# Tutorial 1

**Note**
This tutorial is based on GLUT tutorial from lighthouse3d.com [1].

**Getting Started**

In this tutorial we are going to build the main function of our application. The main function will perform the required initializations and start the event processing loop.

The first part of our main function code will initialize GLUT and create the window.After GLUT enters the event processing loop it gains control of the application. GLUT will wait for an event to occur and then will check if there is a function to process it.So before GLUT enters its event processing loop we need to tell which functions GLUT must call for each event we care to process.

At this point you might wonder what is an event. An event is something like a key pressed, a mouse move, a window that needs to be painted, or a window that was resized, or ... The list can grow to become quite large. We will start by only dealing with the paint event. Telling GLUT which function to call back when an event occurs is registering the callback function. For each event type GLUT provides a specific function to register the callback function.

The skeleton of our main function is going to start as:

```
int main(int argc, char **argv)
{
        // init GLUT and create window

        // register callbacks

        // enter GLUT event processing cycle
}
```

**Init GLUT and create window**

All the functions in GLUT have the prefix glut, and those which perform some kind of initialization have the prefix glutInit. The first thing you must do is call the function glutInit.

void glutInit(int *argc, char **argv);
Parameters:

- argc - A pointer to the unmodified argc variable from the main function.
- argv - A pointer to the unmodified argv variable from the main function.

After initializing GLUT itself, were going to define our window. First we establish the windows position, i.e. its top left corner. In order to do this we use the function glutInitWindowPosition.

void glutInitWindowPosition(int x, int y);
Parameters:

- x - the number of pixels from the left of the screen. -1 is the default value, meaning it is up to the window manager to decide where the window will appear. If not using the default values then you should pick a positive value, preferably one that will fit in your screen.
- y - the number of pixels from the top of the screen. The comments mentioned for the x parameter also apply in here.

Note that these parameters are only a suggestion to the window manager. The window returned may be in a different position, although if you choose them wisely youll usually get what you want.

Next well choose the window size. In order to do this we use the function glutInitWindowSize.

void glutInitWindowSize(int width, int height);
Parameters:

- width - the width of the window
- height - the height of the window

Again the values for width and height are only a suggestion, so avoid choosing negative values.

Then you should define the display mode using the function glutInitDisplayMode.

void glutInitDisplayMode(unsigned int mode)
Parameters:

- mode - specifies the display mode

The mode parameter is a Boolean combination (OR bit wise) of the possible predefined values in the GLUT library. You use mode to specify the color mode, and the number and type of buffers.

The predefined constants to specify the color model are:

- GLUT_RGBA or GLUT_RGB selects a RGBA window. This is the default color mode.
- GLUT_INDEX selects a color index mode.

The display mode also allows you to select either a single or double buffer window. The predefined constants for this are:

- GLUT_SINGLE - single buffer window
- GLUT_DOUBLE - double buffer window, required to have smooth animation.

There is more, you can specify if you want a window with a particular set of buffers. The most common are:

- GLUT_ACCUM - The accumulation buffer
- GLUT_STENCIL - The stencil buffer
- GLUT_DEPTH - The depth buffer

So, suppose you want a RGB window, with double buffering, and a depth buffer. All you have to do is to OR all the respective constants in order to create the required display mode.

```
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
```

After all the above steps, the window can be created with glutCreateWindow.

int glutCreateWindow(char *title);
Parameters:

- title - sets the window title

The return value of glutCreateWindow is the window identifier. You can use this identifier later within GLUT but this is outside of the scope of this section.

So now here is a little bit of code to perform all the initializations:

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

int main(int argc, char **argv) {

    // init GLUT and create Window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("Title");

    return 1;

}
```

Note the include statements at the beginning of the code. The required include file comes with the GLUT distribution, however their location is different on MacOS systems, hence checking is required if the code is to be cross-platform. **The render function and callback registration**

If you run this code, youll obtain an empty black console window but no OpenGL window. There are two things left to do before we are ready to render something. The first is to tell GLUT what is the function responsible for the rendering. This function will be called by GLUT everytime the window needs to be painted.

Lets create an example function for the rendering. The function presented bellow will clear the color buffer and draw a triangle.

```
void renderScene(void) {

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glBegin(GL_TRIANGLES);
            glVertex3f(0.0f,  0.5f, 0.0f);
            glVertex3f(0.5f, -0.5f, 0.0f);
            glVertex3f(-0.5f,-0.5f, 0.0f);
    glEnd();

    glutSwapBuffers();
}
```

The name of this function is up to you. However, now you must tell GLUT that it should use the function we just wrote for the rendering. This is called registering a callback. GLUT will call the function you choose whenever rendering is required.

So lets tell GLUT that the function renderScene should be used whenever the window requires to be painted. GLUT has a function that takes as a parameter the name of the function to use when redrawing is needed. Note: the function you supply as a parameter is also called the first time the window is created. The syntax is as follows:

void glutDisplayFunc(void (*funcName)(void));
Parameters:

- function pointer to your diplay function

**GLUT event processing loop**

One last thing is missing, telling GLUT that were ready to get in the event processing loop. GLUT provides a function that gets the application in a never ending loop, always waiting for the next event to process. The GLUT function is glutMainLoop, and the syntax is as follows:
void glutMainLoop(void)

**All together now**

The code so far is presented bellow. If you try to run this code youll get a console window, and a few instants after the OpenGL window with a white triangle, hopefully at the desired position and with the desired size. To close the window click on the cross icon. We will learn to handle key presses in the later tutorials.

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

void renderScene(void) {

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glBegin(GL_TRIANGLES);
            glVertex3f(0.0f,  0.5f, 0.0f);
            glVertex3f(0.5f, -0.5f, 0.0f);
            glVertex3f(-0.5f,-0.5f, 0.0f);
    glEnd();

        glutSwapBuffers();
}

int main(int argc, char **argv) {

    // init GLUT and create Window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("Title");

    // register callbacks
    glutDisplayFunc(renderScene);

    // enter GLUT event processing cycle
    glutMainLoop();

    return 1;
}
```

**References :**

1 http://www.lighthouse3d.com/tutorials/glut-tutorial/initialization/
2 https://www.opengl.org/sdk/docs/man2/xhtml/