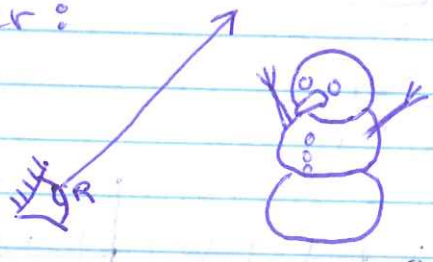
# UNDERSTANDING GRID ACCELERATION

The idea behind grid accelleration is that we want to reduce the number of objects we test against with the intersection of a ray.

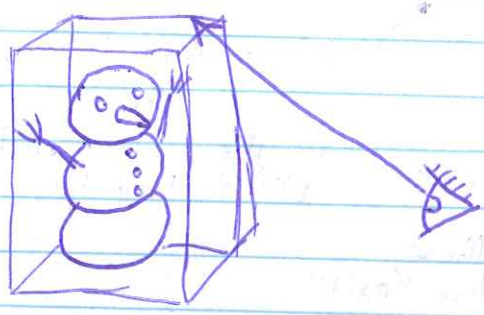
Consider:




unoptimized

R must check collisions with all primitives in the scene

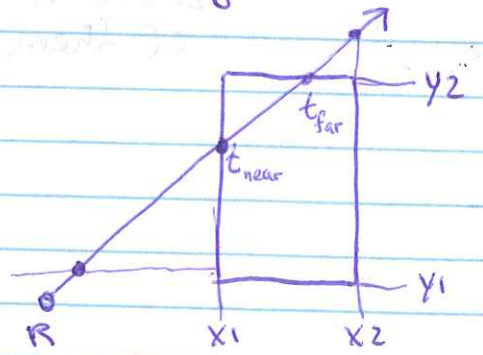
O(num_objects)



Now imagine with a bounding box, if we consider this ray passing behind it, all the primitives in this object can be ignored O(1)

↳ we want to avoid false positives (snug fitting box) but also quickly compute intersections

→ with this we can get the idea of an axis aligned box (grid):

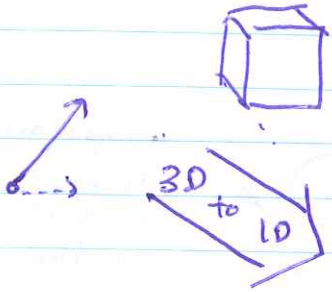


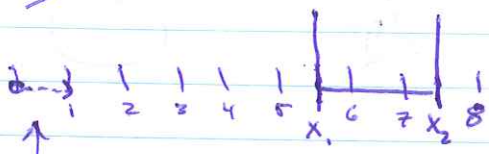

for each axis direction:
calculate $t_1$ and $t_2$

$t_{near} = \max(t_{1x}, t_{1y}, t_{1z})$

$t_{far} = \min(t_{2x}, t_{2y}, t_{2z})$

Basically what this algorithm does is check how many (xyz) components of the direction vector are needed to cross the bounding box




$t_{1x} = 5.5$

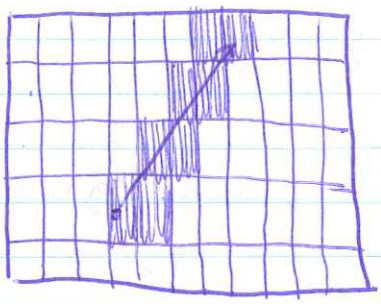
$t_{2x} = 7.5$

if we take the max of all these, that is the "first" intersection point t-value

if we take the min of all these, that is the "exit" intersection point

an intersection can only occur if the vector goes "into" the bounding region for all t-values before it "exits" any of them.

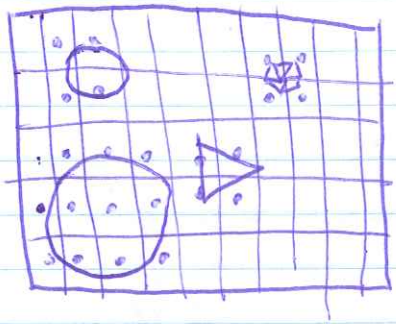
With grid acceleration, we want to "march" our ray through 3D space:



instead of checking 55 areas we only check 8 with this ray

↖ grids needn't be cubes

The grid forms our most basic data structure which each cell has pointers to the list of primitives which occupy it (note: the same object may be present in multiple cells)



↓ This way we 'march' through the grid and ~~only~~ if it contains a hit, we stop and return the nearest hit, if not, move to next grid march.

↳ if the object isn't hit mark it so it won't be checked again → (recheck if there is a hit, but not in this grid)

Hilroy