Multilevel raycasting algorithm

Joseph21

2024-02-04

(have fun with it ☺)

Algorithm overview

Step 1 – Regular ray casting stops upon finding the first hit point (or the boundaries of the map). For multilevel ray casting you have to create a list of hit points. So don’t stop at the first hit point, but only stop at the boundaries of the map. Record all hit points where the height of the block before and after the hit point differs.

This can typically be implemented in the ray casting code.

Step 2 – Using Permadi you can calculate how a wall segment (or a block) projects onto the screen – this technique was already applied in regular (single level) ray casting. Use the same approach to extend the hit point information with the info on how each hit point is projected onto the screen.

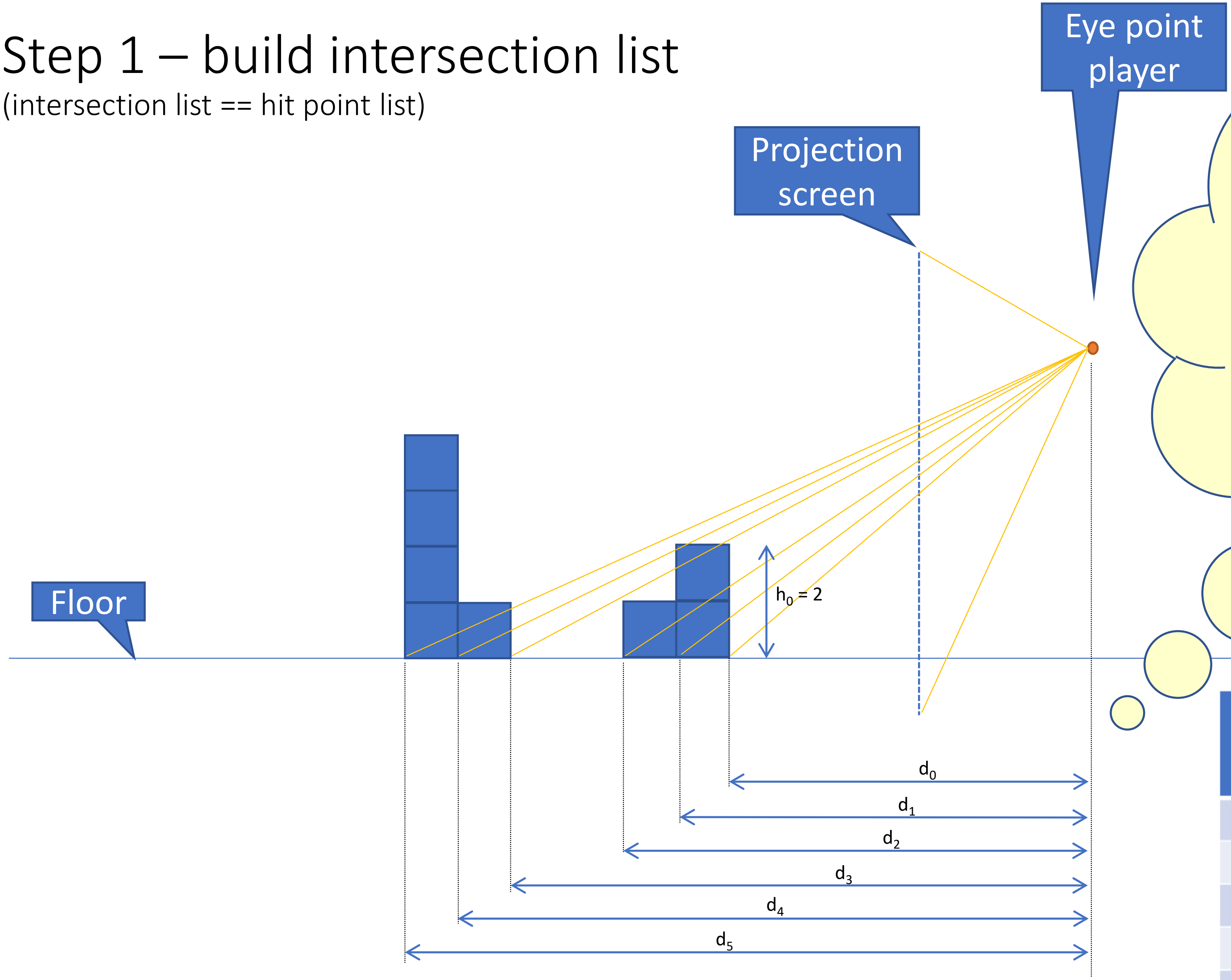
You can put this in a separate function, or integrate it in the rendering code.

Step 3 – When rendering the slices on screen, you use the hit point list to determine how to render each pixel on the screen.

This is typically done in the rendering code.

Step 1 – build intersection list

(intersection list == hit point list)



Do the ray casting, but don't stop at the first intersection (= hit point): Continue analysing the ray until it gets out of bounds, and build a list of intersections where the height of the map cell before and after the intersection differs.

Store the distance from the player to the intersection, and the new map cell 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 that intersection point

Intersection #	Distance to player	Height becomes at intersection
0	d_0	2
1	d_1	1
2	d_2	0
3	d_3	1
4	d_4	4
5	d_5	0

Step 2 – add projection info per hit point

Intersection #	Distance to player	Height becomes at intersection	Projected bottom	Projected top (front)	Projected top (back)
0	d_0	2	b_0	c_0	e_0
1	d_1	1	b_1	c_1	e_1
2	d_2	0	b_2	c_2	e_2
3	d_3	1	b_3	c_3	e_3
4	d_4	4	b_4	c_4	e_4
5	d_5	0	b_5	c_5	e_5

Use the Permadi formula to project block bottom and top onto the screen. For the bottom, only the projected front of the block (at intersection) is relevant. For the top of the block, not only the front but also the projected back of that block is relevant and is stored in the extended collision list. This info is needed to render the roof of a block that is viewed from above.

e_n = projected top height calculated with distance d_{n+1} iso d_n, so the last e value is meaningless

Black text – initial table content, filled after ray casting

Red tekst – added projection info using projection calculations

Step 3 – render slice using the intersection list

```
// y is the screen height value (vertical coordinate) of the pixel that is rendered
// values for b, c, d and e are as defined in previous slides

if ($y > b_i$)
        render floor
else if ($b_i >= y > c_i$)
        render wall (using distance $d_i$)
else if ($c_i >= y > e_i$)
        render roof
else {    // $c_i$, $e_i > y$
    Try next point (i + 1) from intersection list with same criteria
    If (no next point available) → ceil
}
```