There are 4 cases for projecting 3d polygons onto a 2d surface.

1. All points of the polygon are in view pyramid
2. All points of the polygon are in front of the player (maybe some not in view pyramid)
3. Points are in front and behind the player
4. All points behind the player

Case iv is easy as if all points are behind, nothing should be projected to the 2d surface

The other cases are shown below for the left wall. Note

The player is where the dotted red lines meet.

The canvas is the solid red rectangle.

The view pyramid is the pyramid with the canvas as the base and player as the top

 I ii iii

Note that case i and case ii give good projections to the 2d surface but case iii does not.

The projection works for points in front of the player by dividing the x and y coordinates by the depth z.

For points behind the player like for the left wall in case iii, dividing by the depth is like projecting points to a plane behind the player and since depth < 0 swapping the projected points to the other side of the canvas. You can see that the projected image is totally wrong

A solution is to convert case iii to case ii by interpolating the points on the left wall to a point in front of the player but behind the players view pyramid (like in case ii).

You could interpolate straight to the view pyramid which would make sense but this is expensive as you have to calculate the distance in front of the player toward the point that the polygon intersects the view pyramid.

Another solution is to interpolate to a arbitrary value like y=0.1 units in front of the camera however if the wall is very close to the player this interpolation can push points in front of the view pyramid. This is why a 3D game might glitch near walls so you can see through them. You might have experienced this before :)

The way to solve this is to provide a limit to how close a player can get to the wall. Using this distance you can calculate the value you should interpolate points to.

The Solution



A player stands at P as close as they can (defined by the distance ) to the wall. The wall is defined by the red line Q to R. The player is looking at angle (from directly at the wall) so that they are looking at point A. The player’s field of view i.e. the view pyramid is represented by the dotted blue lines each angle from PA to intersect the wall at Q’. The line PD divides points in front of the player and points behind the player. Since Q lies behind the player (left of D) it must be interpolated to somewhere between D and Q’ i.e. in front of the player but on or behind the field of view. For now we will interpolate Q to Q’ where Q’ is at a distance in front of the player.

Together:

Considering as a function of i.e. the domain is between:

The reason for this is at:

Q can be interpolated to any point in front of the camera as it will never intersect the view plane

The other condition:

Is just a reflection of the above diagram

Instead of calculating values of for whenever changes, we can determine the minimum value of for any so that Q’ is neither behind the player or inside the field of view.

Plotting in this domain is:



From the graph, the minimum value of is:

This means Q can be safely interpolated to the point Q’ so that it is the same distance in front of the player so long as that distance is between

For example: In my Pygame where:

* players are restricted to walk at least =0.1 units away from a wall
* the players half field of view is

For polygons that are partially behind the player, you can interpolate any points behind the player to a distance in front of the player where:

For ease chose:

A section of my code in main.py is below where, is proximity of 0.1 and is hiddenY which is > \* 1.41

prox = 0.1 # how close you can get to a wall

# this is the local y value that points behind the camera are interpolated to

hiddenY = prox \* 1.41 # see 'Fixing glitchy walls.docx' for explanation