INF2110 - Introduction à l'Informatique Graphique **M1 Informatique**
TP4 **Fall 2019**
Prof. Caroline Larboulette

3D Transformations

What you need to turn in at the end of the lab session

At the end of the lab session, you should upload a .zip file containing the source code .cpp (+ eventually other classes you have used) of each exercice as well as .pdf shortly describing what you have done for each question with screenshots to illustrate the results you have obtained. Remember the assignment may be graded.

For this lab, we will use the QGLViewer library that provides a trackball and other utilities such as a vector data type. We will work under **Linux** on the lab computers (you may use the system of your choice on your private laptop). You need to have **Qt** installed. Download the library source code from libqglviewer.com, compile the sources and the examples. Take a look at the *simpleViewer* example and start from here.

1. Implement the example "transformation of segments" seen in class

Use the `QGLViewer::Vec` data type to represent points. Have a look at the documentation for available functions/operators on vectors. The technique presented in this example only works for points P_2 having a z value higher than the one of P_1 (why ? what changes otherwise ?). You may use the following points to start with: $P_1(3,4,2)$, $P_2(5,5,3)$ and $P_3(4,6,1)$.

2. Implement the "plane" example seen in class

Create a function to draw a plane (in 3D) and use a `QGLViewer::Vec` to represent the flight direction. Use the keyboard to change $\overrightarrow{DDV}$ and see the plane change orientation.

3. Solar System

We have seen previously that the objects to be drawn were transformed by the `GL_MODELVIEW` matrix. When drawing hierarchies of objects, one may want to draw an object, then apply a transformation like a translation, then draw another object. Once the second object is drawn, we want to get back to the previous transformation matrix without having to apply the inverse transformation of what we just did. To this end, there are two OpenGL commands that allow to push and pop matrices to and from the stack:

```
void glPushMatrix(void);
```
Pushes all matrices in the current stack down one level. The current stack is determined by `glMatrixMode()`. It is by default the `MODELVIEW` matrix when in the `draw()` function of the QGLViewer library.

```
void glPopMatrix(void);
```
Pops the top matrix off the stack, destroying the content of the popped matrix.

First, draw a sphere in `(0,0,0)` to represent the sun. Then, after applying rotations and translations, draw a second sphere in orbit that revolves around itself in a `day`, and around the sun in a `year`. You may then add other planets.
Once the scene is created, you may use the `animate();` function of the QGLViewer library to animate the values of `day` and `year` variables and see your solar system in movement.

```
void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);
```
GLUT function to draw a sphere.