Slide 1
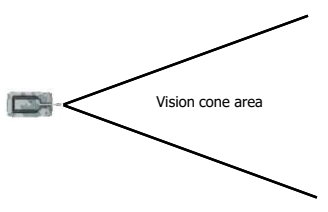
Vision Cone

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

Vision cone area

1

Slide 2

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:
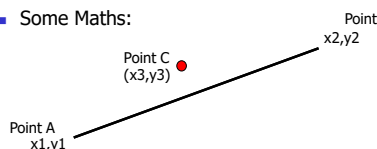

- If we can answer this question for both lines, then we can determine if something is inside the vision cone

2

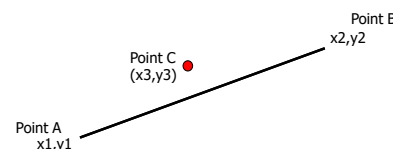## Slide 3

Vision Cone

- Some Maths:

Point B
x2,y2

Point C
(x3,y3)

Point A
x1,y1

- 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

3

Slide 4

Vision Cone

Point B
x2,y2

Point C
(x3,y3)

Point A
x1,y1

- Cross-product is equivalent to *determinant* of a 2D matrix:
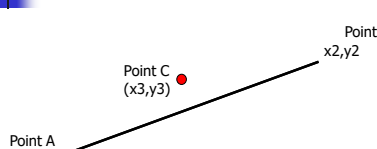
$$| \; x2\text{-}x1 \quad x3\text{-}x1 \; |$$
$$| \; y2\text{-}y1 \quad y3\text{-}y1 \; |$$

i.e. $((x2\text{-}x1) * (y3\text{-}y1)) - ((y2 - y1) * (x3\text{-}x1))$

4

Slide 5

Vision Cone

Point B
x2,y2

Point C
(x3,y3)

Point A
x1,y1

If result of $((x2\text{-}x1) * (y3\text{-}y1)) - ((y2 - y1) * (x3\text{-}x1))$

- == 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)

5

Slide 6

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;
}
```
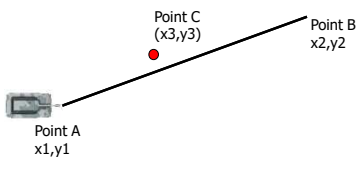
6

1

Vision Cone

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

Point C
(x3,y3)

Point B
x2,y2

Point A
x1,y1

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

7