Intermediate Computer Graphics

Assignment #1

INFR 2350

Dr. Hogue

Safa Nazir - 100654328

Lillian Fan - 100672027

PART 1: Lighting

Description of Math:

The Phong lighting equation requires 4 different types of light to be added together to create a final realistic model.
I = ambient + diffuse + specular

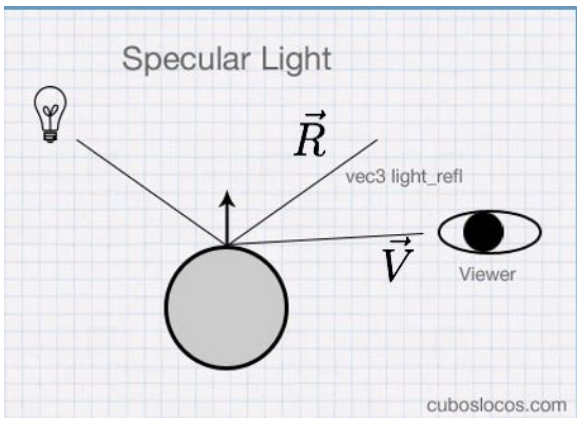
For Ambient lighting we take the light's colour and multiply it with a constant intensity value, K. It is then multiplied by the object's colour and used as the fragment's colour in the frag shader.

The calculate the diffuse component of the lighting model, we use this formula:

$$I = \frac{K_d I_p (\vec{N} \cdot \vec{L})}{d^2}$$

Where Kd is the lighting constant, Ip is the intensity of the light source and N dot L refers to the diffuse impact has on the current fragment.

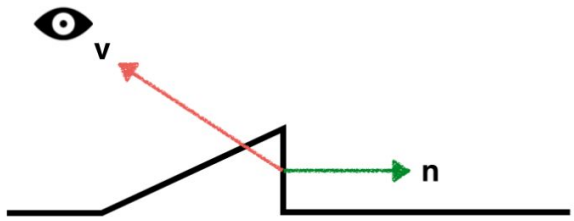
Specular light is the bright spot of light that appears on an object when a light is shined on it. This is based on the light's direction vector and the obj's normal vector but it is based on the view direction, unlike the diffuse and ambient components.



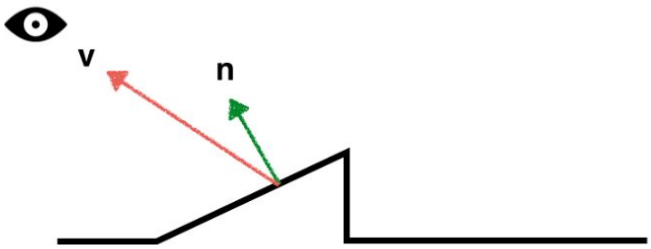
$$I = K_s I_p (\vec{R} \cdot \vec{V})^n$$

We take the same lighting constant and intensity and multiply it by the dot product of the reflect and view direction. Exponent n will change the "shiny" value of the light and we can change as we see fit.

For the rim shader,



The angle between the surface normal and the view direction is needed so we can compute how much rim contribution there will be. If the angle between the normal and view is bigger, the rim contribution will be greater and you can see a better outline on the edges.



If the angle between the 2 vectors is smaller, the contribution will be less.The rim contribution is determined by taking the dot product of the view direction and normal and is set so it cannot go beneath 0. If the rim contributor is too low, you will not be able to see the rim shading.

Shader code:

We start off our basic fragment shader like this:

```
vec2 texOffset = texcoord;
vec4 objectColor = texture(uTexAlbedo, texOffset);
```

A check to see whether or not ambient lighting is active. The ambient strength is multiplied by the light's colour which results in the final outColour value.

```
if(uAmbLightActive)
{
    float ambientStrength = 0.5;
    vec3 ambient = ambientStrength * uLightColor;
    vec3 result = ambient * objectColor.rgb;
    outColor = vec4(result, objectColor.a);
}
else
{
    float ambientStrength = 0.0;
    vec3 ambient = ambientStrength * uLightColor;
    vec3 result = ambient * objectColor.rgb;
    outColor = vec4(result, objectColor.a);
}
```

Check if specular light and the ramp is active or not. If specular light is active, the ramp can be applied onto it. The final outColor.rgb value will take the specular light equation and multiply it by the ramp texture.

```
if(uSpeLightActive)
{
    vec3 normal = normalize(norm);
    vec3 lightVec = uLightPosition - pos;
    float dist = length(lightVec);
    vec3 lightDir = lightVec / dist;
    vec3 reflection = reflect(-lightDir, normal);
    vec3 eye = normalize(-pos);
    float specularStrength = 0.5;
    float viewToReflect = dot(eye,reflection);
    if(uSpecRampActive)
    {
        outColor.rgb += uLightColor * objectColor.rgb *
            texture(uTexToonRamp,vec2(viewToReflect, 0.5)).rgb;
    }
    else
    {
        float spec = pow(max(viewToReflect,0.0),16);
        vec3 specular = specularStrength * spec * uLightColor;
        outColor.rgb += specular * objectColor.rgb;
    }
}
```

Below is the code to check whether or not rim light is active. If active it will take the dot product of the eye direction and normal, which gives us the value of the rim. Then this rim value is multiplied by the object's color and is applied to the final outColor.rgb value.

```
if(uRimLightActive)
{
    vec3 eye = normalize(-pos);
    float vdn = 1.0 - max( dot( eye, norm),0.0 );
    outColor.rgb += objectColor.rgb * vdn;
}
```

Check to see if Diffuse light is active or not. When active, the toon ramp will be applied on top of the diffuse lighting. This is similar to the way the ramp was applied onto the specular light.

```
if(uDiffLightActive)
{
    vec3 normal = normalize(norm);
    vec3 lightVec = uLightPosition - pos;
    float dist = length(lightVec);
    vec3 lightDir = lightVec / dist;
    float NdotL = dot(normal, lightDir);
    if(uDiffRampActive)
    {
        NdotL = NdotL * 0.5 + 0.5;
        outColor.rgb += objectColor.rgb * uLightColor
            * texture(uTexToonRamp2, vec2(NdotL, 0.5)).rgb;
    }
    else
    {
        float diff = max(NdotL, 0.0);
        vec3 diffuse = diff * uLightColor;
        vec3 result = diffuse * objectColor.rgb;
        outColor.rgb += result;
    }
}
```

Finally, we output our final fragment colour by applying the emissive texture.

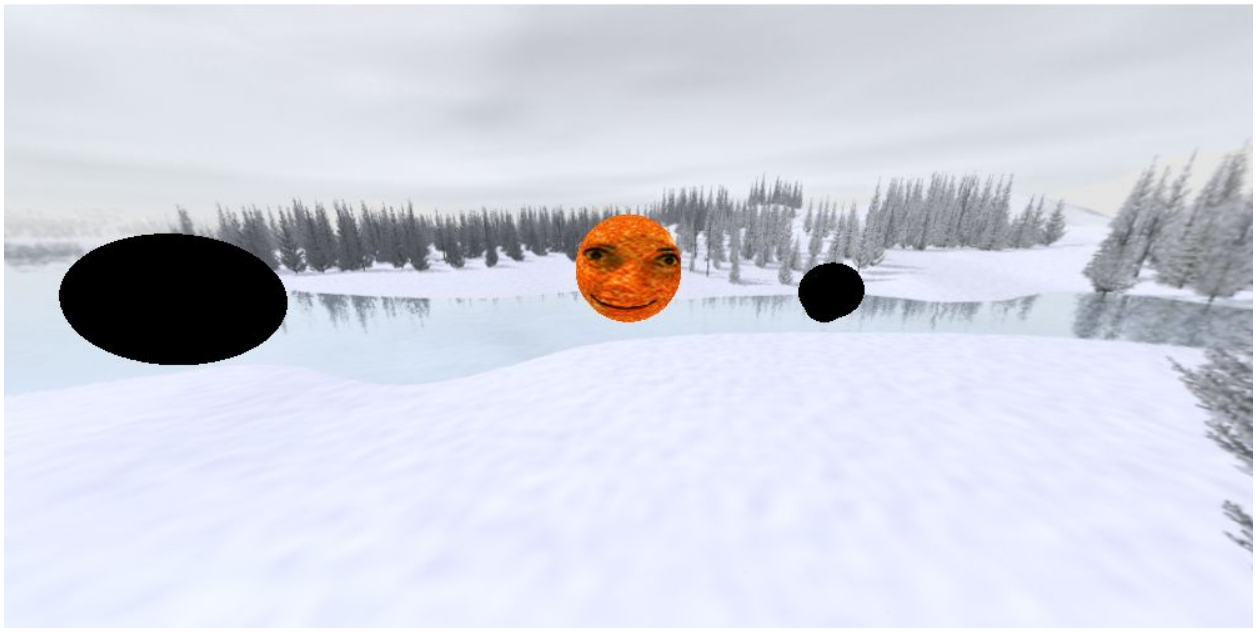
```
outColor.rgb += texture(uTexEmissive, texOffset).rgb;
```

This is the shader for full screen quad needed for the 3D LUTs. It checks if a LUT is active and combine the source image and the lookup table texture, which outputs the final picture of the screen.

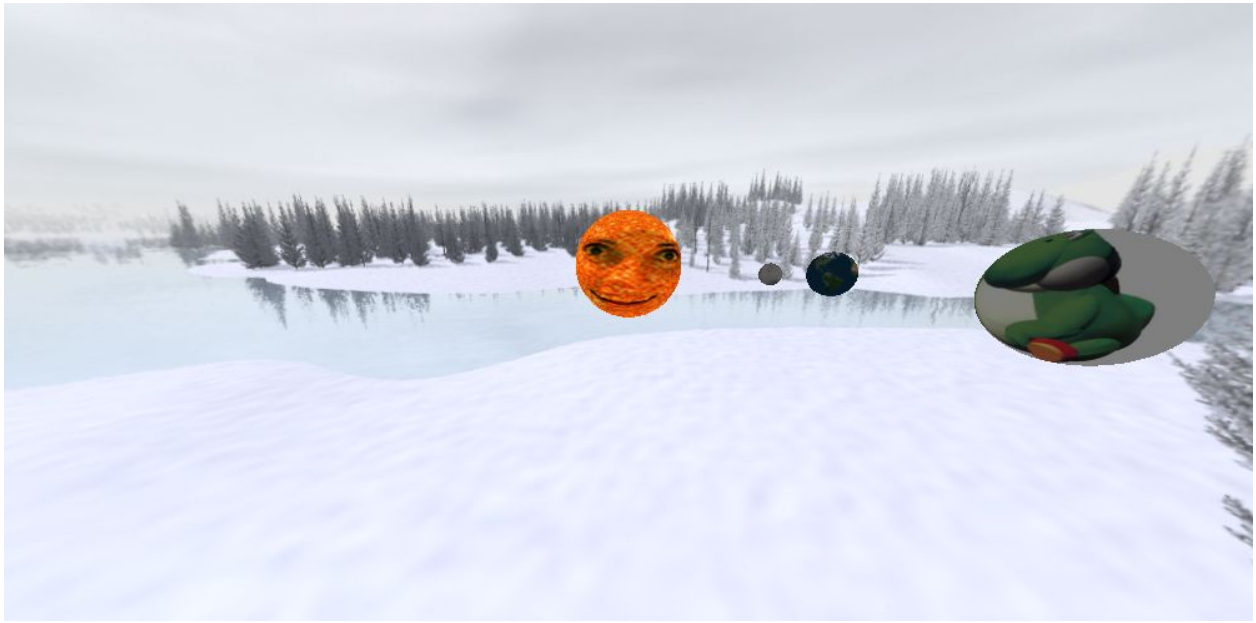
```
vec4 source = texture(uSceneTex, texcoord);
if(uGradWarmActive)
{
    vec3 scale = vec3((lutSize - 1.0) / lutSize);
    vec3 offset = vec3(1.0/2.0*lutSize);
    source.rgb = texture(warmTexure, source.rgb * scale + offset).rgb;
}

if(uGradCoolActive)
{
    vec3 scale = vec3((lutSize - 1.0) / lutSize);
    vec3 offset = vec3(1.0/2.0*lutSize);
    source.rgb = texture(coolTexure, source.rgb * scale + offset).rgb;
}

if(uGradCustomActive)
{
    vec3 scale = vec3((lutSize - 1.0) / lutSize);
    vec3 offset = vec3(1.0/2.0*lutSize);
    source.rgb = texture(customTexure, source.rgb * scale + offset).rgb;
}

outColor = source;
```

Screenshots showing different lighting:

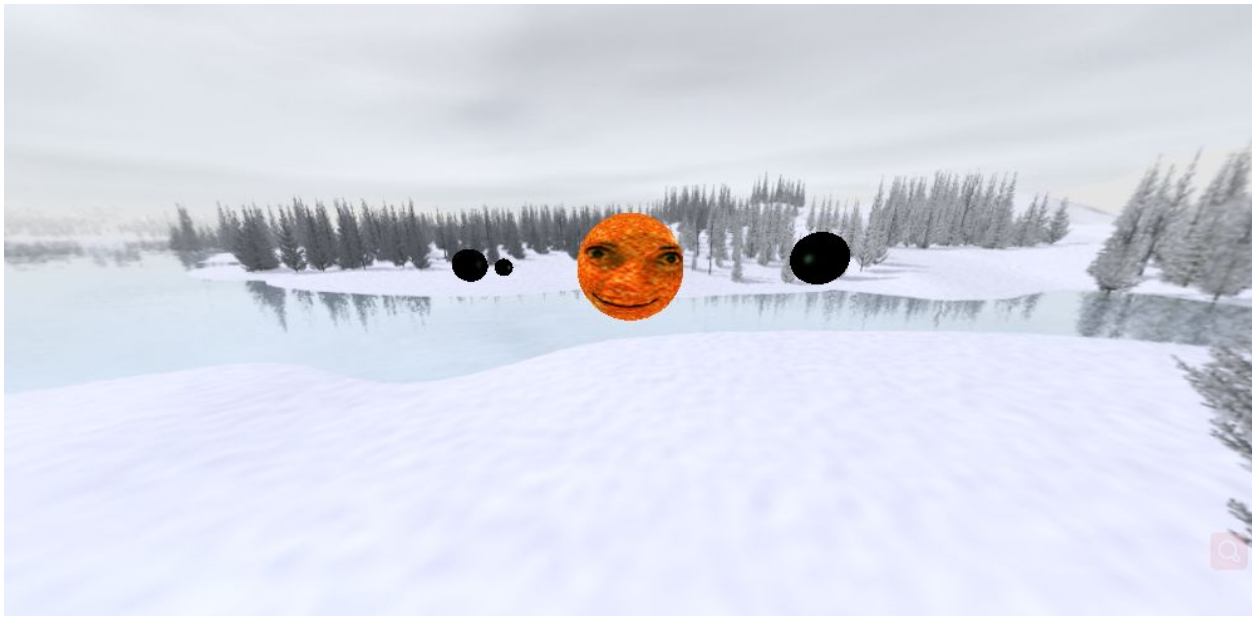
1. No Lighting



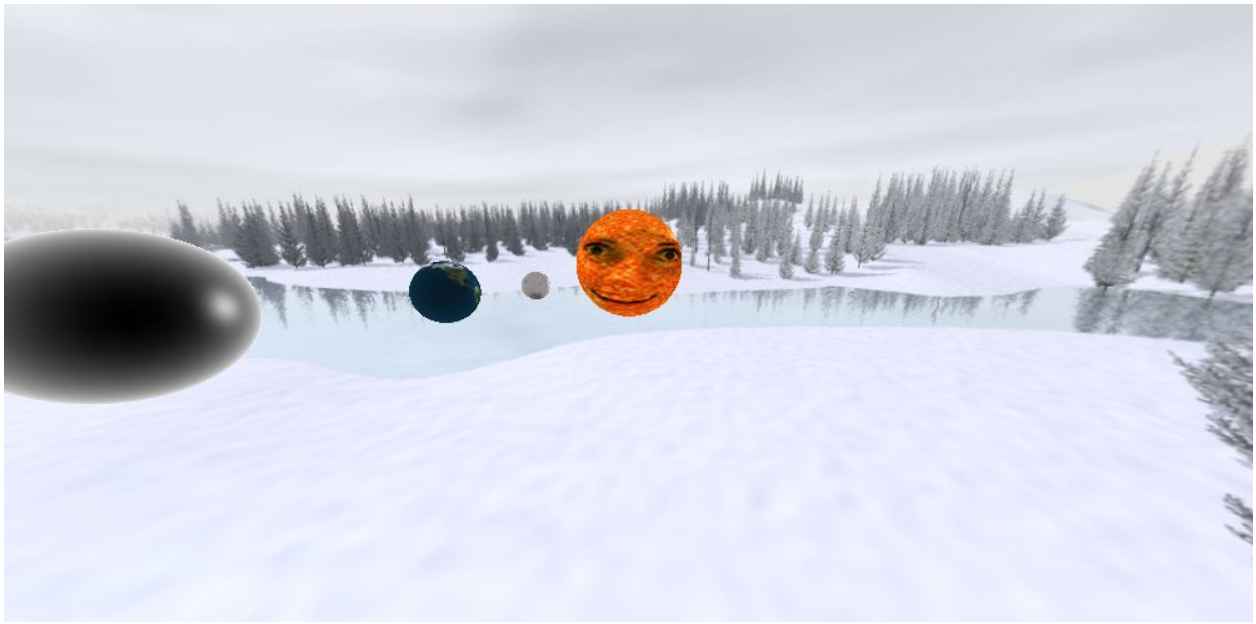
2. Ambient lighting only, shows light that has evenly bounced around the surface



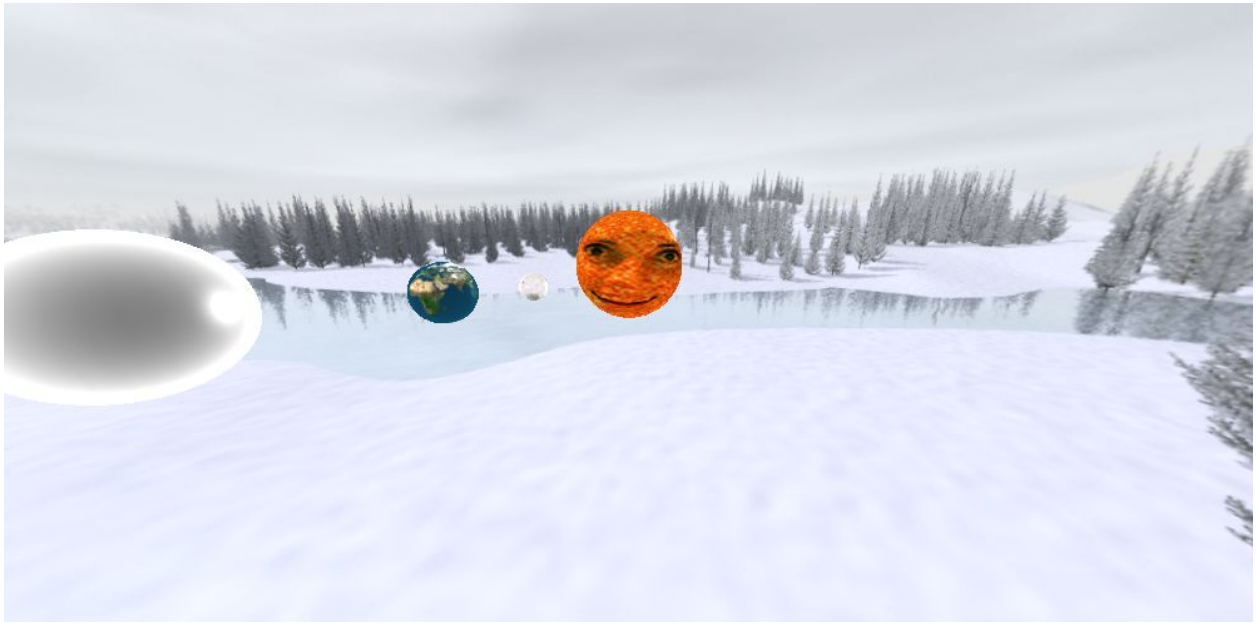
3. Specular lighting only. It depends on the viewing angle and we can see a shiny dot of bright light on the surface of the orbiting planets.



4. Specular + rim lighting. We can see that the edges of the shapes are glowing using the rim shader.

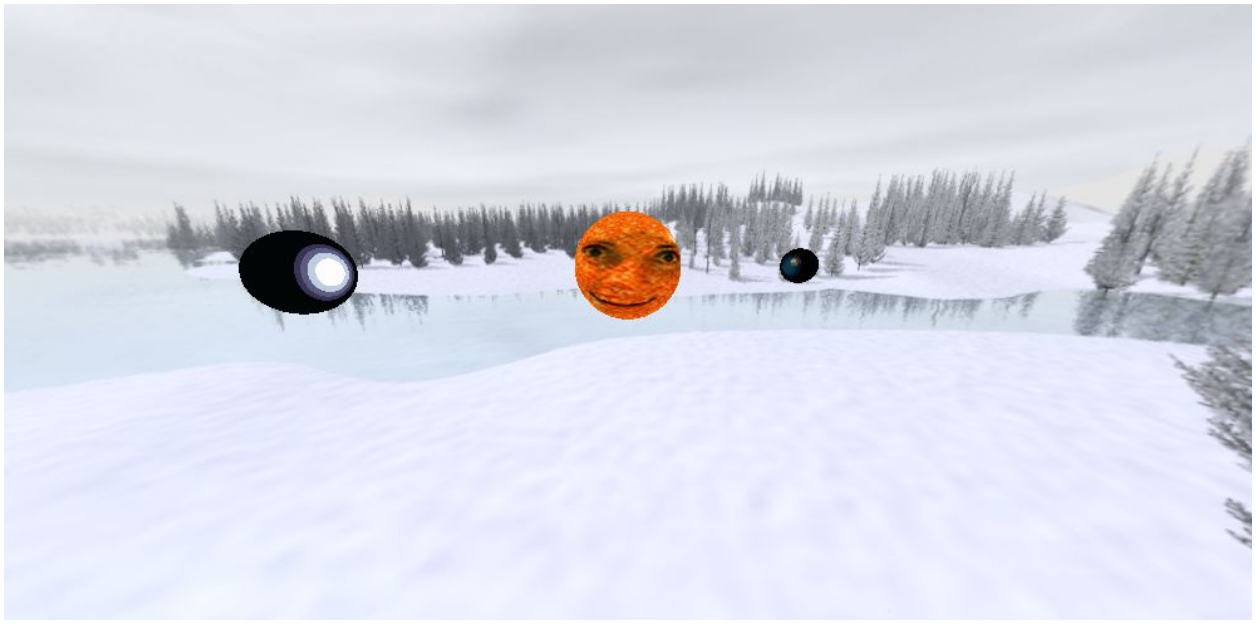


5. Ambient + specular + rim lighting.



6. Diffuse Warp/Ramp lighting. This uses a ramp and it disperses evenly on the surface of the planets.

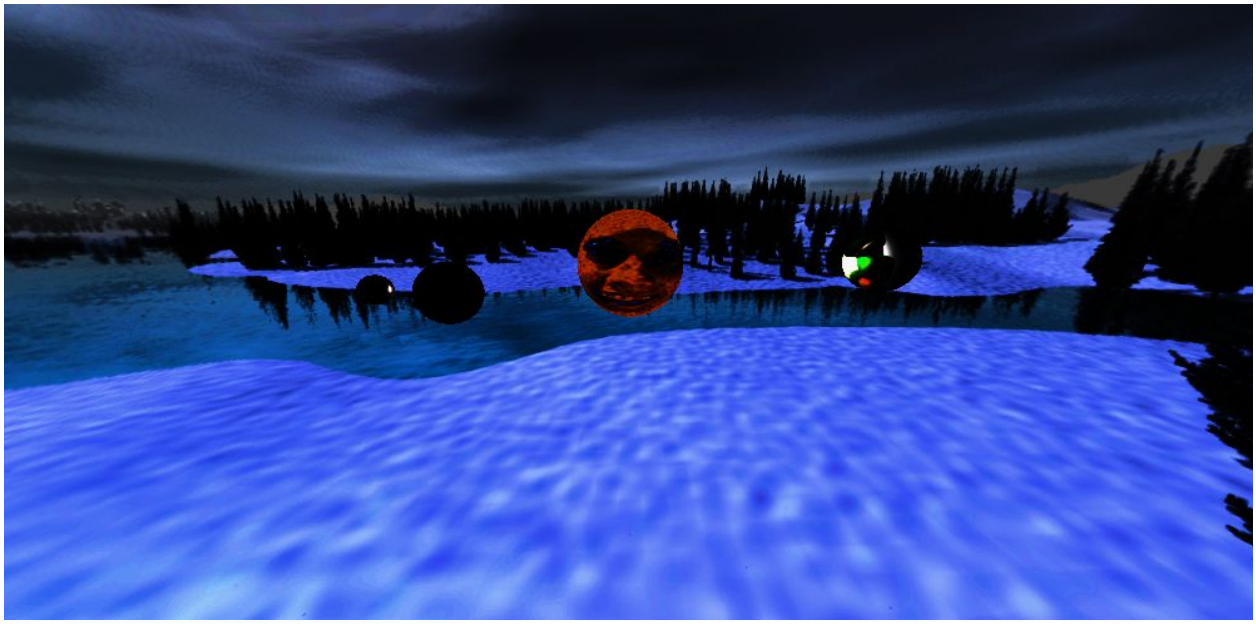
7. Specular Warp/Ramp. In the specular ramp, the ramp is only visible on the specular light, which is a spot of bright light.



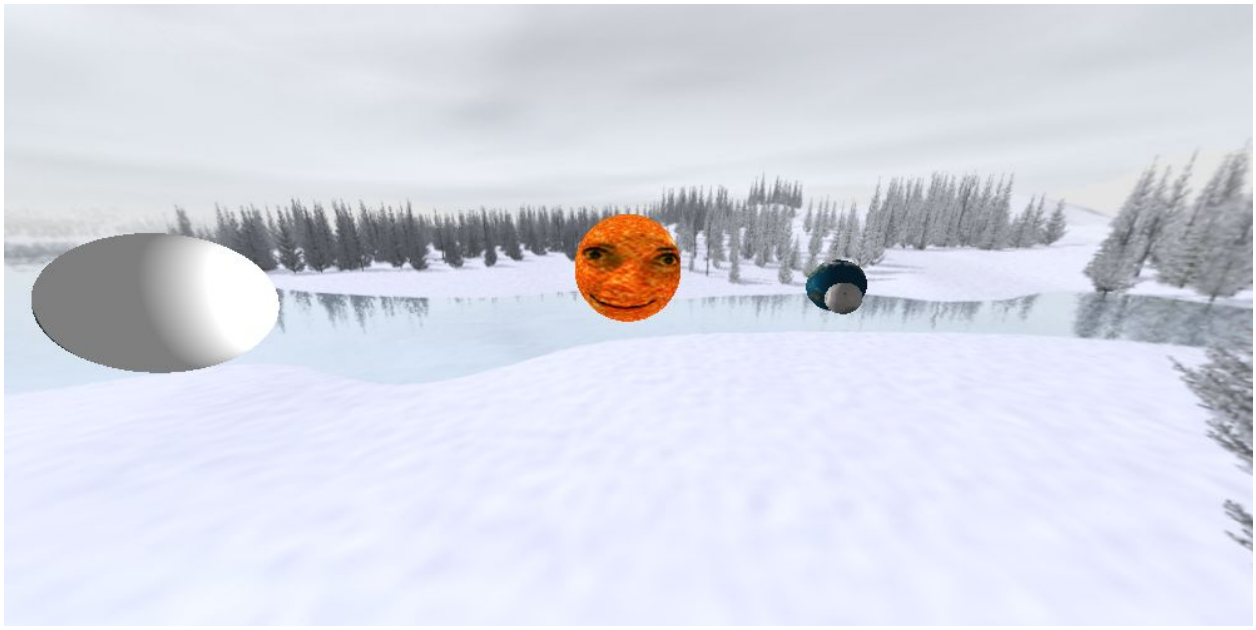
8. Warm Color Grading: We placed a warm LUT table with more yellow/orange rgb values on the fullscreen quad.

9. Cool Color Grading: We used a “cool” LUT that had higher b values and that results in the screen looking more blue.

10. Custom Effect Color Grading: We used a custom LUT that looks different than our warm/cool LUTs, we changed the brightness and contrast of the rgb values and that results in this image:

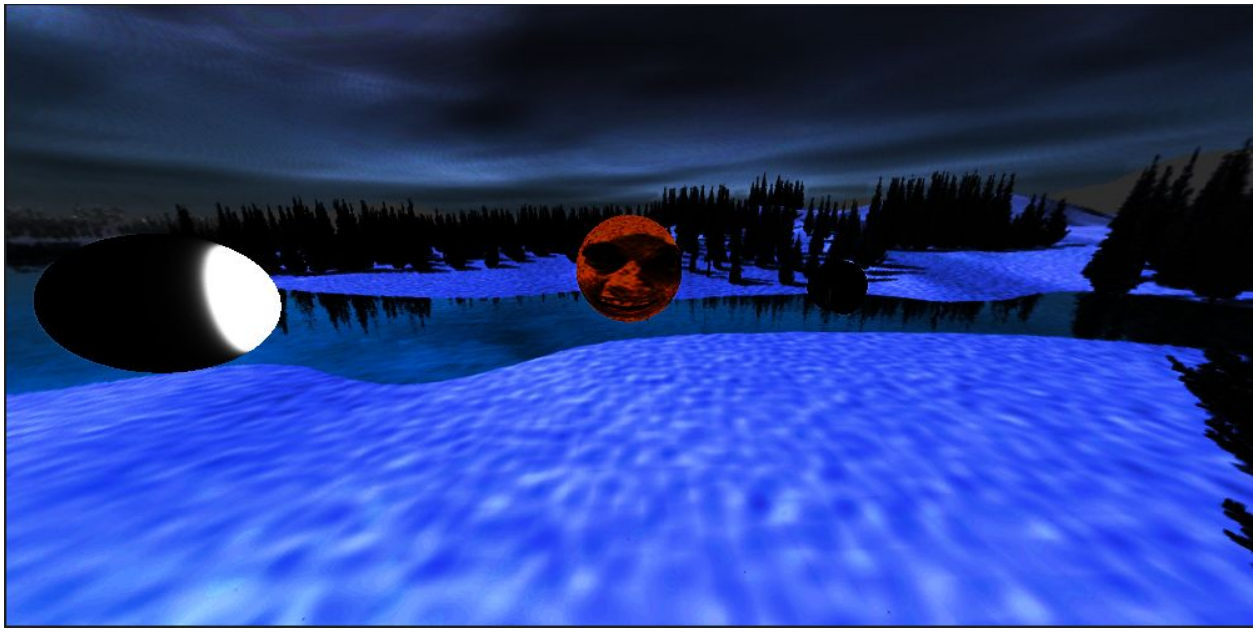


PART 2: Color Grading



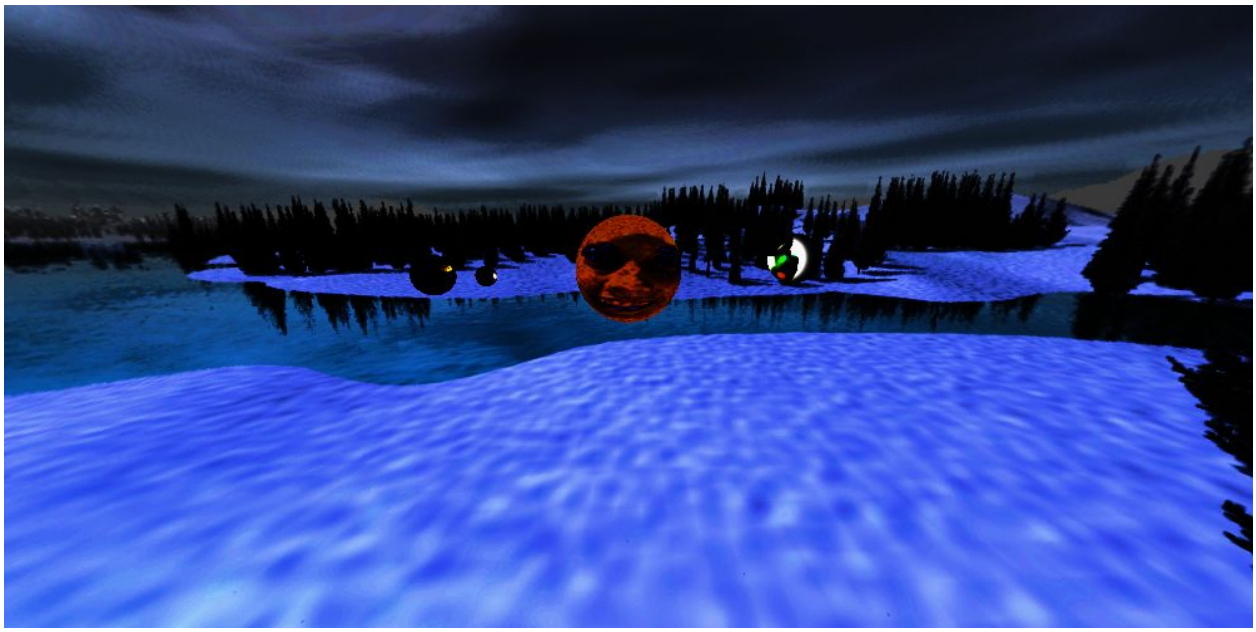
Above is a screenshot from Part 1.

In Part 1, we have already posted screenshots of the before and after using the LUTs. For the custom LUT, I would like to compare what the original screenshot looks like with the filter to the custom LUT we used in the program.



Pictured above is what the screenshot looks like with the filter we applied, and below is an image of what the same scene looks like in OpenGL.

As you can see below, the results look almost identical. The only difference we notice is that the lookup table is not as vibrant. We think that when the LUT gets converted to a .CUBE file, a very small amount of data is lost and it loses a very small amount of it's RGB value, making it slightly less bright.



Sources:

http://roxlu.com/2014/037/opengl-rim-shader
https://learnopengl.com/Lighting/Basic-Lighting
https://generator.iwltbap.com/
lecture03_lightingPPT(1), Dr Hogue.