Peter Peirce

Final Assignment

Basic Controls

Adding Circles

How to do it

Press and hold the left mouse button where you would like a circle to be placed. As you are holding the button, drag it away to change the size of your circle. When you release the mouse button, the position and radius of the circle are saved. To add a velocity to this circle, move the mouse in the direction you would like the circle to travel, then click and release. Moving the mouse further will result in a greater velocity being applied to the circle.
Rinse and repeat.

How it works

When a left mouse button input event is detected, the next two positions in the circle[] array are updated with the x and y coordinates of the mouse. While the mouse is depressed, the cursor position input event is calculating the difference between the mouses position when the left mouse button was pressed down and its current position. When the left mouse button is released, this distance is saved in the radius[] array. Next, the cursor position input event calculates the change in position in the x and y directions from the end of the previous event until the left mouse button is again pressed. The change in position is saved as two values, for the x and y, in the velocity[] array.

Pausing the Scene

How to do it

Press the “p” button.

How it works

When “p” is pressed, a boolean is flipped. This then skips the function call that updates and applies the velocity to the circles position.

Reseting the Scene

How to do it

Press the “r” button.

How it works

When “r” is pressed, a series of functions are called which set all values and arrays to their initial state. This places all circles off-screen and switches their radii to 0. Hint: reset the scene before the 10th circle is drawn.

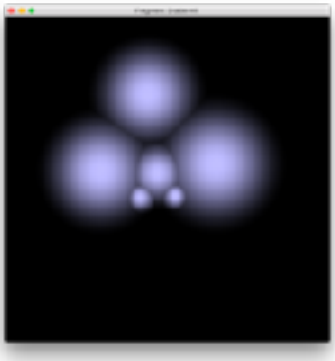
Changing Render Mode

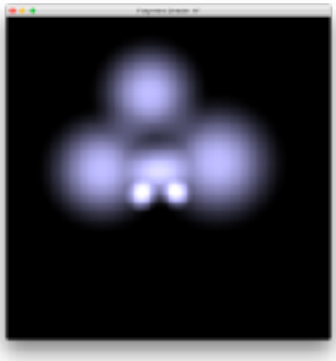
How to do it

Press the “e” button.

How it works

When “e” is pressed, an integer flips between 0 and 1 (obviously this could be done with a boolean, but one imagines building on this code at a later time and adding additional rendering modes). A uniform int is passed to the fragment shader which is used to determine the way in which the fragment colors are determined. The default mode calculates the max intensity of each pixel, for each circle, in the scene. This results in a more segmented appearance. Switching modes calculates the sum of each pixel, for each circle, in the scene. This results in a more fluid appearance.





Adding Gravity to the Scene

How to do it

Press the “w” button.

How it works

The “w” button toggles a boolean that changes the function used to update the velocity and position every time the window is drawn. The default mode simply adds the velocity to the position. Gravity mode first modifies the velocity by a small fraction of the distance from the circle to the center of the window before applying the new velocity to the position.

How the Rendering is Handled

All drawing is done in the fragment shader. Only two triangles, taking up the full view of the window, are sent to the vertex shader.

In the fragment shader, two arrays defining the position and radius of each circle are sent as uniforms. This data is then read into an array of structs defining 10 circles. Individual circles are rendered using a distance field with their edges defined using a smoothstep function. Depending on the rendering mode, the scene is put together either by summing or finding the max of each circle in the scene.