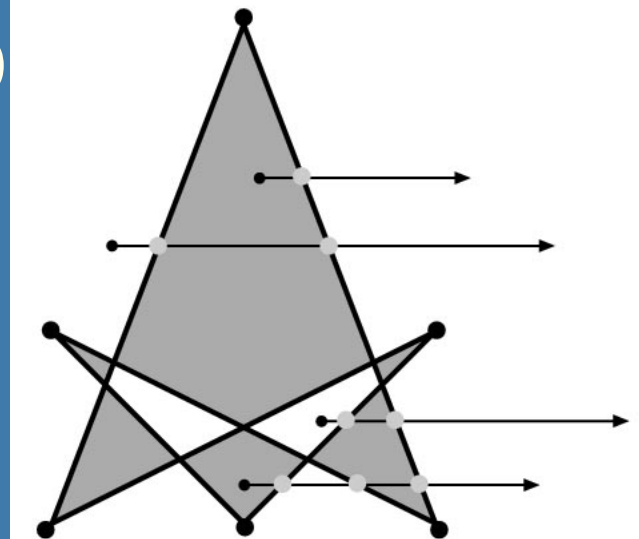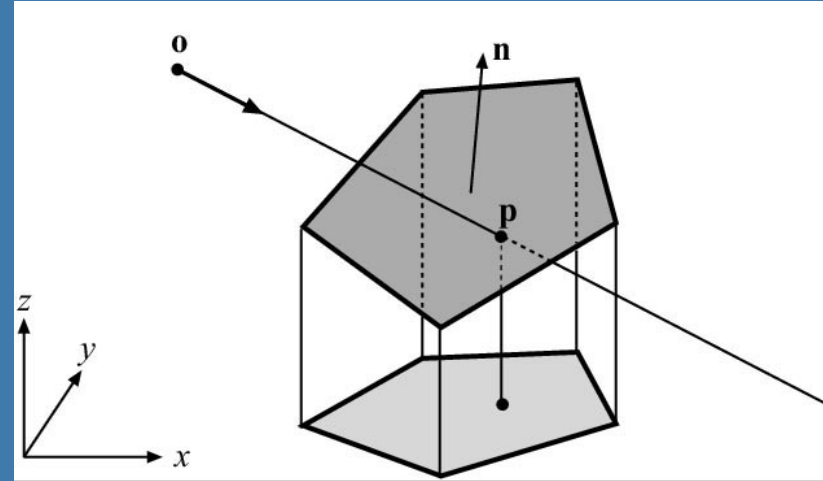$$\mathbf{v}_{neg_x} = (\mathbf{n}_x < 0)?\mathbf{b}_{\max_x} : \mathbf{b}_{\min_x}$$
$$\mathbf{v}_{neg_y} = (\mathbf{n}_y < 0)?\mathbf{b}_{\max_y} : \mathbf{b}_{\min_y}$$
$$\mathbf{v}_{neg_z} = (\mathbf{n}_z < 0)?\mathbf{b}_{\max z} : \mathbf{b}_{\min_z}$$

Ray/Polygon: very briefly

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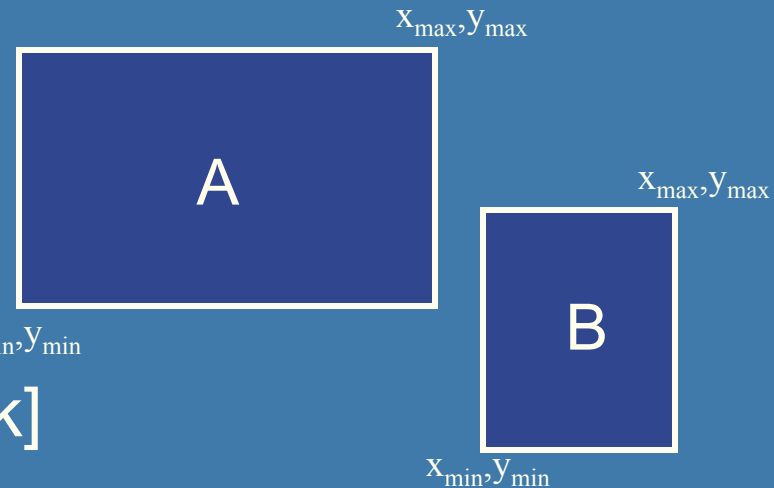- Skip that coordinate!
- Then, count crossing in 2D

Volume/Volume tests

- Used in collision detection

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

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

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

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.

x_{max},y_{max}

A

x_{max},y_{max}

x_{min},y_{min}

B

x_{min},y_{min}

View frustum testing

- View frustum is 6 planes:
- Near, far, right, left, top,
- Create planes from projection matrix
  - Let all positive half spaces be outside frustum
  - Not dealt with here -- p. 773-774, 3rd ed.
- 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…
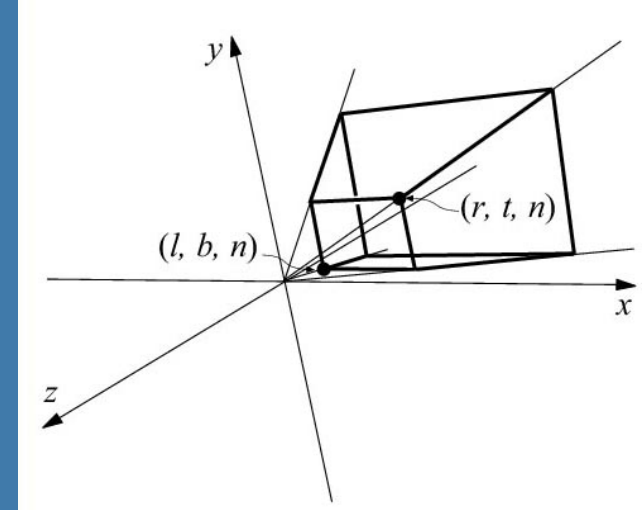
View frustum testing example
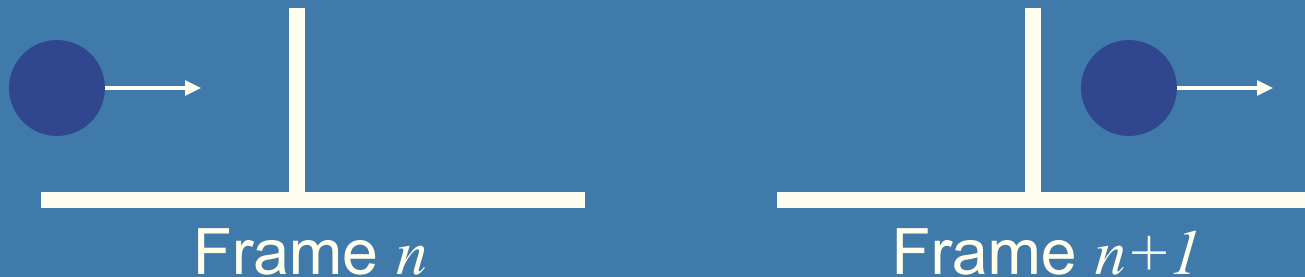
outside
frustum

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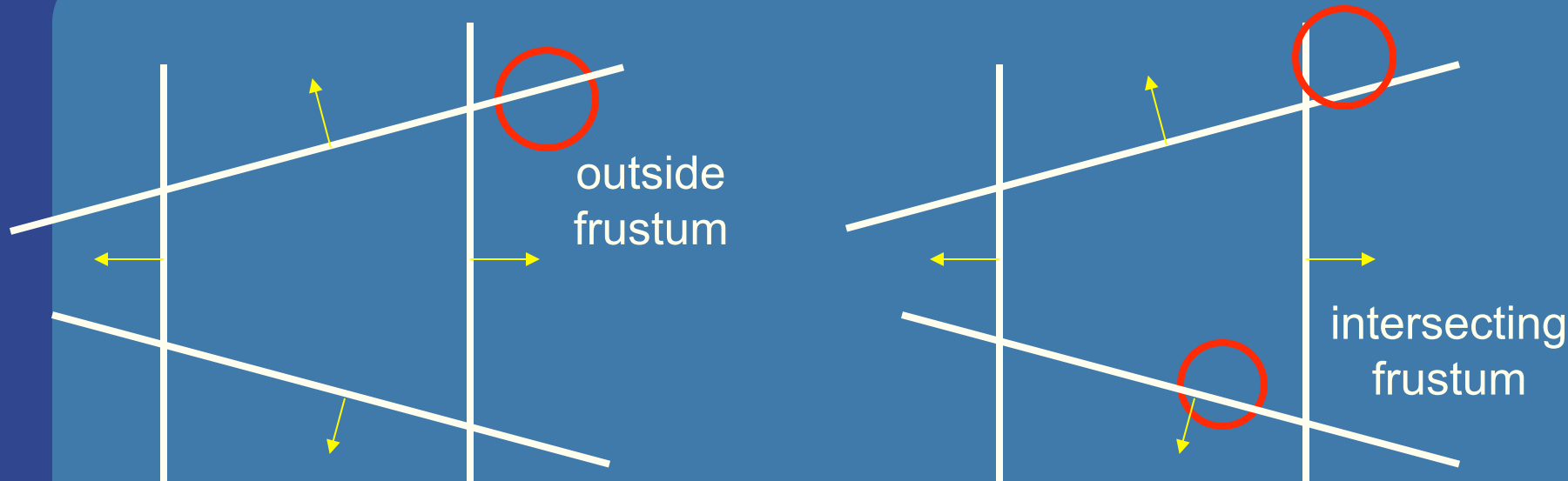
- Not exact test, but not incorrect
  - A sphere that is reported to be inside, can be outside
  - Not vice versa
- Similarly for boxes

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"

Frame n Frame $n+1$

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

Dynamic intersection testing Sphere/Plane

$t{=}n$ r **v** **c**

s_c r

s_c & s_e are signed distances
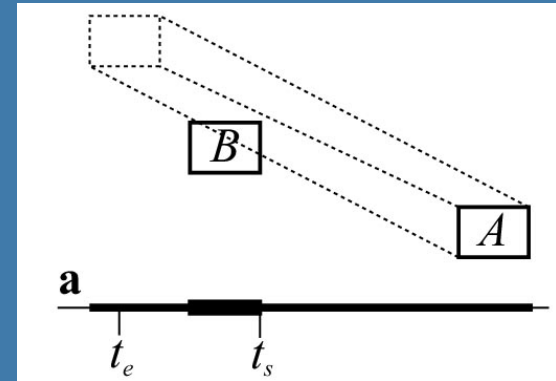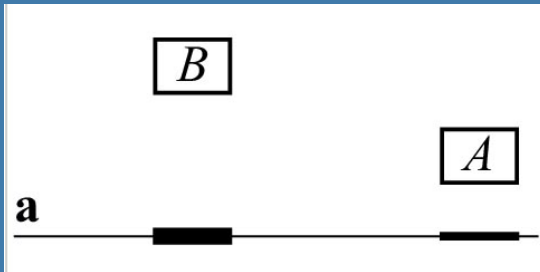
n

s_e

e $t{=}n+1$
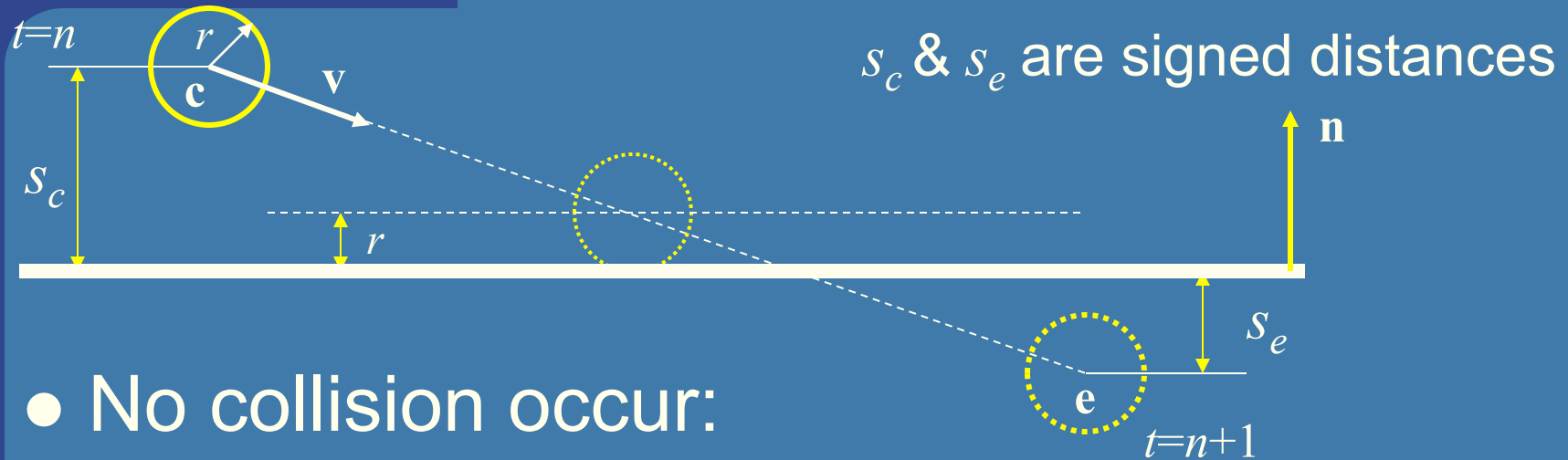
- No collision occur:
  - If they are on the same side of the plane ($s_c s_e > 0$)
    - and: $|s_c| > r$ and $|s_e| > r$

- Otherwise, sphere can move $|s_c|{-}r$

- Time of collision: $t_{cd} = n + \dfrac{s_c - r}{s_c - s_e}$ s_e is signed distance

- Response: reflect **v** around **n**, and move $(1{-}t_{cd})\mathbf{r}$ (**r**=refl vector)

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

Dynamic Sweep-and-Prune

- http://graphics.idav.ucdavis.edu/~dcoming/papers/coming_staadt_vriphys05.pdf

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

Scan Line Fill

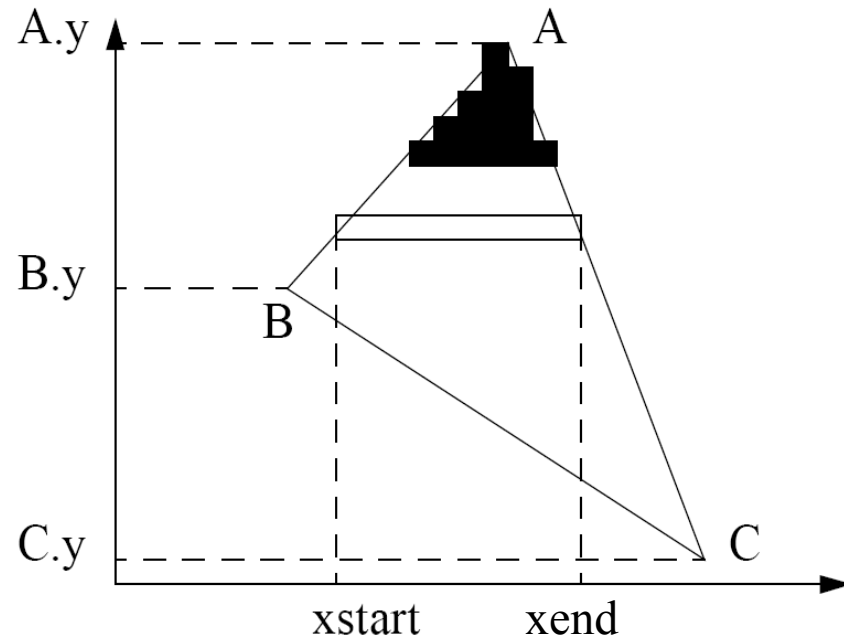Set active edges to AB and AC

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)

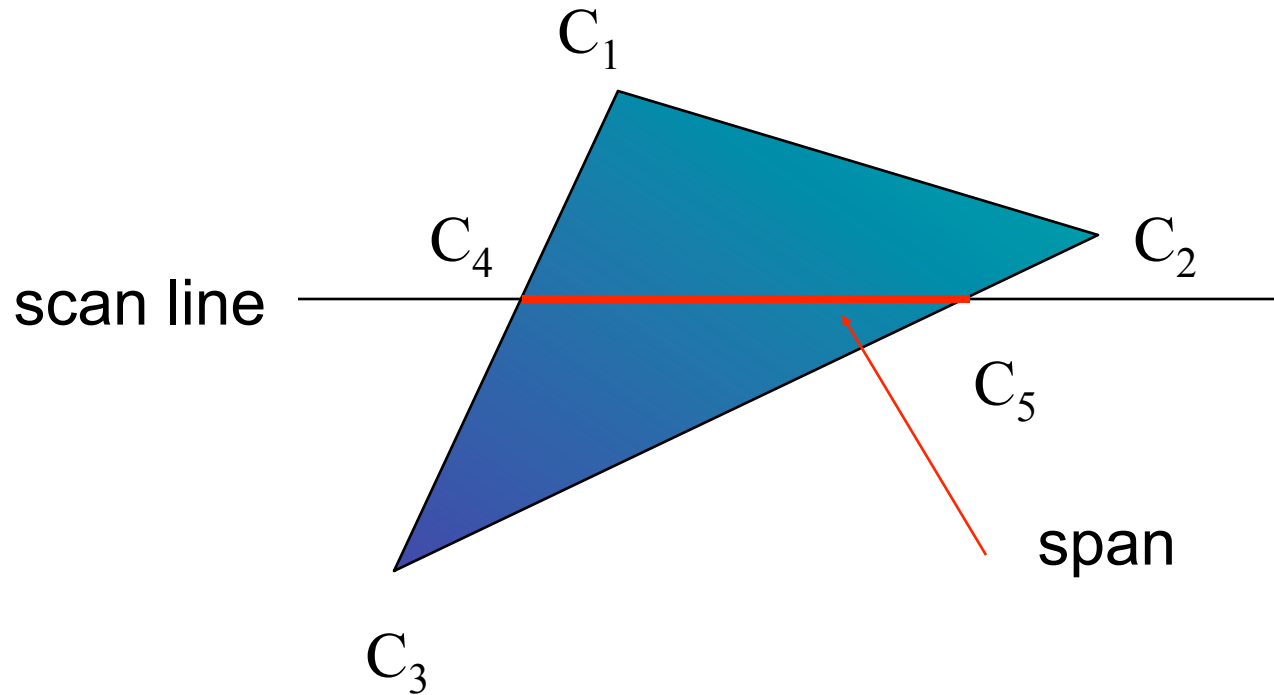   For x = xstart, xstart+1, ...,xend

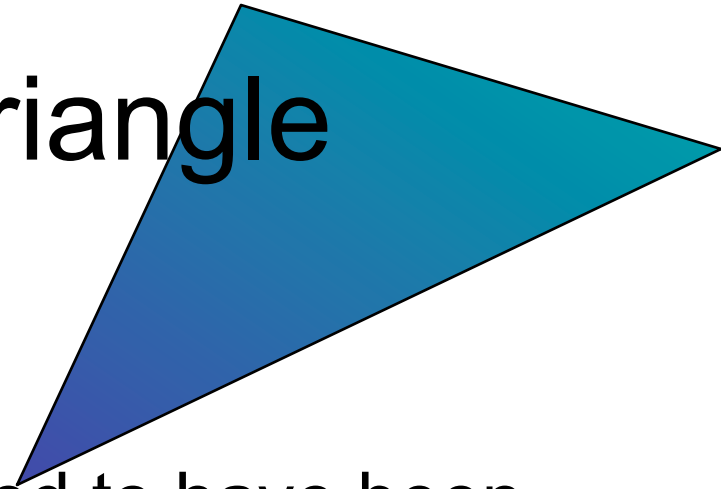   Compute color, depth etc for
(x,y) using interpolation.

**This is one modern
way to rasterize a
triangle**

Using Interpolation

$C_1\, C_2\, C_3$ specified by `glColor` or by vertex shading
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

C_1

C_4

C_2

scan line

C_5

C_3

span

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);
    } }
```

What you need to know

- Analytic test:
  - Be able to compute ray vs sphere or other formula
  - ray vs triangle
- Geometrical tests
  - Ray/box with slab-test
  - Ray/polygon (3D->2D)
  - AABB/AABB
- Other:
  - Point/plane
  - Sphere/plane
  - Box/plane, AABB/plane
- SAT
- Know what a dynamic test is
- Understand floodfill