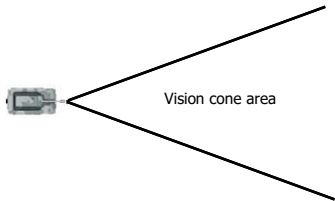


Vision Cone

- Implementing a vision cone - hints
- Goal – to determine whether a point lies inside two skew lines, e.g.




CW208-2-2022/23, Project 2

1

Vision Cone

- Let's consider the left (top) line of the vision cone
- The question really boils to down whether a point is to the left (top) or right (bottom) of this line:



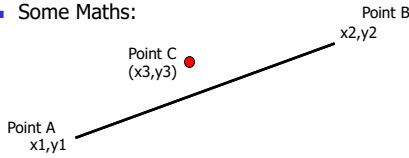
- If we can answer this question for both lines, then we can determine if something is inside the vision cone

CW208-2-2022/23, Project 2

2

Vision Cone

- Some Maths:

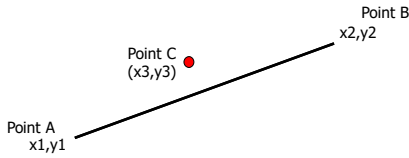


- We can use the cross-product (*2D wedge products*) to determine if point C is to the left or right of the line B->A

CW208-2-2022/23, Project 2

3

Vision Cone



- Cross-product is equivalent to *determinant* of a 2D matrix:

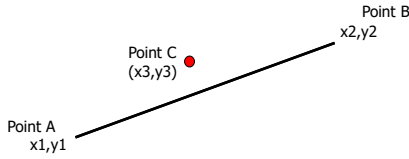
$$\begin{vmatrix} x2-x1 & x3-x1 \\ y2-y1 & y3-y1 \end{vmatrix}$$

i.e. $(x2-x1) * (y3-y1) - (y2 - y1) * (x3-x1)$

CW208-2-2022/23, Project 2

4

Vision Cone



If result of $(x2-x1) * (y3-y1) - (y2 - y1) * (x3-x1)$

- $= 0$ then all three points are co-linear (lie on same line)
- > 0 then point is to left of line (looking from B to A)
- < 0 then point is to right of line (looking from B to A)

CW208-2-2022/23, Project 2

5


Vision Cone

- Sample implementation:

```
bool isLeft(sf::Vector2f t_linePoint1,
            sf::Vector2f t_linePoint2,
            sf::Vector2f t_point) const
{
    // return ( (x2-x1) * (y3-y1) ) - ( (y2 - y1) * (x3-x1) )
    return ((t_linePoint2.x - t_linePoint1.x) *
            (t_point.y - t_linePoint1.y) -
            (t_linePoint2.y - t_linePoint1.y) *
            (t_point.x - t_linePoint1.x)) > 0;
}
```

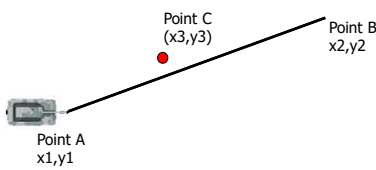
CW208-2-2022/23, Project 2

6



Vision Cone

- We should pass arguments in correct order to **isLeft()** so we take the perspective of the tank, i.e, looking from point A to B (not B to A)



- isLeft(B, A, C) returns true (as expected)
- isLeft(A, B, C) returns false (wrong perspective)

CW208-2-2022/23, Project 2

7