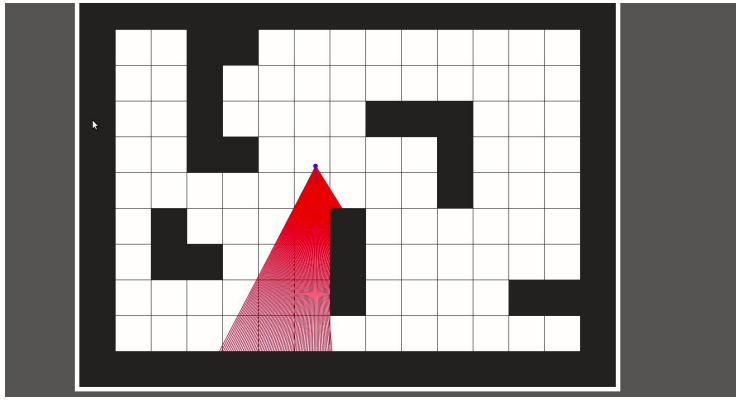
Raycasting in C

Like Wolfenstein 3D & Doom

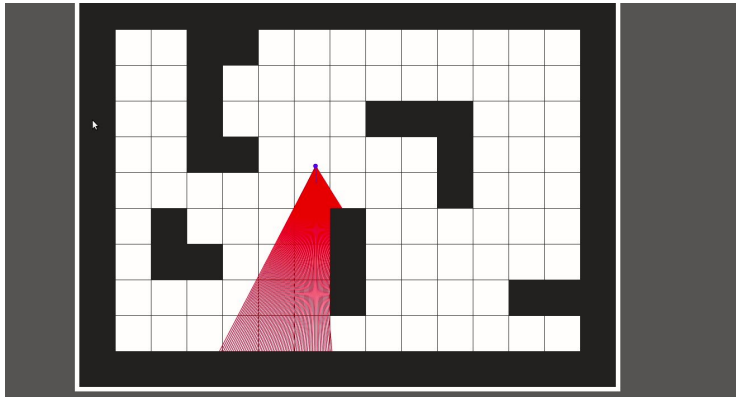
The idea

Raycasting is a process of tracing lines of sight on a 2D plane, and using sightline collision data to render a convincing 3D image.



Imagining the map

The map is a simple 2D array of spaces which are either empty or have a value to indicate a wall. The code uses constants when dealing with the map to dictate the size of the map in pixels - so we have an array of ones and zeroes, but each value in the array represents a position 64x64 pixels large if our TILE_WIDTH/TILE_HEIGHT = 64.

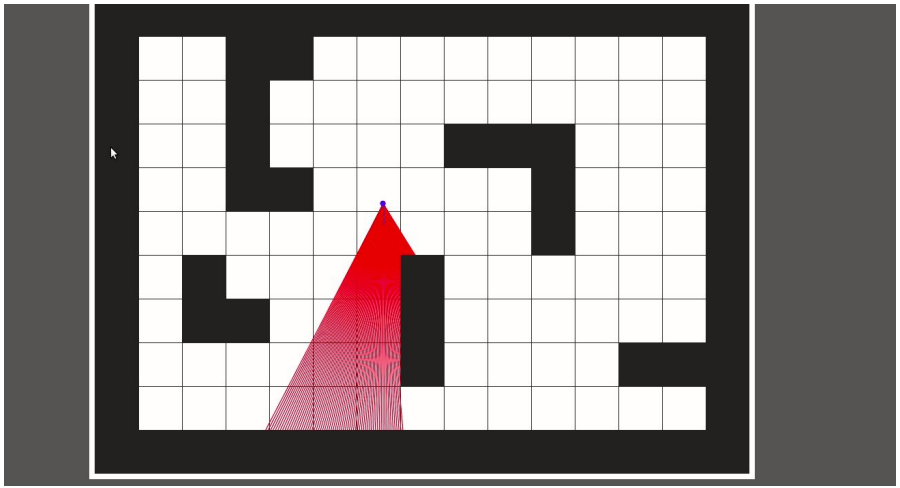


```
const int map[MAP_NUM_ROWS][MAP_NUM_COLS] = {
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1, 1, 1, 1, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 2, 2, 0, 3, 0, 4, 0, 5, 0, 6, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5},
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 5},
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 5},
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 5},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5}
};
```

Casting the rays

Using a bunch of very exciting trigonometry, we can cast straight lines which start from the player's position at angles which cover the player's field of view. When a ray collides with a wall, we save a bunch of data about that ray in a ray array, and with further trigonometry we can render our 3D projection.

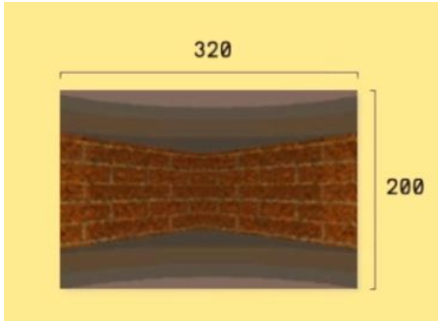
Also, if you're like me and never got a good understanding of why we use both radians and degrees to measure angles… I still don't really understand, but I do know that computer graphics always work in radians.

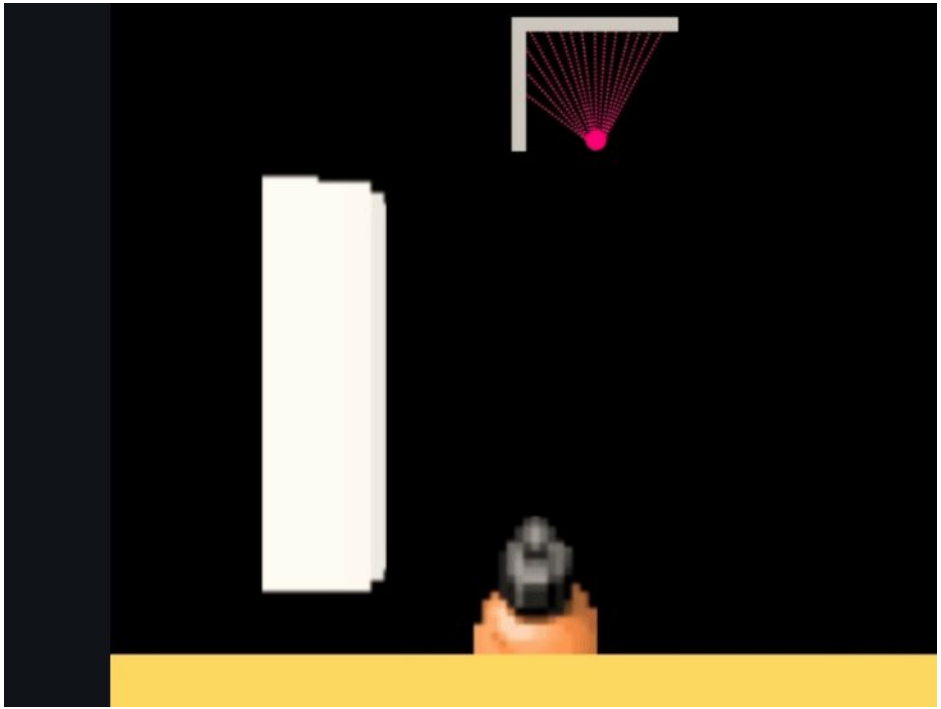


“”3D””

Raycasters have no abstraction of the idea of 3D space in how they render. It is very much a flat, 2D image rendered frame-by-frame to look 3D to the eye. Raycasting simply draws a vertical line for each ray that is cast. Wolfenstein cast 320 rays per frame. The height of each line is dictated by the distance of the ray collision, which is saved when the 2D map collisions are calculated for that frame.

Obviously, no matter how modern a game is, looking at a screen you always see a 2D image, but raycasting is different from modern 3D games because modern games run an internal simulation of 3D space, using 3D models built with flat polygons. No matter how convincing a raycaster looks, you are only ever looking at a row of tall rectangles.



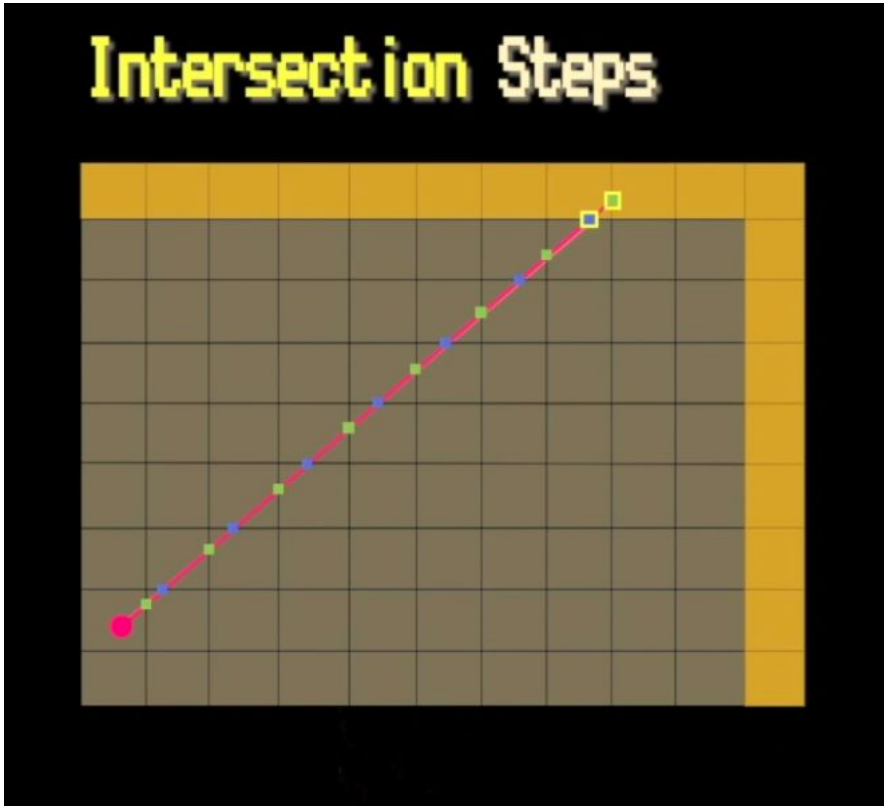


Just a little bit of trigonometry - finding wall collisions

Since we're using a 2D grid for our map, we can check for ray collisions along each grid intersection, instead of pixel-by-pixel (which would be ridiculously expensive).

So, where green dots are vertical intersections and blue are horizontal, we have to calculate the points where the first blue and green dots intersect with the grid. After the first horz/vert intersection, the distance between intersections will always be the TILE_SIZE (which in our example is 64).

The calculations required depend on whether the ray hits a vertical or horizontal intersection first.



Finding wall collisions

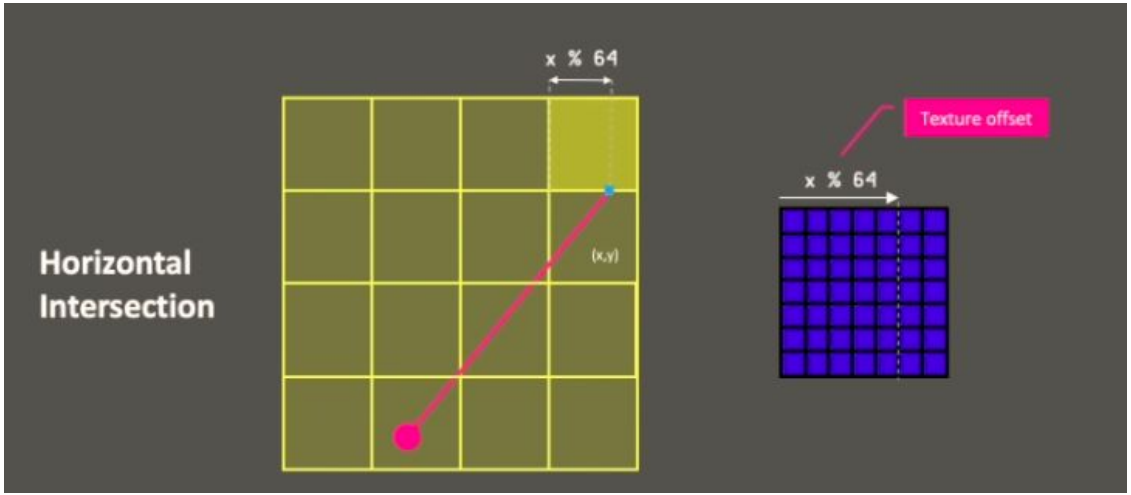
We run loops to find all horizontal collisions up to a wall, and then all vertical ones.

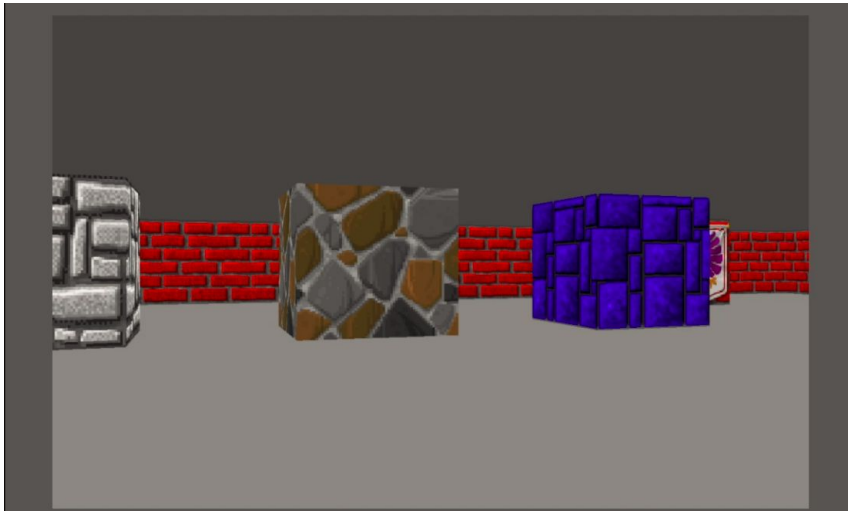
The shortest of the two distances will then be the ‘real’ position of the wall, as seen here.

Textures

Textures are stored in a byte array, and the color for each individual strip of wall is simply chosen out of that byte array based on a modulus operation with the position of the wall strip.

So for example, if the ray hit the wall horizontally at x position 65, then because each cell is 64x64 pixels, we know that the wall strip for this ray should use the color values at byte_array[1], since 65 % 64 = 1.





The executable!

There was a link here. It’s gone now.

RayTRACING!?!?

Raytracing is currently a big point of discussion in games because the next Xbox has hardware-inbuilt raytracing.

Raytracing is the practice of casting a ray for ***every single pixel*** of the screen and tracing how it bounces around the scene off of ***many*** objects, ***just to figure out the intensity of the lighting and reflectivity of objects***.

Reminder that a 4k screen has 8.3 million pixels. Wolfenstein 3D cast 320 rays and used that information as the basis of its' entire graphics rendering!