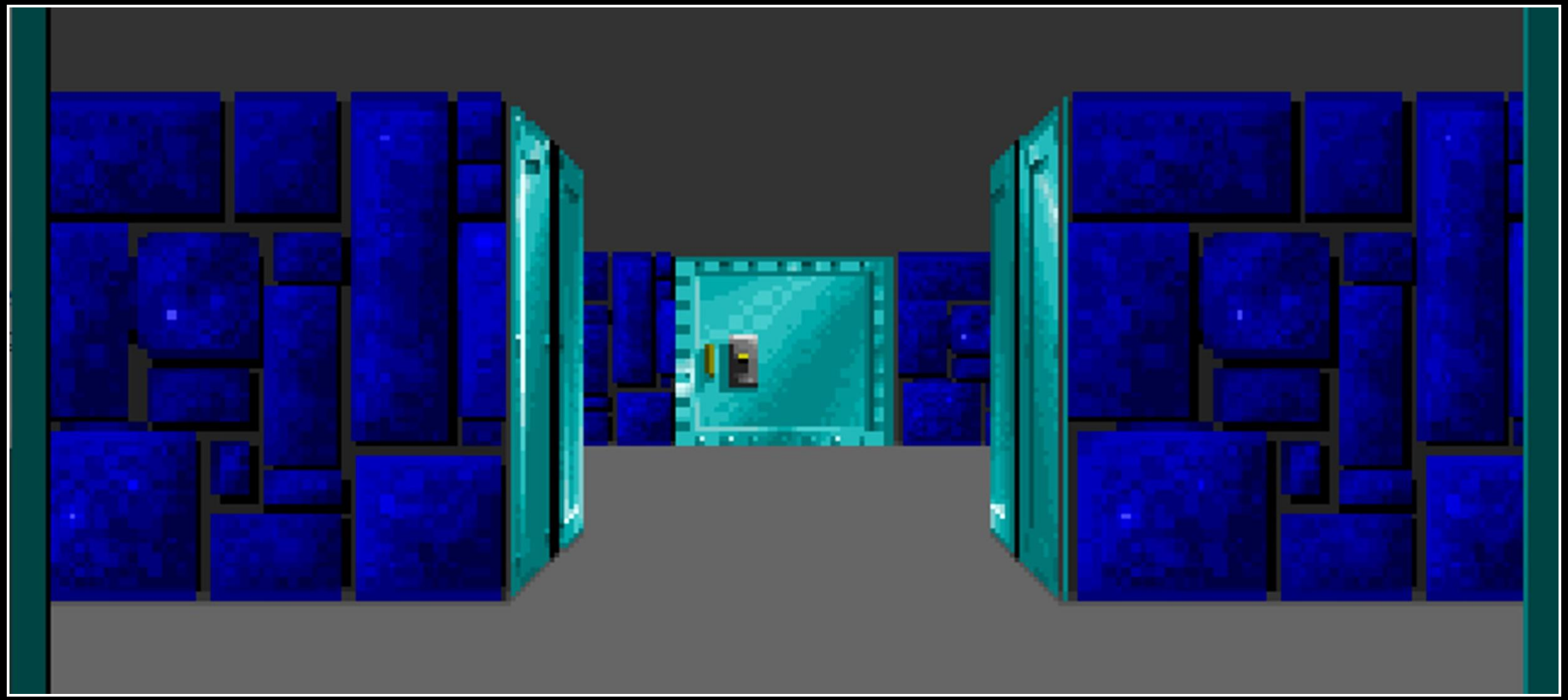
Aben-ham



2d Raycasting

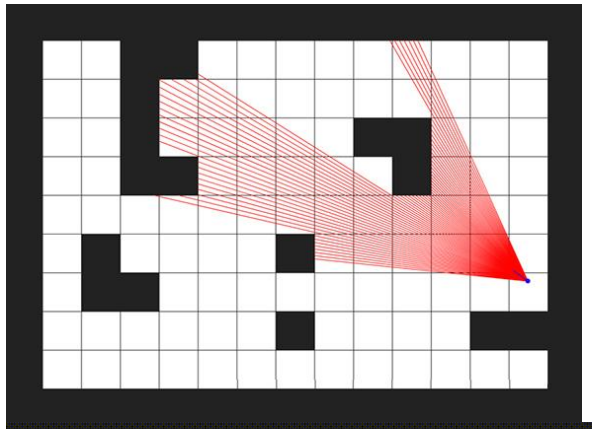
Main resources

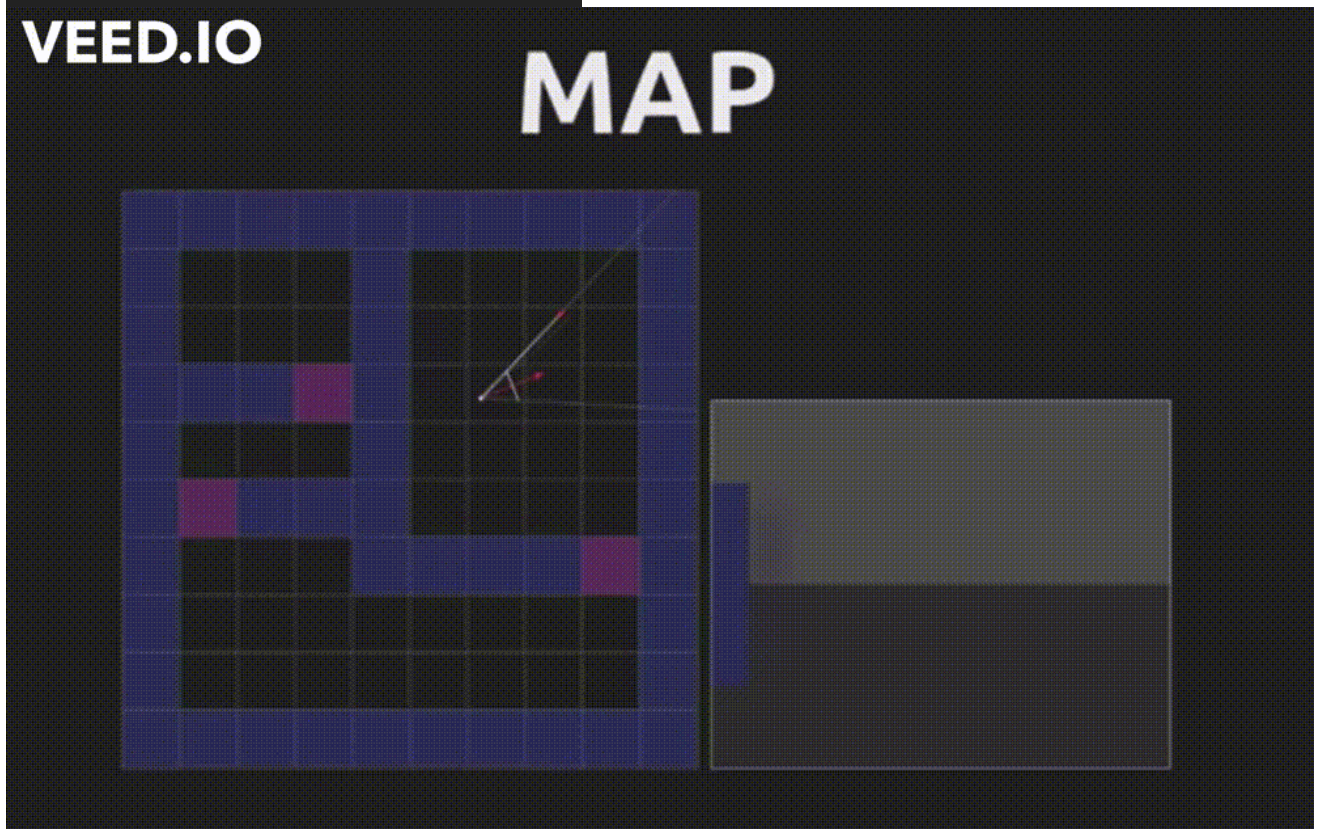
https://www.youtube.com/watch?v=eOCQfxRQ2pY => some illustration

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

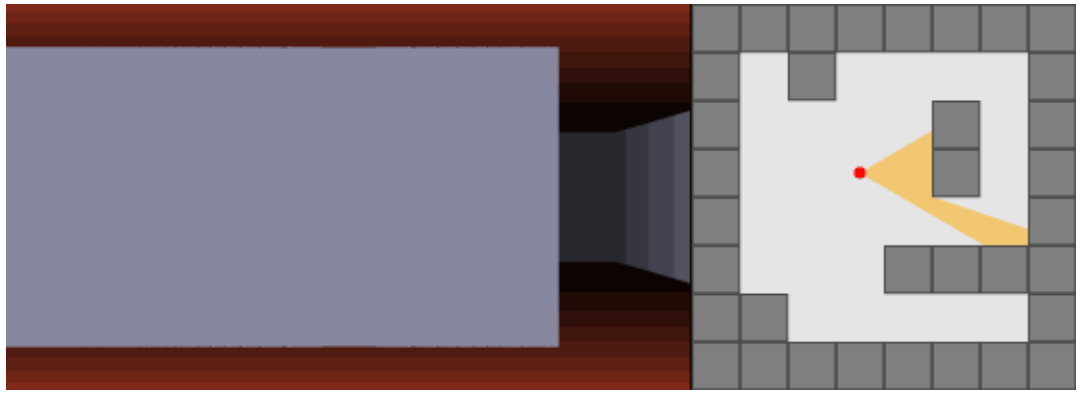
Need Pre knowledge : basic vectors, matrix, math

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.

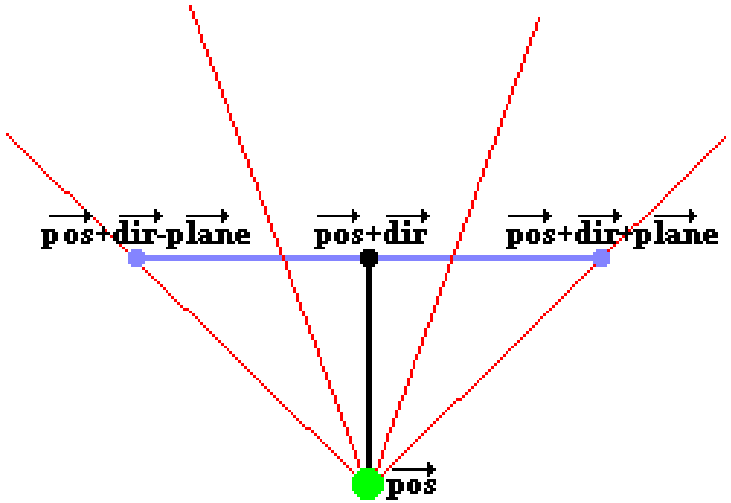


First we have to define how we represent the player and the rays

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 $\uparrow\downarrow$ (W,S)

$$pos.x \mathrel{+}= dir.x * Speed * Direction$$

$$pos.y \mathrel{+}= dir.y * Speed * Direction$$

if we move $\leftarrow\!\shortmid \;\shortmid\!\rightarrow$ (A,D)

$$pos.x \mathrel{+}= -dir.y * Speed * Direction$$

$$pos.y \mathrel{+}= dir.x * Speed * Direction$$

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

Rotating the Player is rotating the vector Dir.

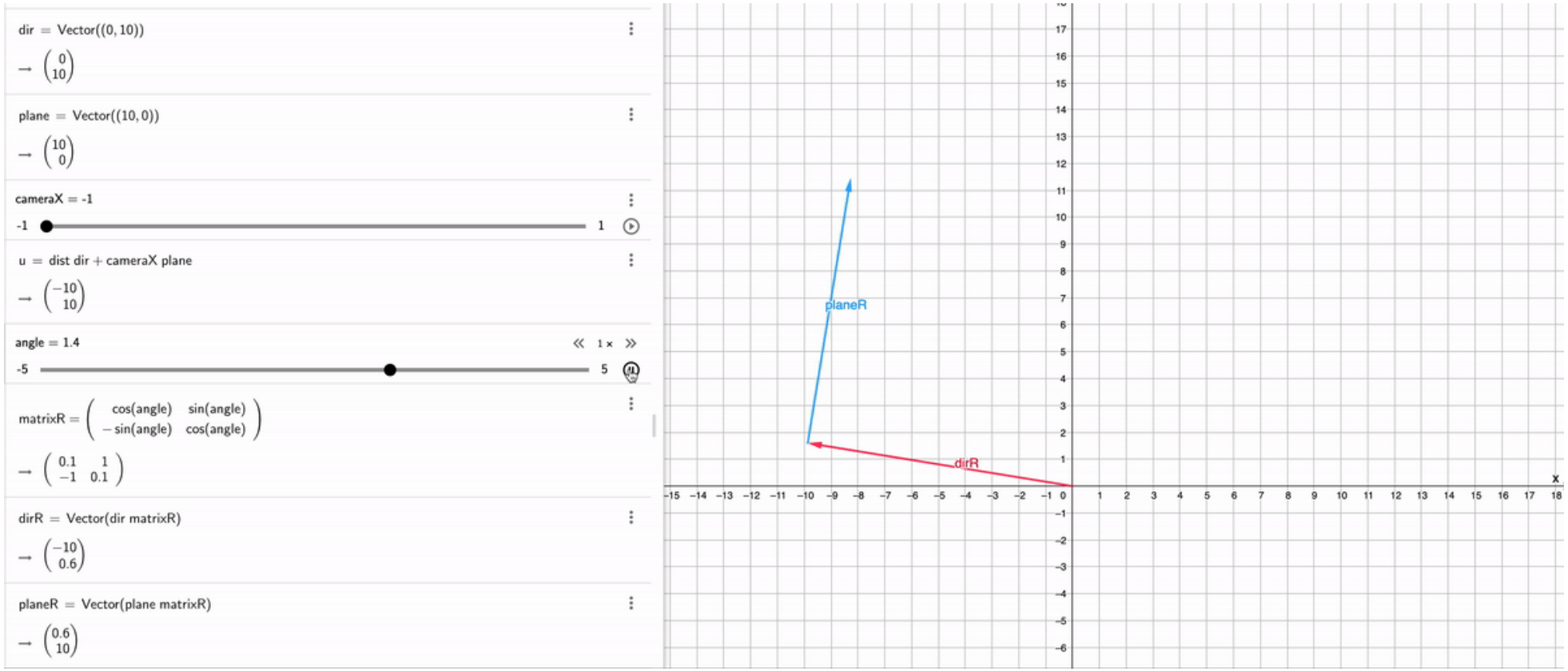
Obviously the vector Plane should be always $\perp$ To Dir

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

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

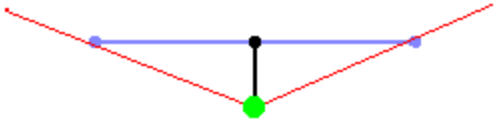
$$Dir = Dir * rotationMatrix$$

$$Plane = Plane * rotationMatrix$$

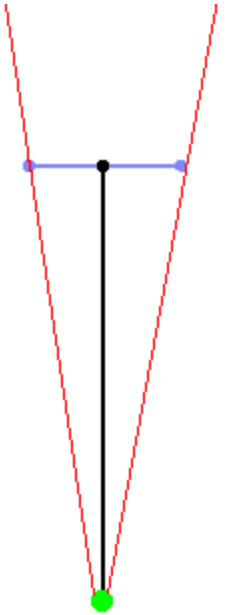


https://www.geogebra.org/calculator/emvkpzer

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 determinated by the ratio of the length Dir vector, and the length of the Plane vector.



the FOV will be larger than 90° (180° 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°, 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° field of view:

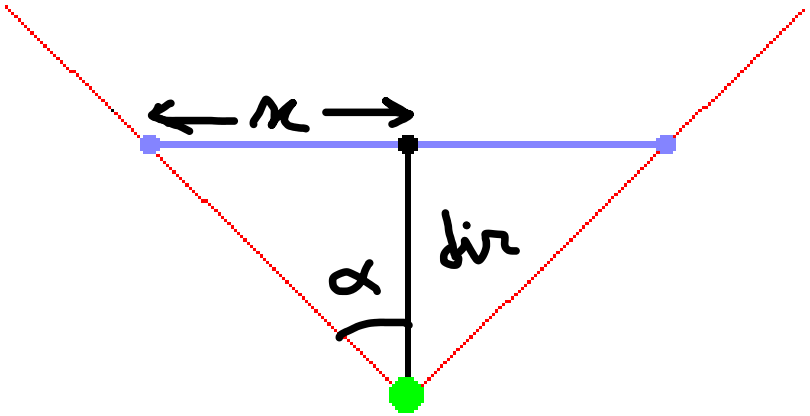
$$tan(\alpha) = \frac{x}{|dir|} = \frac{x}{1} = x \text{ with } \alpha = 33^\circ$$

initialization of pos, dir and plane

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

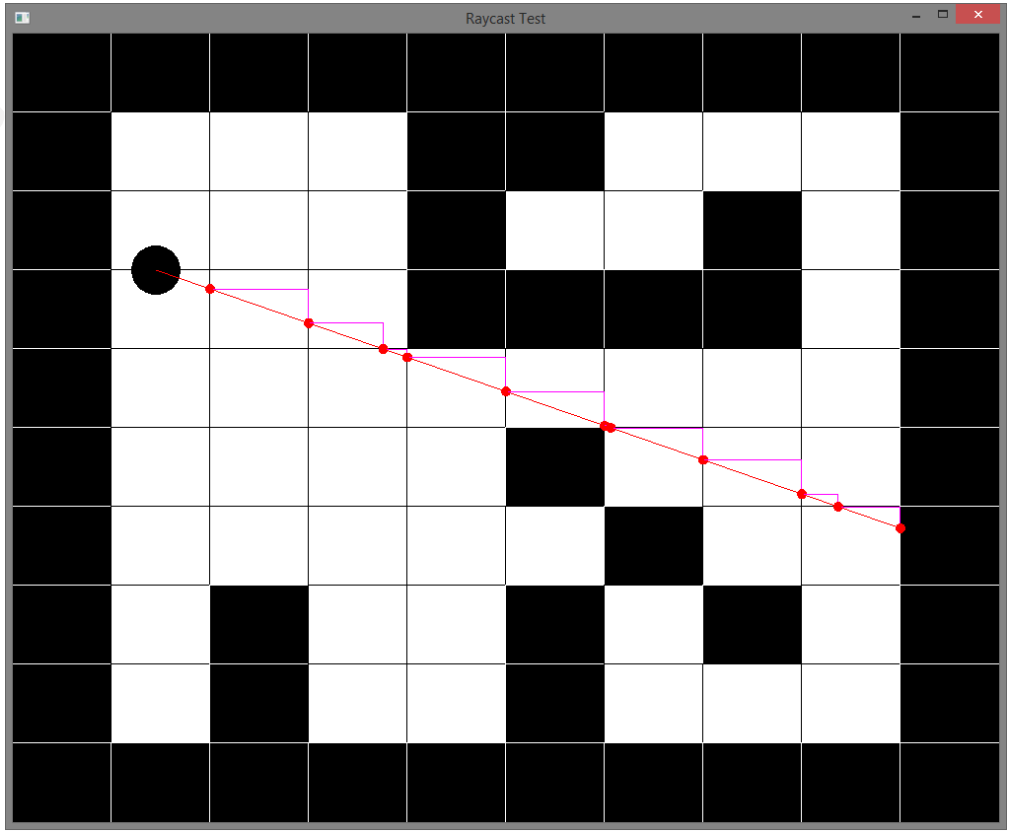
$$dir = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

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



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

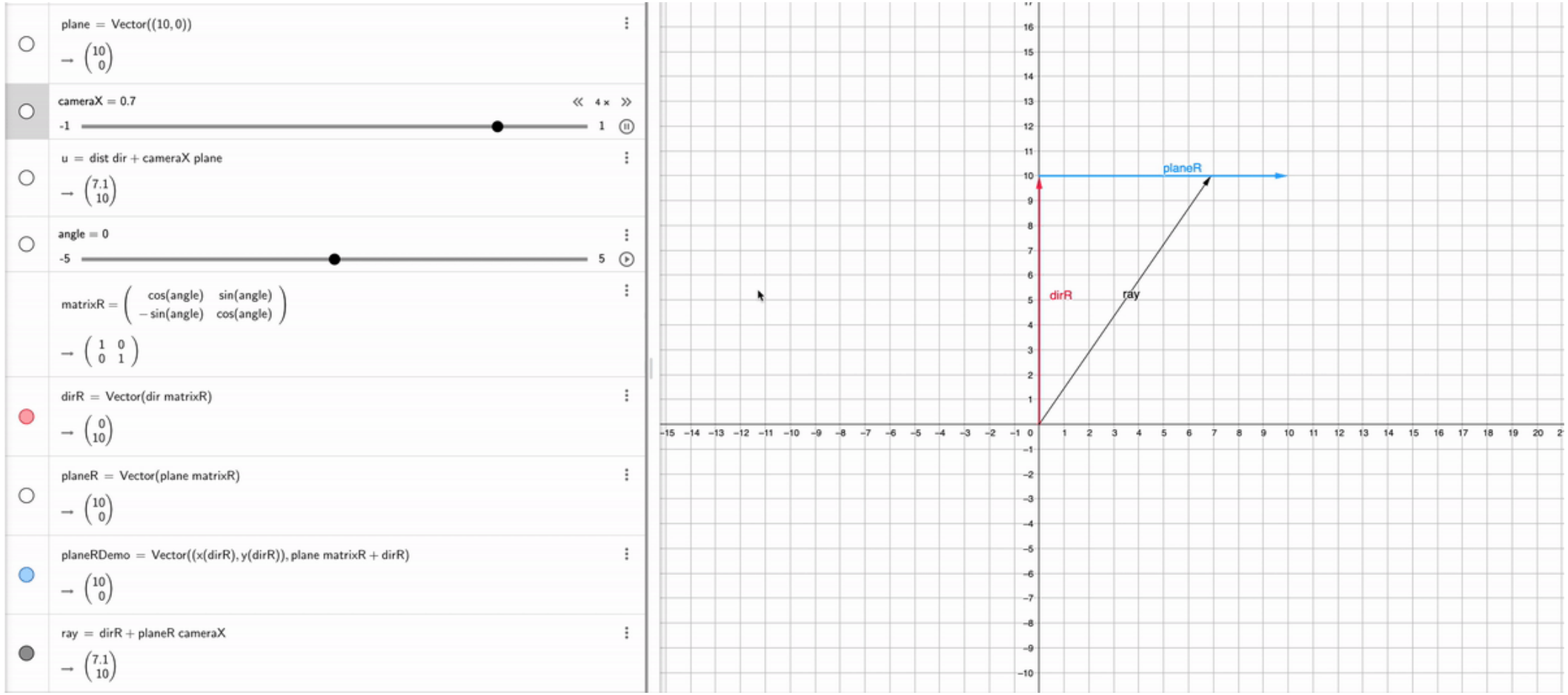
How to calculate the distance between the player and the wall for each ray



* 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

$$x \qquad 0 \rightarrow +W$$

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

$$2x - W \qquad -W \rightarrow W$$

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

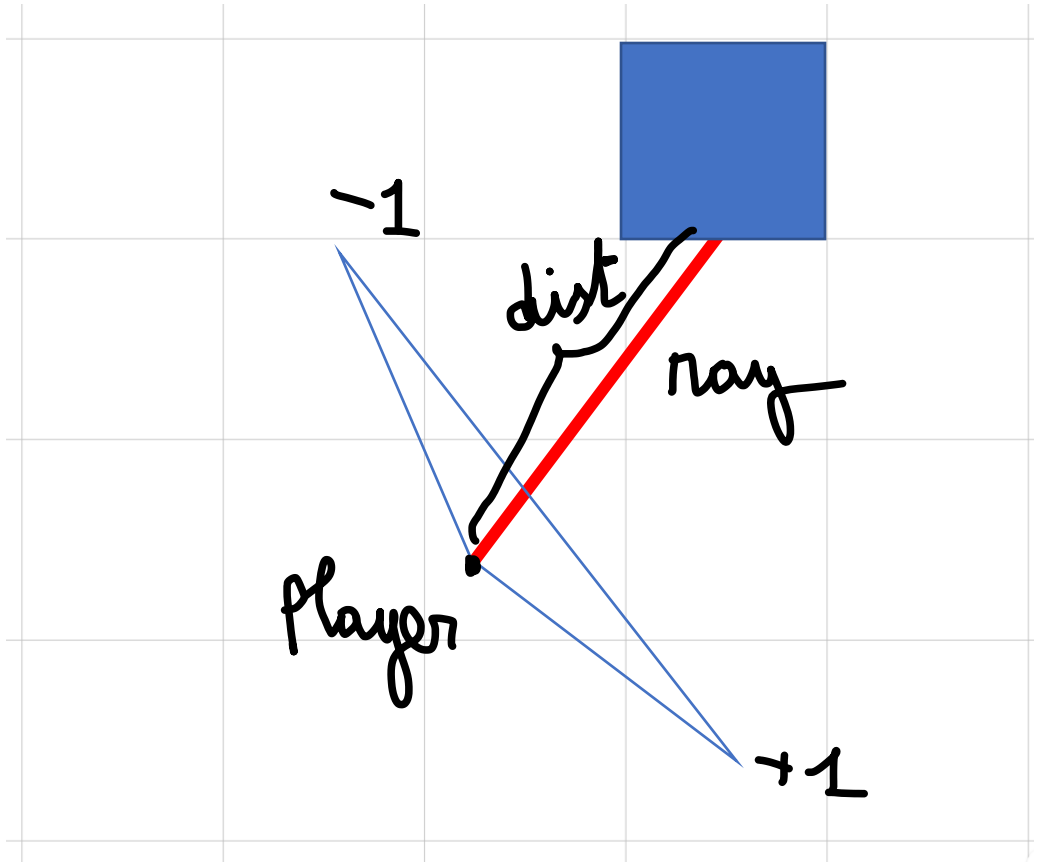
$$cameraX = \frac{2x - W}{W}$$

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

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

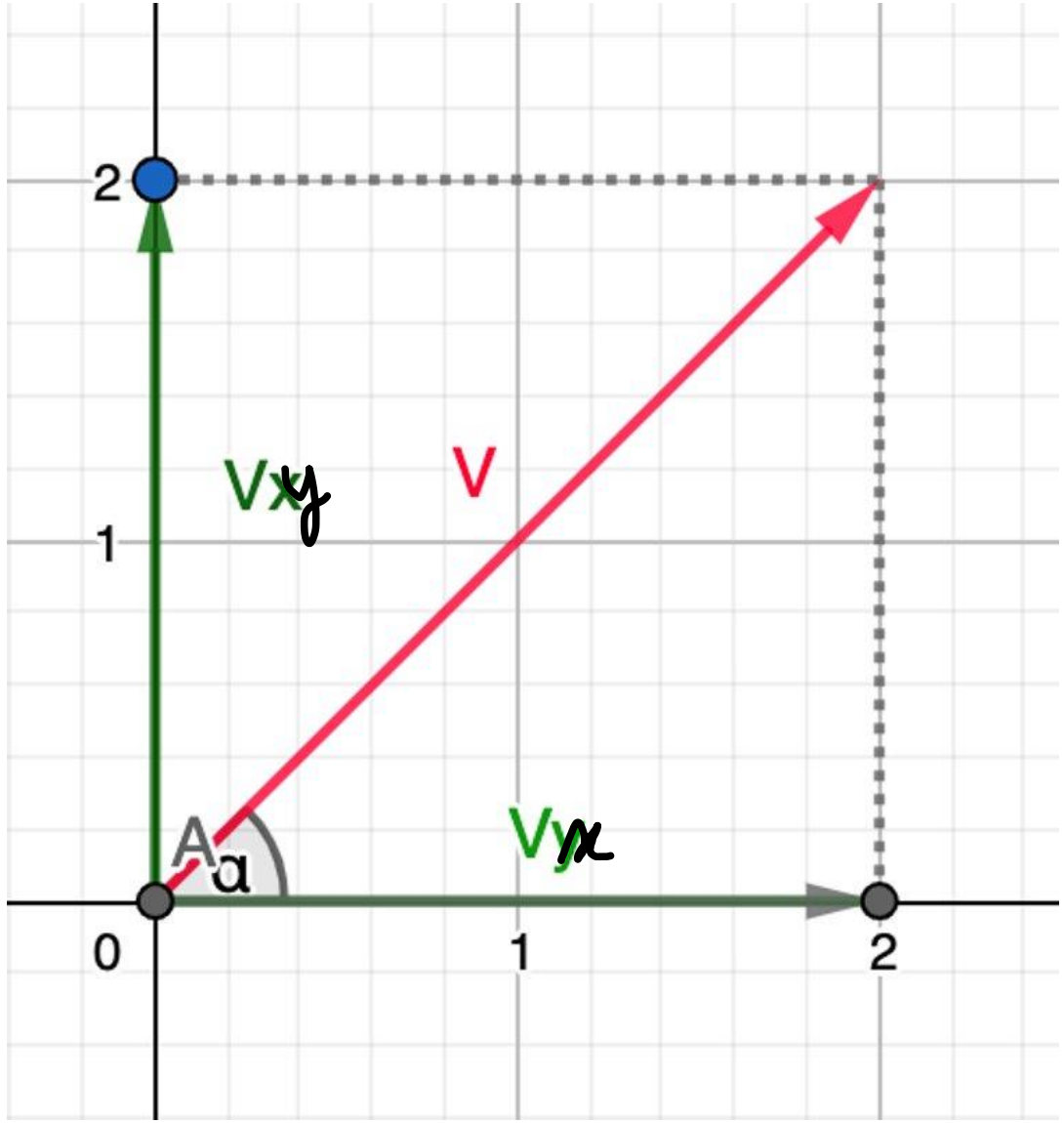
$$rayDir = dir + cameraX * plane$$

We now reedy 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. $\| \vec{v} \| = \sqrt{vx^2 + vy^2}$
3. $\tan(\alpha) = \frac{vy}{vx}$



calculate if the ray pass throught out one unit of y

we have form the right image:

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

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

From 1 and 2:

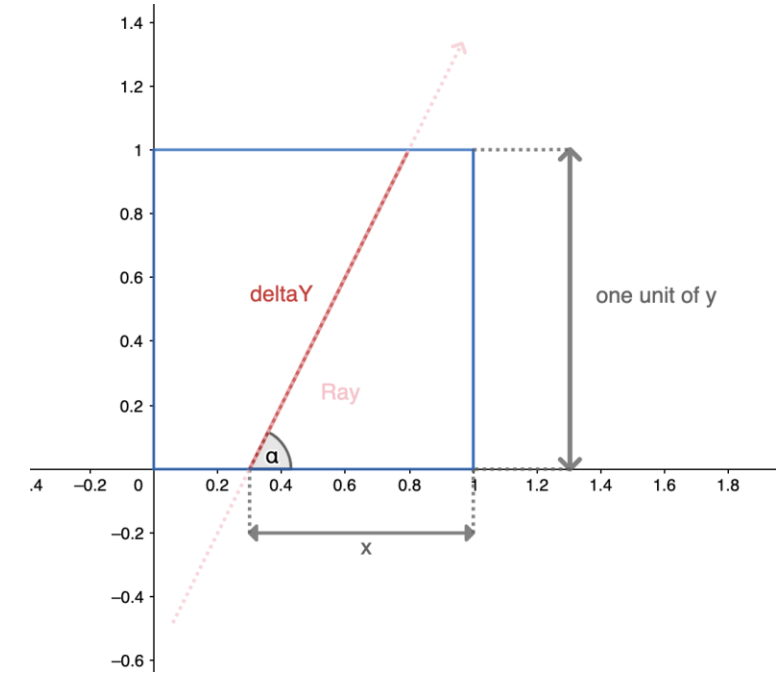
$$\Rightarrow \delta Y^2 = 1^2 + (\frac{raydir.\mathrm{x}}{raydir.\mathrm{y}})^2$$

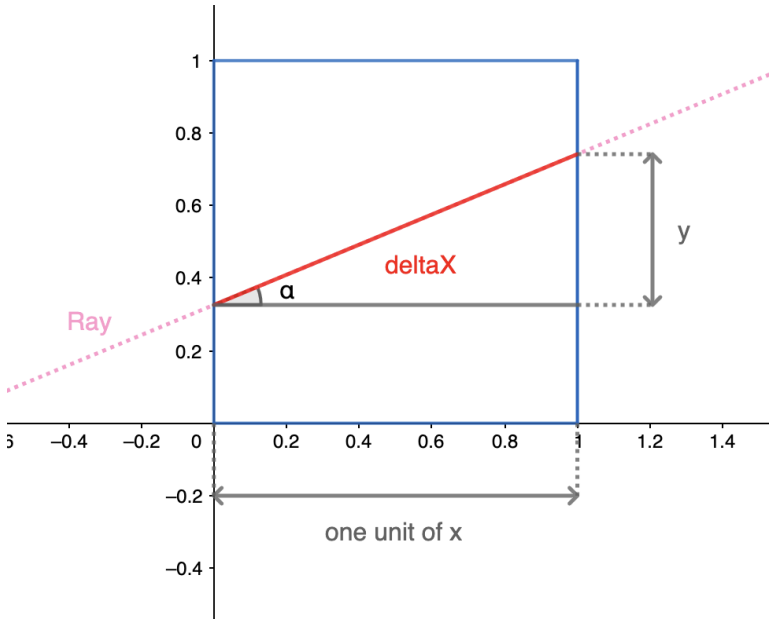
$$\Rightarrow \delta Y^2 = \frac{raydir.\mathrm{x}^2 + raydir.\mathrm{y}^2}{raydir.\mathrm{y}^2}$$

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

same calculation goes from δX where the ray pass throught out one unit of x

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





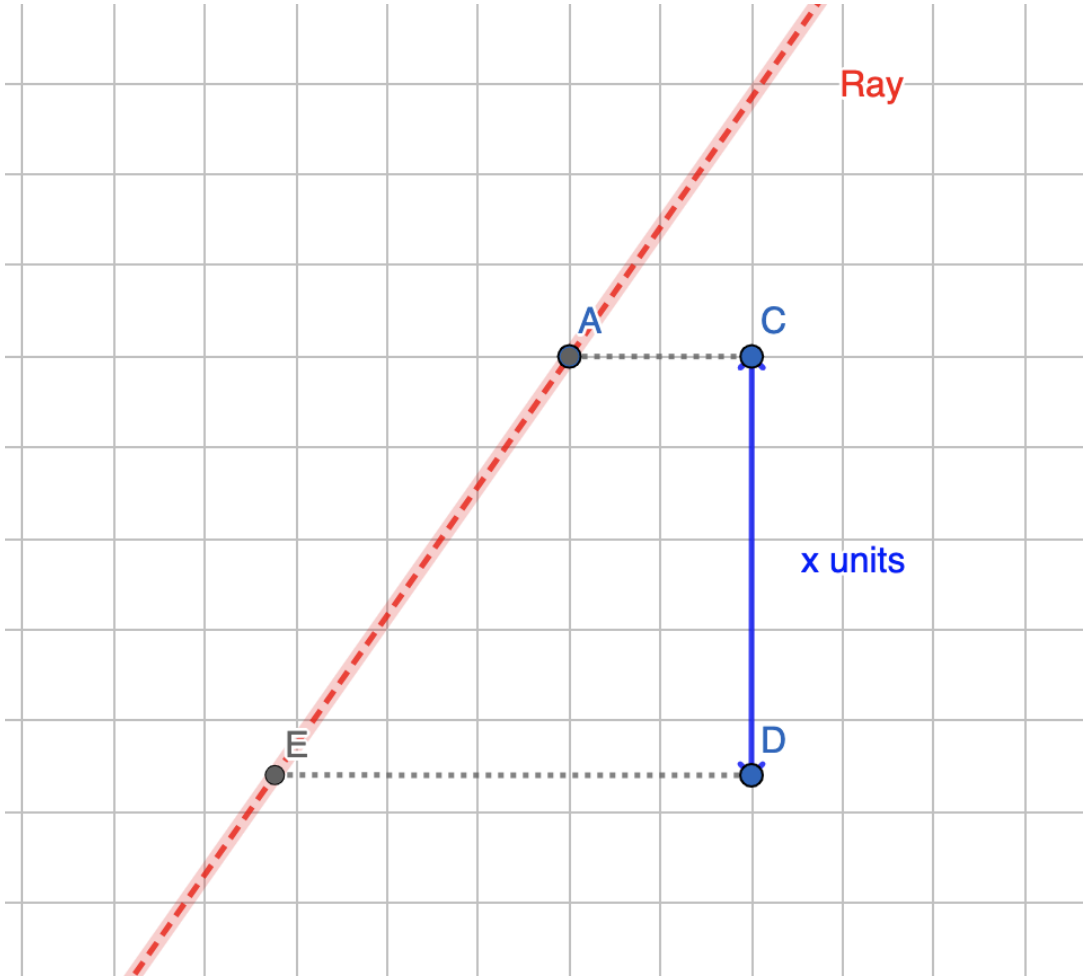
$$\delta Y \quad \rightarrow \quad one\ unit\ of\ Y$$

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

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

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

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



Summary

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

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

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

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

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

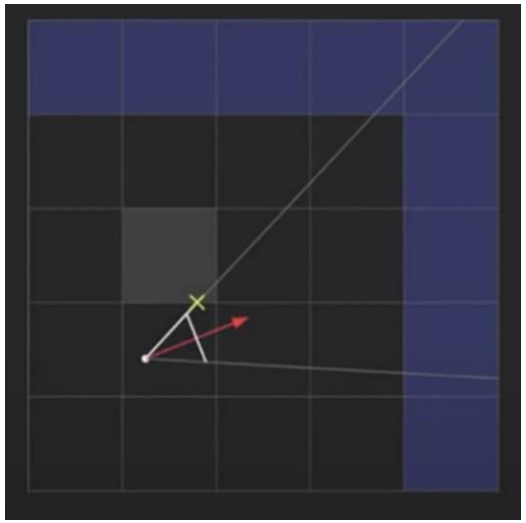
$$\delta Y' = \frac{\|raydir\|}{|raydir.y|}$$

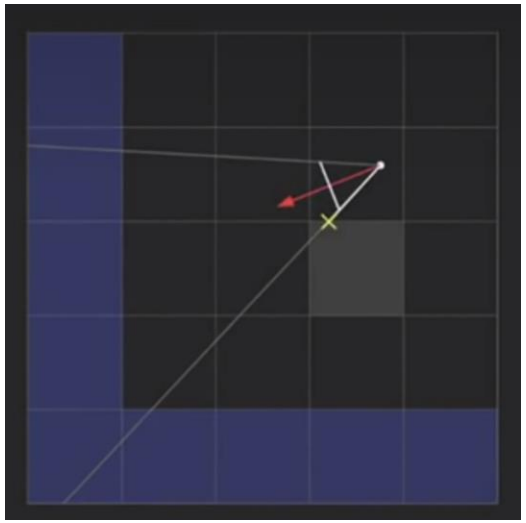
$\Delta X = x.\delta X$ *x is the horizontal distance between the wall and the player*

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

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

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





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

$$\text{ray.mapX} = (\text{int})\text{player.x}$$

$$\text{ray.mapY} = (\text{int})\text{player.y}$$

$$\text{if raydir.y} \,||\, \text{raydir.x} > 0 \text{ player look above } \uparrow \text{ or to to the right } \rightarrow$$

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

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

$$\text{else raydir.y} \,||\, \text{raydir.x} < 0 \text{ player look down } \downarrow \text{ or to the left } \leftarrow$$

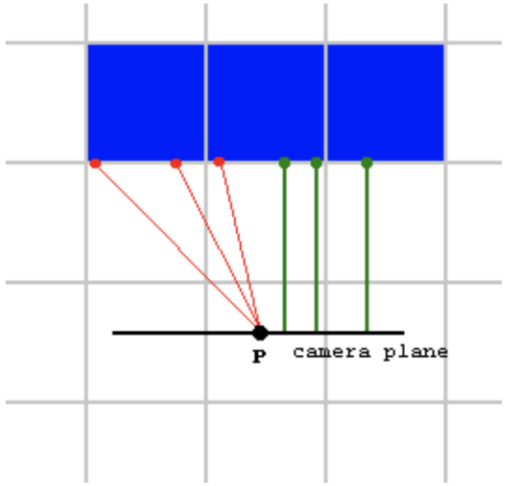
$$\Delta x = (\text{player.x} - (\text{int})\text{player.x}) * \delta x$$

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

Notice: you have to spite 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 δx and δy
    //calculate Δx and Δx
    while (1)
    {
        if (Δx <= Δy)
        {
            Δx += δx
            mapX += stepX
            side = 0 //represent R or W
        }
        else
        {
            Δy += δy
            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 = Δx - δx;
            else
                distance = Δy - δy;
            break ;
        }
    }
}
```

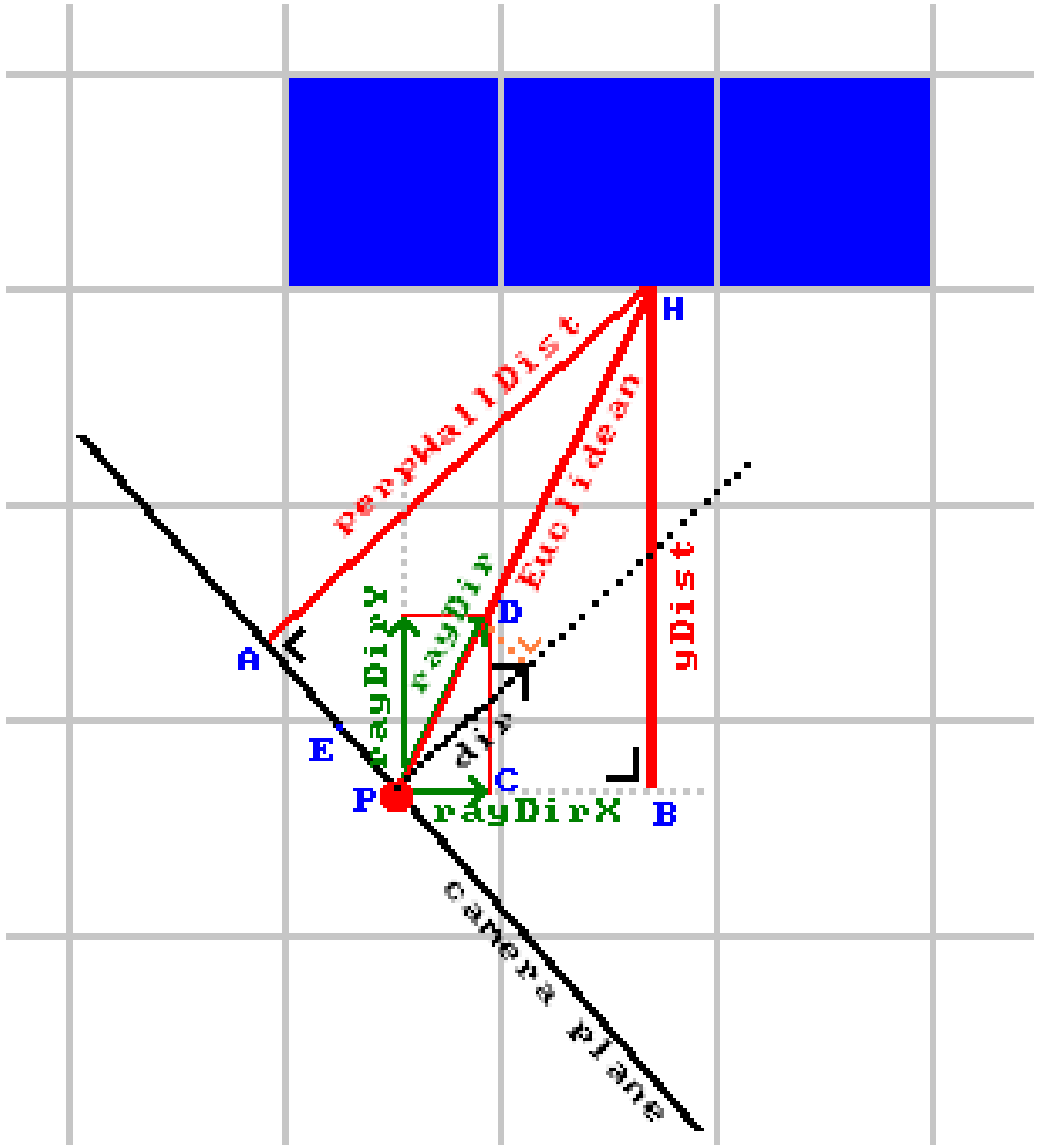
In the image, the player is looking directly at the wall, and in that case you would expect the wall's bottom and top to form a perfectly horizontal line on the screen. However, the red rays all have a different lenght, so would compute different wall heights for different vertical stripes, hence the rounded effect. The green rays on the right all have the same length, so will give the correct result. The same still apllies 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.

To prevent the rounded effect, we will use perpWallDist instead of Euclidean

- This perpenducular distance is called "perpWallDist" in the code. One way to compute it is to use the formula for shortest distance from a point to a line, where the point is where the wall was hit, and the line is the camera plane:



We will using:

1. $E = \Delta Y' = yDist.\delta Y'$
2. $\Delta Y = yDist.\delta Y$
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 Thles's theorem.

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

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

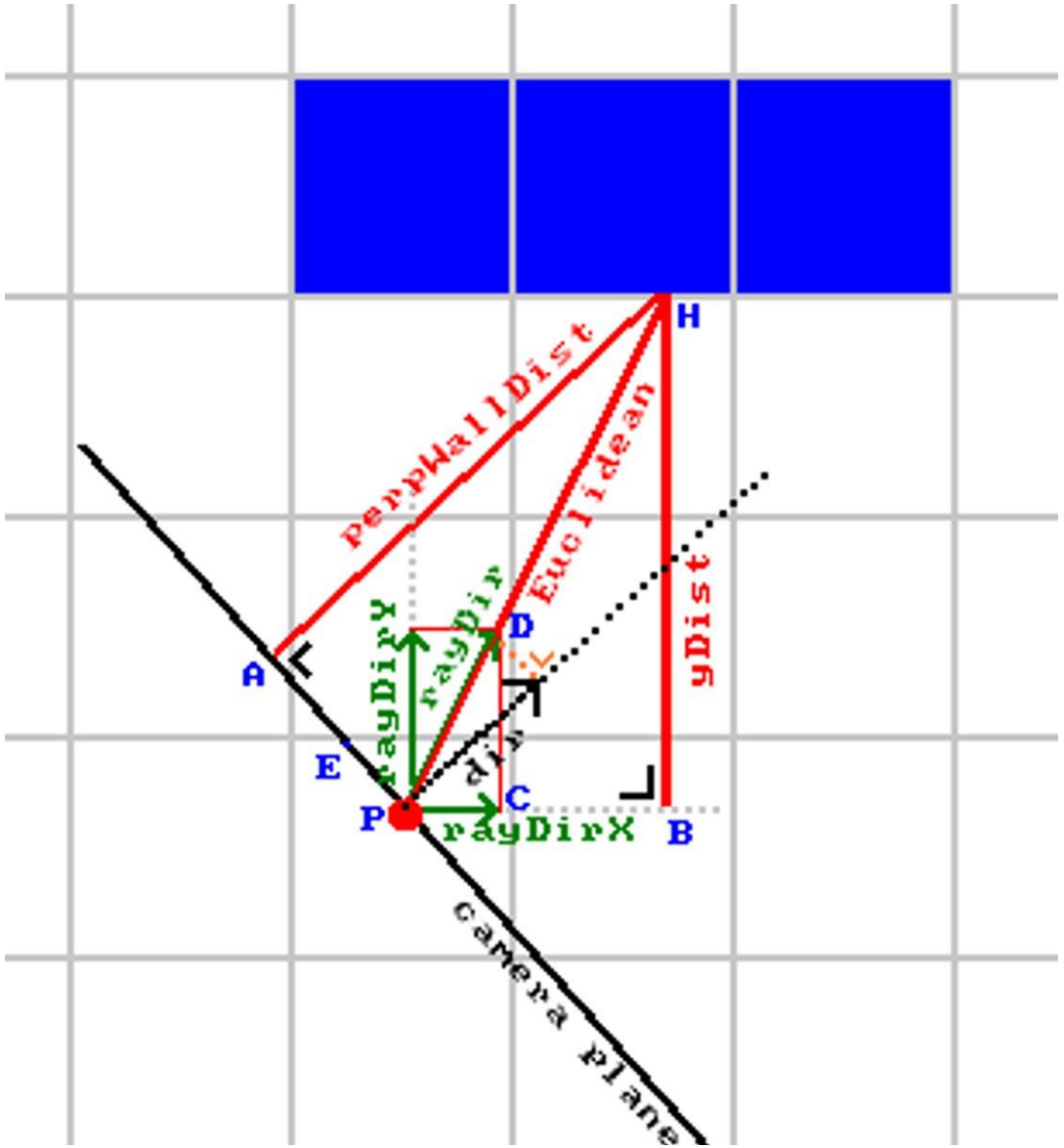
$$\Rightarrow perWallDist = \frac{\delta Y'.yDist}{\|\,rayDir\,\|} \text{ (1) } \textit{and} \text{ (5)}$$

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

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

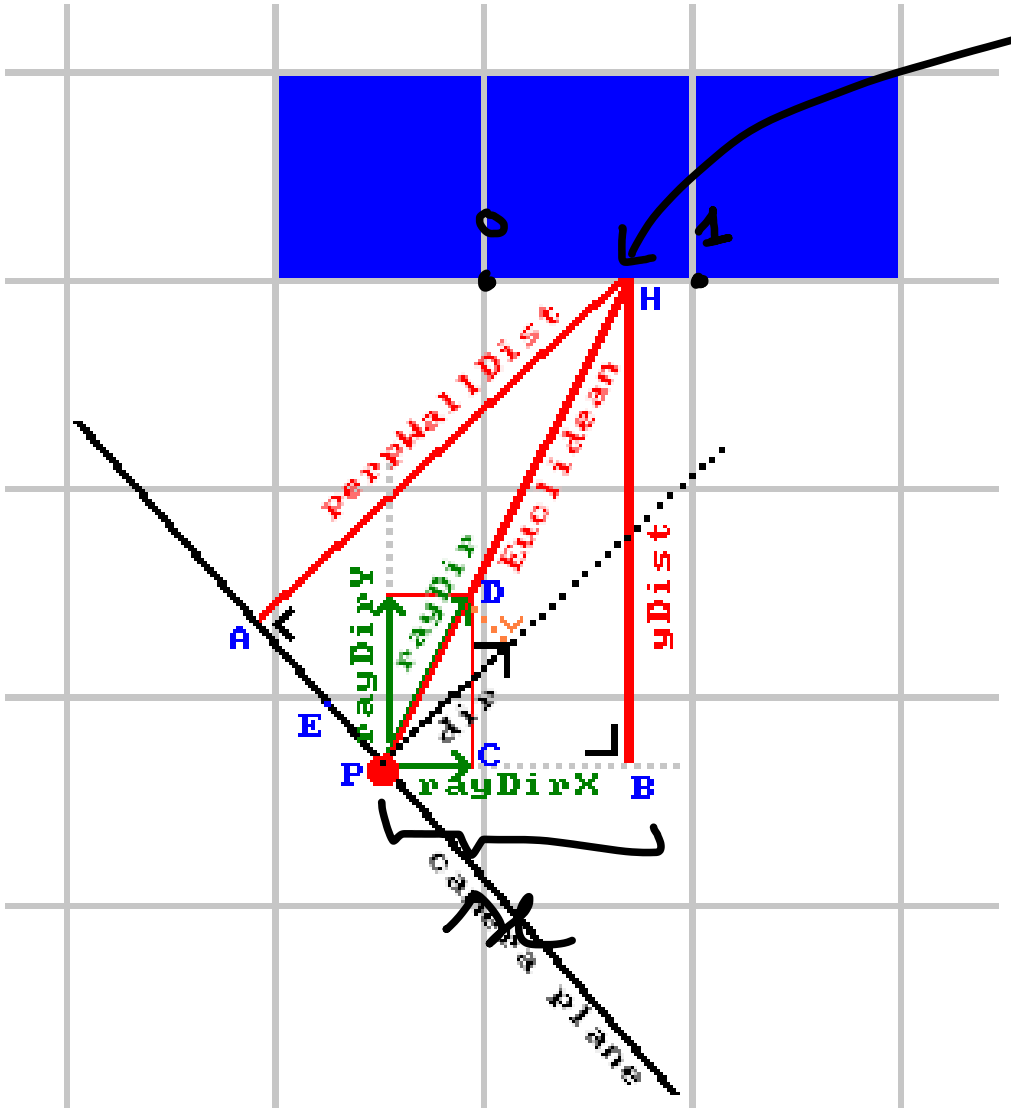
$$\Rightarrow perWallDist = yDist.\delta Y \text{ (4)}$$

$$\Rightarrow perWallDist = \Delta Y \textit{ (2) calculted usign the dda (distance in the code above)}$$



$$
\begin{aligned}
&if\ the\ ray\ hit\ the\ side\ W\ or\ S \\
&\qquad perWallDist\ =\ \Delta y \\
&else\ E\ or\ E \\
&\qquad perWallDist\ =\ \Delta x
\end{aligned}
$$

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

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

else W or E

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

.

We will treat this

we will using

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

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

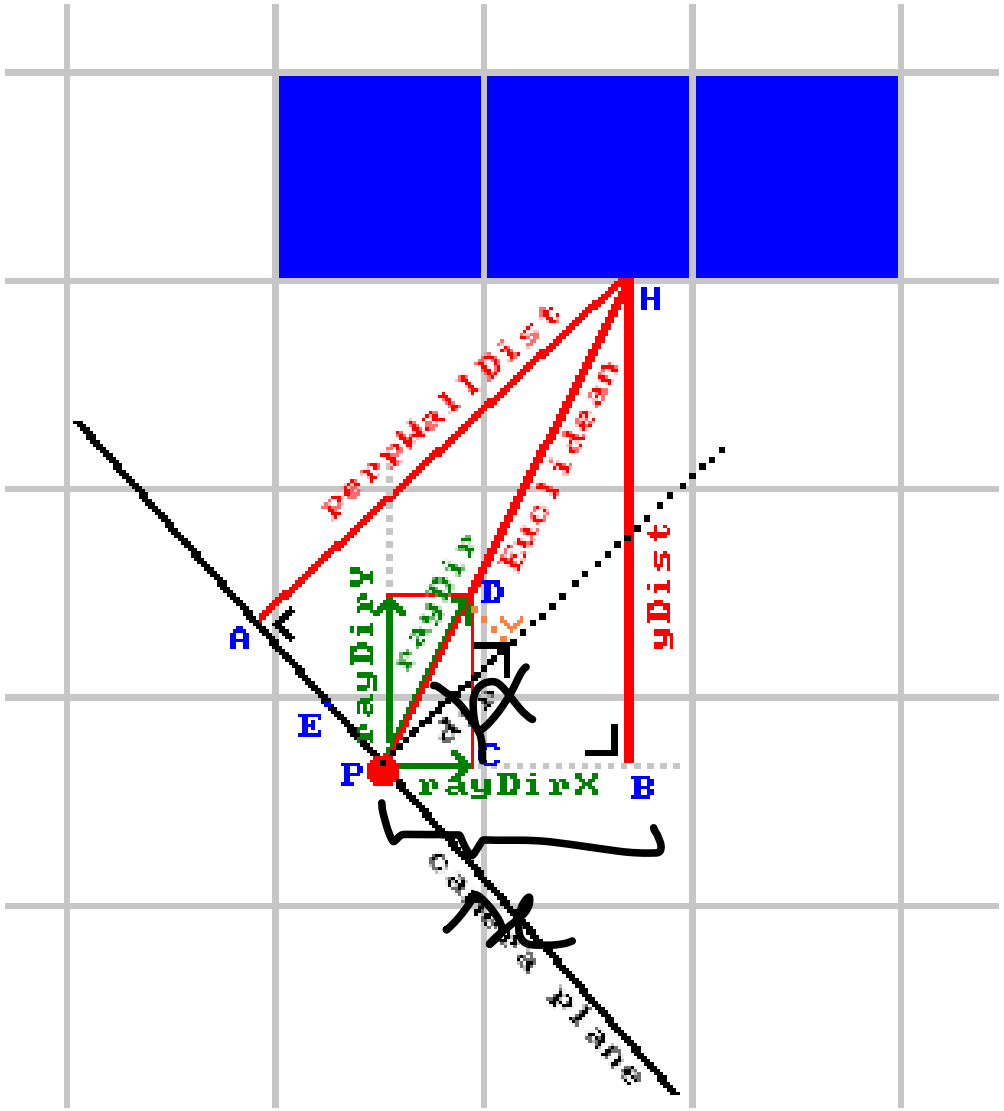
$$\Rightarrow x = yDist \,.\, \frac{rayDir.x}{|rayDir.y|}$$

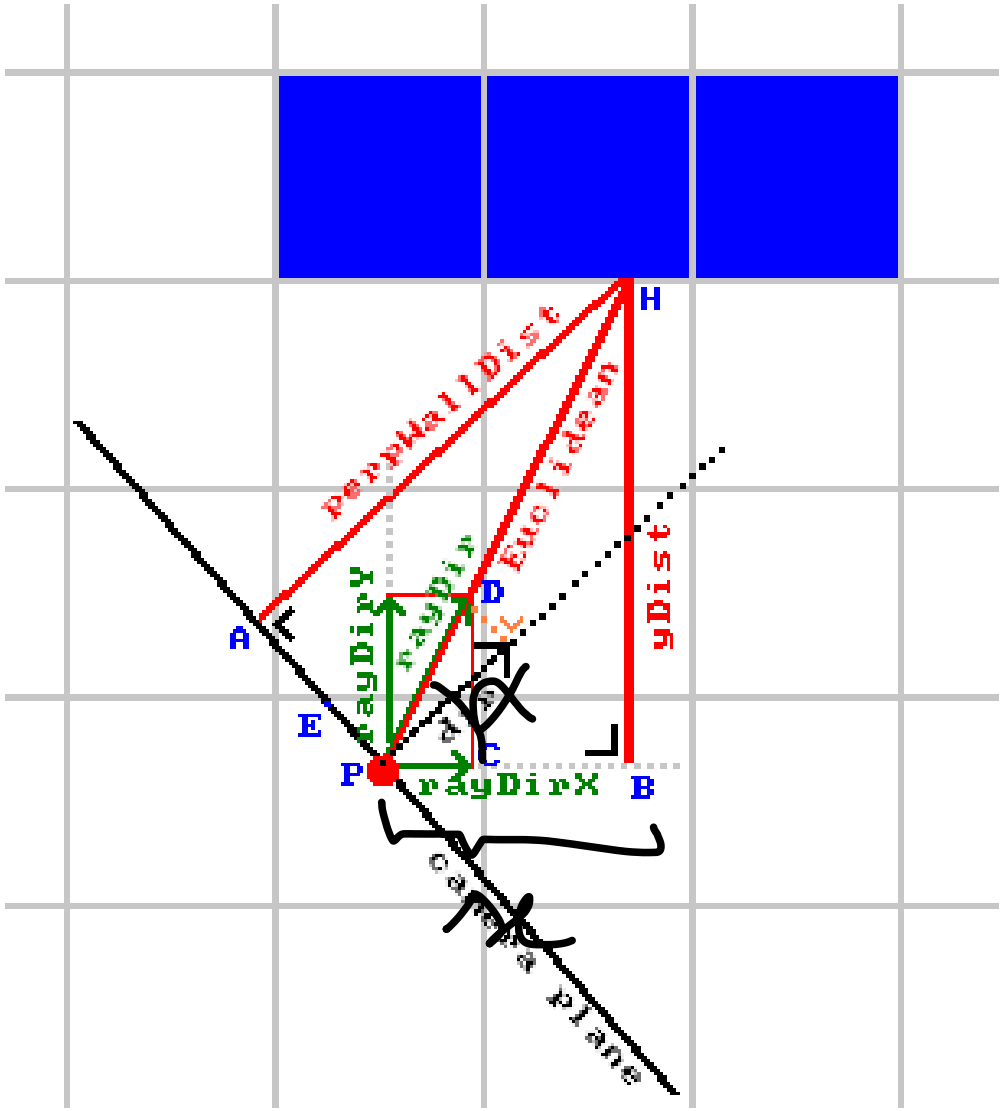
$$\Rightarrow x = \frac{E}{\delta Y'} \cdot \frac{rayDir.x}{|rayDir.y|} \quad (2)$$

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

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

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





we will using

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

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

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

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

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

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

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

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

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

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

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

$$perWallDist\ =\ \Delta y$$

$$else\ E\ or\ E$$

$$perWallDist\ =\ \Delta x$$

$$if\ the\ ray\ hit\ the\ side\ N\ or\ S$$

$$Wallx = player.x\ +\ perWallDist\ .\ rayDir.x$$

$$else\ E\ or\ W$$

$$Wallx\ =\ player.x\ +\ perWaDist\ .\ rayDir.y$$

$$Wallx\ =\ Wallx\ -\ (int)Wallx$$

Getting the color from the texture

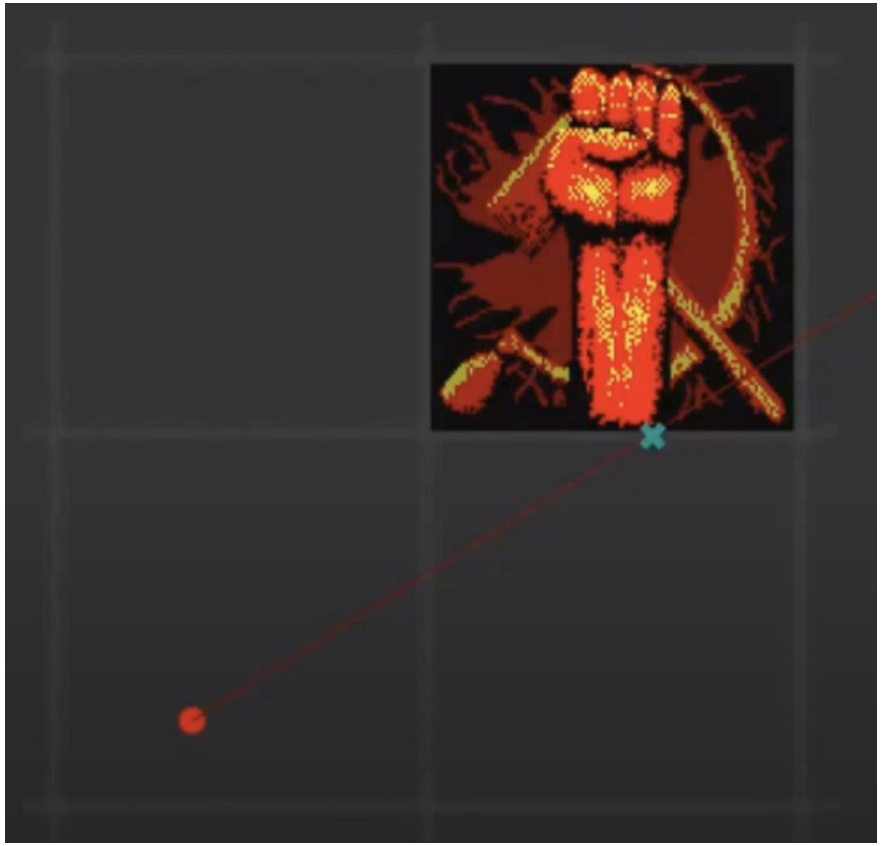
$let\ x\ and\ y\ be\ cooredinate\ in\ the\ texture$

$we\ have$

$$x = texWidth\ *\ wallX;\ sice\ 0 \leq wallX \leq 1$$

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

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



Floor and ceiling

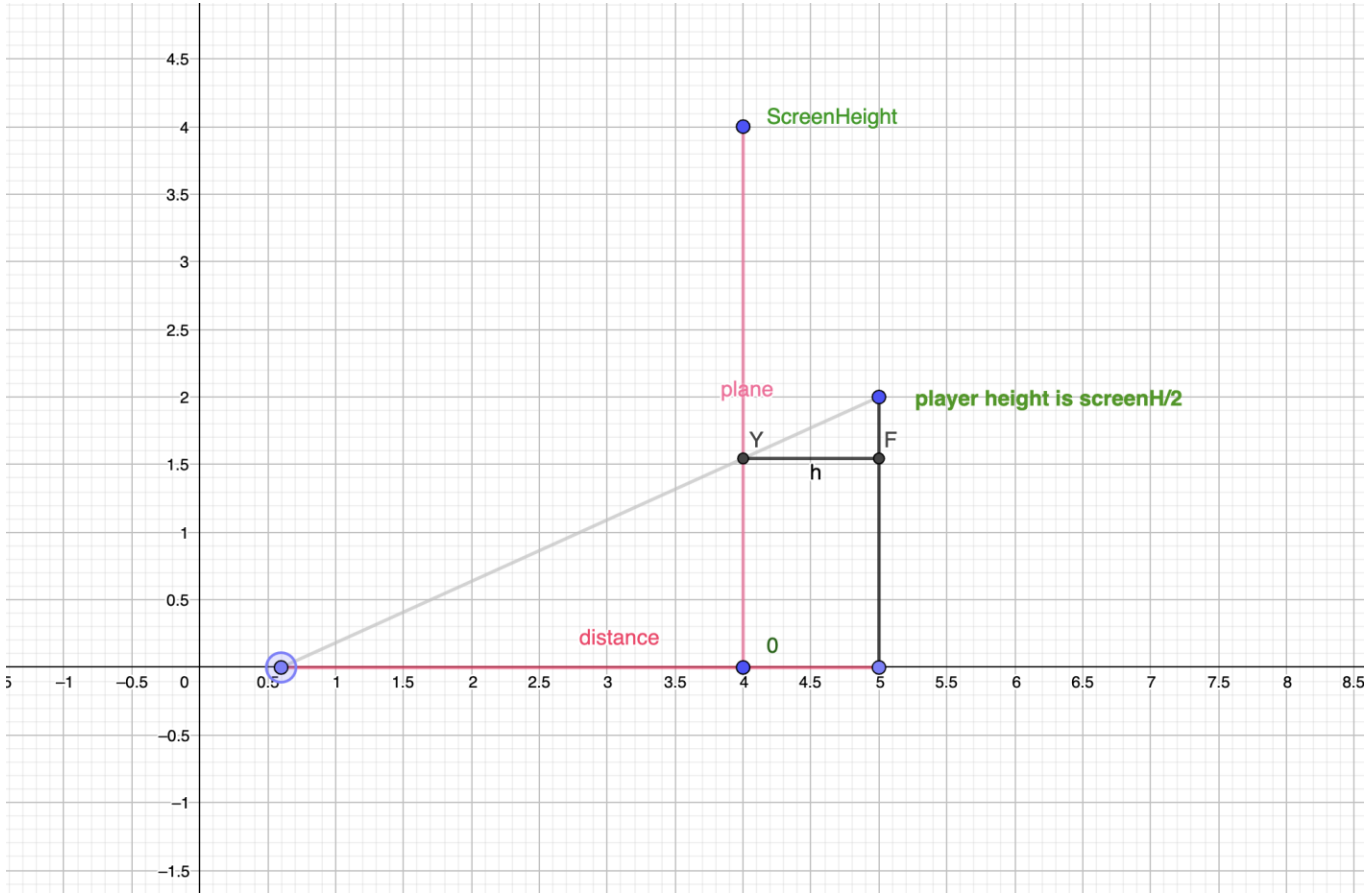
Floor and ceiling

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

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

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

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 form -1 to $+1$ when $screenX$ goes from 0 to $Width$

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

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

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

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

Floor and ceiling

$$Summary\ \{$$

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

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

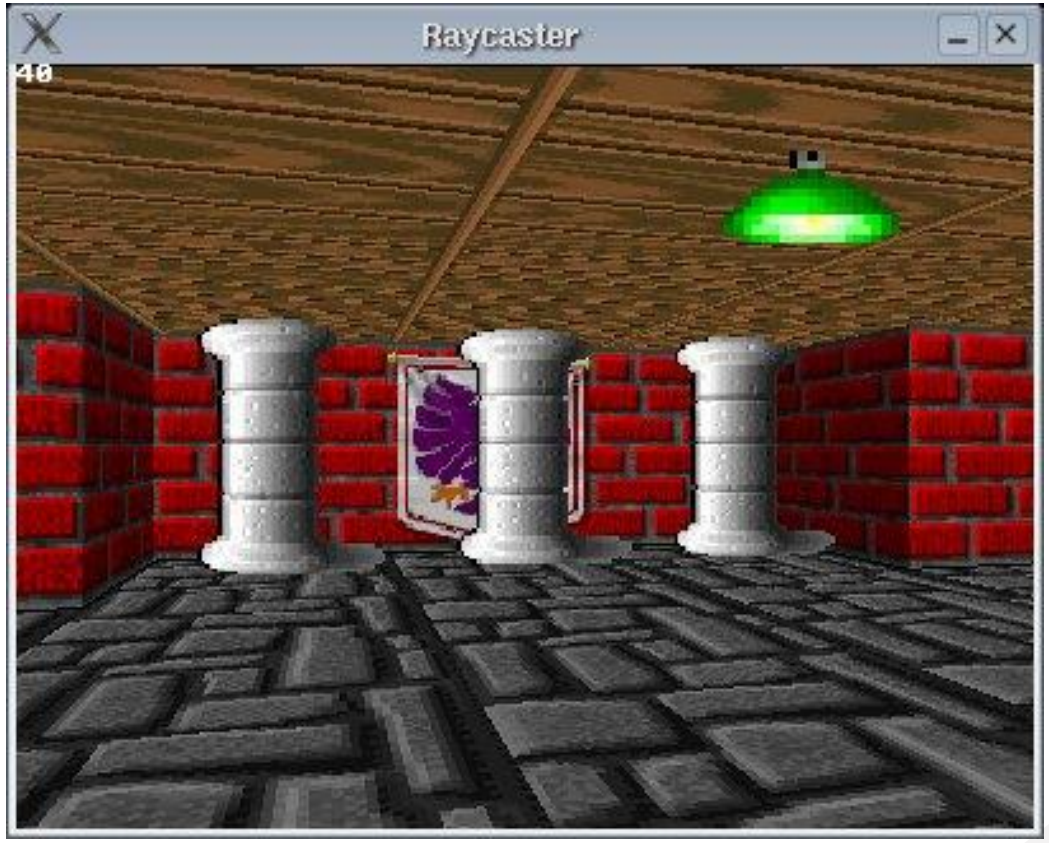
$$distance = \frac{Height\ /\ 2}{Height\ /\ 2\ -\ Y} \qquad Y \quad 0 \rightarrow Height/2$$

$$texX = (X - (int)X) * tex.width$$

$$texY = (Y - (int)Y) * tex.Height$$

$$\}$$

Sprites



Sprites

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

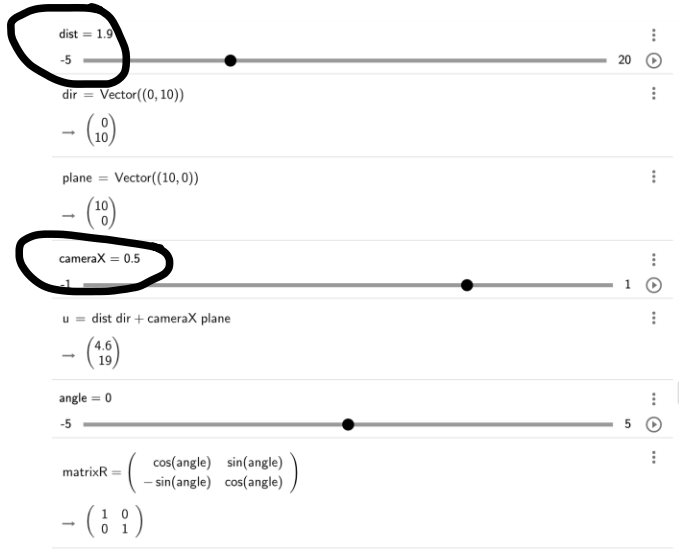
$$cameraMatrix\ *\ 2dVector\ =\ \begin{bmatrix} plane.x & dir.x \\ plane.y & dir.y \end{bmatrix}.\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} x.plane.x\ +\ y.dir.x \\ x.plane.y\ +\ y.dir.y \end{bmatrix}=\begin{bmatrix} X \\ Y \end{bmatrix}$$

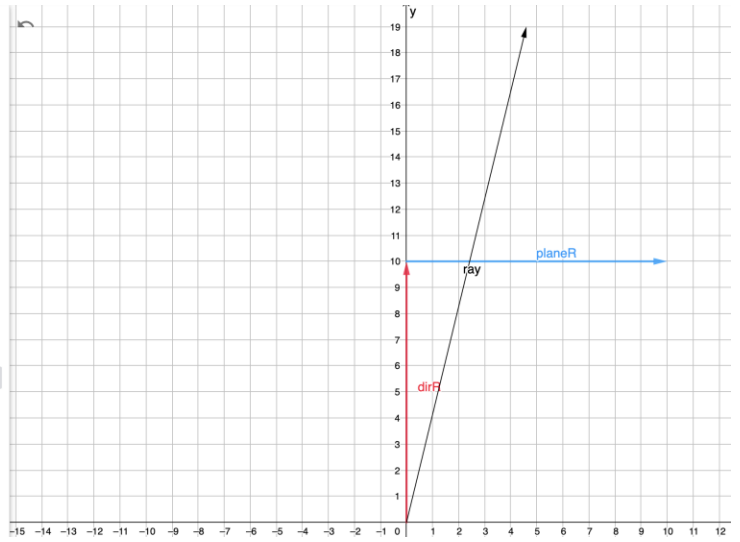
$x,\ y \in \mathbb{R}$

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.

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

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

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

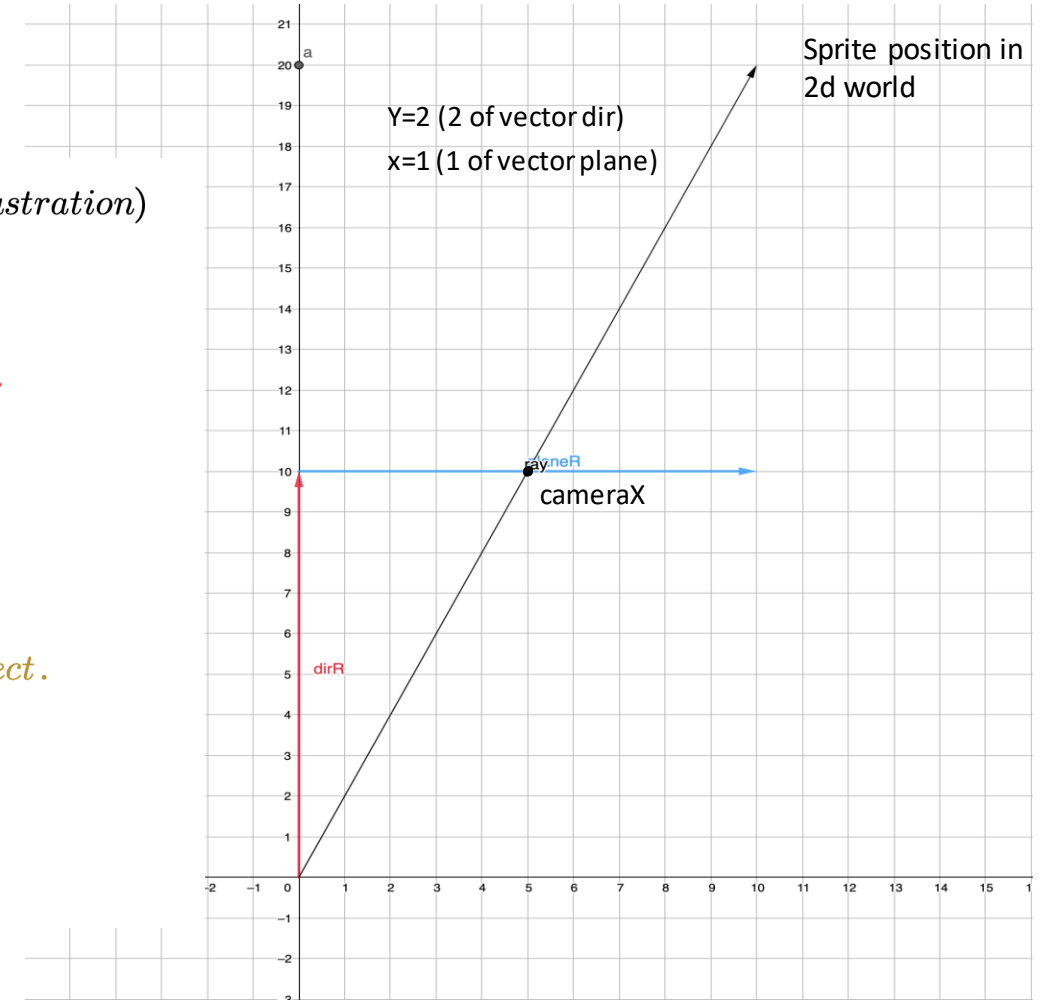
$$\Rightarrow \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} dir.y & -dir.x \\ -plane.y & plane.x \end{bmatrix} . \frac{1}{det(cameraMatrix)} . \begin{bmatrix} X \\ Y \end{bmatrix}$$

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

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

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: $\frac{cameraX}{x} = \frac{|dir|}{y} \Rightarrow cameraX = \frac{x}{y}$

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 $cameraX > 1 \;||\; cameraX < -1$ *represent non visible part of the object.*

1. so we will have $\text{screenX} = \frac{\text{Width} * (\text{cameraX} + 1)}{2} = \frac{\text{Width} * (\frac{x}{y} + 1)}{2}$
2. *the size(with = height) of the sprite in the screen will be* $\frac{ScreenHeight}{y}$

If the sprite positioned in Y=1 we will have SpriteHeight=ScreenHeight
Because the sprite positioned in the plane

Sprites

Final result

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* $\frac{ScreenHeight}{y}$

* $spriteScreenStartX = screenX - \frac{size}{2}$
* $spriteScreenEndX = screenX + \frac{size}{2}$
* $spriteScreenStartY = \frac{ScreenHeight}{2} - \frac{size}{2}$
* $spriteScreenEndY = \frac{ScreenHeight}{2} + \frac{size}{2}$

To center the sprite

Other resources

https://www.youtube.com/watch?v=NbSee-XM7WA&ab_channel=javidx9 => dda

https://www.youtube.com/watch?v=DFZnzCbmlng&ab_channel=JeremyCodes => cub3d

https://www.youtube.com/watch?v=fNk_zzaMoSs => base vectors

https://www.youtube.com/watch?v=S0uzwDKqnsw&ab_channel=JeffreyChasnov => rotation matrix