INFR 2350U Winter Midterm 2022

Group Members:

Ricardo Prato - 100787893
Mark Toufic - 100785011
Ryan Dinh - 100804962

Team Roles:

Ricardo Prato - Artist

- Created character and car models and texturing for said models all within blender
- Helped out with getting imported textures and models into the game.
- Helped out with adding in the shaders used for these characters in the game.
- Helped with PDF explanations

Mark Toufic - Programmer

- Worked on all the OpenGL code for the games functionality
- All the programming stuff basically

Ryan Dinh - Artist

- Created models and textures for the road and railing for our frogger in blender
- Planned out the Slide Deck for the intended scene we wanted for the game in Blender
- Worked on both the PDF explanations and slides

What game did we choose?

The game that we chose was frogger. The playable character of the game will be a frog that must get across the road to get to the other side winning the game. Failure to do so will result in a loss, which will happen if the frog fails to maneuver across the road from the oncoming traffic ahead of him. The player will win the game if he successfully gets the frog to the other side of the road without being hit! The only obstacles will be cars LERPing across the road to make the game difficult.

Result of last 3 digits:

3 + 1 + 2 = 6
Result: Even

Explanation of concepts:

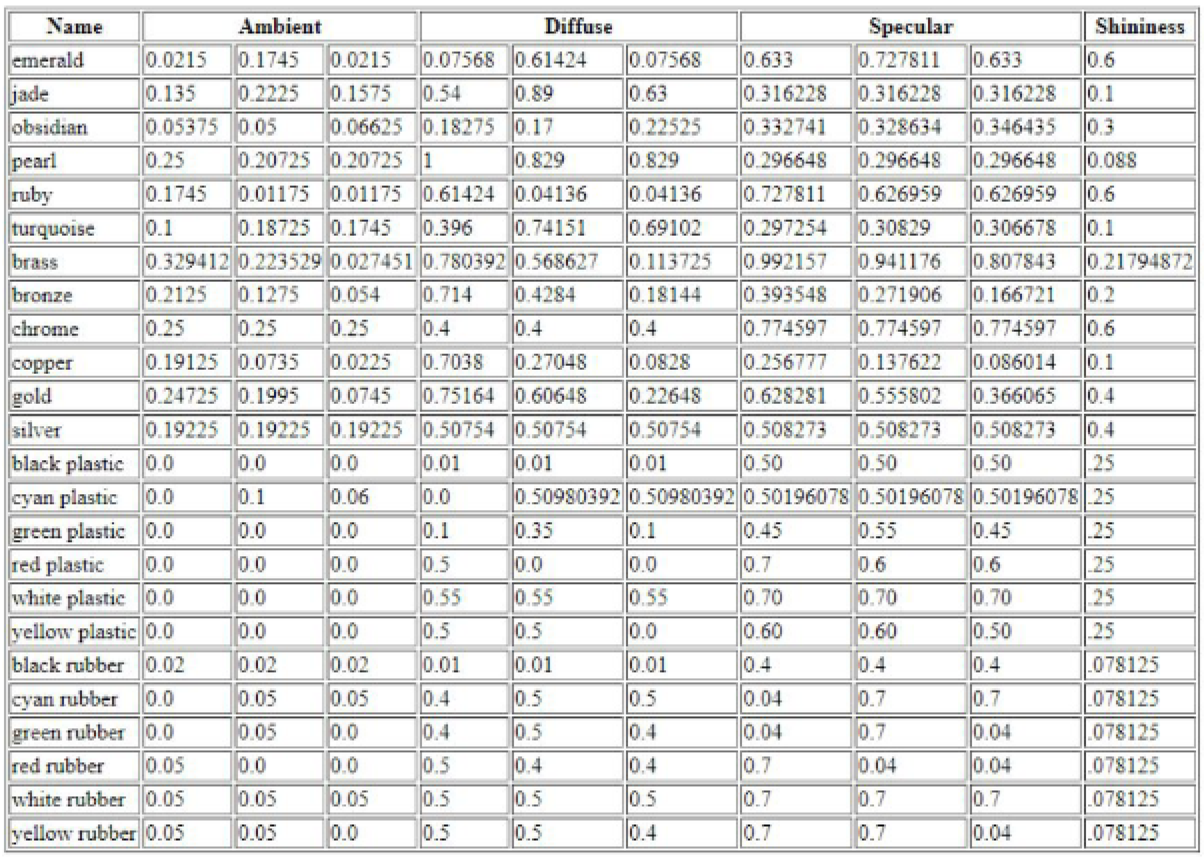
1. The playable character we made for our character was a teeny tiny little cute frog. After creating our character model in a third party software such as blender we then import that model into our game. It first must go through the vertex shader. The vertex shader is able to handle all the vertex data that the models send through. For the sake of this project, not too much had to be done besides making sure that the model maintained its structural integrity and appeared in game as it was modeled.
   The geometry shader can basically create or reduce the amount of geometry. You theoretically can do things such as tessellation with it, but it's not recommended.

The geometry is not at all a necessary thing to implement, but in the context of the frog we could have it create a trail of blocks behind the frog to represent the dust coming off of the frog from his high speed.

Fragment shader writes the colors of a given object as a pixel onto the screen, for our character, it will sample from the texture combined with the lighting models used and to draw the frog right onto the screen so that we can see him!

2. The Phong lighting model allows us to create a metallic feel for the objects within our game through adjusting the Ambient, Diffuse, Specular, and Shininess values in our frag shader. If they're adjusted accordingly will it create a nice replication of a metal object in real life. Referring to fig.1, a material table, we can see an example of how to replicate silver, a metallic material, if the values are adjusted according to the table.

Material Table



Name	Ambient			Diffuse			Specular			Shininess
emerald	0.0215	0.1745	0.0215	0.07568	0.61424	0.07568	0.633	0.727811	0.633	0.6
jade	0.135	0.2225	0.1575	0.54	0.89	0.63	0.316228	0.316228	0.316228	0.1
obsidian	0.05375	0.05	0.06625	0.18275	0.17	0.22525	0.332741	0.328634	0.346435	0.3
pearl	0.25	0.20725	0.20725	1	0.829	0.829	0.296648	0.296648	0.296648	0.088
ruby	0.1745	0.01175	0.01175	0.61424	0.04136	0.04136	0.727811	0.626959	0.626959	0.6
turquoise	0.1	0.18725	0.1745	0.396	0.74151	0.69102	0.297254	0.30829	0.306678	0.1
brass	0.329412	0.223529	0.027451	0.780392	0.568627	0.113725	0.992157	0.941176	0.807843	0.21794872
bronze	0.2125	0.1275	0.054	0.714	0.4284	0.18144	0.393548	0.271906	0.166721	0.2
chrome	0.25	0.25	0.25	0.4	0.4	0.4	0.774597	0.774597	0.774597	0.6
copper	0.19125	0.0735	0.0225	0.7038	0.27048	0.0828	0.256777	0.137622	0.086014	0.1
gold	0.24725	0.1995	0.0745	0.75164	0.60648	0.22648	0.628281	0.555802	0.366065	0.4
silver	0.19225	0.19225	0.19225	0.50754	0.50754	0.50754	0.508273	0.508273	0.508273	0.4
black plastic	0.0	0.0	0.0	0.01	0.01	0.01	0.50	0.50	0.50	.25
cyan plastic	0.0	0.1	0.06	0.0	0.50980392	0.50980392	0.50196078	0.50196078	0.50196078	.25
green plastic	0.0	0.0	0.0	0.1	0.35	0.1	0.45	0.55	0.45	.25
red plastic	0.0	0.0	0.0	0.5	0.0	0.0	0.7	0.6	0.6	.25
white plastic	0.0	0.0	0.0	0.55	0.55	0.55	0.70	0.70	0.70	.25
yellow plastic	0.0	0.0	0.0	0.5	0.5	0.0	0.60	0.60	0.50	.25
black rubber	0.02	0.02	0.02	0.01	0.01	0.01	0.4	0.4	0.4	.078125
cyan rubber	0.0	0.05	0.05	0.4	0.5	0.5	0.04	0.7	0.7	.078125
green rubber	0.0	0.05	0.0	0.4	0.5	0.4	0.04	0.7	0.04	.078125
red rubber	0.05	0.0	0.0	0.5	0.4	0.4	0.7	0.04	0.04	.078125
white rubber	0.05	0.05	0.05	0.5	0.5	0.5	0.7	0.7	0.7	.078125
yellow rubber	0.05	0.05	0.0	0.5	0.5	0.4	0.7	0.7	0.04	.078125

Fig.1
Reference: Lecture 3 Lighting and Shading Slides

3. An approach that we wanted to take to create a winter feel for our game using shaders was a particle effect emitting snow from above. Furthermore, we wanted to add a desaturation shader that allows us to have that nice blur like you would if it were snowing in real life. (Refer to Fig.2) The link provided with the figure explains just how the shader

is done in OpenGL, and perfectly captures what we would want to have for our winter themed frogger.

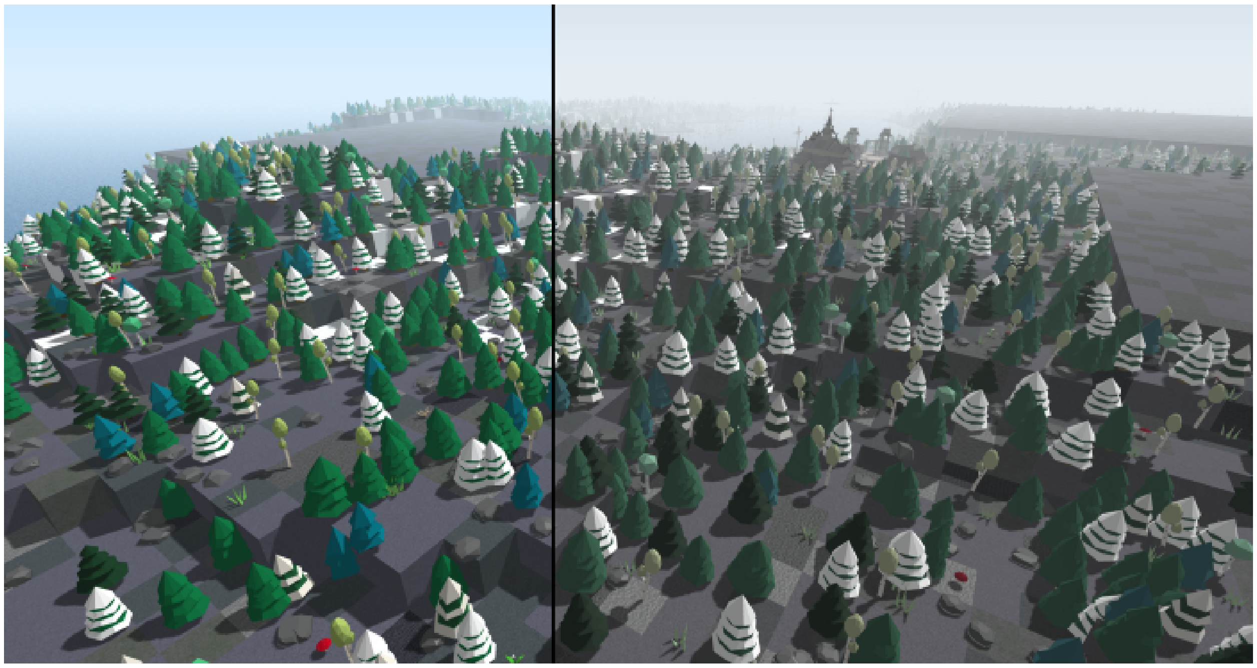


Fig.2 (Right side is Desaturation Shader)
Reference: Behler, C. (2020, July 13). Desaturation Shader in OpenGL - Christian Behler - Medium. Retrieved March 8, 2022, from Medium website: https://pingpoli.medium.com/desaturation-shader-in-opengl-9ef0a59e0f58

4. A dynamic light that gives the effect of changing the scene could be done through as time goes on, the light will change color from a bright white-yellow that represents sunlight to a dark blue to represent the darkness of night. We essentially lerp between these 2 colors to have a day and night cycle!

5. We were going to implement the shader aformentioned in Fig.2, but time was an issue so it was sadly not included. However, if you look into the link you can see that the implementation of the shader itself is fairly simple as you convert color to a grayscale and then mix it into the actual color to create the final render depending on how much desaturation you have.

6. The desaturation shader was chosen as it replicates just what we want for a snow light feel in real life. The mix of the blur and grayscale is perfect.