# CS4481 Assignment #1

## PROGRAM-1

Program-1 generates a PBM image that has a black border and two diagonal lines that form an X-intersection in the center of the image. Figures 1, 2, and 3 show an average use of the program, and figures 2 and 3 show two edge cases where the minimum heights and widths are used.

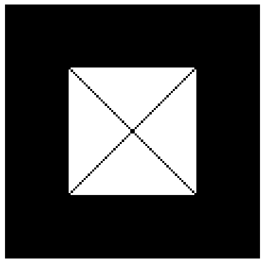

*Figure 1: pbm_120x120_ascii.pbm.* PBM, width-120, height-120, ASCII. In this square, 120x120 image, the borders take up 25% of the pixels on the top, bottom, left, and right (30 pixels all around). In the center, the two diagonal lines form an X-intersection.

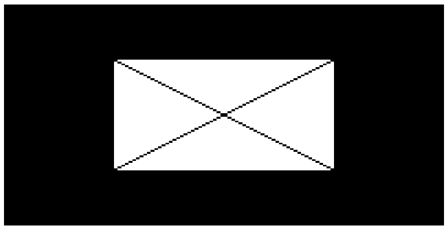

*Figure 2: pbm_240x120_ascii.pbm.* PBM, width-240, height-120, ASCII. This is an example to show how in horizontally longer images, the slope of the lines adjust to still hit all corners of the white box. Also, the border on the left and right become wider to still consume 25% of the image's width each.

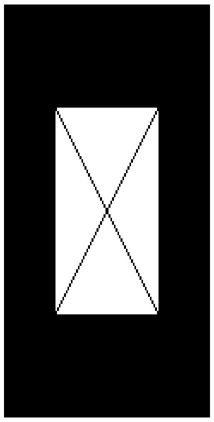

*Figure 3: pbm_120x240_ascii.pbm.* PBM, width-120, height-240, ASCII. This is an example to show how the slope adjusts in vertically longer images to still hit all corners of the white box. Also, the border on the top and bottom become wider to still consume 25% of the image's height each.

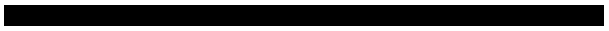

*Figure 4: pbm_120x4_ascii.pbm.* PBM, width-120, height-4, ASCII. In this 120x4 pixel image, the border on the left and right takes up 30 pixels each respectively, the border on the top and bottom take up 1 pixel each respectively, and the diagonal lines fill up the remaining white space in the image. The image is entirely black because there are only 2 pixels of space for the 2 diagonal lines, and each line is 1 pixel wide.

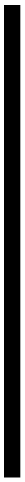

*Figure 5: pbm_4x120_ascii.pbm.* PBM, width-4, height-120, ASCII. In this 4x120 pixel image, the border on the top and bottom takes up 30 pixels each respectively, the border on the left and right take up 1 pixel each respectively, and the diagonal lines fill up the remaining white space in the image. The image is entirely black because there are only 2 pixels of space for the 2 diagonal lines, and each line is 1 pixel wide.

## Program-2

Program-2 generates an image that divides a rectangle into 4 triangles. The color of each triangle from base to tip is a gradient from white to black. The image also has a black border along the edges. Figures 6, 7, and 8 show an average use of the program, and figures 9 and 10 show two edge cases where the minimum heights and widths are used.

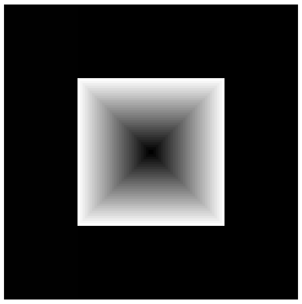

*Figure 6: pgm_120x120_ascii.pgm.* PGM, width-120, height-120, ASCII. In this square, 120x120 image, the borders take up 25% of the pixels on the top, bottom, left, and right (30 pixels all around). The square is divided into 4 triangles colored white at the base, but get darker towards the tip. In the center of the image the color will be completely black.

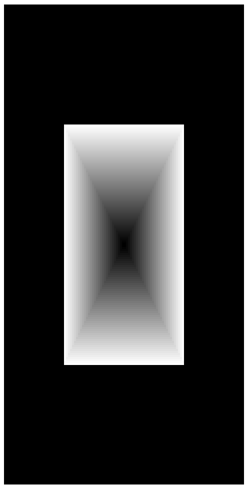

*Figure 7: pgm_120x240_ascii.pgm.* PGM, width-120, height-240, ASCII. This is an example to show how the program adjusts to generate vertically longer images. The top and bottom borders adjust to take up 25% of the image's height each.

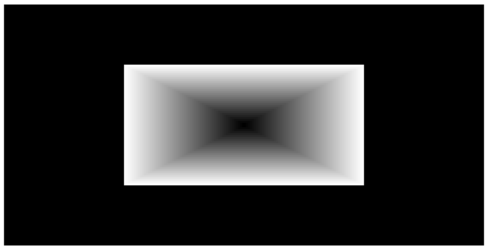

*Figure 8: pgm_240x120_ascii.pgm.* PGM, width-240, height-120, ASCII. This is an example to show how the program adjusts to generate horizontally longer images. The left and right borders adjust to take up 25% of the image's width each.

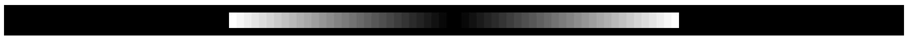

*Figure 9: pgm_120x4_ascii.pgm.* PGM, width-120, height-4, ASCII. In this rectangular, 120x4 image, the borders take up 25% of the width and height, on the top/bottom and left/right side respectively. This means that the left and right borders are 30 pixels each, and the top and bottom borders are 1 pixel each. In this case where the height is at the minimum of 4, the "triangles" are essentially nonexistent, because the top and bottom triangles would be confined to 1 pixel each, and they are right in the center, making them only one color.

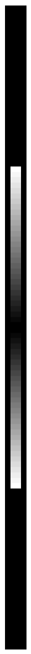

*Figure 10: pgm_4x120_ascii.pgm.* PGM, width-4, height-120, ASCII. With the same logic as the 120x4 image, the borders take up 25% of the top/bottom or left/right sides depending on the height and width respectively. Due to the narrow working area of 2x60 pixels, similar to the 120x4 image, the four triangles are nonexistent, and it results in a gradient from white to black to white again.

## PROGRAM-3

Program-3 generates an image with 5 sections. The first three sections use half the height of the image and a third of the width each to show gradients from red to white, white to green, and blue to white. On the lower half, two sections take up the remaining two quarters of the image, showing a gradient from black to white, and white to black. Figure 13 represents an average image generated by program-3, and figure 14 shows what happens when you isolate the red, green, or blue values in the conversion to a PGM image.

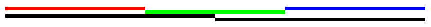

*Figure 11: ppm_120x4_ascii.ppm.* PPM, width-120, height-4, ASCII. In this particular case, each gradient is only given 2 pixels vertically, so the change in color is instant as opposed to gradual.

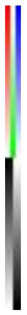

*Figure 12: ppm_6x120_ascii.ppm.* PPM, width-6, height-120, ASCII. In this case, the gradients are given more vertical room to change their color, but they are limited horizontally. The top colors each have a width of two, and the bottom gradients have a width of three each.

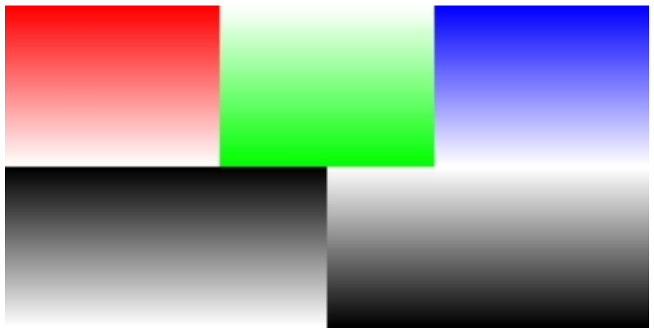

*Figure 13: ppm_240x120_ascii.ppm.* PPM, width-240, height-120, ASCII. This image shows an average use case of program-3.

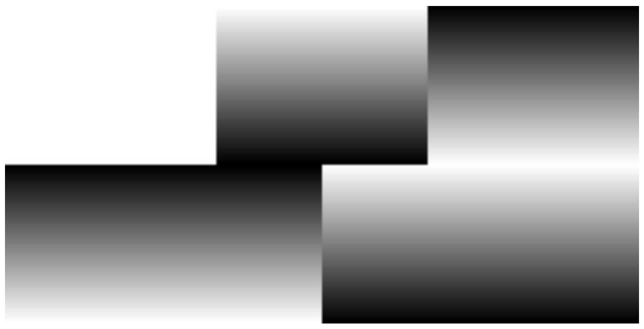

*Figure 14: red_value_only_240_120_ascii_ppm_to_pgm.pgm.* This image is generated alongside figure 13. The color gradient in figure 13 is accomplished by keeping the focus color (in this case red) at its maximum value, while decreasing the values of other colors. By isolating the red value in the conversion to a PGM image, this highlights how the value of red stays constant in the section that its gradient belongs to. This logic applies to the blue and green isolated PGM files as well.