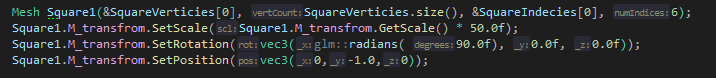
# Shadow mapping

First we will need a floor to project our shadow on to.



This will crate me a floor plane and rotate it appropriately. Your planes transform values might be different, as long as it looks like this:  
A picture containing text

Description automatically generated

It doesn’t matter (that’s a horizontal plane, honest). I’m not drawing the cube, only so I can show the floor off, you will need the cube, so leave it rendering as normal.

A picture containing text, brick

Description automatically generated

Now we can begin on the shadow specific code.

In the mane function, just above the while loop, add the following.

Text

Description automatically generated

This code creates a new frame buffer, which is a fancy way of saying “space, on the gpu, where we can store a rendered image”.

We only want one of them, the dipthMapFBO is our reference to it and then we activate it so we can use it.

Now we need to create a new texture on the gpu.

A screenshot of a computer

Description automatically generated with medium confidence

You have seen most of this before, so I’ll highlight the important bits and skim the rest.

Text

Description automatically generated

This is the location of the shadow map in memory and the size we want the map to be, in pixels.

A screenshot of a computer

Description automatically generated with medium confidence

This tells the gpu to create a new texture and provides it with some parameter details, like how we want it to wrap, etc. note that we use the shadowMapID to reference the new texture and we use the ShadowWidth & ShadowHeight values in glTexImage2D so that GL knows how big to make the texture.

Also note that we are using GL\_DEPTH\_COMPONENT for the internal format and format.



This code forces the boarder of our new image to be white. In depth map, white is the furthest value.

This helps reduce artifacts later, around the edge of the shadow map.



This code attaches the new shadowMap texture to the current active frame buffer, IE the depthMapFBO we made earlier.

We then set Draw and Read buffers to GL\_NONE as we don’t want any draw (colour) or read (data coming out) from this frame buffer, we just want the depth.

Text

Description automatically generated

We now check for errors and, assuming everything is ok, we bind to framebuffer 0 (the default buffer) and bind to texture 0 so that we aren’t using the shadowMap anymore.

Test the code, at this point, nothing should have changed and the program should run as normal.

# The First Pass

Up until now, we have been rendering our scene, presaging the render, and looping to doit all again. Now we are going to introduce another render step, before we draw our scene, the shadow pass!

Everything we have right now is our drawing code (+some event handling), at the top of my draw code, I’m going to add the following above glClear():

Graphical user interface, text

Description automatically generated

Previously, we had set the view port data outside of the while loop, as part of the initial setup process. We will need to mess around with the viewport, as part of the shadow pass, so we will need to set it back to our original view port settings, after the shadow pass (which we don’t have, yet!) and before the render/draw pass.

Again, if you run the program, nothing should have changed.

Now we can begin work on the shadow pass.

Above glClearColor but below the while, add the following:

Text

Description automatically generated

We reset our viewport to match the shadow map width & height, we need the view port to match the texture size so we can render into all the texture.

Now we need to generate an MVP por the light. We will need this so we can transform our geometry and later fragments in to light space.

Text

Description automatically generated

Again, you have seen this before so I’m going to skim it.

We create an autographic projection, rather than a perspective projection, as we are drawing shadows for a directional light. If we want a point light, we will need to render our shadows to a cube map and I don’t want to touch this topic now.

We create a lightView so that the light is looking at our world centre and use all of this to build the light space matrix.

## Shadow Shader

New we need to work on drawing our scene from the lights point of view.

To do this we will need to draw our objects BUT! We will not need to bother with lights, textures or colouring the fragments. We are only interested in drawing our models for their depth information.

So we will make some new, very simple, shaders

Text

Description automatically generated

And they will contain the following code:

Vertex shader:

#version 430 core

in vec3 position;

uniform mat4 model;

uniform mat4 lightSpaceMatrix;

void main()

{

gl\_Position = lightSpaceMatrix \* model \* vec4(position, 1.0f);

}

Fragment Shader:

#version 430 core

void main()

{

}

We don’t need the fragment shader to do anything, so just leave it blank. The Vertex shader need to transform the current vertex into camera space, to do this we need to apply the models transform (any movements, rotations and scales) and then multiply that by the light space matrix to shift the verts in to light space.



Create a new shader object. We wont need cam in this shader so its probably best to overload the constructor so that we don’t need to pass it in. For now, this will do. BUT its worth cleaning up the code later to make it more readable and logical.

We WILL, however, need a new update method. Update handles updating the uniforms in the shader. Since we have different uniforms in the depthShader we will need a different update method.

A screenshot of a computer

Description automatically generated

Something like this should do.

Now we just need to use our shader and draw our cube.

Graphical user interface, text

Description automatically generated

3 things to note here

1. Before we draw our cube, we need to tell GL to cull the front faces of our object, this will help prevent peter panning
2. We MUSH set it back to culling back faces after the draw or our render pass will cull the wrong faces.
3. Don’t forget to bind to frame buffer 0 so the render pass will draw into the correct buffer.

Now we need to get the shadow map over into our basic shader.

Text

Description automatically generated

In the frag shader, add a new smpler2D for the shadow map texture.

Just to test everything, change the frag\_color to use the shadow map only.

A screenshot of a computer

Description automatically generated with medium confidence

And now we need to send the actual shadow map to the shader.

Text

Description automatically generated

Now if you run the program you should see this:

A picture containing graphical user interface

Description automatically generated

Or something like it.

Ok now you can remove the frag\_color line you just added, and we can more on.

## Applying the Shadow map

In the fragment shader we will need to do a few things, first define the following function above the main function:

Text

Description automatically generated

Then update frag\_color to look like this:

Text

Description automatically generated

Now we can fill in the CalculateShadowValue function.

Text

Description automatically generated

So we start off by defining out frag coordinate (in terms of light space (light is world 0,0,0) ) and dividing its location by w. this creates a perspective like shift so that the closer points cover a larger part of the “screen” … or depth map.

We multiply the projection coordinates by 0.5 then add 0.5 so that the result in in 0 to 1 space. This is needed so we can properly look up the texture coordinate in the depth map.

Then we look up the texture coordinate in the depth map.

The current fragment depth is the depth of this fragment in light space (i.e. its distance from the camera).

The bias is a value that will be used to prevent shadow acne.

Next we set shadow to 0 and find the size of this texel.

we then loop over a square of 9 texels (3 above, 3 below, 2 on ether side and this text) adding there values to the shadow value.

We use current-bias to “indent” the shadow map value in to the surface of the fragment to prevent shadow acne.

Out side of the loops, we divide the shadow by 9 (the 9 texels we added together) to get the average shadow value.

We then check to see if the projection coordinate.z is > 1. If it is, then we set shadow to 0 this is to catch any issues with fragments that are currently no in the shadow map.

Finally, we return shadow.

NOTE that we are missing a definition for FragPosLightSpace

Add the following to the in variables, at the top of the shader file:



Since it is an in variable, we will need to pass it in from the vertex stage, so open the vert shader.

Define the following, with the rest of the global in variables



In the vertex main, add the following (you should already have FragPos and FragNormal, I’m just covering them again to show they are also needed for shadows).

Text

Description automatically generated

READ TO NEXT PAGE!!!!

Your code will show errors, “FragPosLightSpace & VsNormal are not defined”. So define them in the global out variables.

Text

Description automatically generated

Now we need to update the shader update function to take a light space matrix



Don’t forget to pass in the matrix for our light (in the main function).



In the shader update, find the “lightSpaceMatrix” in the shader program and pass it the LightspaceMatrix.  


WARNING!!

This tutorial has not been user tested. If you find an error, congratulations! You are the first to get this far. Please let Nick know and he’ll help you figure out what has gone wrong.

Run Your Code!

With a little luck you should see some shadows!

A picture containing text, dark

Description automatically generated