This application is to perform an 3D sight when the player walks through a maze.

Here’s how I implement it.

Step 1. I divide the viewer’s FOV (field of view) by 25,000 to get 25,000 rays and then call the “Clip\_To\_Cell” function to get the point crossed by the ray passing through the viewer’s position and the first opaque wall.

Step 2. Upon getting the point, I check which wall the point is on and record the wall’s index. While getting all the point those 25,000 rays pass through, if I get a different wall’s index from the previous recorded wall’s index, I will draw the wall by its starting point, ending point, starting distance, and ending distance. (The starting/ ending distance means the distance between viewer’s position and the starting/ ending point.)

Step 3. I draw the quadrilateral by calculating the point projected on viewer’s sight. I don’t use any matrix, just pure calculation. Because it’s hard to illustrate without being face-to-face, you can see the source code to know what I’m doing.

Step 4. Repeat Step2 and Step3 till all the rays have been performed.

There are a few little problems about this method. First of all, there can be some little gap between wall and wall because of the insufficient rays. I avoid this problem by dividing the viewer’s FOV into more pieces/ rays. I also record the last position I draw on the screen and ignore the next starting point’s X position to connect two walls. Second, there are some little bugs in “Clip\_To\_Cell” function, so when a ray passes near a junction of two walls, it may be clip into another cell and set “xe” and “ye” on to a point of another wall. I avoid this problem by adjusting the “min\_crossing” value in that function and not drawing all the walls which can’t be clipped to over 100 times.