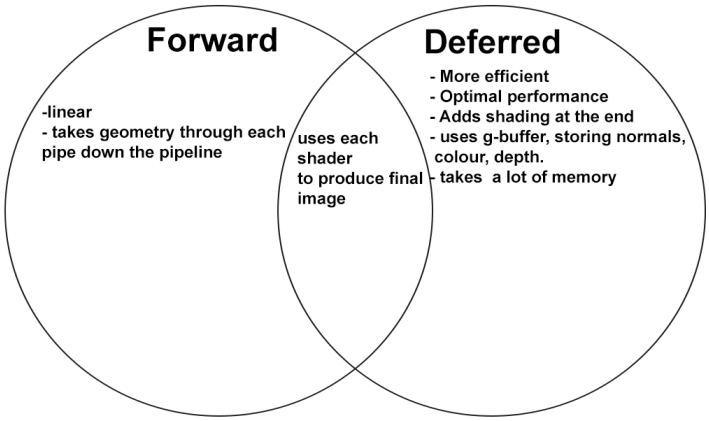
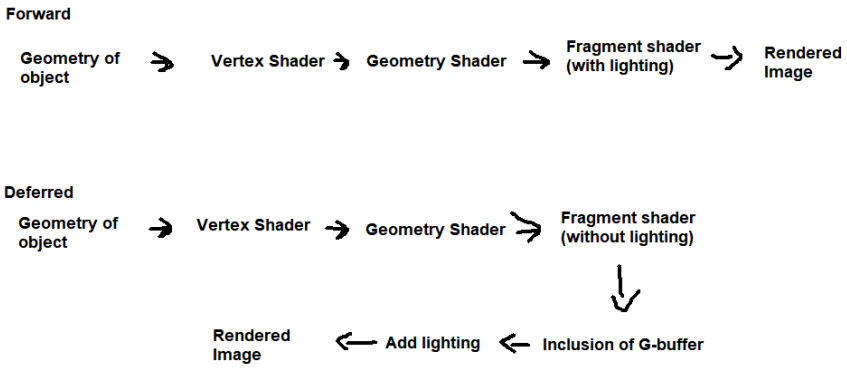
Create Github: **4 minutes**

Forward & Deferred Rendering: **20 minutes**

**Forward rendering** is a linear way to pass geometry down the pipeline to produce a final image of the object with light added to it from the scene.

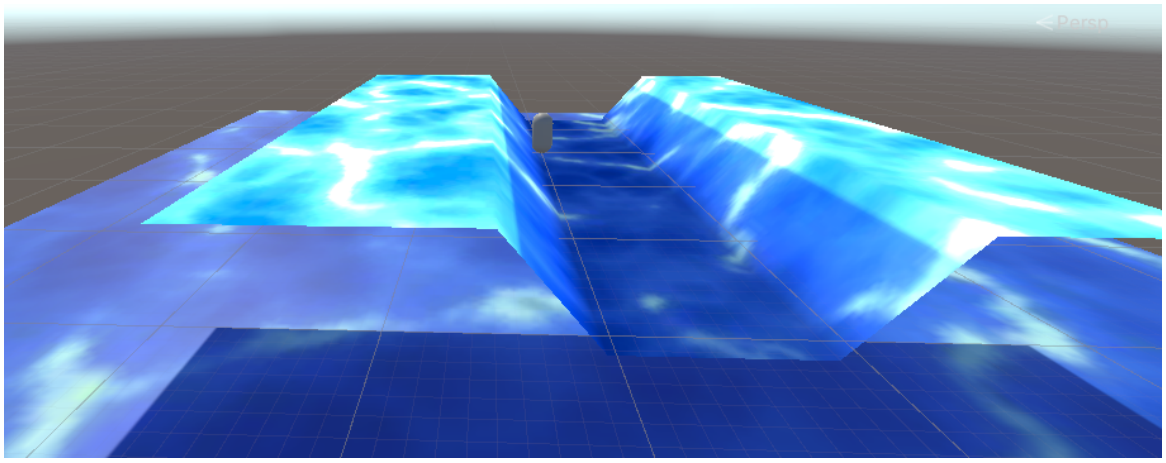
**Deferred rendering** is a more efficient way to add light to an object, it also passes geometry down the pipeline, but it uses the g-buffer to store lighting data and textures before rendering light onto the final image, adding another step.




**Part 3: 1hr 20**

Textures from FreePbr



Explanation: For the square waves, I simply included an if statement based on the wavelength that would ask if the wavelength was above a certain threshold, to set it to a specific wavelength value. In my case I asked:

```
if (waveHeight > .1)
{
    waveHeight = _Amp;
}
else {
    waveHeight = -1;
}
```

It would set the wavelength to -1 if waveHeight wasn't over 0.1.

- For the shader, I added the toonramp lighting to the water shader taught in class to accommodate the toon lighting requirement.
-

```
#pragma surface surf ToonRamp vertex:vert

sampler2D _MainTex;
sampler2D _RampTex;

float4 LightingToonRamp(SurfaceOutput s, fixed3 lightDir, fixed atten)
{
    float diff = dot(s.Normal, lightDir);
    float h = diff * 0.5 + 0.5;
    float2 rh = h;
    float3 ramp = tex2D(_RampTex, rh).rgb;

    float4 c;
    c.rgb = s.Albedo * _LightColor0.rgb * (ramp);
    c.a = s.Alpha;
    return c;
}
```

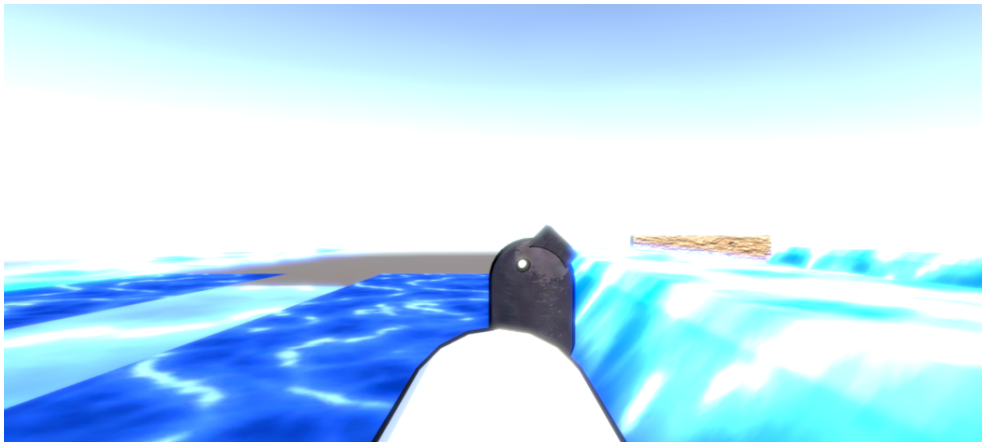
**Part 4: Explain Code <u>20 minutes</u>**

```
void OnRenderImage(RenderTexture source, RenderTexture
destination){
int width = source.width / integerRange;
int height = source.height / integerRange;
```

```
RenderTextureFormat format = source.format;
RenderTexture[] textures = new RenderTexture[16]; -> Progressively upsamples iteratively
RenderTexture currentDestination = textures[0] =
RenderTexture.GetTemporary(width, height, 0,format);
Graphics.Blit(source, currentDestination);
RenderTexture currentSource = currentDestination;
Graphics.Blit(currentSource, destination);
RenderTexture.ReleaseTemporary(currentSource);
int i = 1;
for (; i < iterations; i++) { -> Downsampling iterations loop, helps with detail despite
lower resolution
width /= 2;
height /= 2;
currentDestination = textures[i] =
RenderTexture.GetTemporary(width, height, 0,format);
if (height < 2) {
break;
}
currentDestination =
RenderTexture.GetTemporary(width, height, 0,
format);
Graphics.Blit(currentSource,currentDestination);
RenderTexture.ReleaseTemporary(currentSource);
currentSource = currentDestination;
}
for (; i < iterations; i++) { -> upsampling iteration loop
Graphics.Blit(currentSource,
currentDestination);
// RenderTexture.ReleaseTemporary(currentSource);
currentSource = currentDestination;
}
for (i -= 2; i >= 0; i--) {
currentDestination = textures[i];
textures[i] = null;
Graphics.Blit(currentSource,
currentDestination);
RenderTexture.ReleaseTemporary(currentSource);
currentSource = currentDestination;
}
Graphics.Blit(currentSource, destination);
}
```

- This is part of a c# code for a blur shader. It functions by downsampling the image rendered by the camera.
- This blur effect could be used as a screen effect in response to an in-game event such as taking damage.

**Part 5: Bloom and Outlining - <u>40 minutes</u>**(Shark texture from cgaxis)



- Bloom was applied to a camera to further enhance the sunny day look and to brighten the water.
- The outline was applied to the shark and cannon objects in the scene on them a cartoony look.
- I added toon ramp lighting as well as a bump map to the outline shader from class which is an addon to what was previously just a main texture property.
- 

```
_Outline ("Outline Width", Range(.002,0.1)) = .00
_myBump("Bump Texture", 2D) = "bump" {}
_mySlider("Bump Amount", Range(0,10)) = 1
```

```
#pragma surface surf ToonRamp //creating basic toonramp shader
sampler2D _MainTex;
sampler2D _myBump;
half _mySlider;
sampler2D _RampTex;

float4 LightingToonRamp(SurfaceOutput s, fixed3 lightDir, fixed atten)
{
    float diff = dot(s.Normal, lightDir);
    float h = diff * 0.5 + 0.5;
    float2 rh = h;
    float3 ramp = tex2D(_RampTex, rh).rgb;

    float4 c;
    c.rgb = s.Albedo * _LightColor0.rgb * (ramp);
    c.a = s.Alpha;
    return c;
}
```

- The bloom shader is the same as the one from the lecture.

**Part 6: Explain Code <u>20 minutes</u>**

```
Shader "ColoredShadow"
{
Properties{
_Color("Main Color", Color) = (1,1,1,1)
_MainTex("Base (RGB)", 2D) = "white" {}
```

```
_ShadowColor("Shadow Color", Color) = (1,1,1,1)
```
-> Setting inspector property to modify colour outside of script

```
}
SubShader{
Tags { "RenderType" = "Opaque" }
LOD 200
CGPROGRAM
#pragma surface surf CSLambert
sampler2D _MainTex;
fixed4 _Color;
fixed4 _ShadowColor;
```
-> Define variables within subshader

```
struct Input {
float2 uv_MainTex;
};
half4 LightingCSLambert(SurfaceOutput s, half3 lightDir, half atten) {
```
-> Outputs all final colour values. Uses atten value to figure out how shadows are displayed based on the brightness of the surface it's being rendered on.

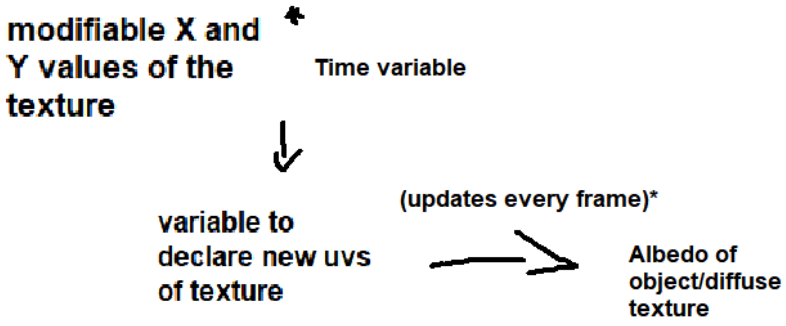
```
fixed diff = max(0, dot(s.Normal, lightDir));
half4 c;
c.rgb = s.Albedo * _LightColor0.rgb * (diff *
atten * 0.5);
//shadow color
c.rgb += _ShadowColor.xyz * (1.0 - atten);
c.a = s.Alpha;
```
- uses shadow colour value defined by the user to output shadow colour.

```
return c;
}
void surf(Input IN, inout SurfaceOutput o) {
half4 c = tex2D(_MainTex, IN.uv_MainTex) *
_Color;
o.Albedo = c.rgb;
o.Alpha = c.a;
}
ENDCG
}
Fallback "Diffuse"
}
```

- This code renders the shadow of an object so that it can either have shadows casted onto it or so it can cast shadows onto other objects. What is different about this code than a regular shadow caster is that this code changes the colour of the shadow being rendered.
- This shadow code could be used if you are trying to achieve a more cartoony or colourful art style and don't want the shadows to be coloured black and simply differed by opacity. You could also give different objects different coloured shadows with this shader.

**Part 7: Scrolling Texture <u>14 minutes</u>**

- This shader displaces the uv of the texture over time so that it seems as if the texture is moving but the object is staying in place.




- This shader could be used to simulate a skybox/moving clouds, water, a moving background in a side scroller, or any sort of menu ticker or moving screen/billboard.

| | Task | Value |
|---|---|---|
| Add all of the items and calculate the assignment's total | Create repository and Unity project | **1** /1 |
| | Explain the difference between forward and deferred rendering using a diagram | **0.9** /1 |
| | Create a toon shaded square-shaped wave. | **2.8** /3 |
| | Explain the code snippet | **0.4**/0.5 |
| | Add to shaders to the designated scene | **2.5**/3 |
| | Explain the code snippet | **0.5**/0.5 |
| | Explain any shader of your choosing | **1** /1 |
| Total | | |