**GROUP 11:**  
Joseph Muema - SCT211-0039/2022  
Brian Kipng’eno - SCT211-0078/2022  
Jayden Mathenge - SCT211-0030/2022  
Kelly Kasina - SCT211-0038/2022

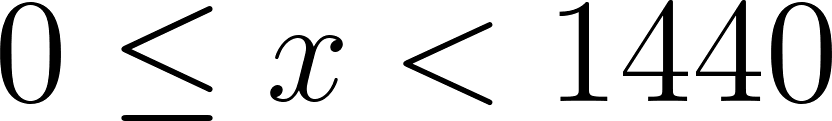
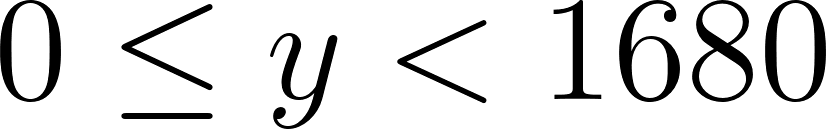
**QUESTION B**

Suppose you have a system with a 12 inch by 14 inch video monitor that can display 120 pixels per inch. If memory is organized in one byte words, the starting frame buffer address is 0, and each pixel is assigned 4 bits of storage, what is the frame buffer address of the pixel with screen coordinates (x,y)?

**Solution**

Compute your screen’s resolution:

* **Width (pixels)** = 12 in × 120 ppi = **1 440** pixels
* **Height (pixels)** = 14 in × 120 ppi = **1 680** pixels

Each pixel uses **4 bits** (half a byte), and pixels are laid out in **row‑major order**. Let the coordinates be (x,y) with and .

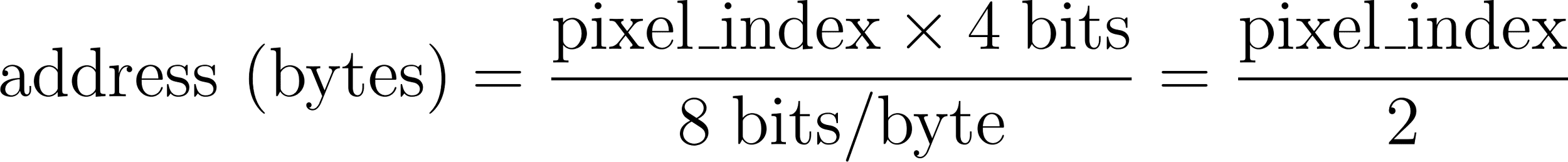
**1. Pixel Index**

Index the pixels from 0 in row‑major order:

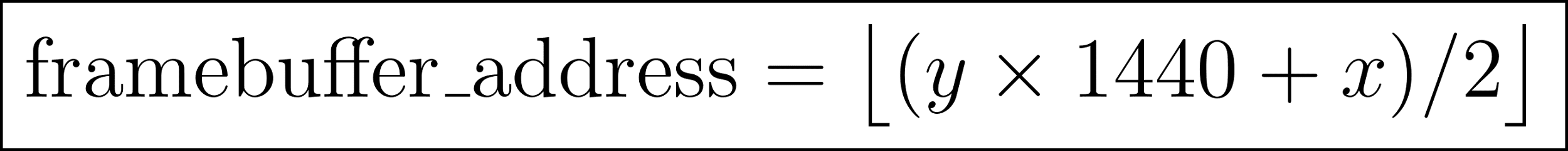
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctext%7Bpixel%5C_index%7D%20%3D%20y%20%5Ctimes%20%5Ctext%7Bwidth%7D%20%2B%20x%20%3D%20y%20%5Ctimes%201440%20%2B%20x#0)

**2. Byte Address**

Since each pixel is 4 bits, two pixels share one byte (packed‑pixel format). Thus:

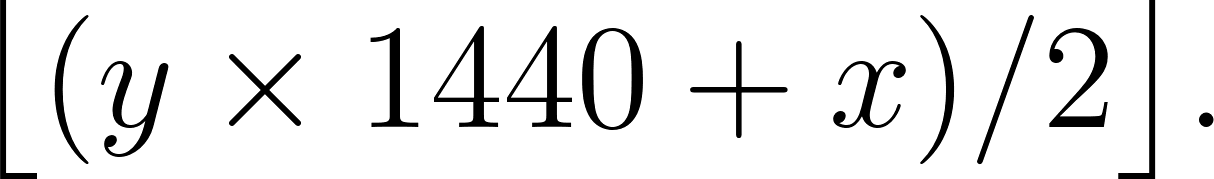
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctext%7Baddress%20(bytes)%7D%20%3D%20%5Cfrac%7B%5Ctext%7Bpixel%5C_index%7D%20%5Ctimes%204%5C%3B%5Ctext%7Bbits%7D%7D%7B8%5C%3B%5Ctext%7Bbits%2Fbyte%7D%7D%20%3D%20%5Cfrac%7B%5Ctext%7Bpixel%5C_index%7D%7D%7B2%7D#0)

Because we can only address whole bytes, take the integer division:

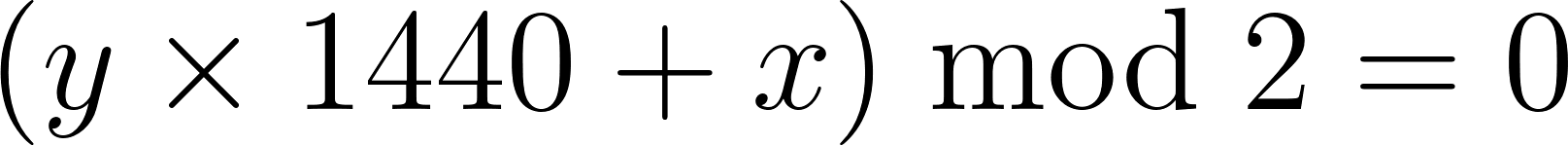
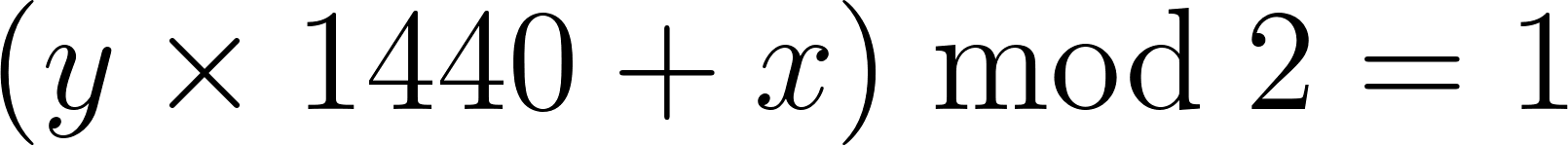
[](https://www.codecogs.com/eqnedit.php?latex=%5Cboxed%7B%20%5Ctext%7Bframebuffer%5C_address%7D%20%3D%20%5Cbigl%5Clfloor%20(y%20%5Ctimes%201440%20%2B%20x)%20%2F%202%20%5Cbigr%5Crfloor%20%7D#0)

**3. Nibble Selection**

The “nibble” (4‑bit half of a byte) doesn’t change the **byte address**, that’s always

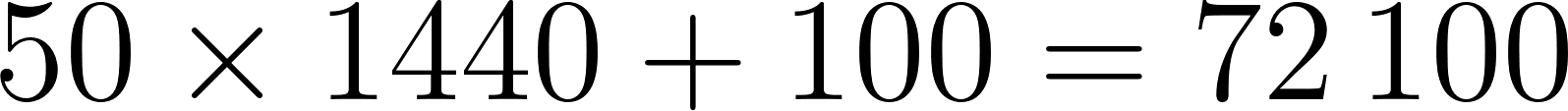
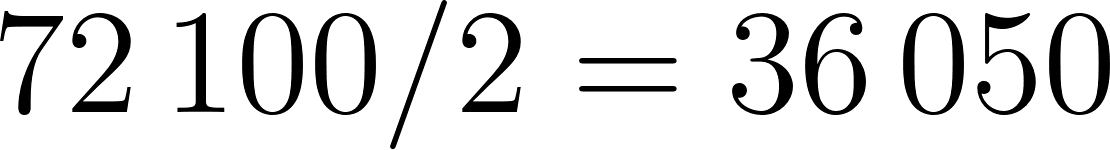
[](https://www.codecogs.com/eqnedit.php?latex=%5Cbigl%5Clfloor%20(y%20%5Ctimes%201440%20%2B%20x)%20%2F%202%20%5Cbigr%5Crfloor.#0)

Instead, it tells **which half** of that byte holds the pixel’s 4 bits:

* If , the pixel is in the **high nibble** (bits 7–4) of the byte.
* If , it’s in the **low nibble** (bits 3–0).

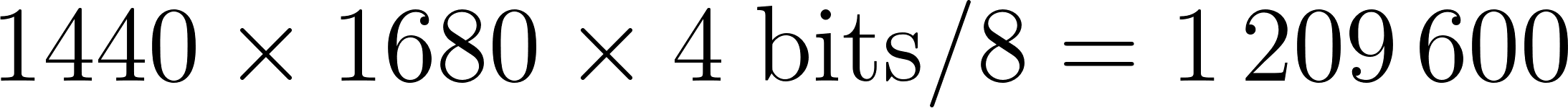
**4. Practical Example**

For pixel (100,50):

1. Compute index:  
      [](https://www.codecogs.com/eqnedit.php?latex=50%20%5Ctimes%201440%20%2B%20100%20%3D%2072%5C%2C100#0)
2. Byte address:  
      [](https://www.codecogs.com/eqnedit.php?latex=72%5C%2C100%20%2F%202%20%3D%2036%5C%2C050#0)
3. Nibble:  
    [](https://www.codecogs.com/eqnedit.php?latex=72%5C%2C100%20%5Cbmod%202%20%3D%200#0) → high nibble of byte 36 050

So you’d read/write byte **36 050** and mask out the top or bottom 4 bits accordingly.

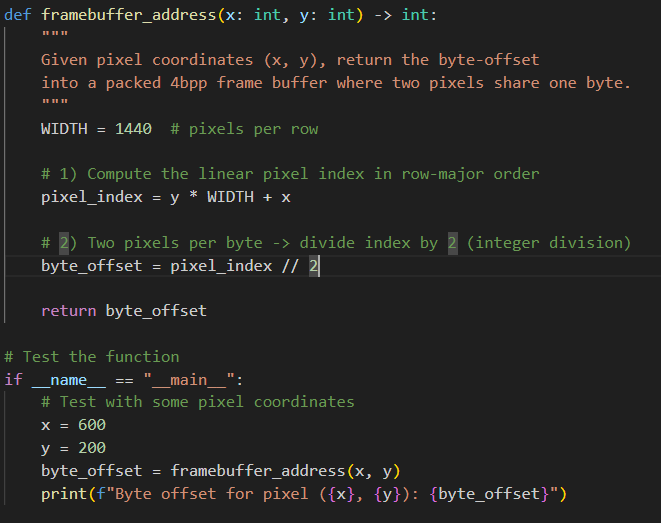
**Why This Works**

* Total framebuffer size:  
      bytes
* Packed‑pixel format ensures two 4‑bit pixels per byte, halving the per‑pixel byte offset.

This formula gives the exact byte offset in your linear frame buffer for any (x,y).

**Code Example**

In this example, the values of the x and y coordinates are given. The *framebuffer\_address* function returns the byte offset for the pixel coordinates in memory;



**Output**



**QUESTION C**  
Using OpenGL Draw a figure with coordinate points A(0, 4), B(3, 4), C(4, 0), D(0, 0).

i. Apply the translation with distance 2 towards X axis and 2 towards Y axis. Obtain the new coordinates of the square.

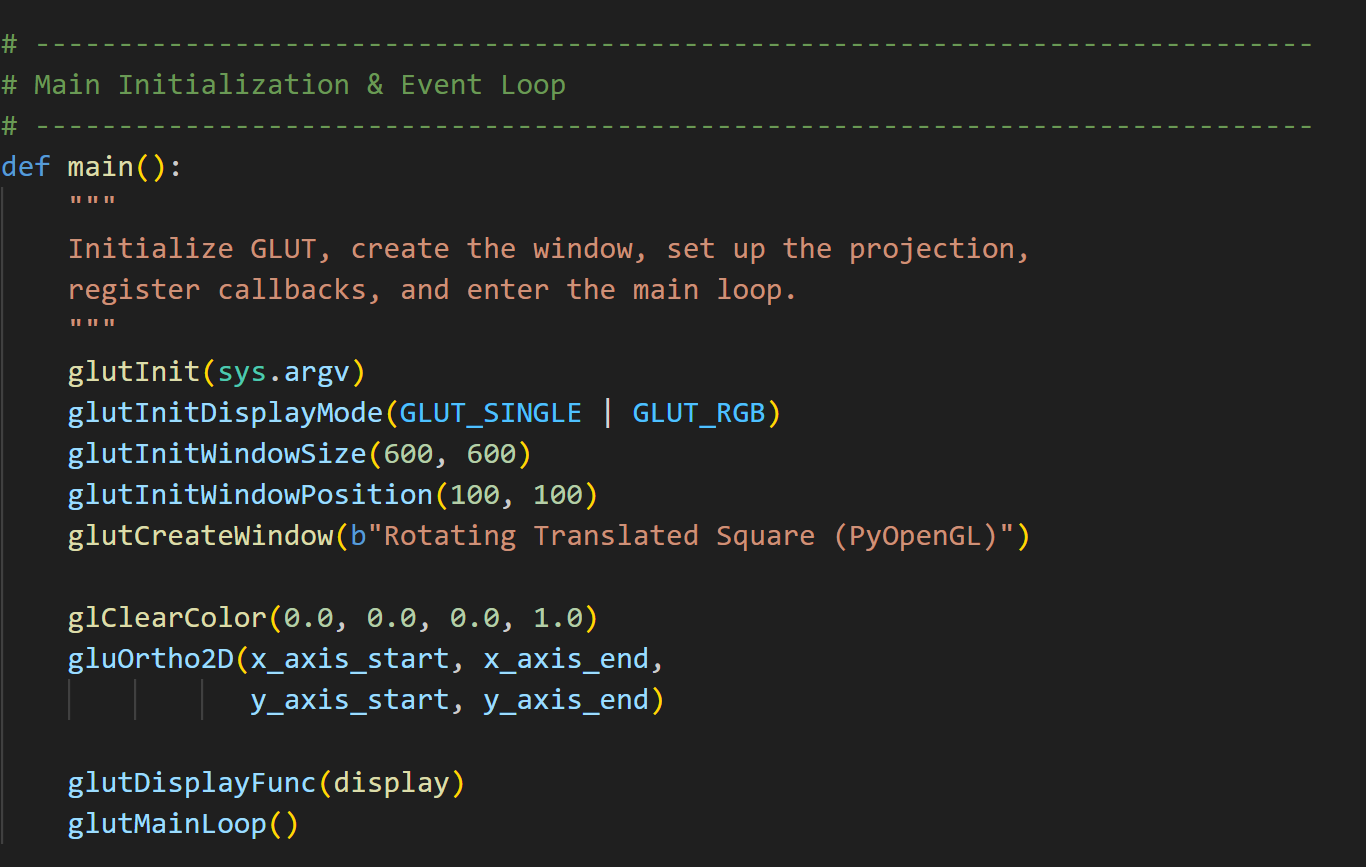
ii. The translated figure should have a green border line and cream inner shading

iii. Write an openGL program to rotate the translated figure on Rotation angle = θ = 55º and show the output

**Solution (Steps)**

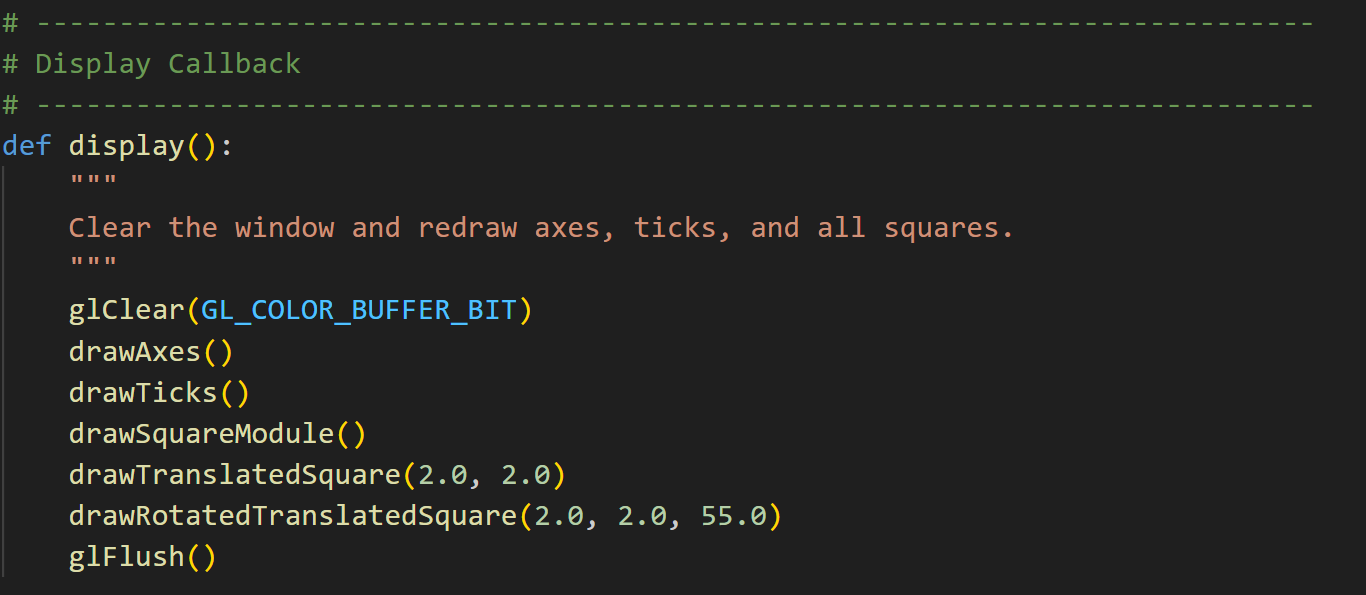
### **1. Initialization & Window Setup**

* **GLUT Initialization:** The program begins by initializing GLUT with command‑line arguments.
* **Window Creation:** It requests a single‑buffered RGB window of size 600×600 pixels, positioned on screen, and gives it a title.
* **Background & Projection:** It sets the clear color to black and establishes a 2D orthographic projection so that your logical coordinate range (–10 to +10 on both axes) maps directly to the window.



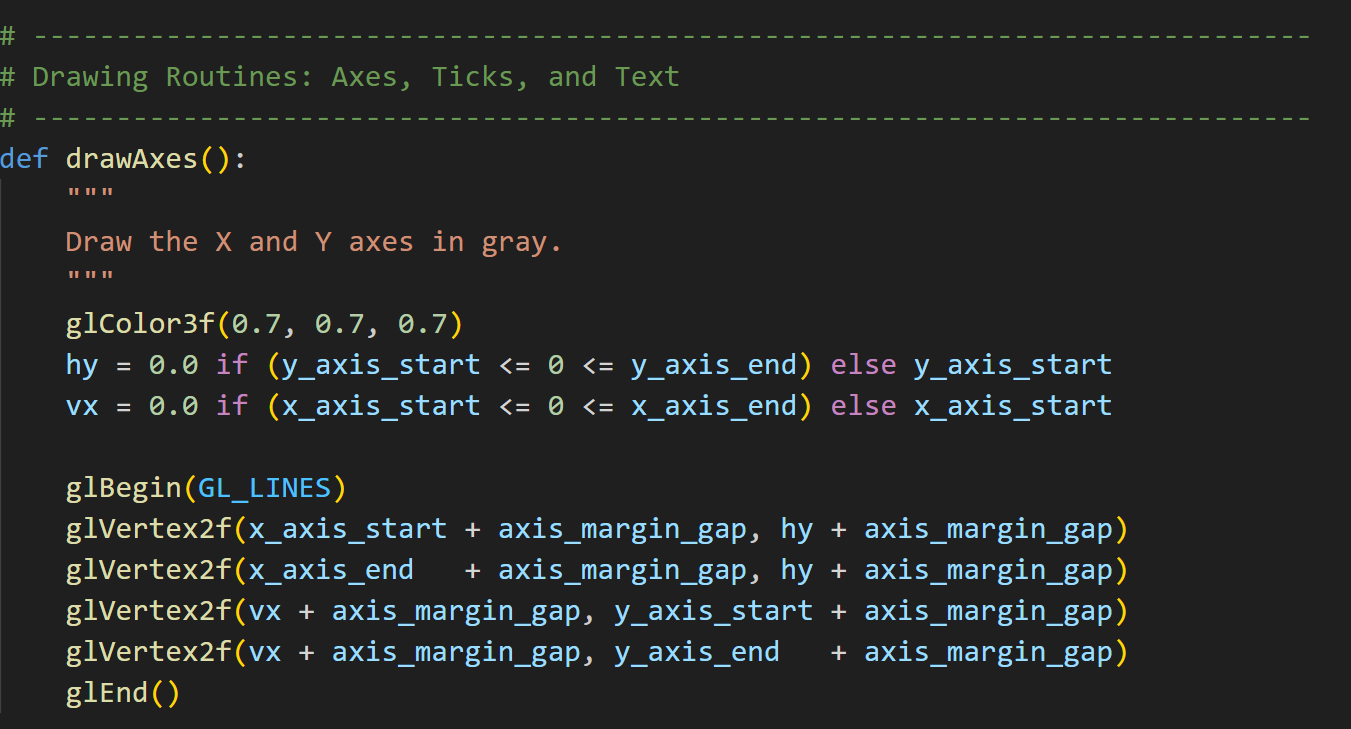
### **2. Main Loop & Display Callback**

* **Display Function:** A single display callback is registered. Whenever the window needs redrawing, this function is invoked.
* **Event Loop:** GLUT’s main loop takes over, listening for events and calling the display function as needed.



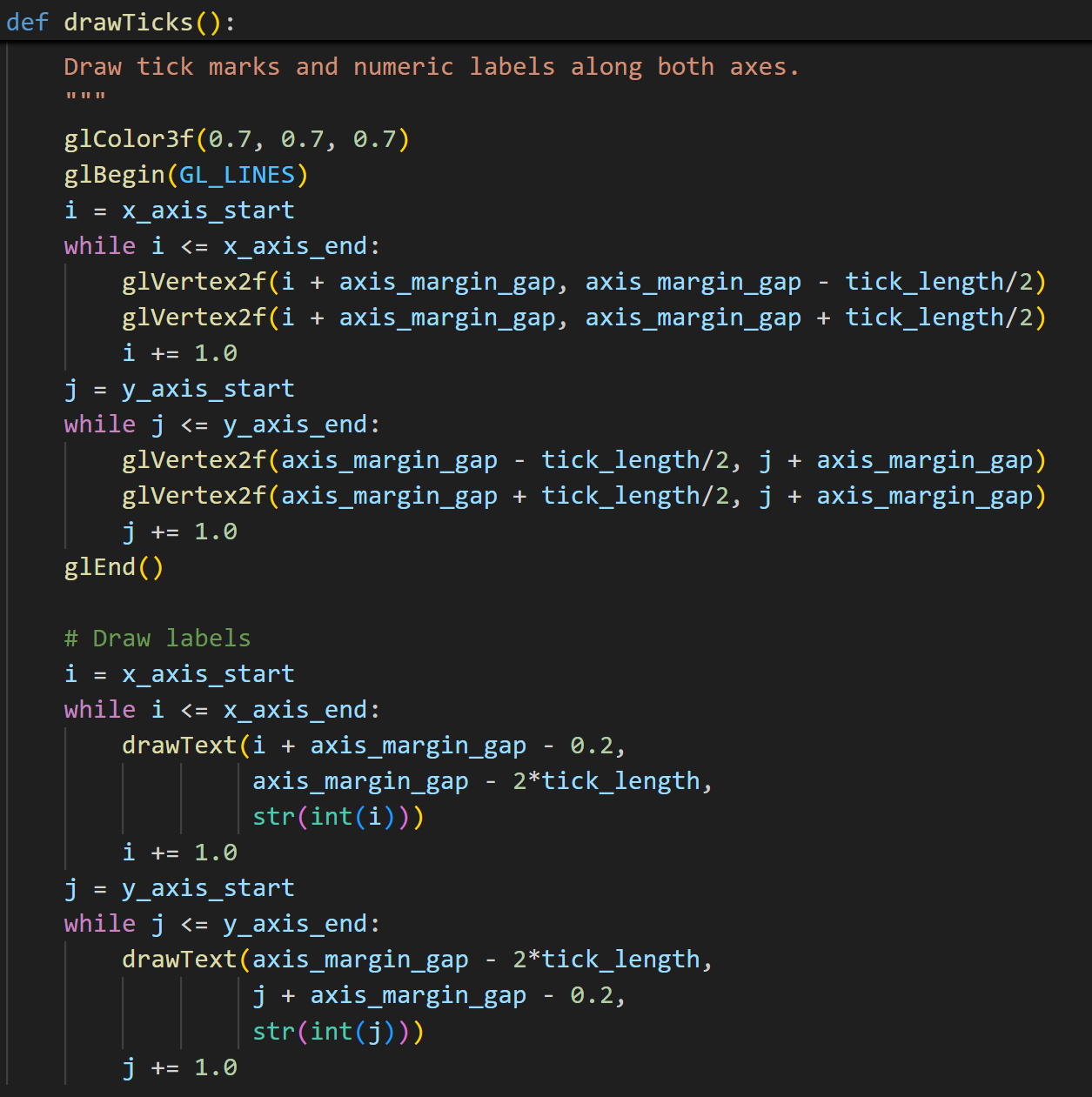
### **3. Drawing the Axes**

* **Axis Lines:** The program computes where the X and Y axes should lie (at zero if zero is in range, otherwise at the minimum).
* **Rendering:** It draws two gray lines spanning the logical coordinate range, representing the horizontal and vertical axes.



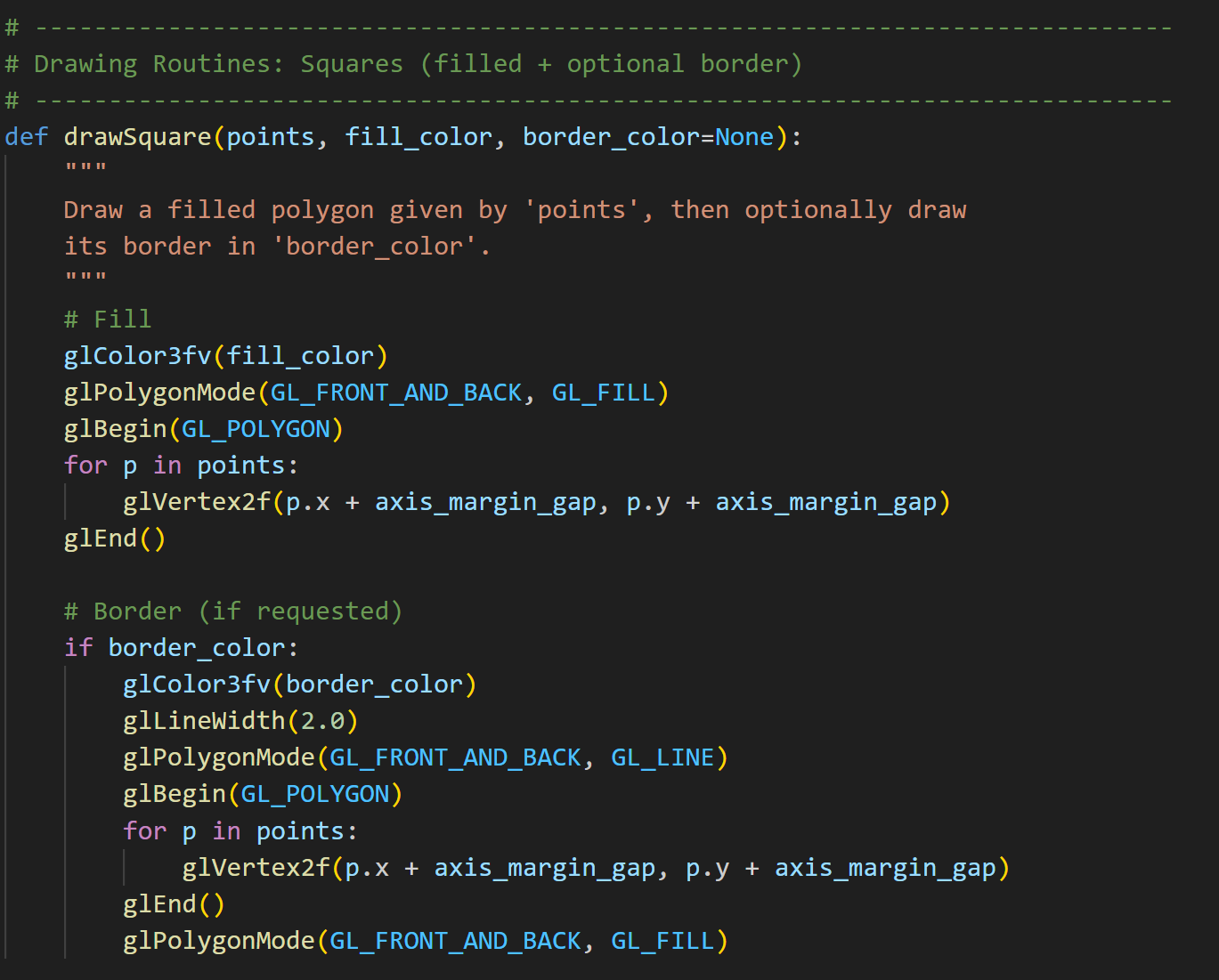
### **4. Adding Tick Marks & Labels**

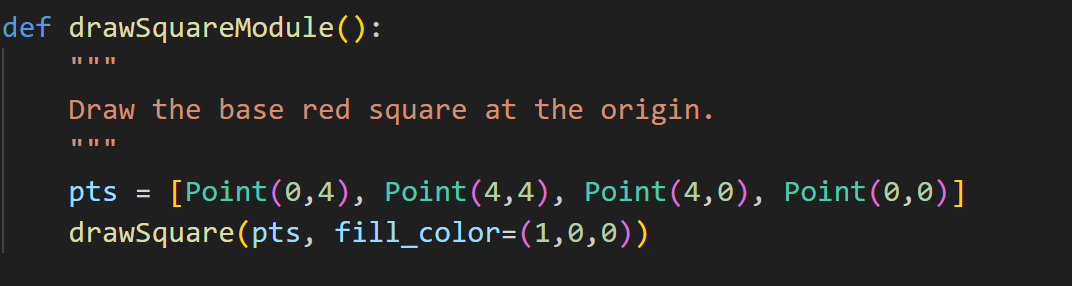
* **Tick Lines:** Along each axis, it steps through integer positions and draws small perpendicular ticks.
* **Numeric Labels:** At each tick, it renders a tiny bitmap numeral to indicate the coordinate value.



### **5. Drawing the Square (Base)**

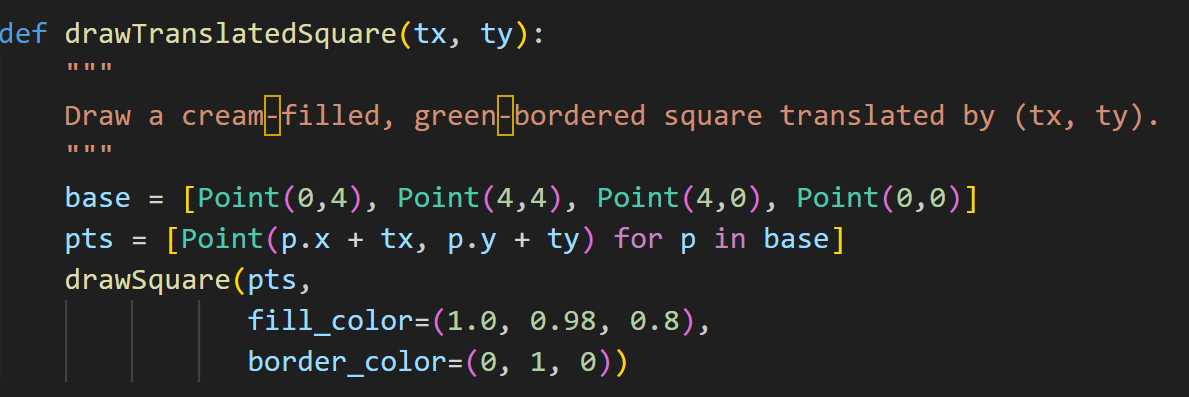
* **Unit Square:** A red, filled square of size 4×4 units is drawn at the origin. This demonstrates basic polygon drawing.

  
*For General Square Drawing*

  
*Specifically for the Base Square*

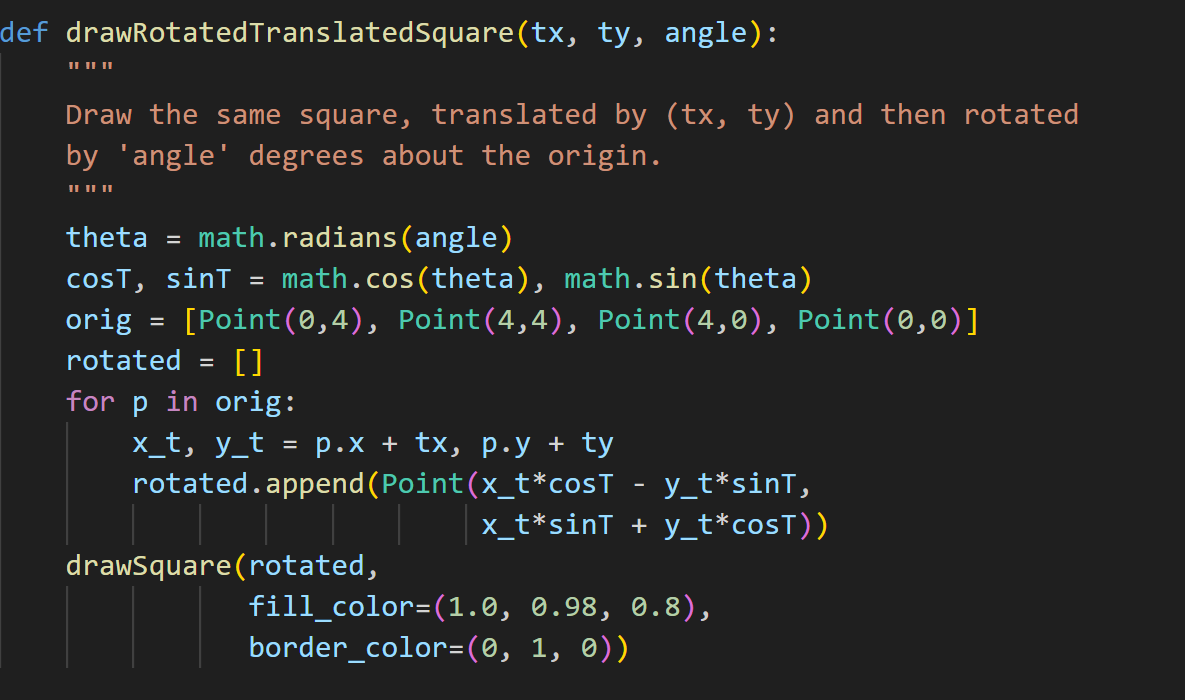
### **6. Drawing a Translated Square**

* **Translation:** The same 4×4 square is shifted by (2, 2) units.
* **Styling:** It’s filled in a “cream” color, then outlined in green—by first drawing a filled polygon, then redrawing its border in line mode.



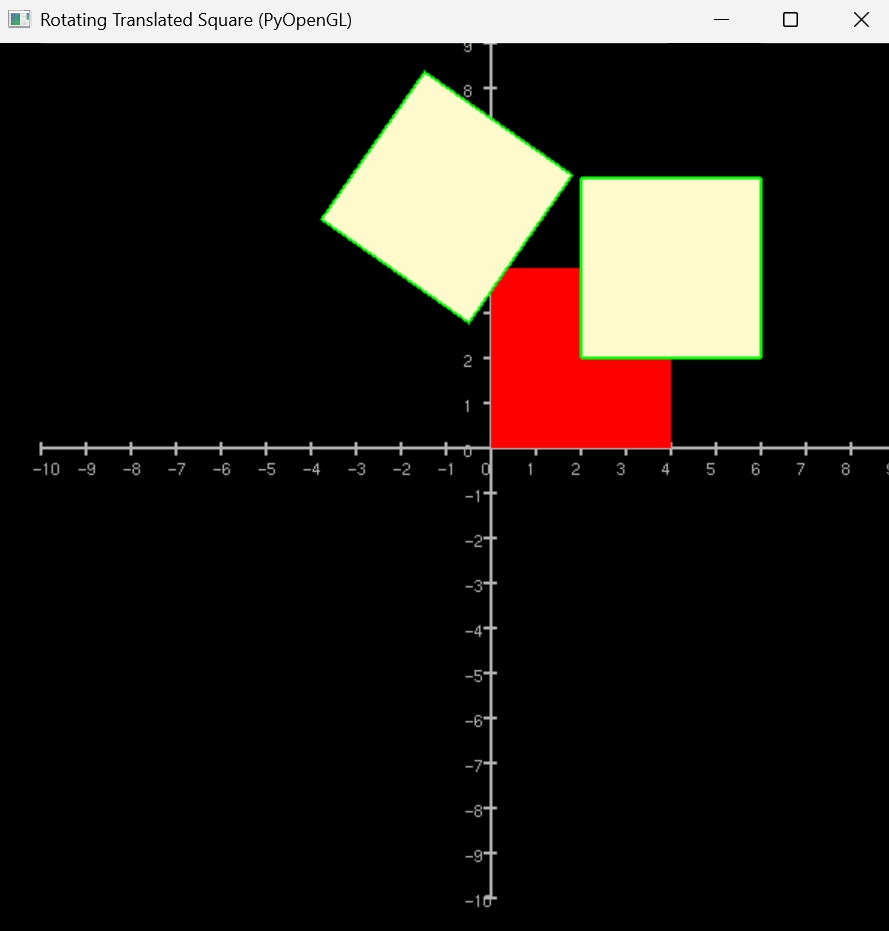
### **7. Drawing a Rotated & Translated Square**

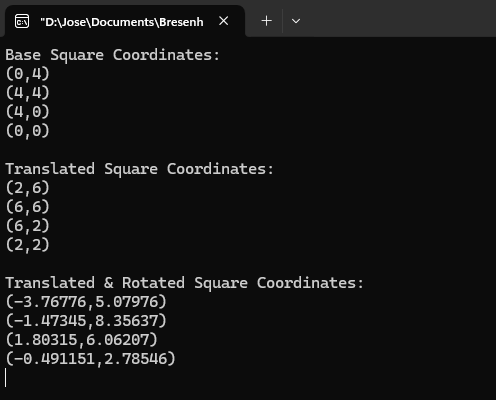
* **Rotation + Translation:** The square is first translated by (2, 2) and then rotated by 55° about the origin.
* **Same Styling:** Again filled in cream and outlined in green, illustrating how you can combine transformations before rendering.



### **8. Output**

Each frame clears the screen, draws axes and ticks, then renders the three squares in turn. The result is a static scene showing coordinate axes with tick marks, plus a red base square, a shifted square, and a shifted + rotated square.

  
*Window Output*

*  
Square Coordinates*