09-0: **Spirte collision**

- Given:
  - Two 2D square images
    - (some of the pixels are transparent – not part of the actual image)
  - Two points (upper-left corner of each image)
- How can we determine if the two images intersect?

09-1: **Spirte collision**

- Brute force:
  - For each point in image A:
    - If the point is not transparent, comapre it to the corresponding point in image B
    - Need to account for translation difference!

09-2: **Spirte collision**

- Sprite A at position (x_1, y_1), width w_1, height h_1
- Sprite B at position (x_2, y_2), width w_2, height h_2

How do we determine if there is intersection between Sprite A and Sprite B, assuming a 2D array of transparancy values? Brute force is OK! 09-3: **Spirte collision**

- Sprite A at position (x_1, y_1), width w_1, height h_1
- Sprite B at position (x_2, y_2), width w_2, height h_2

```
deltax = x2 - x1; deltay = y2 - y1
for (int x = 0; x < w1; x++)
    for (y = 0; y < h1; y++)
        if (x + deltax > 0 && x + deltax < w2 &&
            y + deltay > 0 && y + deltay  h2 &&
            point at T1[x][y] not tranparent &&
            point at T2[x + deltax][y + deltay] not tranparent)
               collision!
```

09-4: **Checking Transparency**

```
Texture2D texture;
Color[] data = new Color[texture.Width * texture.Height];
texture.GetData(data);

if (data[x +  y*texture.width].A > 0)
{
    // point at texture[x,y] not completely
    // transpanent
}
```

09-5: **Spirte collision**

- Of course, this is incredibly wasteful ...
- We can be more efficient by looking at *bounding regions*
- Simple shapes that contain the sprite
- Start with bounding boxes

09-6: **AABB**

- Axis-Aligned Bounding Box
  - A Bounding box is a box that completely contains your sprite
  - An Axis aligned bounding box has its sides aligned with the global x and y axes
  - What would we need to store in memory to represent the axis aligned bounding box for an object?

09-7: **AABB**

- To represent an axis-aligned bounding box, we need two points (4 floats)
  - Minimum x, minimum y
  - Maximum x, maximum y

09-8: **AABB**

- There are other ways to represent an AABB
  - Minimum x,y and vector from min to max point
  - center of the AABB and vector to the maximum point
  - ... etc

09-9: **AABB Translation**

- We have an AABB for a model
  - How should the AABB change if the model translates?

09-10: **AABB Translation**

- We have an AABB for a model
  - How should the AABB change if the model translates?
    - Just translate the minimum and maximum point in the AABB

09-11: **AABB Rotation**

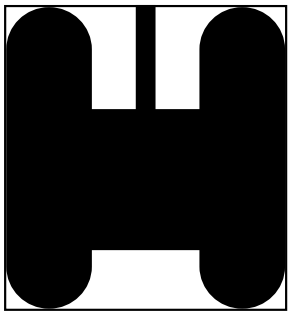
- We have an AABB for a model
  - How should the AABB change if the model rotates?

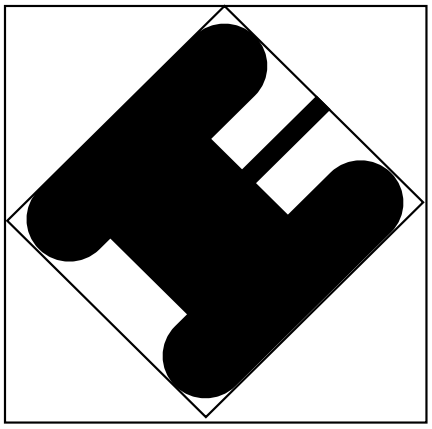
09-12: **AABB Rotation**

- We have an AABB for a model
  - How should the AABB change if the model rotates?
    - Can we just rotate the two minimum and maxiumum points? Why or why not?

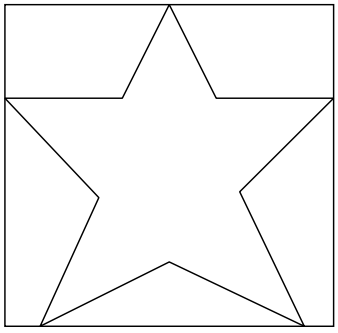
09-13: **AABB Rotation**

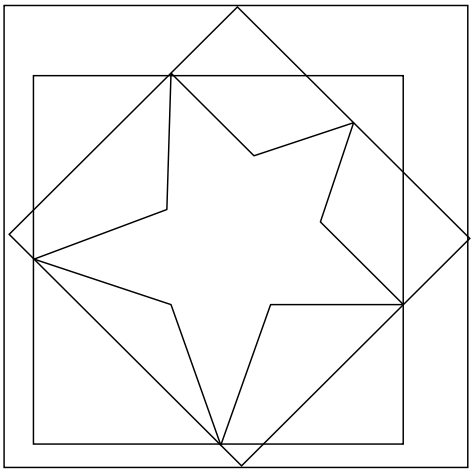
- We have an AABB for a model
  - How should the AABB change if the model rotates?
    - Rotate each of the 4 points in the AABB (why all 4)?
    - Construct a new AABB that contains these points





09-14: **AABB Rotation**





09-15: **AABB Rotation**

09-16: **AABB Rotation**

- Smallest possible AABB after rotation?

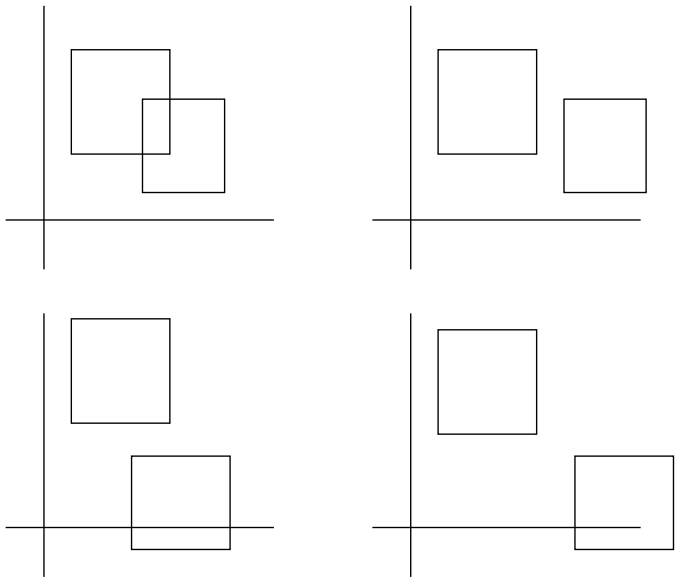
09-17: **AABB Rotation**

- Smallest possible AABB after rotation?
    - Go through all rotated points, find max / min
- Middle ground – rotate original AABB (examples)

09-18: **AABB Intersection**

- Given two AABBs, how can we determine if they intersect?

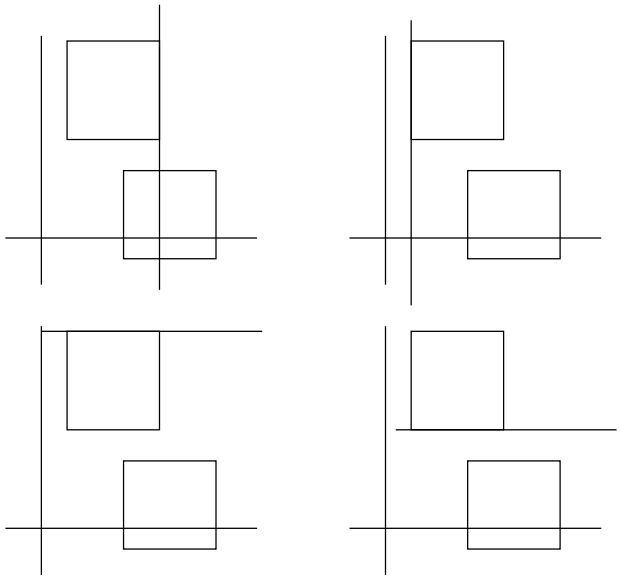
09-19: **AABB Intersection**



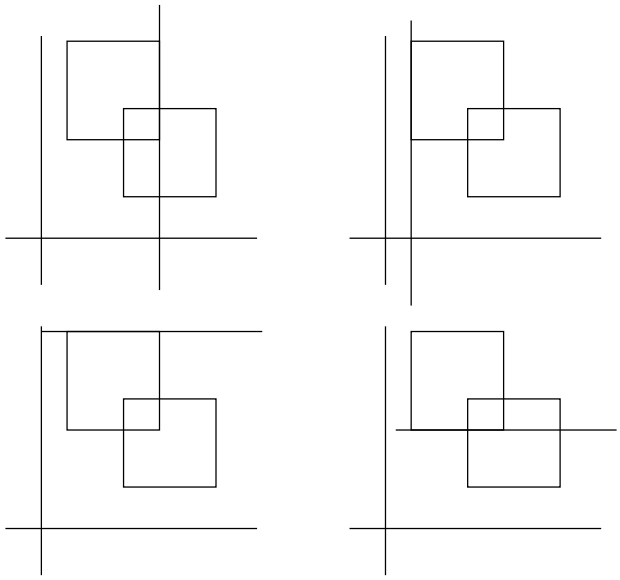
09-20: **AABB Intersection**

- Two 2D AABBs *don't* intesect if we can create a line such that one AABB is one one side of the line, and the other is on the other side of the line
- We only need to test 4 lines – the 4 sides of one of the boxes

09-21: **AABB Intersection**



09-22: **AABB Intersection**

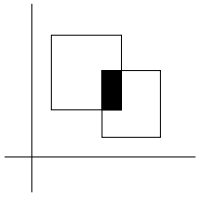


09-23: **AABB Intersection**

- Given two 2D AABBs B_1 and B_2, they *don't* intersect if:
  - $B_1.x.max < B_2.x.min$
  - $B_1.x.min > B_2.x.max$
  - $B_1.y.max < B_2.y.min$
  - $B_1.y.min > B_2.y.max$

09-24: **AABB Intersection**

- Once we know that our two AABBs intersect, we'd like to find the actual area of overlap



- Given that we know that the two AABBs overlap, how can we find the intersection?

09-25: **AABB Intersection**

- Upper left of the intersection AABB is:
- Lower right of the intersection AABB is:

09-26: **AABB Intersection**

- Upper left of the intersection AABB is:
  - Max of the mins of the two AABBs
- Lower right of the intersection AABB is:

- Min of the maxes of the two AABBs

09-27: **AABB Intersection**

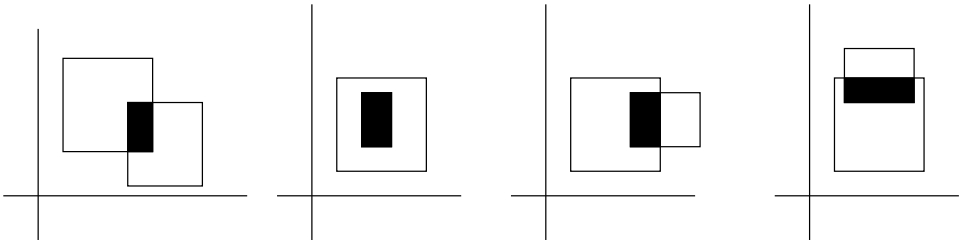
- AABB1 is $(x1_{min}, y1_{min}), (x1_{max}, y1_{max})$
- AABB2 is $(x2_{min}, y2_{min}), (x2_{max}, y2_{max})$
- AABB of intersection (assuming that they intersect) is:

09-28: **AABB Intersection**

- AABB1 is $(x1_{min}, y1_{min}), (x1_{max}, y1_{max})$
- AABB2 is $(x2_{min}, y2_{min}), (x2_{max}, y2_{max})$
- AABB of intersection (assuming that they intersect) is:
  - $(\max(x1_{min}, x2_{min}), \max(y1_{min}, y2_{min}), (\min(x1_{max}, x2_{max}), \min(y1_{max}, y2_{max}),$

09-29: **AABB Intersection**

- Sanity Check



09-30: **AABB Intersection**

- Once we have the area of the AABB intersection, we could use that area to do a pixel-by-pixel check of the objects – iterate over the intersection region instead of over one of the boxes

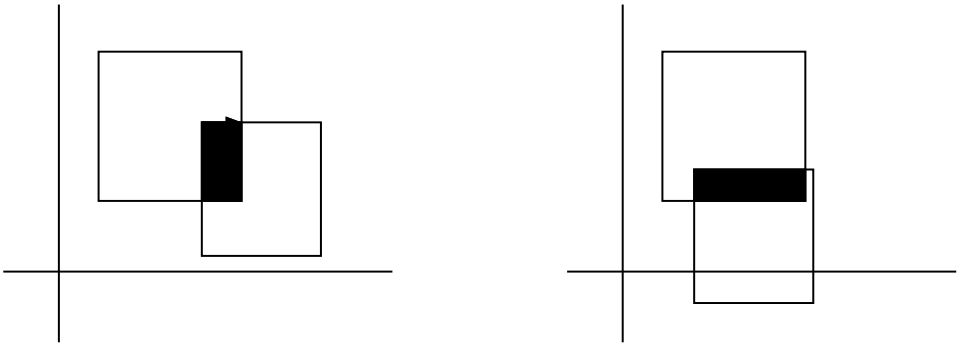
09-31: **AABB Intersection**

- Let's say that we were using AABBs only for intersection (our world objects are nice squares that don't rotate)
- Once we know that two AABBs intersect, we want to push them apart so that they no longer intersect
- We need a vector to translate one of the squares so that they no longer intersect
- **M**inimum **T**ranslation **D**istance vector, or MTD vector

09-32: **AABB MTD**

- What is the MTD for two AAABBs that intersect?

09-33: **AABB MTD**



09-34: **AABB MTD**

- What is the MTD for two AAABBs that intersect?
  - Minimum dimension of AABB intersection

09-35: **OOBB Defintion**

- Axis-aligned bounding boxes are good as a first step
  - Computing insersections is fast
  - Actual intersection =¿ AABB intersection
- AABBs are not great for final collision detection – not accurate enough
- We can do a little better with oriented bounding boxes

09-36: **OOBB Defintion**

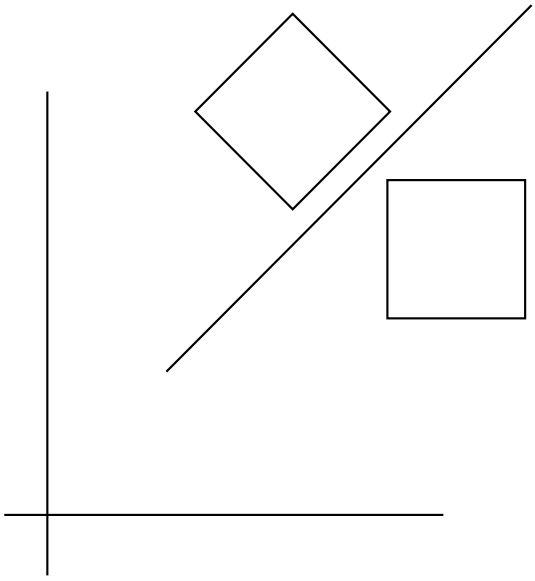
- We will define an OOBB using 4 points
  - Points of each corner of the box
- Our techniques will work for any convex polygon, not just boxes

09-37: **OOBB Intersection**

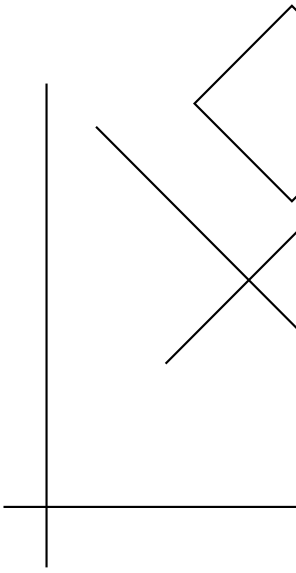
- We can define intersection of OOBB similarly to intersection of AABB:
  - If a line can be inserted between two OOBBs, such that one object is on one side of the line, and the other object is on the other side of the line, the boxes do not intersect
  - If no such line exists, then the boxes do intersect

09-38: **OOBB Intersection**

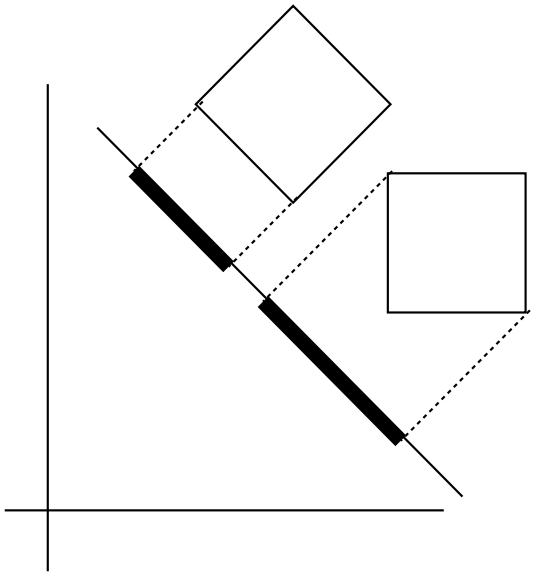
- Let’s look at another way doing the same thing ...
  - Pick any line you like (axis of separation)
  - Project both boxes onto this axis
  - If the projections don’t overlap, no intersection



09-39: **OOBB Intersection**



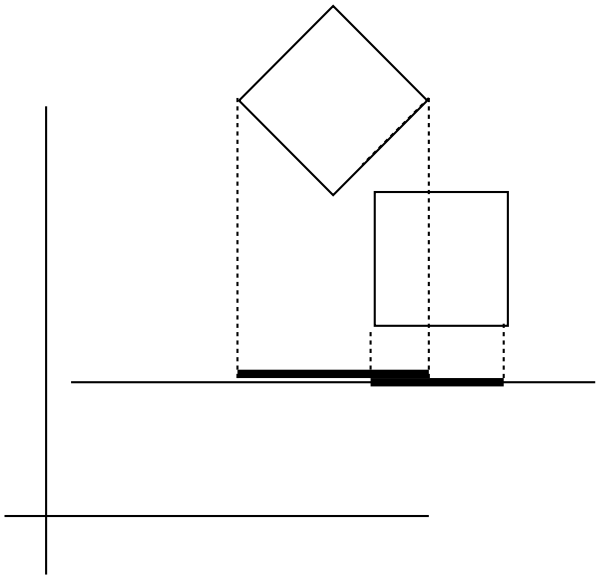
09-40: **OOBB Intersection**



09-41: **OOBB Intersection**

09-42: **OOBB Intersection**

- If there is some axis of separation with no overlap, no intersection
- However, if there is an axis of separation with overlap, may not be any intersection
- Need to pick the correct one!



09-43: **OOBB Intersection**

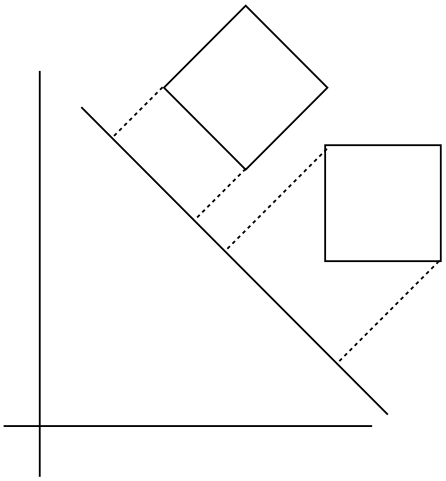
09-44: **OOBB Intersection**

- There are a infinite number of axes to try!
- Can't try them all!
- If there is an axis of separation, then it must be ...

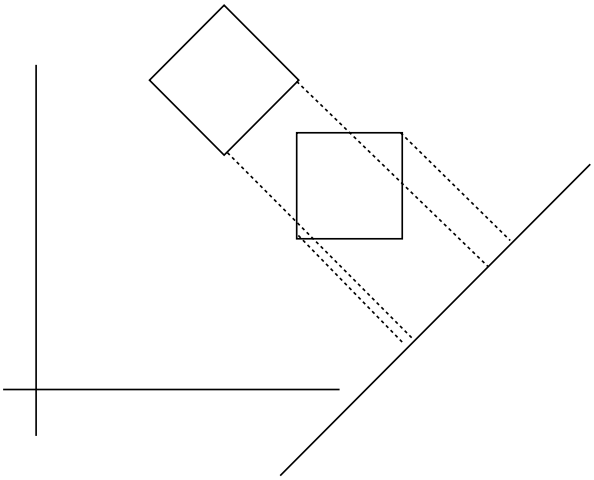
09-45: **OOBB Intersection**

- There are a infinite number of axes to try!
- Can't try them all!
- If there is an axis of separation, then it must be
  - Perpendicular to one of the edges
  - There are only 4 possibilities if we are working with OOBBs

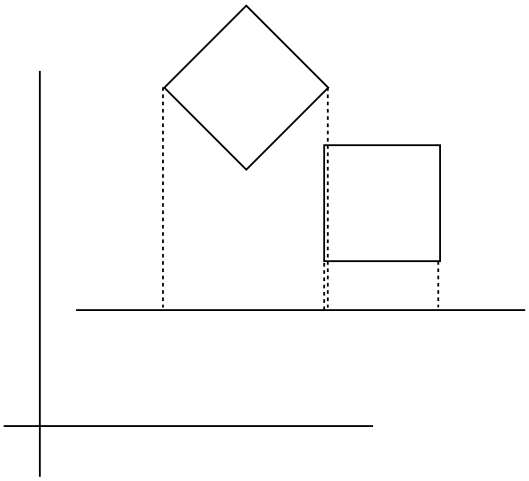
09-46: **OOBB Intersection**



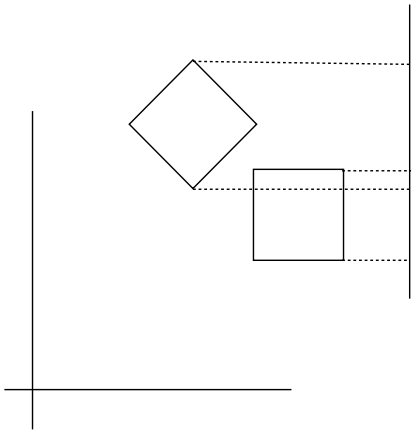
09-47: **OOBB Intersection**



09-48: **OOBB Intersection**



09-49: **OOBB Intersection**



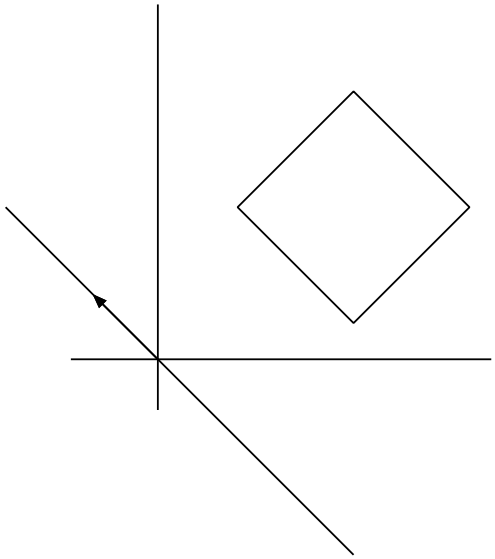
09-50: **OOBB Intersection**

- Pick an axis of separation (will need to try 4 in all)
- Project boxes onto axis
- See if there is overlap
- If there is no overlap on any axis, we can stop – no intersection

09-51: **OOBB Intersection**

- How do we project a box onto our axis?
  - Project each point onto axis
  - Find the maximum and minimum values

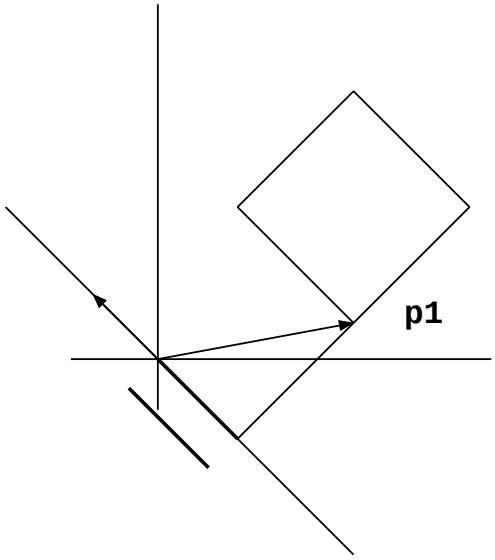
09-52: **OOBB Intersection**



Box: p1, p2, p3, p4

Axis: [x,y]
(normalized)

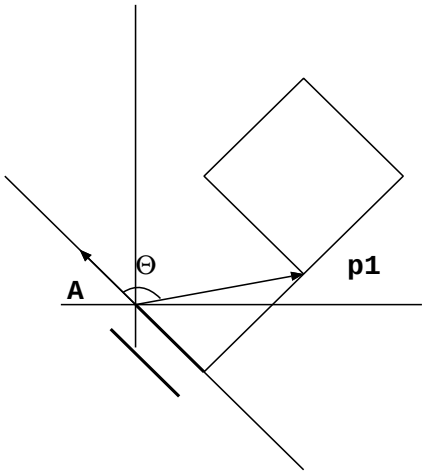
09-53: **OOBB Intersection**



Box: p1, p2, p3, p4

Axis: [x,y]
(normalized)

09-54: **OOBB Intersection**



Box: p1, p2, p3, p4

Axis: [x,y]
(normalized)

A * p1 = ||A|| ||p1|| cos Θ
= ||p1|| cos θ
= length of projection

09-55: **OOBB Intersection**

- Given a box p_1, p_2, p_3, p_4, what are the two axes of intersection?

09-56: **OOBB Intersection**

- Given a box p_1, p_2, p_3, p_4, what are the two axes of intersection?
  - Perpendicular to an edge
  - Vector parallel to an edge: $v = p_2 - p_1$
  - Vector perpendicular to $v = [v_x, v_y]$?

09-57: **OOBB Intersection**

- Given a box p_1, p_2, p_3, p_4, what are the two axes of intersection?
  - Perpendicular to an edge
  - Vector parallel to an edge: $v = p_2 - p_1$
  - Vector perpendicular to $v = [v_x, v_y]$?
    - $[-v_y, v_x]$
  - Axis of intersection: $\frac{[-v_y, v_x]}{\|[-v_y, v_x]\|}$

09-58: **OOBB Intersection**

- Given two boxes $(p_{11}, p_{12}, p_{13}, p_{14})$ and $(p_{21}, p_{22}, p_{23}, p_{24})$
  - $v_{\parallel} = p_{12} - p_{11}$
  - $a = \frac{[-v_{\parallel_y}, v_{\parallel_x}]}{\|[-v_{\parallel_y}, v_{\parallel_x}]\|}$
  - Calculate $a \cdot p_{11}$, $a \cdot p_{12}$,$a \cdot p_{13}$, $a \cdot p_{14}$, store minimum and maximum values
  - Calculate $a \cdot p_{21}$, $a \cdot p_{22}$, $a \cdot p_{23}$, $a \cdot p_{24}$, store minimum and maximum values
  - Check for overlap
  - Repeat for $p_{13} - p_{12}$, $p_{22} - p_{21}$, $p_{23} - p_{22}$

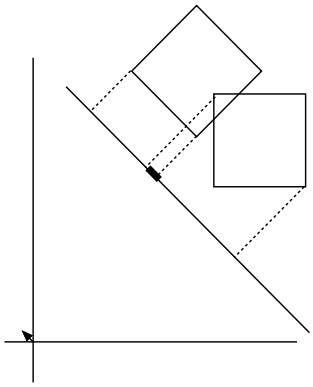
09-59: **OOBB MTD**

- How can we calculate the Minimum Translation Distance vector?

09-60: **OOBB MTD**

- How can we calculate the Minimum Translation Distance vector?
  - Find the separating axis with the minimum overlap
  - Vector is along the separating axis, length equal to the overlap

09-61: **OOBB MTD**

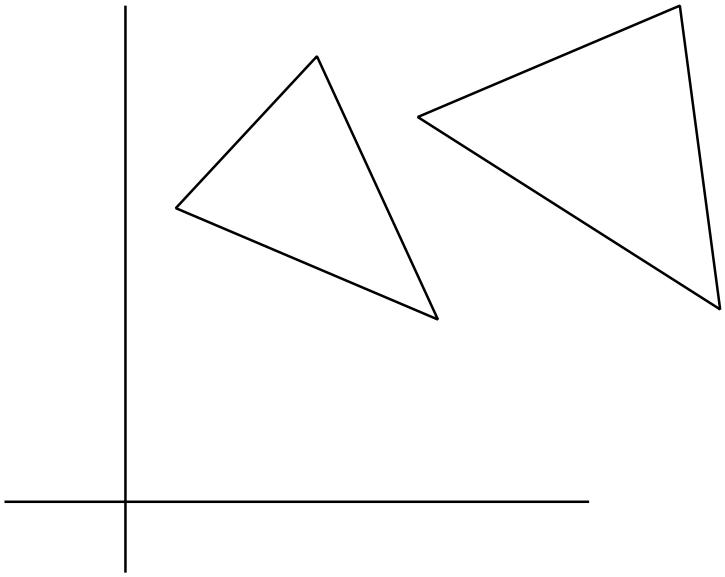


- We've already calculated the separating axis vector, and the amount of offset – multiply the offset size (scalar) by separating axis vector (already normalized)

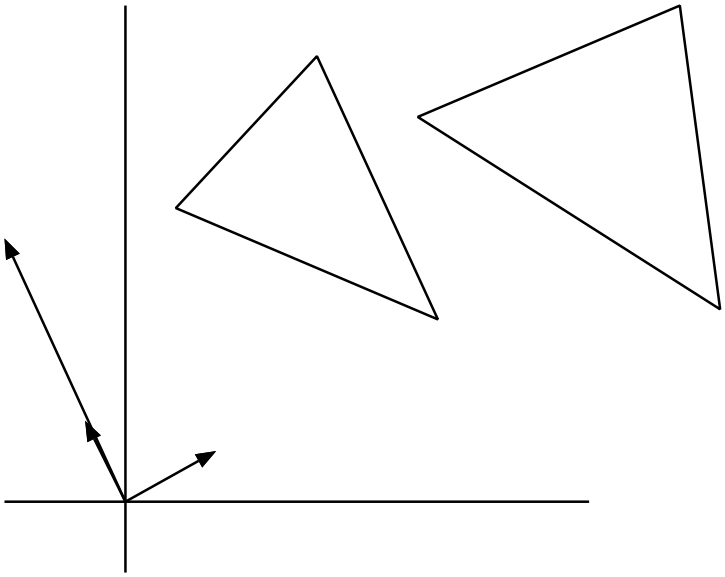
09-62: **General Intersection**

- This method works for any convex surface:
  - For axes of separation, try vector perpendicular to each edge

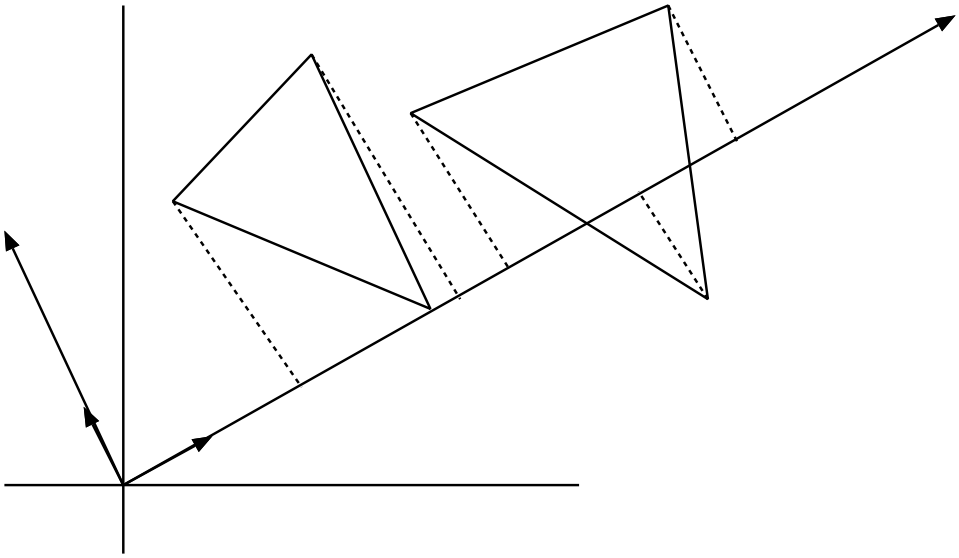
09-63: **General Intersection**



09-64: **General Intersection**

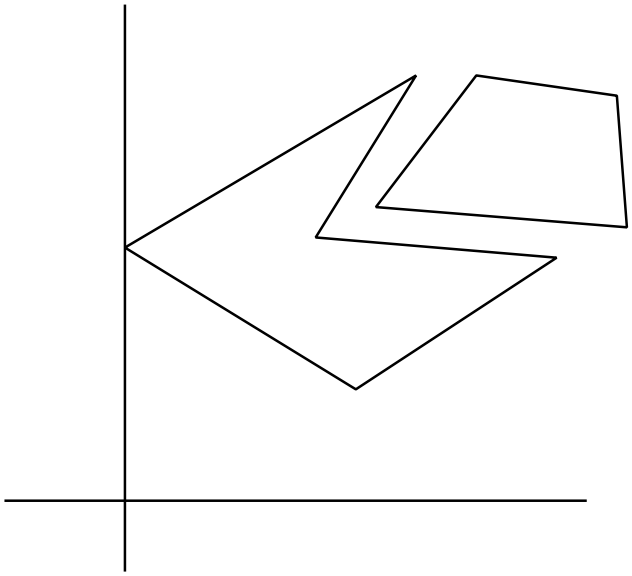


09-65: **General Intersection**

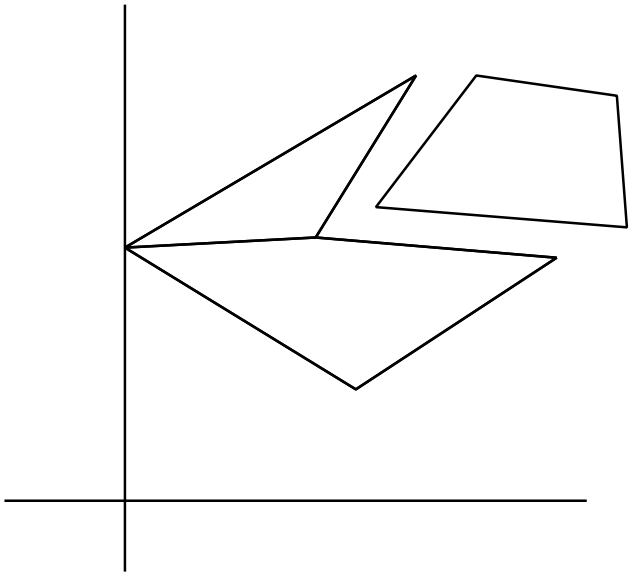


09-66: **General Intersection**

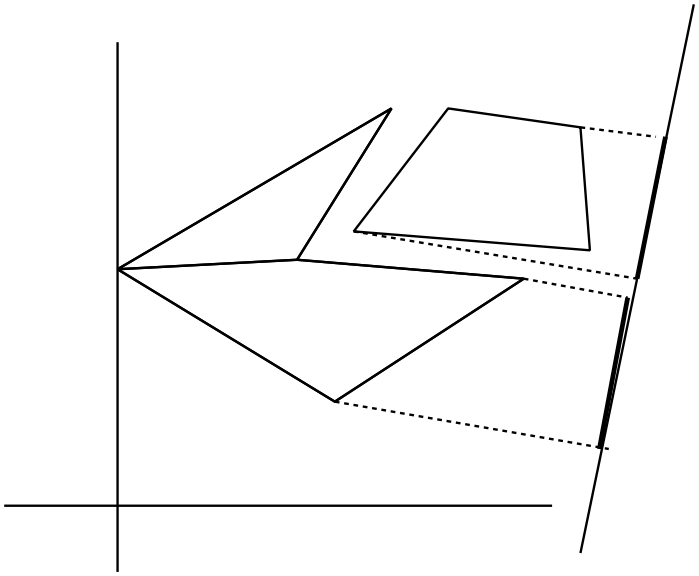
- Only works for convex objects, not concave



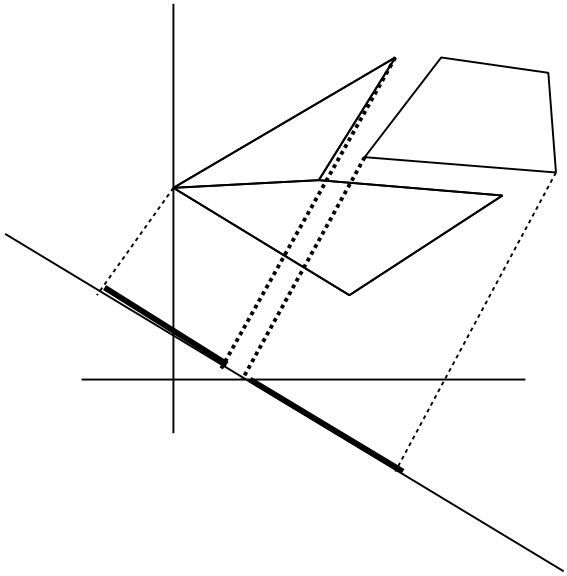
09-67: **General Intersection**



09-68: **General Intersection**



09-69: **General Intersection**



09-70: **More Intersection!**

- Bounding Circles
  - Circle represented as a center point $c = [c_x, c_y]$ and a radius r
  - Given c_1, r_1, c_2, r_2. How can we determine if two circles have intersected?

09-71: **Bounding Circles**

- Given c_1, r_1, c_2, r_2. How can we determine if two circles have intersected?
  - $||c_1 - c_2|| < r_1 + r_2$
  - $||c_1 - c_2||^2 < (r_1 + r_2)^2$
- What is the Minimum Translation Distance (MTD) vector?

09-72: **Bounding Circles**

- Given c_1, r_1, c_2, r_2. How can we determine if two circles have intersected?
  - $||c_1 - c_2|| < r_1 + r_2$
- What is the Minimum Translation Distance (MTD) vector?
  - $\frac{c_1 - c_2}{||c_1 - c_2||}(r_1 + r_2 - ||c_1 - c_2||)$

09-73: **Line Intersection**

- Given two lines
  - $A_1x + B_1y = C_1$
  - $A_2x + B_2y = C_2$
- How do we determine their point of intersection?

09-74: **Line Intersection**

- Solve for y in equation 1

$$\begin{aligned} A_1x + B_1y &= C_1 \\ y &= \frac{C_1 - A_1x}{B_1} \end{aligned}$$

- Substitute for y in equation 2:

$$\begin{aligned} A_2x + B_2\frac{C_1 - A_1x}{B_1} &= C_2 \\ x(A_2 - \frac{B_2A_1}{B_1}) &= C_2 - \frac{B_2C_1}{B_1} \\ x &= \frac{C_2B_1 - B_2C_1}{A_2B_1 - B_2A_1} \end{aligned}$$

09-75: **Line Intersection**

$$x = \frac{C_2B_1 - B_2C_1}{A_2B_1 - B_2A_1}$$

- Do the same thing for y: (solve for x in equation 1, and then substitute into equation 2)

$$y = \frac{C_1A_2 - A_1C_2}{A_2B_1 - B_2A_1}$$

- What does it mean if $A_2B_1 - B_2A_1 = 0$?

09-76: **Line Segment Intersection**

- Given two segments: $p_{11}, p_{12}, p_{21}, p_{22}$, how do we determine if they intersect?

09-77: **Line Segment Intersection**

- Given two segments: $p_{11}, p_{12}, p_{21}, p_{22}$, how do we determine if they intersect?
  - Given two points $(x_1, y_1), (x_2, y_2)$, we can calculate A, B, C:
    - $A = y_2 - y_1$
    - $B = x_1 - x_2$
    - $C = Ax_1 + By_1$ (or $C = Ax_2 + By_2$)
- Once we have $A_1, A_2, B_1, B_2, C_1, C_2$, we can determine where the two lines defined by the two segments intersect
- What next?

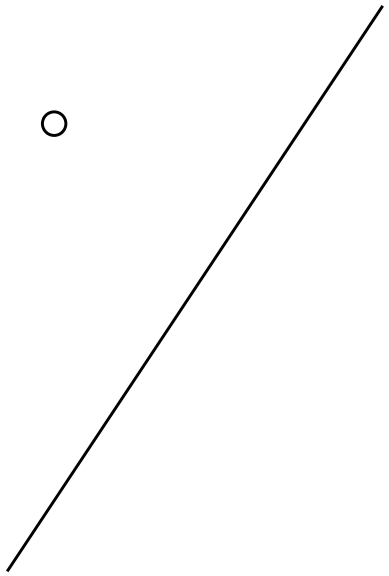
09-78: **Line Segment Intersection**

- Given two segments $p_{11}, p_{12}, p_{21}, p_{22}$, and the point x, y where the lines defined by each segment intersect:
  - Check to see if x, y is within the range of each segment
    - $x >= p_{11_x}$ and $x <= p_{12_x}$ (or $x >= p_{12_x}$ and $x <= p_{11_x}$)
    - $y >= p_{11_y}$ and $y <= p_{12_y}$ (or $y >= p_{12_y}$ and $y <= p_{11_y}$)
    - $x >= p_{21_x}$ and $x <= p_{22_x}$ (or $x >= p_{22_x}$ and $x <= p_{21_x}$)
    - $y >= p_{21_y}$ and $y <= p_{22_y}$ (or $y >= p_{22_y}$ and $y <= p_{21_y}$)

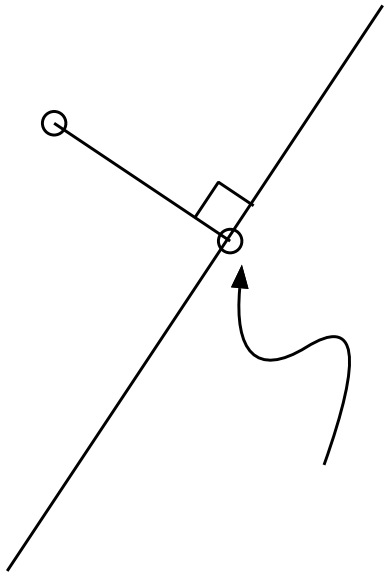
09-79: **Closest Point on Line**

- Given a point p and a line L (defined by two points p_1, p_2), how can we determine the closest point to p along L?

09-80: **Closest Point on Line**



09-81: **Closest Point on Line**



09-82: **Closest Point on Line**

- Given a point p and a line L (defined by two points p_1, p_2), how can we determine the closest point to p along L?
  - Create a line $L_\perp$ that is perpendicular to L and goes through p
  - Find the intersection of L and $L_\perp$

09-83: **Closest Point on Line**

- $L : A_1x + B_1y = C$
- $L_\perp : -B_1x + A_1y + C_2$
  - $C_2 = -B_1p_x + A_1p_y$
  - (plug the known value of the point into the formula, and solve for C_2)

09-84: **Closest Point on Segment**

- Given a line segment defined by points p_1, p_2 and a point p, how can we find the point closest to p along the segment?

09-85: **Closest Point on Segment**

- Given a line segment defined by points p_1, p_2 and a point p, how can we find the point closest to p along the segment?
  - If the closest point to the line defined by the segment is in the segment, that is the correct point
  - Otherwise, endpoint of segment closest to intersection point

09-86: **Segment/Circle**

- How can we determine of a line segment and a circle intersect?

09-87: **Segment/Circle**

- How can we determine of a line segment and a circle intersect?
  - Find the point $\mathbf{p}$ on the segment closest to the center of the circle
  - Check to see if the distance from $\mathbf{p}$ to the center of the circle $\mathbf{c}$ is $<$ radius r
  - How can we calculate the MTD vector?

09-88: **Segment/Circle**

- How can we determine of a line segment and a circle intersect?
  - Find the point $\mathbf{p}$ on the segment closest to the center of the circle
  - Check to see if the distance from $\mathbf{p}$ to the center of the circle $\mathbf{c}$ is $<$ radius r
  - How can we calculate the MTD vector?
    - Direction of vector: $\mathbf{p} - \mathbf{c}$
    - Length of MTD: $r - ||\mathbf{p} - \mathbf{v}||$

09-89: **Polygon / Circle**

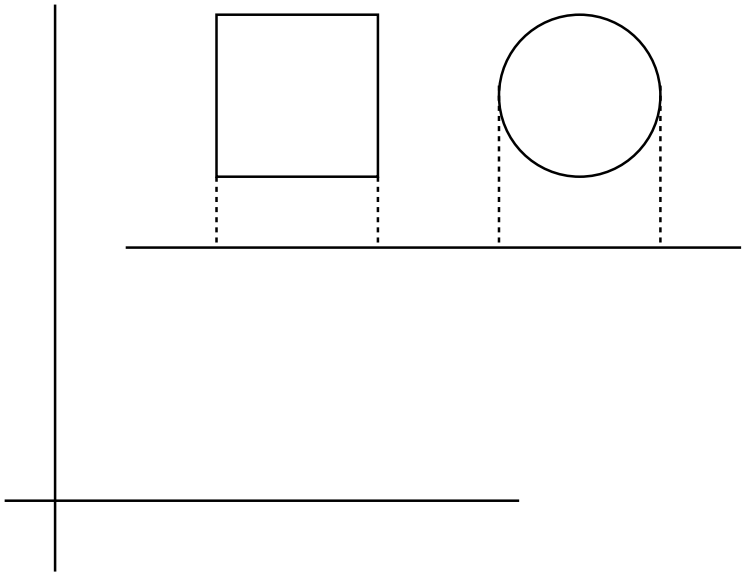
- How can we determine if a polygon and a circle intersect?

09-90: **Polygon / Circle**

- How can we determine if a polygon and a circle intersect?
  - For each segment in the polygon, check for intersection of polygon and line segment
  - Find the point p closest to the center c of the circle from all segments in the polygon
  - MTD:
    - Direction of MTD vector: $\mathbf{p} - \mathbf{c}$
    - Length of MTD: $r - ||\mathbf{p} - \mathbf{v}||$

09-91: **Polygon / Circle**

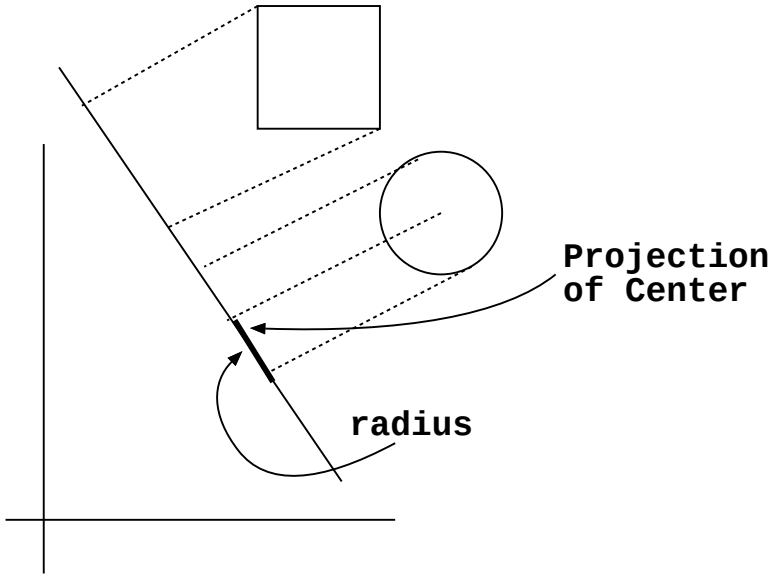
- We can also use Axis of separation to determine if polygon and circle intersect
- Once we have a candidate axis of separation, how can we project the circle onto it?



09-92: **Polygon / Circle**

09-93: **Polygon / Circle**

- Once we have a candidate axis of separation:
  - Project the center of the circle to the axis
  - Projection extends r units to either side of this point
- How do we find candidate axis of separation?



09-94: **Polygon / Circle**

09-95: **Polygon / Circle**

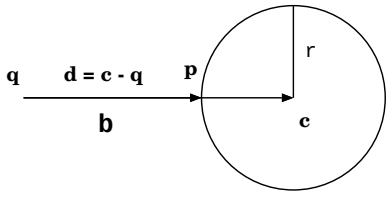
- How do we find candidate axis of separation?
  - Lines perpendicular to the sides of the polygon
  - Will those always work?

09-96: **Polygon / Circle**

- How do we find candidate axis of separation?
  - Lines perpendicular to the sides of the polygon
  - Line from the center of the circle to the closest point on the polygon to the circle

09-97: **Closest point on a Circle**

- Given a circle ($\mathbf{c}$, r) and a point $\mathbf{q}$, what is the point on the circle closest to q?



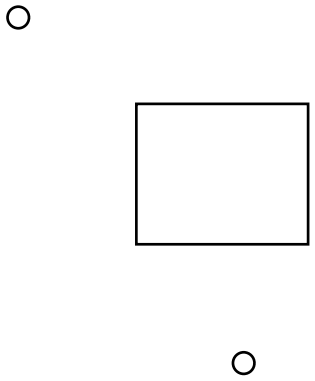
09-98: **Closest point on a Circle**

$$
\begin{aligned}
b &= \frac{||d|| - r}{||d||} d \\
p &= q + \frac{||d|| - r}{||d||} d
\end{aligned}
$$

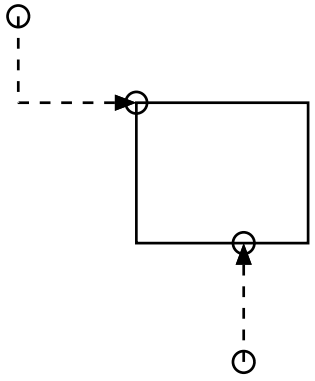
09-99: **Closest point in AABB**

- Given
  - AABB (minX, minY, maxX, maxY)
  - point q
- What is the point closest to q that is inside the AABB?

09-100: **Closest point in AABB**



09-101: **Closest point in AABB**



- "Push in" the point along each axis

09-102: **Closest point in AABB**

```
if (q.x < minX)
   p.x = minX
else if (q.x > maxX)
   p.x = maxX
else
   p.x = q.x
if (q.y < minY)
   p.y = minY
else if (q.y > maxY)
   p.y = maxY
else
   p.y = q.y
```

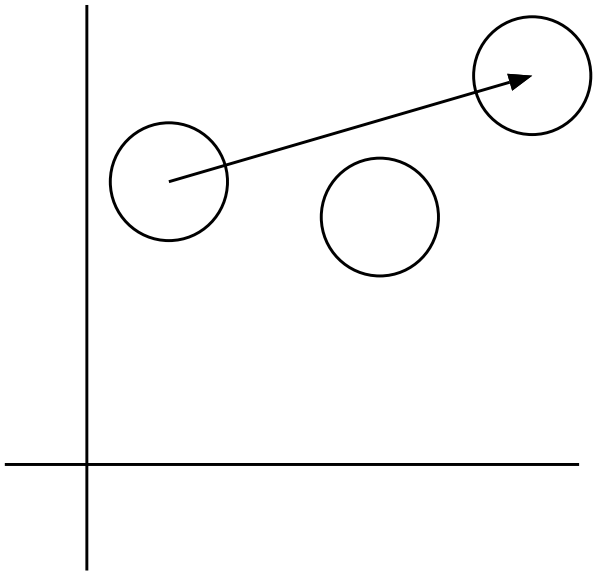
09-103: **Dynmamic Intersection**

- Doing only static intersections will miss intersections that happen between frames
- Small, fast-moving objects (like bullets) can penetrate thin objects (like walls)
- We can model a small object like a bullet using a line / line segment
    - Already know how to do intersection wtih line segments

09-104: **Dynamic Circle Intersection**

- Two circles, one moving and one static
    - Did the circles intersect between the frame?

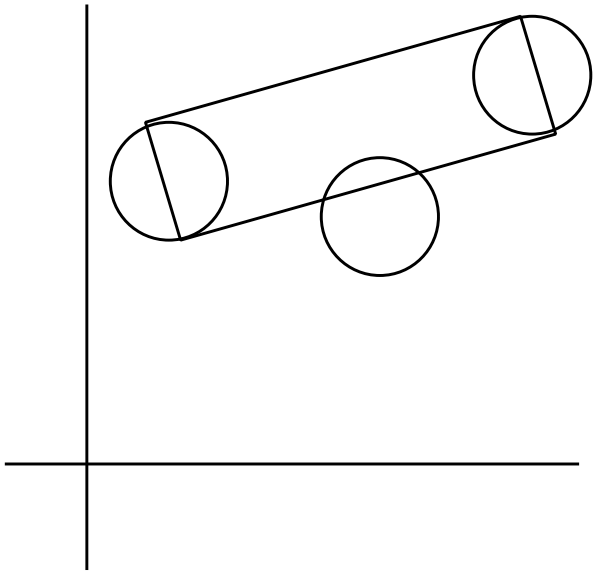
09-105: **Dynamic Circle Intersection**



09-106: **Dynamic Circle Intersection**

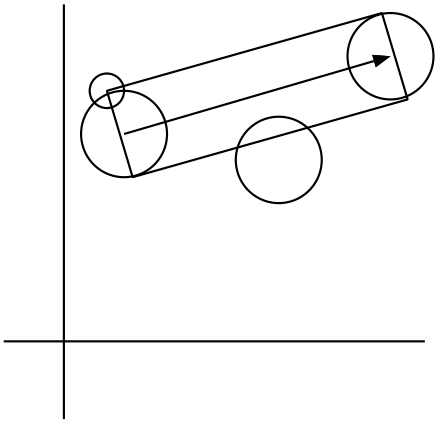
- Method 1:
    - Sweep the moving circle
    - Create a rectangle and two circles
    - Do interesction as before

09-107: **Dynamic Circle Intersection**

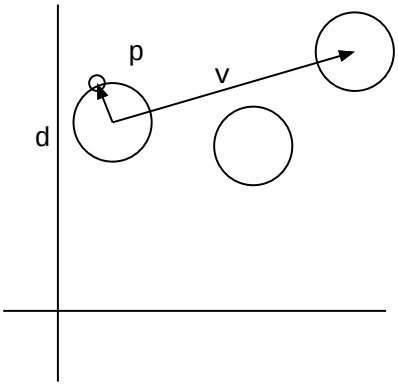


09-108: **Dynamic Circle Intersection**

- How do we find the vertices?



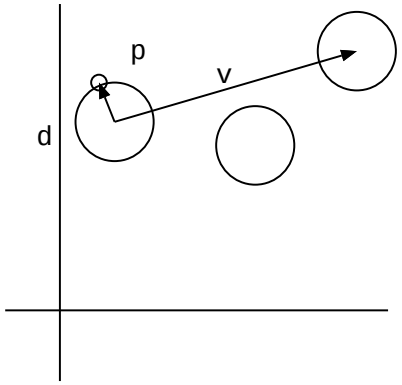
09-109: **Dynamic Circle Intersection**



- c = center of circle
- $v = [v_x, v_y]$ velocity vector

- r = radius of circle

09-110: **Dynamic Circle Intersection**

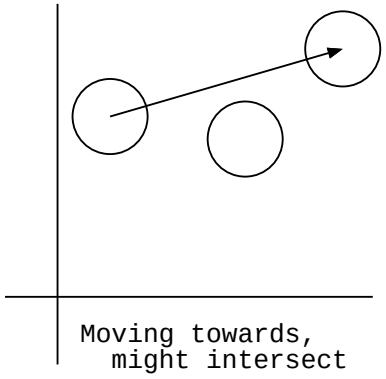


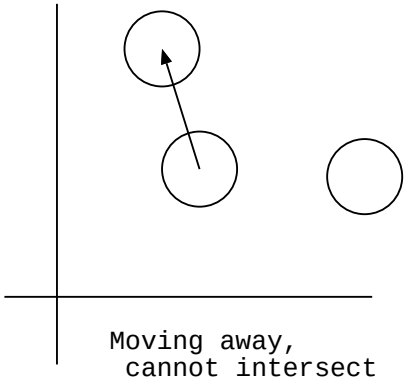
- $d = \frac{[-v_x, v_y]r}{||v||}$
- $p = c + d$

09-111: **Dynamic Circle Intersection**

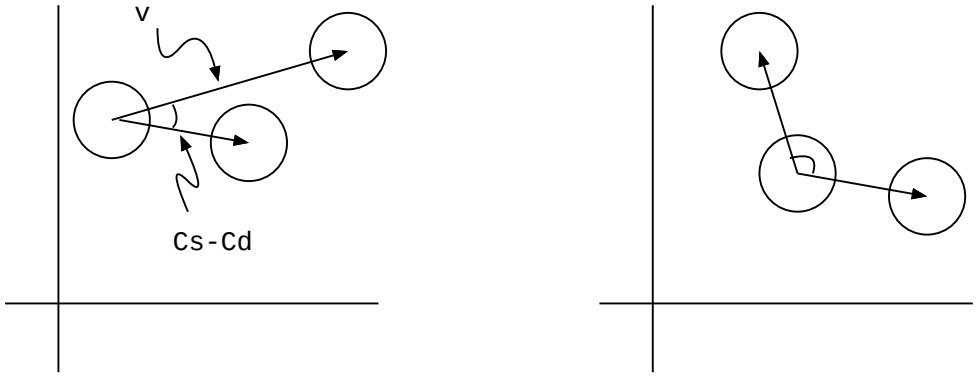
- Method #2:
  - 3 Step Process
    - Is the moving circle even moving towards the stationary circle?
    - If the moving circle continued along its course, would it eventually interect with the stationary circle
    - Will the moving circle actually intersect with the stationary circle?

09-112: **Dynamic Circle Intersection**





09-113: **Dynamic Circle Intersection**



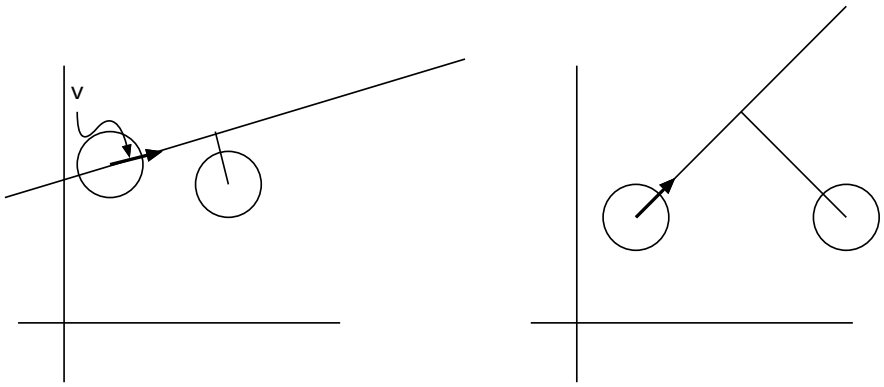
09-114: **Dynamic Circle Intersection**

- Angle bewteen velocity and line between centerpoints > 90
- $(c_{static} - c_{dynamic}) \cdot v < 0$
- Circles moving apart, can't intersect

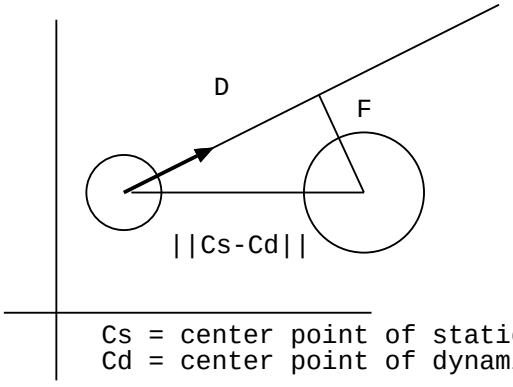
09-115: **Dynamic Circle Intersection**

- If the dynamic circle is moving towards the static circle
  - Find the closest point to the static circle along the movement vector
  - See if this point is $> r_d + r_s$

09-116: **Dynamic Circle Intersection**



09-117: **Dynamic Circle Intersection**




```
Cs = center point of static circle
Cd = center point of dynamic circle
```

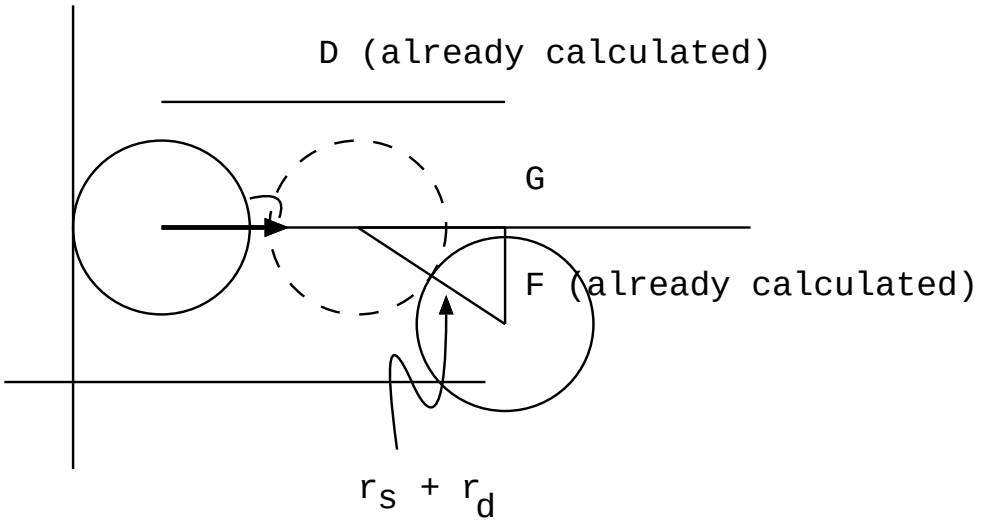
09-118: **Dynamic Circle Intersection**

- $D = \frac{v}{||v||} \cdot (C_s - C_d)$
- $F^2 = ||C_s - c_d||^2 - D^2$
- $G^2 = (r_s + r_d)^2 - F^2$
- Check to see if $||v|| > D - G$

09-119: **Dynamic Circle Intersection**

- Dynamic circle is moving toward static circle
- Might intersect, if dynamic circle moves far enough
- How can we determine the exact position of first intersection?

09-120: **Dynamic Circle Intersection**



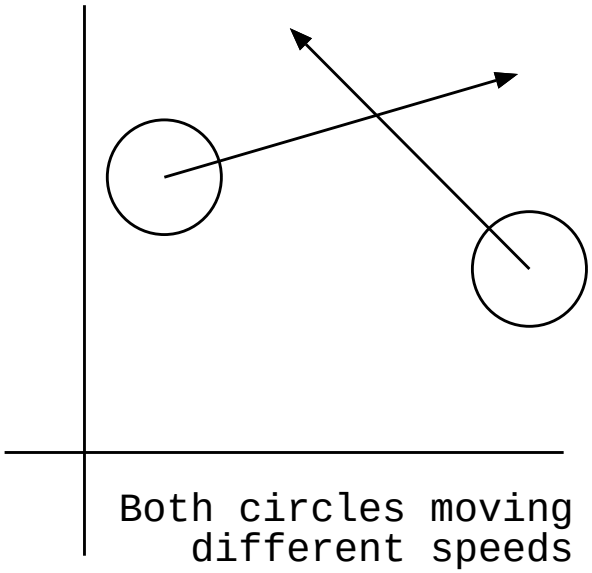
09-121: **Dynamic Circle Intersection**

- $D = \frac{v}{||V||} \cdot (C_s - C_d)$
- $||C_s - c_d||^2 - D^2 = F^2$
- Check to see if $F^2 > (r_s + r_d)^2$

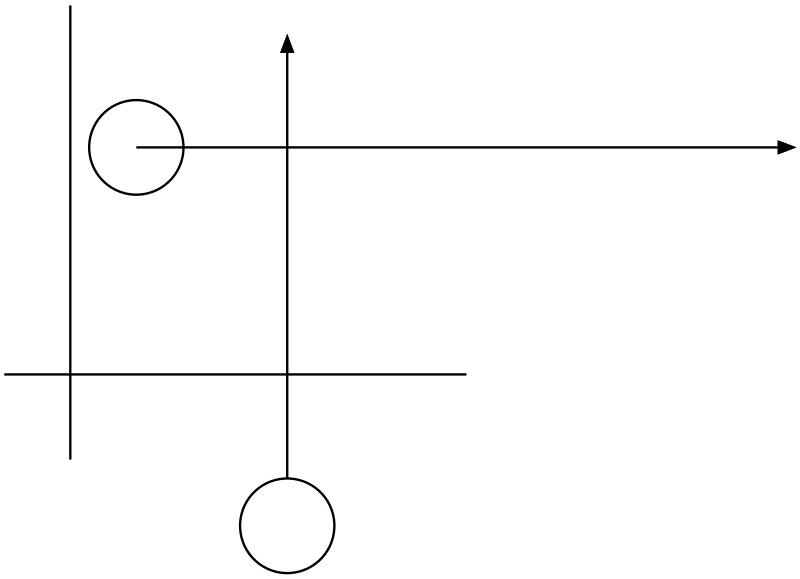
09-122: **Dynamic Circle Intersection**

- $D = \frac{v}{||V||} \cdot (C_s - C_d)$
- $||C_s - c_d||^2 - D^2 = F^2$
- $G^2 = (r_s + r_d)^2 - F^2$
- $G^2 = (r_s + r_d)^2 - (||C_s - c_d||^2 - D^2)$

09-123: **Dynamic Circle Intersection**



09-124: **Dynamic Circle Intersection**



09-125: **Dynamic Circle Intersection**

- Can’t sweep both circles
- Math we did a moment ago gets really hard if both circles are moving
- ... but there is an easy solution, reuse everything we’ve just done

09-126: **Dynamic Circle Intersection**

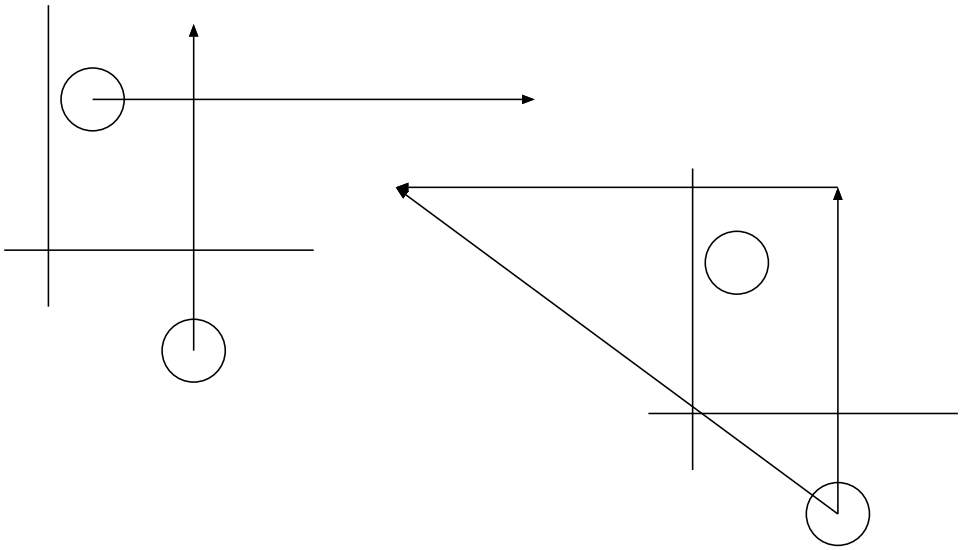
- Changing reference frames to the rescue!
- Pretend that Cirle 2 is not moving at all

- Velocity of Circle 1 in Cirlce 2's frame of reference:

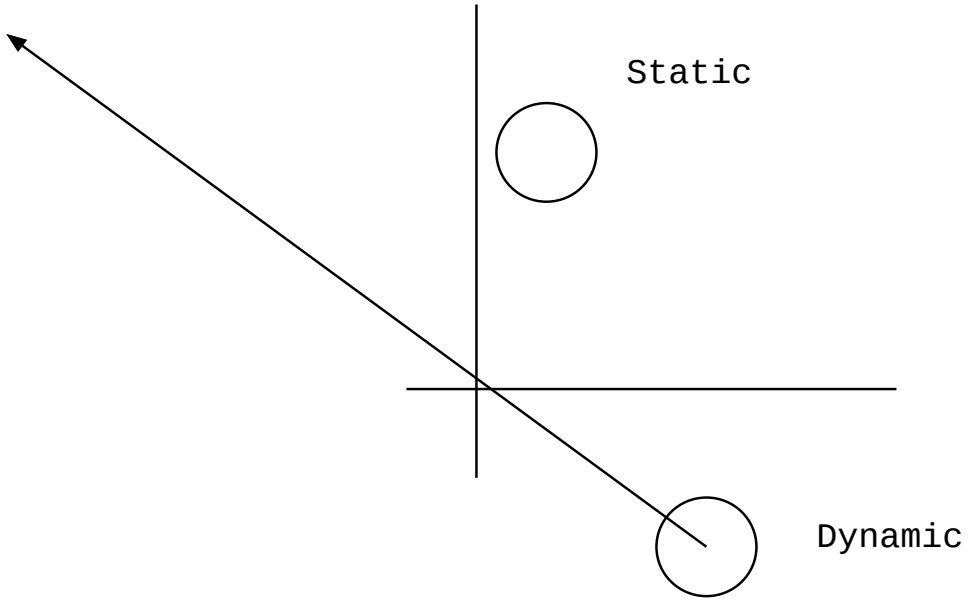
09-127: **Dynamic Circle Intersection**

- Changing reference frames to the rescue!
- Pretend that Cirle 2 is not moving at all
- Velocity of Circle 1 in Cirlce 2's frame of reference:
  - $v_1 - v_2$

09-128: **Dynamic Circle Intersection**



09-129: **Dynamic Circle Intersection**



09-130: **Dynamic Circle Intersection**