By Andrew Weller and Trey Nevitt

Demise, a 3d Game Engine and Game

Many video games today utilize prebuilt libraries and game engines that make game development far more accessible to hobbyist and AAA studios alike. Common 3D libraries like OpenGL and DirectX are used by lots of developers, but how exactly do they work? The goal of our project is to develop a rendering engine that can then be used to create our own game.

How do you render a scene in 3d at all though? This is done with a process called 3d projection. All it does it it transforms 3d coordinates (x,y,z) relative to an observer into 2d coordinates (x,y). This can be done simply by doing a "homogeneous transform" and diving all three coordinates by z. This results in a new vector ((x/z), (y/z), 1). Now we have two unique x and y coordinates (x/z, y/z) that can be used to draw the vertices of polygons or circles at that point onto a 2d plane.

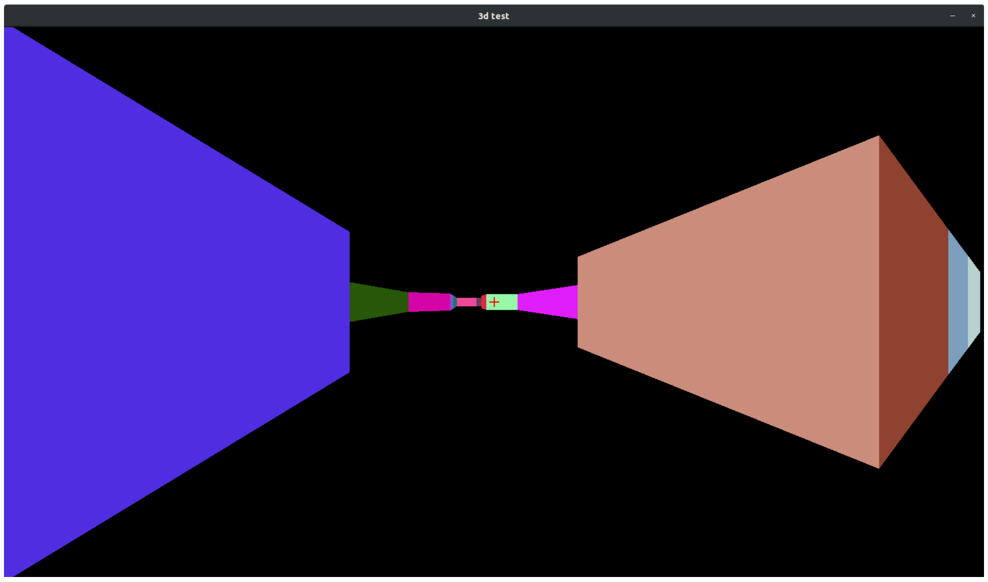
In order to draw these polygons to the screen we will be using an external graphics library "SFML" (Simple Fast Multimedia Library) for c++. This library gives us tools to create a window that we can draw polygons to in 2d, which is exactly what we need for our game.

Now this is all great, but this is a data structures class, so where does the data structure come in? Notice that above are instructions for drawing a single polygon. But our level is not just a single polygon, it has many polygons. This is a problem because now we might be projecting the polygons correctly in 3d, but they might not be in the correct order on the screen. We don't want to draw something that is behind something else on top of it. That is not an accurate portrayal of our 3d world.

The solution to this problem is using a special kind of binary search tree. It is called the binary space partition. It has a simple structure. For one node of our tree, all nodes to the left of it in the tree are behind it, and all nodes to the right of it in the tree are in front of it.

Now that we've built a tree in this manner, drawing polygons in the correct order is trivial. All we have to do is check if the player is in front of the root or not. If the player is in front of the root of the tree, first we draw the sub-tree which is behind it, then we draw the polygon at the root, then we draw the sub-tree in front of it. These sub-trees are drawn recursively in the same fashion as above.

This can be used to create visuals like this:



Each of the polygons you see on screen are walls in a game level. This is ultimately what the game will tend to look like in the end, and it is rendered using this technique.