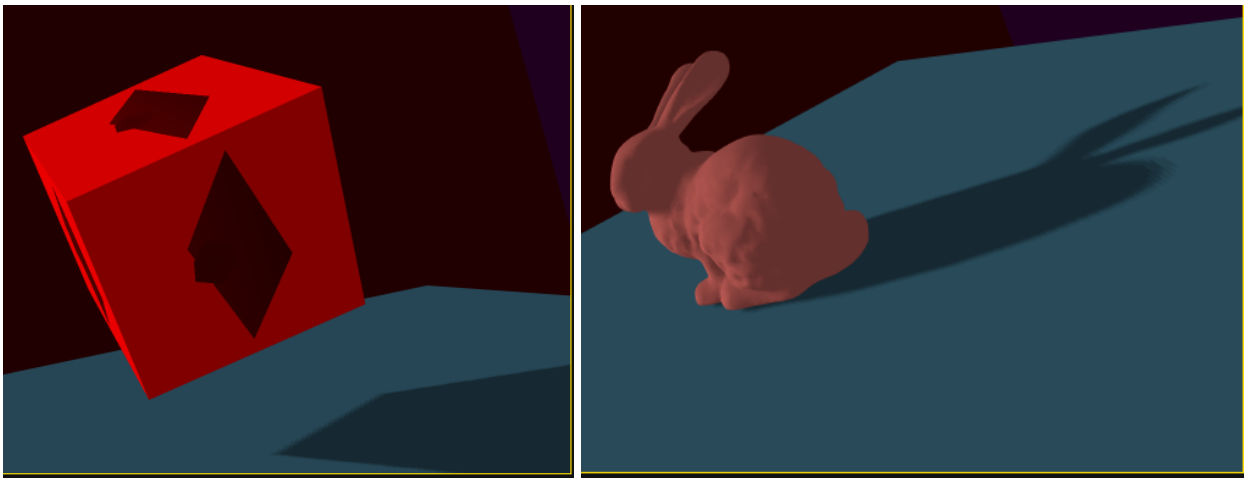
Graphics programming - home exam 1

Description of solution

This report describes what I believe to be a functioning implementation of a shadow map algorithm and how I got there.



A cube rendering the depth texture as its own texture and the stanford bunny casting a shadow

Shaders

Using depth textures, the light textures are pretty simple and we can just pass the depth value along. As per the Khronos spec, we extract the value from the RED channel.

```
#version 300 es
precision highp float;
out float fragDepth;

void main (void) {
  fragDepth = gl_FragCoord.r;
}
```

With depth textures, just pass the r-value along

The camera/model vertex shader needs the shadow bias matrix to convert coordinates from the [-1, 1] space to [0, 1]. The matrix halves the values, giving us [-0.5, 0.5], and then translates them to [0, 1].

```
#version 300 es
// {ins, outs, uniforms ...}
```

```
const mat4 shadowBiasMatrix = mat4(
  0.5, 0.0, 0.0, 0.0,
  0.0, 0.5, 0.0, 0.0,
  0.0, 0.0, 0.5, 0.0,
  0.5, 0.5, 0.5, 1.0
);

void main(void) {
  // using the shadow bias matrix to get the right shadow coordinates
  vShadowPosition = shadowBiasMatrix * uLightProjectionMatrix *
uLightModelViewMatrix * aVertexPosition;

 // other transformations, lighting, etc. Mostly from the MDN tutorial
}
```

… and then calculates the shadow based on the light value of its nearest neighbors. To avoid areas in shadows to become completely blackened, we add the ambient light values to the visibility values. The depth of the shadow coordinate is reduced a tiny amount (on the order of 0.001) to avoid shadow acne.

```
#version 300 es
// {...}
void main(void) {
  // ...variables

  // percentage closer filtering
  for (int x = -1; x <= 1; x++) {
    for (int y = -1; y <= 1; y++) {
      if (shadowCoord.z < texture(uDepthTexture, shadowCoord.xy + vec2(x, y)
* texelSize).r) {
        visibility += 1.0;
      }
    }
  }
  visibility /= 9.0;
  //add ambience to color
  fragColor = vAmbientColor + visibility * vec4(vColor.rgb * vLighting,
vColor.a);
}
```

GL general

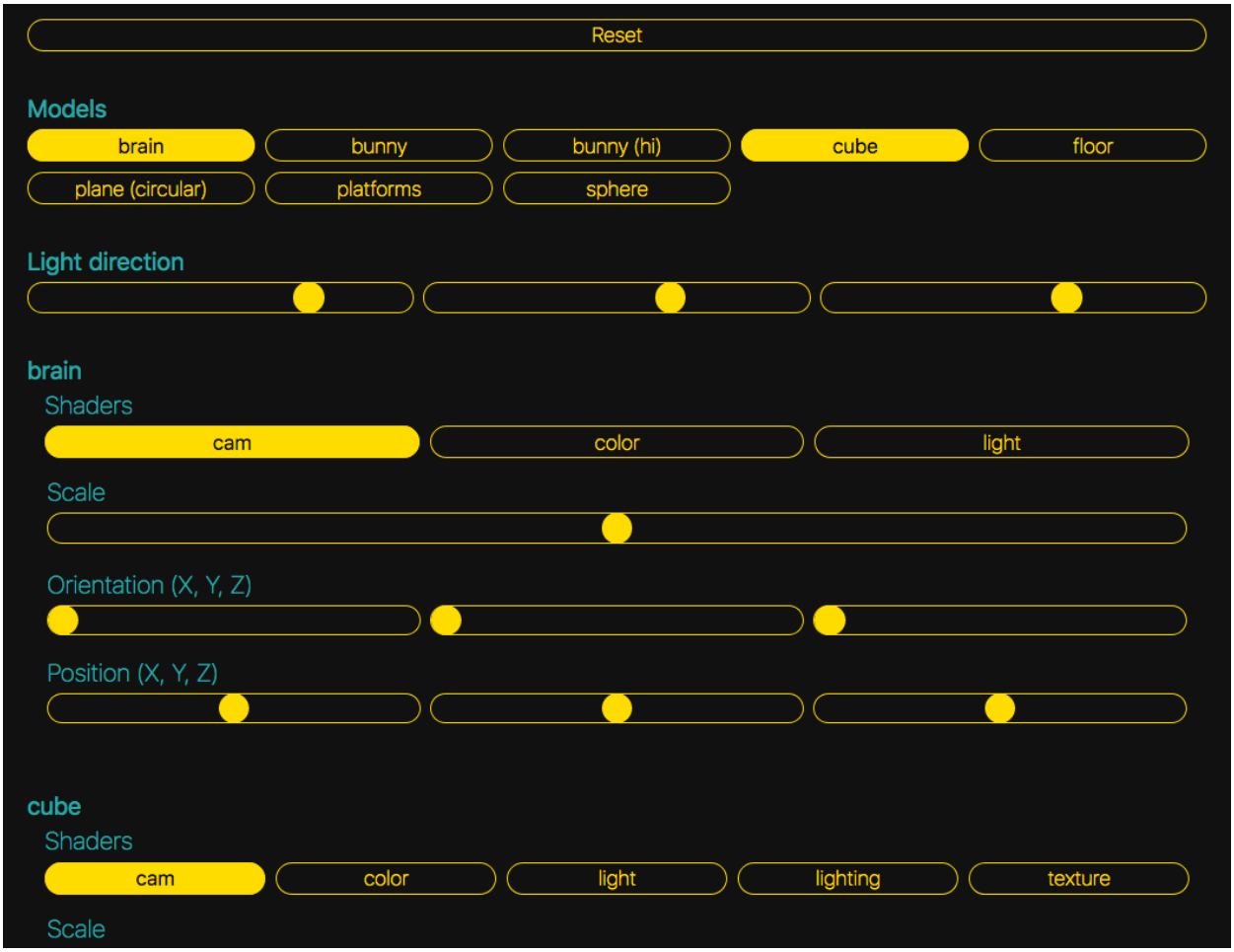
The render loop is split into three stages:

1. Map the model data (positions, faces, etc) for the shadow casting objects into model and normal matrices
2. Do the first render pass for depths, using the light's projection matrix. Render to the allocated frame buffer object.

3. Do a second render (map and include the remaining models that don't affect shadows, such as the ‘light source’, and the ‘room’) using the depth texture's values to calculate what parts of the scene should be in shadow and and what parts should not.

The render loop culls front facing triangles during the shadow map rendering and back facing triangles during the actual rendering to help combat shadow acne even further. At the moment, though, the effect is negligible.

The light projection matrix uses a simple `lookAt` function to always face in towards the center of the world and can easily be controlled through the UI.



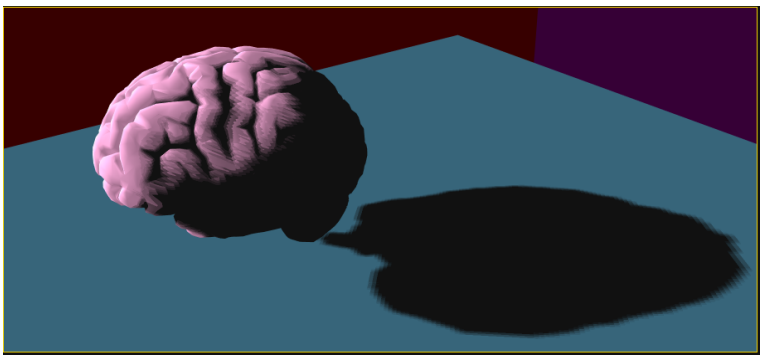
The main UI controls, including model selection, light sliders, and individual model controls (different models may have different shader programs available, for instance)

Known limitations and where to go next

Given more time, I would have liked to look more into shading and shadow mapping. In particular, it would have been interesting to explore other options to 'percentage closer filtering' or how that could be improved. I would also have liked to look more into specular shading and more advanced shader scripting.

Other comments

I personally really like the effect of the shadow mapping when the parts that are in shadow are completely obscured, but I realize that it's not always appropriate. However, because it looks really good and it was part of the process, I think it's worth including this image from before the ambient light was added back into the shader:



Technical challenges met during implementation

The assignment states to use the data type `gl.UNSIGNED_INT_24_8` for the texture. However, despite spending a lot of time looking for resources I was unable to find any that showed how to use this. The data type was mentioned several times, including in MDN's WebGL docs, on WebGLFundamentals, and in the spec but neither those nor any other resources I came across have any examples on how to use it.

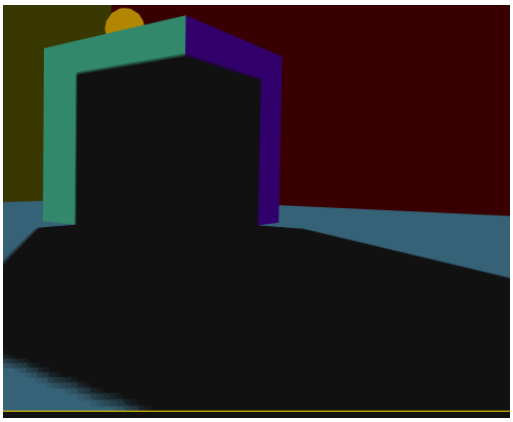
In fact, the single greatest issue I faced was finding out how to use depth textures and how to reference them in the shaders. It turned out to be pretty simple in the end, but it was quite the journey.

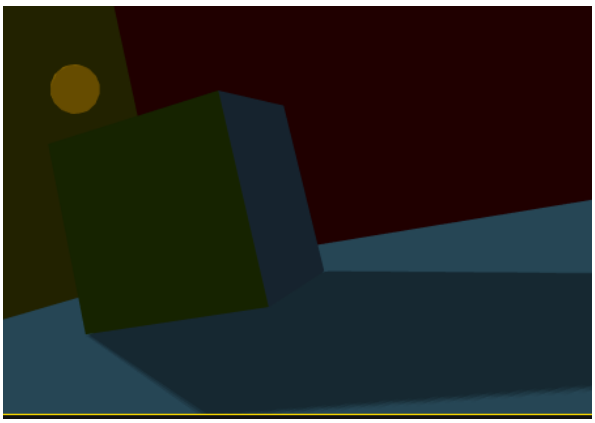
Once I had managed to render to the texture and could confirm that it looked correct by rendering to the canvas instead of to the texture, the next thing was displaying the shadows correctly. A bit of fiddling around revealed that I was missing a multiplication with one of the light matrices, which made the shadows follow the camera's position.

I struggled to find any good information on render texture dimensions, but it seems the standard is to make it square. The larger you make it, the more information it can hold and the smoother your shadows will look due to the increased resolution that comes with the larger size. Naturally, this is a tradeoff between performance for visual quality.

I had an issue with the shadow acne threshold used in my fragment shader. At first it was much too coarse which caused the shadows to be severely reduced creating strange artefacts like the thick outline at the edges of the cube being lit when they were clearly in

shadow. By making the acne threshold much smaller I was able to greatly reduce the effect, and in the end, adding the ambient light seemed to take care of the issue, as can be seen in the 'after' photo on the right.





Known bugs and probable causes

At the time of writing, I am not aware of any bugs.

Testing methods utilized

While I am very much an avid proponent of testing, there was no formal or rigorous testing used as part of this project. This is due in part to the fact that testing graphics is inherently difficult due to the visual aspect, and due in part to the fact that WebGL is a very stateful system, where setting up test cases and assertions can be very difficult.

As such, and also because I do not yet possess the requisite knowledge of the system that is WebGL, there were no tests written for this project, although I would like to explore how to write tests for a system like this.

Sources

Throughout the assignment I have had to lean heavily on the few sources I could find, and as such I would be remiss not to mention the most valuable ones.

- WebGL2 Fundamentals article for rendering to textures from whence I got the use of depth textures
- opengl-tutorial article on shadow mapping, which informed my use of the shadow bias matrix, shadow acne removal, and front and back face culling for rendering
- this CFN shadow mapping tutorial, which influenced the use of the 'percentage closer filtering' algorithm for smoothing the shadows, as well as some more basic information around the mapping process itself. Also uses the shadow bias matrix.