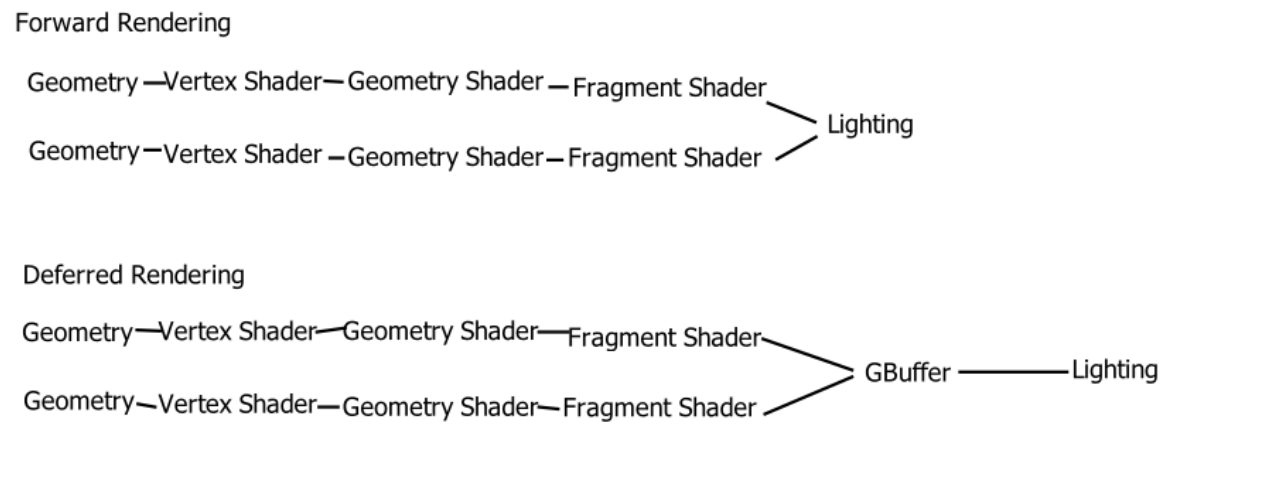
Task 1

Time: 40Min

Forward rendering involves doing multiple lighting passes (depending on the number of lights affecting it )on each object after rendering to give it lighting.

Deferred Rendering applies shading to the objects all at once making the lighting happen in one pass, giving a better performance.



In the diagram, the key difference between deferred and forward is that in forward rendering every object is rendered one at a time to render the final image, while in deferred rendering every object is put through the pipeline to the Gbuffer, which would then contain properties like position, normals, spec instencity, and colors, where the lights can now be applied all at once.

In pseudocode forward rendering is

foreach object

foreach light

render object with light

while deferred rendering would be

foreach object

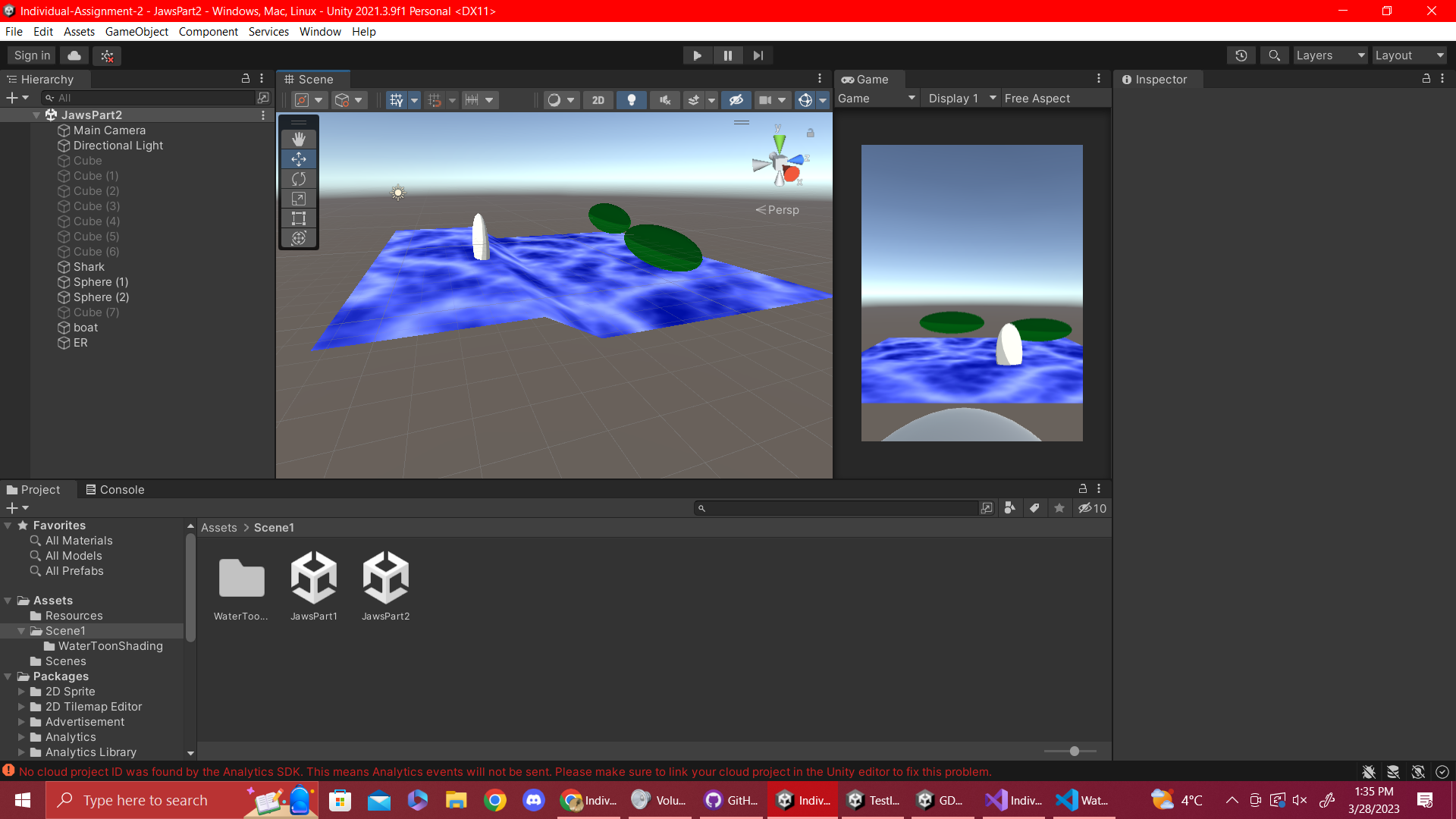
store properties

foreach light

apply shader to properties.

Task 2:

Time 1:30



The toon shader was applied on the shark and island and a wave shader on the water which modifies the normals of the plane.

The main modification I made was a brightness variable that modifies the color of the toon ramp color to make it lighter or darker. I mainly wanted to make the shark more brighter as it is the main object in the picture.

For the water, since the water’s height is based on a sine wave, I added an if statement on how the water’s height would work, if the water’s height would normally be above 0 it would go to the max height and if not it would go the the min-height. I also implemented the toon ramp code inside the water shader.

Task 3:

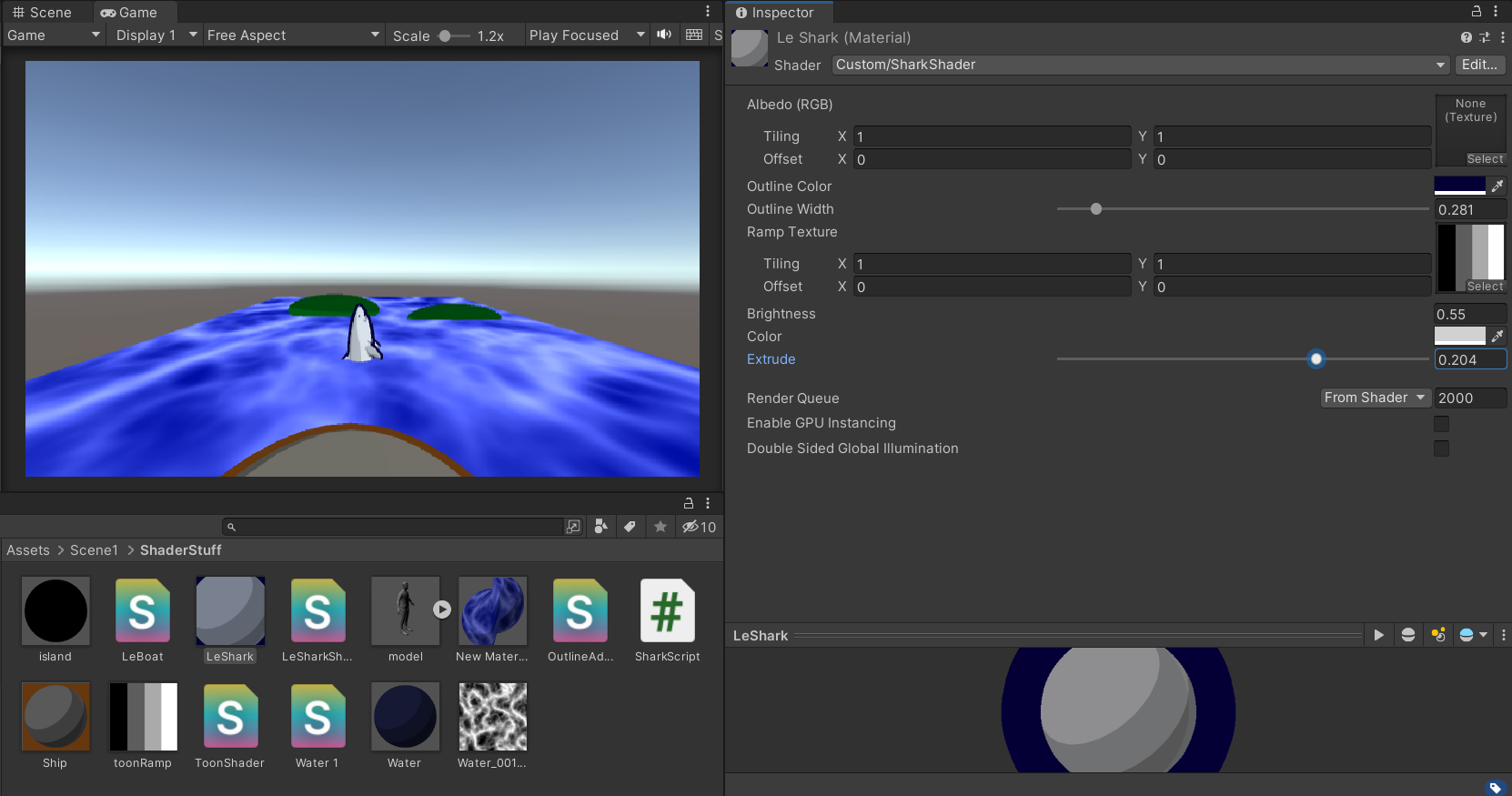
Time: 20 Min

The code uses progressive downsampling to convert an original image from a higher resolution to fit the screen’s resolution without the loss of any pixels. The iteration variable controls the amount of times it can downsample, if the variable is too large for the resolution of the screen it breaks the loop giving the maximum amount possible for your screen resolution, assuming your screen height isn’t bigger than its width.

This can be useful for players who use higher resolutions on their games than what their monitor have like in my case with Cyberpunk 2077, a 2560x1600 resolution on a 1920x1080 laptop screen.

Task 4:

Time: 20 min



For outlines, it was implemented on the shark and the boat, while vert extrude was implement on the shark since it could be used as a way to show the shark getting fatter by having a script modify the extrude variable.

The main modification is in the shark shader where I had both the extrude and outline effect on the shark. The main problem I found was that it was possible for the extrusion to become bigger than the outline effect so I had the outline’s size also be affected by the extrusion strength, unless the extrusion strength is at a negative as to avoid having negative outlines.

Task 5:

Time: 20 Min

fixed diff = max(0,dot(s.Normal,lightDir));

half4 c;

c.rgb= s.Albedo \* \_LightColor0.rgb \* (diff \* atten \* 0.5);

Implements standard diffuse lighting on the material.

c.rgb += \_ShadowColor.xyz \* (1.0 - atten);

c.a = s.Alpha;

return c;

Implements standard diffuse lighting on the material.

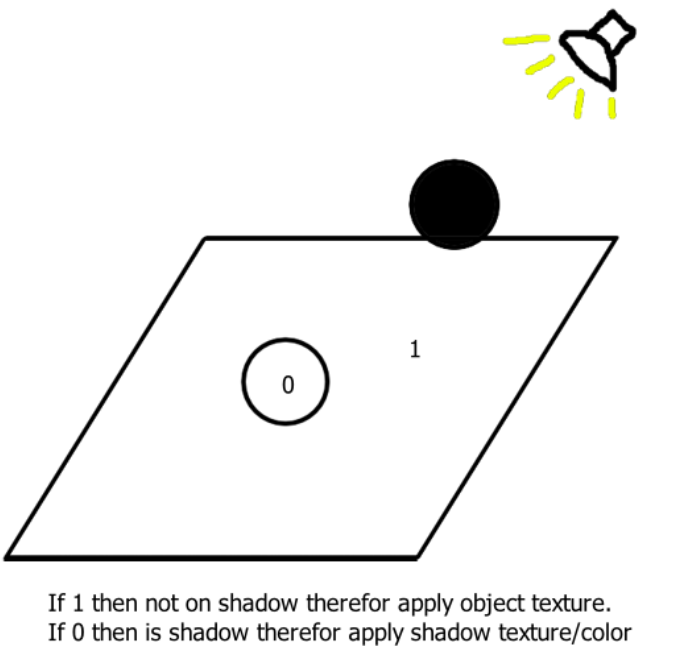
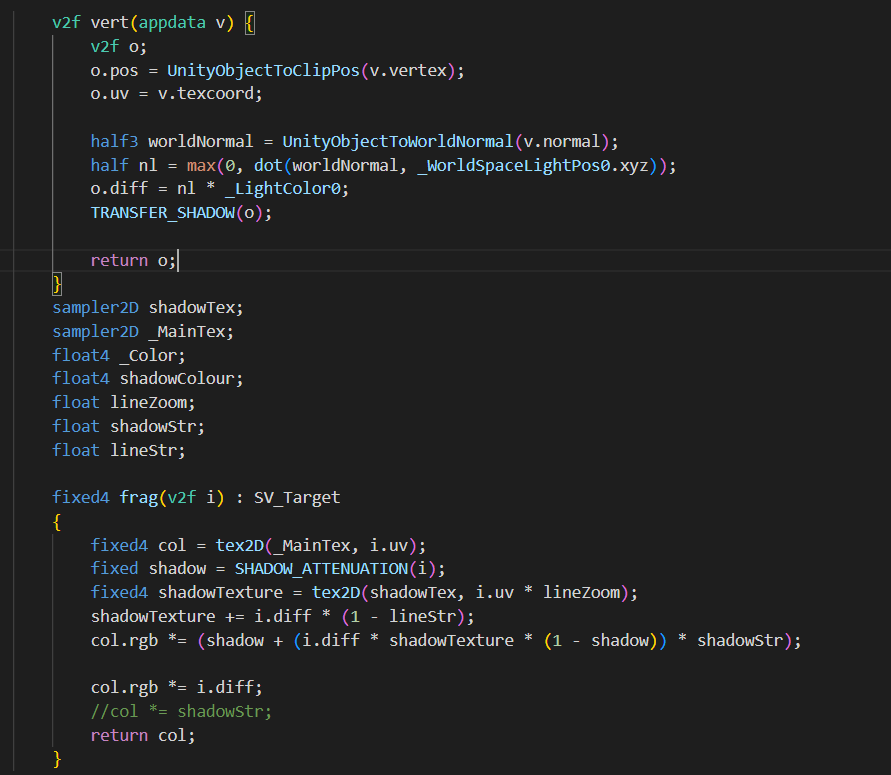
Since atten (light attenuation) gives the value based on how much light is spread on the object (basically the strength of the light), the inverse of atten gives the strength of the shadow, this means that the shadow color is applied to the material based on how the strength of the shadow.

Overall the code first applies the colour of the object based on diffuse lighting and then applies the shadow colour based on the inverse of the light attenuation. This would result to possibly making no dark spots in the object as the would be dark spots are being covered by ShadowColor.

I can see this being used recreate certain artstyles, similar to the artist for JoJo’s Bizzare Adventure, where shadows were changed to have odd colours to enhance the bizarre feel of the art, like purple shadows over a cloudy day, I think they might have done this for the new 3D JoJo’s fighting game.

Task 6: Shadow Texture Shader

Time: 10 Min



For the shader, first find dot product between the normals of the world and the position of the world light source, if its lower than 0 then its a shadow (Shadow Attenuation). Then if the shadow attenuation is 1 apply the object’s texture else if its a 0 apply the shadow’s texture. If you want to have shadow texture’s colour slightly affected by the object’s colour multiply the shadow’s colour with the object and a strength variable to have control on how strong shadows are.

A good application of this shader is on cartoon and anime style games, Hi-Fi rush uses shadow textures by applying a line texture over their shadows to give it a anime/manga feel to the game. Same can be done if DC or Marvel wants to make a superhero comic style game that wants to capture the feeling of reading a 3D interactive comic by having a repeating circle texture on dark areas of the screen.

Uploading to Github:

20Min