Lab1-Getting Started

For this course we will be using the Anjuta C/C++ IDE configured on a Ubuntu machine. The lab should have the software installed and configured however, if you would like a similar installation at home, you would have to use a fast internet connection and a lot of commands of the form

sudo apt – get install <package name>

This would automatically use the system proxy settings to locate the latest packages with that name on the internet repository as well as the dependencies and install them automatically. You can also use the Synaptic package manager, which has an GUI. We will be interested in installing OpenCV for image processing and OpenGL for computer graphics. Since it can be quite tedious to code in linux based system, we will appreciate a nice IDE, in this case Anjuta.

Installing and configuring OpenGL/GLUT

In order to be sure that you have C/C++ up and running correctly, write down a small "Hello World" program and execute it. This should help you to identify and correct any potential mal-configurations before you delve into OpenGL.

Just a quick though i think i installed the following packages :

```
The Anjuta IDE        anjuta
Automake utility      automake
SDL Library           libsdl1.2-dev libsdl-image1.2-dev libsdl-mixer1.2-dev libsdl-ttf1.2-dev
GL utility toolkit    glutg3-dev glutg3 glut-doc libglut3-dev libglut3 libglew-dev libglew1
```

Now launch Anjuta, and go to File and then to New Project.
Under project type, select a generic/terminal project.
Now select C++ or both (whichever you are more comfortable with, however we will be needing C++ support), and the target has to be executable.

Now go to the Compiler and Linker Settings and under Libraries, type in SDL and then click Add, then type in glut and click Add. Close the dialog box, and click Yes so that a complete build is performed on the next iteration.
Now paste the following code to test GLUT into the source file:

```
#include "GL/glut.h"
void myDisplayFunction(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_POINTS);
		glVertex2i(100,200);
		glVertex2i(200,200);
		glVertex2i(200,100);
		glVertex2i(100,100);
	glEnd();
	glFlush();
}
void main(int argc, char* argv[])
{
	glutInit(&argc, argv); //Initialize GLUT passing all args to it
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //set the buffering
and color space
	glutInitWindowPosition(100,100); //initial window position
	glutInitWindowSize(640,480);   //initial window size
	glutCreateWindow("CGIP-03, CS&E, UET, LHR"); //create window with
title

//Projection matrix,
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0,640.0,0.0,480.0);
```

```
//Register callbacks
     glutDisplayFunc(myDisplayFunction);

//Start display loop
     glutMainLoop();
}
```

Now paste the following code to test SDL into the source file:

```
#include "SDL/SDL.h"
int main( int argc, char* args[] )
{
     //Start SDL
     SDL_Init( SDL_INIT_EVERYTHING );
     //Quit SDL
     SDL_Quit();
     return 0;
}
```

What this code means will be discussed later, however we are concerned with whether the settings we made are okay or not. Now save the file, compile the project (F9), build (F11) the project and execute the project (F3) to be sure there are no errors.

Understanding the OpenGL/GLUT application

Now we will discuss the first of our program, the one concerned with OpenGL and GLUT. The following steps describe the program.

1) Our basic application began with the inclusion of glut.h header file.
2) The advantage of using OpenGL is that there is no need for complex OS based programming, and all the development can be done using standard C/C++ constructs. One essential element of OpenGL, atop which most of the interactivity is based, is a callback function:

   A call back function is a function which is not explicitly called; instead it is registered with some source of events (e.g. mouse click). Whenever such an event occurs, it would call the registered function, hence callback.

3) First, we will initialize our environment and OpenGL in the main function. For now, this is done by using the following statements.

```
glutInit(&argc, argv); //Initialize GLUT passing all args to it
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); //set buffering and color space
glutInitWindowPosition(100,100); //initial window position
glutInitWindowSize(640,480);   //initial window size
glutCreateWindow("CGIP-03, CS&E, UET, LHR"); //create window with title
```

4) Now we need a mapping of the OpenGL world onto our screen (we will discuss this in theory class). For this we need a matrix, and the projection would be orthographic (just copy these lines for now).

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,640.0,0.0,480.0);
```

5) Now we will be registering our drawing function, which will be responsible for drawing a scene. This is done by means of the following statement:

```
glutDisplayFunc(myDisplayFunction);
```

6) Finally we would implement our drawing function. It can be anything but for now we will just draw some points (4 to be exact)

```
void myDisplayFunction(void){
glClear(GL_COLOR_BUFFER_BIT); //clear everything
```

```
glBegin(GL_POINTS); //drawing points
     glVertex2i(100,200); //points coordinate
     glVertex2i(200,200);
     glVertex2i(200,100);
     glVertex2i(100,100);
glEnd(); //end drawing
glFlush();} //flush data
```

7) And to tell OpenGL that we have prepared everything, so it can now execute the fetch – decode – execute cycle, we will use the following statement inside the main function:

```
glutMainLoop();
```

8) Hence our complete application is described. If the above does not make much sense to you, try to experiment by removing or changing parts of the program to see what happens. Also look up the OpenGL/GLUT documentation as well as go through the text (F.S.Hill's textbook contains a good introduction to this).

Installing and Configuring OpenCV

In the previous section, we configured Anjuta and the OpenGL library for programming. In this manual, we will be looking to configure OpenCV on a Ubuntu machine with Anjuta (similar configuration to the one in the previous section). Configuration of OpenCV is a bit more complicated than OpenGL, since the packages are not directly installable.

First of all you need to get the libraries as below.

- libcv-dev
- sudo apt-get install libcv-dev
- GTK+ 2.x or higher including headers
- pkgconfig
- libpng, zlib, libjpeg, libtiff, libjasper with development files.

- libavcodec, etc. from ffmpeg 0.4.9-pre1 or later + headers.

This should install all the required packages as well. Once you have done that, proceed to download the OpenCV tarball from sourceforge website. This should be a file like opencv1.0.0.tar.gz Once you have downloaded this file, extract the archive to any appropriate place. Now use the terminal to enter the directory and enter the following commands:

- ./configure
- make
- sudo make install
- add /usr/local/lib to /etc/ld.so.conf (and run ldconfig after)
- sudo ldconfig

By now, you should have installed the OpenCV library. In order to check the installation, you can run the following two tests:

- make check

which implicitly runs the test scripts for cv and cxcore, or alternatively you can move to the samples directory (/usr/local/share/opencv/samples) and compile one of the samples for instance using the following command:

- g++ `pkg-config --cflags opencv` -o morphology morphology.c `pkg-config --libs opencv`

Setting up the library paths in Anjuta is also very similar, all you have to do is add the libraries “cv”, “cxcore”, “highgui” in the project settgings for the projects you intend to use OpenCV in. If you want to test the application, all you have to do is to create a new project (terminal application) and paste the following code :

```
#include "opencv/cv.h"            /* required to use OpenCV */
#include "opencv/highgui.h"       /* required to use OpenCV's highgui */
int main(int argc, char *argv[])
{
     IplImage* image=0; /* pointer to an image */
     if(argv[1] != 0)
     {
          image = cvLoadImage(argv[1], 1); // 1 for color
```

```
            printf("File Opened ... %s\n\r", argv[1]);
      }
      if(image != 0)
      {
            cvNamedWindow("Display", CV_WINDOW_AUTOSIZE); // create a window
            cvShowImage("Display", image); // show image in window
            cvWaitKey(0); // wait until user hits a key
            cvDestroyWindow("Display");
      }
      return 0;
}
```

Now, as we did earlier, compile the application, build it and before executing, go to “Set Program parameters” in the build menu and enter a valid image file name which is present in the projects output directory and then execute it. Alternatively you can use the terminal to go to the output directory and execute the program with a command line argument. This should display a dialog box containing the image.

BEST OF LUCK...