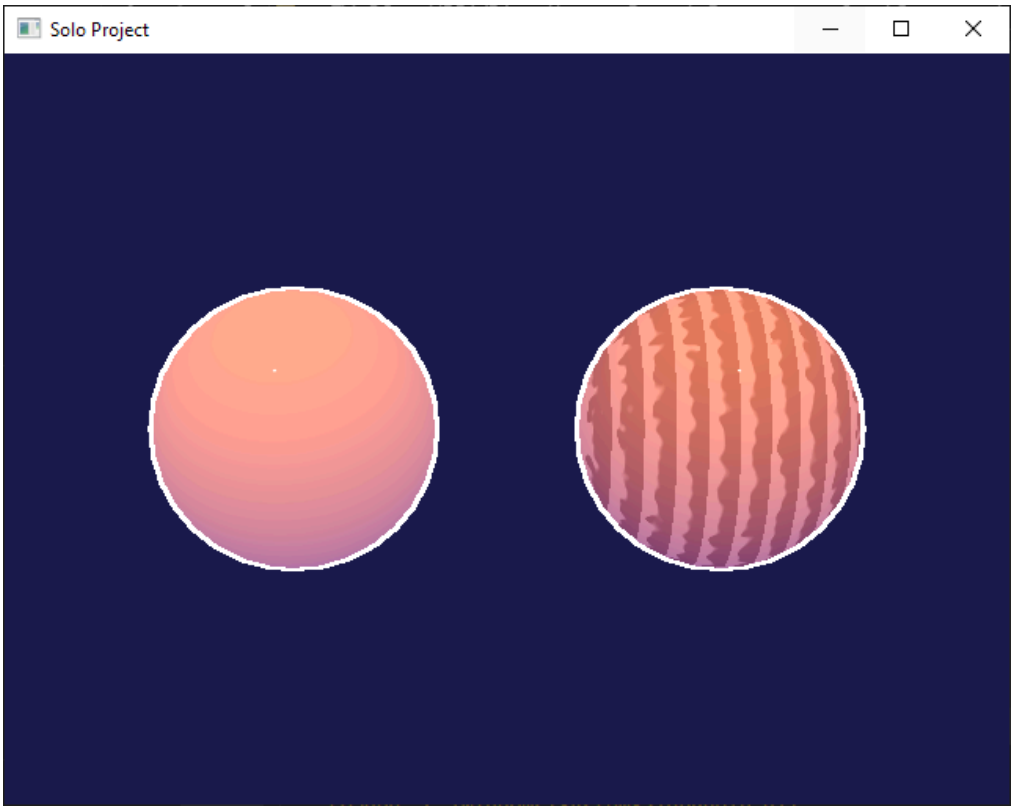
Computer Graphics for Games 2024

Required:
Project Code (step 1) + Report (step 2)
(2-step online submission)

<Illumination Model Examples>

<A realistic and stylised illumination model by itself
and using a texture for comparison>



<Bernardo Damasceno>
<99056>
<Leic-T>
<bernardo.damasceno@tecnico.ulisboa.pt>

Abstract

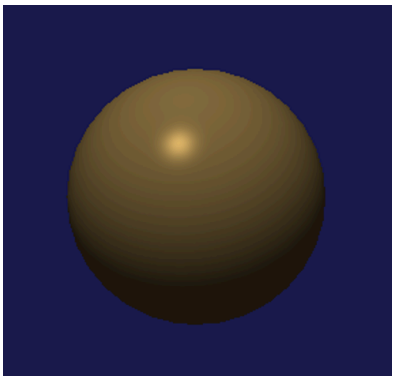
The final idea was to have a little demonstration of two illumination models and how they look applied to an object with some texture. To this end, I used blinn-phong for a more realistic approach, since it has both specular and diffuse light. Gooch was chosen since it visually is very distinct from a realistic light model, since it is used for more technical models, its goal is not to be realistic but clear, allowing for an easy understanding of form.
The addition of a texture is to allow this demonstration to give a better understanding of how these illumination models would look when used in something more developed, giving a better idea of where they should be used.
A rudimentary saving and loading of the scene was also implemented to help get references from this demonstration.

1. Technical Challenges (max 2 pages including illustrations)

1.1. Generic Scenegraph

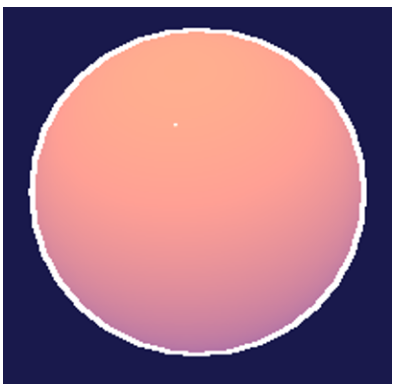
The scene graph has the challenge of connecting the different nodes, and having them update in response to a node higher in their parental hierarchy, how they are created and how they function.

1.2. 'Blinn-Phong'



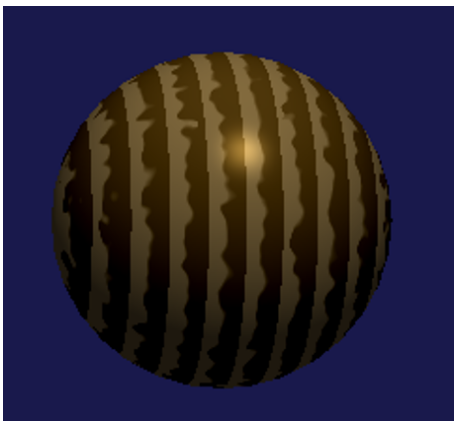
The Blinn-Phong light model requires the calculation of the diffuse and specular area of light. The specular highlight is also required to be calculated and the moves in the specular are in regards to where the camera is looking from. We must then integrate these calculated areas to the color of the object.

1.3. 'Gooch Shading'



Gooch Shading requires much of the same as Blinn-phong, however it has some extra peculiarities. To start it discards the more realistic approach for a more stylised look aimed at visual clarity. The diffuse areas should have a warm colour where the shadow areas should have a cold colour, such as red and blue, and have a contour line much like in cell shading.
As such, there is also a need to blend these colours with the base colour.

1.4. 'Procedural Wood Texture'



To create a procedural wood texture we will need noise, and as such, we will need to create algorithms for it. As explained in the theoretical classes, we will require a ring

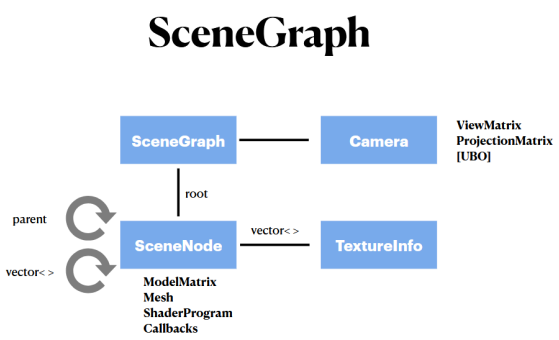
noise algorithm and a perlin noise algorithm, the latter needing to be able to create 3D noise. With the saw wave function as a base, it is then needed to blend these together into the wood texture.

1.5. 'Saving and Loading Scene'

Saving and loading is fairly straightforward, being that firstly it must be chosen what is required to save and load so that the scene is loaded in the same state and what should be saved. Secondly, a way to save and load the scene in a fairly quick and simple manner is required, this should have in consideration what was found to be needed from the previous point.

2. Technical Solutions (max 7 pages including illustrations)

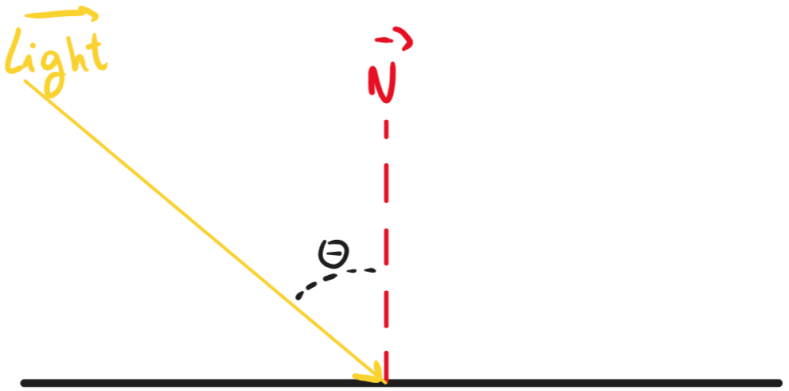
2.1. Generic Scenegraph



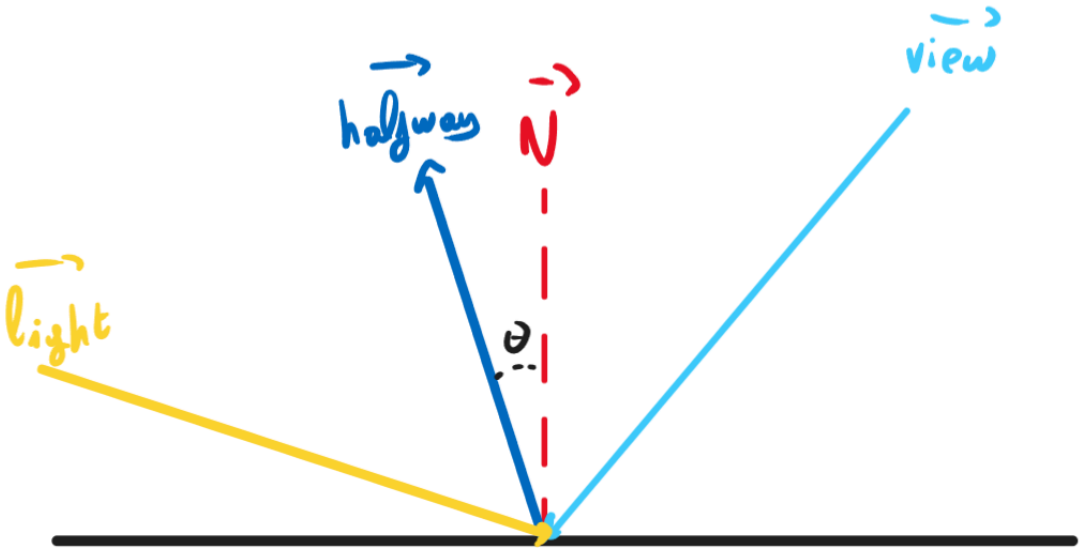
The scene graph was implemented like it was suggested in the 6th handout powerpoint of the theoretical lectures. A scene graph class was created that holds a head node and the camera. This class is called to draw every node and to get data related to the camera.
Scene nodes contain all information pertaining to a mesh(such as model matrix, colour, orientation, shader, etc…) and a vector holding every child node. They also contain their inherent draw method where they draw themselves and then tell their children to draw themselves too.

2.2. 'Blinn-Phong'

To calculate the diffuse light, it must be calculated the dot product between the light source and the normal of the fragment we are rendering, the bigger the resulting angle the less impact the light source has.



For the specular light, the reflected light from the object must be calculated and then used to calculate the dot product with the view direction to get the specular component. However this creates problems with the angle bigger than 90º and as such this uses an halfway vector. This vector is calculated by adding the view vector and the light vector and then normalizing the sum. This way the specular light is the resulting angle between the norm and the halfway vector, the closer the stronger the specular light.



There is also an element of ambient light that can be added, which is calculated simply using the light colour and multiplying it by the light intensity.
To get the color of the fragment, it is simply required to sum the diffuse, specular and ambient light components and then multiply them by the colour of the object.
It should be noted that here we want values between 0 and 1, and as such the dot products are all clamped to not go below zero.

2.3. ‘Gooch Shading’

Gooch Shading is very close to the above mentioned Blinn-Phong lighting model, however here we change the way in which the diffuse component is calculated, since now it requires that a warm and a cold colour be blended. To calculate that we use the following equation:

$$(\frac{1+\hat{l}\cdot\hat{n}}{2})k_{cool} + (\frac{1+\hat{l}\cdot\hat{n}}{2})k_{warm}$$

In this equation we have the diffuse calculation from before, however it is not clamped, and as such, as it goes from -1 to 1, it allows for a full range of colour blending between the two colours.
The *k*'s are calculated with the following equation:

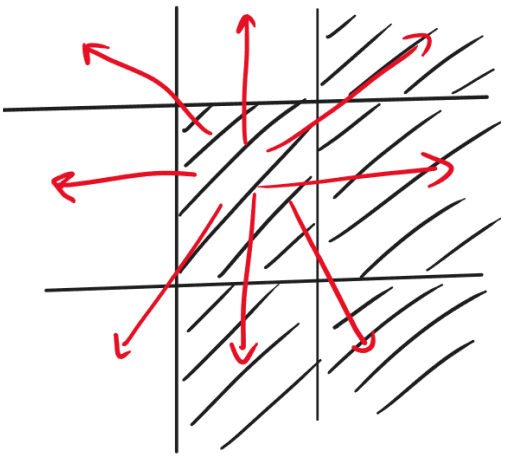
$$k_{cool} = b + \alpha K_d$$
$$k_{warm} = y + \beta K_d$$

On the left of the equations we have a variable that represents the intensity of the warm or cool colour, and on the right we have how strong the original colour of the object is. The alpha and beta values go from between 0 and 1 and the *b* and *y* are, more in essence, the vector of the actual colours that were picked as the cool and warm colours. In terms of shading, this is how gooch shading is done, however to finish it is still needed to add a contour line.

To create the contour line, first the scene is rendered on a frame buffer then the colour buffer and the depth buffer are sent as textures to the main frame buffer that will use a shader using canny edge detector algorithm and render the result in a quad drawn in clip space.
The algorithm itself is pretty simple, however we can't use the depth buffer directly since the value changes are very quick, it uses a very narrow spectrum so we wouldn't be able to get any information from it.
So, firstly, it is needed to linearize the value gotten from the depth buffer, that can be done by multiplying the depth value by 2 and the subtracting 1 so that it's domain is between 0 and 1 and the using the following formula:

$$\frac{2 * near * far}{far + near - z*(far - near)}$$

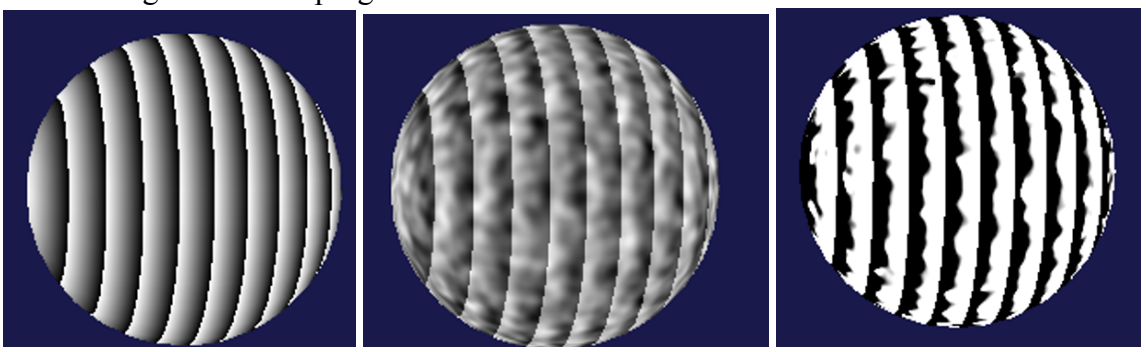
Now it is possible to apply the Canny Edge Detector algorithm. The first step is to look at all the areas around the one being processed and get their depth values(still using linearized depth).



Then we calculate the difference between opposite sided areas to check if there is a big discrepancy, since if there is there is probably an edge there. So if the difference is large enough in any of these checks we set that fragment's colour to our arbitrary line colour and move on onto the next fragment until the render is done.

2.4. ‘Procedural Wood Texture’

The method for procedural wood texture generation, is quite literally, exactly the one mentioned in the theoretical handout pdf.
On the fragment shader we generate a saw wave, then we use sine wave smoothing on the saw wave. After that we create perlin noise and use the mix function to interpolate it with the smoothed saw wave. Next we create ring noise with the formulas given in the pdf and with the smoothed saw wave and noise as the x and y inputs respectively. Then the ring noise is added to half the perlin noise and clamp it between 0 and 1.
To complete it now it only needs to be interpolated using the mix function between a darker and lighter colour(to have shading could be the result of Blinn-Phong for example) and using the resulting float as the progress element.



2.5. ‘Saving and Loading Scene’

The variables required to be saved were simply the camera properties and the state of the spheres, so in a .txt file, on closing the program, the vectors for the eye, center and up vectors of the camera were saved in a line and next up would be the yaw and pitch of the camera. Next up would be if the shading is Blinn-Phong or Gooch, and following that would be if the contour line is on or not.
The loading is also fairly simple, just reads the file and sets the variables according to what was saved.

3. Post-Mortem (max 1 page)

3.1. What went well?

Overhaul, the results are alright, even though at the end I was scrambling to get the last two implementations done(wood pattern and save/load system). I’m quite happy with the Blinn-Phong and Gooch shading, they look fairly nice.

3.2. What did not go so well?

Time management was definitely an issue, I underestimated the time that would be taken from me during the festive season. I was hoping to do a particle system of the GPU, however I started that bit far too late to have a proper thought about it, it was too close to the finish, i really needed a whole week to get it working. The tree texture is also not quite there, I couldn't get it to wiggle like in the theoretical pdf, and quite honestly I’m not very sure why yet. I also wanted the saving/loading system to be more robust and its own class, but it was made last minute and I didn’t quite have the time. If I either had started on the particle system first or given up on it faster, this project would have certainly gone a lot better.

3.3. Lessons learned

What I really learned was that in this project, I really should have started with the hardest system or not added it at all, since after struggling with it I was left with a half-baked idea and had to make do with what I already had, leading to a not so satisfying project to finish. Programming directly on shaders is really quite hard so I should have been more careful with what I picked when deciding what to do.

References

https://cplusplus.com/doc/tutorial/files/
https://www.glfw.org/docs/3.3/window_guide.html#window_events
https://www.glfw.org/docs/3.3/quick_guide.html#quick_key_input
https://learnopengl.com/Advanced-Lighting/Advanced-Lighting
https://learnopengl.com/Advanced-OpenGL/Depth-testing
https://learnopengl.com/Advanced-OpenGL/Advanced-GLSL
https://learnopengl.com/Lighting/Basic-Lighting
https://www.youtube.com/watch?v=welK2U7UkzE
https://www.youtube.com/watch?v=tvKLXbhVBnw
8-lighting-2024-handouts.pdf
10-mapping-2024-handouts (part 2).pdf
6-geometry-management-2024-handouts.pdf

I also used github co-pilot to make changes in the code, such as changing variable names on sections of code or to quickly write repeated code, occasionally some snippets of code generated by co-pilot were more or less kept, but they are however minimal and tweaked. Additionally, I also used chat-GPT to simplify down some documentation and concepts I wasn't understanding.