Assignment 09: Smooth Mesh

This assignment is an extension of Assignment 05, where you have to create smooth meshes, whose vertices includes both a position and a normal vector direction. As in A05, you have to create 5 different types of meshes, to encode or approximate the following geometrical objects: a square, a cube, a cylinder, a cone and sphere. Except for the square, all the primitives are supposed to be *2-manyfold objects*: that is, you should not see any holes when you look at them from any direction. Back-face culling is enabled, so the order in which vertices of a triangle are listed matters: please be aware that if you do not see a triangle or an entire object, could be because it has been defined with the vertices in the opposite order. In this case, you should be able to see the missing piece if you look at it from the opposite side, and the solution is simply swapping the order of two indices of the triangle(s).

All the created mesh, should be returned in two arrays:

- **`std::vector<std::array<float,6>> &vertices`** contains the local coordinates of the *vertices* and the corresponding *direction of the normal vector* for the primitive. In this case, each vertex is specified as an array of 6 floating point values:

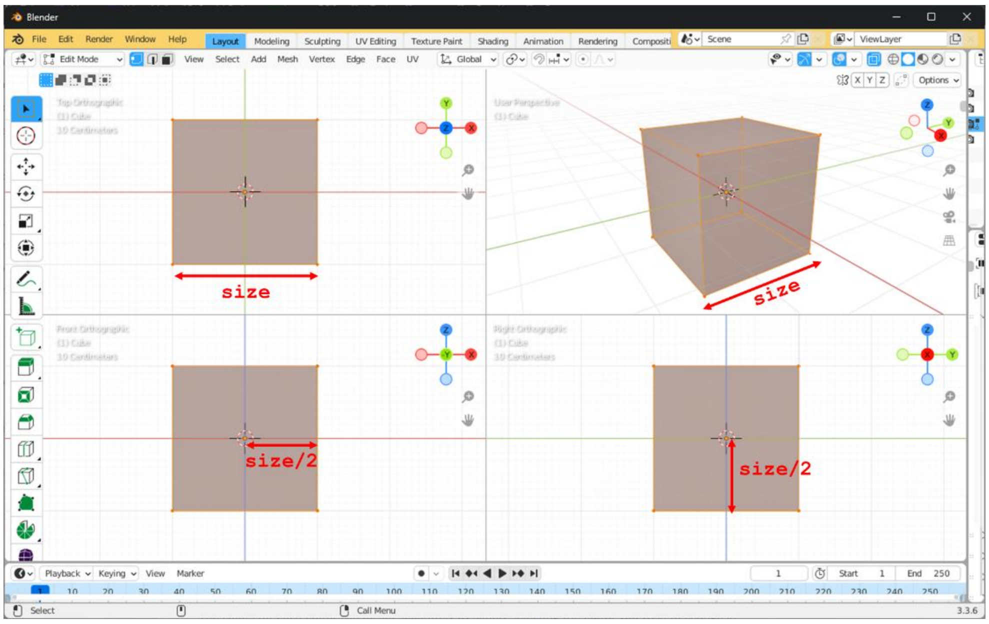
Index [0]	Position x
Index [1]	Position y
Index [2]	Position z
Index [3]	Normal vector direction x
Index [4]	Normal vector direction y
Index [5]	Normal vector direction z

- **`std::vector<uint32_2> &indices`** contains the *integer indices of the triangles*. Its element should be in the *0 to the size of the vertices array minus one range*. Moreover, the size of this array must be a *multiple of three*.

Both arrays are passed as a *reference* (this is the meaning of the **`&`** symbol preceding their name): this mean that you can directly modify them in your procedure, to change them in the caller application. Primitives are encoded as *triangle's list*, for this reason the size of the index array must always be a multiple of three. You are supposed to try to minimize as much as possible size of the vertex array, and create the primitives using the approximations, sizes, positions and orientation specified below, in the description of each of the primitives given below. You should implement primitive creation with four functions contained inside the file **`Mesh.hpp`** :

```
void MakeCube(float size, std::vector<std::array<float,6>> &vertices,
std::vector<uint32_t> &indices)
```

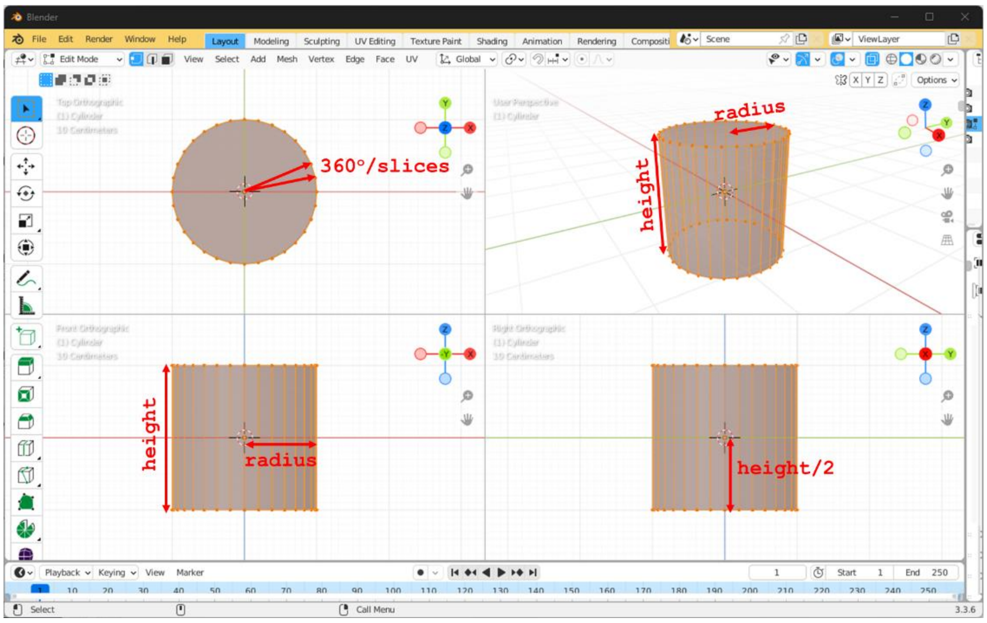
Creates a cube, with the faces perpendicular to the main axis, and centered in the origin. The length of one edge of the cube is given in parameter **`float size`**.



HINT: the procedure already included in the code creates a square. You can use it to create a side of the cube (please remember to change the value of the y component of the vertices, otherwise the result will be in the wrong position).

```
void MakeCylinder(float radius, float height, int slices,
std::vector<std::array<float,6>> &vertices, std::vector<uint32_t> &indices)
```

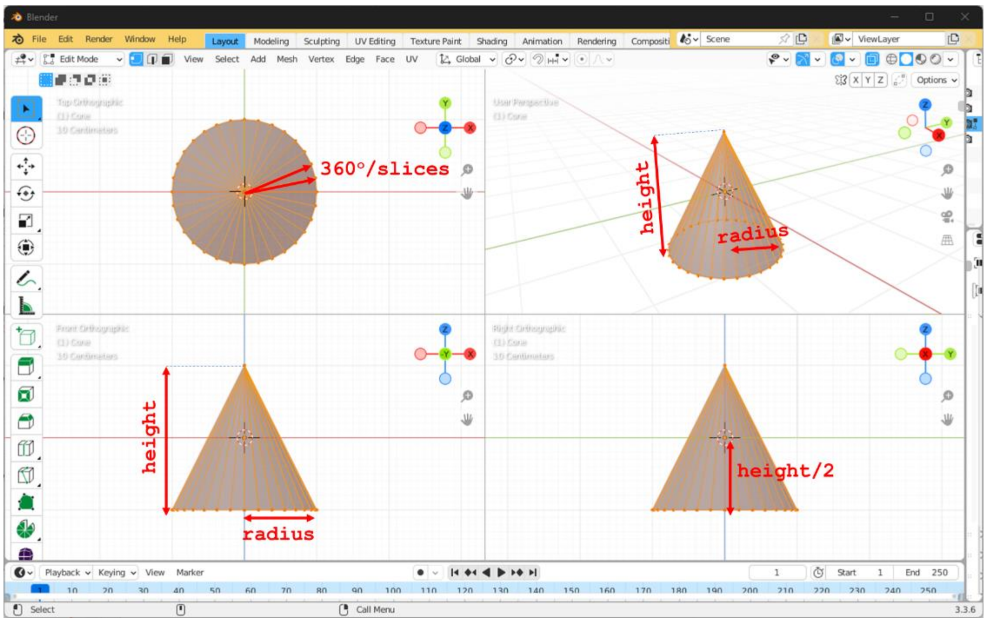
Creates a cylinder, approximated by a prism with a base composed by a regular polygon with **`int slices`** sides. The radius of the cylinder is given in parameter **`float radius`**, and its height is passed in **`float height`**. The cylinder has its centered in the origin, and spans half above and half below the plane that passes through the origin. The top and bottom caps are aligned with *xz-plane* and perpendicular to the *y-axis*.



HINT: the procedure already included in the code creates a rectangle, which rotated around the y-axis would create the required cylinder. You have to remove it, and replace it with the correct one. You should use a ***for-loop***, and you should start from the procedure to create a circle seen during the lesson (also implemented here in the initial version of the sphere, presented below).

```
void MakeCone (float radius, float height, int slices,
std::vector<std::array<float,6>> &vertices, std::vector<uint32_t> &indices)
```

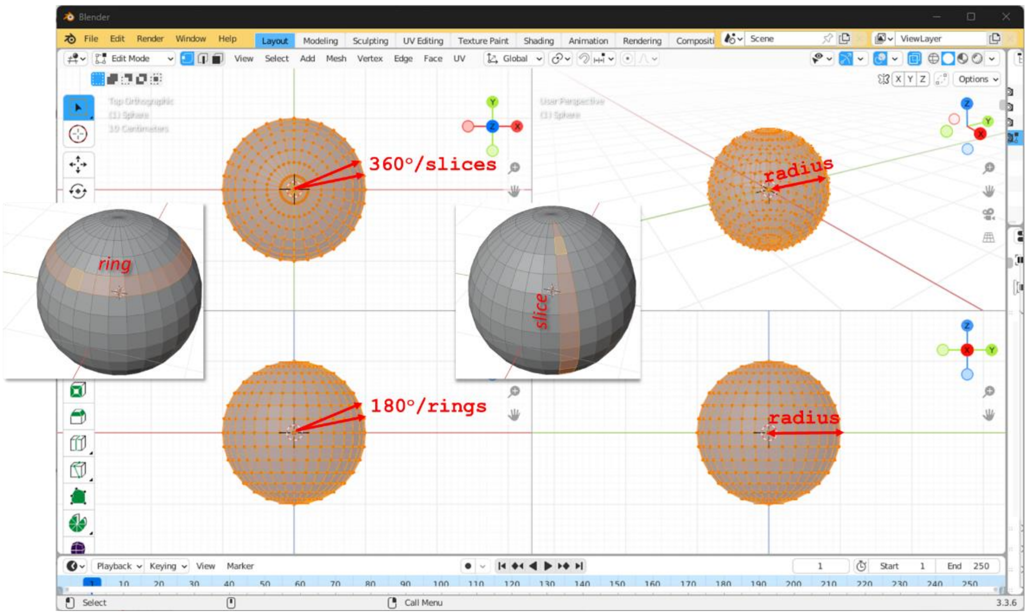
Creates a cone, approximated by a pyramid with a base composed by a regular polygon with **`int slices`** sides. The radius of the cone is given in parameter **`float radius`**, and its height is passed in **`float height`**. The cone has its centered in the origin, and spans half above and half below the plane that passes through the origin. The bottom cap is aligned with *xz-plane* and perpendicular to the *y-axis*.



HINT: the procedure already included in the code creates a triangle, that rotated around its height will generate the required cone. You have to remove it, and replace it with the correct one. You should use a ***for-loop***, and you should start from the procedure to create a circle seen during the lesson (also implemented here in the initial version of the sphere, presented below).

```
void MakeSphere(float radius, int rings, int slices,
std::vector<std::array<float,6>> &vertices, std::vector<uint32_t> &indices)
```

Creates a sphere, approximated by a polyhedron composed by `int rings` bands, each one composed by `int slices` flat faces. The radius of the sphere is given in parameter `float radius`, and center is in the origin.

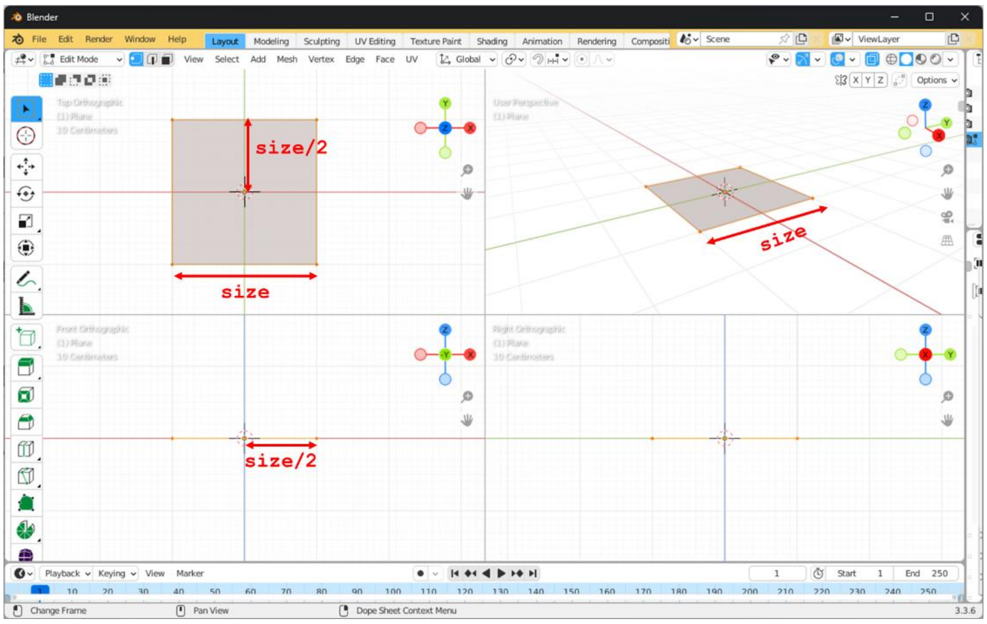


HINT: the procedure already included in the code creates a circle, approximated by with `int slices` sides. To generate a sphere, you should use two nested ***for-loop***, one used to consider the rings, and another compose it with the given number of slices. Please keep in mind that the faces of the rings for the top most and bottom most slices have triangular faces, while the inner rings have quadrilateral faces.

File **`Mesh.hpp`** also includes the following procedure to create a square:

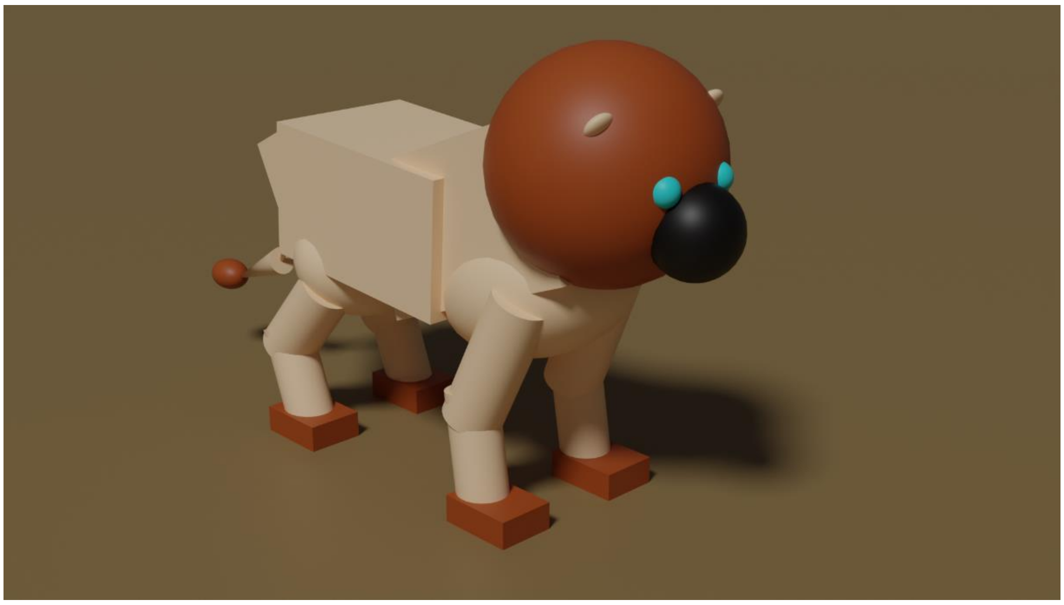
```
void MakeSquare(float size, std::vector<std::array<float,6>> &vertices,
std::vector<uint32_t> &indices)
```

Creates a square, on the *xz-plane*, with the faces aligned with the main axis, and centered in the origin. The length of one side of the square is given in parameter **`float size`**.



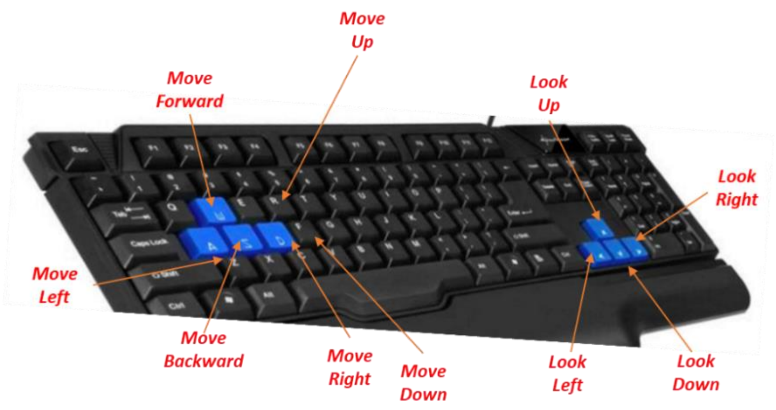
The procedure is already fully implemented as an example, and you are not required to modify it.

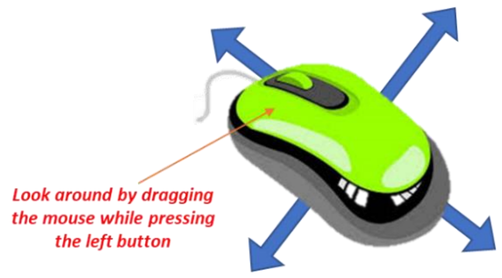
To help you in building your primitives, you can show them in wireframe mode pressing the **X** key on the keyboard. This also disables back-face culling, so that you can see if at least vertices and triangles are placed correctly. Key `N` will show the direction of the normal vector in each vertex, and `M` will replace the diffuse color with an encoding of the normal vector direction in each pixel. Pressing SPACE allows to change between two scenes: the first simply showcasing all the primitives, and the second one, showing the blocky-beast displayed below

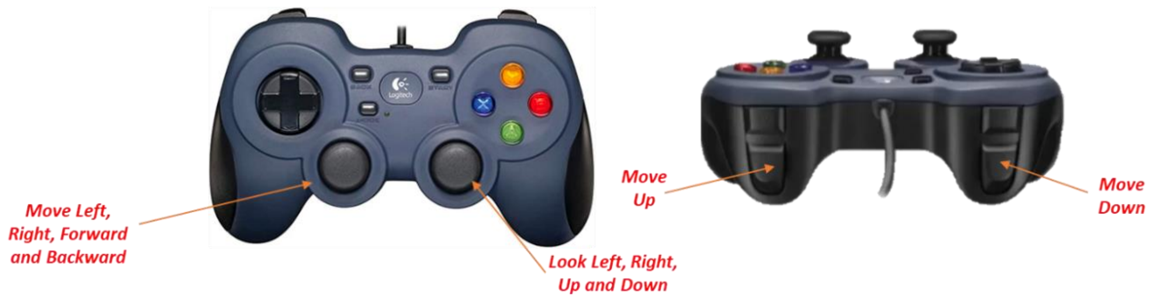


After the two views have been shown once, the third press of the SPACE key will save the screenshots of your results in files **`A09_1.png`** to **`A09_11.png`**. Please check that their content matches your window, as such files will be an important part of the final delivery of this assignment.

You can move the view using either the keyboard, the mouse or a game pad, using the controls below:




Look around by dragging the mouse while pressing the left button


Move Left, Right, Forward and Backward
Look Left, Right, Up and Down
Move Up
Move Down