CITS3003 Project Report

David Charkey (22477478)

Process

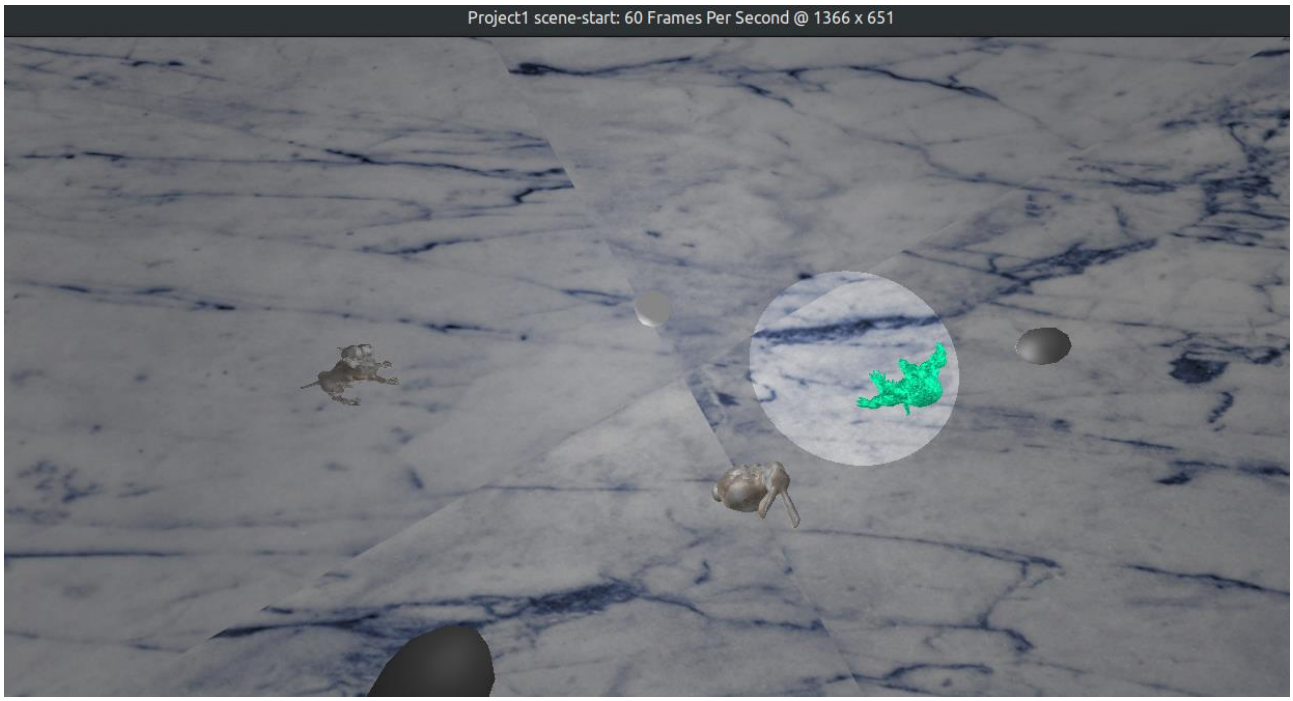
- To build my program, I "added" each part to my project in order, starting with Part A and working progressively to Part J. While working on each part, I documented my findings and changes in the sections below.
- For each change I made to scene-start.cpp and the vertex/fragment shaders, I added a comment nearby, containing my initials (DC).
- I modified *gnatiread.h* to solve the "jumpy" camera. I did this by modifying the *activateTool()* function – I replaced the unused variable "clickPrev" with "prevPos". I posted this fix to the help forum. I also made some other very minor changes to *gnatiread.h* which I commented.

Part A – Camera Rotation

I was successful in implementing this feature.

- The variables *"camRotUpAndOverDeg"* and *"camRotSidewaysDeg"* were being adjusted but not applied to the camera.
- In the display function, I inputted these variables into the *RotateX()* and *RotateY()* functions respectively.
- I concatenated the matrices outputted from these functions with the view distance translation matrix, and used the final result to update the view matrix.

Screenshot for Part A: A scene where the camera is viewing from above

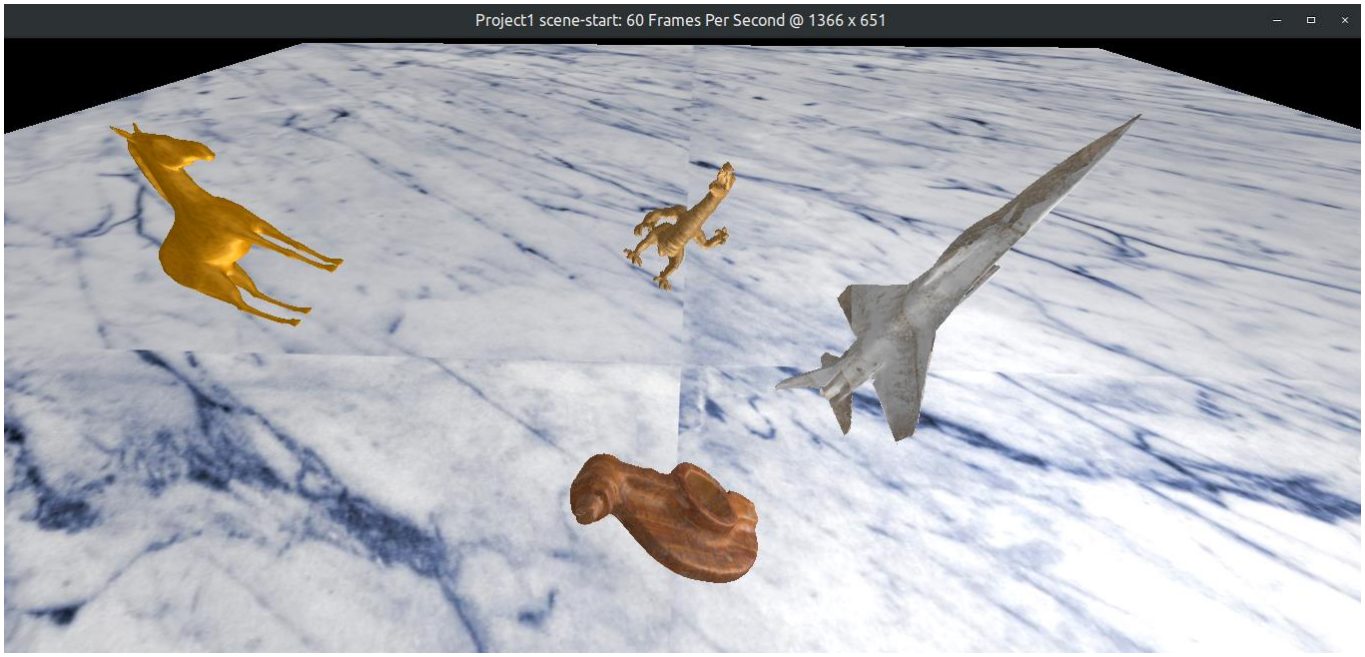


Part B – Object Rotation

This part was successfully integrated into my project.

- In the *drawMesh()* function, the values in the *angles[]* array in the *sceneObj* structure were not being used to transform the scene objects.
- I extracted these angles in the *RotateX/Y/Z()* functions, concatenated the matrices returned, and integrated the resulting matrice into the model matrix.
- As for texture scaling, in the *display()* function I gave the scene object's *texScale* variable to the shaders. I then made a uniform float variable in the fragment shader of the same name, and used it to multiply the texture coordinate given to *texture2D()* (which then affects the assignment to *gl_FragColor*).

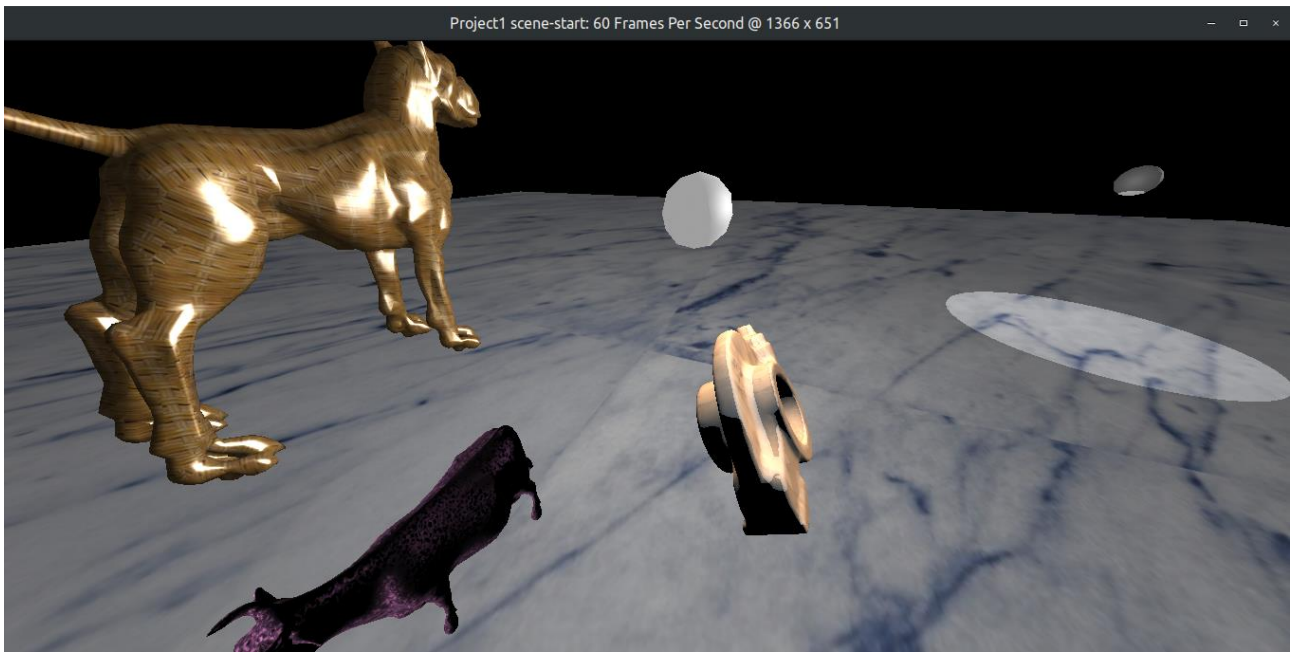
Screenshot for Part B: A scene with rotated objects



Part C – Object Light Components

I was successful in completing this part of the project. Firstly, in *makeMenu()*, I removed "UNIMPLEMENTED" from the menu option "Ambient/Diffuse/Specular/Shine". I also noted that its id was 20. Secondly, in *materialMenu()*, I made a new case (id equals 20), in order to attach an action to this option. I modelled the code for this case on the above case code used to adjust the RGBA values of the scene objects. Following this strategy, I made two callback functions, each modifying two of the four relevant properties (Ambient/Diffuse/Specular/Shine) of the scene objects. I then inputted the functions into the setToolCallbacks() with the default 2x2 matrix modifiers. Finally, I modified the second matrix modifier to amplify the shininess value, enabling it to reach a value of 100. I also added some additional checks to prevent the shininess from going beyond the correct range.

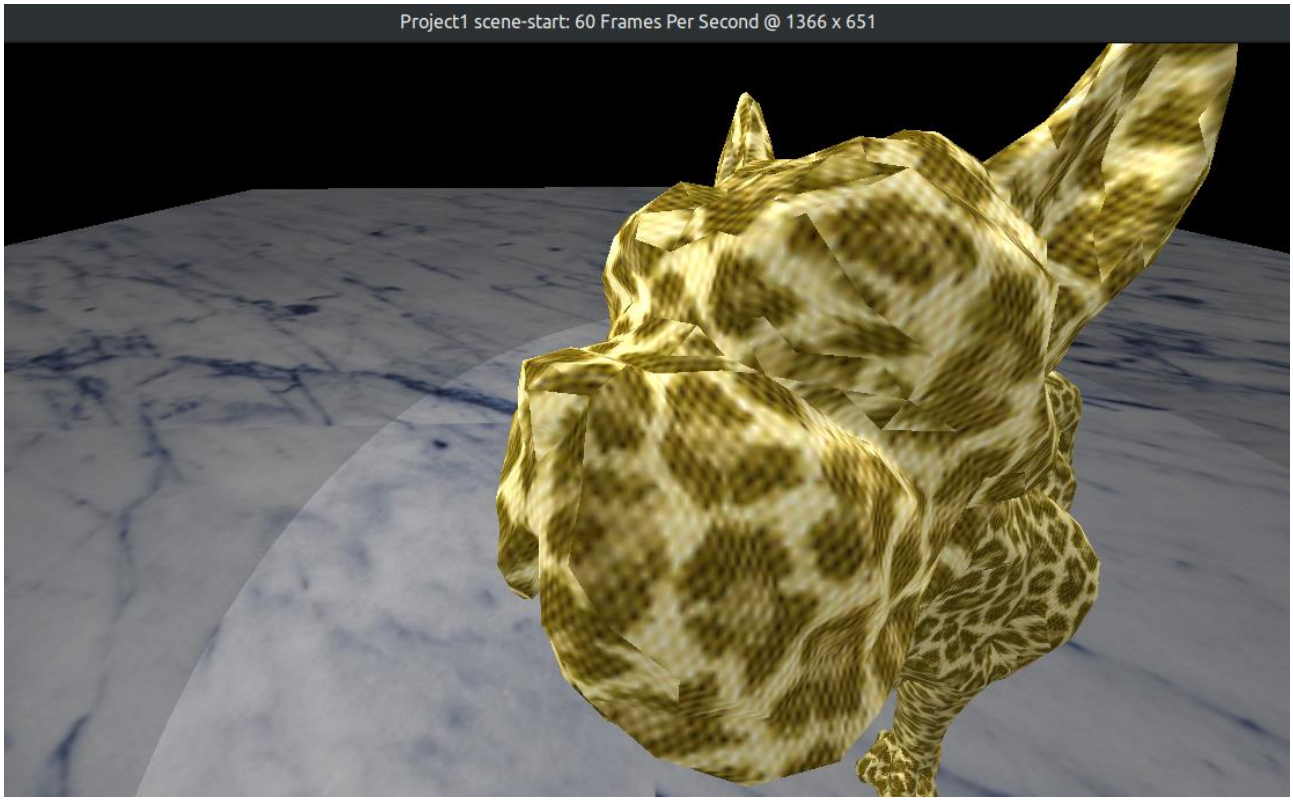
Screenshot for Part C: A scene with objects with modified light components



Part D – Close Up Camera

I was able to successfully implement this feature in my project. I accomplished this by altering the values of the *near* and *far* parameters of the Frustum function that returns the projection matrix. I reduced the *near* value to 0.01 and increased the *far* value to 1000.

Screenshot for Part D: A scene where the camera is very close to a dog object

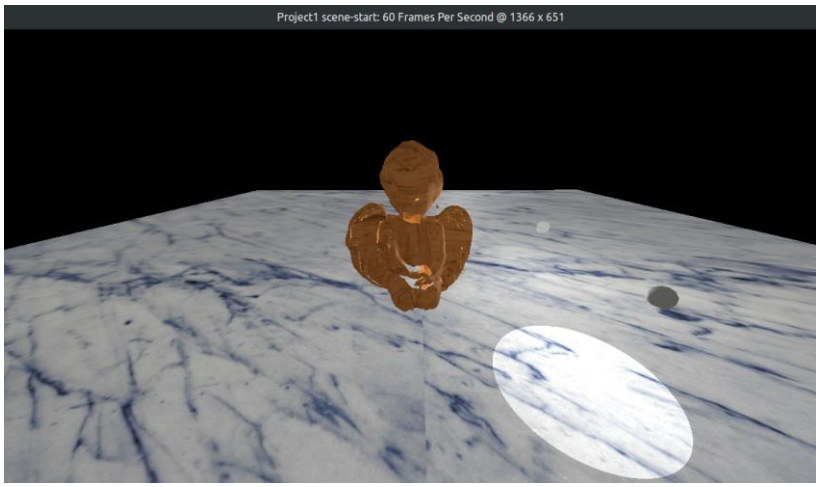


Part E – Window Resize Behaviour

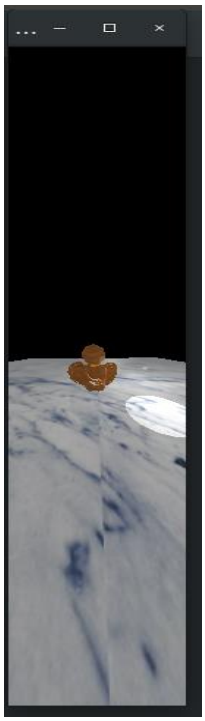
I was successful in incorporating this part in my project. As in Part D, I altered the Frustum function that generates the projection matrix. I made it so that the *left, right, bottom,* and *left* parameters are multiplied by different factors/ratios based on whether width is greater/less than the height. When width is less, *left* and *right* are multiplied by 1, while the *bottom* and *top* are multiplied by height divided by the width. On the other hand, when the height is greater, *bottom* and *top* are multiplied by 1, while *left* and *right* are multiplied by width divided by height.

Screenshots for Part E:

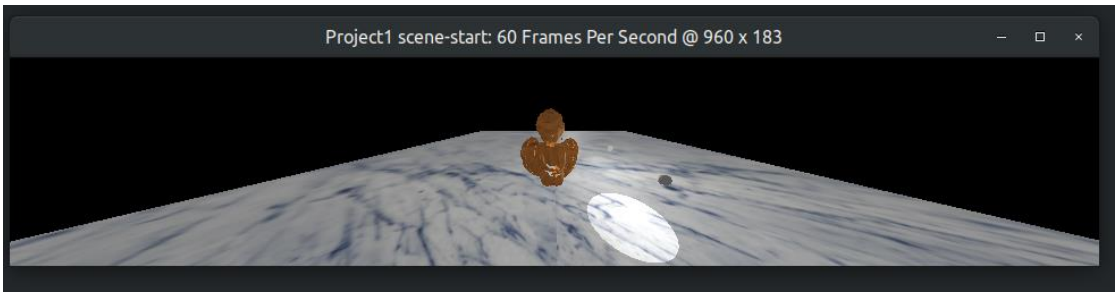
Normal Window for Comparison



Narrow Window:



Wide Window:

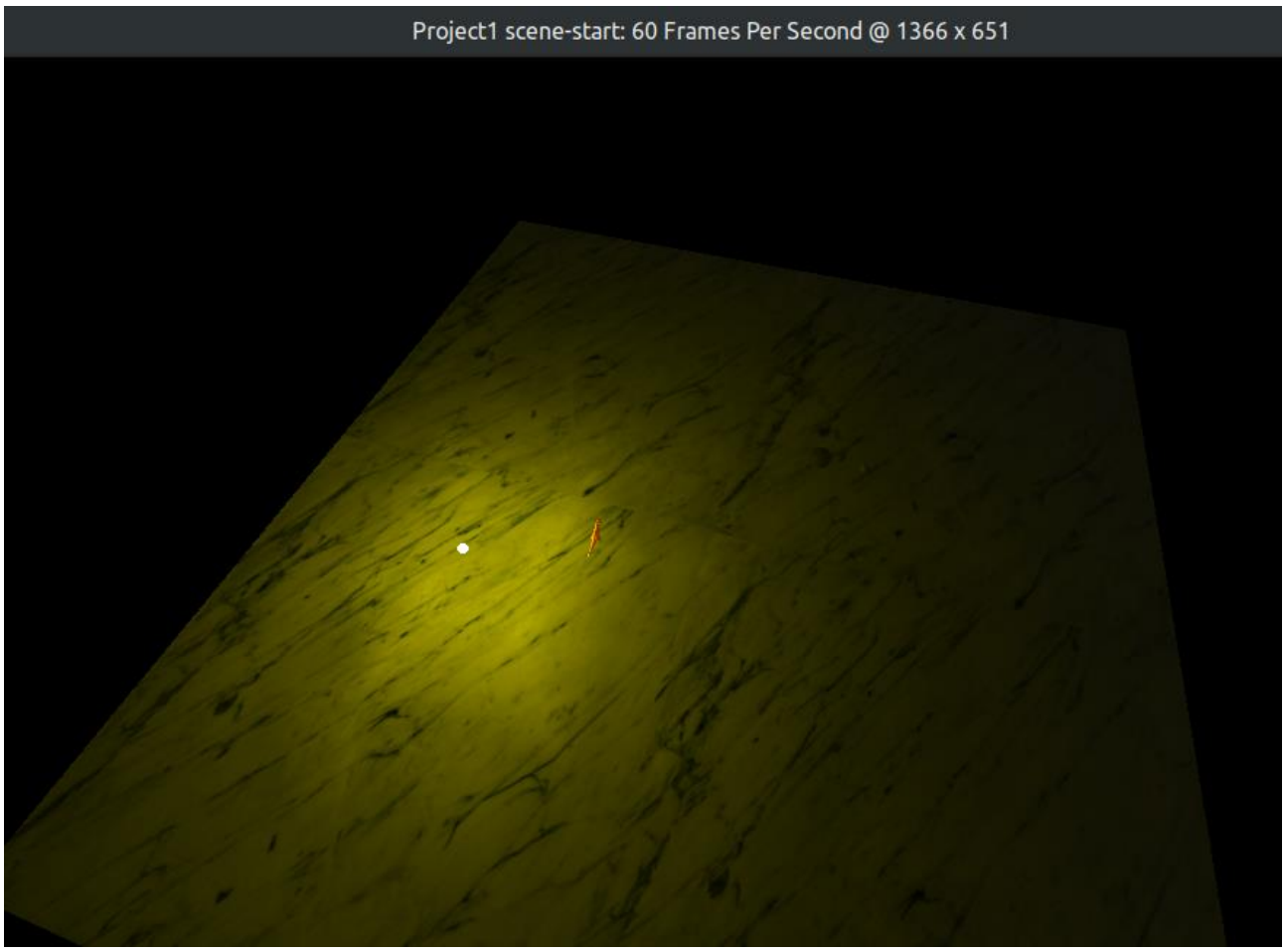


Part F – Per-Vertex Lighting Attenuation

This feature was added successfully to my project. I placed the shaders relevant to this part in the folder "PartFShaders" in my "project" directory. For these shaders to be used by the program, the char variable "*ACTIVE*" in the *init()* function must be equal to *'F'*.

In the vertex shader, I first calculated the distance from the light source to the current vertex by taking the length of the vector between the two. Next, I inputted this distance into a quadratic equation, to produce an attenuation factor constant. I spent some time adjusting the "a", "b" and "c" terms in the quadratic equation to get the right attenuation rate. I finally multiplied the attenuation factor by the ambient, specular, and diffuse light components, so that they reduce in intensity with distance.

Screenshot for Part F: A scene using (rough, blocky) per-vertex lighting attenuation



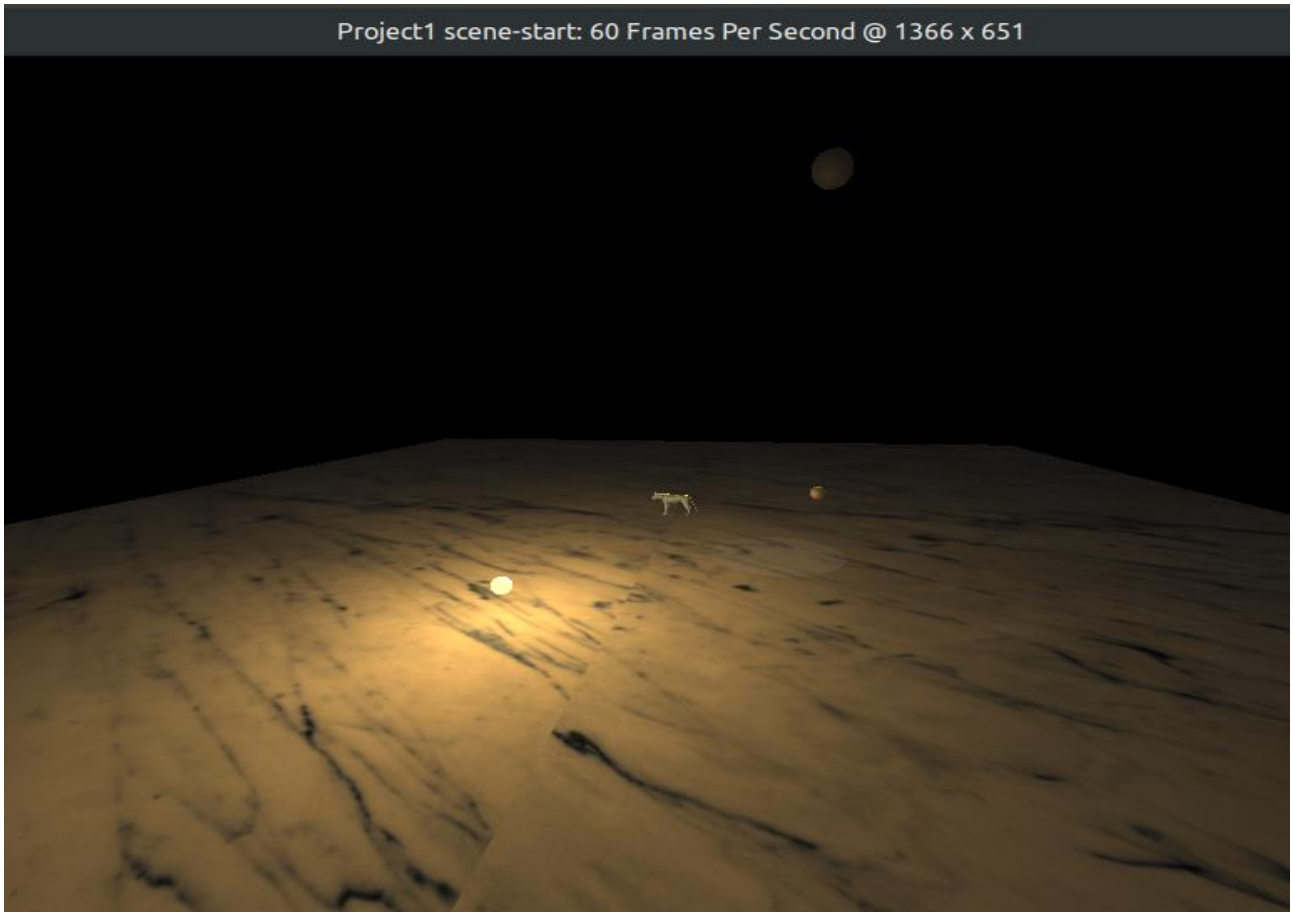
Part G – Per-Fragment Lighting Attenuation

I was successful in implementing light attenuation in the fragment shader. Similar to Part F, the shaders relevant to this part are in the folder "PartGShaders" in my "project" directory. For these shaders to be used by the program, the char variable *"ACTIVE"* in the *init()* function must be equal to *'G'*.

I first modified the vertex shader to pass the values of the vertex position (*vPos*) and normal (*vNorm*) to the fragment shader using the *'varying'* qualifier. This enabled me to move a large portion of code from the vertex shader into the fragment shader. The light equation, light attenuation factor, and colour are now calculated in the fragment shader.

Upon running my code, I noticed that the lighting effect was realistic and smoother compared to that produced by Part F. **Note:** The shaders for Part G was used for the rest of the project, and any subsequent changes (Part H onwards) were applied to them.

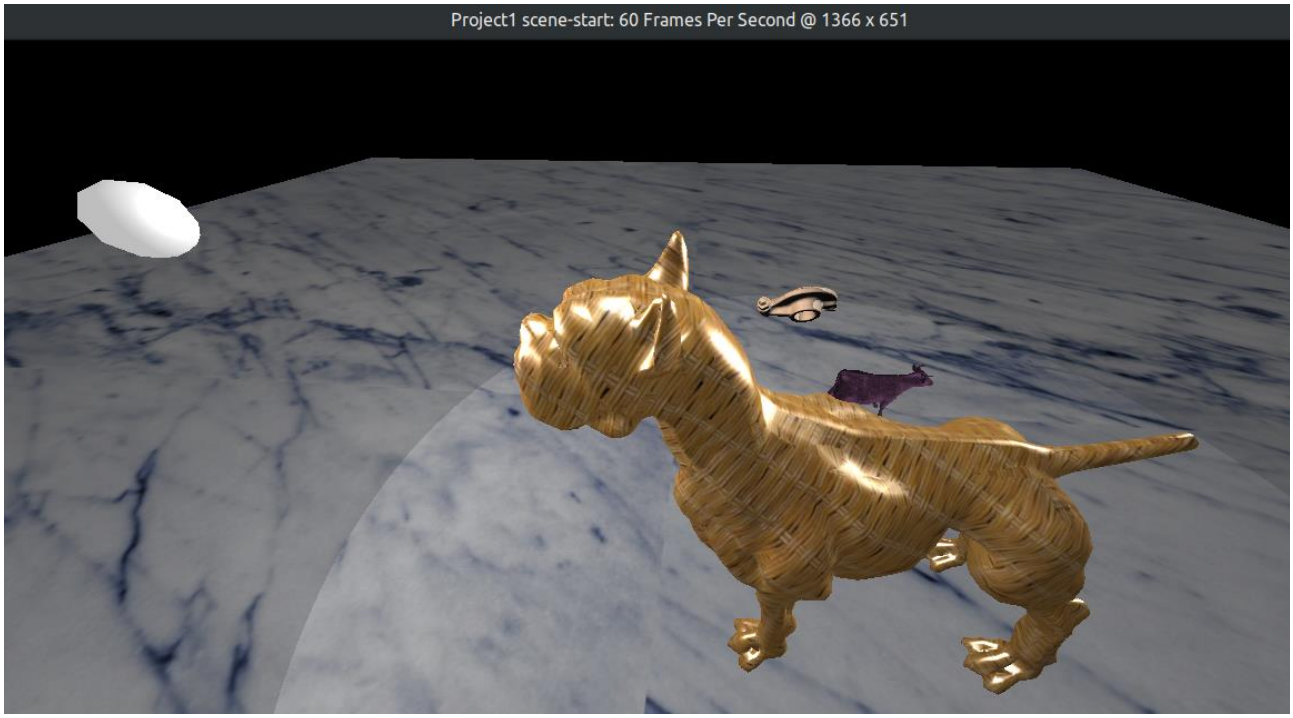
Screenshot for Part G: A scene using (smooth, circular) per-fragment lighting attenuation



Part H – White Specular Highlights

I believe I was successful in adding this feature. In the fragment shader for Part G, I removed the specular component from the calculation of the "*colour*" variable. I made it so that the specular component is calculated separately from the texture and colour, and is instead added at the end. I still applied the light attenuation factor to it.

Screenshot for Part H: A dog object with white specular highlights



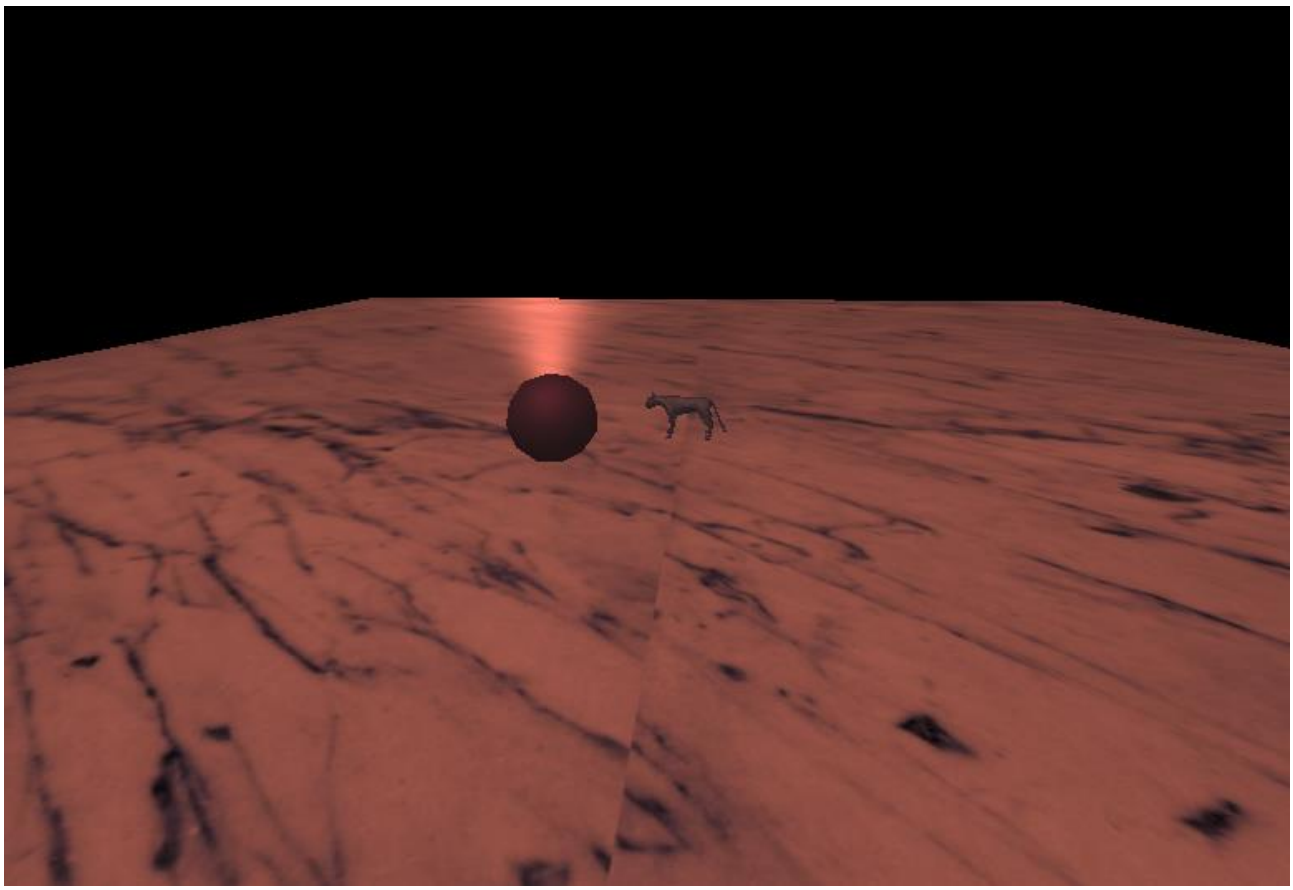
Part I – Directional Light Source

I successfully adding this part to my project. I first modified the *init()* function to add another light source scene object (a larger sphere at a different location).

Next, I updated the display callback to give the position of the second light source to the shaders. Also, in the display callback, I integrated the second light source into the colour calculation. In the *lightMenu()* function, I had to add cases for 80 (Move Light 2) and 81 (R/G/B/All Light 2), and attach call backs to make these menu items work.

In the fragment shader for Part G, I added a LightPosition2 variable to hold the position of the new light source, and used the variable to calculate the associated vector and light components, and integrated them into the final fragment colour. I did not attenuate the light from this new light source.

Screenshot for Part I:



Part J – Object Selection, Deletion and Duplication

(a) and (b)

I was successful in implementing this feature in my project. Firstly, in the *makeMenu()* method, I added three menu entries ("Set Current Object", "Delete Current Object" and "Duplicate Current Object") and linked a method to each (*setObject(), deleteObject()* and *duplicateObject()* respectively).

setObject() prints out a list of objects in the scene and asks the user to input a new scene object index into the terminal. The index provided will change the current object index (*currObject* value). Validity checks are performed and the user is informed accordingly. I have also added a "quick" method of changing the current object index by providing a menu that allows for number values up to 20 to be selected.

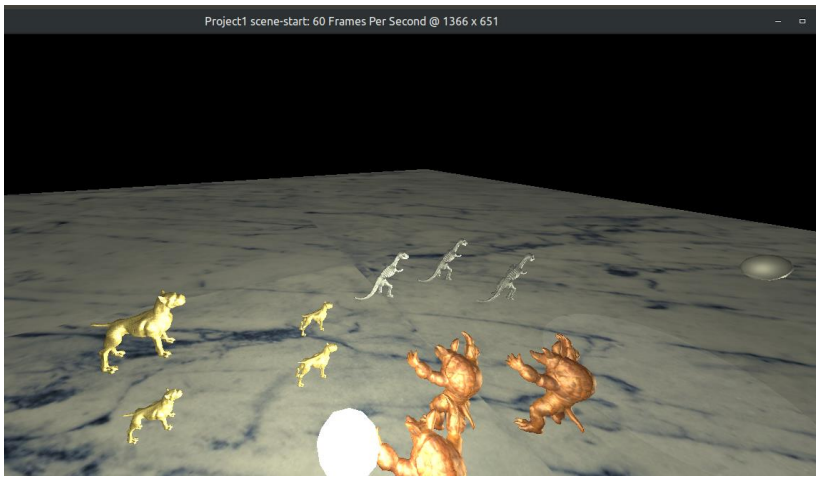
deleteObject():

1. Checks the current object index for validity
2. Deletes the current object and shifts the other scene objects left in the scene array
3. Reduces the total number of objects
4. Resets the current object index to nothing (-1)
5. Updates the display

duplicateObject():

1. Checks the current object index for validity
2. Checks the number of objects against the maximum (stops if will be breached)
3. Saves the current object
4. Slightly modifies its position so it will not appear imposed into the original object
5. Increases the number of objects
6. Sets the current object index as the new object index
7. Places the copied object into the scene array
8. Updates the display

Screenshot for Part J (a) and (b): A scene with duplicated objects



Part J – Adjustable Spotlight

(c)

This feature was successfully integrated into my project. My spotlight's position and direction can be changed.

In the application program:

- In the *init()* function, I added a third light source scene object (small sphere, angled downward)
- In the *display()* callback function, I give the spotlight's position, cone size, and direction via uniform variables to the shaders.
- In the makeMenu() function, I added two entries to the light menu: Move Spotlight (id = 82) and Change Spotlight Direction (id = 83).
- In the *lightMenu()* function, I added cases for the IDs above, setting the appropriate tool call backs for each.

In the fragment shader:

- I added three uniform variables for the spotlight's position, cone size and direction
- I then calculated the vectors and light components of the light source extremely similar to how I did so for Light Source 1. I applied an attenuation factor to the spotlight.
- I added a conditional statement to create the spotlight effect: If the fragment is outside of the spotlight cone, nullify the light components of the spotlight (i.e. set them to zero).

Screenshot for Part J (c):