Game Notes

Aresh Pourkavoos

April 26, 2022

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

101	011	001	011
-3	-1	1	3

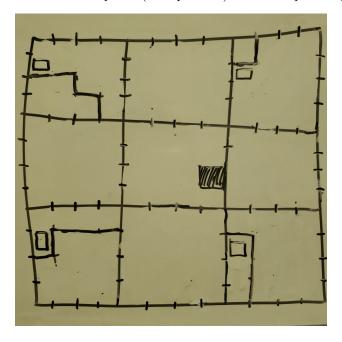
Requires entities to have odd pixel dims to be centered Edge/vertex states are not possible Updating position requires doubling velocity first Store position as x and y seen on screen or relative to a square's axes? Screen position:

- 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 \rightarrow fewer singularities \rightarrow less harm in covering them However, smaller squares \rightarrow more convenient to travel/execute holonomy

Rendering function is recursive:

- Accept left+right boundary points, square to render, position/orientation of square, quadrant
- Render given square in full
- Check if furthest vertex given by quadrant ("splitting vertex") falls strictly within bounds
- If not, call with same bounds/quadrant, single adjacent square with new position/orientation
- If so, make 2 recursive calls changing appropriate bounds to splitting vertex
- Store the result of the 2nd call in a separate buffer
- Could maintain a stack of buffers and add depth as an argument: depth necessary anyway
- Mask buffer with line between camera and splitting vertex (anti-aliasing here!)
- Overlay with original buffer

4 base cases starting from current square with left/right bounds along x/y axes Renders current square 4x and squares in same row/col 2x

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