ENSIM, 5A INFO IPS

Image Synthesis, Lab1

1.1. Creating an OpenGL Framework

1- Introduction

To write OpenGL programs we must be able to interact with the underlying graphics system. In particular, displaying output requires manipulation of the operating system's window manager. To simplify this process, we use GLUT - OpenGL Utility Toolkit – module. Through some simple initialization, GLUT will take care of all the window management as well as provide for basic user interaction via handling keyboard and mouse events. GLUT is an ***event driven*** API, meaning that we will be defining ***callback functions*** that will be executed in response to various system generated events, e.g. if the window needs to be redrawn, a mouse button has been clicked, a key has been pressed, etc.

2- Initialize GLUT

In order to have GLUT handle the window management tasks, typically our main program will contain the following sections:

1. initialize the library (module)
2. set the various window properties
3. create the window
4. register any needed callbacks
5. begin the (infinite) event loop

Write the corresponding Python/OpenGL code.

3- Draw in display()

The display callback is executed every time the window needs to be displayed, including when the program first starts. Later we will see how to use this routine to perform animations and interactive scenes. We will begin by simply drawing a red square by (1) clearing the screen, (2) draw the scene (simply a red square), (3) flushing the buffer, or (4) swapping the buffer to display it on the screen (via double buffering).

Write the corresponding Python/OpenGL code.

4- Rendering the scene

To add more modularity in our code we can put all the drawing commands (rendering) in the user defined `render_Scene()` that we call from within the `display()` function (otherwise we can put them in the `display()` function itself).

The Python/OpenGL code corresponding to drawing a square is given hereafter (we will see more details about squares drawing later in the following lab sessions).

Write the corresponding Python/OpenGL code.

5- Exercises (homework)

1. the Internet to find how we can change the background color of the window (black by default) and set it to white (or any other color of your choice).
2. Draw the previous window in the middle of the screen (horizontally and vertically) with dimensions (width and height) equal to 500x500.

Hint: you can get your screen resolution using the function `size()` of the Python module `pyautogui` (search how to use it on the Internet).

3. Add the code allowing to display the size and the position of the current OpenGL window and update this display every time the window is moved or resized.

Hint: you can use the function `glutGet` (search how to use it on the Internet).

1.2. Drawing basic shapes in OpenGL

1. Introduction

OpenGL can draw only a few basic shapes, including points, lines, and triangles. There is no built-in support for curves or curved surfaces; they must be approximated by simpler shapes. The basic shapes are referred to as primitives. A primitive in OpenGL is defined by its vertices.

The code we are going to write in the following will be based on the template file (framework) we created during the last lab (file template.py). More particularly, our task is to write the code that draws the scene in the user defined `render_Scene()` function.

2- Drawing points

There is only one kind of point primitive: GL_POINTS. This will cause OpenGL to interpret each individual vertex in the stream as a point. The function related to drawing points in OpenGL are:

- `glVertex2f()` - specify a vertex. The parameters are x-coordinate and y-coordinate on the 2D Plane.
- `glVertex3f()` - specify a vertex. The parameters are x-coordinate, y-coordinate and z-coordinate on the 3D Plane.
- `glColor3f()` - Sets the color of the point. The parameters are red, green and blue (RGB).

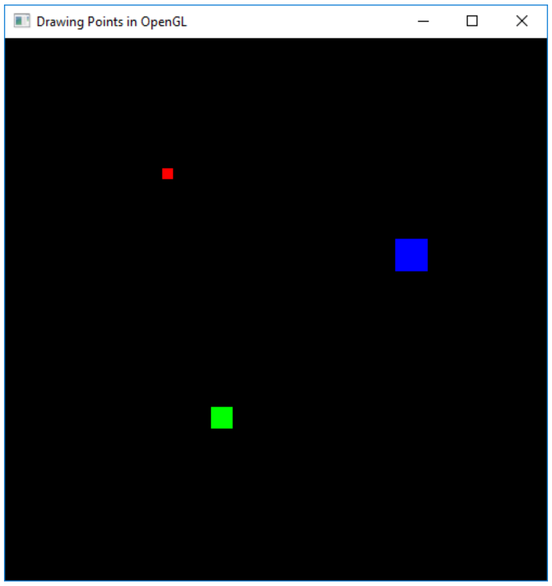
`glPointSize()` - Sets the size of the point.

The basic code that allows to draw a single point is the following:

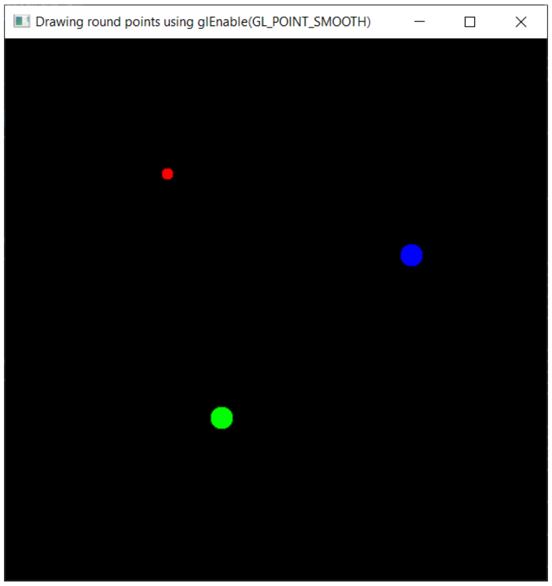
```
glBegin(GL_POINTS)
glVertex2f(0.0, 0.0)
glEnd()
```

To draw several points, the previous bloc of code must be repeated as many times as necessary.

Give the code that allows to draw three points with different positions, colors and sizes.



The points are rasterized by default as squares but we can change them to circles by enabling the point smooth option using the call `glEnable(GL_POINT_SMOOTH)` (once at the beginning of the `render_Scene(). Give the code to change the form of point to circles`



If we do not want to change neither the color nor the size of the point/vertex we can use a single bloc/container GL_POINTS and add all the vertices to it.

3- Drawing a circle using points

We can draw more complex shapes using points.

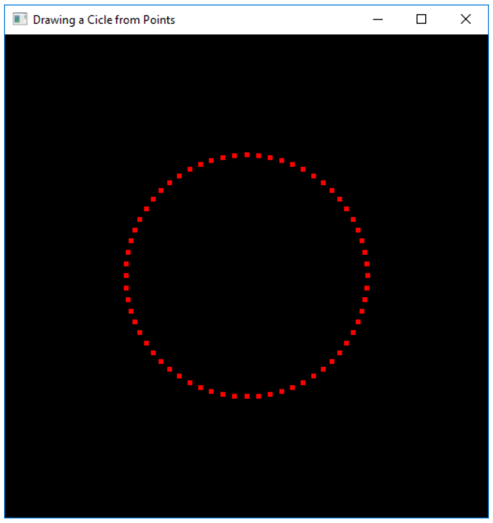
For example, we can draw a circle by drawing its individual points within a loop after specifying their coordinates using the well-known following formulae (and changing the value of the angle α in the interval $[0,360]$ to scan a whole circumference)

$$x = r\cos(\alpha)$$

$$y = r\sin(\alpha)$$

The previous equations suppose that the center of the circle is the origin of the axes (0,0) but we can change the position of the center by adding the values of its abscissa and oordinates to x and y respectively.

Give The corresponding Python code (`render_Scene()`

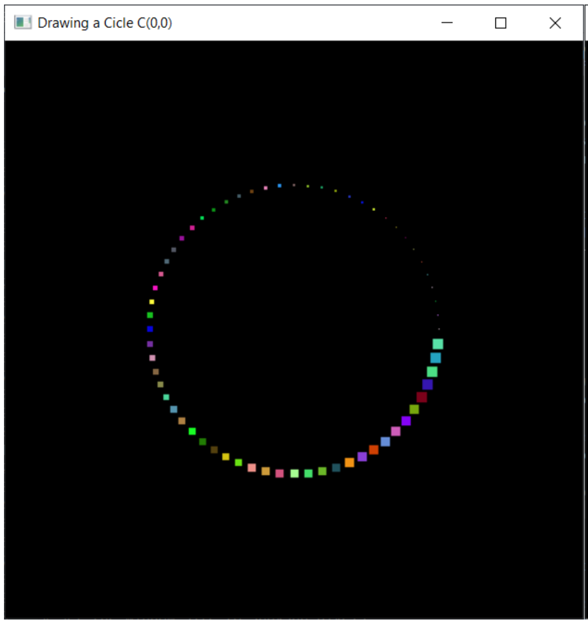


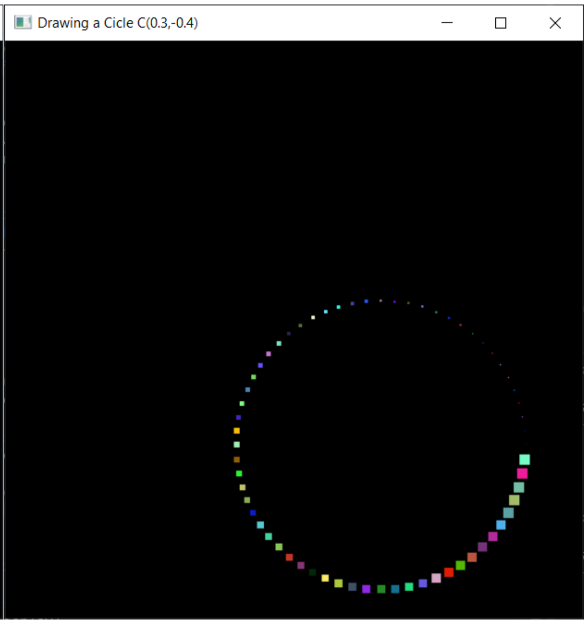
Exercise

Modify the previous code so that the center of the circle is chosen at the beginning of the code (e.g. global variable) and the sizes of the points increase progressively from 1 (first point) to 10 (last point) and their colors are randomly chosen.

Hint: the method random from the module random returns a random float number between 0 and 1.

Examples of the expected outputs are:

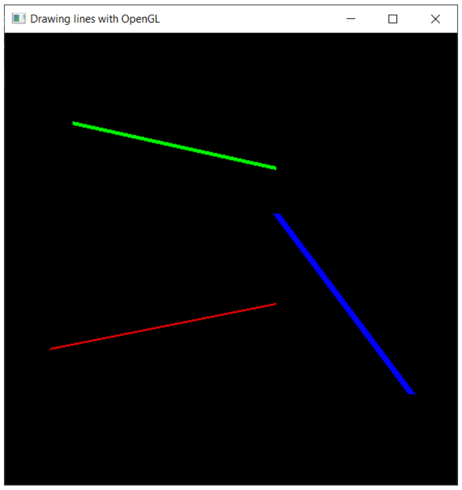




Drawing lines

There are three primitives for drawing line segments: GL_LINES, GL_LINE_STRIP, and GL_LINE_LOOP. GL_LINES draws disconnected line segments; specify two vertices for each segment that we want to draw. The other two primitives draw connected sequences of line segments. The only difference is that GL_LINE_LOOP adds an extra line segment from the final vertex back to the first vertex.

- `glVertex2f()` - specify the vertices of the segment. The parameters are x-coordinate and y-coordinate on the 2D Plane. Two vertices must be specified for each segment.
- `glVertex3f()` - specify a vertex. The parameters are x-coordinate, y-coordinate and z-coordinate on the 3D Plane. Two vertices must be specified for each segment.
- `glColor3f()` - Sets the color of the line. The parameters are red, green and blue (RGB).
- `glLineWidth()` — specify the width of rasterized lines



The basic code that allows to draw a line segment is the following:

```
glBegin(GL_LINES)
glVertex2f(-0.4, 0.6)
glVertex2f(0.2, 0.2)
glEnd()
```

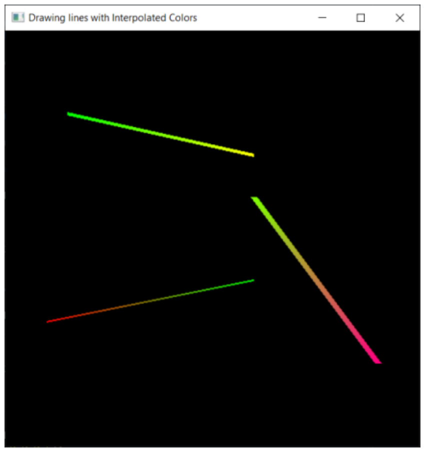
To draw several lines, the previous bloc of code must be repeated as many times as necessary.

Write the code that allows to draw three lines with different positions, colors and widths is given below (render_Scene())

Colors interpolation when drawing lines

When the colors chosen for the two points of the line are different, OpenGL perform an interpolation to draw the line. That is, given two points, OpenGL will determine the set of pixels that the line would cross through and then change their color based on the color of the line. Bresenham's algorithm is a famous algorithm for drawing lines, and it (or some extension of it) is probably the one used internally by OpenGL.

Give the code draws the same lines of the previous paragraph choosing different colors for the vertices corresponding to each line (`render_Scene()` function).



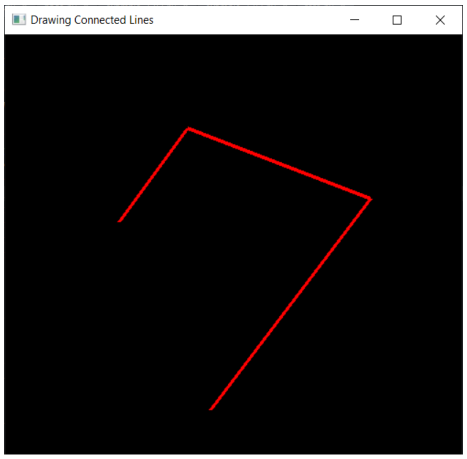
Draw connected sequences of line segments

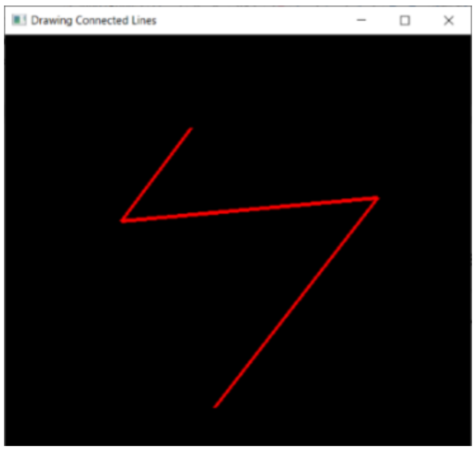
To draw connected sequences of line segments, we must use the GL_LINE_STRIP primitive and then precise the set of vertices to connect.

Write the code for drawing a connected line strip **N.B.**

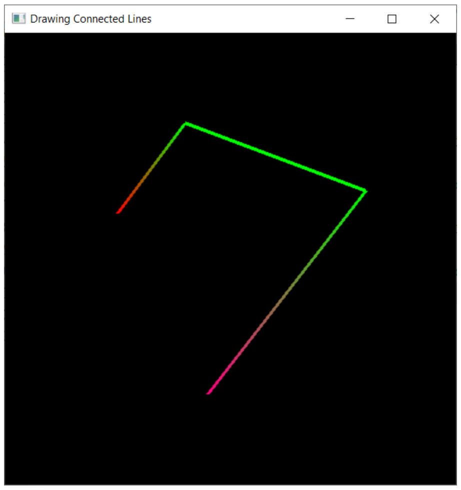
N B: The **order** of adding the vertices matters in the form of the shape to be drawn. For example, if we change the order of appearance of the same four vertices of the previous paragraph, we will have another shape different than the previous one.

Write a code with **different colors** chosen for the vertices of each line.



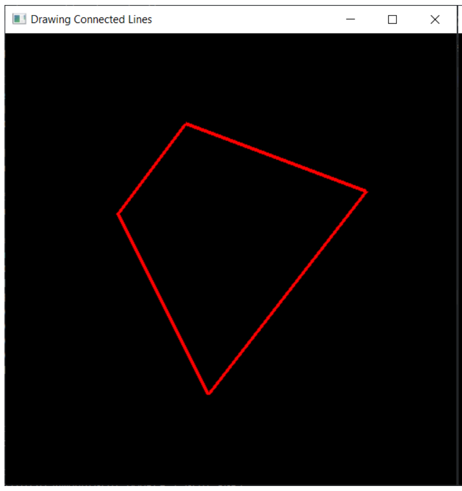


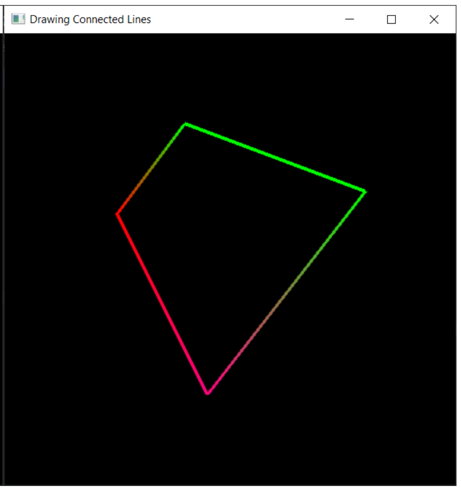
The corresponding output is:



To allow drawing a connected group of line segments from the first vertex to the last, then back to the first, we must use the GL_LINE_LOOP primitive and then precise the set of vertices to connect.

The same examples of the previous paragraph replacing GL_LINE_STRIP with GL_LINE_LOOP will give the following outputs:





Question

How many calls to the function glVertex2f (or glVertex3f) should we have, when using each of the three primitives GL_LINES, GL_LINE_STRIP, and GL_LINE_LOOP, in order to have a shape composed of n connected lines?

Exercises

1. Draw an X-Y axis system using **red** lines (in the middle of the drawing window)
2. Draw a **yellow** triangle using closed-loop connected lines (GL_LINE_LOOP primitive) built around the following coordinates:
   x1, x2, x3 = 0.3, 0.7, 0.5
   y1, y2, y3 = 0.2, 0.5, 0.7
   Note that the triangle is located in the first quadrant of the X-Y plane
3. Draw the reflection of the triangle in all the remaining 3 quadrants of the X-Y Plane
   You should have a figure similar to the following one: