Individual Assignment 2 - Review

INFR 2350 - Intermediate Computer Graphics 2023

Purpose: In this assignment, you will demonstrate your knowledge in preparation for the final exam.

This assignment is weighted at 10% of your final mark for this course.

This assignment must be done individually and done during class time on March 28, 2023. This activity is open book, but limited to course materials for exam practicing purposes.

Deliverables:
- This PDF document with your self-evaluation completing the assignment's rubric.
- A working repository with evidence supporting the self-evaluated items including screenshots and text description of what was done. Please note that items without descriptions are not awarded any points. Incomplete, or inadequate description and supporting materials will have the point being deducted to half its value.

Tasks:

	Task	Deliverable	Value
Create a repository for this assignment. Suggested time limit: 10 minutes	Make the repository is public	Proof of completing the task with screenshots and descriptions on the repository's readme	0.2/1
	Add unity gitignore		0.2/1
	Create an empty Unity 3D project		0.2/1
	Build the project		0.2/1
	Upload the build as a release to GitHub		0.2/1
Total			

	Task	Deliverable	Value
Explain the difference between forward and deferred rendering using a diagram Suggested time limit: 15 minutes	Define with your own words what deferred and forward rendering are	Proof of completing the task with screenshots and descriptions on the repository's readme	0.25/1
	Create a diagram that shows how each of these work and their differences		0.25/1
	Use the diagram to explain the differences		0.25/1
	Provide an example by describing a scene and how it could be implemented employing pseudocode or a flowchart		0.25/1
Total			

	Task	Deliverable	Value
Create a toon shaded square-shaped wave. Note the water moves. Suggested time limit: 30 minutes	Edit the empty scene on your project. Even student numbers will aim to have a scene similar to this one from Jaws the video game: While odd student numbers will use this one: Please keep in mind that you are not being asked to recreate this scene faithfully. You are being asked to create one that is similar. You can explain your decisions on how you decided to tackle this task to ensure the scene resembles the designated one. The scene should present basic movements controlling the ship or the shark.	Proof of completing the task with screenshots and descriptions on the repository's readme	0.75/3

	Explain how the shaders were implemented.		0.75/3
	Explain the modifications done to the shaders and how they differ from the ones given in class and tutorials. If the shader does not present modifications, no points are awarded.		0.75/3
	Create a build for this task and upload it as a release on GitHub		0.75/3
Total			

	Task	Deliverable	Value

Explain the following code snippet	Proof of completing the task with screenshots and descriptions on the repository's readme	
| Suggested time limit: 10 minutes | ```
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];

    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++) {      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++) {      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);    }
``` | |
Highlight text to explain the code		0.1/0.5
Explain what the code does		0.2/0.5
Provide an example of where this could be used		0.2/0.5
Total		

	Task	Deliverable	Value
Add any two of the following: Bloom Shadows Outlining Vertex extrusion Suggested time limit: 50 minutes	Use the previous Jaws scene and build on top of it. Even student numbers will aim to have a scene similar to this one from Jaws the video game: While odd student numbers will use this one: Please keep in mind that you are not being asked to recreate this scene faithfully. You are being asked to create one that is similar. You can explain your decisions on how you decided to tackle this task to ensure the scene resembles the designated one.	Proof of completing the task with screenshots and descriptions on the repository's readme	
	Explain how the shaders were implemented.		1/3
	Explain the modifications done to the shaders and how they differ from the ones given in class and tutorials. If the shader does not present modifications, no points are awarded.		1/3
	Create a build for this task and upload it as a release on GitHub		1/3
Total			

Task		Deliverable	Value

Explain the following code snippet
 	
Suggested time limit: 10 minutes
	``Shader "ColoredShadow"
	{
	  Properties{
	  _Color("Main Color", Color) = (1,1,1,1)
	  _MainTex("Base (RGB)", 2D) = "white" {}
	  _ShadowColor("Shadow Color", Color) = (1,1,1,1)
	  }
	    SubShader{
	
	    Tags { "RenderType" = "Opaque" }
	    LOD 200
	
	    CGPROGRAM
	    #pragma surface surf CSLambert
	
	    sampler2D _MainTex; fixed4
	_Color; fixed4 _ShadowColor;
	
	    struct Input {
	      float2 uv_MainTex;
	    };
	
	    half4 LightingCSLambert(SurfaceOutput s, half3 lightDir, half atten) {
	
	      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; 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" }```
		Proof of completing the task with screenshots and descriptions on the repository's readme	

Highlight text to explain the code			0.1/0.5
Explain what the code does			0.2/0.5
Provide an example of where this could be used			0.2/0.5
Total			

Individual Assignment 2 - Review

INFR 2350 - Intermediate Computer Graphics 2023

| Choose any shader seen in the second half of the term that has not been covered in your previous responses

Suggested time limit: 15 minutes	Task	Deliverable	Value
	Explain how the chosen shader works	Proof of completing the task with screenshots and descriptions on the repository's readme	0.25/1
	Create a diagram that shows how it works		0.25/1
	Explain where this can shader be used		0.5/1
Total			

Once you complete the activities, please input the values on the following table and calculate the total. Do note that this activity allows you practice and identify strengths and weaknesses towards the final. After this is done, compare notes with classmates. Most importantly, make sure you understand the shaders seen in the second half of the term and you can use them beyond what was provided in class, making them suit any given scene. I hope you have enjoyed this activity.

	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	1/1
	Create a toon shaded square-shaped wave.	2.5/3
	Explain the code snippet	0.3/0.5
	Add to shaders to the designated scene	3/3
	Explain the code snippet	0.5/0.5
	Explain any shader of your choosing	1/1
	Total	9.3

Forward rendering is essentially a rendering method in which you supply the graphics card with geometry and it projects it and breaks it down into vertices which are then turned into fragments or pixels. This is a fairly linear process and each geometry is passed down one by one throughout the pipe until the final image is produced.

Deferred rendering is a bit different as the rendering is deferred until all of the geometries have gone through the pipe, afterwards, it applies the shading to complete the final image.

As we can see from the diagram if we were to use forward rendering it could be relatively fast and efficient but it could start to struggle when faced with a large number of complex materials, light sources, or even complex scenes. That's, where something like deferred rendering could be more beneficial as this method, separates the rendering process into two passes the geometry and then the lighting pass. This could be more beneficial when handling the previous problems faced with forward rendering.

For the wave shader I followed along with the lecture slides, Basically what the script is doing from what I understand is it takes vertices within the vertex shader and transforms them on some time-dependent function. We start off by creating a shader script. In this script, we want to create a waving surface by transforming the verts of our plane based on a sine wave. In the shader, we create a vertex shader. This is where things get a bit different from the lecture slides. We were tasked to create a square wave. This was a lot more challenging than I expected. I tried first playing around with certain values in amplitude and frequency thinking maybe that could change the shape of a wave. But I soon realized that doing that just changes the peak height and explosiveness of the wave. Overall it would keep the same sine wave shape. By adjusting the sine wave function I realized that I was now getting the triangle wave which was closer but not square. I thought maybe adjusting more numbers in the equation would fix my issue. But after a bit of research, I realized that to get a square wave you actually need to get 2 values for the wave. To do this I basically just looked for the 0 and 1 of the wave. I decided to use the step function which takes a value of 0-length where the length is 1. If the value is larger than 0.5 then it detects the wave as its peak which returns a value of 1. Whereas if the value was less than 0.5 it detects the wave as down which returns a value of 0. I also use the modulus operator to create periodic waves that repeat after a chosen amount of units. As seen in the example I chose 1, but this can be changed to create different extents of square waves.


```
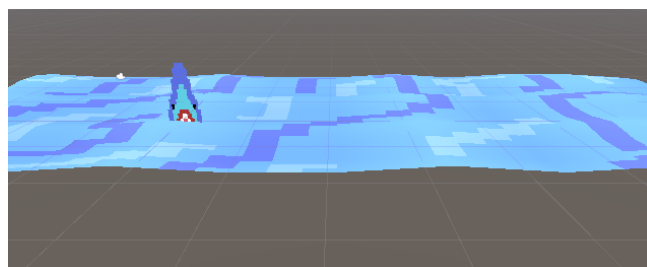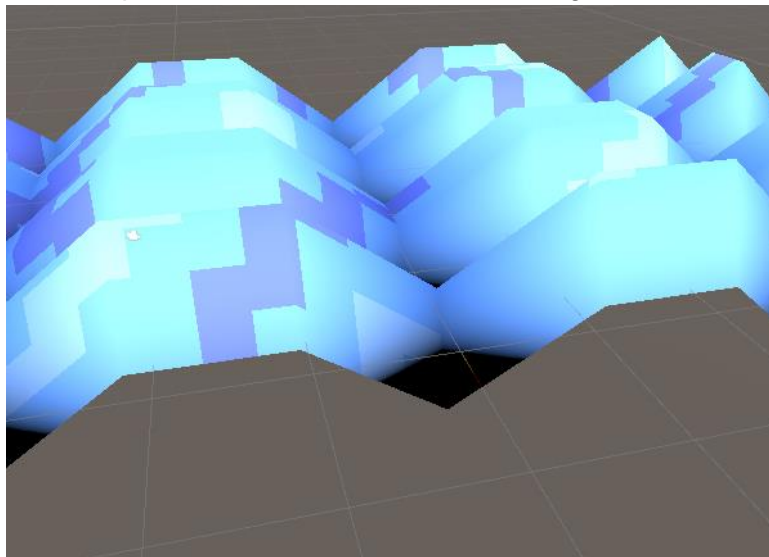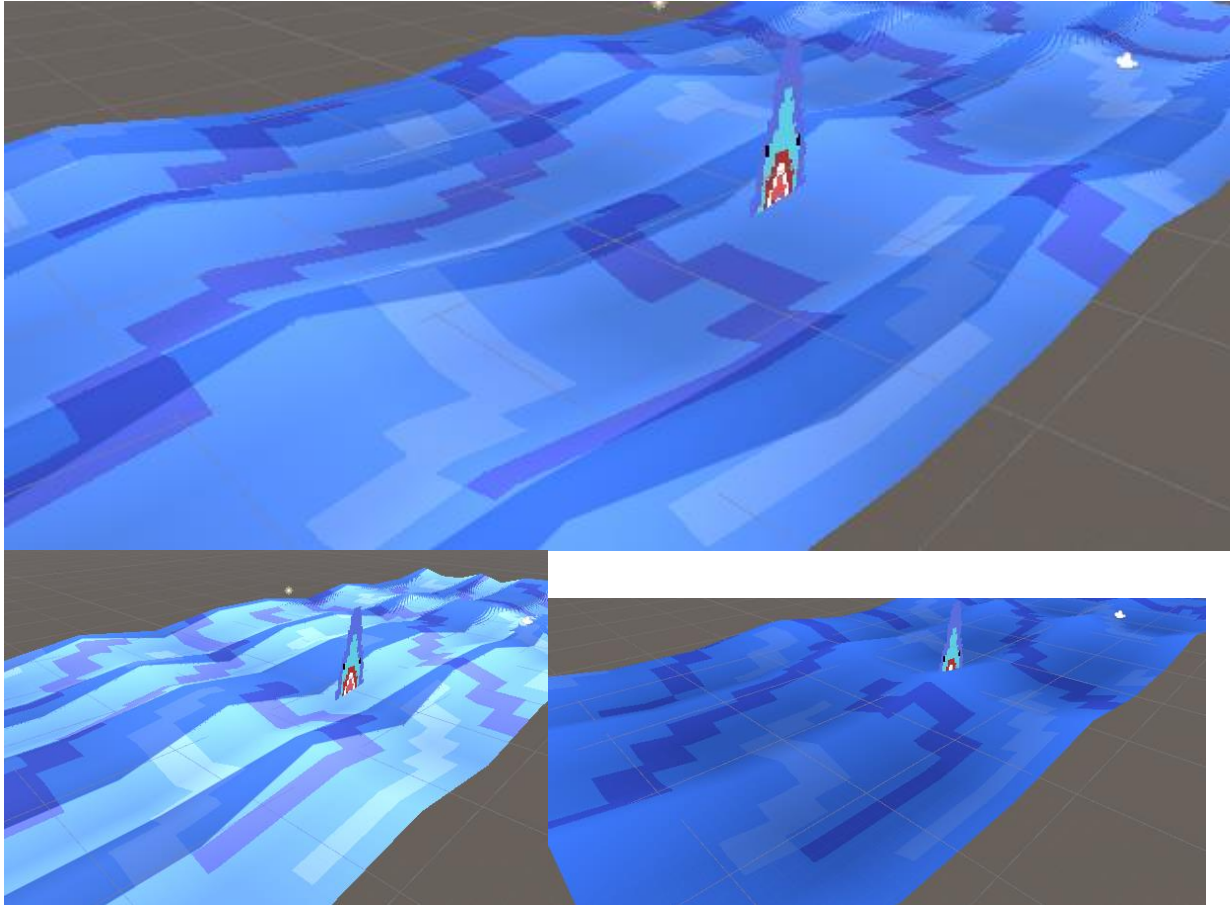float t = _Time.y * _Speed;
float waveHeightX = ((_Amp * (step(0.5, (t + v.vertex.x * _Frequency) % 1.0) * 2.0 - 1.0)) * 2.0);
float waveHeightZ = ((_Amp * (step(0.5, (t + v.vertex.z * _Frequency) % 1.0) * 2.0 - 1.0)) * 2.0);
float waveHeight = waveHeightX + waveHeightZ;
v.vertex.y += waveHeight;
```

In terms of toon shader, this shader basically works by calculating the color of a pixel based on the lighting and properties of the object. In simple forms, it's a non-photorealistic rendering designed to make graphics appear flat by using simpler shading colours instead of gradients, tints, or shades. In terms of how the code works, it takes 3 parameters including the surface being shaded, the light direction, and the attenuation of the light. It first calculates the amount of light reflected by the surface. It basically does this by taking the dot product of the surface's normal and light direction and then scaling it, and finally multiplying it by the attenuation value. After this, we call for our ramp texture and store the colours in the texture in the ramp variable. Finally, the function then makes a new colour vector that sets the RGB values depending on the surface albedo, ramp value, and the colour of the light. After I created the toon shader I added my own modification to it by adding a ramp smoothness variable. I multiply this by the ramp within my code to apply it to the shader correctly. By multiplying the ramp to ramp smoothness it controls the degree of smoothing done within the ramp texture. It assigns a certain value, if the value is 0 then the smoothness of the texture will result in hard edges between each of the shades, while if the value is 1 it will create a very smooth transition between the shades. For the toon shader to function correctly we need to add a ramp texture. I basically created this in Aseprite and just made different shades of colours lined up beside each other varying from white to black.

```
float diff = (dot(s.Normal, lightDir) * 0.5 + 0.5) * atten;
float2 rh = diff;
float3 ramp = tex2D(_RampTexture, rh).rgb;
float4 c;
c.rgb = s.Albedo * _LightColor0.rgb * (ramp * _Smoothness);
c.a = s.Alpha;
```

1ST CODE SNIPPET EXPLAINED

The code basically is used for image processing and created multiple textures before producing the final output. Its basically just upscaling and downscaling multiple textures. After the final result is applied to the destination texture it releases the other textures so they can be reused. It basically just calculates the height and width of the image by the source and integer

```
int width = source.width / integerRange;
int height = source.height / integerRange;
RenderTextureFormat format = source.format;
RenderTexture[] textures = new RenderTexture[16];
```

range. It then creates an array of 16 textures.
We then use the Graphics.Blit method so that we can copy the image to the current destination. After doing so it sets the current source to its destination.

```
Graphics.Blit(source, currentDestination);
RenderTexture currentSource = currentDestination;
Graphics.Blit(currentSource, destination);
RenderTexture.ReleaseTemporary(currentSource);
```

In the for loops, it basically just repeats this process repeatedly depending on if it's downscaling or upscaling, it checks to see what it is doing, and if it's not upscaling then it exits and checks for downscaling. Once the final loop is finished, it is copied to the destination image.

```
        for (i -= 2; i >= 0; i--) {
            currentDestination = textures[i];
            textures[i] = null;
            Graphics.Blit(currentSource,
currentDestination);
            RenderTexture.ReleaseTemporary(currentSource);
            currentSource = currentDestination;
        }

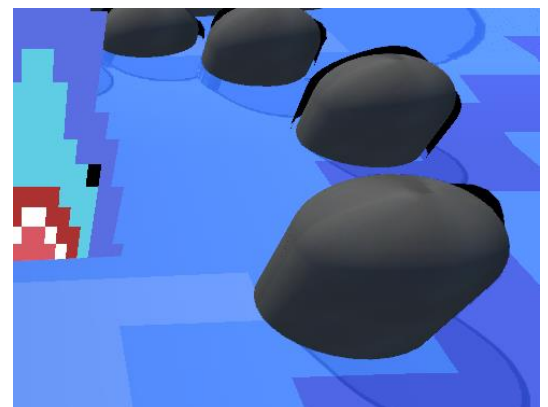        Graphics.Blit(currentSource, destination);
    }
```

Adding in Outlines and Vertex Extrusion

For the outline shader I decided to add rocks into the scene and add outlines to them. The reason why I chose this approach over adding outlines to the sea is that when I added outlines to the sea it was not visible. I did a bit of research and found out that outlines cannot appear on a plane. I then tried taking a different approach and made a cube. While it worked with the cube, the wave stopped working properly and would be very janky. I assumed this was because of the cube not being subdivided. So I opened up blender and made a cube with a ton of subdivisions. Although when I brought it into unity it still didn't work as expected. So here is how I added the outline shader to the rocks and my modifications that were made. Starting off we got the main texture, this is just in the code so that I could apply a texture to the object as well as adjust the colour. We then render the object again through code by running it through pass. We are then moving the vertex position from object space to clip space. We then apply the offset alongside a colour and return it to the object. How I decided to modify this code was kind of interesting. I remembered back earlier into the assignment when using sine waves to create the wave shader. I thought what if I were to use sine waves to randomise the lines within this shader. I upped the speed of which the line moved so it wouldn't be stationary. When I tried doing this, I had a really cool wavey effect. After playing around with my sine equation a bit I realized multiplying the position by variables could make a more jagged cool looking animated outline. So I made a couple of sliders for them, hopped into unity and had a pretty cool adjustable jagged outline. (Next page for vertex extrusion)

```
offset *= sin(o.pos.x * _xVar + o.pos.y * _yVar + _Time.y * _outlineSpeed);
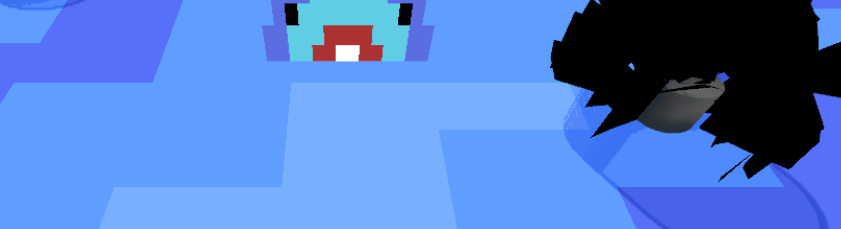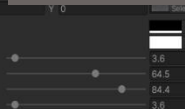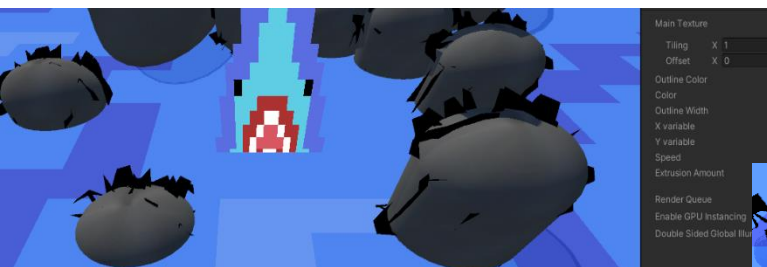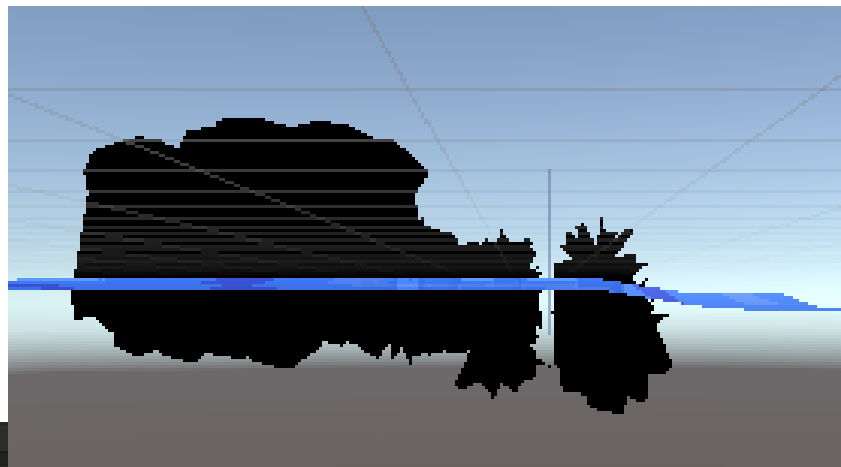```

## Adding in Outlines and Vertex Extrusion continued

I then decided to add a vertex extrusion. The vertex extrusion essentially works. It gets the vertices and position of our objects. It then multiplies the vertex normal by an extrusion amount which I made an adjustable slider for. With this you can make your object much thicker. But I also tried putting it in the negatives and got some cool weird object formations where the object kind of went inwards. It created a cool effect but not really one I was looking for. I then started to modify the extrusion and see what I would be able to come up with. After a ton of playing around I decided to try and animate my extrusion. But in my attempt to animate the extrusion I probably found one of my favourite things I have ever done. I tried multiplying the equation by _Time.y. At first I thought it made the object invisible, I was trying to figure out how it was doing that because it made no sense to me. Turns out, it just made no sense in general. I found out that this allows it to exponentially grow or you could also make if grow negatively if that makes sense. I found that out while trying to make it shrink but we were experimenting a bit. Overall though I realized that the extrusion of the rock slowly over time was actually pretty cool. That is the modification I decided to stick with and I actually think it would be cool to further experiment with this as I could possibly animate objects such as like a bomb to show off the bopping to an explosion. But the thing is having it grow continuously in one direction without shrinking back is kind of weird. So I think potentially adding a pingpong effect and making it switch between positive and negative time would be really cool to try out and having them switch when hitting certain values or after a set time.







```
struct appdata {
    float4 vertex: POSITION;
    float3 normal: NORMAL;
    float4 texcoord: TEXCOORD0;
};
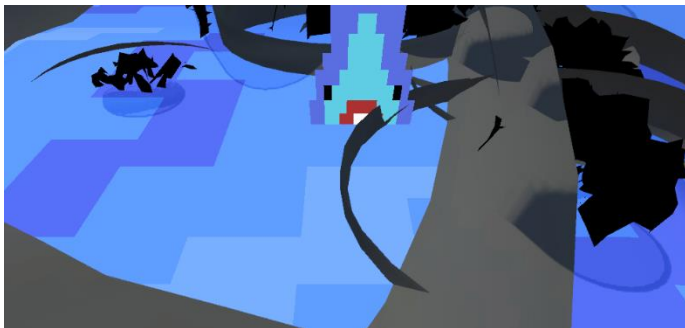
void vert(inout appdata v) {
    v.vertex.xyz += v.normal * _Amount * _Time.y;
}
```

2ND CODE SNIPPET EXPLAINED

This little code snippet is actually a bit interesting. It's a shadow colour shader! To start off we are going to get our main texture, shadow colour, and colour.

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

We can then see that the material is going to be rendered as opaque with a level of detail of 200

```
Tags { "RenderType" = "Opaque" }
LOD 200
```

We then add lambert lighting model which calculates all lighting applied to the object. After we do this, we can see where the shadow is going to be. From there we select a shadow colour and multiply it by the albedo colour as well as the colour of light.

```
        half4 LightingCSLambert(SurfaceOutput s, half3
 lightDir, half atten) {

        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;
        return c;
        }
```

For the final section we take the main texture and multiply it by the main colour of the material. Finally it sets the albedo to the color and the alpha to the sampled texture.

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

I believe this could be used in a situation such as if you were trying to create a very warm environment and show a room lit up with a torch or fireplace. You could potentially make the objects in the room have shadows with a slight reddish orange colour. Another way I believe this could possibly work could be if you have like a jello like game environment where everything is made of the same textures, you could apply the shadows to have a greenish colour applied. The final thing I could possibly think of a situation for is maybe a sunset over a city, having certain lamp posts or benches off roads give a slight colour shading change.

Explaining a shader of choice

For this section I decided to go with a Glass shader. Glass shaders do pretty much exactly what they say, they create glass. Firstly, we need a glass texture. Once we have one the shader takes the vertices from the window pane and put them into clip space instead of world space. It then calculates the position within the screen space. Afterwards the fragment positions everything on the screen. Finally we add a tint that will allow to create a see-through type effect by multiplying the colour by the one in the texture. We could also add more detail to the glass by adding bump nodes. Note this isn't something that is required but for things such as coloured glass it could add a nice touch. We can add this bump node by creating an offset that can be scaled using its UV scale. From there we can get our bump texture and we have our bump node applied! In terms of where this shader could be used. I could say there are many case scenarios where a glass shader would be useful. Whenever you have any sort of window you could use a glass shader to achieve these effects. You could also use a glass shader to make the lenses of glasses or maybe even monitor/tv screens. Actually one great example I could connect to our GDW game where we have a digital clock and each capsule holding the time is made of glass, which would make this shader a perfect fit for that scenario. Last scenario I'll list is potentially lightbulbs or even fishbowls. Overall I think the glass shader can really come in handy and its definitely a great shader to understand and have in your toolbelt.

Optional

Glass texture → Shader Vert Function → Shader Frag Function → Bump Node → Glass