Lab 6

- Modeling using triangle meshes
- Model refining by recursive subdivision
- Exact representation of polyhedral models
- Approximate representation of curved models
- Computing and visualizing the unit normal vector for each triangle of a mesh

1.1 Triangle meshes: mesh refinement by recursive subdivision, using the midpoint of each edge

Analyze the incomplete example **WebGL_example_21.html**.

Identify the main changes regarding the previous example:

- The file **models.js**, allowing the implementation of functions operating on the triangle mesh that defines the surface of each model.

- The implementation of (incomplete) additional functions in the **maths.js** file to operate with 3D points and vectors.

- Additional buttons in the HTML page, to apply operations on the triangle mesh defining each model.

- Note that those new functionalities are not yet producing any effect: it is still required to complete the code of some functions for that to happen.

Tasks:

- In the **maths.js** file, complete the function that allows computing the midpoint of a line segment defined by points **p1** and **p2**:

computeMidPoint(p1, p2)

- Analyze the code, in the file **models.js**, of the two functions which allow refining each triangle of a mesh, using the **midpoint** of each one of its edges.

- Apply the recursive mesh refinement operation to different models and with different recursion depth values. Analyze the features of the resulting models, by visualizing them using the **wireframe** representation.

Question:

- If the initial mesh is defined by ***n*** triangles, and ***k*** is the value set for the recursion depth, how many triangles define the mesh that results from the recursive subdivision?

1.2 Triangle meshes: mesh refinement by recursive subdivision, using the centroid of each triangle

The goal is now to develop an alternative recursive refinement procedure: each triangle will be subdivided into three smaller triangles, which have the **centroid** of the original triangle as a common vertex.

Tasks:

- In the **maths.js** file, complete the function that allows computing the centroid of the triangle defined by the vertices **p1**, **p2** and **p3**:

```
computeCentroid( p1, p2, p3 )
```

- Implement, in the **models.js** file, the two **incomplete functions** which allow computing the refinement of each mesh triangle, using the respective centroid.

- Apply the recursive mesh refinement operation to different models and with different recursion depth values. Analyze the features of the resulting models, by visualizing them using the **wireframe** representation.

Questions:

- If the initial mesh is defined by ***n*** triangles, and ***k*** is the value set for the recursion depth, how many triangles define the mesh that results from the recursive subdivision?
- Given the spatial distribution of the vertices defining the resulting, final mesh, which one of the two recursive subdivision procedures produces, in your opinion, a “better” result?

1.3 Approximate representation of a unit radius sphere

A model representing the unit radius sphere, centered at the origin of the coordinate system, can be constructed as follows:

- Given a convex polyhedral model, centered at the origin of the coordinate system and with an appropriate level of detail.

- Move each one of the mesh vertices to the surface of the unit radius sphere, centered at the origin; each one of the vertices P is displaced along the half-line OP.

Tasks:

- In the **maths.js** file, complete the function that normalizes a given vector **v**:

```
normalize( v )
```

- Implement, in the **models.js** file, the **incomplete function** which allows displacing each one of the mesh vertices to the surface of the unit sphere.

- Apply that procedure to different models and with different levels of detail. Analyze the features of the resulting models, by visualizing them using the **wireframe** representation.

1.4 Visualizing the normal vector to each mesh triangle

Based on the previous one, develop a new example: **WebGL_example_22.html**.

The goal is to compute and visualize, for each mesh triangle, its perpendicular unit vector.

Implement the required functionalities to achieve that objective.

Suggestions:

- In the **maths.js** file, create a function to compute, given two 3D vectors, the result of their vector product (i.e., cross product).

- Store the components of the computed unit vectors in a 1D array, similarly to what is done for the coordinates of the mesh vertices.

- In the **models.js** file, create a function to compute and store the desired unit vectors.

- Develop a method to enable the representation of the computed vectors superimposed on the triangle mesh.