

2d Raycasting

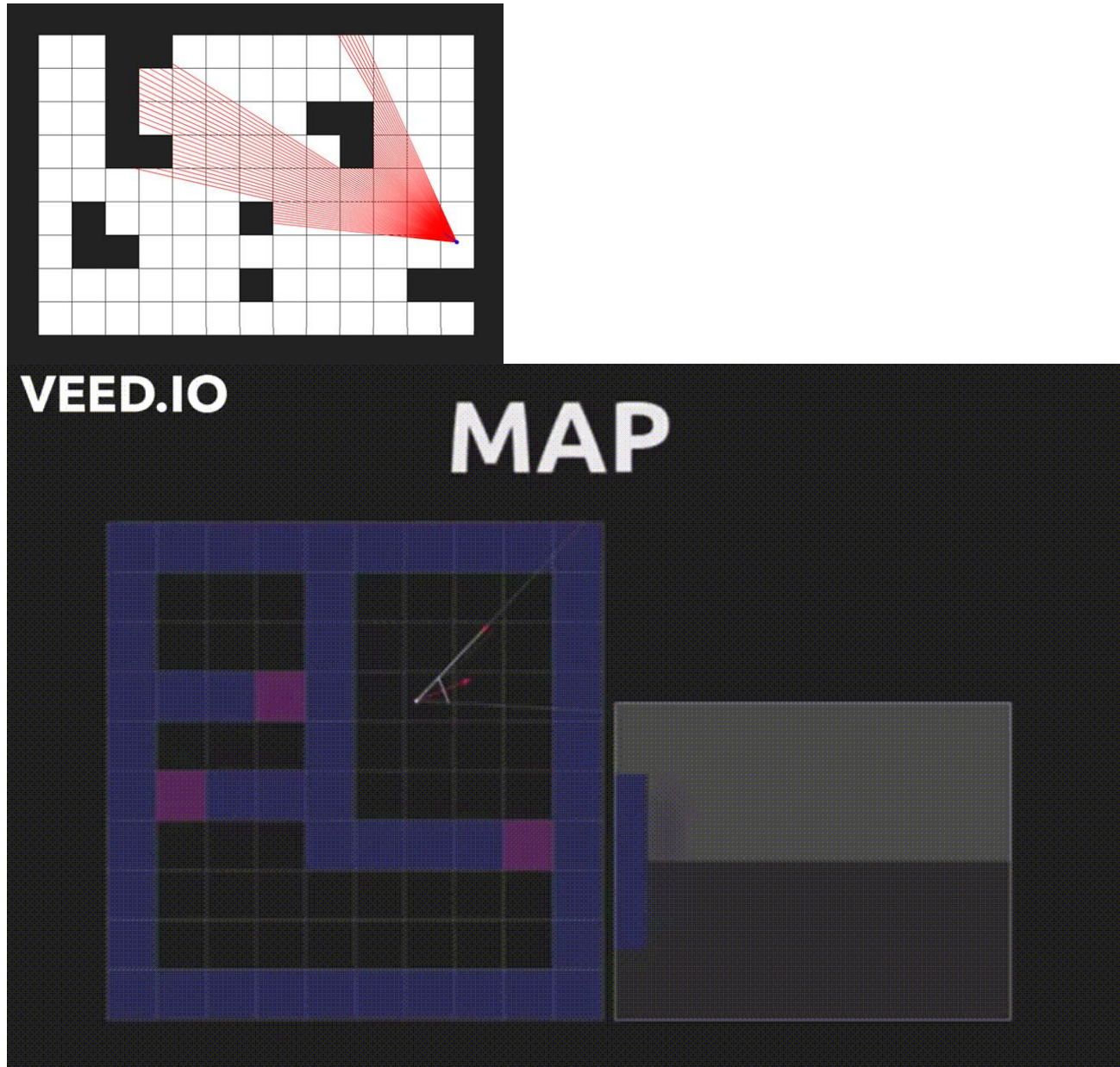
## Main resources

<https://www.youtube.com/watch?v=eOCQfxRQ2pY> => some illustration

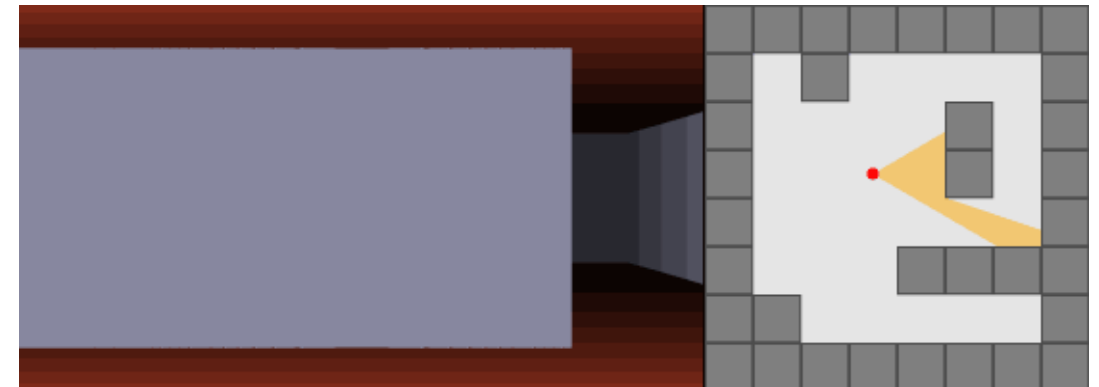
<https://lodev.org/cgtutor/raycasting.html> => based on this documentation

Need Pre knowledge : basic vectors, matrix, math

The basic idea of raycasting is as follows: the map is a 2D square grid, and each square can either be 0 (= no wall), or a positive value (= a wall with a certain color or texture).



send out a **ray** that starts at the player location and with a direction that depends on both the player's looking direction, and the x-coordinate of the screen. Then, let this ray move forward on the 2D map, until it hits a map square that is a wall. If it hit a wall, **calculate the distance of this hit point to the player**, and **use this distance to calculate how high this wall** has to be drawn on the screen: the **further away the wall, the smaller it's on screen**, and the **closer, the higher it appears to be**. These are all 2D calculations.



First we have to define how we represent the player and the rays

The blue line is the camera plane and it should be always perpendicular on the direction vector of the player. The camera plane represents the surface of the computer screen.

Since the camera plane is 1D and the computer screen is 2D, we need an extra dimension, this extra dimension is obtained by counting on the ray length (which is the distance from the player to the wall following the direction of the ray).

- \* The player is represented by the vector Pos
- \* The direction on which the player look represented by the vector Dir
- \* The Plane of the camera is represented by the vector Plane

\* moving the player is performed by changing the Vector Pos

if we move  $\updownarrow$  (W,S)

$$pos.x + = dir.x * Speed * Direction$$

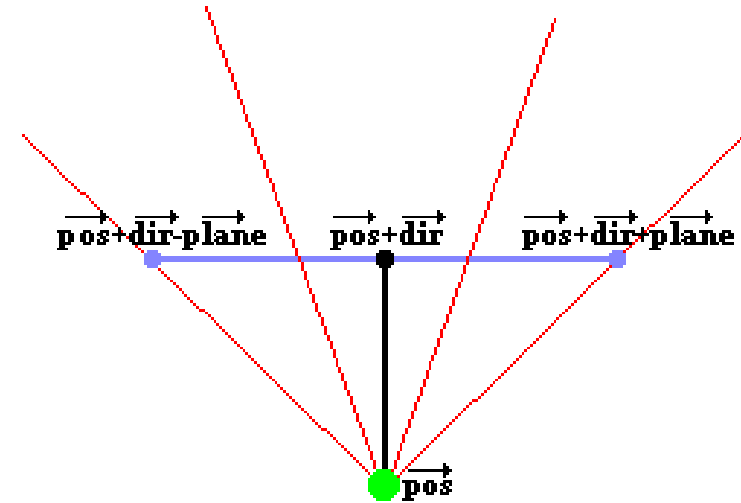
$$pos.y + = dir.y * Speed * Direction$$

if we move  $\leftrightarrow$  (A,D)

$$pos.x + = -dir.y * Speed * Direction$$

$$pos.y + = dir.x * Speed * Direction$$

Direction can be  $-1$  or  $+1$  depending on (W,S,A,D).



Rotating the Player is rotating the vector Dir.

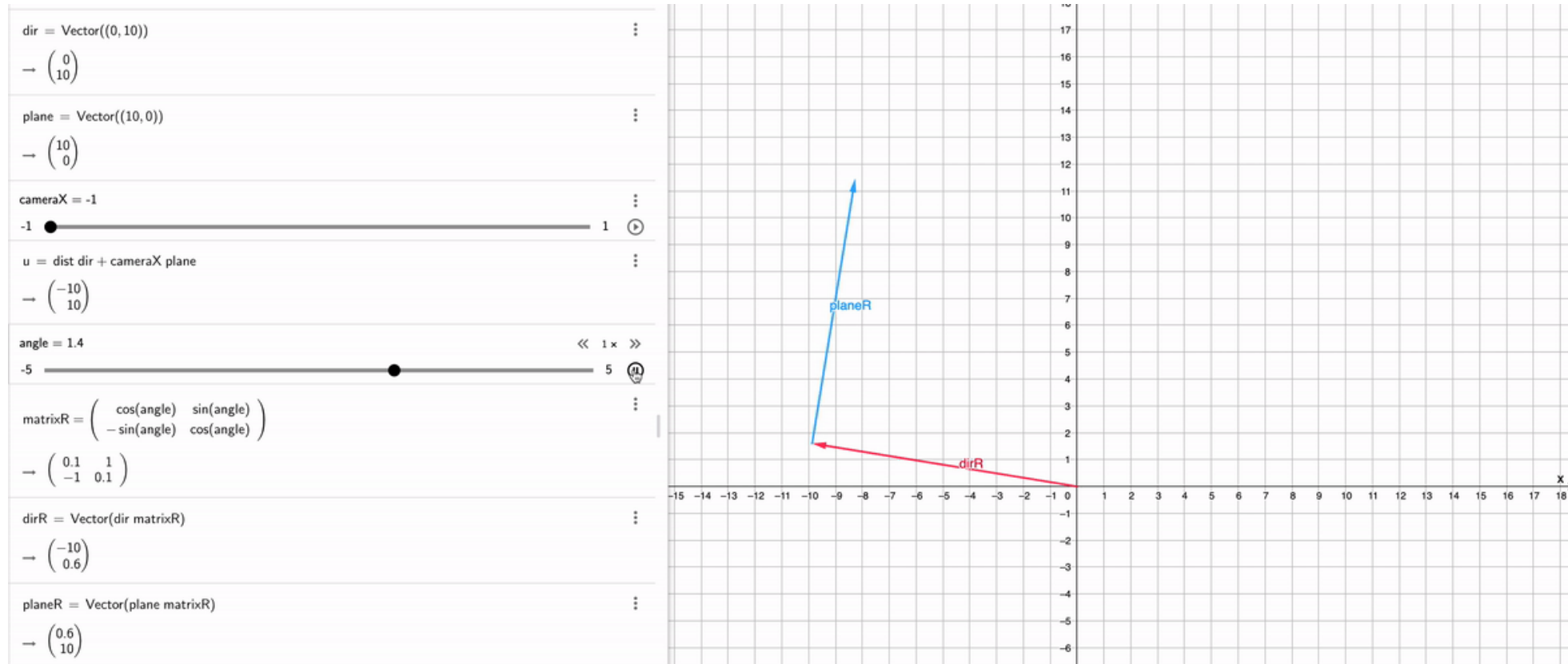
Obviously the vector Plane should be always  $\perp$  To Dir

to rotate the vector Dir or Plane by a specific angle we have to multiplay each one of them by the 2d rotation matrix.

$$\text{rotationMatrix} = \begin{pmatrix} \cos(\text{angle}) & \sin(\text{angle}) \\ -\sin(\text{angle}) & \cos(\text{angle}) \end{pmatrix}$$

$$\text{Dir} = \text{Dir} * \text{rotationMatrix}$$

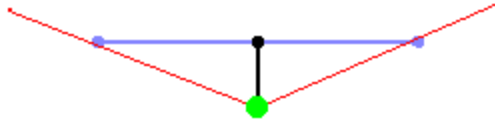
$$\text{Plane} = \text{Plane} * \text{rotationMatrix}$$



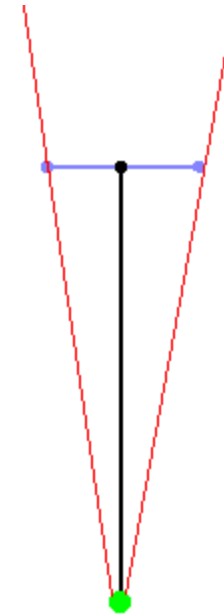
The two red lines, are the left and right border of the screen.

And the angle between those two lines is called the **Field Of Vision or FOV**.

The FOV is determined by the ratio of the **length Dir vector**, and the length of the **Plane vector**.



the FOV will be larger than  $90^\circ$  ( $180^\circ$  is the maximum, if the direction vector is close to 0), and you'll have a much wider vision, like zooming out



the FOV will be much smaller than  $90^\circ$ , and you'll have a very narrow vision. You'll see everything more detailed though and there will be less depth, so this is the same as zooming in

if we want  $66^\circ$  field of view:

$$\tan(\alpha) = \frac{x}{|dir|} = \frac{x}{1} = x \text{ with } \alpha = 33^\circ$$

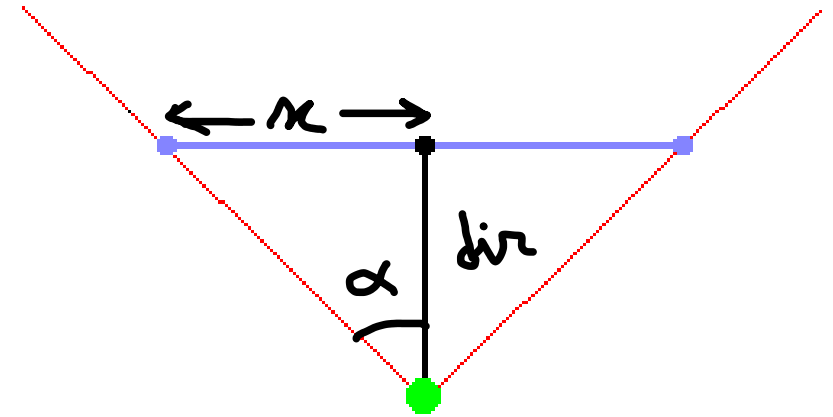
initialization of pos, dir and plane

$$pos = \begin{pmatrix} pos\ x\ of\ player\ in\ map \\ pos\ y\ of\ player\ in\ map \end{pmatrix}$$

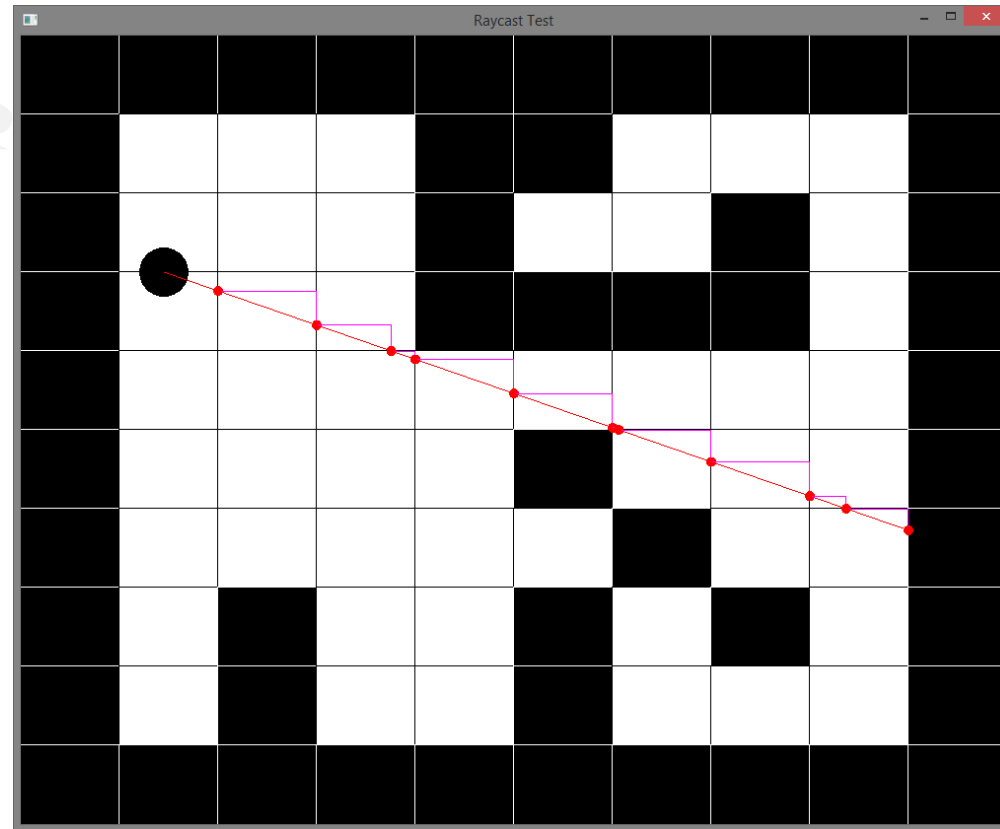
$$dir = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

$$plane = \begin{pmatrix} 0 \\ 0.65 = \tan(33^\circ) \end{pmatrix}$$

of course you have to *rotate the vectors dir and plane* based on the initial player direction eq: North or South or East or West



How to calculate the distance between the player and the wall for each ray

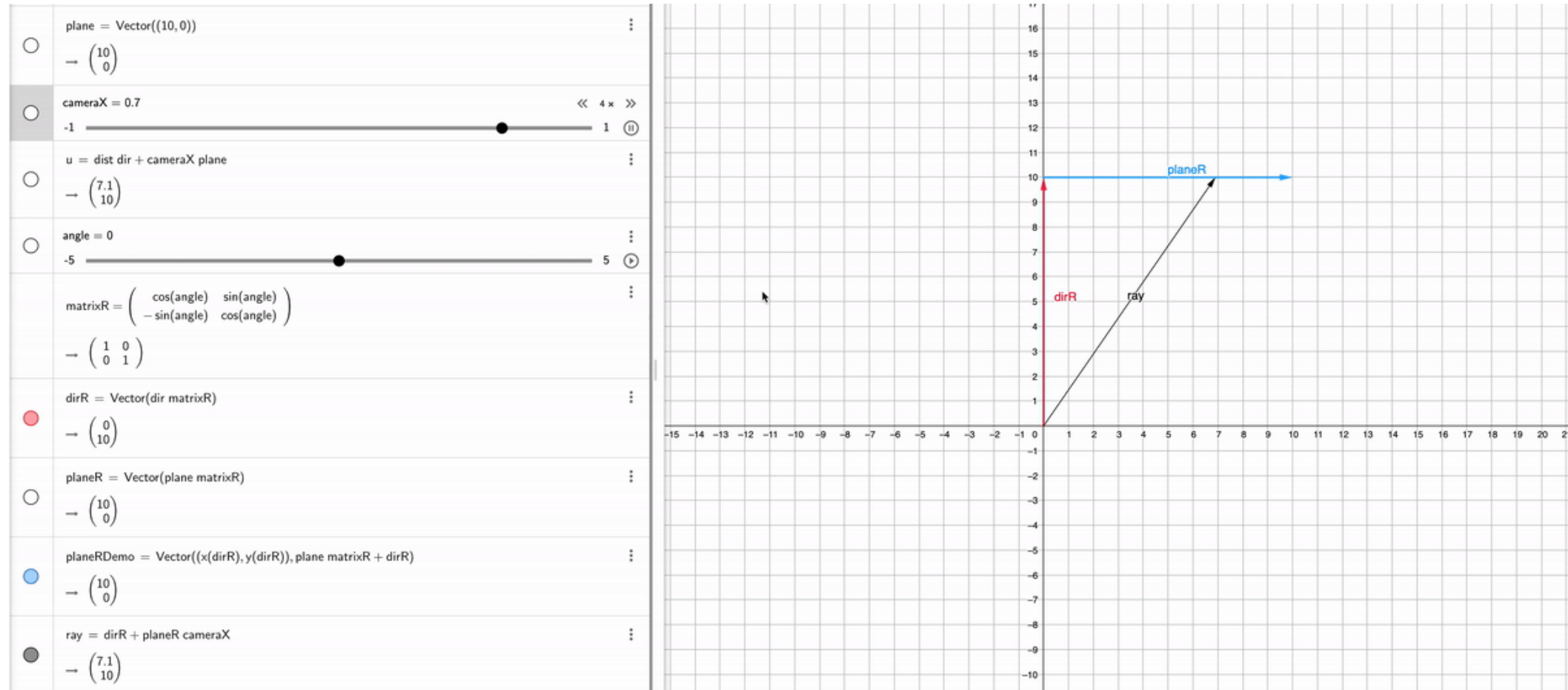




- \* First we have to know how to produce the rays on each strip X of the screen.
- \* if we have a window (eq screen) with 500px WIDTH, that means that we have 500 strip.

*rayDir is defined as*

*rayDir = dir + cameraX \* plane Where cameraX start from -1 to +1 to scan all the plane from left to right*



Relation  
between X  
strip and  
cameraX

$$x \quad 0 \rightarrow +W$$

$$2x \quad 0 \rightarrow +2W$$

$$2x - W \quad -W \rightarrow W$$

$$\frac{2x - W}{W} \quad -1 \rightarrow +1$$

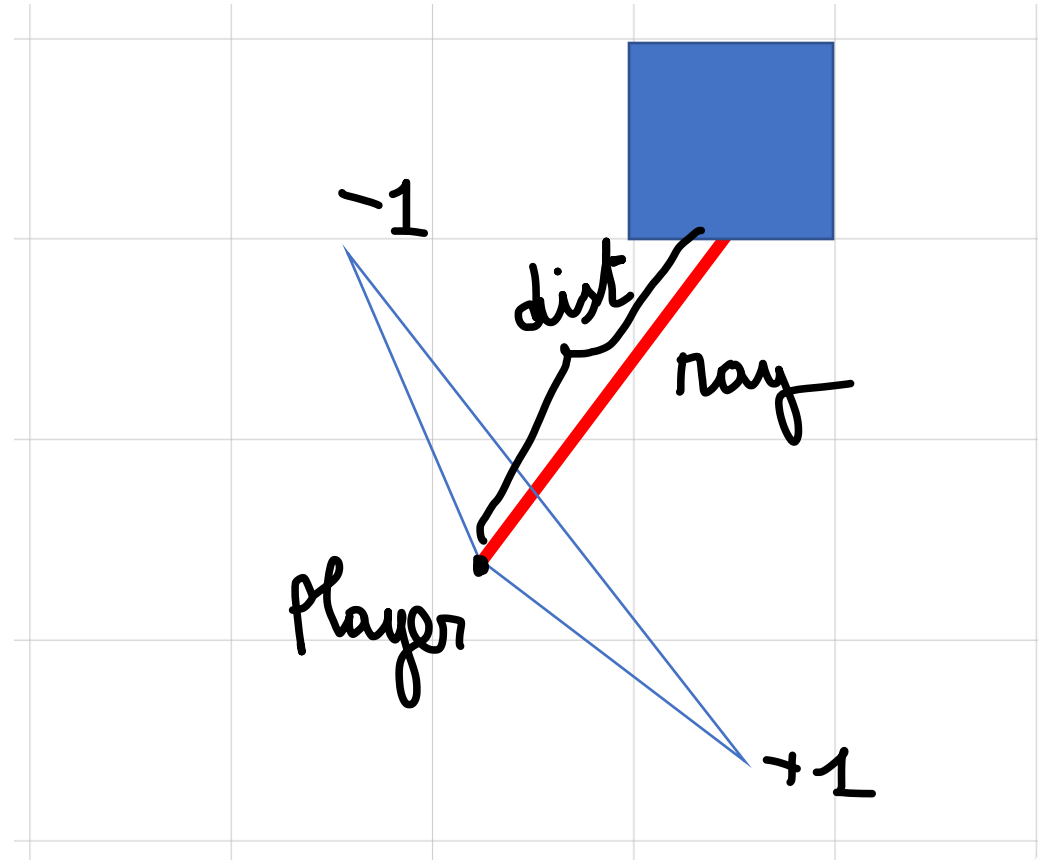
$$\text{cameraX} = \frac{2x - W}{W}$$

Basically we make X go from 0 to Width and calculate cameraX and finally we got our ray.

$$\text{cameraX} = \frac{2x - W}{W}$$

$$\text{rayDir} = \text{dir} + \text{cameraX} * \text{plane}$$

We now need to calculate the distance between the player and the wall in each ray

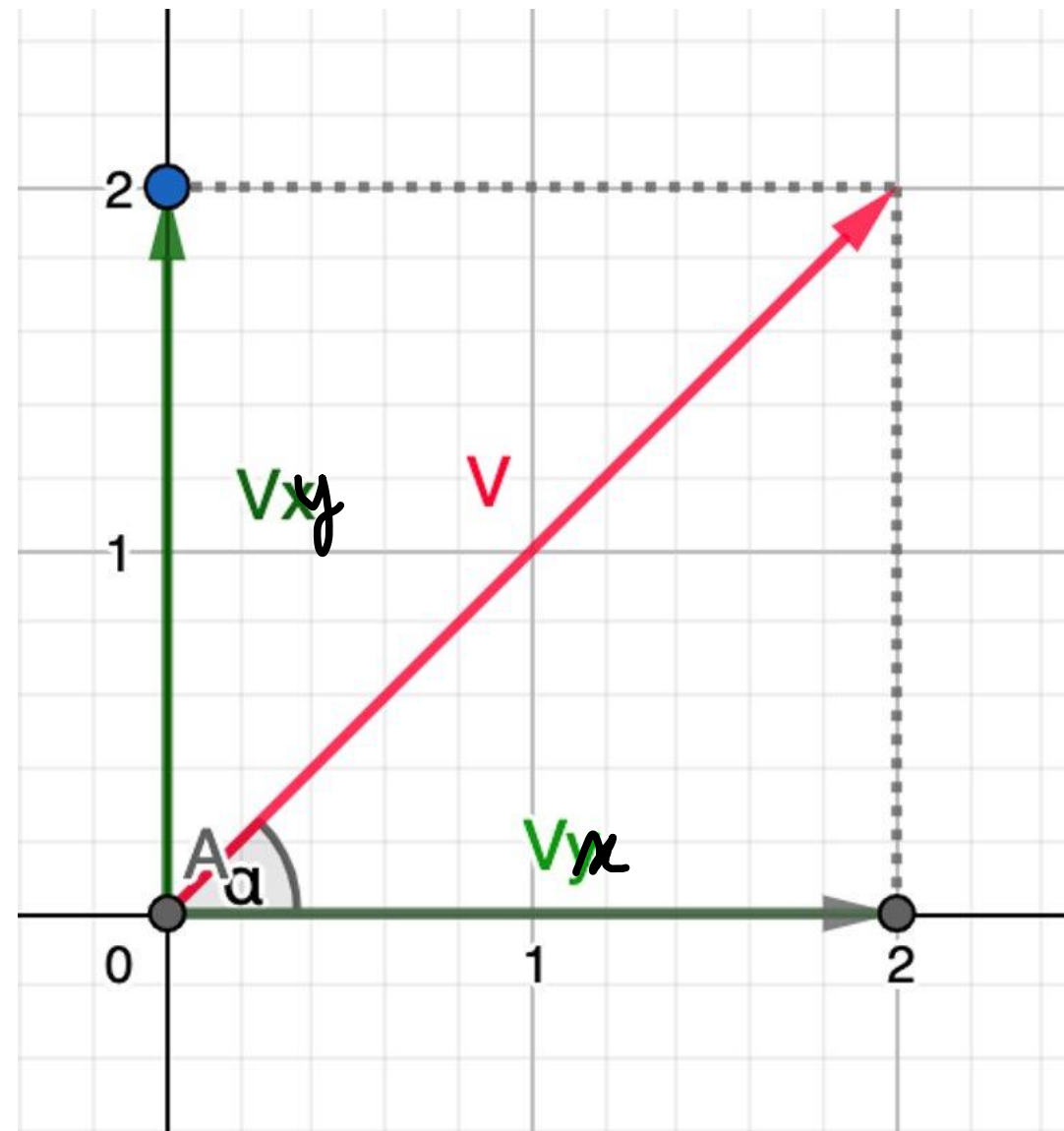


*Recall :*

1.  $\sqrt{x^2} = |x|$  where  $x \in \mathbb{R}$

2.  $\|\vec{v}\| = \sqrt{v_x^2 + v_y^2}$

3.  $\tan(\alpha) = \frac{v_y}{v_x}$



calculate if the ray pass through out one unit of y

we have from the right image:

$$1. \delta Y^2 = 1^2 + x^2$$

$$2. \tan(\alpha) = \frac{\text{raydir.y}}{\text{raydir.x}} = \frac{1}{x}$$

From 1 and 2:

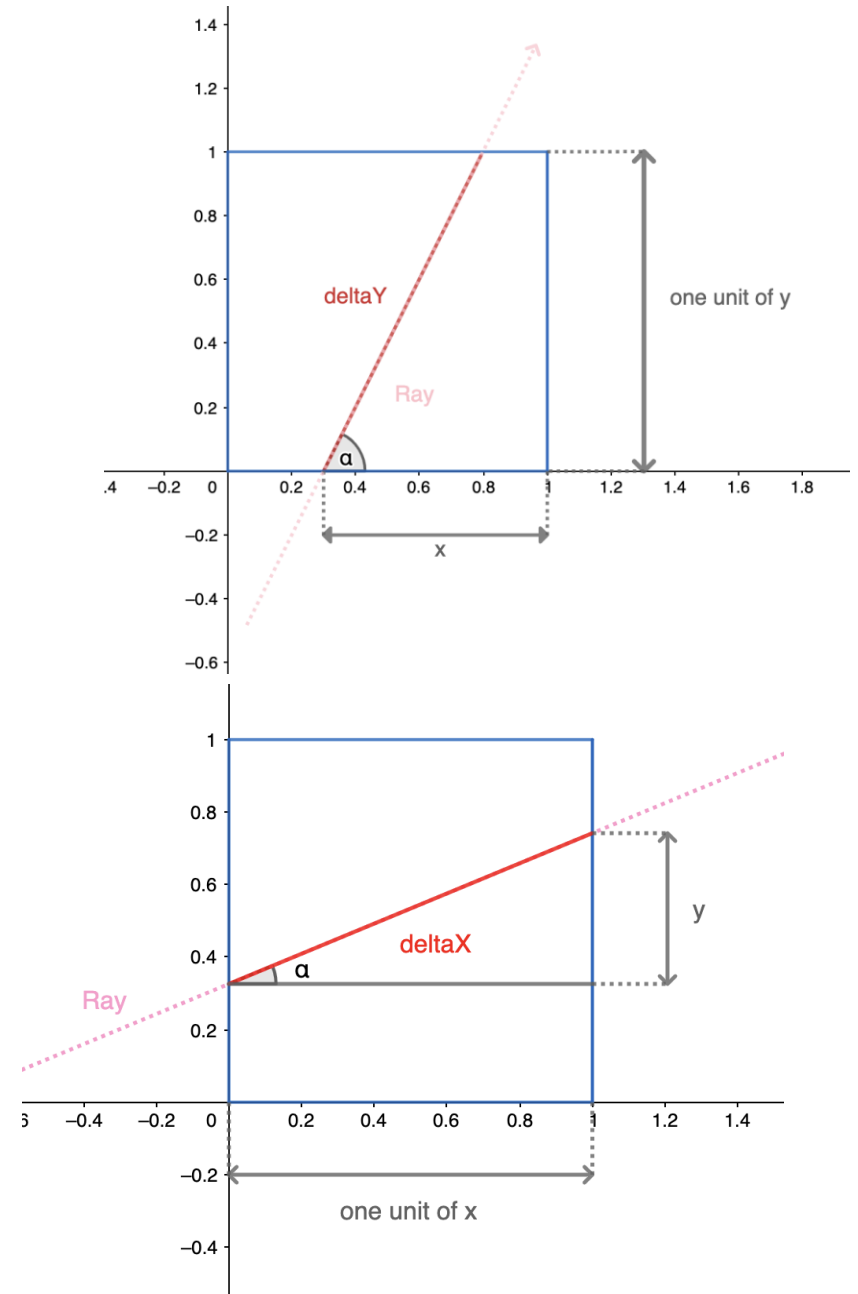
$$\Rightarrow \delta Y^2 = 1^2 + \left(\frac{\text{raydir.x}}{\text{raydir.y}}\right)^2$$

$$\Rightarrow \delta Y^2 = \frac{\text{raydir.x}^2 + \text{raydir.y}^2}{\text{raydir.y}^2}$$

$$\Rightarrow \delta Y = \frac{\|\text{raydir}\|}{|\text{raydir.y}|}$$

same calculation goes from  $\delta X$  where the ray pass through out one unit of x

$$\Rightarrow \delta X = \frac{\|\text{raydir}\|}{|\text{raydir.x}|}$$



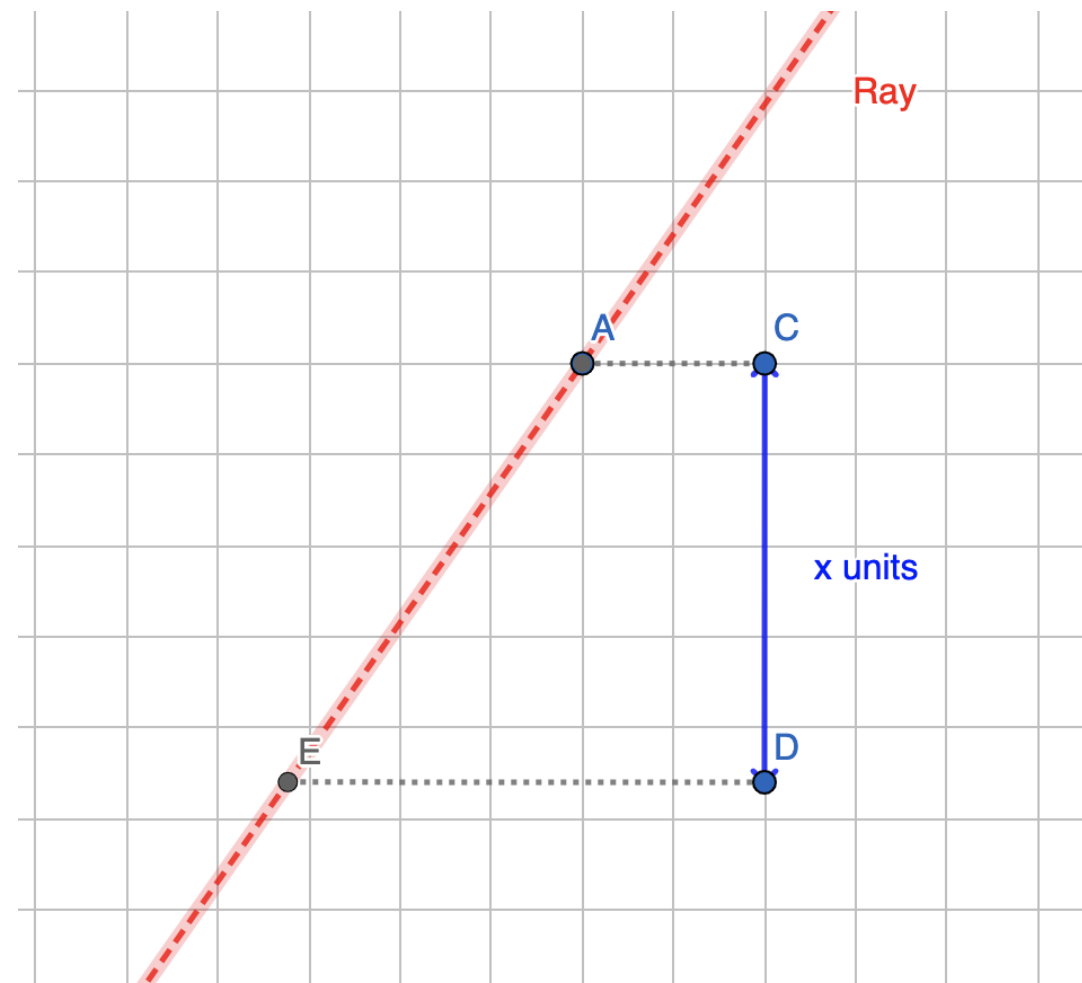
$\delta Y \rightarrow$  one unit of  $Y$

$\Delta Y \rightarrow$   $x$  unit of  $Y$

$$\Delta Y = \frac{x \cdot \delta Y}{1} = x \cdot \delta Y$$

the same for  $\Delta X$  for  $x$  unit of  $X$

$$\Delta X = \frac{y \cdot \delta X}{1} = x \cdot \delta X$$



## Summary

*\* We can use 1 instead of  $\| raydir \|$  because only the ratio between  $\delta X$  and  $\delta Y$  matters for the DDA algo.*

$$\Delta X = \frac{y \cdot \delta X}{1} = x \cdot \delta X$$

$$\delta X = \frac{1}{|raydir.x|}$$

$$\delta Y = \frac{1}{|raydir.y|}$$

$$\delta X' = \frac{\| raydir \|}{|raydir.x|}$$

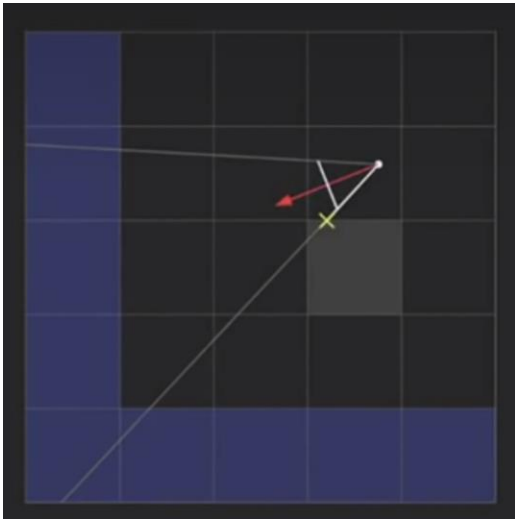
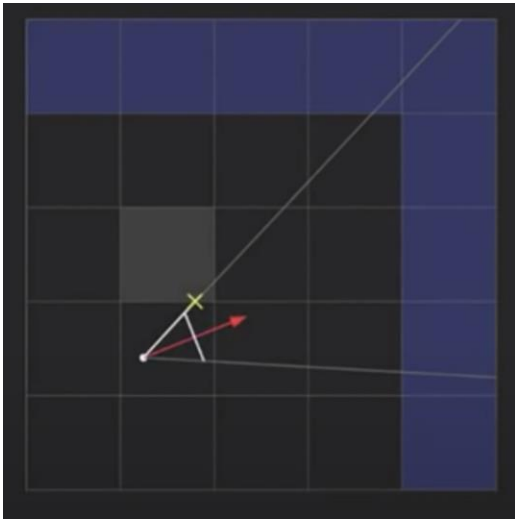
$$\delta Y' = \frac{\| raydir \|}{|raydir.y|}$$

$\Delta X = x \cdot \delta X$  *x is the horizontal distance between the wall and the player*

$\Delta Y = y \cdot \delta Y$  *y is the vertical distance between the wall and the player*

$$\Delta X' = x \cdot \delta X'$$

$$\Delta Y' = y \cdot \delta Y'$$



Initialize delta X and delta Y depending on the direction of the ray

```
ray.mapX = (int)player.x
```

```
ray.mapY = (int)player.y
```

if raydir.y || raydir.x > 0 player look above ↑ or to the right →

$$\Delta x = ((\text{int})\text{player.x} + 1 - \text{player.x}) * \delta x$$

$$\Delta y = ((\text{int})\text{player.y} + 1 - \text{player.y}) * \delta y$$

else raydir.y || raydir.x < 0 player look down ↓ or to the left ←

$$\Delta x = (\text{player.x} - (\text{int})\text{player.x}) * \delta x$$

$$\Delta y = (\text{player.y} - (\text{int})\text{player.y}) * \delta y$$

Notice: you have to split the if into two distinct if . If for x and if for y

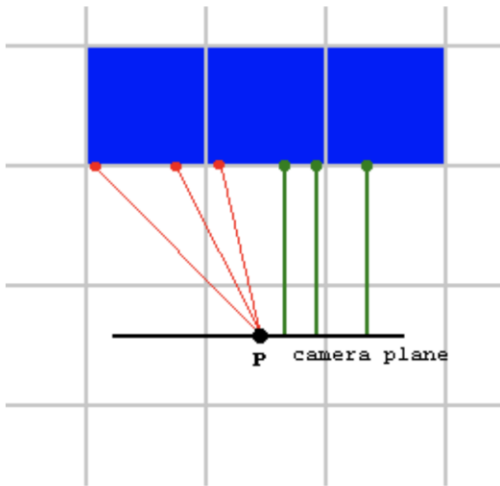


```

function dda()
{
→ //mapX and mapY will be the start position of the ray
→ //stepX and stepY will be -1 or +1 depending in rayDir.x and rayDir.y
→ //calculate  $\delta x$  and  $\delta y$ 
→ //calculate  $\Delta x$  and  $\Delta y$ 
→ while(1)
→ {
→     if( $\Delta x \leq \Delta y$ )
→     {
→          $\Delta x += \delta x$ 
→         mapX += stepX
→         side = 0 //represent R or W
→     }
→     else
→     {
→          $\Delta y += \delta y$ 
→         mapY += stepY
→         side = 1 //represent N or S
→     }
→     if(hitWall())
→     {
→         //the DDA steps above,,, we went one step further to end up inside the wall.
→         if(side == 0)
→         {
→             distance =  $\Delta x - \delta x$ ;
→         }
→         else
→         {
→             distance =  $\Delta y - \delta y$ ;
→         }
→         break;
→     }
→ }
→ }
}

```

In the image, the player is looking directly at the wall, and in that case you would expect the wall's bottom and top to form a perfectly horizontal line on the screen. However, the red rays all have a different length, so would compute different wall heights for different vertical stripes, hence the rounded effect. The green rays on the right all have the same length, so will give the correct result. The same still applies for when the player rotates (then the camera plane is no longer horizontal and the green lines will have different lengths, but still with a constant change between each) and the walls become diagonal but straight lines on the screen. This explanation is somewhat handwavy but gives the idea.



Note that this part of the code isn't "fisheye correction", such a correction isn't needed for the way of raycasting used here, the fisheye effect is simply avoided by the way the distance is calculated here. It's even easier to calculate this perpendicular distance than the real distance, we don't even need to know the exact location where the wall was hit.



We will using:

$$1. E = \Delta Y' = yDist \cdot \delta Y'$$

$$2. \Delta Y = yDist \cdot \delta Y$$

$$3. \delta Y' = \frac{\|rayDir\|}{|rayDir.y|}$$

$$4. \delta Y = \frac{1}{|rayDir.y|}$$

$$5. \|dir\| = 1$$

in PED and PAH triangles we have using Thales's theorem.

$$\Rightarrow \frac{AH}{ED} = \frac{PH}{PD}$$

$$\Rightarrow \frac{perWallDist}{\|dir\|} = \frac{E}{\|rayDir\|}$$

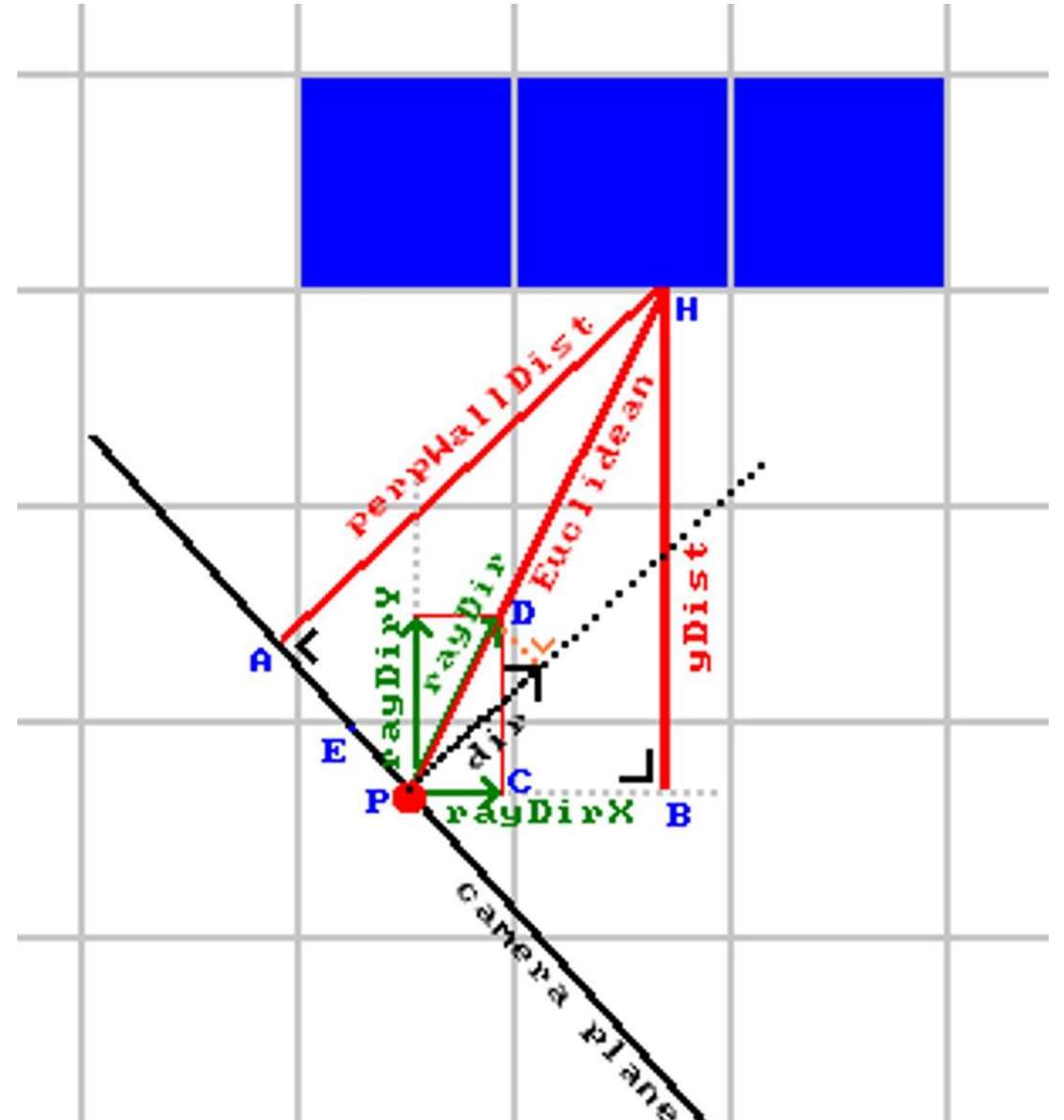
$$\Rightarrow perWallDist = \frac{\delta Y' \cdot yDist}{\|rayDir\|} \quad (1) \text{ and } (5)$$

$$\Rightarrow perWallDist = \frac{\frac{\|rayDir\|}{|rayDir.y|} \cdot yDist}{\|rayDir\|} \quad (3)$$

$$\Rightarrow perWallDist = \frac{yDist}{|rayDir.y|}$$

$$\Rightarrow perWallDist = yDist \cdot \delta Y \quad (4)$$

$$\Rightarrow perWallDist = \Delta Y \quad (2) \text{ calculated usign the dda (distance in the code above)}$$



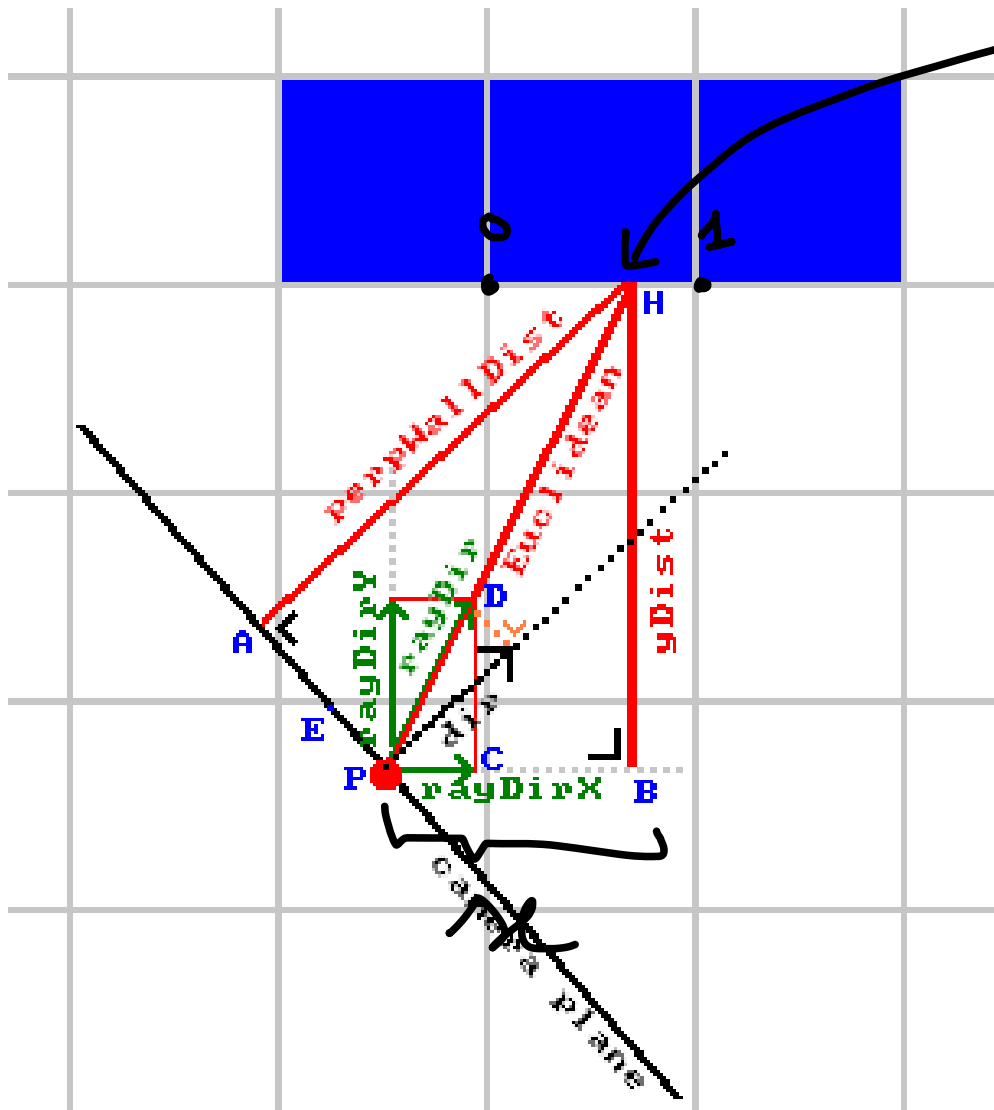
*if the ray hit the side  $W$  or  $S$*

$$\textit{perWallDist} = \Delta y$$

*else  $E$  or  $E$*

$$\textit{perWallDist} = \Delta x$$

The final step is to calculate the wall\_x



Wall\_x: where the ray hit the wall

if the ray hit the side W or S

$$\text{Wallx} = \text{player.x} + x$$

else W or E

$$\text{Wallx} = \text{player.y} + y$$

.

We will treat this

$$1.perWallDist = \frac{E}{||rayDir||}$$

$$3.\delta Y' = \frac{\| rayDir \|}{|rayDir.y|}$$

$$\Rightarrow \tan(\alpha) = \frac{rayDir.y}{rayDir.x} = \frac{yDist.\vec{j}}{x}$$

\* we will consider only the distance

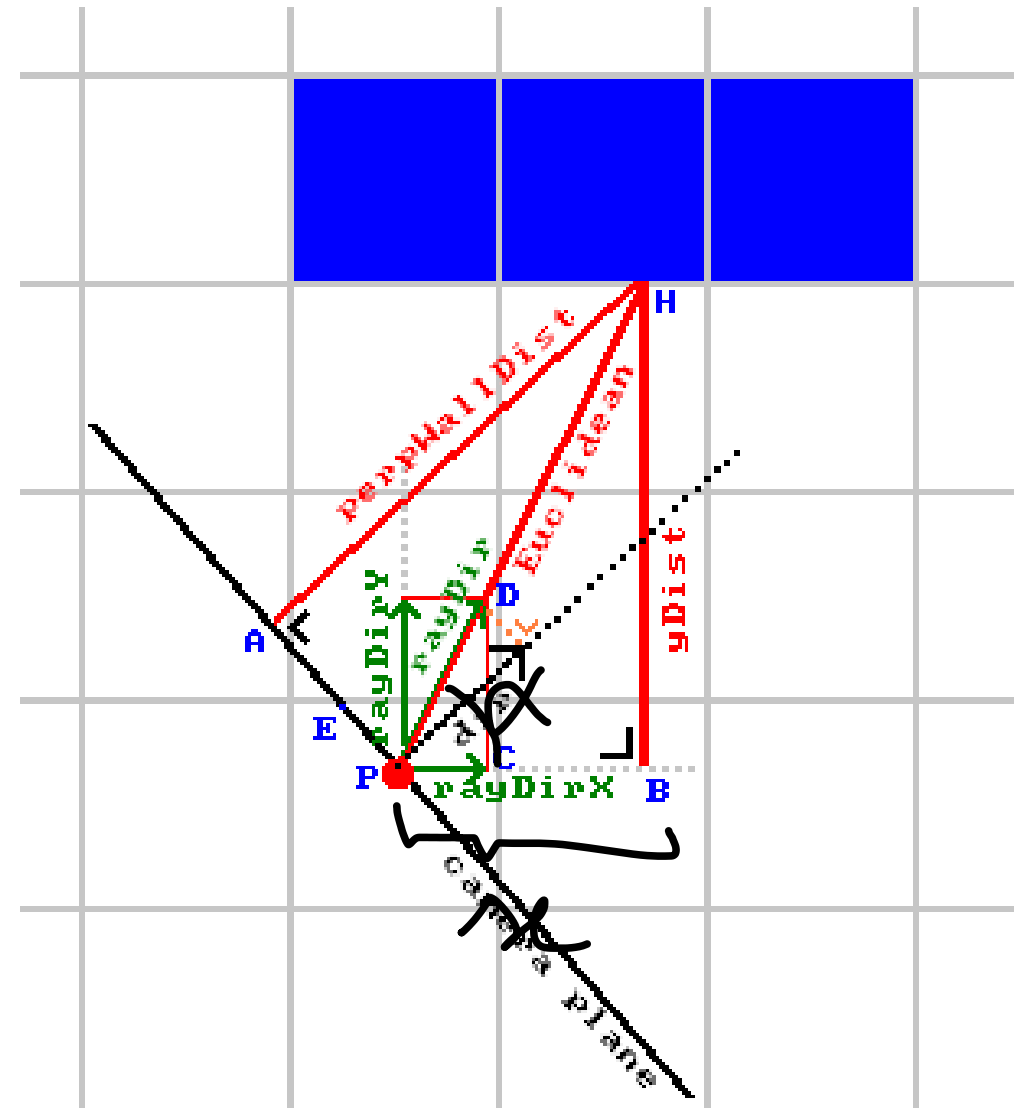
$$\Rightarrow \tan(\alpha) = \frac{|rayDir.y|}{rayDir.x} = \frac{yDist}{x}$$

$$\Rightarrow x = yDist. \frac{rayDir.x}{|rayDir.y|}$$

$$\Rightarrow x = \frac{E}{\delta Y'} \cdot \frac{rayDir.x}{|rayDir.y|} \quad (2)$$

$$\Rightarrow x = E \cdot \frac{|\cancel{rayDir.y}|}{\|rayDir\|} \cdot \frac{rayDir.x}{|\cancel{rayDir.y}|} \quad (3)$$

the same goes for  $y$ :  $y = perWallDist * rayDir.y$





$$1.perWallDist = \frac{E}{||rayDir||}$$

$$2. E = yDist. \delta Y'$$

$$3. \delta Y' = \frac{\| rayDir \|}{|rayDir.y|}$$

*we Have From the image:*

$$\Rightarrow \tan(\alpha) = \frac{rayDir.y}{rayDir.x} = \frac{yDist.\vec{j}}{x}$$

\* Notice if  $rayDir.y < 0$  that means that  $yDist.\vec{j}$  is also  $< 0$  and vice versa

\* we will consider only the distance

$$\Rightarrow \tan(\alpha) = \frac{|rayDir.y|}{rayDir.x} = \frac{yDist}{x}$$

$$\Rightarrow x = yDist. \frac{rayDir.x}{|rayDir.y|}$$

$$\Rightarrow x = \frac{E}{\delta Y'} \cdot \frac{rayDir.x}{|rayDir.y|} \quad (2)$$

$$\Rightarrow x = E \cdot \frac{|\cancel{rayDir.y}|}{\|rayDir\|} \cdot \frac{rayDir.x}{|\cancel{rayDir.y}|} \quad (3)$$

$$\Rightarrow x = perWallDist * rayDir.x \quad (1)$$

the same goes for  $y$ :  $y = perWallDist * rayDir.y$



*if the ray hit the side W or S*

$$\text{perWallDist} = \Delta y$$

*else E or E*

$$\text{perWallDist} = \Delta x$$

*if the ray hit the side N or S*

$$\text{Wallx} = \text{player.x} + \text{perWallDist} \cdot \text{rayDir.x}$$

*else E or W*

$$\text{Wallx} = \text{player.x} + \text{perWaDist} \cdot \text{rayDir.x}$$

$$\text{Wallx} = \text{Wallx} - (\text{int})\text{Wallx}$$

# Getting the color from the texture

let  $x$  and  $y$  be coordinate in the texture

we have

$$x = texWidth * wallX; \text{ since } 0 \leq wallX \leq 1$$

$$y = \frac{a}{lineHeight} * texHeight; \text{ } a \text{ going from } 0 \text{ to } lineHeight \quad 0 \leq \frac{a}{lineHeight} \leq 1$$

$$lineHeight = \frac{HeightOfTheScreen}{perWallDist}$$



## Floor and ceiling



# Floor and ceiling

<https://www.geogebra.org/classic/e8fgsehy>

from the image we have  $\frac{1}{\text{distance}} = \frac{\text{Height}/2 - Y}{\text{Height}/2}$

$$\Rightarrow \text{distance} = \frac{\text{Height}/2}{\text{Height}/2 - Y}$$

Y goes from 0 to Height / 2



# Floor and ceiling

*every ray is composed as follow:  $ray = dir + cameraX * plane$*

*get the X coordinate on the floor :  $X = player.x + distance * ray.x$*

*$\Rightarrow X = player.x + distance * (dir.x + cameraX * plane.x)$*

*we have cameraX goes from -1 to +1 when screenX goes from 0 to Width*

*$\Rightarrow cameraX = \frac{2.screenX - ScreenWidth}{ScreenWidth}$*

$$X = player.x + distance * \left( dir.x + \frac{2.screenX - ScreenWidth}{ScreenWidth} * plane.x \right)$$

$$X = player.x + distance * \left( dir.x - plane.x + plane.x * \frac{2.screenX}{ScreenWidth} \right)$$

$$X = player.x + distance * (dir.x - plane.x) + distance * (plane.x * \frac{2.screenX}{ScreenWidth}) \text{ screenX goes from 0 to ScreenWidth.}$$

# Floor and ceiling

*Summary {*

$$X = \text{player}.x + \text{distance} * (\text{dir}.x - \text{plane}.x) + \text{distance} * \left( \text{plane}.x * \frac{2.\text{screen}X}{\text{ScreenWidth}} \right)$$

$$Y = \text{player}.y + \text{distance} * (\text{dir}.y - \text{plane}.y) + \text{distance} * \left( \text{plane}.y * \frac{2.\text{screen}X}{\text{ScreenWidth}} \right)$$

$$\text{distance} = \frac{\text{Height} / 2}{\text{Height} / 2 - Y} \quad Y \quad 0 \rightarrow \text{Height} / 2$$

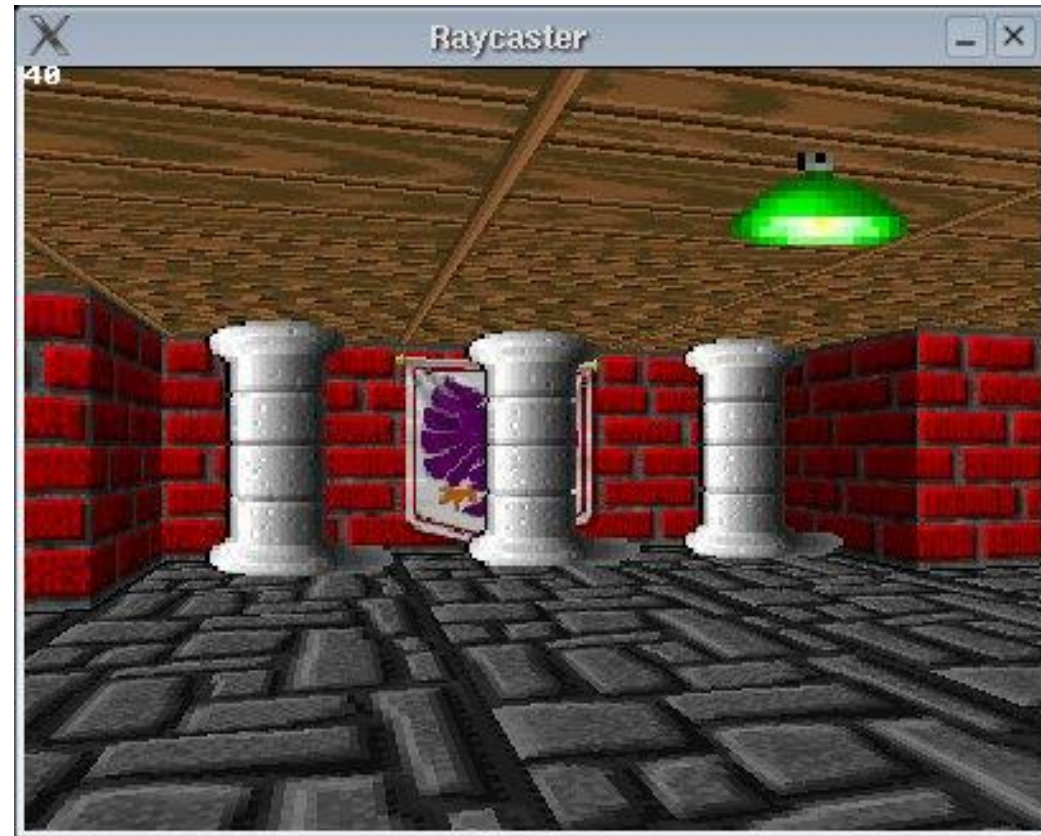
$$\text{tex}X = (X - (\text{int})X) * \text{tex}.width$$

$$\text{tex}Y = (Y - (\text{int})Y) * \text{tex}.Height$$

*}*



## Sprites



# Sprites

if we want to map every point in the 2d world using the dir and plane vectors we will use this formula:

$$\text{cameraMatrix} * 2dVector = \begin{bmatrix} \text{plane}.x & \text{dir}.x \\ \text{plane}.y & \text{dir}.y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x.\text{plane}.x + y.\text{dir}.x \\ x.\text{plane}.y + y.\text{dir}.y \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix}$$

$x, y \in \mathbb{R}$

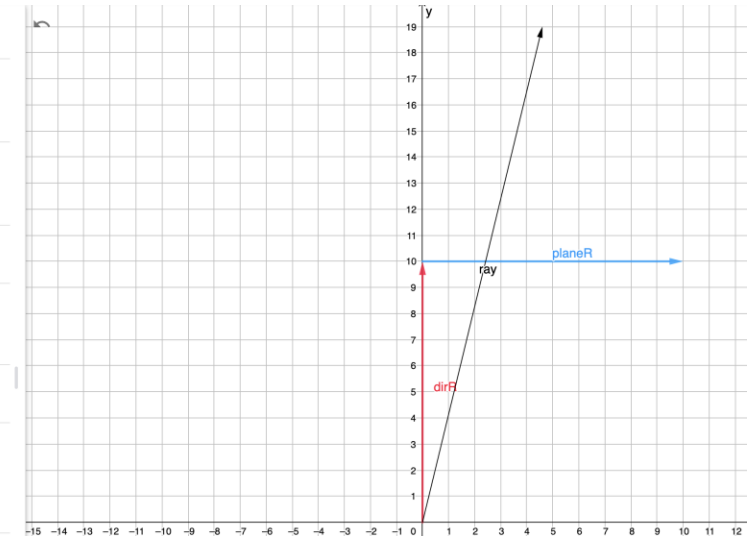
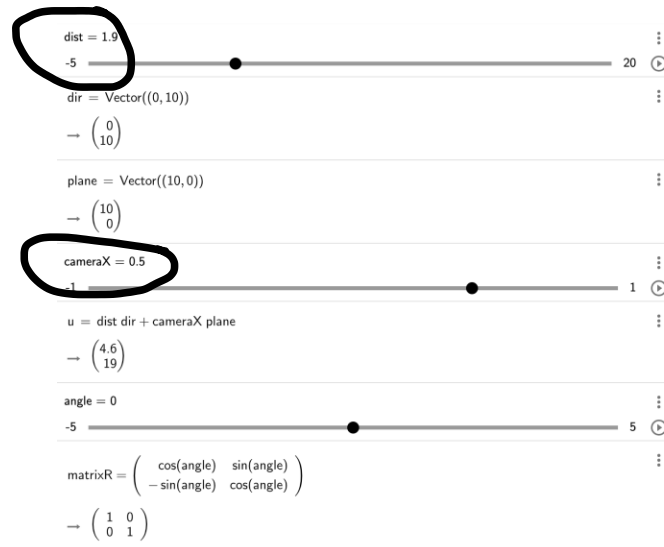
using  $x$  and  $y$  values we can map all the points in the 2D world using the plane and dir vectors.

<https://www.geogebra.org/calculator/w5pd57rm>

CameraX represent  $x$ .

Dist represent  $y$ .

You can play with value of cameraX and Dist in the above link to get an idea.





# Sprites

*what we know about the sprites is its position in 2D world (the 2d map).*

*since we have the coordinate  $X, Y$  of the sprite we have to find  $(x, y)$  coordinates for that we will use the inverse matrix of the cameraMatrix.*

$$\text{cameraMatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix}$$

$$\Rightarrow \text{cameraMatrix}^{-1} \cdot \text{cameraMatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \text{cameraMatrix}^{-1} \cdot \begin{bmatrix} X \\ Y \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \text{cameraMatrix}^{-1} \cdot \begin{bmatrix} X \\ Y \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \text{dir.y} & -\text{dir.x} \\ -\text{plane.y} & \text{plane.x} \end{bmatrix} \cdot \frac{1}{\det(\text{cameraMatrix})} \cdot \begin{bmatrix} X \\ Y \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \text{dir.y} & -\text{dir.x} \\ -\text{plane.y} & \text{plane.x} \end{bmatrix} \cdot \frac{1}{\text{plane.x} * \text{dir.y} - \text{dir.x} * \text{plane.y}} \cdot \begin{bmatrix} X \\ Y \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{\text{plane.x} * \text{dir.y} - \text{dir.x} * \text{plane.y}} \cdot \begin{bmatrix} \text{dir.y} * X - \text{dir.x} * Y \\ -\text{plane.y} * X + \text{plane.x} * Y \end{bmatrix}$$

# Sprites

<https://www.geogebra.org/m/w5pd57rm>

we have  $X = 20$  and  $Y = 10$  using the formula above we will get  $y = 2$  and  $x = 1$  (illustration)

looking to the image we have:  $\frac{\text{cameraX}}{x} = \frac{|dir|}{y} \Rightarrow \text{cameraX} = \frac{x}{y}$

we know that cameraX will go from  $-1$  to  $+1$  when screenX will go from  $0$  to Width.

cameraX  $-1 \rightarrow +1$

cameraX  $+1$   $0 \rightarrow 2$

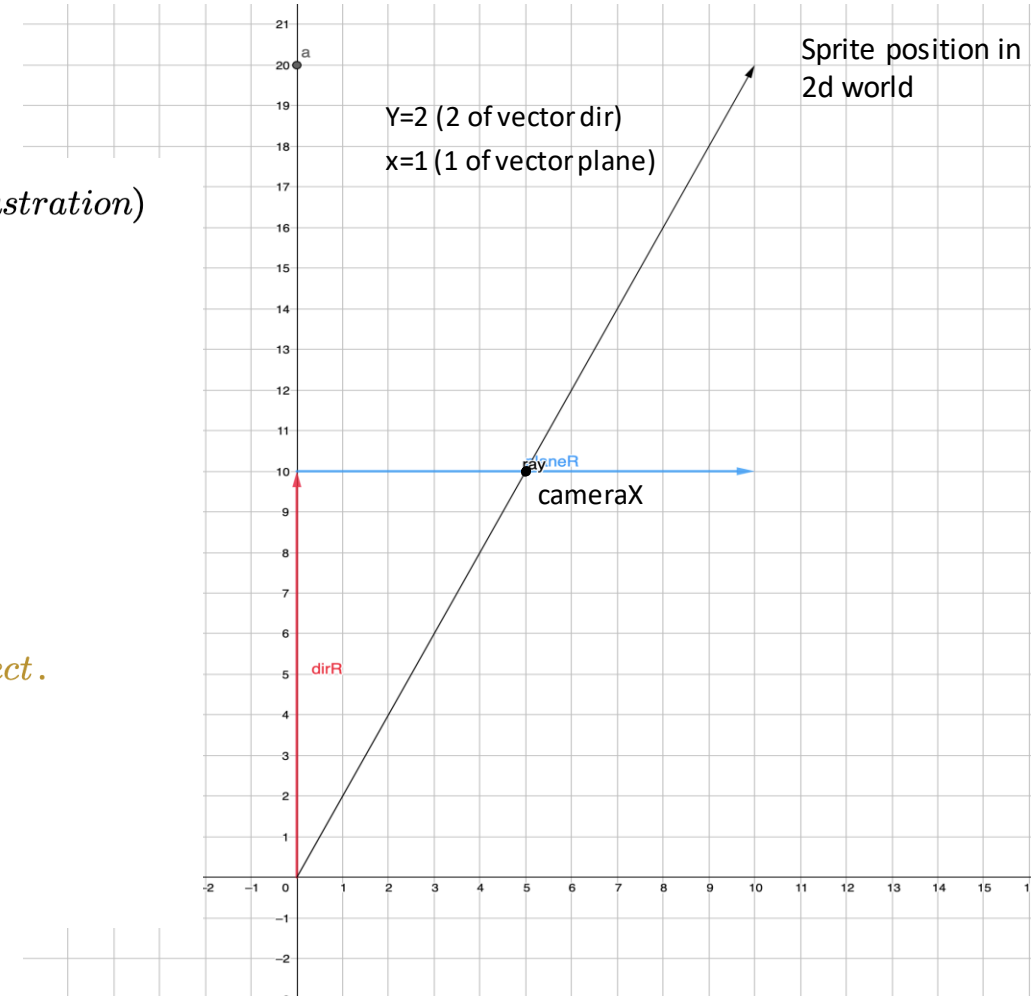
$\frac{\text{Width} * (\text{cameraX} + 1)}{2}$   $0 \rightarrow \text{Width}$

values if  $\text{cameraX} > 1$  ||  $\text{cameraX} < -1$  represent non visible part of the object.

1. so we will have screenX =  $\frac{\text{Width} * (\text{cameraX} + 1)}{2} = \frac{\text{Width} * (\frac{x}{y} + 1)}{2}$

2. the size(with = height) of the sprite in the screen will be  $\frac{\text{ScreenHeight}}{y}$

If the sprite positioned in  $Y=1$  we will have  $\text{SpriteHeight} = \text{ScreenHeight}$   
Because the sprite positioned in the plane



# Sprites

## Final result

1. so we will have  $screenX = \frac{Width * (cameraX + 1)}{2} = \frac{Width * (\frac{x}{y} + 1)}{2}$

2. *the size(with = height) of the sprite in the screen will be  $\frac{ScreenHeight}{y}$*

\*  $spriteScreenStartX = screenX - \frac{size}{2}$

\*  $spriteScreenEndX = screenX + \frac{size}{2}$

\*  $spriteScreenStartY = \frac{ScreenHeight}{2} - \frac{size}{2}$

\*  $spriteScreenEndY = \frac{ScreenHeight}{2} + \frac{size}{2}$

To center the sprite

## Other resources

[https://www.youtube.com/watch?v=NbSee-XM7WA&ab\\_channel=javidx9](https://www.youtube.com/watch?v=NbSee-XM7WA&ab_channel=javidx9) => dda

[https://www.youtube.com/watch?v=DFZnzCbmlng&ab\\_channel=JeremyCodes](https://www.youtube.com/watch?v=DFZnzCbmlng&ab_channel=JeremyCodes) => cub3d

[https://www.youtube.com/watch?v=fNk\\_zzaMoSs](https://www.youtube.com/watch?v=fNk_zzaMoSs) => base vectors

[https://www.youtube.com/watch?v=S0uzwDKqnsW&ab\\_channel=JeffreyChasnov](https://www.youtube.com/watch?v=S0uzwDKqnsW&ab_channel=JeffreyChasnov) => rotation matrix