Lab 7

- Computing the unit normal vector for each mesh triangle
- The Phong Illumination Model
- Computing the illumination components – CPU
- Computing the illumination components – GPU – *Per Vertex Shading*
- Computing the illumination components – GPU – *Per Fragment Shading*

1.1 The Phong Illumination Model — Step-by-step implementation

Analyze the incomplete example **WebGL_example_23.html**.

The aim of this example is to use a point light source to illuminate a 3D model, using the Phong illumination model.

The use of the Phong illumination model requires the implementation of additional functionalities, regarding the code of the previous examples.

Identify the main changes regarding the previous examples:

- The models represented in the text files are just described by the **coordinates** of the vertices defining the triangle mesh; there is no longer any color information associated to every vertex – check the various model files.

- To accommodate that, the function that reads a model from a file had to be modified.

- Since the illumination computations are caried out on the CPU, there are no changes in the shaders' code. The color values computed for every mesh vertex are an input to the shaders.

- **Global variables** were added that allow storing the features of the point light source and the features of the material defining the surface of the model.

- Additional functions have been defined in the **maths.js** file which allow operating with 3D points and vectors – check them.

- An (incomplete) additional function has been defined in the **models.js** file, which associates a unit normal vector to each one of the mesh vertices: the normal vector defined by the triangle to which that vertex belongs.

- Note that the new functionalities are not yet producing any effects: it is still required to complete the code of the **drawModel()** function for that to happen.

Questions:

- Which model is represented at first? What is its color?
- Is any illumination being computed at first?

Tasks:

- In the **maths.js** file, analyze the code of the additional functions for operating with points and vectors.
- In the **models.js** file, complete the function that associates to each mesh vertex the unit normal vector of the respective triangle:

computeVertexNormals(coordsArray, normalsArray)

Questions:

- What is the location of the point light source illuminating the scene? What are its features?
- Which color is assigned to the model? What are the material properties associated to the model?

Tasks:

- Complete, step-by-step, taking into account the comments in the code, and checking the successive computations, the Phong illumination model:
    - Ambient illumination
    - Diffuse reflection
    - Specular reflection
- Test the developed code using the available models, which have different levels-of-detail. Analyze the features of the obtained illumination (e.g., using animation and perspective projection). Visualize also the models in ***wireframe*** mode.

Suggestions:

Add functionalities that allow:

- Changing the color of the light source.
- Changing the location and distance to the scene of the light source (directional light source vs. point light source).
- Adjusting the reflection coefficients defining the material associated to the model.

1.2 The Phong Illumination Model — GPU — Per Vertex Shading

Analyze the example **WebGL_example_23_GPU_per_vertex.html**.

As in the previous example, a point light source illuminates the 3D model, using the Phong reflection model.

But, now, the illumination computation for each mesh vertex is carried out on the **GPU**, which required significative modifications in the code of the previous example.

Identify the main changes regarding the previous example:

- For very vertex, the illumination is computed by the ***vertex-shader***, which has a larger set of arguments and a different structure.

- As before, the color values computed for every vertex are input to the *fragment-shader*, which was not modified.

- The functions **drawModel()**, **drawScene()** and **initBuffers()** set the different arguments for the *vertex-shader*.

Questions:

- Which model is represented at first? What is its color?

- Is any illumination being computed at first?

Tasks:

- Analyze the ***vertex-shader*** code and check the differences / similarities regarding the **drawModel()** function of the previous example.

- Analyze the code of the **drawModel()** function, which is now simpler: it computes the global transformation matrix and sets the *vertex-shader* arguments pertaining to the model being represented.

- Analyze the code of the **drawScene()** function: it computes the projection matrix and sets the *vertex-shader* arguments pertaining to the scene being represented.

Suggestions:

Add functionalities that allow:

- Changing the color, the location and the distance to the scene for the light source (directional light source vs. point light source).

- Adjusting the reflection coefficients defining the material associated to the model.

1.3 The Phong Illumination Model — GPU — Per Fragment Shading

Analyze the example **WebGL_example_23_GPU_per_fragment.html**.

As in the previous example, a point light source illuminates the 3D model, using the Phong reflection model.

But, now, the illumination computation is done by the ***fragment-shader***, which required significative modifications in the code of the ***vertex-shader*** and the ***fragment-shader***.

However, regarding the previous example, **no modifications were necessary on th JavaScript code**.

Identify the main changes regarding the previous example:

- For very vertex, the ***vertex-shader*** computes the normal vector, the vector defining the direction to the light source and the vector defining the direction to the viewer.

- Those vectors are passed as an input argument to the ***fragment-shader***, where they will be interpolated.

- The **fragment-shader** uses the illumination model and computes the color that is assigned to each pixel.

Questions:

- Which model is represented at first? How many triangles define the model's surface? What is its color?

- What are the features of the light source? Is it a directional light source or a point light source?

Tasks:

- For the same model and the same light source, **compare** the illumination effects of the previous example and the current example. Which one better reproduces the specular illumination component? Why?

- Analyze the ***vertex-shader*** code and the ***fragment-shader*** code; check the differences / similarities regarding the ***vertex-shader*** code of the previous example.

Suggestions:

Add functionalities that allow:

- Changing the color, the location and the distance to the scene for the light source (directional light source vs. point light source).

- Adjusting the reflection coefficients defining the material associated to the model.