CS 3220 Assignment 4 Report

William Bolton
David Benas

For our implementation we provided several regs to store the values of p0x, p0y, p1x, p1y, dx, dy, d, incrE, and incrNE in order to follow the Bresenham algorithm discussed in class. As a result, our algorithm involves no floating point operations. This being said, our implementation involves multiple extra regs for the purpose of storing values used for special cases when x2 < x1, y2 < y1, steep slopes, and reset.

In the initial phase, all temp values are set to the initial values of x1... y2. We first check if x2 < x1, in this case we swap the two. Dy is then calculated as an absolute value – if y1 < y2, dy = y2 – y1, else it is y1 – y2. We then set our extra regs dy2, dx2 to equal dy and dx respectively for the purpose of comparisons later. Then, if dy > dx (that is, the slope m > 1), we perform appropriate swaps in order to draw the line correctly. After this, another check must be made in the case that it is both steep and the slope is negative – we make the appropriate swaps. Finally, we calculate d according to the algorithm, along with incrE, incrNE, and set values labeled reset_p0x... reset_p1y to the final values of p0x... p1y in the case of a reset. In addition, to avoid (most) random pixels or the first line of the screen being jibberish, we set a screen_clear reg = 0.

The writing to screen algorithm naturally first checks if the reset button has been pressed. In this case, the GPU_ADDR is reset, and the background color is set to black. Additionally, screen_clear is set to 0, and p0x... p1y are reset to their original values (reset_p0x... reset_p1y mentioned above). In the case that reset is not pressed, the next check is that screen_clear < 2 && !I_VIDEO_ON. Screen_clear's value is important for clearing the screen before drawing the line. This has an important impact for many reasons. For one, if the screen is not drawn to completely with a specific color in between changing the slope of the line, pixels from a previous line will still be present on the screen. Next, the screen must be written to more than once (hence the screen_clear < 2), because we found that without this check, the first line of the screen would be jibberish. Our guess (which might not be correct) is that the row is being incremented during the first clock cycle, therefore the first row has not been defined. Nevertheless, clearing the screen more than once seems to do the trick.

Finally, after the screen has been cleared more than once, we begin drawing the line. We first check if dy < dx and then if x1 < x2, then we follow the Bresenham algorithm accordingly, checking if the current pixel equals a pixel contained in the line. If It does, we change the pixel on the screen to be yellow (yellow on black, gatech, right?). We must check if x1 < x2, otherwise our algorithm is finished and we have no reason to continue drawing the line. If dy > dx, we follow the appropriate procedures for drawing a steep line.

The only performance constraint in our algorithm happens when we draw a line with a negative slope that is not steep. In this case the line is drawn from left to right with decreasing y values. The line is drawn, but it is not instant. This is because of the manner that our x1... y2 variables are incremented and decremented – the first pixel check happens at the lower left part of the line, but at this point any row above this row has already been drawn black, so the screen must make as many cycles until equal to dy in order to complete the line.