Multilevel raycasting algorithm

Joseph21

2022-12-24

Algorithm overview

Step 1 – extend the ray tracing function / code to create a list of hitpoints. So don’t stop at the first hitpoint, but only stop at the boundaries of the map. Record all hitpoints where the height of the block before and after the hit point differs.

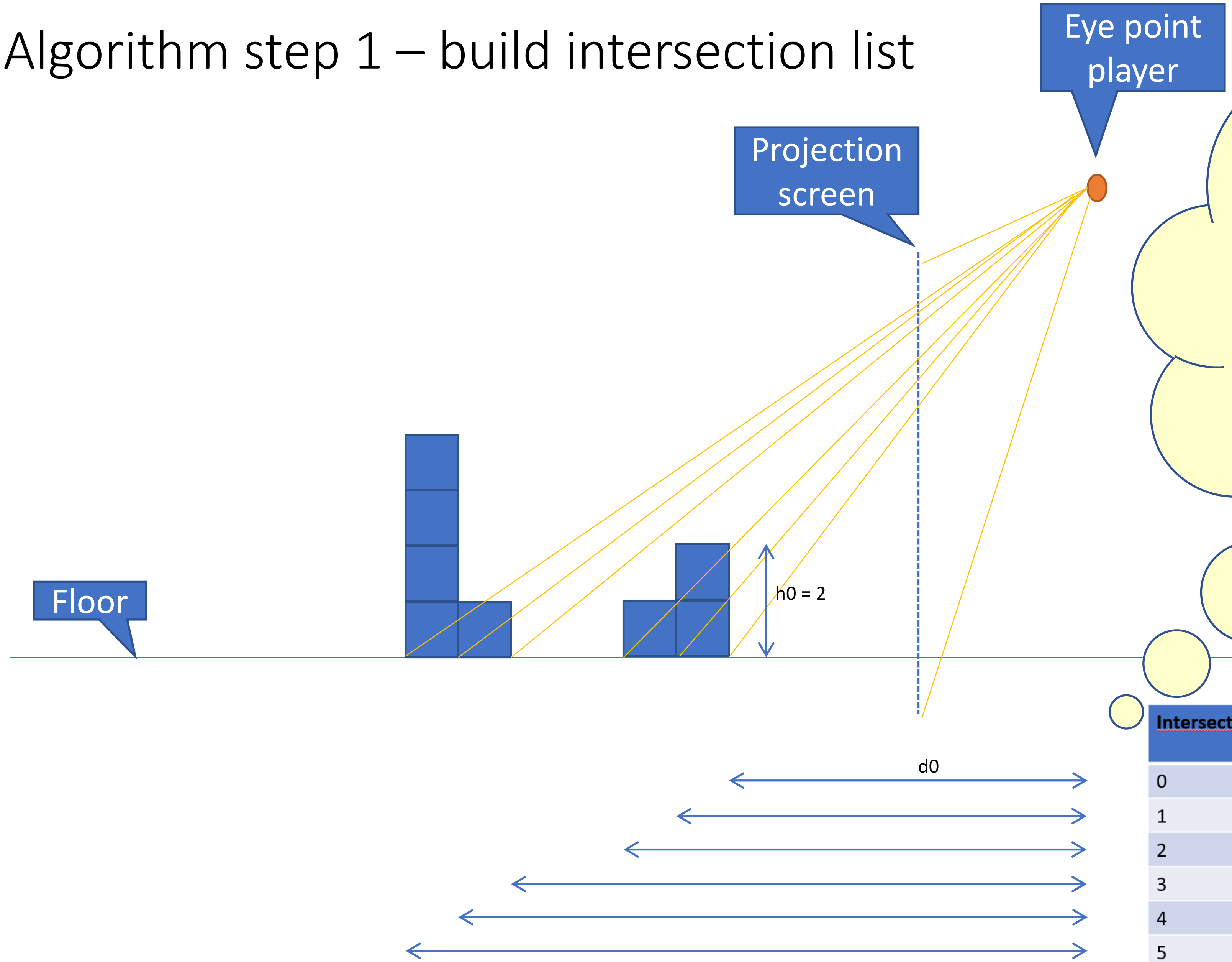
So this must be done in the ray tracing function/code.

Step 2 – you can calculate how a wall segment projects onto the screen (this was already used in regular ray casting). Use the same approach to extend the list of hitpoints with the info on how each hitpoint is projected onto the screen.

You can choose if you put this in a separate function, or integrate it in the rendering function. Either way could work.

Step 3 – When rendering the slices on screen, you use the hitpoint list to determine how to render each pixel on the screen. **This must be done in the rendering function / code.**

Algorithm step 1 – build intersection list



Do the raytracing, but don't stop at the first intersection: Continue tracing until the ray gets out of bounds, and build a list of intersections where the height of the level before and after the intersection differs.

Store the distance from the player to the intersection, and the new level height after the intersection for each entry in the list. For intersections on the map boundary this value should be 0.

height at intersection – means height of the block that is entered at intersection

Intersection #	Distance to player	Height at intersection
0	d0	2
1	d1	1
2	d2	0
3	d3	1
4	d4	4
5	d5	0

Algorithm step 2 – extend intersection list

Black content – initial table content, filled after ray casting
Red content – added using projection calculations

Intersection #	Distance to player	Height at intersection	Projected bottom	Projected ceiling front	Projected ceiling back
0	d0	2	b0	c0	e0
1	d1	1	b1	c1	e1
2	d2	0	b2	c2	e2
3	d3	1	b3	c3	e3
4	d4	4	b4	c4	e4
5	d5	0	b5	c5	e5

Use the formula to project block bottom and ceilings onto the screen. For the bottom, only the projected front of the block (at intersection) is relevant. For the top of the block (the ceiling), not only the front but also the projected back of that block is relevant and is stored in the extended list.

en = projected ceiling height calculated with distance dn+1 iso dn, so the last e value is meaningless

Algorithm step 3 – render slice using the intersection list

```
// y is the screen height value of the pixel that is rendered
if (y > bi)
       render floor
else if (bi >= y > ci)
       render wall (using distance di)
else if (ci >= y > ei)
       render roof
else {    // ci, ei > y
    Try next point (i + 1) from intersection list with same criteria
    If (no next point available) → ceil
}
```