PMC Week 7

Introduction to GLSL Continued
More Fragment Shaders

Tasks From Memory

- How easy was it for you to complete the code from memory?
- What made it difficult?
- I encourage you to practice this at home

Shading the circle

- Did anyone manage to shade the circles?
- Let’s look at some solutions
- What I actually asked you to do is to alter the colour based on the y coordinate

Shading the circle

- basic solution should produce this image

gl_FragColor =(vec4(pos.y * 0.01,pos.y * 0.01,pos.y * 0.01,pos.y * 0.01));

https://live.codecircle.com/d/ouZcShD8W9QeySyyz

Other approaches

- Using the distance from the centre gives the illusion on depth

- This is because we image the circle as a sphere

GLSL 2D Rotation

- In the example, I define a 2D rotation matrix and multiply the 2D coordinates by these values
- I ask you to work out how to rotate around the centre.
- Did anyone do this?

GLSL 2D Rotation

- What we need to do is subtract the half-width and half-height from the current coordinate.
- This shifts everything up and across.
- Answer here:
- https://live.codecircle.com/d/WHNn7zxizqDE9KXmy

GLSL 2D Rotation

- What does this tell us about the GLSL coordinate space?
- Where is 0. 0.?
- https://live.codecircle.com/d/iK4osaFJAgNwjw8yS

GLSL 2D Polar Coordinates

- https://live.codecircle.com/d/LdQdc79FHpWHMtSWB

- We had a look at this briefly last week
- Who looked at it in the lab?
- Do we understand it?

GLSL 2D Polar Coordinates

- First we get the distance from the current coordinate to the centre the screen:

```
vec2 normCoord = gl_FragCoord.xy/resolution.xy;

vec2 pos = vec2(0.5)-normCoord;
```

GLSL 2D Polar Coordinates

- Then we determine the current length as a single unit.
- This is our radius
- It’s a cartesian to polar conversion

```
float radius = length(pos)*2.0;
```

GLSL 2D Polar Coordinates

- Then we calculate the angle
- Remember - this is the inverse tangent of y/x

atan(pos.y, pos.x)

- What happens if you swap x and y?
- Does this matter?

Polar Coordinates

- The number of petals is equal to the integer multiplier of the cosine of the angle

```
float petals = cos(angle*1.); // change this value
```

- What happens if you use sin instead of cos?
- Does this matter?

Polar Coordinates

- The number of petals is equal to the integer multiplier of the cosine of the angle

```
float petals = cos(angle*1.); // change this value
```

- What happens if you use sin instead of cos?
- Does this matter?

Polar Coordinates

```
vec3 colour = vec3(step(petals,radius) );
```

In pairs, try to come up with an explanation of how this is generating either black or white.

Polar Coordinates

- Let’s look at the ‘developed’ example:
- This is closer to the whitney formula we looked at

https://live.codecircle.com/d/uKkNQWDeaBCx4yDyM

Polar Coordinates

- There are only three major differences
- Two of those differences are that we're using the mouse, and we're using smoothstep
    - Not that big a deal
- The other, bigger thing is that we are getting the sin of the length in order to calculate the radius.

```
float radius = sin(length(pos * mouse.y) * 50.);
```

That's the only difference…but it looks very different

Drawing A Square

- So we've done circles and polar forms, but we haven't done lines and squares yet.
- Amusingly, we are going to start with a square, and then do lines.
- Then we will do functions

Drawing A Square

- In order to draw a square, you just need to know if the distance from the centre X to the edge is greater than a certain amount
- You need to know the same for Y also.
- If this is the case, you have a square. Simple.

Drawing A Square

```
if (length(pos.x) < size && length(pos.y) < size) {

colour = 1.;

}
```

https://live.codecircle.com/d/rFpN3nAyJFswWmoFP

Drawing A Square

How could we draw lots of squares?

In the circles example, we used modulo, which was cool, but a bit confusing.

How else could we draw lots of things?

Using Functions

We could define functions.

This way, you could pass a position into a function and get it to produce colours.

You could then the results of these functions together - or subtract them - in order to create a scene.

Drawing Lines with a function

- Rather than doing squares, let's define a function that allows us to draw lines.
- We can then use this GLSL function to plot *functions* in the traditional way
- Let's have a look.

Drawing Lines with a function

- Rather than doing squares, let's define a function that allows us to draw lines.
- We can then use this GLSL function to plot *functions* in the traditional way
- Let's have a look.

https://live.codecircle.com/d/fupYhMkruJuihpQma

Drawing Lines with a function

```
float line(vec2 normalised_Coordinate, float funct) {

// This if statement does the same job as the step function
if (normalised_Coordinate.y - funct < 0.1 &&
normalised_Coordinate.y - funct > 0.0)

                return normalised_Coordinate.y;

}
```

Drawing Lines with a function

Or we can use the step function

```
return step(funct,normalised_Coordinate.y)-
step(funct,normalised_Coordinate.y-0.1);
```

Drawing Lines with a function

- Our example plots a line across the x dimension, with increasing y values
- But we can use it to do much more than that
- We can plot full functions, and express them graphically

Drawing Lines with a function

- https://live.codecircle.com/d/sPC9w7mFD5GFfPrSS

Management Code

- Let’s take a brief look at the management code
- It’s in JavaScript, but is basically the same set of functions.
- https://live.codecircle.com/d/KddxGTEhyC9FENyTJ