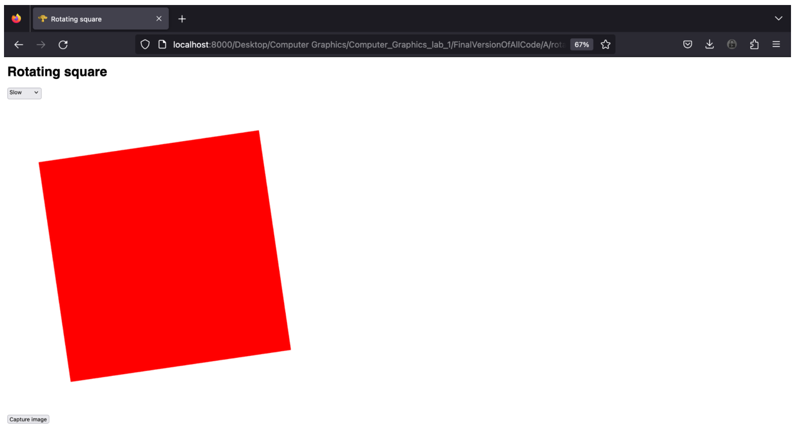
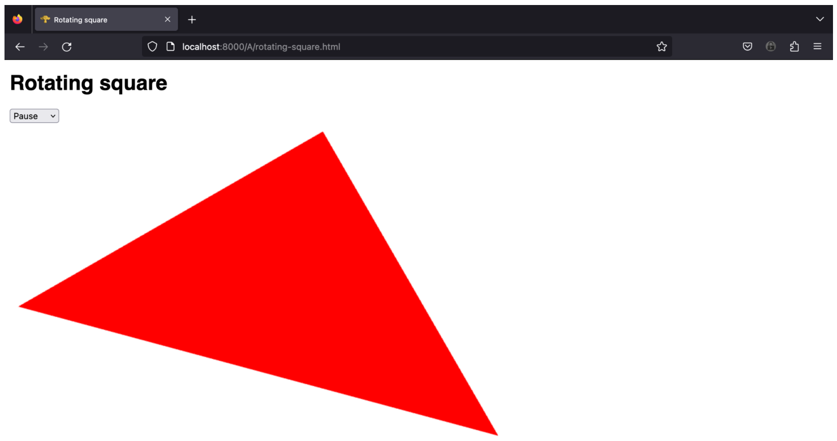
Computer Graphics - Lab 1


Rotating square
localhost:8000/Desktop/Computer Graphics/Computer_Graphics_lab_1/FinalVersionOfAllCode/A/rota
67%
Rotating square
Slow
Capture image

A0:

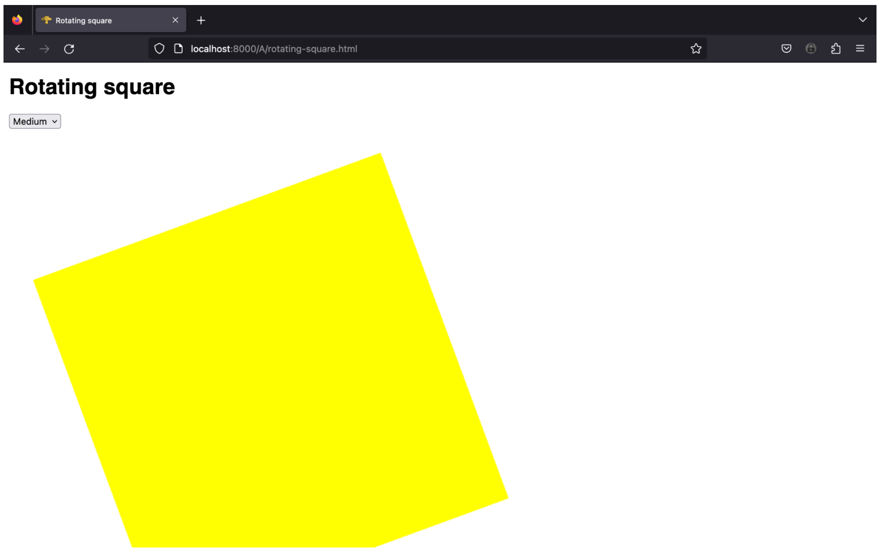
In this code, I had gotten the set speed function to work, setting the speed to the required 0.005,0.01, 0.02 as said in the lab sheet. I also added a pause function which sets it to zero and I set that as the default.



A1:

In the code, I had changed the num_strip_vertices to 3. This means that I had changed the amount of vertices the shape I had constructed had from the variable num_strip_vertices to 3. This was then used as an argument for the function gl.drawArrays(). As seen from the image, this led to the shape being transformed from a square to a shape with 3 vertices (that being a triangle).

A2:

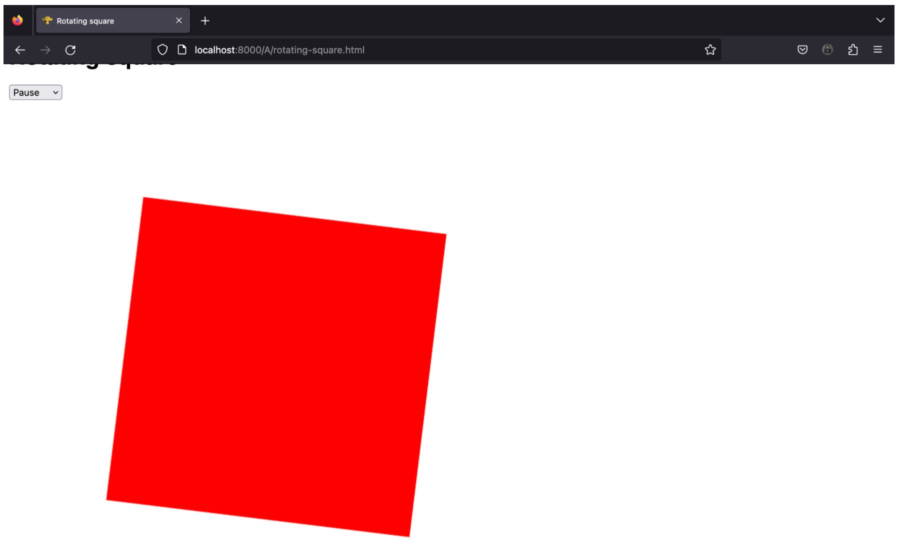


Rotating square

Medium

As I created two vec3 variables, one with the RGB colour red and the other with the RGB colour green, I had added them together using the '+' operator to form another colour. With one red and one green added together this created yellow as it would have been the same as vec4(1.0, 1.0, 0.0, 1.0), which in RGBA is saying combine R + G. This formed what could be seen above the yellow colour.

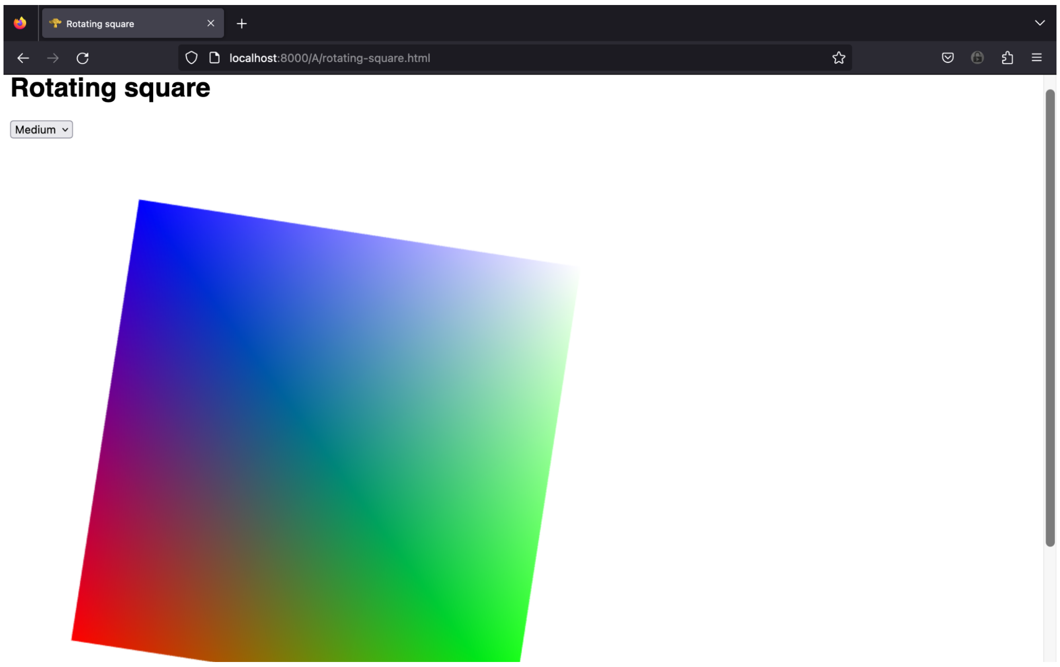
A3:



Pause

The animation of the square now whilst rotating also shrinks and enlarges repeatedly, this occurs at the speed which is set too. This means that its rate of enlarging and shrinking increases or decreases too depending on the selected speed. This is done with the line of code: gl_Position.x *= (1.0 + s) / 2.0; and gl_Position.y *= (1.0 + s) / 2.0; (s referring to sin(theta)). What this does is, it changes the x and y position of the vectors so that it increases and decreases in size (it multiplies by the initial value to do this).

A4:

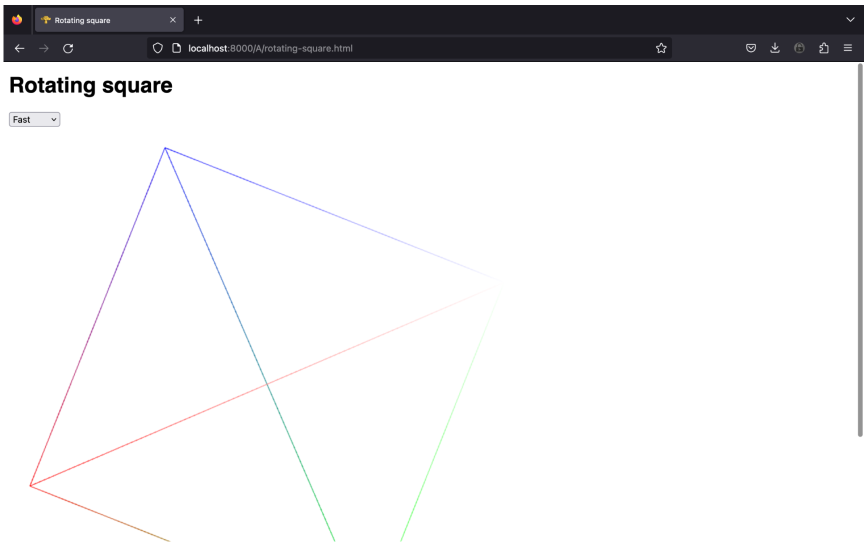


Rotating square

The animation now remains the same, but now displays the rotating square with multi-colours blending into one another, with each corner being one part of the RGBA values so red in one corner, blue in another, green in another one and the final corner begin opacity which can be seen as the opacity decreases as it goes closer to that corner. This works as each vertex has a colour and in between each vertex the colours are interpolated using the varying variable we had declared previously and set the gl._fragcolour to. After this we made use of gl.vertexAttribPointer(colour_loc, 4, gl.FLOAT, false, 0, 0); to essentially say that colour_loc has 4 gl.float components which are not normalised and gl.enableVertexAttribArray(colour_loc); to enable the attributes. This allows us to call drawarrays() to create the actual primitive

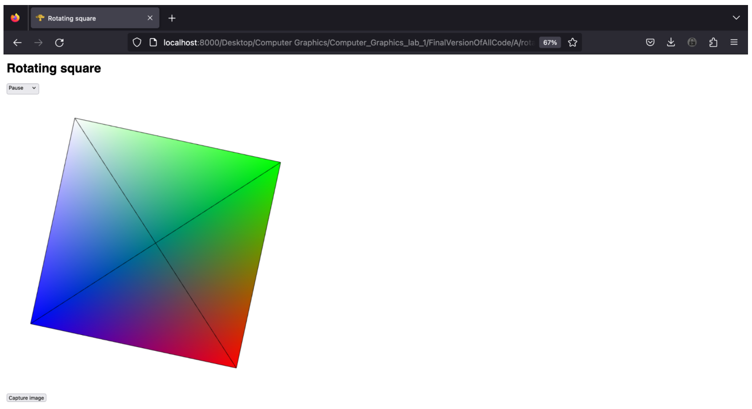
.

A5:



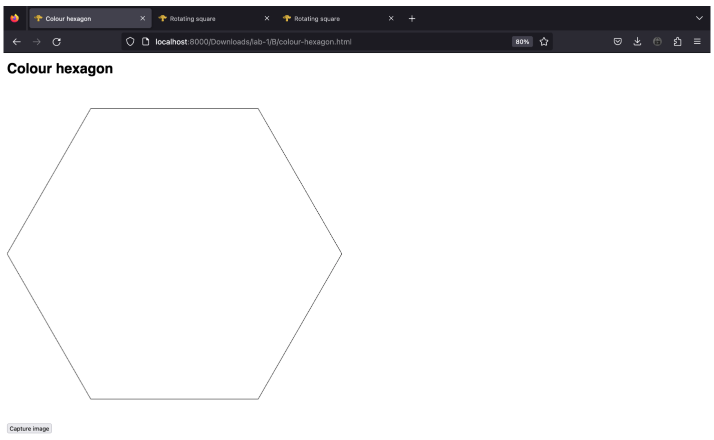
Here we have removed the actual inside of the square and only have the lines and x in the middle. This is done by connecting the vertices to each other by stating which one to connect (e.g. [0,1,0,2,...]). This is done by making use of gl.lines in the drawElements() function, this allows the vertices to connect in the method I had stated above.

A6:


Rotating square
localhost:8000/Desktop/Computer Graphics/Computer_Graphics_lab_1/FinalVersionOfAllCode/A/rot
67%
Rotating square
Pause
Capture image

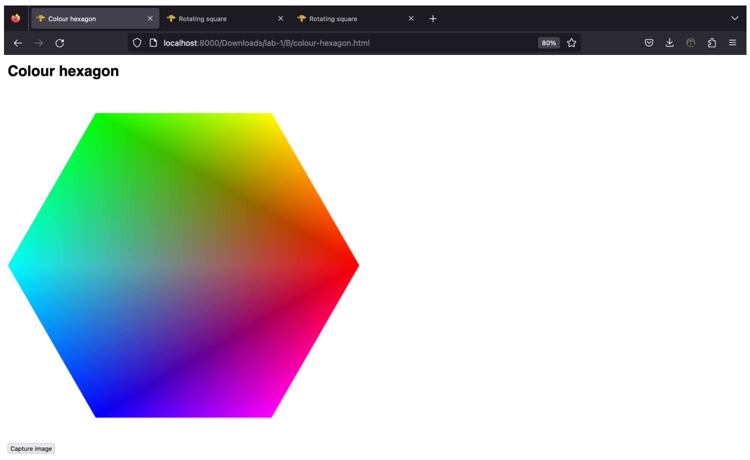
Here as you can see, the lines of the previous part (A5) have been changed to black, and in the actual square the colours are the same as how it was in A4. The reason why the line colours are constant whilst the varying colour isn't is due to the fact that they are drawn at different points each after different gl.vertexAttribPointer()'s are called one of which has the black _offset. This is used so that the correct number of bytes would be skipped to reach the new black entries we had added.

B1:



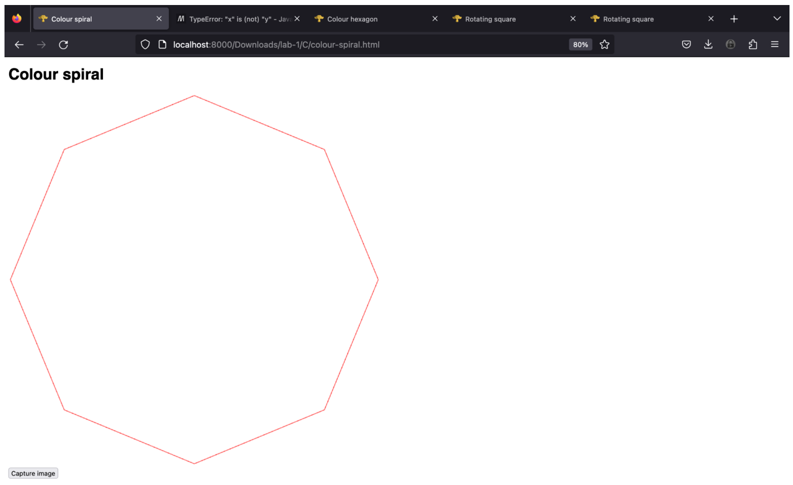
Here as you can see, I have drawn a basic hexagon with the sides set to a length of one. This was done by dividing 2π by the number of vertices to get the number of degrees to add each time to get to the next vertex. This was then used in a for loop with sin and cos to figure out the xy coordinates for the next vertices until 2π is reached which would be the original starting point.

B2:



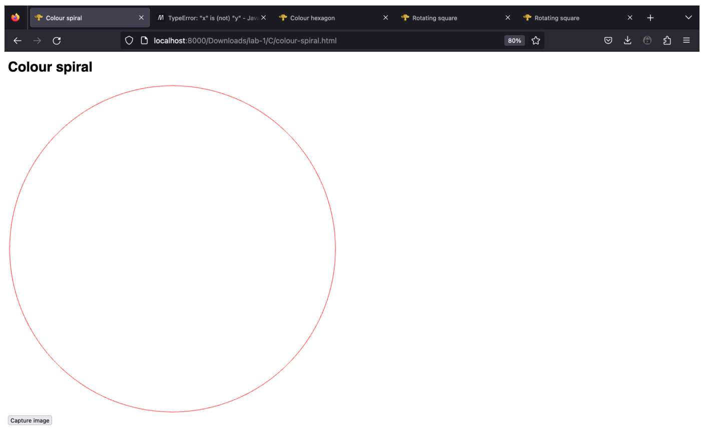
As you can see here, I have taken the hexagon I had created from the previous part (B1) and have added a colour into each vertex by altering the colour arrays which are in the format RGBA (for example the yellow was [1.0, 1.0, 0.0, 1.0]). I had also altered the drawElements function and changed gl.LINE_STRIP to gl.TRIANGLE_FAN.

C1:



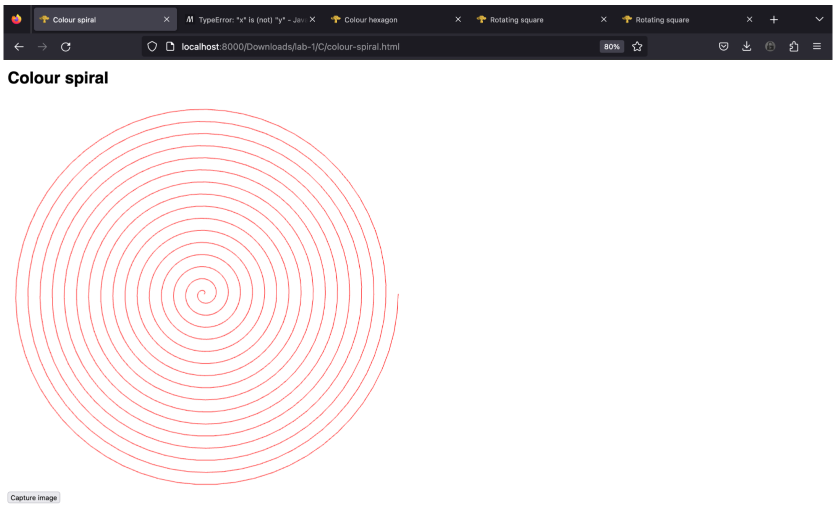
As you can see here, I have taken my previous code from part B and adjusted a few things to create an octagon. I had firstly changed num_verticies at top of the code to 8 to ensure the shape has 8 vertices, changed the colour to red by changing the red value to 1 and changed the side length to 0.99.

C2:



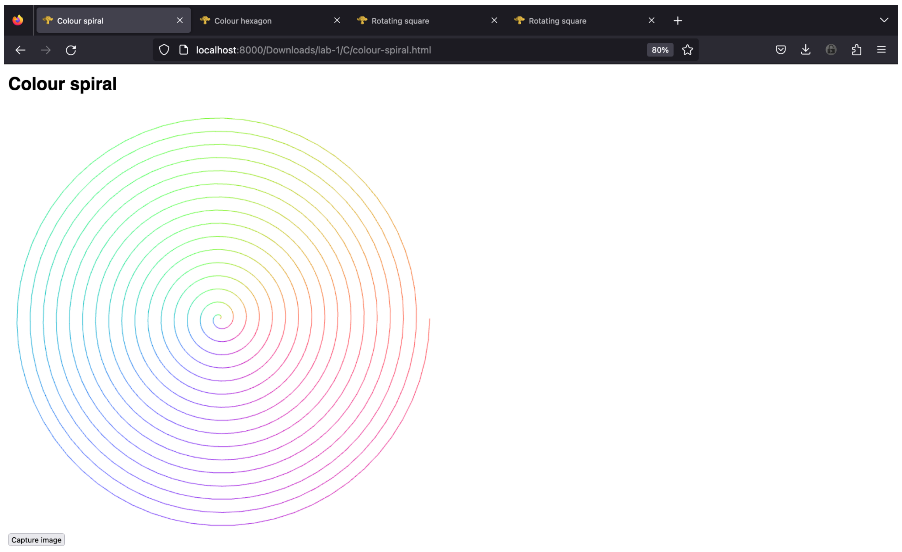
As you can see here I have produced a circle here. This was done by simply altering the value of num_verticies from 8 to 100. However whilst doing this I had found that the circle did not appear and realised that I had to also include the colour array in the for loop so that each vertex could be coloured the right colour (red)

C3:



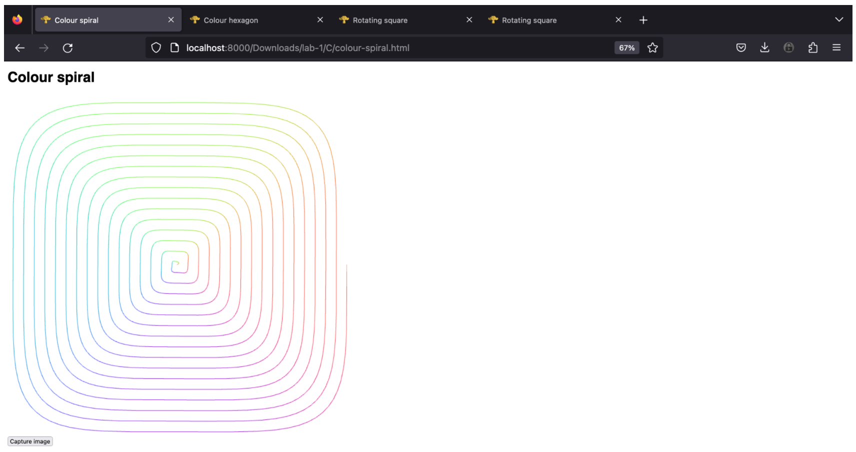
As you can see here I have produced a spiral with 16 cycles. I've done this by adding a scale parameter which would always be a value between 0 and 1 and increases by 1/num_vertices (in this case it would be 1/1000) for every loop, this value is multiplied by the x and y coordinates.I had also added a num_cycles variable which is multiplied by the degrees to add for each loop so that the amount of spirals can be easily changed.

C4:



As you can see here this is a spiral which changes colour as it goes through a cycle, each cycle having a certain section for each colour/colour combination. This was done by simply altering the colour array in the for loop to calling and storing the returned value of rgba_wheel() every loop with the argument being the currentDegrees.

C5:



As you can see here this is a squiral which changes colour as it goes through a cycle, each cycle having a certain section for each colour/colour combination. This was done by making use of the squircle() function I had written which took in the degrees/radians and a shapeParameter to bring back the new coordinates. The image shown used a shapeParameter of 5, but when other shape parameters are used the shape changes. At a shapeParameter of 1 it becomes a diamond-spiral, at 0.75 the sides of the diamond seem to curve inwards between the vertices, at 10 it resembles the shape shown above but the edges are more resemblant of a square (has less of a curve) and at 100 it becomes even more square like but has rounded edges.