ComS 455: Homework 2b
Magnets

1 Introduction

In this homework, you will investigate using scale, rotation, and translation transformations to produce an interactive magnet simulation. Each of two magnets will be composed entirely of four instances of a rectangle (one large rectangle for the overall magnet body, one small one marking the negative pole, and two small ones marking the positive pole). When a user drags a magnet close to the other, the undragged magnet is attracted or repelled, depending on which poles are nearest.

2 Requirements

To receive full credit for this assignment, you must:

- Render two magnets, each as a horizontal unscaled rectangle overlaid with three smaller scaled rectangles labeling the poles. The negative pole is labeled with a scaled down instance of the rectangle, appearing as a '-' sign. The positive pole is labeled similarly, but with an additional scaled and rotated rectangle to form a '+' sign. The labels should be small enough to fit within the magnet and colored distinctly different from the rest of the bar.
- Use only one vertex array object. Apply matrix transformations to this vertex array to produce the magnet bodies and their poles.
- Magnets can only be translated and flipped. (Rotation is not supported by the simple physics model available in the provided code.) On translation, the moved magnet should alter the other magnet according to the poles and their distance from each other. Upon a flip, the flipped magnet must interact with the other as necessary. For example, flipping a joined magnet will result in repelling the other, as the similarly charged poles will lie adjacent. Map your user input events to mouse interaction like so:
  - On any mouse click, determine which magnet was clicked, if any.
  - On a left mouse drag, move the magnet according to the amount of drag and have it interact with the other.
  - Respond to a right-mouse click on a magnet by flipping its poles and having it interact with the other.

3 Magnet Struct

You are encouraged to use the following `Magnet` pseudocode, which encapsulates all the details of a magnet, including a `contains` method which determines if a specified position (e.g., a mouse click position) falls within the magnet's bounds and an `interact` method which adjusts the other magnet's position. Alter it at will.

```
enum direction (left, right)
enum pole (positive, negative)

struct Magnet
  const real WIDTH = 150
  const real HEIGHT = 50
  const real THRESHOLD = 90

  Vector2 position # center of magnet
  boolean is_positive_left # default to true

  flip
    is_positive_left = !is_positive_left

  boolean contains(int x, int y)
    position.x - WIDTH / 2 <= x <= position.x + WIDTH / 2 and
    position.y - HEIGHT / 2 <= y <= position.y + HEIGHT / 2

  Vector2 get_pole_position(direction d)
    if d is left
      position + (WIDTH - HEIGHT) * -0.5, 0)
    else # d is right
      position + (WIDTH - HEIGHT) * 0.5, 0)

  interact(Magnet that) {
    # First find which end of each magnet is influencing the other. If this
    # magnet is shifted right past at least 25% of that magnet, then its active
    # end is its left end. Similarly, if this magnet is shifted left past at
    # least 75% of that magnet, that magnet's active end is its left end.
    direction this_active_end = position.x > that.position.x - WIDTH / 4 ? left : right
    direction that_active_end = position.x < that.position.x + WIDTH / 4 ? left : right

    # Now we must determine the polarity of the active ends.
    if this_active_end is left
      this_active_pole = is_positive_left
    else // active end is right
      this_active_pole = !is_positive_left

    if that_active_end is left
      that_active_pole = that.is_positive_left
    else // active end is right
      that_active_pole = !that.is_positive_left

    # From here on out, we need only consider the active ends. How far are they
    # from each other?
    Vector2 diff = that.get_pole_position(that_active_end) -
                   get_pole_position(this_active_end)
    float distance = diff.length

    # If they are close, we have to move that magnet. How we move it depends on
    # the polarity of the two ends.
    if distance <= THRESHOLD

      # If the polarities are the same, we push that magnet away an amount
      # proportional to the gap between them.
      if this_active_pole == that_active_pole
        that.position += diff * ((THRESHOLD - distance) / distance)
      else

        # If the polarities are opposite, we link the two magnets together.
        # We start by completely aligning them.
        that.position = position

        # If the magnets are aligned horizontally, the y gap will be small
        # and the x gap will be large. So shift that magnet to the left or
        # right of this magnet.
        if |diff.x| >= |diff.y|
          if diff.x < 0
            that.position.x -= WIDTH
          else
            that.position.x += WIDTH

        # Otherwise, the magnets are more or less aligned vertically.
        else
          # If opposite ends of the magnets are attracting, then only
          # one pole of that magnet will link to this magnet. We'll have
          # to shift that magnet over to stagger the alignment.
          that.position.x += get_pole_position(this_active_end).x -
                             that.get_pole_position(that_active_end).x

          # Shift that magnet up or down to link to this one.
          if diff.y < 0
            that.position.y -= HEIGHT
          else
            that.position.y += HEIGHT
```

4 Submission

Push your code to Bitbucket. Post an example image or two on the course blog. Describe your thought processes and any hurdles you had to overcome. Use categories `spring 2015`, `cs455`, and `postmortems`.