Game Notes

Aresh Pourkavoos

May 5, 2022

Position within square stored as an odd signed integer in half-pixels, e.g.

101	011	001	011
-3	-1	1	3

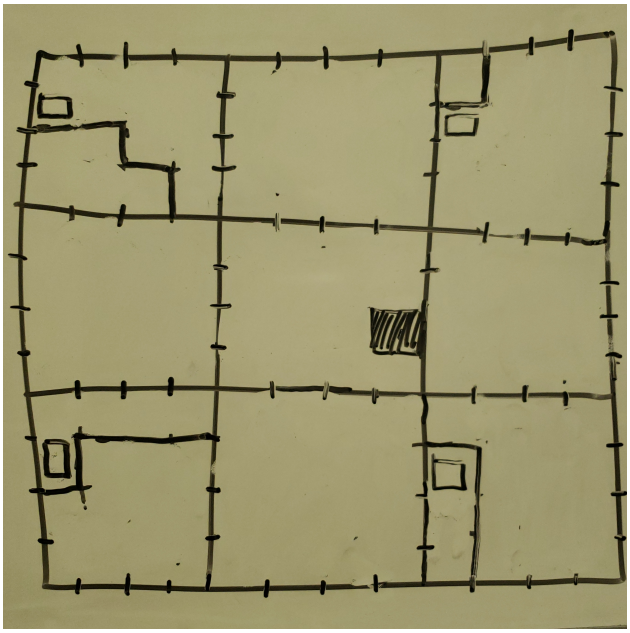
Requires entities to have odd pixel dims to be centered
Edge/vertex states are not possible
Updating position requires doubling velocity first
Store position as x and y seen on screen or relative to a square's axes?
Screen position:

- Graphics and movement are easier
- Collision would be most convenient by loading the current square rotated

Relative position:

- Collision is easier, just check against stored square
- Need to ensure that rendering is done correctly

View splitting is decided by determinant sign: will always give edge to cell further (counter?)clockwise
Vertices/edges are on the border between pixels (even position): do not require a special case



Shaded pixel: camera
Lines: region boundaries

Inlined pixels: edge cases (given to clockwise region in this case)
Going through a singularity and back is a holonomy loop
Entity gravity is ambiguous when not in the same square as the player:

- Freeze when player leaves the square: unintuitive, esp for flat regions
- Based on last player interaction: better, but initial direction must be set: could be none

Have "naked" singularities or cover them up?
Naked is easier to implement if accounted for at the cost of real physics:
an object of finite size can't actually pass through one
Not checking self-collisions would obviate this but may result in graphical glitches
Covering singularities would prevent glitches and restore accuracy but might hurt level design
Larger squares → fewer singularities → less harm in covering them
Also, must be regions accessible in only one orientation: side longer than 2x jump height
However, smaller squares → more convenient to travel/execute holonomy
Leaning towards covering, especially since multi- entity collisions near them can get hairy

Render method (pixel shader):
Build a tree of regions, starting from current square
Region info: left and right boundary points, square rendered, position, and orientation of square
Region is split if strictly contains singularity (i.e. not on edge)
4 quadrants constructed separately, contains current square 4x and squares in same row/col 2x
Build tree using an array/queue for BFS
Regions in the same position are contiguous in the queue
Store index of first region in array
Given a pixel, find position, access array for queue index, iterate through queue until pixel is in bounds
Apply orientation of region to find color
"Raycasting" each pixel is slower than this, but the function will be useful later for collision resolution

Art style between pixel and vector: each "pixel" is not a solid color but one of a few predefined shapes, e.g. solid color, 2 colors split diagonally, split by circular arc

Physics
Single entity moving in static background: all collisions are with axis-aligned surfaces
Collision multiplies perpendicular velocity by integer ≤ 0, keeps parallel velocity constant
Intersection point could be any rational number, so can't store intermediate positions
Instead, reflect both beginning and end points across plane of collision (adjusted for size of player), recursively/iteratively resolve collision with "mask," i.e. ignoring portion of trajectory "before" hitting wall
Collision state consists of beginning/end points and rectangle to check
Initialized to previous frame position and rectangle encompassing path
E.g. moving up+right: rectangle starts at lower left corner of starting position, extends to top right of ending position
Collision with vertical surface changes only x-coordinates of beginning, end, and quadrant
But this can cause bugs with large players and irregular walls: maybe rational coords are the easiest
45 degree slopes are possible as well, as long as coefficient of restitution (bounciness) is odd
Doesn't work with pixel buffer for collisions: need to add "static" entity

Game loop
Build region tree
Take player input
Update velocity accordingly (also gravity here?)
Resolve collisions
Render