Visual Computing

Lab Exercises

Samuel Silva

October 25, 2022

4 Let There Be Light

In this hands-on we will be exploring illumination and shading. It is expected that, by completing it you should have an understanding of:

- The different components of illumination models and their impact on a scene
- The different parameters that need to be defined to compute illumination regarding the properties of the light and the objects (materials)
- The impact of different shading methods in how illumination effects are seen
- The basic ideas behind the implementation of each shading method, particularly concerning the tasks of the vertex and fragment shaders

4.1 The Code Base

The code base used for this hands-on has several notable differences when compared with the previous ones.

- Shaders are no longer defined inside the file, as a string. They are read from an external file. This is to avoid having their code mixed with the Python code and since we will be experimenting with several shaders it is more versatile.
- The objects presented in the scene are now read from a file on disk. The reason for this is to have a more complex model to "play" with, with more resolution and which already has the vertex normals computed. These are **fundamental** to compute illumination.
- There is also a class light source that will be used to configure one light source. This hands on will not deal with more than one, sorry! But you may expand it, at the end...

4.2 Boring Shading, a.k.a. per-Nothing

Open and run file `CV_HandsOn_04_Ex1_Start.py`. You should see a green circle at the middle of the screen. The white cube represents the position of a white point light source. You can move it with the arrow keys (look at the terminal for more options). Nothing happens and the circle keeps the same. Some light is needed...

4.3 Flat Shading, a.k.a. per-Primitive

Shading is dealt with at the shaders. So start by opening the file `vertex_-shader_Illumination_`**`pertriangle`**`.glsl` containing a vertex shader prepared for supporting illumination and the corresponding fragment

While running the Python script, you activate this shader by pressing **1**.

shader, `fragment_shader_illumination_`**`pertriangle`**`.glsl`. However, some aspects are still left to do.

- Familiarize yourself with the contents of the shader.
- What is sent to the vertex shader and what goes out to the fragment shader?
- Where is the data sent to the shader, in the Python code? Do you remember what to look for?
- What kind of illumination is currently being implemented for the scene?
- Add the diffuse component to the computed color and run the Python script to observe the results? Move the light source around, with the arrow keys, to check the effect.
- Now, add the specular component and check what happens.
- You can now experiment with just the specular or diffuse components to better understand their effect.
- What is the effect of the `shininess` parameter, in the shader?

It is common to define the color using a line like this: `vColor = max(diffuse , ambient);`. `Remember that the ambient component is there to “simulate” indirect reflections. So, if the other components are larger, they should be used, instead.`

As it is clear, while you can appreciate the effects of applying the **Phong Reflection model**, the shading is quite low quality. Are you able to explain why? Check the vertex and fragment shaders, again. Do you notice the use of `flat`? Take some time to understand its impact. It helps if you remember what kind of data reaches the fragment shader, from the vertex shader. You can think about the first example when you rendered your first multicoloured triangle, a few classes back. You had a colour per vertex and what happened?

The **provoking vertex**... ask what this is!

4.4 Gouraud's Shading, a.k.a. per-Vertex

OK, now let us move up one notch. You can press the number '2' to move into using the next pair of shaders. It should give you a blue screen, for now. Just close the shaders you have open and open those in files `vertex_shader_Illumination_`**`pervertex`**`.glsl` and `fragment_-shader_illumination_`**`pervertex`**`.glsl`.

- So, can you understand the reason why you only see blue?
- Progressively had the different components of the Phong's Reflection Model to the end result and play with each, at each time, in order to better see their contribution.
- You can move the light source around and change between the previous shading (pressing '2') and this new one (pressing '3')
- What is the main difference between the two? What is now happening different, in the shaders? Do you understand why it is called per-vertex shading?

The quality has greatly improved, but some highlights have some mediocre quality. We need to refine our approach.

4.5 Phong's Shading, a.k.a. per-Fragment

If you press '4', you will activate two new shaders that you can find in files `fragment_shader_illumination_`**`perfragment`**`.glsl` and `vertex_shader_-`

`illumination_perfragment.glsl`. There are big changes for these two shaders. You will find the same basic code for implementing the Phong's Reflection Model, but now it is at the fragment shader!

- So, what is happening at the vertex shader and what will be available to the fragment shader?
- Experiment with the different components of the Phong's Reflection Model added to the final result. You can meddle with the strength of the specular component, so exagerate it in order to better check its impact
- Press '2', '3', and '4' to move around the different shading alternatives you have available.
- Explain why this provides different/better results than the previous example.

4.6 Not Everything that Shines is Gold

One aspect that impacts the effect of illumination/shading is the material of the objects. Now, if you go back to the Python script and search for the method where the vertex buffers are initialized, `def set_geometry_buffers(self):`, you can find the line adding the sphere we have been using to experiment with lighting. Now, delete that line and uncomment the five lines above it, which add a few different objects to the scene, and run the script, again. Welcome to the crypt.

- Notice that each object has different material properties and this has different impacts in how light interacts with them.
- Move the light source around and explore its effects. If you check per-vertex shading (by pressing '2'), it is not that bad, when compared with per-fragment shading (pressing '3'). Why is that?
- Go back to the Python code and understand how the material properties for each object are defined. I added a few materials, already: chrome, gold, red plastic, etc. These can be easily found online. You can add a few, yourself, and experiment them on the models.
- You can also play with the light color. It is currently set to white, but you can set its color to something different and see how that impacts shading.

4.7 Exit Music for a Scene

Finally, just for fun, just compose the scene with you favourite materials and lighting and add this to the Python script, just before the game loop:

```
soundtrack = pg.mixer.Sound("soundtrack.mp3")
soundtrack.play()
```