CS430 Homework Assignment 1

Program problem:

1. **Understand the PBM file format.**
    1. **The description of the Plain PBM format can be found here (for assignments 1 --> 4 and the Extra Credit assignment).**
    2. **The description of the Plain PPM format can be found here (for assignment 5).**
    3. **Here are a PBM file and a PPM file.**

2. **Understand the Simplified Postscript file format (*.ps)**
    1. **The input file will be a Postscript file, but your program only needs to handle the following command**
    2. **x1 y1 x2 y2 Line**
    3. **The text of these commands will be bounded by two delimiters where %%%BEGIN marks the beginning, and %%% END marks the end.**
    4. **These tokens will both appear at the beginning of the lines they occur on.**
    5. **There may be blank lines anywhere between these delimiters.**
    6. **ALL text outside of these delimiters should be ignored by your program.**
    7. **Postscript assumes that the origin (0,0) is located in the lower left corner of the window.**

    The purpose of using a Postscript-like format is simply to give you an idea of what the input should look like before you start implementing a reader.
    You are not actually implementing a full blown postscript reader. An example of the simplified format is given below:

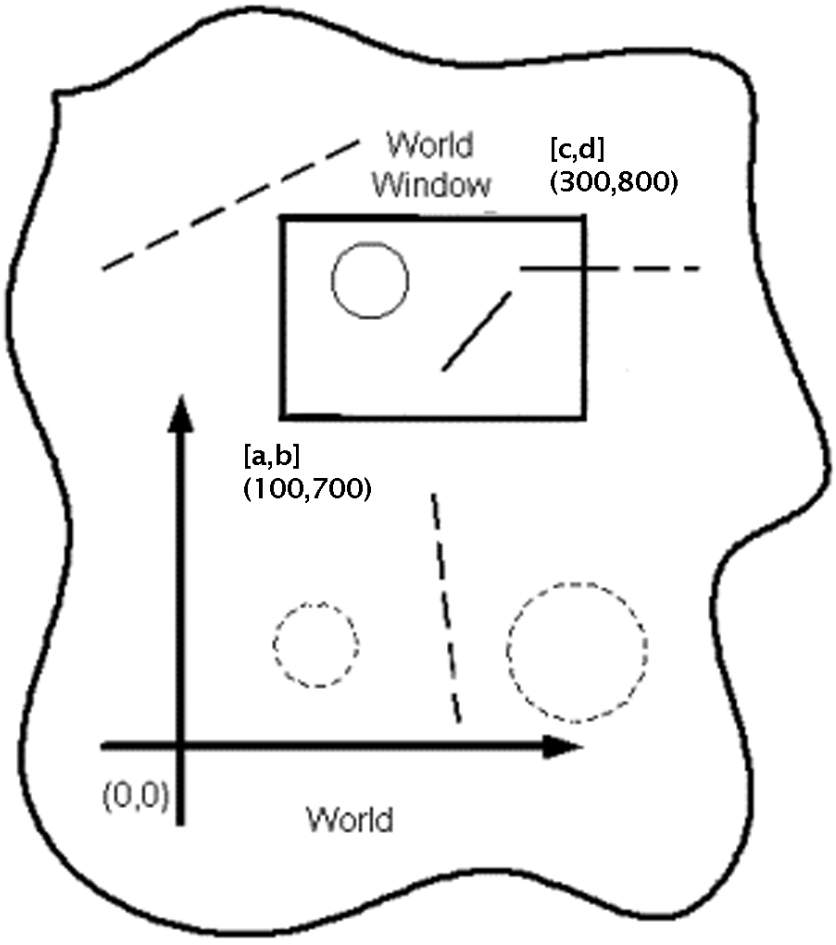
    %%%BEGIN
    375 100 300 230 Line
    499 0 0 250 Line

    170 450 400 350 Line
    350 300 120 400 Line
    %%%END

    An example input file. This file can be used to visualize arbitrary input data, in our simplified format, within the world window.
    A free cross-platform tool, gsview, for viewing Postscript. There are other known free viewers such as gv (Ghostview),
    and ggv(GNOME/GTK Ghostview).

3. **Write a program, called CG_hw1 and in the language of your choice, that accepts our simplified Postscript-like format file as input and generates a PBM image as output. Example input file here.**
    1. **The only Postscript command you need to implement for this homework assignment is "Line".**
    2. **You will have to implement the DDA or Bresenham algorithm for scan-conversion of lines.**
    3. **Clip your lines to the world window with the Cohen-Sutherland algorithm.**
    4. **The output image (screen) dimensions/resolution will be determined by the -[abcd] parameters (i.e. the size/shape of your world window).**
    5. **You must handle the following command-line options. You should be able to process any subset of the options in any arbitrary order. *Default values that should be implemented in your code are in bold.***

6. **[-f] The next argument is the input "Postscript" file. (hw1.ps)**
7. **[-s] This next argument is a float specifying the scaling factor in both dimensions about the world origin. (1.0)**
8. **[-r] This next argument is an integer specifying the number of degrees for a counter-clockwise rotation about the world origin. (0)**
9. **[-m] The next argument is an integer specifying a translation in the x dimension. (0)**
10. **[-n] The next argument is an integer specifying a translation in the y dimension. (0)**
11. **Assume that the objects of your scene are scaled, then rotated and finally translated in the world coordinates.**
12. **[-a] The next argument is an integer lower bound in the x dimension of the world window (0)**
13. **[-b] The next argument is an integer lower bound in the y dimension of the world window (0)**
14. **[-c] The next argument is an integer upper bound in the x dimension of the world window (499)**
15. **[-d] The next argument is an integer upper bound in the y dimension of the world window (499)**
16. **You may assume that a will always be less than c, and that b will always be less than d.**
17. **There are no limits on the input values. They can be positive and negative.**
18. **Your program should print output images in the PBM file format to stdout (cout); debugging messages should be printed to stderr (cerr).**
19. **All the pixels should be initialized to white. Draw your objects in black.**



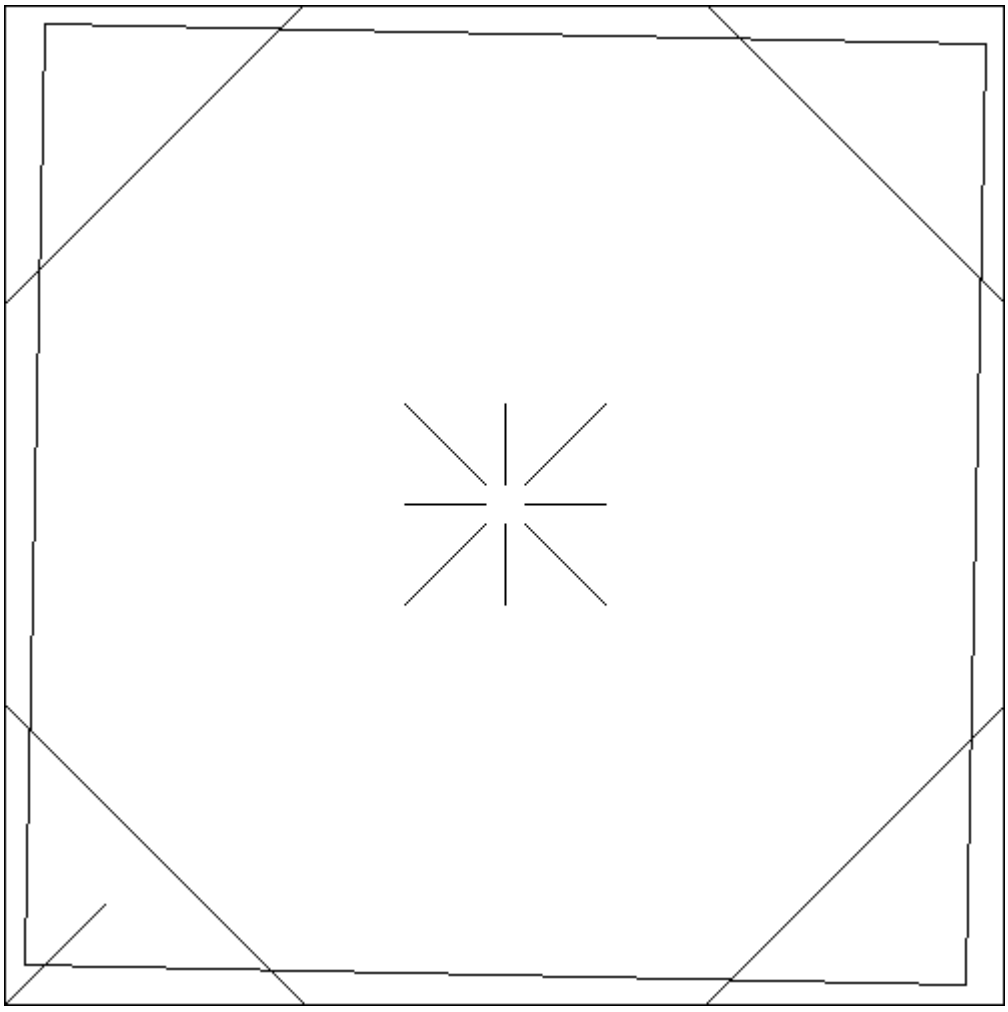
20. **Steps in assignment**
    1. **Read in line segments**
    2. **Apply transformations to them in world coordinates**
    3. **Clip transformed lines to window**

4. **Translate lines into screen/image coordinates**
5. **Scan convert (i.e. draw) clipped lines into software frame buffer**
6. **Write frame buffer to standard out in PBM format**

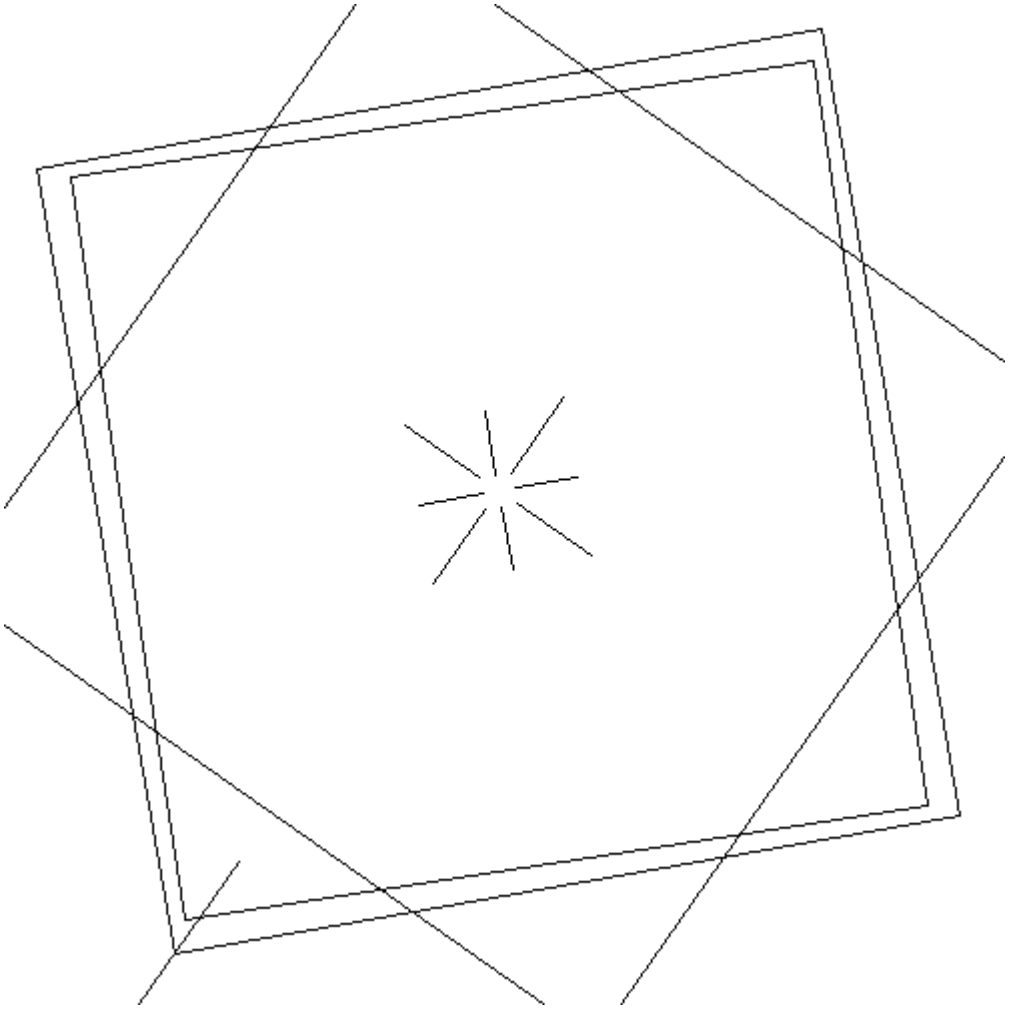
21. **If the tester chooses not to enter any options and thus enter ./CG_hw1 > out.pbm, your program should produce the same results as:**

    ./CG_hw1 -f hw1.ps -a 0 -b 0 -c 499 -d 499 -s 1.0 -m 0 -n 0 -r 0 > out.pbm

    and generate the following image.



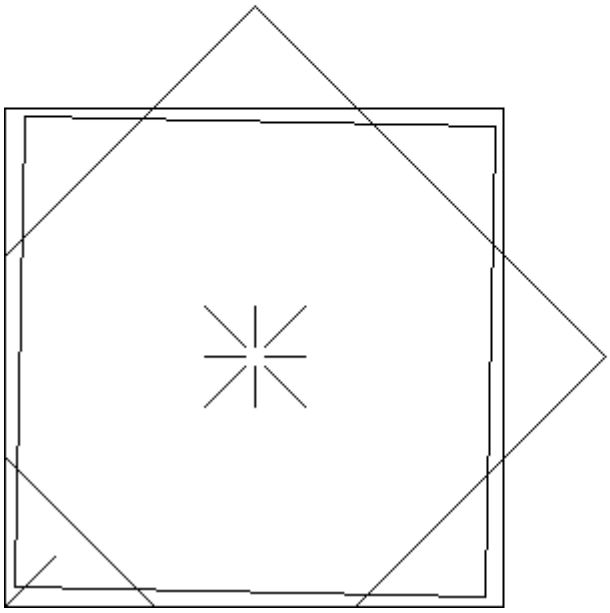
22. **Input/Output Example:**
    i. The input was: ./CG_hw1 -f hw1.ps -a 0 -b 0 -c 499 -d 499 -s 0.8 -m 85 -n 25 -r 10 > hw1.pbm
    ii. The output was:



23. Input/Output Example:

i. The input was: ./CG_hw1 -f hw1.ps -s 0.5 > scale.pbm

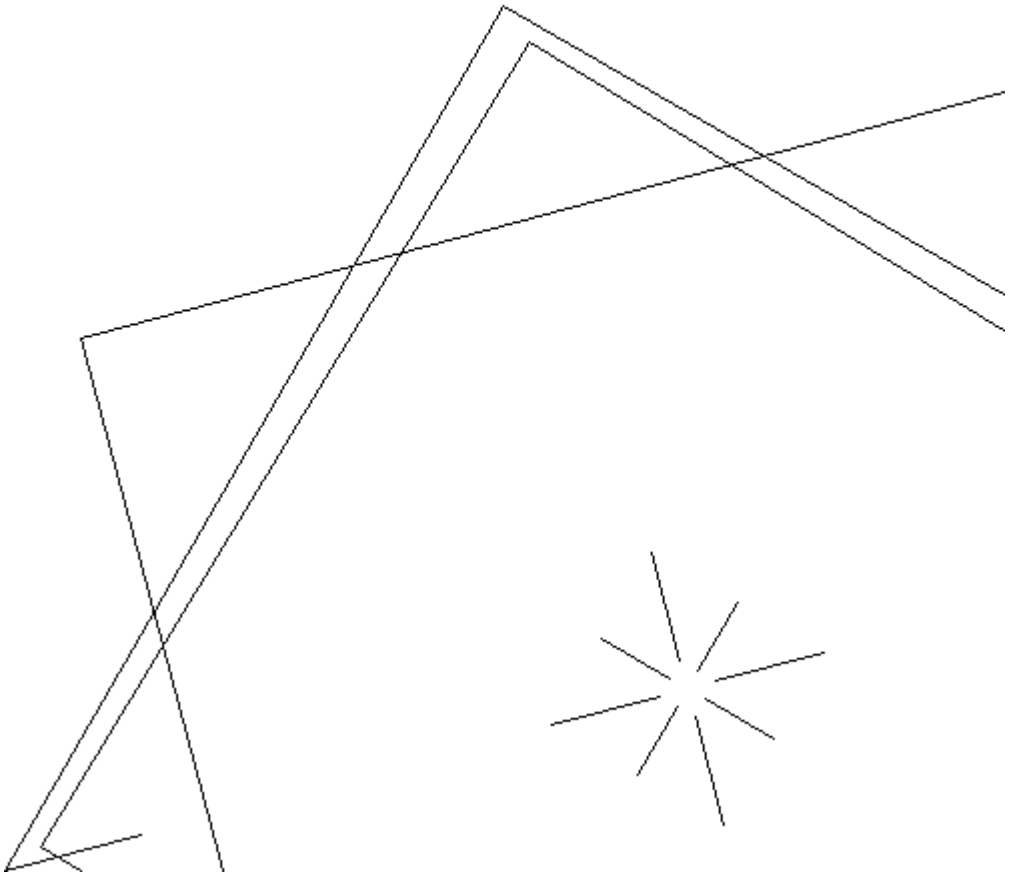
ii. The output was:



24. Input/Output Example:

i. The input was: ./CG_hw1 -f hw1.ps -r -30 > nrotate.pbm

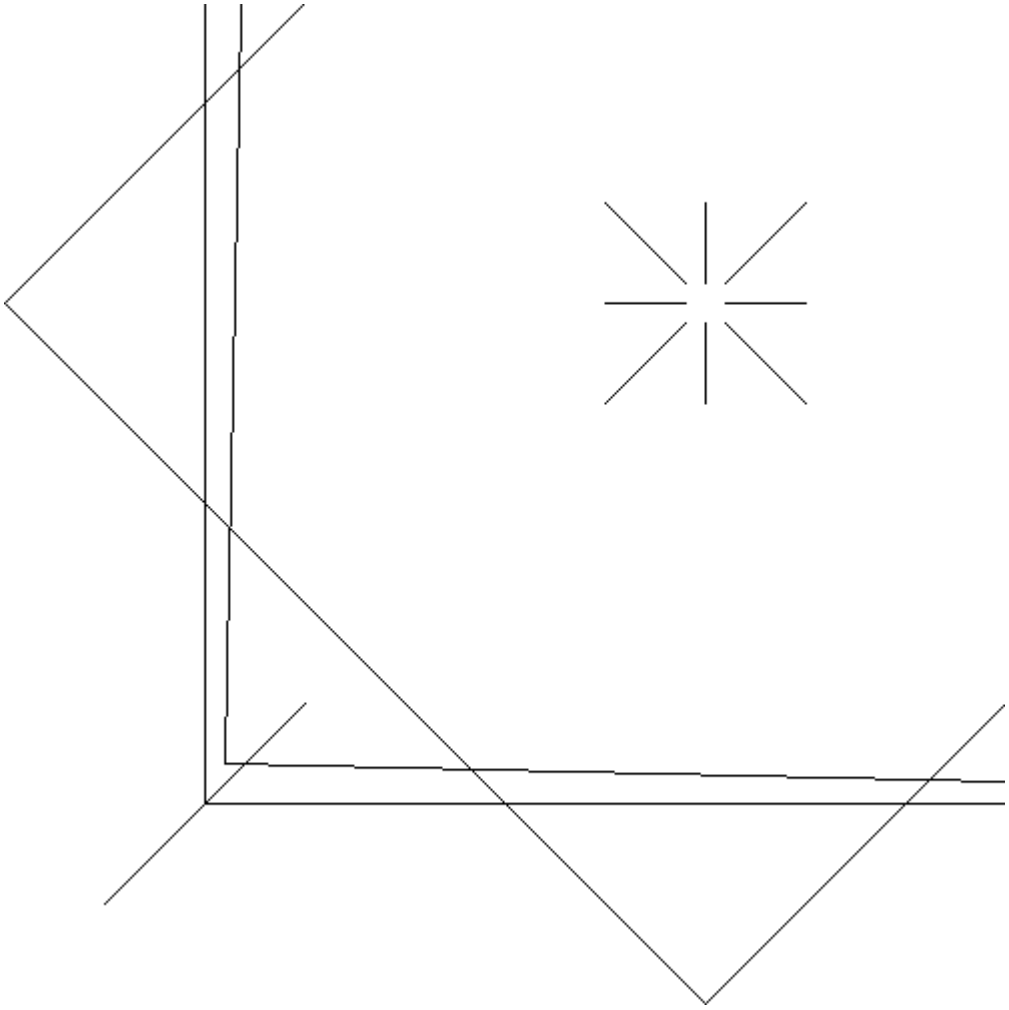
ii. The output was:



25. Input/Output Example:

i. The input was: ./CG_hw1 -f hw1.ps -m 100 -n 100 > translate.pbm

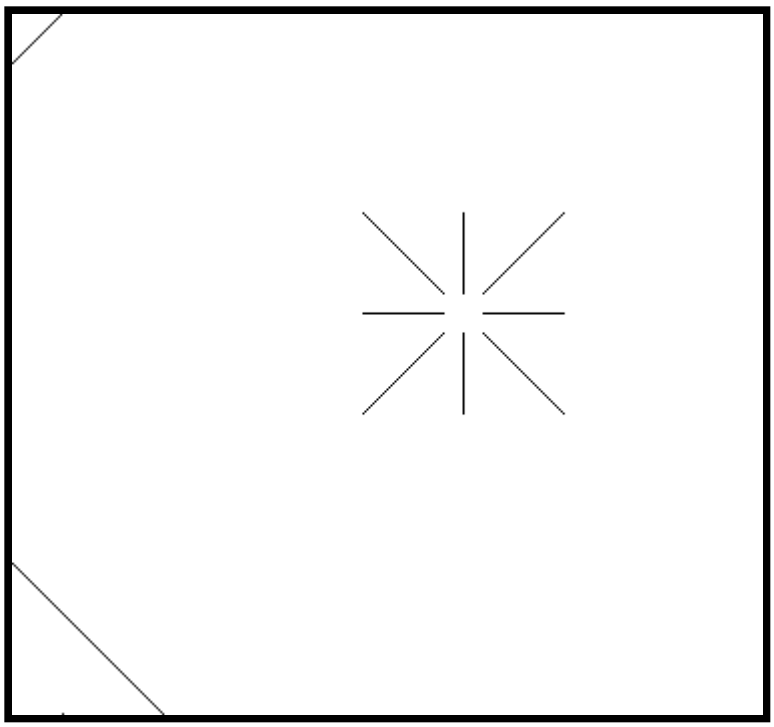
ii. The output was:



26. Input/Output Example: (Dark border is the window boundary)

i. The input was: ./CG_hw1 -f hw1.ps -a 25 -b 50 -c 399 -d 399 > worldwindow.pbm

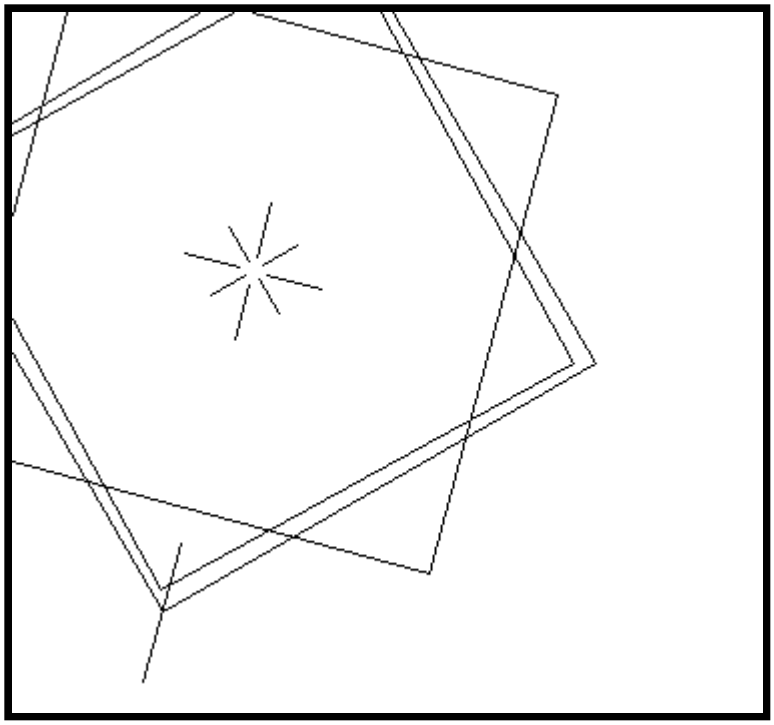
ii. The output was:



27. Input/Output Example: (Dark border is the window boundary)

i. The input was: ./CG_hw1 -f hw1.ps -a 25 -b 50 -c 399 -d 399 -r 30 -m 100 -n 100 -s 0.5 > all.pbm

ii. The output was:



4. Grading Scheme

1. **Command-line argument reading : 1 point**
2. **Postcript reading : 1 point**
3. **Image format : 1 point**
4. **Correct window/image dimensions : 1 point**
5. **Line drawing : 2 points**
6. **Clipping : 2 points**
7. **Transformations : 2 points**

5. Submission Guidelines:

1. **Assignments must be submitted via Bb Learn.**
2. **README file: explain the features of your program, language and OS used, compiler or interpreter used, name of file containing main(), and how to compile/link your program. Text files only. Word and PDF documents will NOT be accepted.**
3. **All source code. Your code must compile and run on tux (Linux).**
4. **You may program in any language you like as long it can produce a usable executable named CG_hw1 on tux.**
5. **Your program will be run by the TA. Please do NOT submit any image files, Visual C++ project files, or anything not requested in this section. Your program must run on tux without the installation of "special" libraries.**
6. **Makefile: have the default rule compile your program.**
7. **If you are using a language that doesn't produce an executable file, e.g. python, then be sure to include a script called CG_hw1 that accepts arguments and prints PBM to standard out.**
8. **Points will be deducted if submission guidelines are not followed.**
9. **Further details about Bb Learn**

1. **You can reach Bb Learn through DrexelOne.**
2. **Choose Computer Graphics among your list of courses. There is an "assignments" link in the left frame which will give you the list of assignments in the right frame.**
3. **Click on the assignment you wish to submit.**
4. **Find your file and click Upload button.**
5. **Hit Submit button. DO NOT FORGET TO HIT THE SUBMIT BUTTON AFTER YOU UPLOAD ALL YOUR FILES.**

NOTE: Your source code for all programming assignments will be run through a plagiarism detection system. This program uses compiler techniques which are invariant of syntax and style. If you are sharing code with other classmates, you will get caught. Please refer to the student handbook for actions that will be taken.

Last modified on September 24, 2020.