1 (Practice 1) Introduction to OpenGL

This first practical is intended to get you used to OpenGL command. It is mostly a copy/paste work. Try to do it smartly by tweaking and messing around with parameters, don't hesitate to google the function's name you'll get used to the OpenGL documentation and you'll understand more about these functions.

First download the file 'materials.zip' associated to this practice. Extract everything inside a folder and try to compile it. We use Cmake which will generate automatically a makefile. The file CmakeList.txt defines the list of source files to be compiled and the libraries to be linked.
In your command line at the project directory location type:

- '**cmake .**' this will generate the makefile.
- '**make**' to effectively compile your project.

Note: You need to call cmake only the first time, after that if you change the code just call make to compile again.

If your are using your own system it is likely you'll have to install some package such as glut, glew, cmake and maybe some others.

For this practice you'll only have to modify the file 'main.c'. Throughout the subject there will be questions, try to answer them directly it'll save you some time as you will have to do a report for each practice. It is not mandatory to answer by yourself to the various questions. There are here to help you focus/understand what you are doing. Don't hesitate to ask your teacher if you're stuck.

1.1 Analyzing main.c

As you edit 'main.c' you can see there is not much. The majority of the code is related to the GLUT library ('glut_xxxx()' functions).

1) What is the purpose of the calls 'glutKeyboardFunc()' and 'glutDisplayFunc()'?
2) What is happening when calling the function 'glutMainLoop()'?
3) In what circumstance the function 'key(...)' is called? What does it do? Same questions for the function 'display()'

1.2 Initialization

Run the program. You'll see nothing which is normal because the display doesn't draw anything, worse we don't even clear the screen buffer.

- Add the instructions to clean the window with a given color.

You can use void glClearColor(float red, float green, float blue, float alpha); which sets the color used to clear the screen buffer. Values range are between [0 1] alpha is for transparency you can set it to '0.f' for no transparency.

The function glClear(GL_COLOR_BUFFER_BIT); effectively clear the screen buffer with the color defined with the last call to 'glClearColor()'

The best is to use these functions at the beginning of 'display()' so that each time we draw something we clear the previous drawings.

• Display a teapot with: glutSolidTeapot(float size); (size is the maximum length from the center of the teapot, chose 1.f for instance)

1) What do you see? Why?

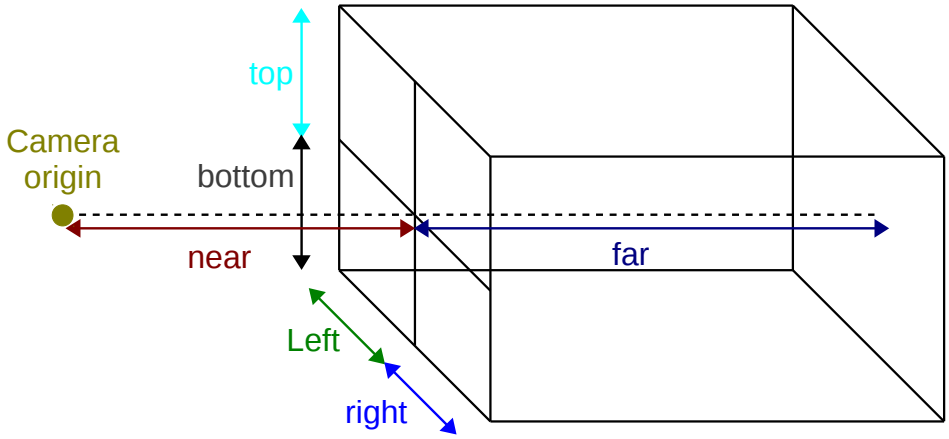
1.3 Setup an orthographic projection

So far your teapot doesn't look very good:

Which is normal since the camera characteristics has not be specified. Also lighting is disabled. Setup the camera characteristics is done through OpenGL matrices. First try to find an orthographic projection to draw the entire teapot:

As described in the course you'll have to do in the display() function:

• switch to the projection matrix mode with glMatrixMode (GL_PROJECTION);
• initialize it with glLoadIdentity();
1) What are the coefficients of the projection matrix after this call?
• Use void glOrtho(float left, float right, float bottom, float top, float near, float far); to setup the viewing volume. See the figure below to understand the parameters:

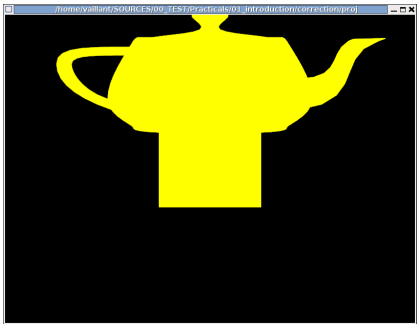


2) Give the coefficients of a projection matrix which project on the 'xy' plane.

1.4 Translation

To do transformations such as translation/rotation/scaling you need to switch the matrix mode to the modelview matrix. This matrix is in charge of moving the objects.

- Add glMatrixMode(GL_MODELVIEW); to swith the matrix mode
- Initialize it with thanks to glLoadIdentity();
- Draw a cube of size one with glutSolidCube(size);
- Draw in yellow the teapot and the cube with glColor3f(float red, float green, float blue); (parameters ranges are between [0 1])
- Translate the teapot as to lie on top of the cube with glTranslate3f(float x, float y, float z);



1.5 Setup an perspective projection

Orthographic projection is not very realistic, lets try perspective by replacing the call to glOrtho() to this function:

```
gluPerspective(GLdouble aperture,     /* Aperture in degrees          */
               GLdouble aspect_ratio, /* window ratio == width/height */
               GLdouble near,         /* near clipping plane          */
               GLdouble far           /* far clipping plane           */
               );
```

1) Sketch the viewing volume (called frustum) with the different parameters (use your course if you need to)

By default the camera is centered at the origin of the world, if you've followed the instructions correctly your objects are also centered at the origin. This means you won't see anything just by using gluPerspective() because the camera is inside your objects, at most you'll see plain yellow.

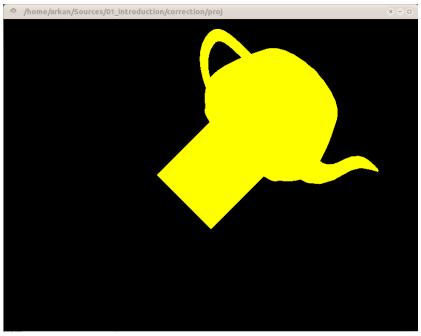
- You must positioned the camera outside the objects and look at them. This is done with:

```
gluLookAt(GLdouble eyeX    , GLdouble eyeY    , GLdouble eyeZ,    /*Camera origin       */
          GLdouble centerX, GLdouble centerY, GLdouble centerZ,/*Point aimed camera  */
          GLdouble upX     , GLdouble upY     , GLdouble upZ     /*y axis of the camera*/
);
```

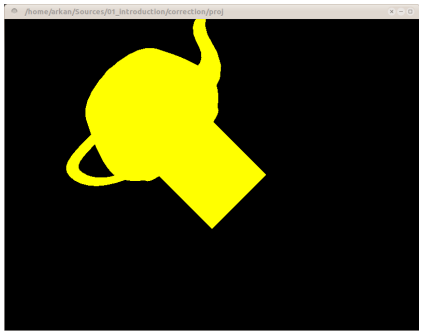
You must use gluLookAt() in the modelview matrix mode. Also note the up vector and the direction (center-eye) must be orthogonal between each other.

This is what is expected if you go toward the -z axis and look at the origin (different values are shown for the up vector):

Up = (0, 1, 0)



Up = (1, 1, 0)



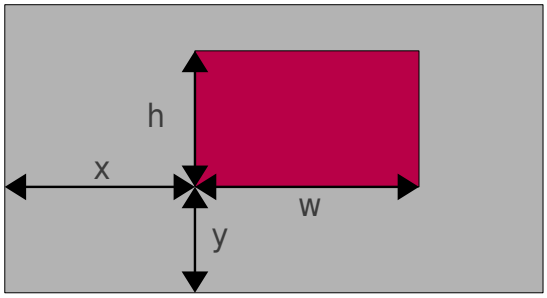
Up = (-1, 1, 0)

- Add a shortcut to 'o' in the key() function in order to switch between orthogonal projection and perspective. Don't forget to tell glut to refresh the screen buffer with glutPostRedisplay() at the end of the key() callback.

If you can't see the objects widen the frustum of the orthogonal projection.

1.6 Setup the viewport

The viewport is the area inside your frame/screen buffer where the image is drawn:

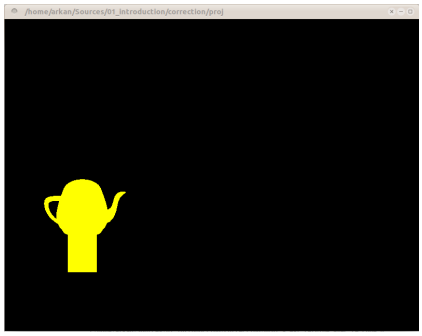


The grey rectangle being the window you can specify the (x, y) position of the viewport as well

as its width 'w' and height 'h'.

- Use glViewport(int x, int y, int w, int h); to setup a square viewport about 300 pixels large. The best is to call the function at the top of the display loop function.

Here is what you should see:



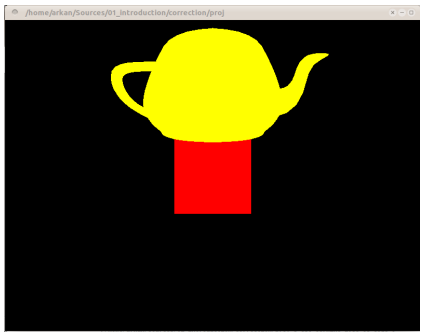
As you can see the image has been scaled down. Note also that because the viewport doesn't respect the original aspect ratio of the window the objects appears distorted. Try to resize your window you'll notice the viewport isn't moving.

- For now you can comment the call to glViewport() we'll use it later.

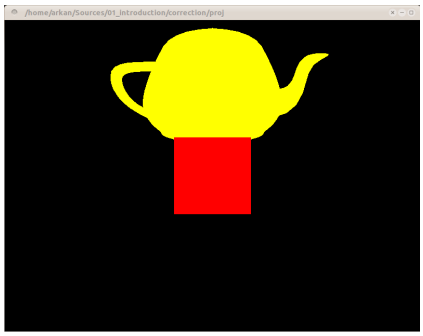
1.7 Z-buffer and depth test

- Draw the cube in red and the teapot in yellow.

This is what you'll saw:



The top of the cube is hidden by the teapot although we should see it. This is because the teapot is drawn before the cube. We need to enable the depth test to see this:



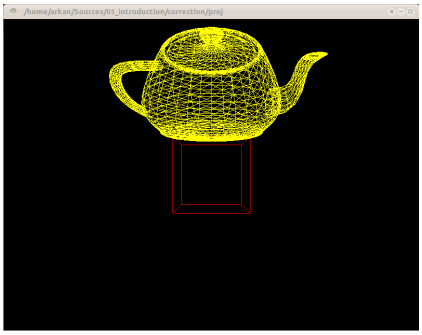
You can enable and disable the depth test using glEnable(...) or glDisable(...) with the parameter GL_DEPTH_TEST.

- Enhance the key() function to activate/deactivate the depth test with the key 'z'.

Don't forget:
- To tell glut to use a depth buffer by adding the flag GLUT_DEPTH to glutInitDisplayMode().
- To clear the depth buffer by adding the flag GL_DEPTH_BUFFER_BIT to glClear().

1.8 Wire-frame view

Often used by artist using modeling software this view can help understand the topology of the objects:



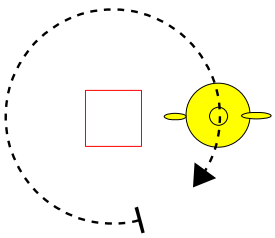
- Modify the function key() (use the 'w' key) to allow the user to switch between FILL and LINE modes

To do so you can use:

- glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
- glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

1.9 Moving around

The goal here is to make our teapot turning around the cube over time. Here is the movement we try to achieve (top view):



We need to increment a global variable for the angle of rotation and rotate the teapot accordingly.

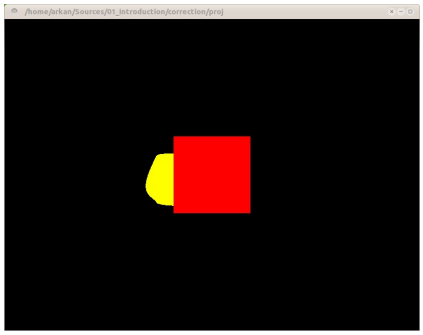
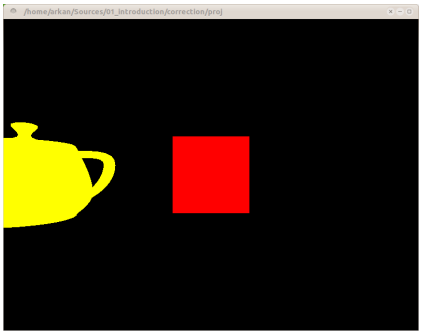
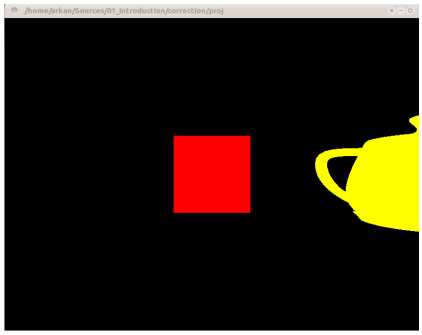
- Add a GLUT callback for the idle event (i.e. called when there is not events) you can use the

function void glutIdleFunc(void (*func)(void)). You'll need to define a new function (void no_events() for instance) in which you'll increment the variable between [0 360].
- Don't forget to call glutPostRedisplay() to refresh the screen.

To realize the above movement you'll need to use in the display loop:
- glTranslate3f()
- glRotatef(float angle, float x, float y, float z); where angle is in degrees and the vector (x, y, z) the axis of rotation

- Add a key shortcut to activate/deactivate the animation with 'a'.

Some screenshots of what is expected:

1.10 Double buffering

Double buffering as it's name stands is a technique which involves two screen buffers instead of one.

1) What problem could appear if only a single buffer is used?
2) Briefly describe how double buffering works

Activate double buffering in your applictation:

- You must change the flag GLUT_SINGLE to GLUT_DOUBLE in glutInitDisplayMode()
- Use glutSwapBuffers() instead of glFlush()

3) Do you noticed changes?

1.11 Reshape the window

So far when you resized your window distortions occurs because the perspective is setup with fixed parameters. the aspect ratio is fixed but when you resize the window it changes thus the wrong aspect. You need to setup the projection matrix each time the window is resized as well as the viewport size.

- Add a glut callback void reshape() for the event of resizing the window with:
  void glutReshapeFunc(void (*func)(int width, int height));

Inside your reshape function:

- Add glViewport()
- Update two global variables float _width; float _height;
- Refresh the screen

In display():
- Make sure to use the parameters width/height instead of fixed parameters to setup the perspective projection.

In main():
- Make sure to use the parameters width/height instead of fixed parameters