Game Programming II

Vectors, Matrices & Collision Detection
- in 2D

Vectors

What is a Vector?

What is a Vector?

- [1, 6, 4, 3, …]

- A vector is an one dimensional array

What is a Vector?

- [1, 6]
- [X, Y]

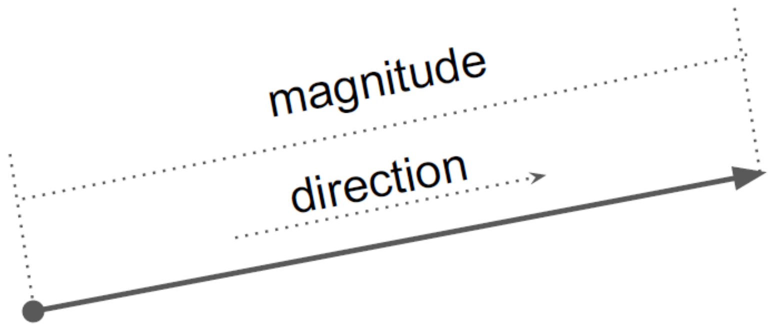
- In 2D, a 1-dimensional array with length 2
- Normally first and second element are named X & Y

What is a Vector?

- Struct Vector {
  float x;
  float y;
  }

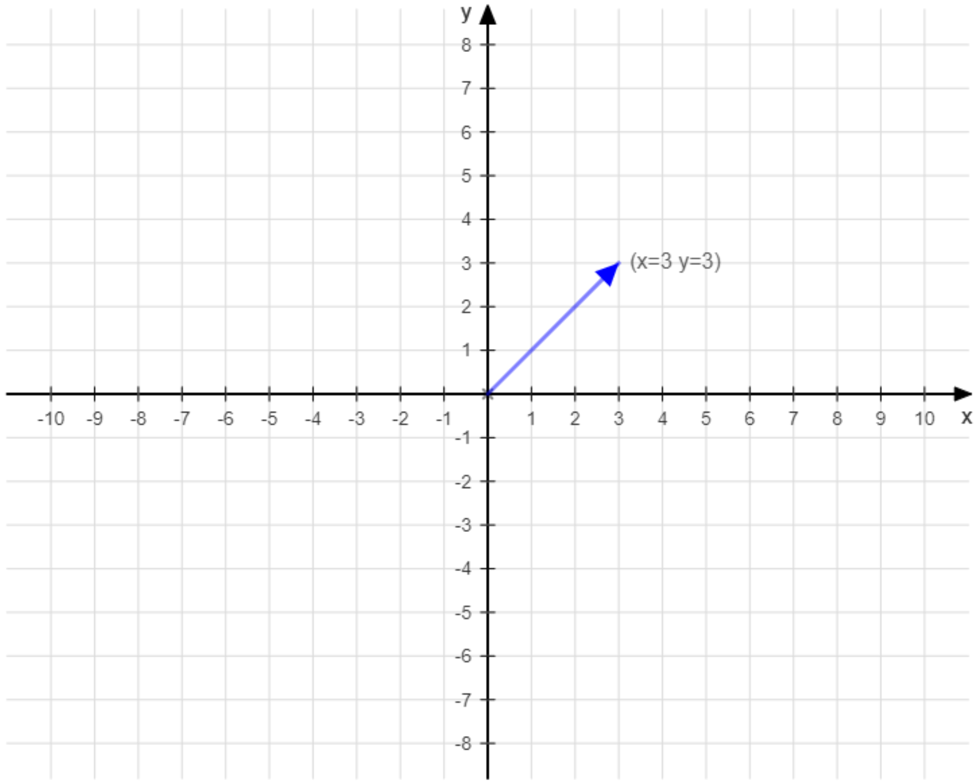
- Here defined as a struct with variables X & Y

What is a Vector?



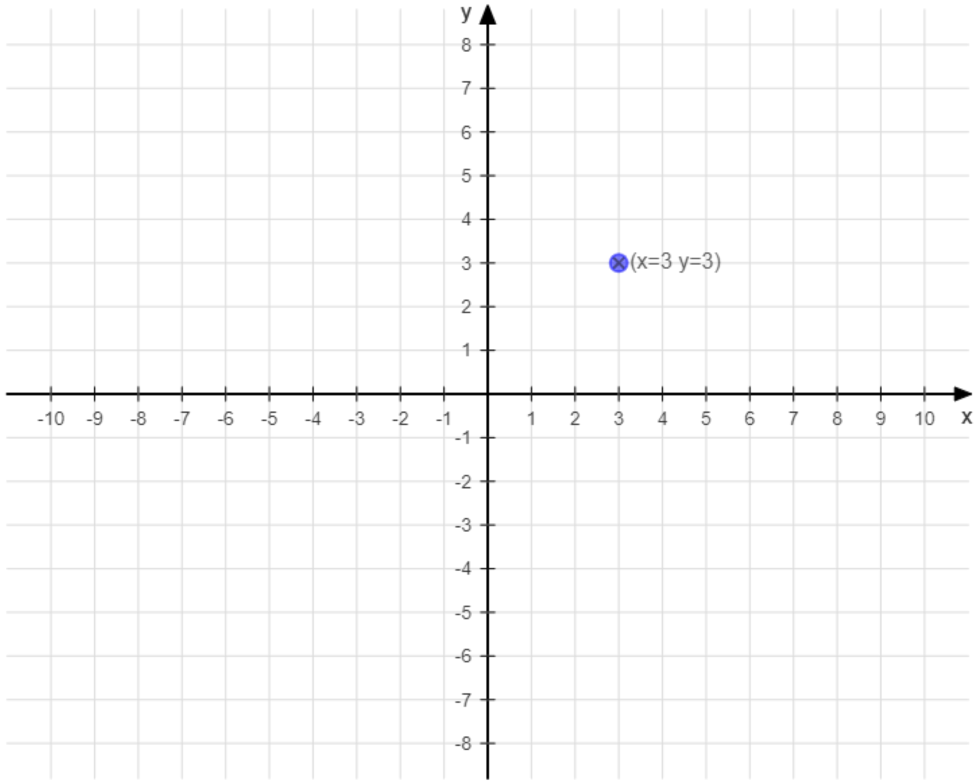
› A vector is a direction and a magnitude.

What is a Vector?



- A vector is a direction and a magnitude.

What is a Vector?



- A vector is a position in a grid.

What is a Vector?

- Struct Bird {

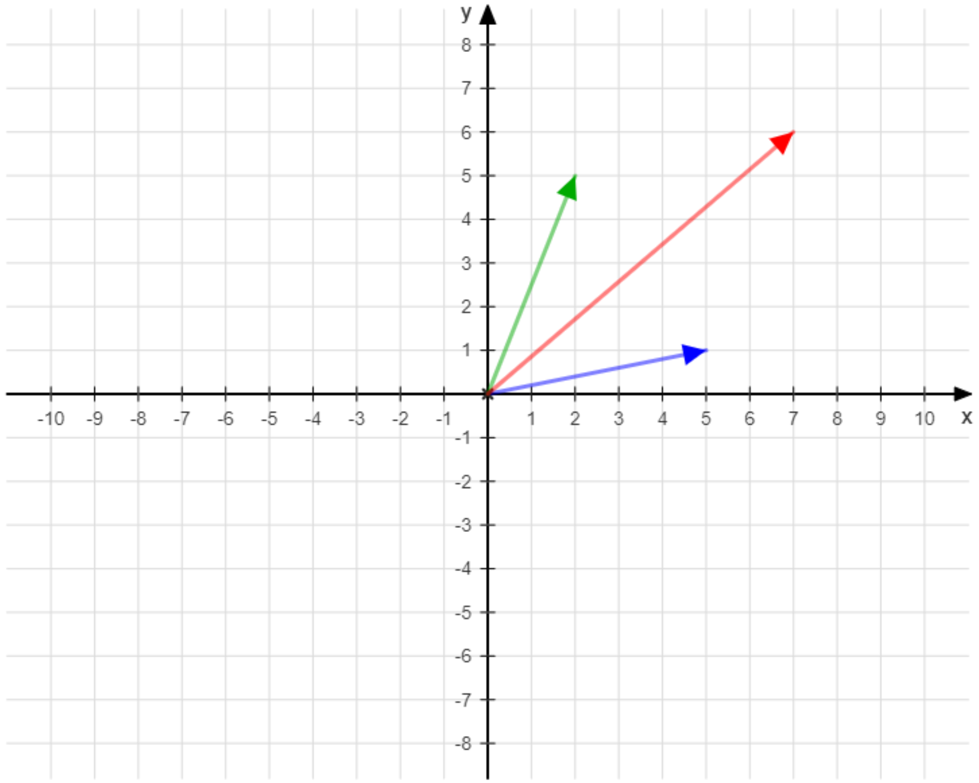
```
    Vector Position;
    Vector Velocity;
}
```

- Position is the position of a bird.
- The direction of velocity is the direction the bird is moving
- The magnitude of Velocity is the speed the bird is moving at.

Vector operations

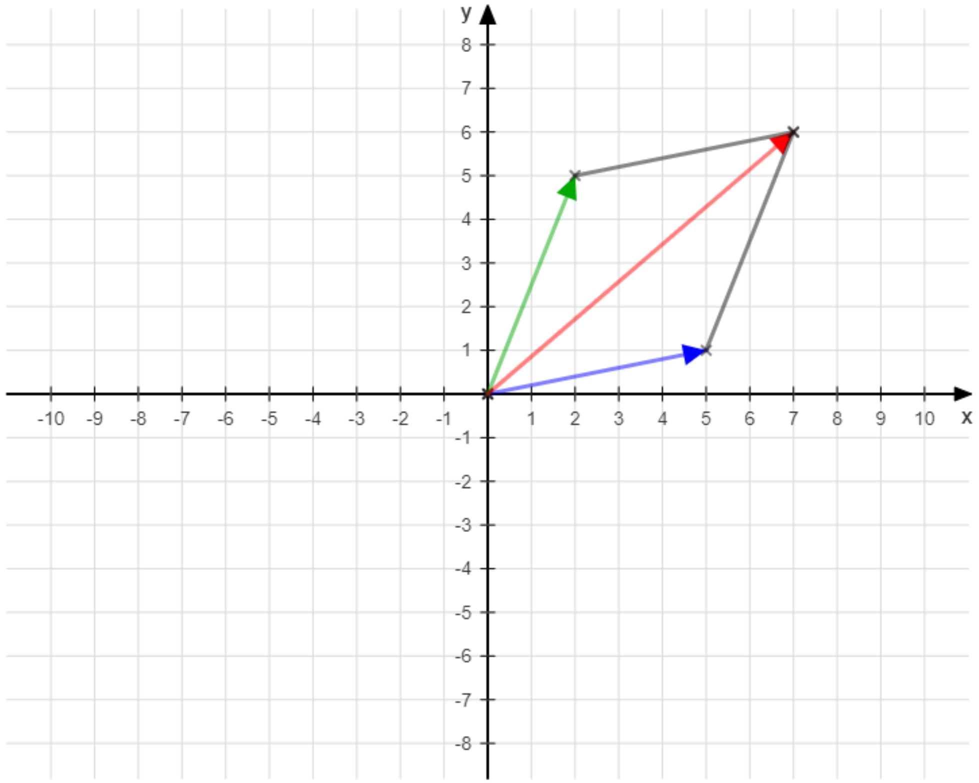
- V = [Vx, Vy]
- U = [Ux, Uy]

Vector - Addition



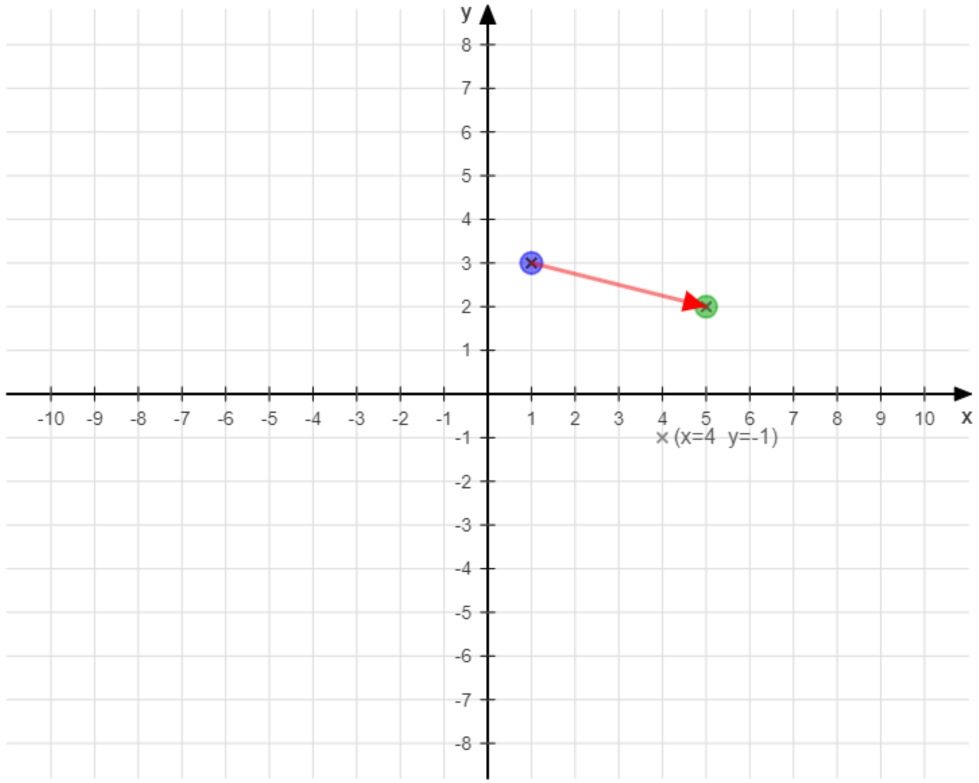
- V+U=[Vx+Ux, Vy+Uy]
- V = [5, 1]
  U = [2, 5]
  V + U = [5+2, 1+5]
  = [7, 6]

Vector - Addition



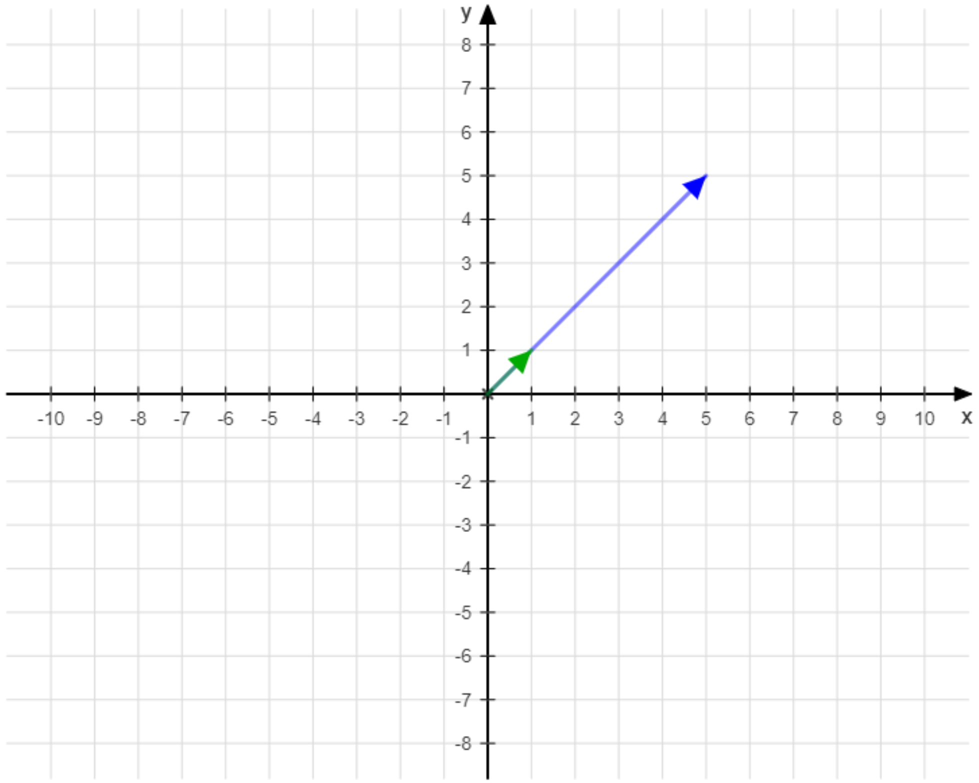
- V+U=[Vx+Ux, Vy+Uy]
- V = [5, 1]
  U = [2, 5]
  V + U = [5+2, 1+5]
  = [7, 6]

Vector - Subtraction



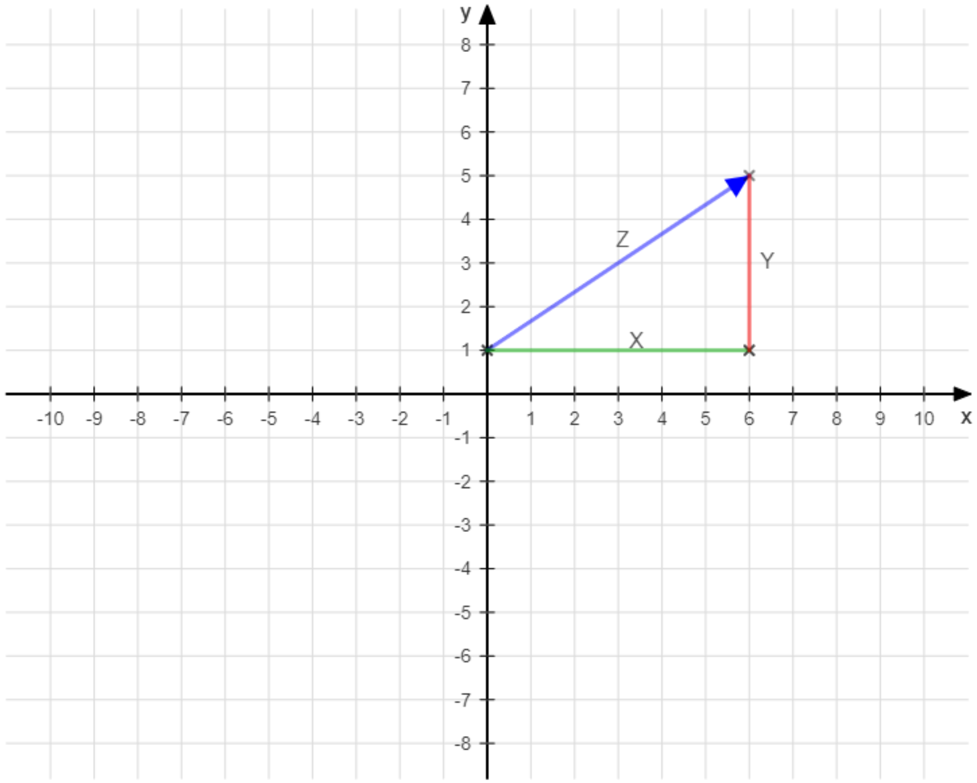
- We get the difference between to positions by subtracting them.
- The magnitude of this vector is the distance between two points
- V-U=[Vx-Ux, Vy-Uy]
- V = [1, 3]
  U = [5, 2]
  U - V = [5-1, 2-3]
  = [4, -1]

Vector - Multiplication by scalar



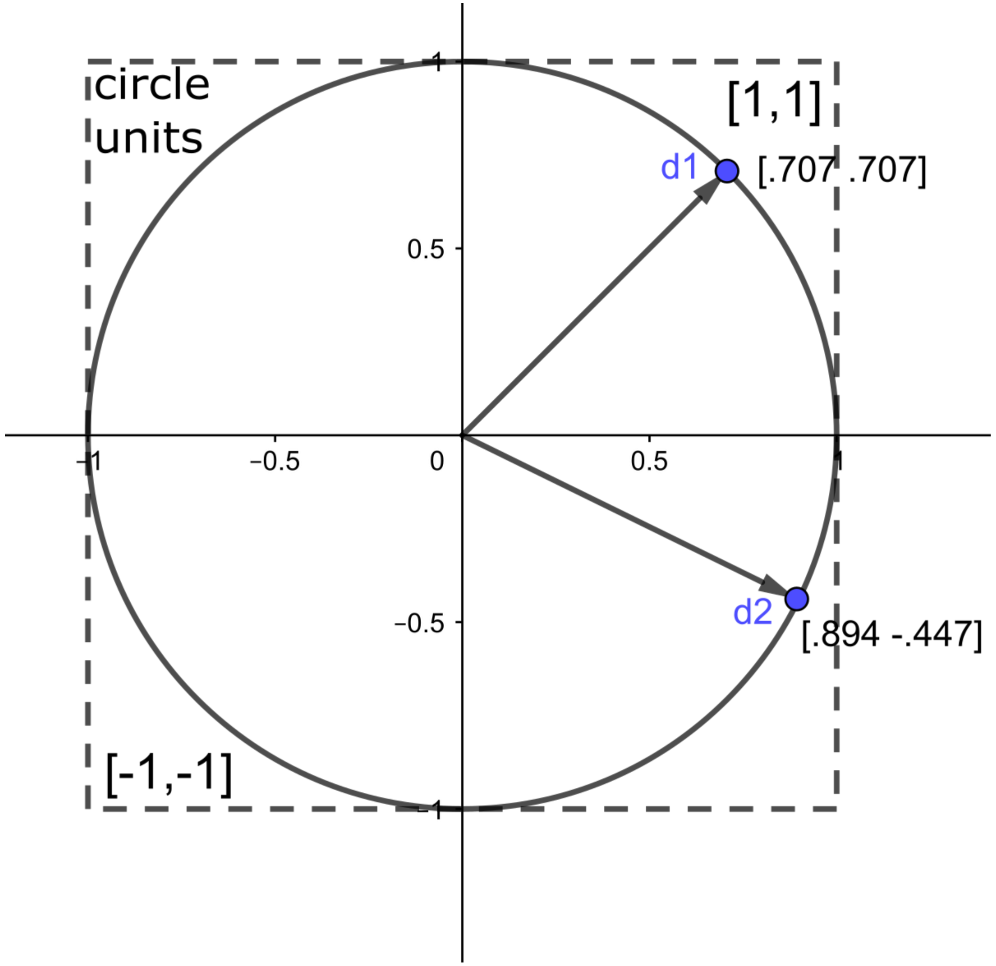
- V*s = [Vx*s, Vy*s]
- U = [1, 1]
  s = 5
  U * s = [1*5, 1*5]
  = [5, 5]

Vector Magnitude



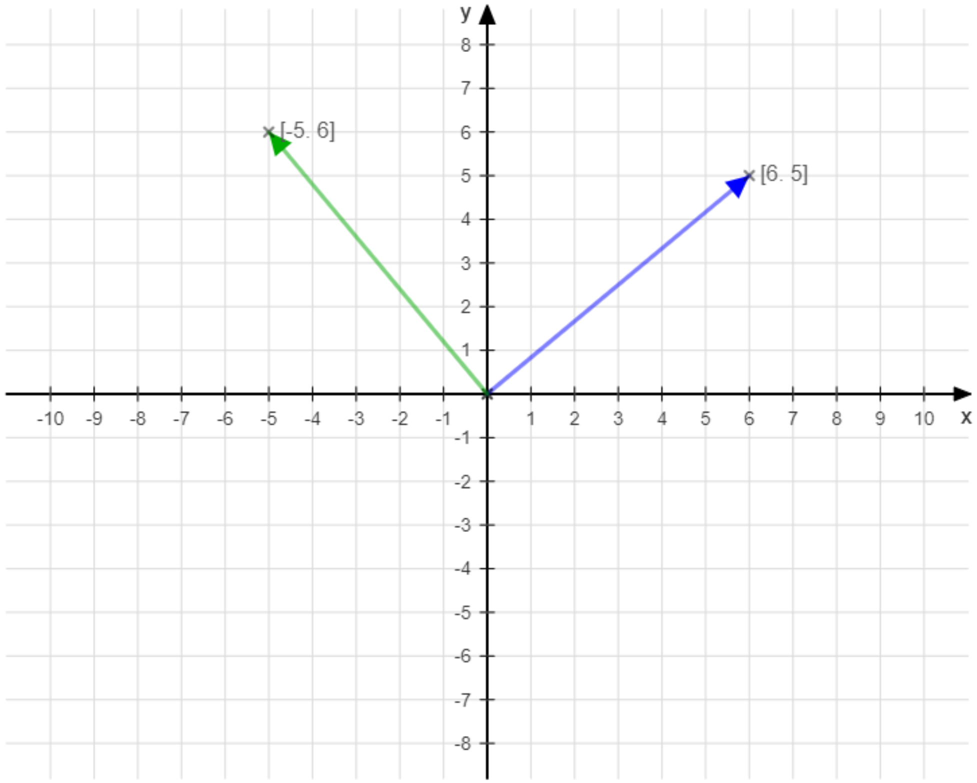
- We get the magnitude, |V|, of a vector, V, using Pythagorean theorem
- $z^2 = x^2 + y^2$
- |V| = sqrt(Vx*Vx + Vy*Vy)

Unit Vector



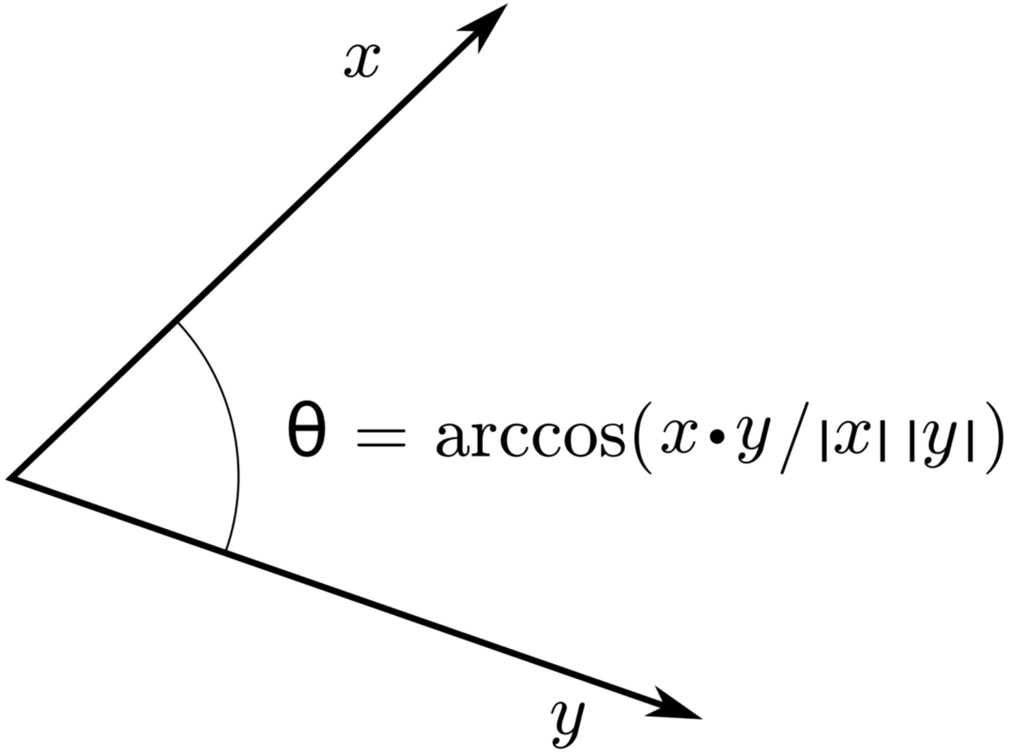
- Unit vector is a vector where |v|=1
- It is a point on a circle with radius 1
- We get the unit vector by normalizing a vector.
  Scale Vector by the inverse magnitude.
- Normalize(V)
  = V * (1 / |V|)

Perpendicular Vector



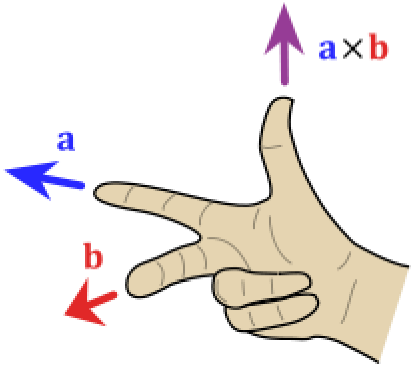
- Flip X & Y, and negate one of them.
- Perpendicular(V) = [-Vy, Vx]

Dot Product



- V.U = Vx*Ux + Vy*Uy
- Dot product is useful for finding the angle between two vectors (see image on left).
- You may use it to project a point upon a line.
- It is also useful for handling collision responses.

Cross Product



- OBS! Only applicable in 3 dimensions.
- For this course you can ignore it, just be aware of its existence when making games in 3 dimensions.
- *The statements above are not 100% true. It can be used in 2D in some special cases (z is then 0)*
- AxB != BxA

Matrix

Matrices in 2D games

What is a Matrix?

$$\begin{array}{cc}
 & \begin{array}{cccc} 1 & 2 & \cdots & n \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} & \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}
\end{array}$$

- A Matrix is a 2-dimensional array.
- In 2D, and in the context of this course. Matrices are "always" 3x3.
- We can use a Matrix of size 3x3 to transform a Vector of size 2.
- We can use matrices to move, rotate and scale vectors.

Identity Matrix

$$I_{2x2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad I_{3x3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Default “unit” matrix.
- Serves as a starting point for creating matrices.
- Multiplying a vector with the identity matrix results in the same vector.

Matrices

IDENTITY

- M =

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

TRANSLATION (MOVE)

- M =

$$\begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

Matrices

SCALE

- M =

$$M = \begin{bmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ROTATION

- M =

$$M = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Vector Transformation

- $M * V =$

$$\begin{bmatrix} M_{00} & M_{01} & M_{02} \\ M_{10} & M_{11} & M_{12} \\ M_{20} & M_{21} & M_{22} \end{bmatrix} * \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

$V' =$

$$\begin{bmatrix} M_{00}*P_x + M_{01}*P_y + M_{02}*1 \\ M_{10}*P_x + M_{11}*P_y + M_{12}*1 \\ M_{20}*P_x + M_{21}*P_y + M_{22}*1 \end{bmatrix}$$

Matrix Multiplication

- Combine multiple transformation by multiplying them together.
- Order matters! A * B != B * A

Matrix Multiplication

Matrix Multiplication

$$A * B = \begin{bmatrix} \boxed{A_{00} \quad A_{01} \quad A_{02}} \\ A_{10} \quad A_{11} \quad A_{12} \\ A_{20} \quad A_{21} \quad A_{22} \end{bmatrix} * \begin{bmatrix} \boxed{\begin{matrix} B_{00} \\ B_{10} \\ B_{20} \end{matrix}} & \begin{matrix} B_{01} \\ B_{11} \\ B_{21} \end{matrix} & \begin{matrix} B_{02} \\ B_{12} \\ B_{22} \end{matrix} \end{bmatrix}$$

$$[\ A_{00}*B_{00}+A_{01}*B_{10}+A_{02}*B_{20}$$

Matrix Multiplication

Matrix Multiplication

$$A * B = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{bmatrix}$$

$$[\ A_{00}*B_{00}+A_{01}*B_{10}+A_{02}*B_{20} \qquad A_{00}*B_{01}+A_{01}*B_{11}+A_{02}*B_{21}$$

Matrix Multiplication

Matrix Multiplication

$$A * B = \begin{bmatrix} \boxed{A_{00} \; A_{01} \; A_{02}} \\ A_{10} \; A_{11} \; A_{12} \\ A_{20} \; A_{21} \; A_{22} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{bmatrix}$$

$$[\; A_{00}*B_{00}+A_{01}*B_{10}+A_{02}*B_{20} \quad A_{00}*B_{01}+A_{01}*B_{11}+A_{02}*B_{21} \quad A_{00}*B_{02}+A_{01}*B_{12}+A_{02}*B_{22}$$

Matrix Multiplication

Matrix Multiplication

$$A * B = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{bmatrix}$$

$$\begin{bmatrix} A_{00}*B_{00}+A_{01}*B_{10}+A_{02}*B_{20} & A_{00}*B_{01}+A_{01}*B_{11}+A_{02}*B_{21} & A_{00}*B_{02}+A_{01}*B_{12}+A_{02}*B_{22} \\ A_{10}*B_{00}+A_{11}*B_{10}+A_{12}*B_{20} & & \end{bmatrix}$$

Matrix Multiplication

Matrix Multiplication

$$A * B = \begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{bmatrix}$$

[$A_{00}*B_{00}+A_{01}*B_{10}+A_{02}*B_{20}$ $\quad$ $A_{00}*B_{01}+A_{01}*B_{11}+A_{02}*B_{21}$ $\quad$ $A_{00}*B_{02}+A_{01}*B_{12}+A_{02}*B_{22}$

$A_{10}*B_{00}+A_{11}*B_{10}+A_{12}*B_{20}$ $\quad$ $A_{10}*B_{01}+A_{11}*B_{11}+A_{12}*B_{21}$

Matrix Multiplication

Continue…

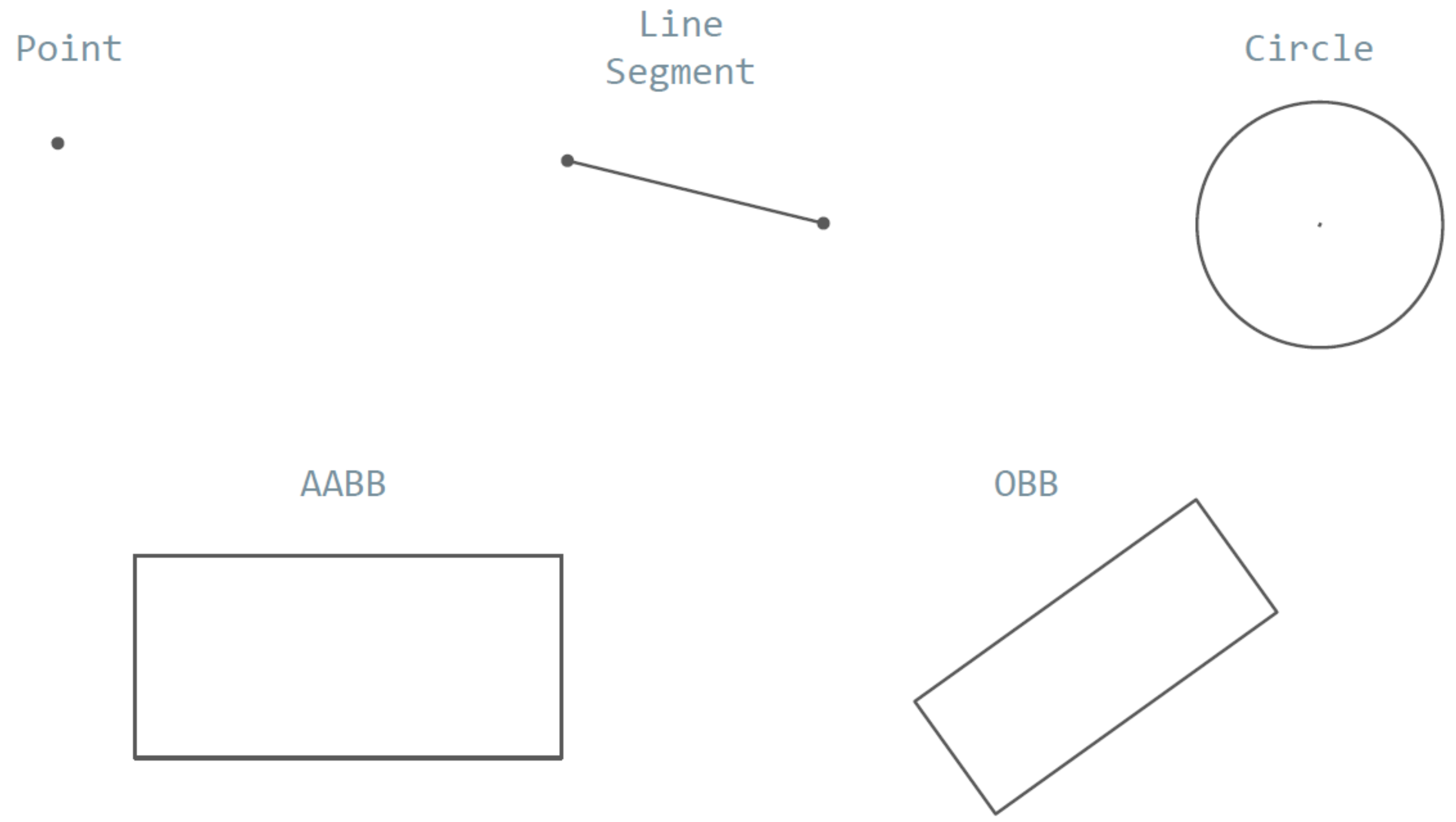
Collision Detection

Collision Detection in 2D games

Primitives

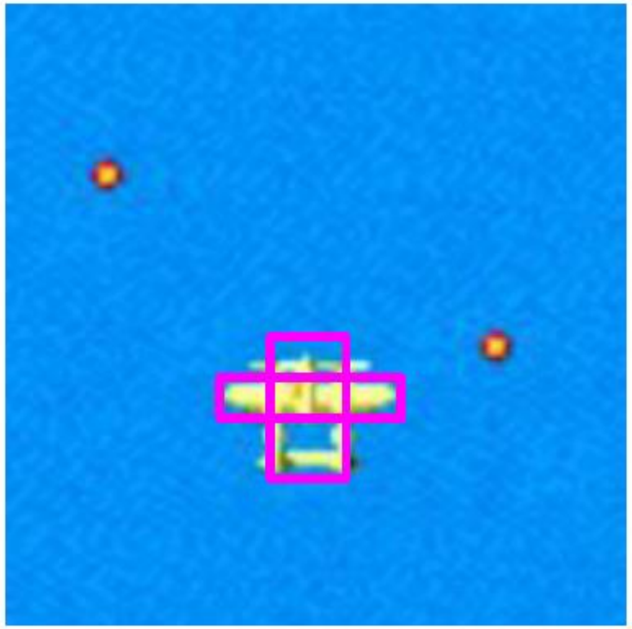
› Point
› Circle
› Line Segment
› Axis aligned bounding box (AABB)
› Object oriented bounding box (OBB)
› Convex polygons
› Concave polygons

Primitives



Primitives

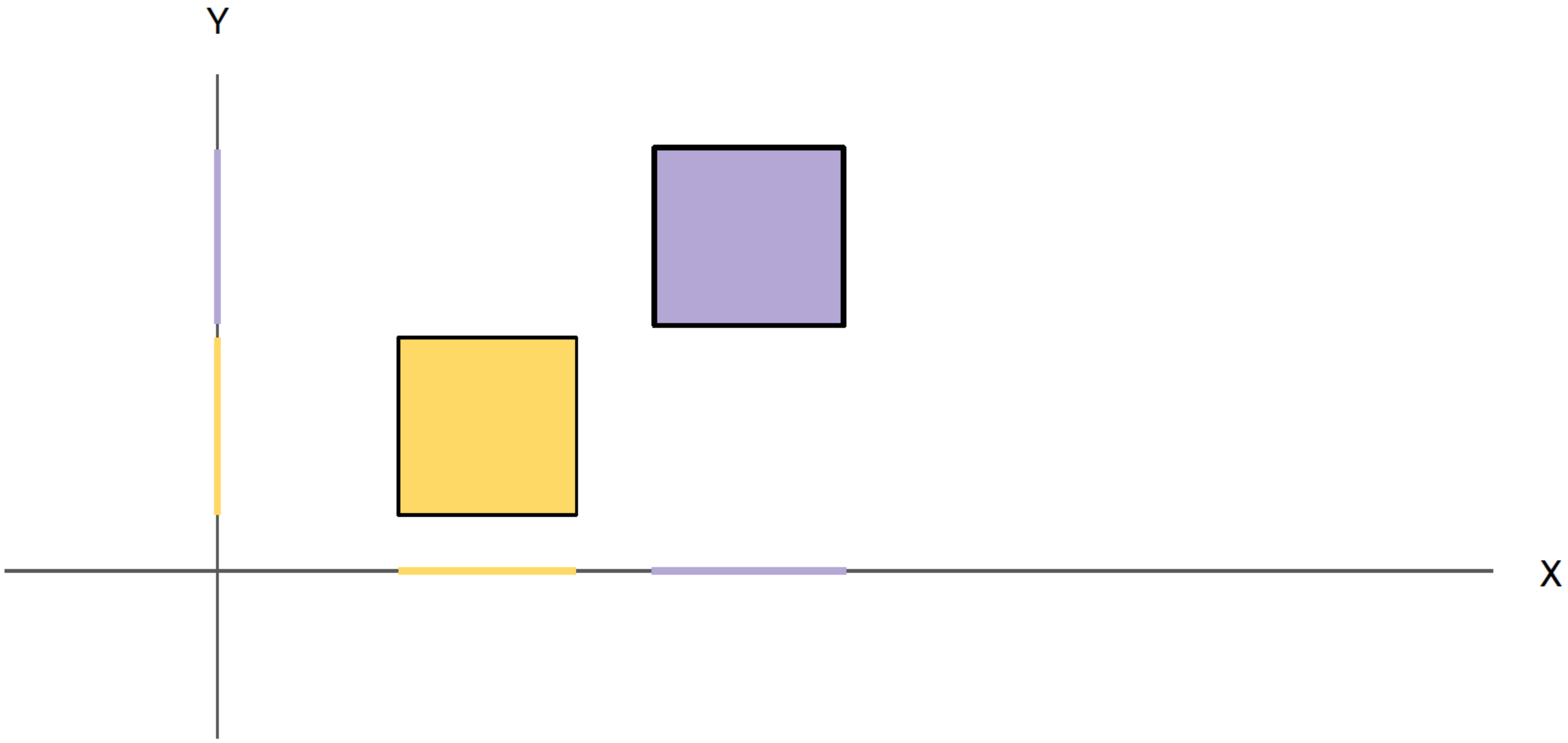
Capsule



Separating Axis Theorem (SAT)

- **Separating Axis Theorem** is used to find an intersection between two by projecting the shapes into all available axis. This simplifies calculations by looking at one axis at a time.
- Intersect: If the projection on all axis overlap
- !Intersect: If we find one axis where there is no overlap

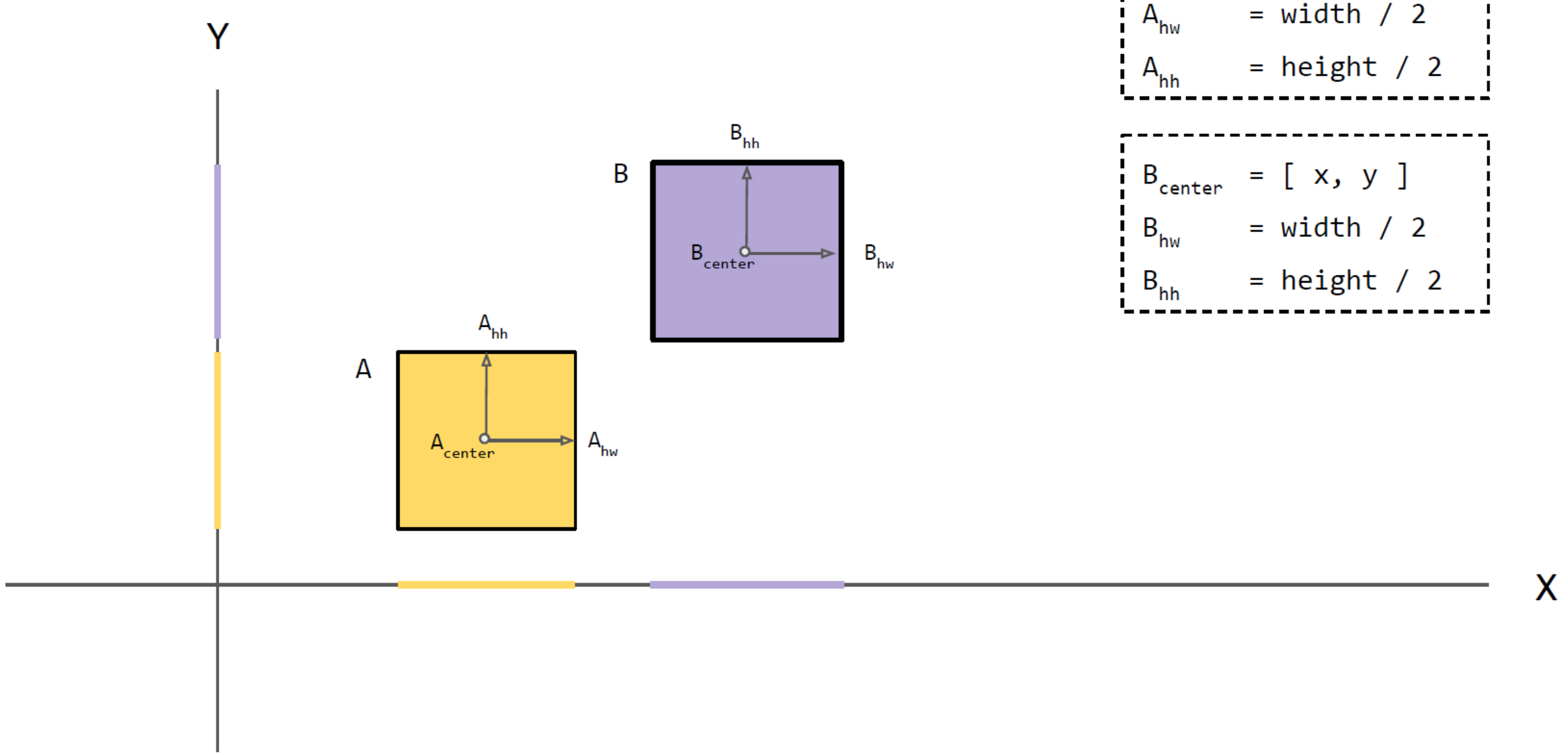
Separating Axis Theorem (SAT)



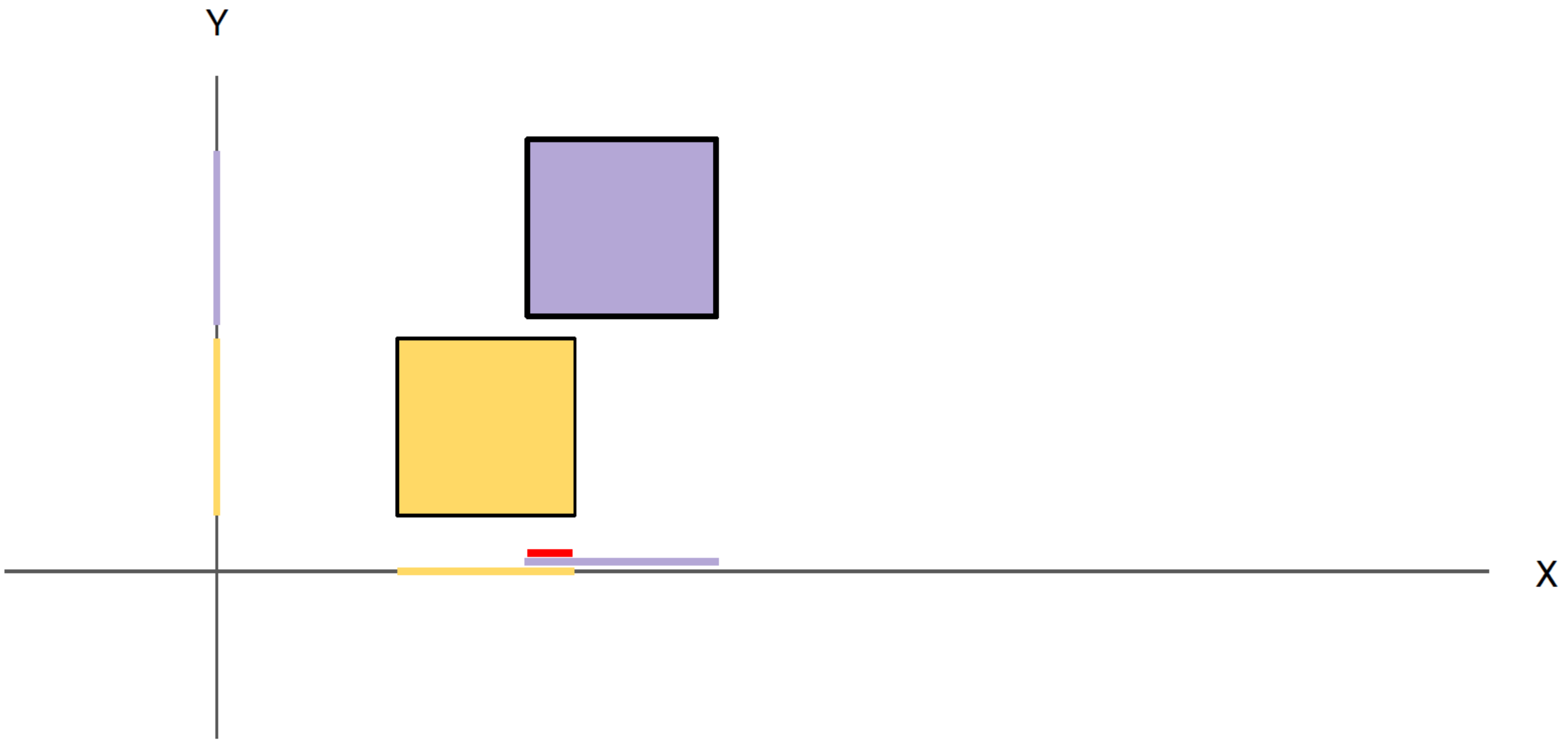
Separating Axis Theorem (SAT)

$$A_{center} = [\ x,\ y\ ]$$
$$A_{hw} = width\ /\ 2$$
$$A_{hh} = height\ /\ 2$$

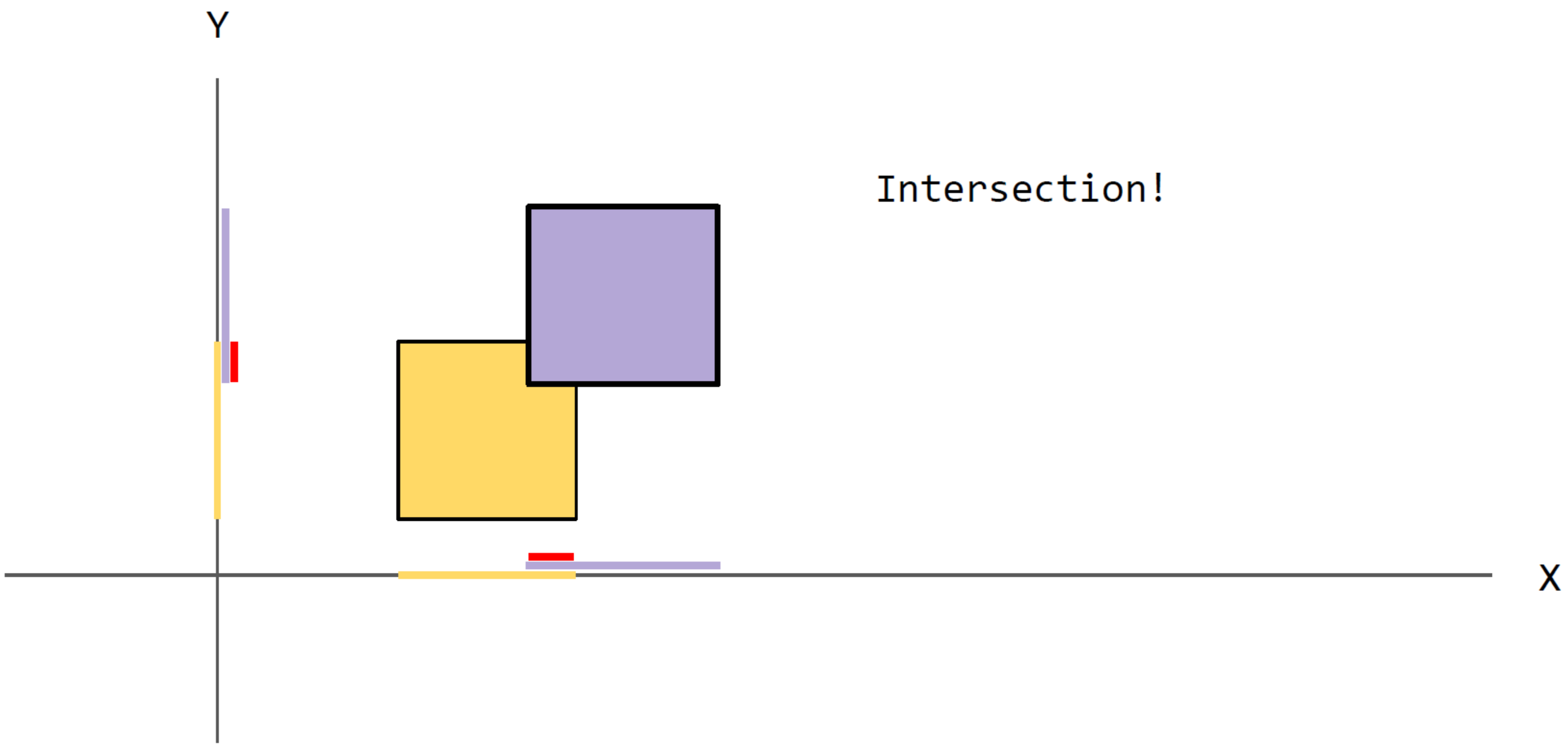
$$B_{center} = [\ x,\ y\ ]$$
$$B_{hw} = width\ /\ 2$$
$$B_{hh} = height\ /\ 2$$



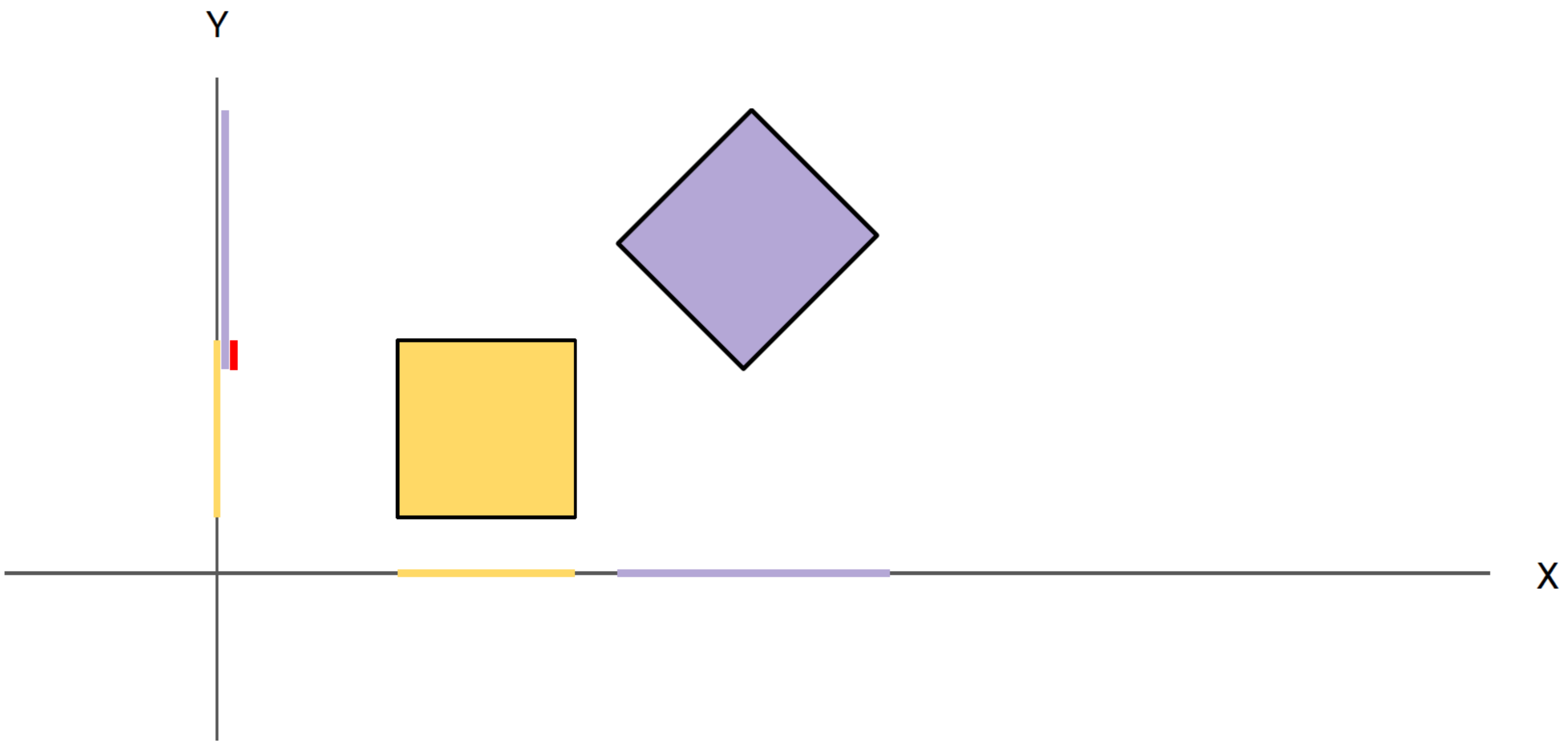
Separating Axis Theorem (SAT)



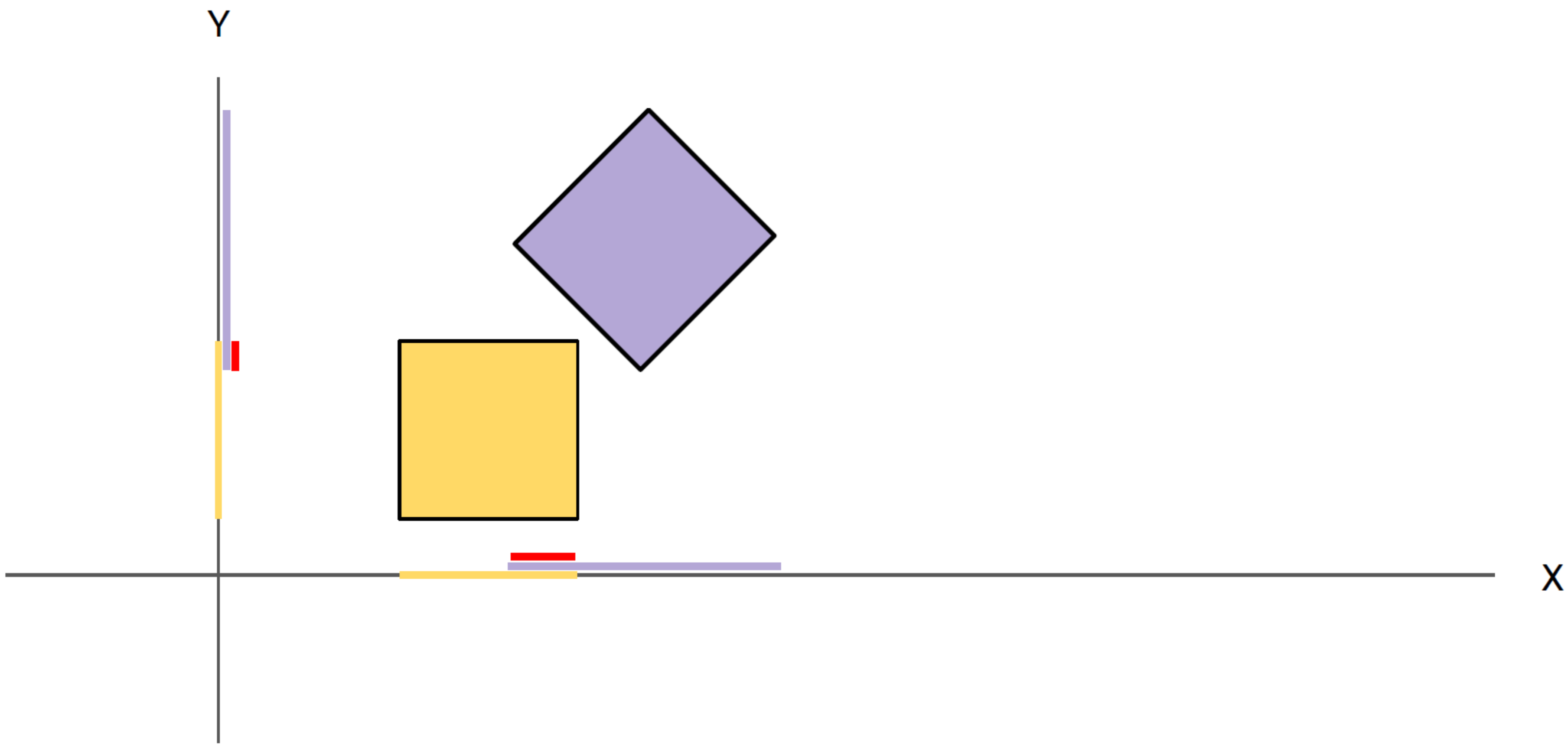
Separating Axis Theorem (SAT)



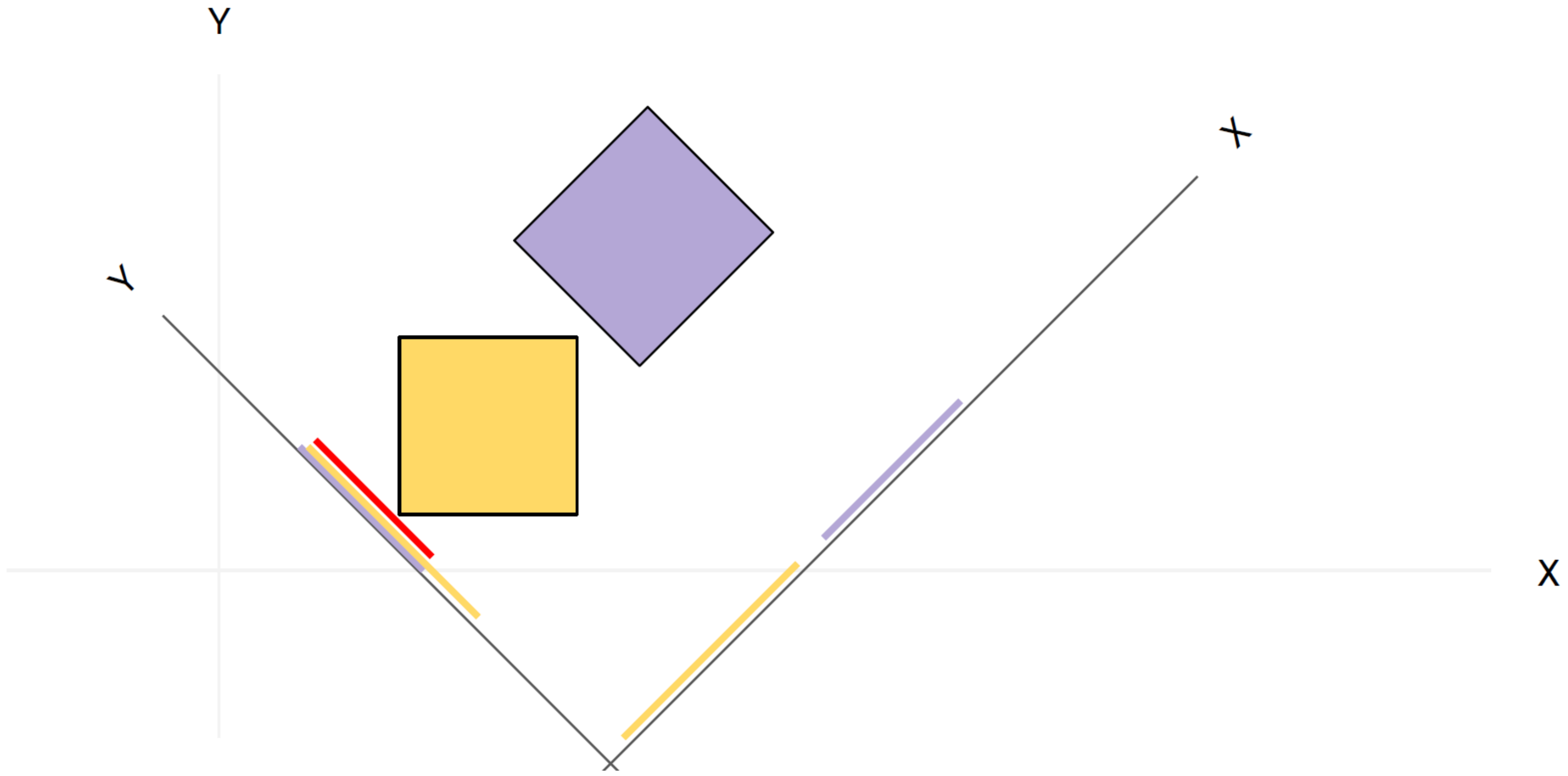
Separating Axis Theorem (SAT)



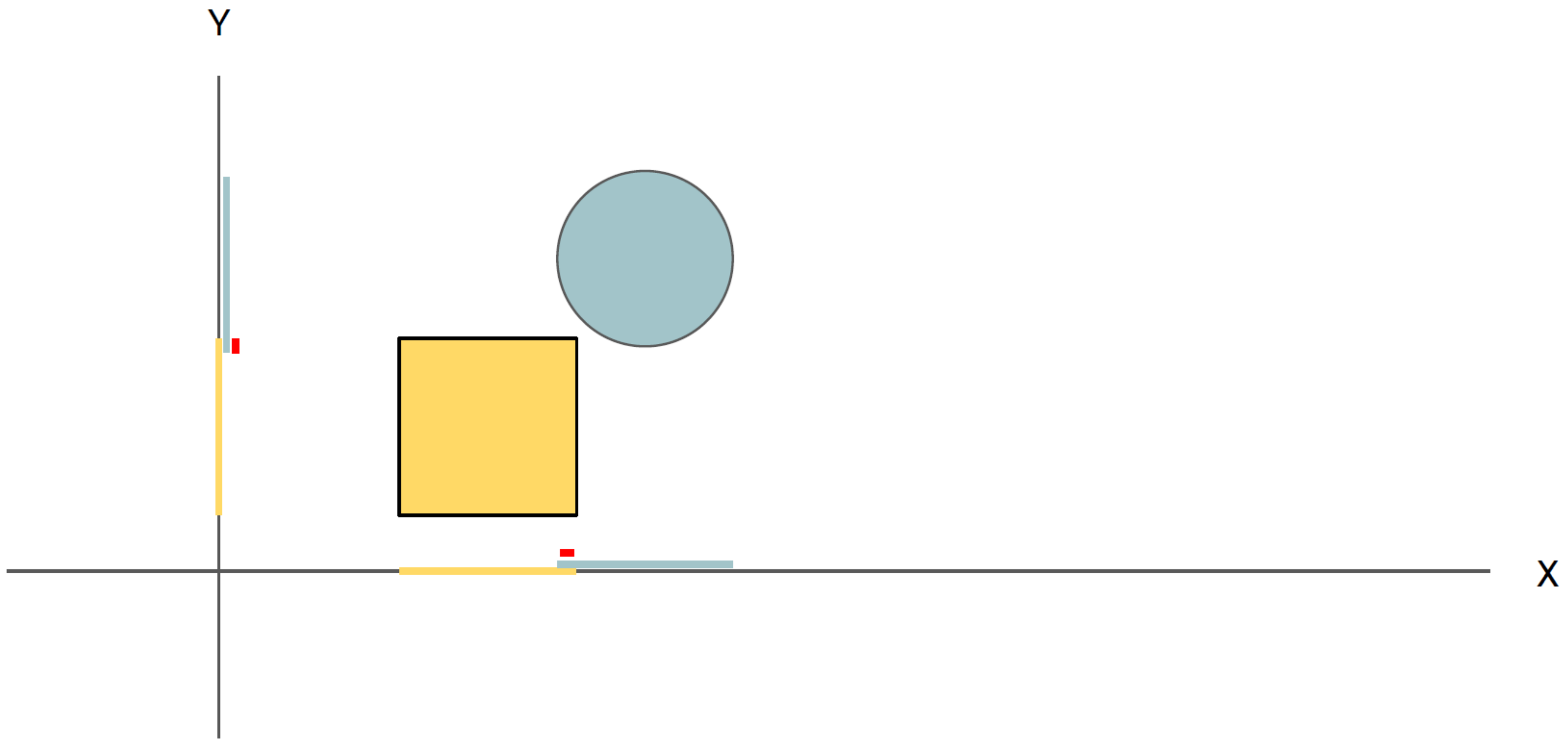
Separating Axis Theorem (SAT)



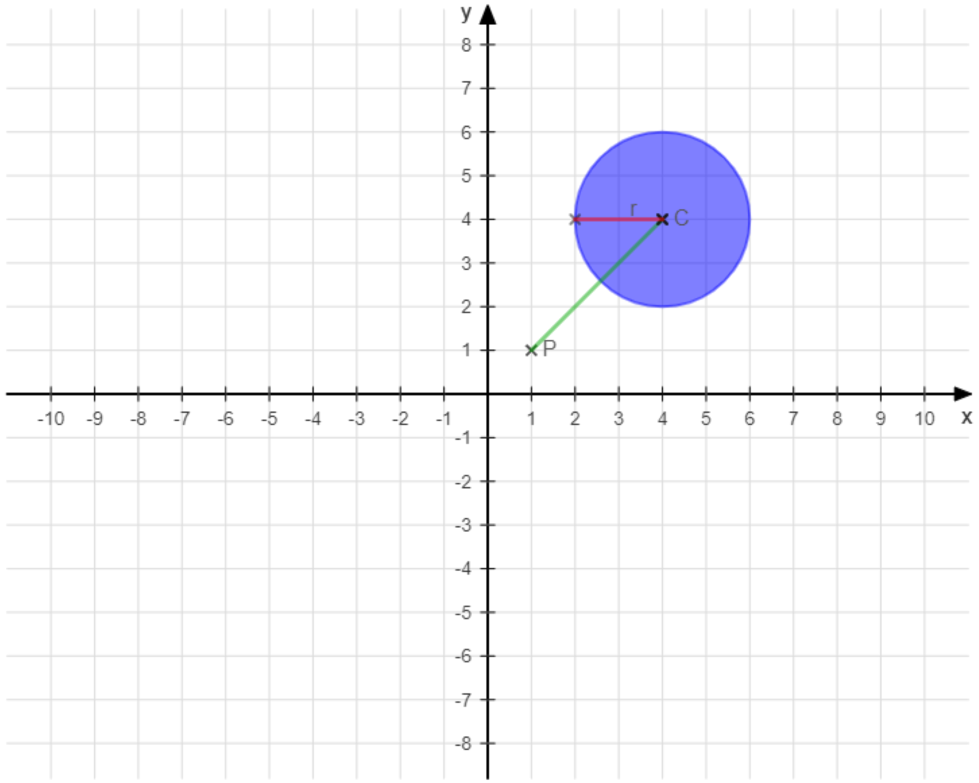
Separating Axis Theorem (SAT)



Separating Axis Theorem (SAT)

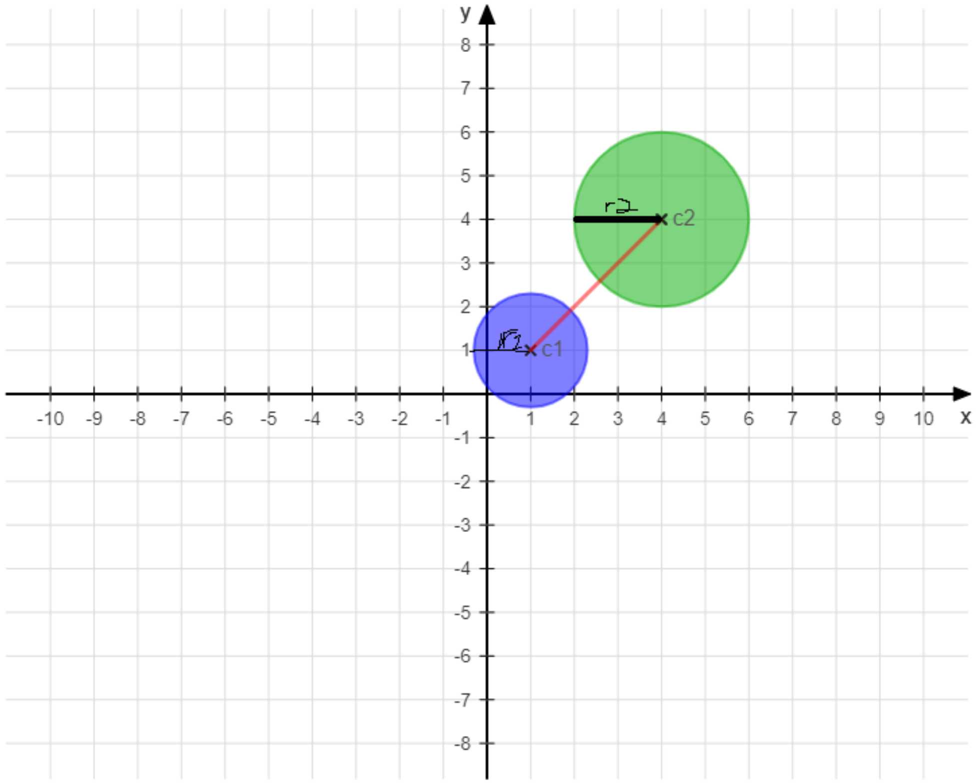


Point - Circle



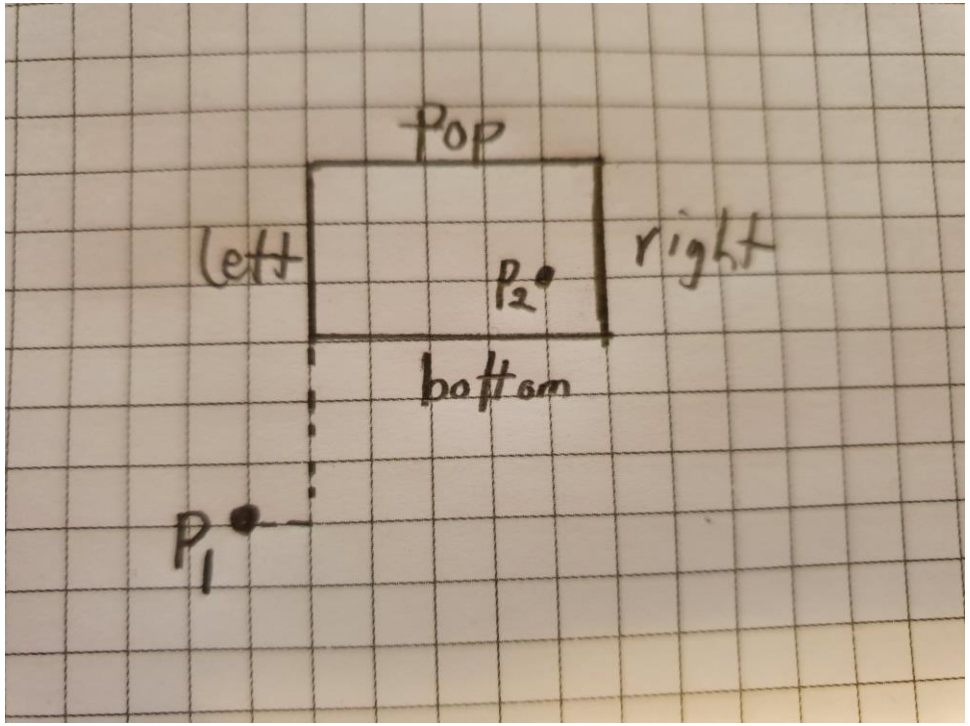
- A point is inside a circle if the distance from the point to the circle is less or equal to the circle's radius
- v = c - p
- Is Colliding IF |v| <= r

Circle - Circle



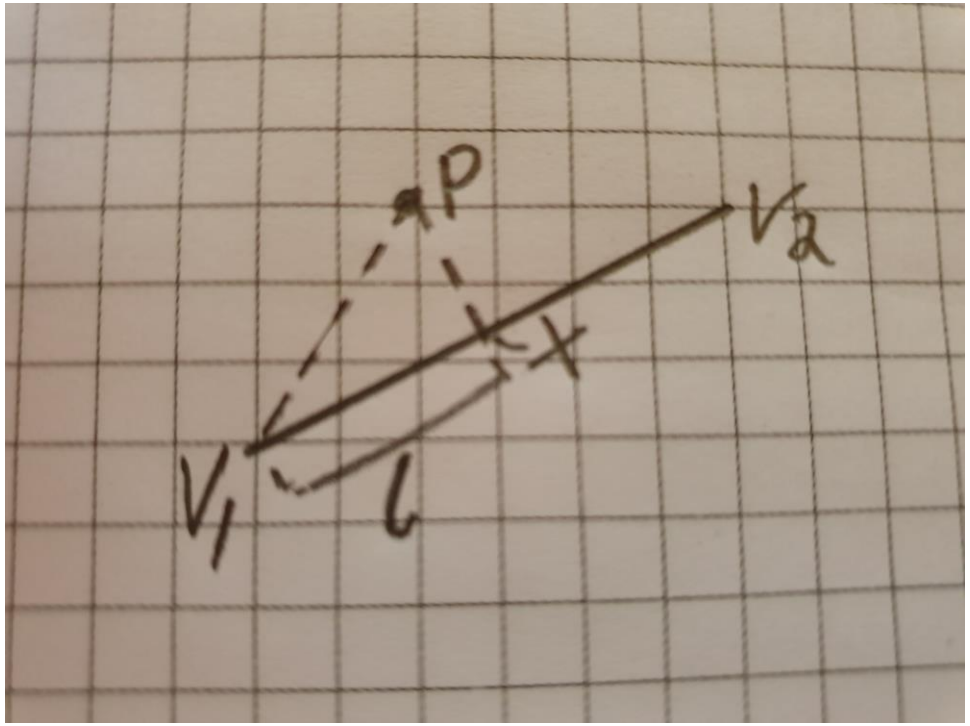
- Two circles are colliding if the distance between them is less than the sum of their radiuses.
- v = c2-c1
- r = r1+r2
- Is Colliding IF |v| <= r

Point - AABB



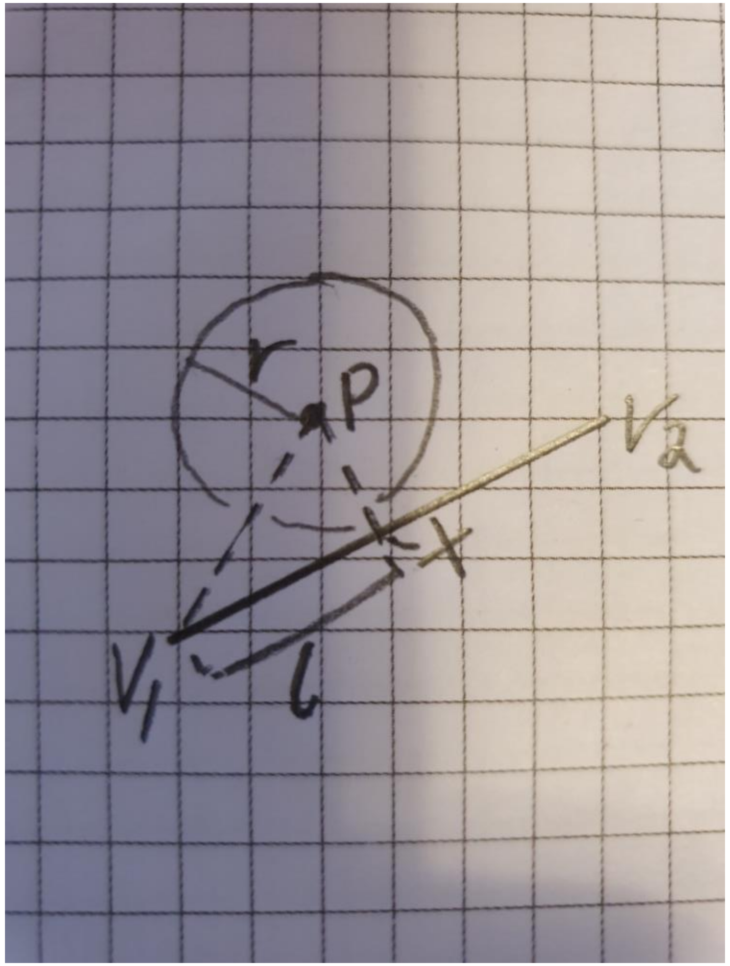
- IF left of left -> No Collission
- IF right of right -> No Collision
- IF over top -> No Collision
- IF under bottom -> No Collision
- Note p1 is left of left & under bottom, so it's separated in two axis.

Closest point on line segment to point



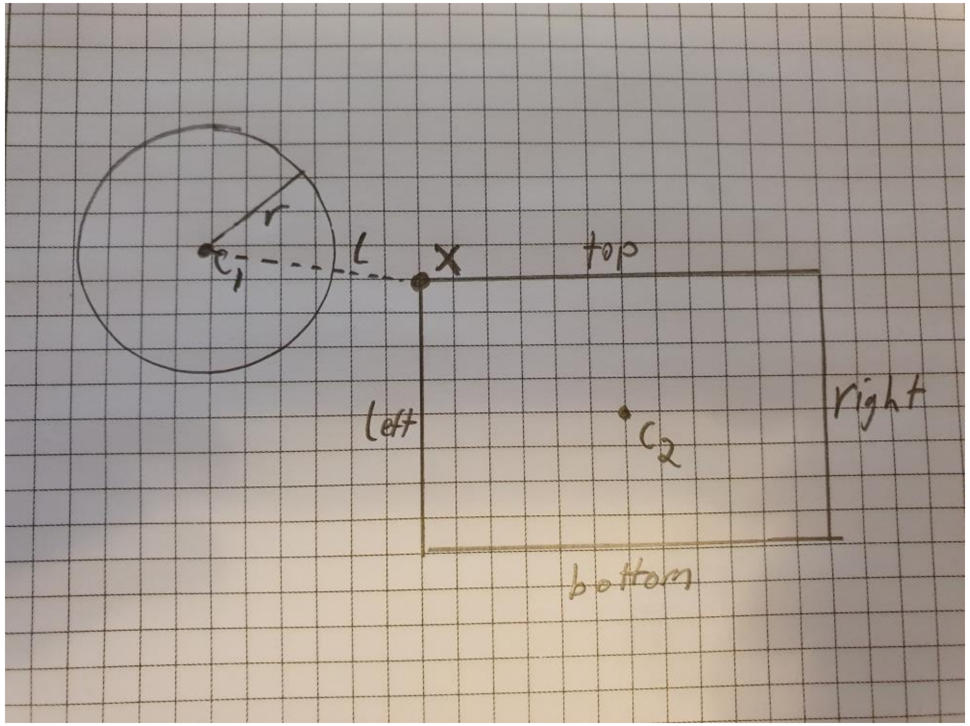
- A line segment is defined by two points
  - v1 & v2
- A point is defined by
  - p
- Create a vector:
  - v = v2 - v1
- v's unit vector:
  - u = v / |v|
- Get distance from v1:
  - l = u dot (p-v1)
- Limit l to line segment:
  - 0 <= l <= |v|
- Closes point:
  - x = v1 + u * l

Line Segment - Circle



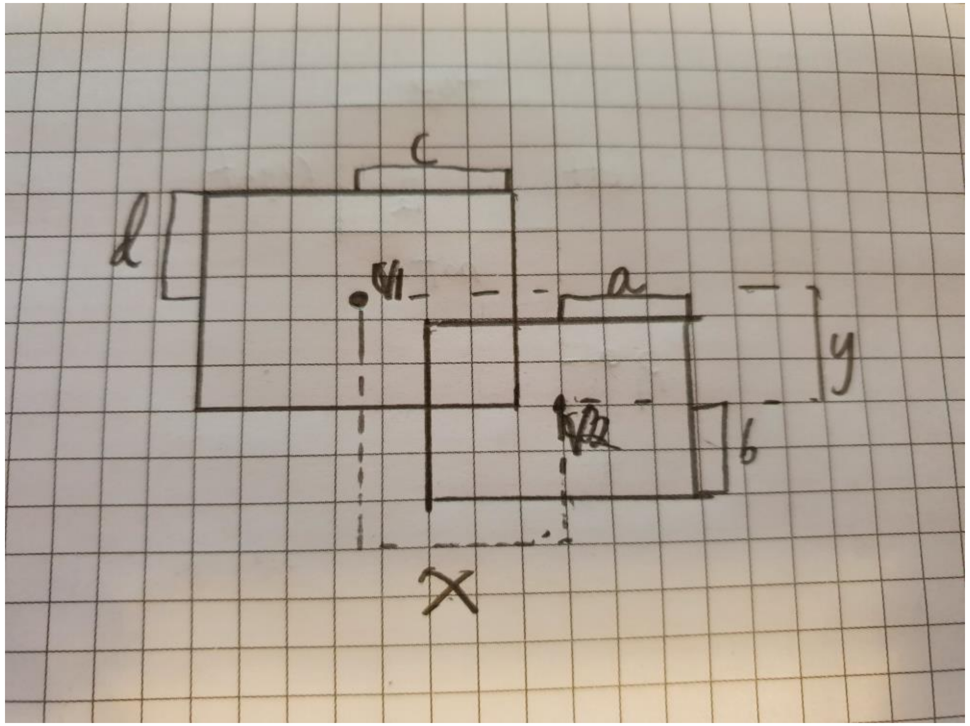
- A circle is colliding with a line segment IF the distance from the closest point to centre of circle is less or equal to it’s radius.
  - |x-p| <= r

Circle - AABB



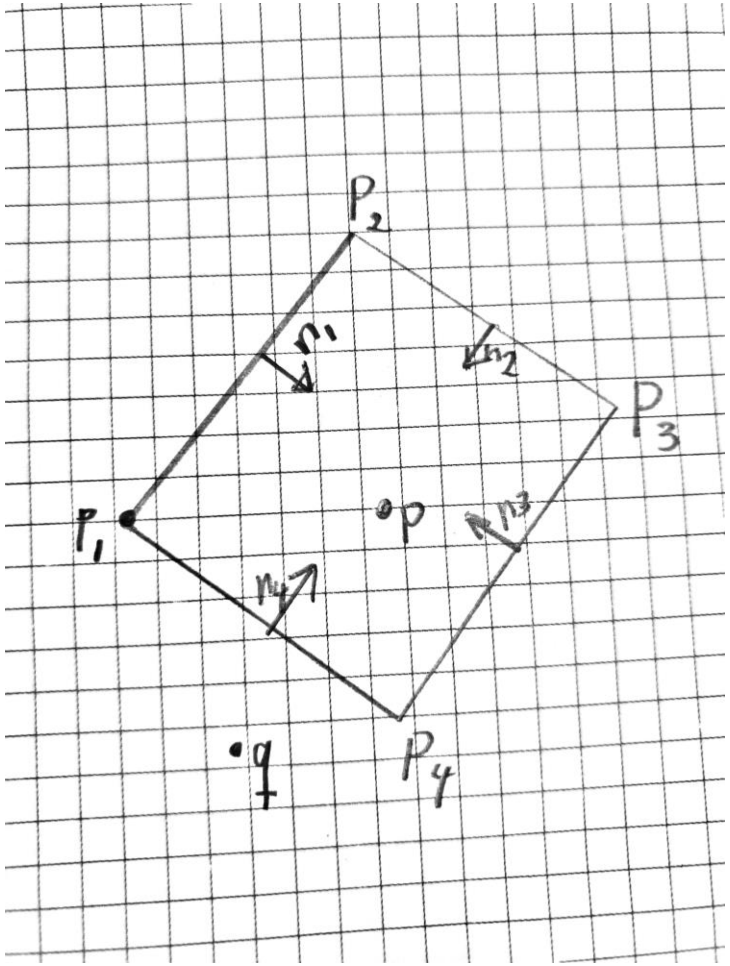
- Circle collides if the closest point x inside a box is less or equal to the circle's radius.
- x is obtained by clamping c1 to left, bottom, top, right.
  - x = [clamp(c1.x, left, right), clamp(c1.y, bottom, top)]
- Collision IF
  - |x-c1| <= r

AABB - AABB



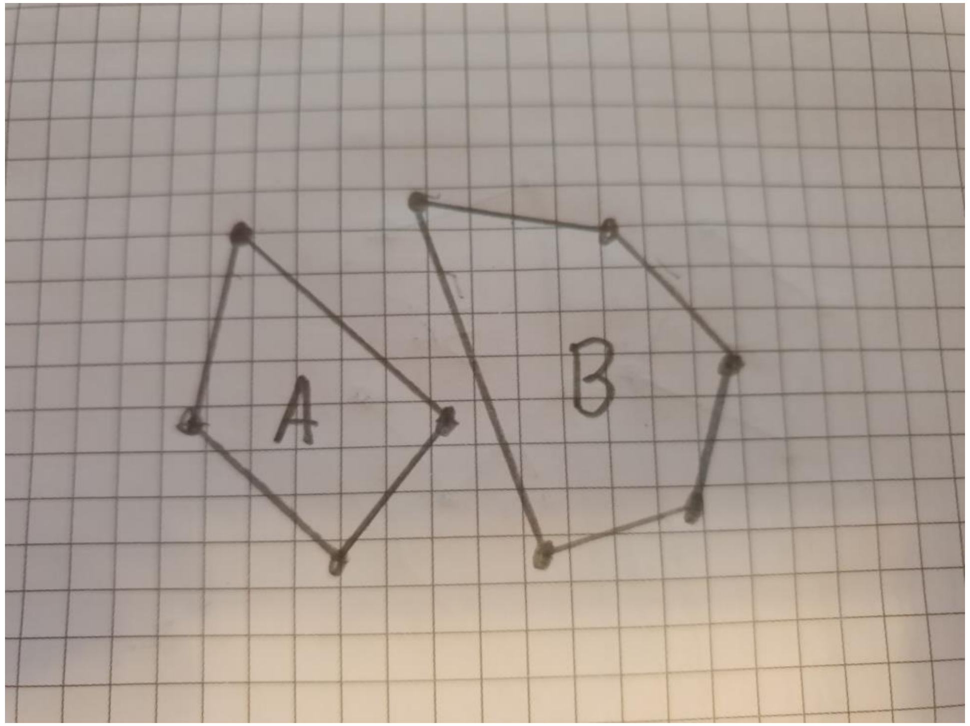
- if x > c+a: No Collision
- if y > d+b: No Collision
- else: Yes Collision

Point – OBB (or any convex polygon)



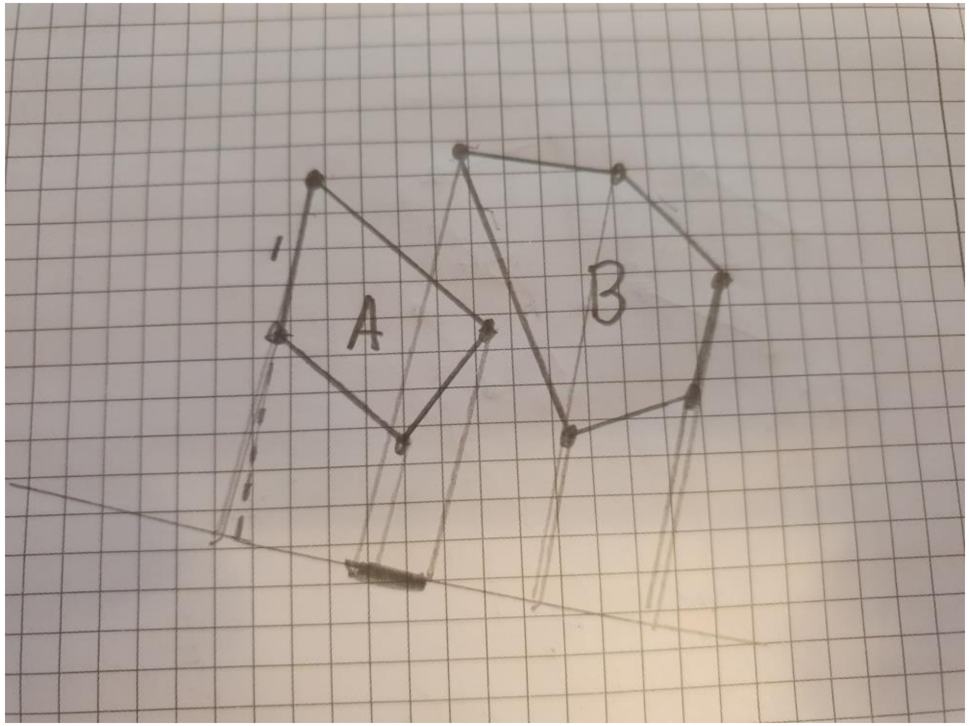
- For every edge (p2-p1, p3-p2, p4-p3, p1-p4)
  - calculate normal n
  - IF n dot point < 0: No Collision
- A point is inside polygon only if its on the positive side of all normal vectors.

Convex Polygon – Convex Polygon SAT



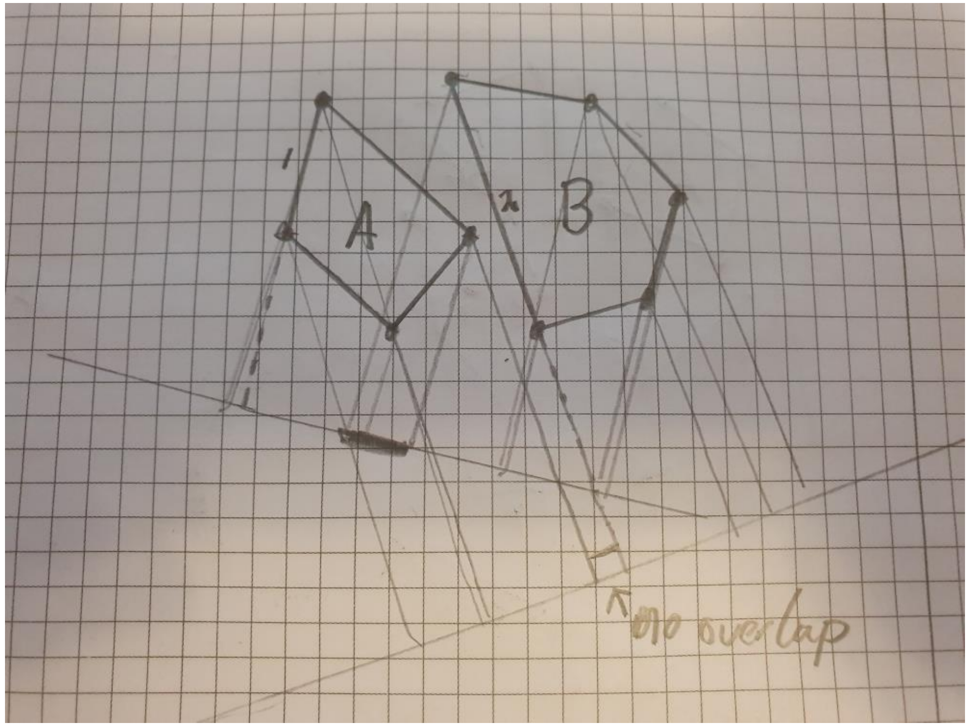
› Two polygons are defined by their vertices.

Convex Polygon – Convex Polygon SAT



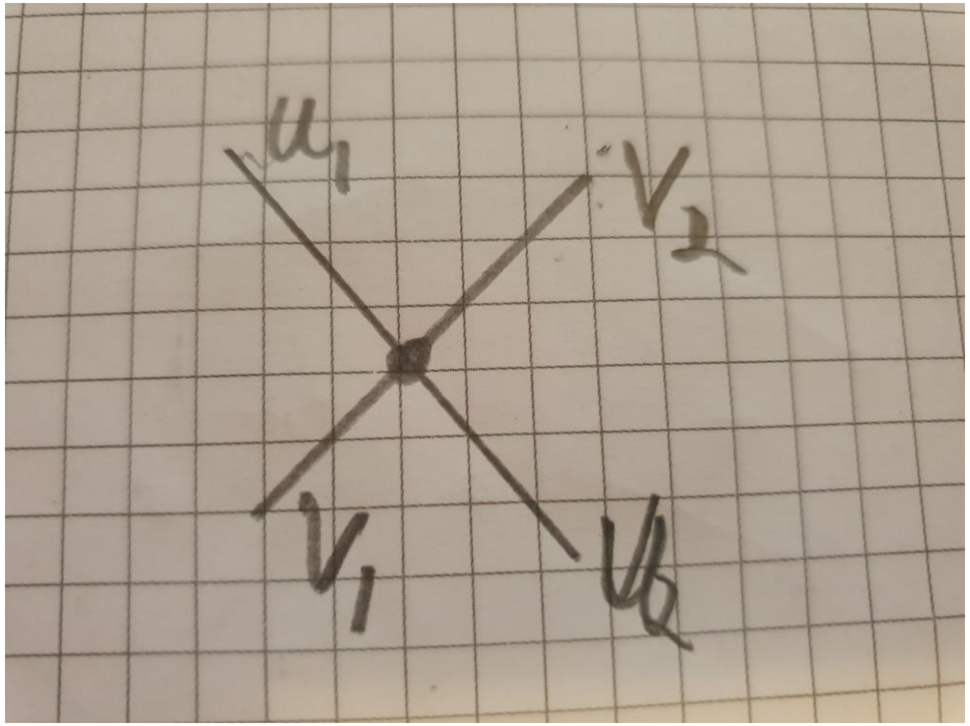
- Two polygons are defined by their vertices.
- For every edge, project all points to the edges normal. If no overlap, there is no collision.
- (1) has overlap, continue iterating through the edges.

Convex Polygon – Convex Polygon SAT



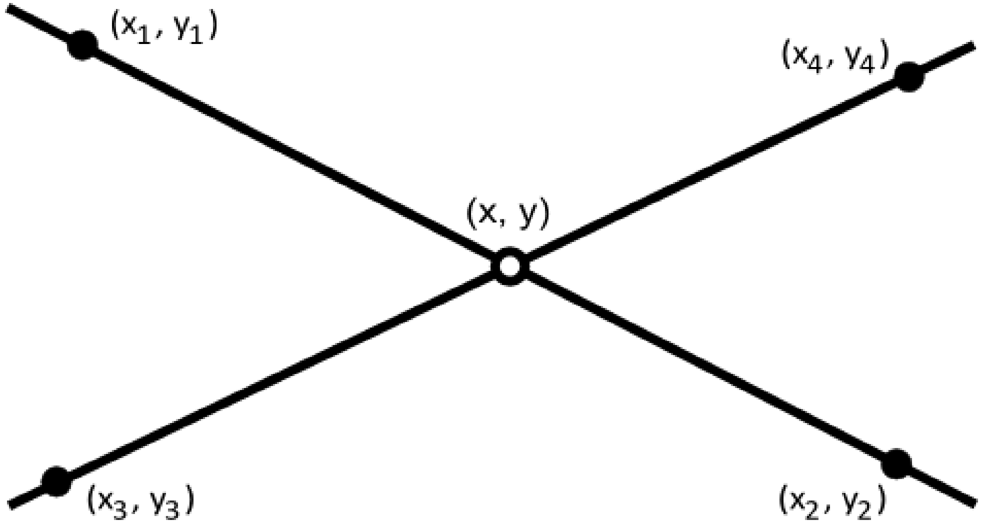
- Two polygons are defined by their vertices.
- For every edge, project all points to the edges normal. If no overlap, there is no collision.
- (2) has no overlap. No collision.

Line segment – Line segment Method 1



- A point on a line from v1 to v2 can be defined by
  - f(j) = v1 + j(v2-v1)
  - f(0) = v1
  - f(1) = v2
- f(k) = u1 + k(u2-u1)
- An equation can be defined
  - v1 + j(v2-v1) = u1 + k(u2-u1)
  - If 0 <= j&k <= 1 then there is a collision

Line segment – Line segment Method 1



- Separating x & y in to separate equations
  - a = x1
  - b = x2-x1
  - c = x3
  - d = x4-x3

  - p = y1
  - q = y2-y1
  - r = y3
  - s = y4-y3

  - a + bj = c + dk
  - p + qj = r + sk

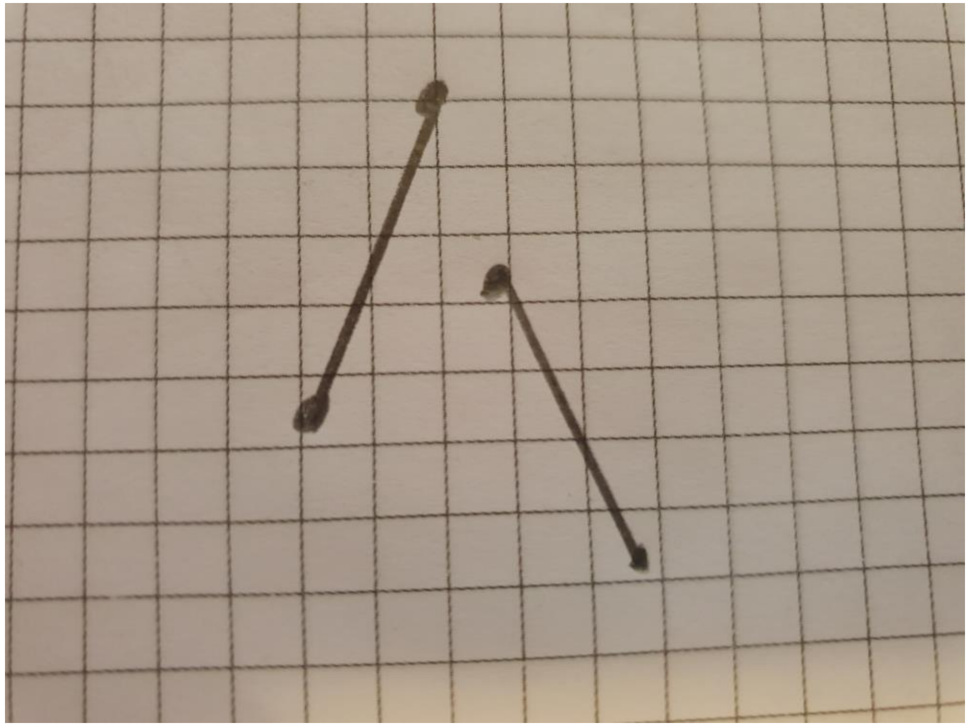
  - k = (-aq + bp - br + cq) / (bs - dq)
  - j = (-as + cs + dp - dr) / (bs - dq)

Line segment – Line segment Method 1

- k = (-aq + bp - br + cq) / (bs - dq)
- j = (-as + cs + dp - dr) / (bs - dq)

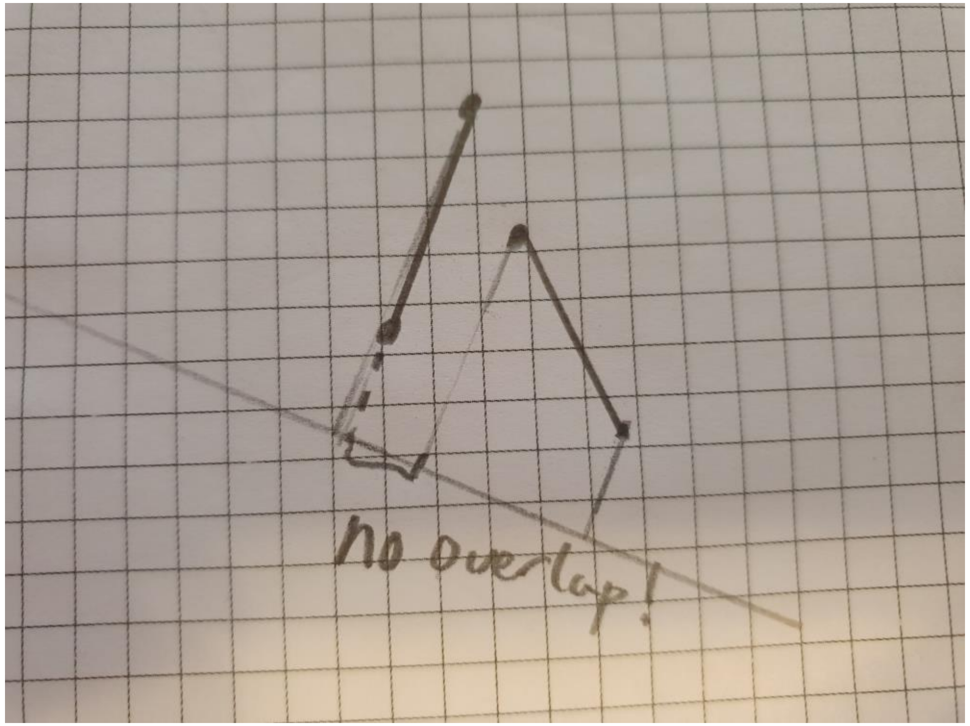
- If (bs - dq) is zero, then the line segments are parallel. No collision unless they reside on the same line and overlap.
- ELSE: collision IF j & k are >= 0 & <= 1

Line Segment – Line Segment Method 2 - SAT



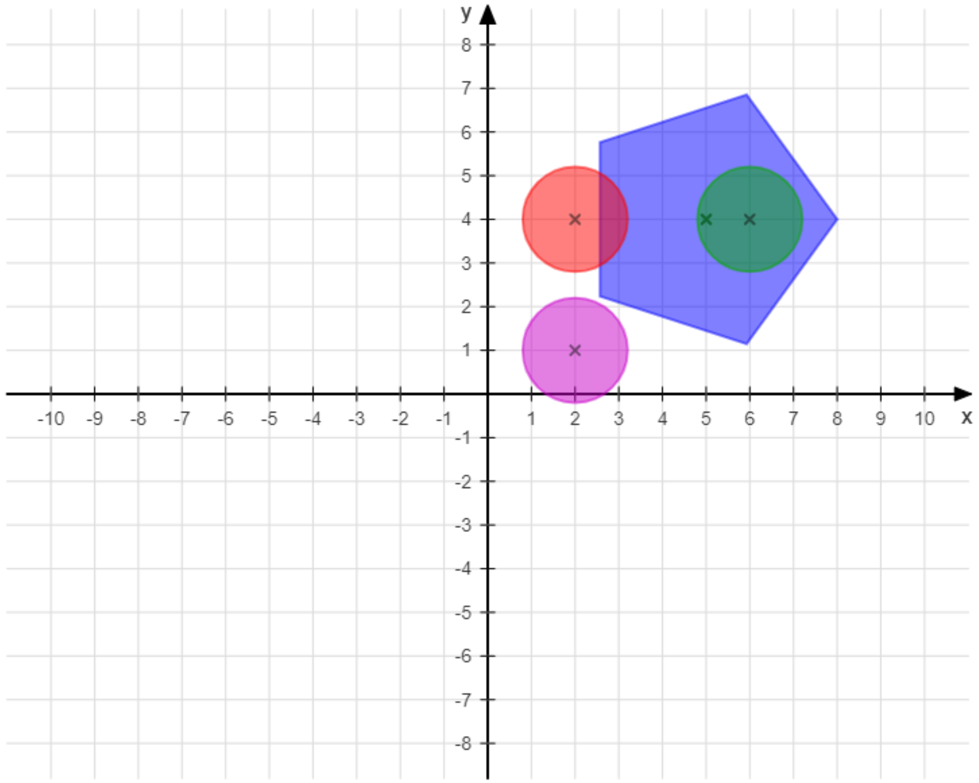
- Use SAT to check for collision.

Line Segment – Line Segment Method 2 - SAT



- Project the endpoints on all edges.
- If there exists an axis with no overlap, there is no collision.

Circle – Convex Polygon



- A circle that is colliding with a Convex Polygon is either
  - A. Completely inside the polygon. Or
  - B. Colliding with an edge on polygon.