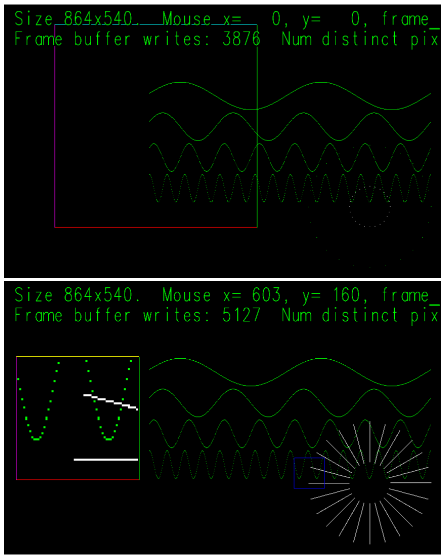
LSU EE 4702-1 **Homework 1** **Due: X2 September 2022**



```
Size 864x540.  Mouse x=   0, y=   0, frame_
Frame buffer writes: 3876  Num distinct pix
```

```
Size 864x540.  Mouse x= 603, y= 160, frame_
Frame buffer writes: 5127  Num distinct pix
```

Problem 0: Follow the instructions on the `https://www.ece.lsu.edu/koppel/gpup/proc.html` page for account setup and programming homework work flow. Compile and run the homework code unmodified. It should initially show the square from the `cpu-only/demo-01-frame-buffer.cc` code used in class, along with some sine waves and faint dotted concentric circles. See the screenshot to the upper-right. The lower screen shot is from a correctly solved assignment. The square is positioned on the upper-left, and it now shows a magnified version of what is in a blue square (which is where the mouse pointer is). Lines now connect the dots that formed the two concentric circles.

User Interface

Press `Ctrl=` to increase the size of the green text and `Ctrl-` to decrease the size. Press `F12` to generate a screenshot. The screenshot will be written to file `hw01.png` or `hw01-debug.png`.

Display of Performance-Related Data

The top green text line shows performance-related and other information. `Size` refers to the size of the window. `Mouse` refers to the coordinates of the mouse pointer. Coordinate $(0, 0)$ is at the **lower** left of the window. Text `frame_buffer[N]` shows the index of the frame buffer corresponding to the point under the mouse pointer. (In the assignment file `frame_buffer` is abbreviated to `fb`, for convenience.)

`Render Time` and `Potential Frame Rate` show the CPU time needed to write the frame buffer. They can be ignored for this assignment. See Problem 2 for the second line, the one that starts `Frame buffer writes.`.

Code Generation and Debug Support

The compiler generates an optimized version of the code, `hw01`, and a debug-able version of the code, `hw01-debug`. The `hw01-debug` version is compiled with optimization turned off, which makes it easier to debug. When needed, you are strongly encouraged to run `hw01-debug` under the GNU debugger, `gdb`. See the material under "Running and Debugging the Assignment" on the course procedures page. **You must learn how to debug.** If not, you will be at a severe disadvantage.

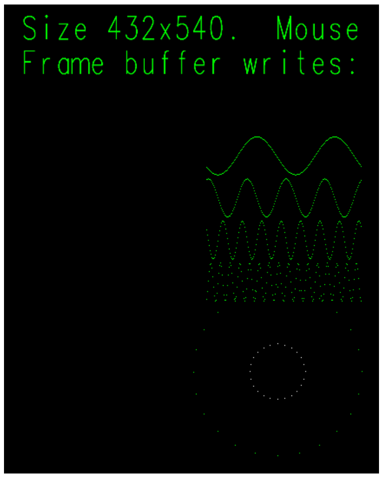
Problem 1: The code in routine `render_hw01` includes a fragment that draws a square (by writing the frame buffer), which is based on what was done in class on Wednesday, 24 August 2022:

```
for ( int x=100; x<500; x++ )
  {
    fb[ 100 * win_width + x ] = color_red;
    fb[ 500 * win_width + x ] = 0xffff;
    fb[ x * win_width + 100 ] = 0xff00ff;
    fb[ x * win_width + 500 ] = 0xff00;
  }
```

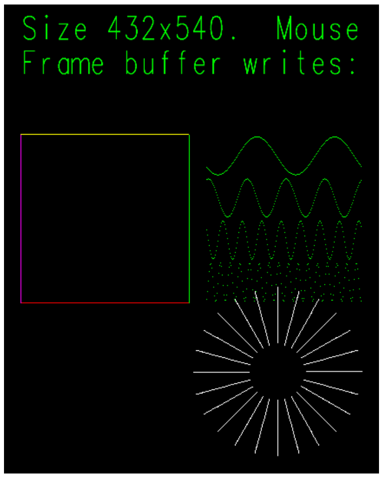
The position of this square is hard-coded to coordinates $(100, 100)$ (meaning $x = 100$, $y = 100$) lower-left and $(500, 500)$ upper-right. That will place the square in the lower-left portion of the window.

Modify the routine so that the square is drawn at (`sq_x0`,`sq_y0`) lower-left and (`sq_x1`,`sq_y1`) upper-right, where `sq_x0`, `sq_y0`, `sq_x1`, and `sq_y1`, are variables in the code. Do this by using these variables in the routine that draws the square. If it helps, variable `sq_slen` can also be used. If done correctly the square will be at the upper-left of the window vertically aligned with the sine waves, and the size of the square will be determined by the minimum of the window width and height. The square will adjust whenever the window is resized. See the lower screenshot at the beginning of this assignment.

Problem 2: The homework file has a placeholder routine `line_draw(fb,x0,y0,x1,y1,th,co)` which is to draw a line of color `co` from coordinate `x0,y0` to `x1,y1`. Add code to `line_draw` so that it draws such a line. The placeholder code writes a pixel at either end of the line. Code in `render_hw01` uses `line_draw` to draw lines radiating from a point. With the placeholder code, that results in two concentric circle of dots, see the screenshot to the lower left. If the `line_draw` routine is written correctly the lines should be drawn as shown on the screenshot to the right.



```
Size 432x540.  Mouse
Frame buffer writes:
```



```
Size 432x540.  Mouse
Frame buffer writes:
```

The line should appear solid, not dotted. Also, try not to write a pixel more than once when drawing the line. The green text to the right of `Frame buffer writes:` shows how many times the frame buffer was accessed. The text to the write of `Num distinct pixels:` shows how many different pixels were accessed. If the two numbers are the same then each pixel that was accessed was accessed just once. If `Frame buffer writes:` is higher than `Num distinct pixels:` then some pixels were written multiple times. This might happen, for example, when two lines cross. The pixel at their intersection is written twice. That's not a problem. But, it can also happen if code is carelessly written, for example, drawing the same line twice (in exactly the same place).

The text to the right of `Redundancy:` shows the percentage of pixels that were written more than once. When solving this problem make sure that the redundancy stays below 50%.

Problem 3: Modify the code near the end of `render_hw01` so that a blue square is drawn around the mouse pointer, and the contents of the frame buffer covered by the blue square is copied to the square in the upper left. (See the lower screenshot at the beginning of this assignment. The mouse pointer is not visible in the screenshots.) Set the length of an edge of the square to `(sq_slen-2)/zoom`, where `sq_slen` and `zoom` are variables in the code. Variable `zoom` is hardcoded to 4, so the square should show a 4× magnification of what is near the mouse pointer. (The length is `(sq_slen-2)/zoom` rather than `sq_slen/zoom` so that the copied pixels do not overwrite the sides of the square.)

To get you started, there is code to copy one pixel from the mouse pointer to the center of the square:

```
const int mouse_x = fb.mouse_x;
const int mouse_y = fb.mouse_y;
const int zoom = 4;

fb[ ( sq_y1 + sq_slen/2 ) * win_width + sq_x0 + sq_slen/2 ]
  = fb[ mouse_y * win_width + mouse_x ];
```

The code above copies just one pixel. To complete the assignment that needs to be replaced by a loop nest that will copy each pixel in the blue square to `zoom*zoom` locations in the big square.

The location of the mouse pointer is in variables `mouse_x` and `mouse_y`. You may use the `line_draw` routines to draw the blue square.

Pay attention to the following:

- All code for this problem must be placed after the `fb.fb_stats_off();` statement.
- The code to draw the blue box must be placed after the code that copies the frame buffer.
- Do not overwrite the square edges.