Set-up

You received an executable, which attemps to render something in OpenGL 3.3 / GLSL 330. The executable tries to load two files:

- shaderplay.vert &
- shaderplay.frag,

representing the vertex and the fragment shader, respectively. Just use the two files that are already there, as those are where the executable expects them to be.

Pressing 'R' will reload the files, and attempt to compile and link the shaders. If they are invalid, the previous shaders will be kept, otherwise the new shaders will be used for drawing.

Pressing 'W' will toggle on and of wireframe mode.

The executable will specify several input variables for you:

Per-vertex attributes:

- position:

  The x,y,z vertex position. The vertices represent a set of concentric circles, packed so tightly that they appear as a continuous region. The x and y components are within the range [(-1, -1), (1, 1)], z is zero. There are 90 circles in total.

- texcoord:

  u,v texture coordinates. They all lie within [(0, 0), (1, 1)].

- circleId:

  The id of the circle this vertex belongs to, with 0 being the innermost circle.

The fragment shader is expected to output a RGBA color, called fragColor.

Uniform variables:

• matScene :	the scene (model) matrix, a 4x4 matrix.
• matView :	the view matrix, 4x4.
• matProjection :	the projection matrix, 4x4.
• matRotation :	a 2x2 matrix, contains counterclockwise a rotation in 2D.
• time :	a time factor, to be used for animation.

- colorTexture : a RGB 2D texture.
- alphaTexture : an Alpha 1D texture.

The executable also specifies the following states:

- Alpha blending is enabled, with blend functions GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA.
- The clear color is black, the clear depth is 1.0.
- Depth Testing is enabled and set to GL_LESS

Your task

Code some simple shaders, get a feeling for GLSL, use different datatypes. The following tasks should be carried out sequentially. Try to always create a working shader version, and don't change too much in between reloading - debugging a shader can be a very frustrating experience, and if you change a lot of code at once, you might be totally lost.

Editing shaders can be either done in notepad++, there's a glsl plugin available here.

Alternatively, you can use VC++ 10 with this addin.

If you need help with GLSL functions, check either the cheat sheet (scroll down) or the online manual.

If you don't finish the tasks during the lesson, you first assignment is to finish them at home. Deadline: 23/10/12.

A) Basic Vertex Attributes

1) Write a pair of shaders that just draw the vertices. Remember, they are all within the range of [(-1, -1), (1, 1)]. If you succeed, you should see one solid circle on your screen. Give your fragments any color you like for the time being.

2) Now, calculate the distance between the vertex position and the origin. Create a new vertex attribute in your vertex shader, store the distance and use it as color in your fragment shader.

3) In the fragment shader, discard every 4th colum of pixels with the help built-in pixel position gl_FragCoord, and every 3rd circle with the help of the circleId attribute. Since the

circleId is constant for a primitive anyway, switch off the per-fragment interpolation by using the qualifier 'flat' on the attribute in both the fragment and the vertex shader.

B) Texturing

1) Texture the fragments based on the RGB texture with the help of the texcoord attribute. Why is the screen suddenly black (if you did it right)? Solve that problem, so you can see the textured circle again. Modulate the color with the previously calculated distance.

2) For front-facing primitives, change the output color from RGB to BGR with the built-in attribute gl_FrontFacing.

3) Use the circleId as lookup into the alpha texture. Since this is not a real texture coordinate in [0, 1], use texelFetch to get the exact value instead (use mipmap level 0). Use the alpha value to give your fragments transparency.

4) the 2x2 rotation matrix specifies a counter-clockwise rotation around the origin. Let the texture rotate. (This is actually a bit trickier than it sounds. What happens to the texcoords when they rotate around the origin? Get rid of this effect.)

C) Transformation Matrices

1) The scene matrix contains an animation (rotation around the Y-Axis). Use it to let the circles rotate.

2) Use the view- and projection matrices to get a perspective effect.

D) Vertex Displacement

1) Create a displacement effect. Start by calculating the angle between the x-Axis and the xy-Position. In case you've forgotten how to calculate the angle, read this explanation on how to get it right. (note that GLSL's atan function returns radians, also, you will definitely need to correct the angle based on the quadrant the vertex is in. In order to keep your shader easily readable, create a function outside of main that does this correction for you).

Once you have a continous angle for each vertex, calculate the displacement value based on the cosine of the time and the angle, i.e. something like this:

```
displacement = amplitude * cos( frequency_factor * angle + time );
```

and add it to the vertex's z-component. Play around a bit with the amplitude and frequency factors a bit until you get an effect you like, as they are left open to you. (Hint: you most definitly want an amplitude < 1).

In order to make the displacement look nicer, attune it with the squared distance: there should be almost no displacement next to the origin, and maximum displacement in the outermost circle.

Make sure the displacement works correctly when the scene, view and projection matrices are applied!

2) Add a second displacement effect, again in the z-Direction. This time, it should not be based on the angle - instead, create a radial wave effect, like a droplet falling into water.