KJ Sweet
2/12/16
CS210 Final Project Report
All work is mine unless otherwise cited

Drawing Rectangles in C and MIPS

Project Description

This project is an experiment in working with the MARS Bitmap Display tool. The original conception of the project was to create a program using MIPS in with one could draw a rectangle of width and height of their own choosing within parameters set by the size of the display. Currently, the MIPS part of this program does not ask for input, and draws a rectangle on the outer edges of the display. The C portion of the program, however, does ask for input and draws a rectangle of the size defined by the user. The C program prints to the terminal and as such does not need to be constrained to the same size of the Bitmap Display tool, but I feel the project is more coherent that way.

Explanation of Code

The MIPS portion of this project, because I could not figure out how to get the input working correctly, draws a green rectangle in the Bitmap display. It is based off of both bitmap.asm and bitmap-demo.asm that were provided to me. This program uses the encode function, written by Bob Roos, and loops to draw a green rectangle at the borders of the bitmap display when set to the correct dimensions. Each edge is drawn using a separate loop, and the following code demonstrates the top edge:

```
rect:   lw      $s0,green       #color to draw
        li      $s1,0           # loop counter
        li      $a0,0           # row 1
        li      $a1,0           # starting column
loop3:  beq     $s1,32,rect2    # THIS IS NOT A LEGAL MIPS COMMAND (but MARS allows it)
        jal     encode          # get the address for row a0, col a1
```

```
sw      $s0,0($v0)      # draw a green pixel there
addi    $a1,$a1,1       # move over a column
addi    $s1,$s1,1       # add 1 to counter
j       loop3
```

The subsequent three loops are very similar, but use different row and column values. These could be easily set to variables, if I could have gotten the user input portion to work, but as it is, the display size is set and so are the row and column values of each loop.

The C Portion of this project was fairly easy to implement. It asks for input, stores the values x (width) and y (height), and uses those values to creates a 2 dimensional loop in which each (x, y) combination can be mapped to a coordinate in the parameter space we create. The program then draws a rectangle at the bounds of this parameter space, much like the mips program but using a less complicated structure as its much easier to evaluate logical expressions in C.

```
for(j = 0; j<y; j++) {
  for(i = 0; i<x; i++) {
    if(j == 0 || j == (y-1)){
      printf("* ");
    }//top and bottom
    else if(i == 0 || i == (x-1)) {
      printf("/ ");
    }//sides
    else {
      printf("  ");
    }//middle
  }//over
  printf("\n");
}//down
```

As you can see, the program checks to see if it on an edge, and draws a symbol or just a space depending on whether it is or not. I put an extra space between every symbol horizontally because of the spacing of the lines vertically in the terminal; a square looks much more square this way.

How to Run the Project

To run the Bitmap part of the project, one needs the MARS program and the FinalProj.asm file from the FinalProj folder. Open the file in MARS, and use the “Tools” drop-down menu to open up the Bitmap Display. In the Bitmap Display tool, set “Unit Width in Pixels” and “Unit Height in Pixels” to 16. “Display Width in Pixels” and “Display Height in Pixels” should be set to 512 and 256, respectively. “Base address for display” should be set to 0x10010000 (static data). Then hit “Connect to MIPS” and compile and run the program using the “Run” menu.

To run the C part of this project, one needs the FinalProj.c file from the FinalProj folder. Then simply compile and run the program. The Program will ask you for input. The first thing it asks for is the width of the rectangle you would like to draw, so enter an integer (between the given values!) and hit enter. It then asks for the height of the rectangle you’d like, and you just need to repeat and then it prints the specified rectangle to the terminal.

Lessons Learned

Something that is weird about the Bitmap Display (that might be useful in some contexts) that I discovered while experimenting with it is that it does not clear the display when you compile and run a program, only when you close the tool and reopen it. That is why FinalProj.asm begins by clearing the display, defined as making every pixel black. It is not entirely obvious with how the code runs right now, but if one were to give it input, then immediately run it again with different input, it would be noticeable. After writing this code, I discovered that the display can be cleared using the “Reset” button right next to the “Connect to MIPS” button in the Bitmap Display tool, but I’m a physics major and solving simple problems using excessively convoluted solutions is what we do best.

I enjoyed playing around with the Bitmap Display. It was an interesting experiment with seeing how memory is stored. I was also in John's Visual Computing class this semester and took a lot of processing shortcuts for granted. I can't say that MIPS has been my favorite language to program in, but I certainly have a greater appreciation for the intricacies of the "behind the scenes" work that makes up computer programming.

References

Bob Roos's bitmap.asm and bitmap-demo.asm were drawn from to make the MIPS program.

"Learn C the Hard Way" was used to help format the C program's scanf statements.