CS 430 – Computer Graphics
Fall 2016
Assignment 5 – 2D Transformations and Viewports

From your previous assignment(s) in this assignment you should be able to:

1. Read in data from a file pertaining to creating polygons, and organize that data into structures for drawing polygons.
2. Set up a software frame buffer to render to
3. Clip polygons to a clipping window.
4. Perform scan filling to draw filled polygons into the software frame buffer.
5. Output the software frame buffer as an XPM file.

In addition, in this assignment you should be able to:

1. Perform 2D transformations on objects using homogenous coordinates.
2. Change the clipping world window dimensions.
3. Map the clipping world window to a viewport within the frame buffer.

Make sure you give yourself adequate time. The programming components in particular can be quite time consuming.

As a reminder you may use the programming language of your choice, though I recommend C/C++ and also make sure that your program can run on the Drexel tux cluster to insure its system independence.

Submission Guidelines

1. Assignments must be submitted via Bd Learn
2. Submit a single compressed file (zip, tar, etc..) containing:
    a. A PDF file with solutions to the theory questions.
    b. A README text file (**not** Word or PDF) that explains
        i. Features of your program
        ii. Language and OS used
        iii. Compiler or interpreter used
        iv. Name of file containing main()
        v. How to compile/link your program
    c. Your source files and any necessary makefiles, scripts files, etc… to compile and run your program.

Theory Questions

1. What are the matrices for a homogenous 2D translation and a homogenous 2D scale transformation? (4pts)

   Translation:

   $$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

   scale:

   $$\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Describe the difference between the world window, the viewport, and the image/screen window (3pts)

   A world window is the rectangle in the world defining the region that is to be displayed.

A viewport is the rectangular portion of the interface window that defines where the image will actually appear. It is the sub-window that we render to on the screen/framebuffer.

The image/screen window is the space which the image is displayed

Assignment Details

We're now going to add a lot of parameters to our program!

Write a program that accepts the following command arguments. Defaults are in parenthesis. You should be able to process any subset of the options in any arbitrary order.

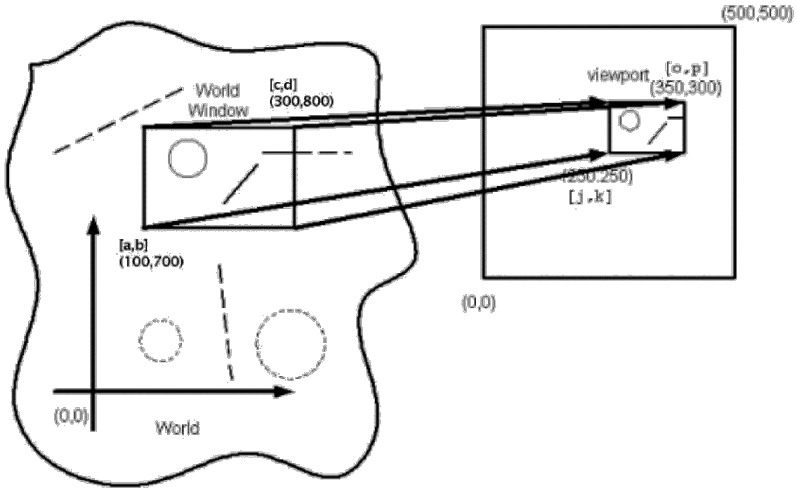
a. [-f] The next argument is the input "Postscript" file (hw5_1.ps)
b. [-s] The next argument is a float specifying the scaling factor in both dimensions about the world origin. (1.0)
c. [-r] The next argument is an integer specifying the number of degrees for a counter-clockwise rotation about the world origin. (0)
d. [-m] The next argument is an integer specifying a translation in the x dimension. (0)
e. [-n] The next argument is an integer specifying a translation in the y dimension. (0)
f. [-a] The next argument is an integer lower bound in the x dimension of the world window. (0)
g. [-b] The next argument is an integer lower bound in the y dimension of the world window. (0)
h. [-c] The next argument is an integer upper bound in the x dimension of the world window. (499)
i. [-d] The next argument is an integer upper bound in the y dimension of the world window. (499)
j. [-j] The next argument is an integer lower bound in the x dimension of the viewport (0)
k. [-k] The next argument is an integer lower bound in the y dimension of the viewport (0)
l. [-o] The next argument is an integer upper bound in the x dimension of the viewport window (499)
m. [-p] The next argument is an integer upper bound in the y dimension of the viewport window (499)

Additional things to note:

The viewport coordinate system origin is in the lower left hand corner of your image. The X direction is right, the Y direction is up

Your frame buffer size should be 500x500

You may assume that the viewport parameters are greater than or equal to zero and less than 500



Note: Although on this image the framebuffer goes to (500,500) in ours the last pixel is (499,499)

Your program should print output images in the XPM file format to stdout (`cout`) such that it can be piped to a file. All pixels should be initialized to white. Draw your objects in black.

Your general program flow should be:

1. Read in polygon specifying data (PS)
2. Apply 2D transformations to them in word coordinates
3. Clip against world window using Sutherland-Hodgman algorithm
4. Apply world-to-viewport transformations
    a. You could also apply the w-t-v transformation before clipping then clip against the viewport window
5. Round polygon vertices to integers
6. Scan fill transformed polygons into software frame buffer

Grading Scheme:

In order to get any credit at all you must be able to generate at least read in a PS file and write out a valid XPM file

1. Theory questions (7pts)
2. Can perform translations with clipping (15pts)
3. Can perform scaling with clipping (15pts)
4. Can perform rotation with clipping (15pts)
5. Can perform combination of transformations correctly (25pts)
6. Can perform world-window to viewport transformation correctly (23pts)

Common deductions:

1. Nothing drawn (-100pts)
2. Implementation not done in homogenous coordinates (-40pts)
3. Missing files (including readme) (-5pts each)
4. Cannot compile/run out-of-the-box on TUX (-10pts)

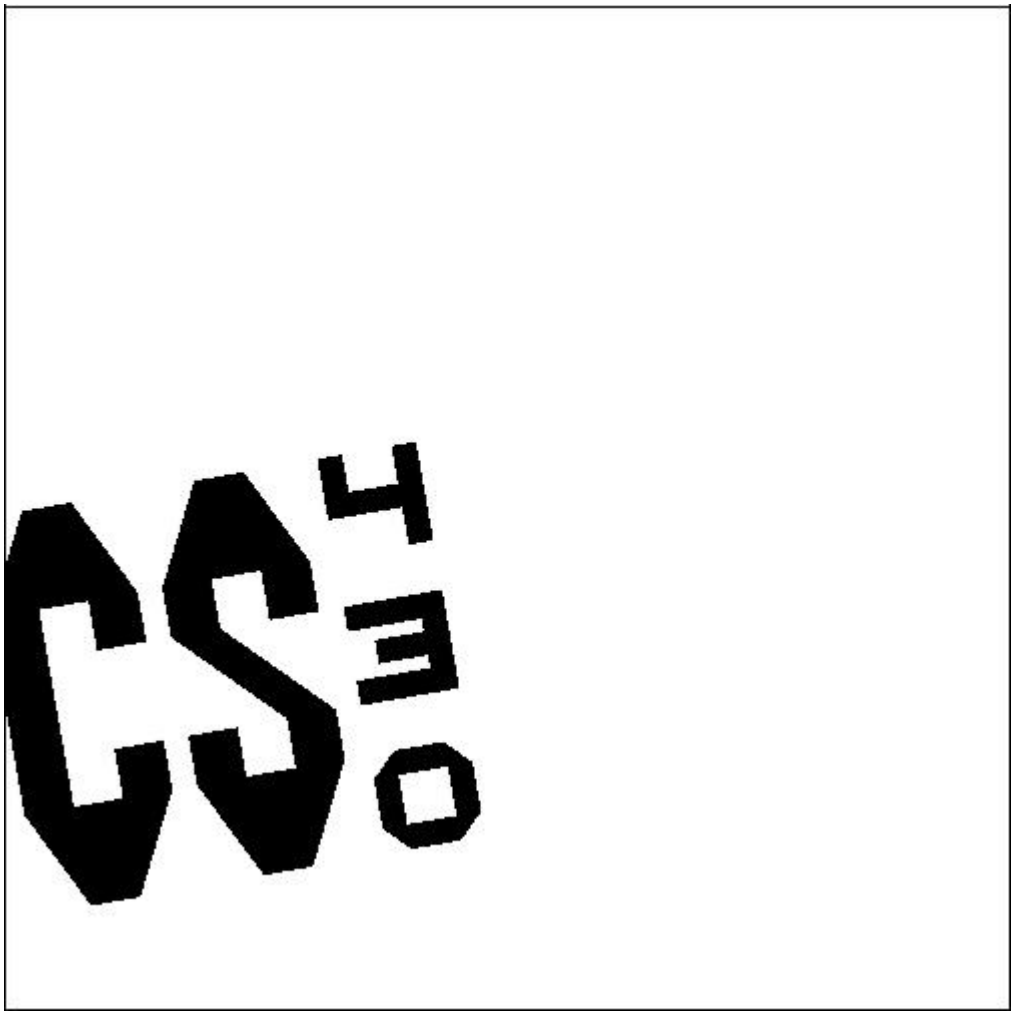
Provided Tests

For testing we'll use the last test from the previous homework (the filled in CS 430) image.
Here are tests of incremental difficulty/features

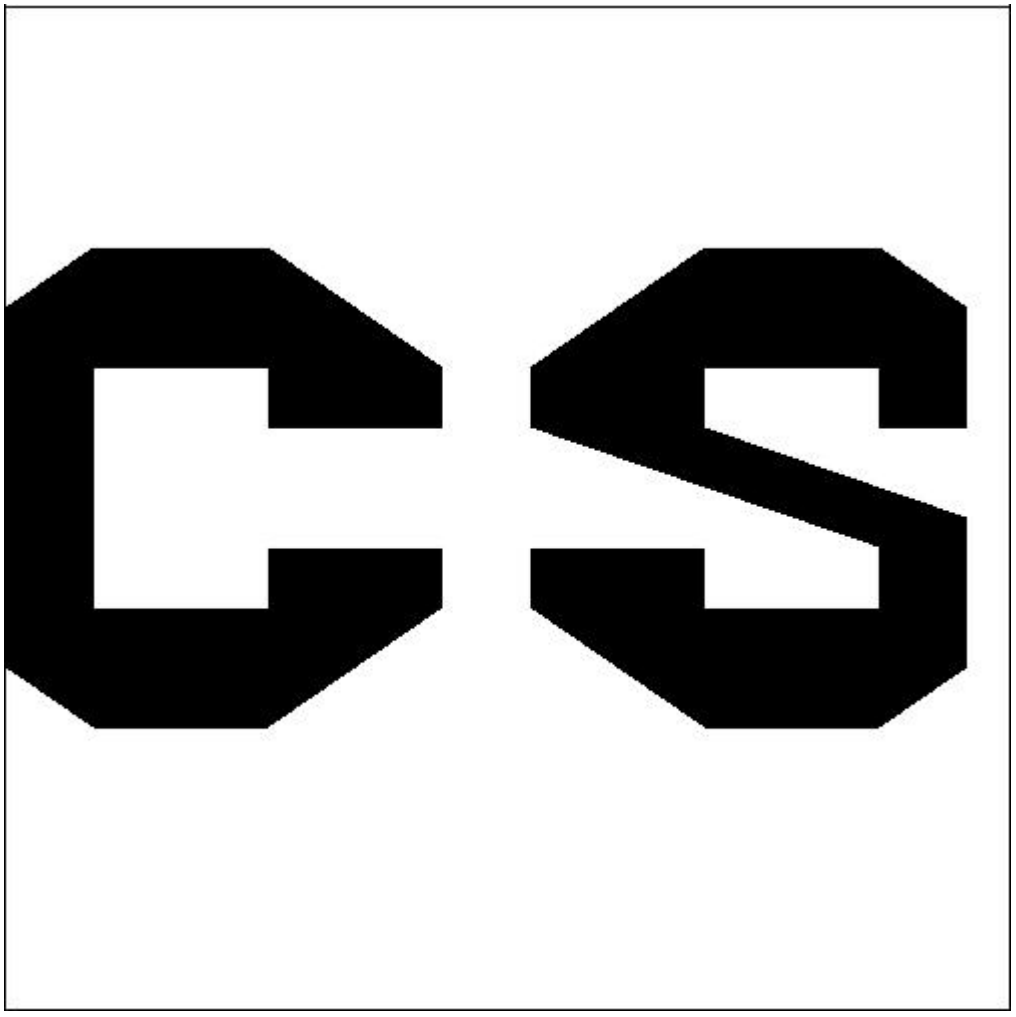
1. The default settings. Should be the same output as the last examlpe in HW4
   ./A5 -f hw5_1.ps -a 0 -b 0 -c 499 -d 499 -j 0 -k 0 -o 499 -p 499 -s 1.0 -m 0 -n 0 -r 0 > out1.xpm
2. Let's scale this object down by ½
   ./A5 -f hw5_1.ps -a 0 -b 0 -c 499 -d 499 -j 0 -k 0 -o 499 -p 499 -s 0.5 -m 0 -n 0 -r 0 > out2.xpm
3. Let's translate the original image by (30,50)
   ./A5 -f hw5_1.ps -a 0 -b 0 -c 499 -d 499 -j 0 -k 0 -o 499 -p 499 -s 1.0 -m 30 -n 50 -r 0 > out3.xpm
4. Let's rotate the original image 10 degrees about the origin:
   ./A5 -f hw5_1.ps -a 0 -b 0 -c 499 -d 499 -j 0 -k 0 -o 499 -p 499 -s 1.0 -m 0 -n 0 -r 10 > out4.xpm
5. Let's combine these by first scaling, then rotating, then translating
   ./A5 -f hw5_1.ps -a 0 -b 0 -c 499 -d 499 -j 0 -k 0 -o 499 -p 499 -s 0.5 -m 30 -n 50 -r 10 > out5.xpm
6. Using the original, non-transformed object, let's render the part of the world within a world window spanning from (50,0) to (325,500) into a viewport on the screen with vertices (0,110) to (480,410)
   ./A5 -f hw5_1.ps -a 50 -b 0 -c 325 -d 500 -j 0 -k 110 -o 480 -p 410 -s 1 -m 0 -n 0 -r 0 > out6.xpm
7. Then there's some example where we combine all this stuff…

./A5 -f hw5_1.ps -a 0 -b 0 -c 499 -d 499 -j 0 -k 0 -o 499 -p 499 -s 1.0 -m 0 -n 0 -r 10 > out4.xpm

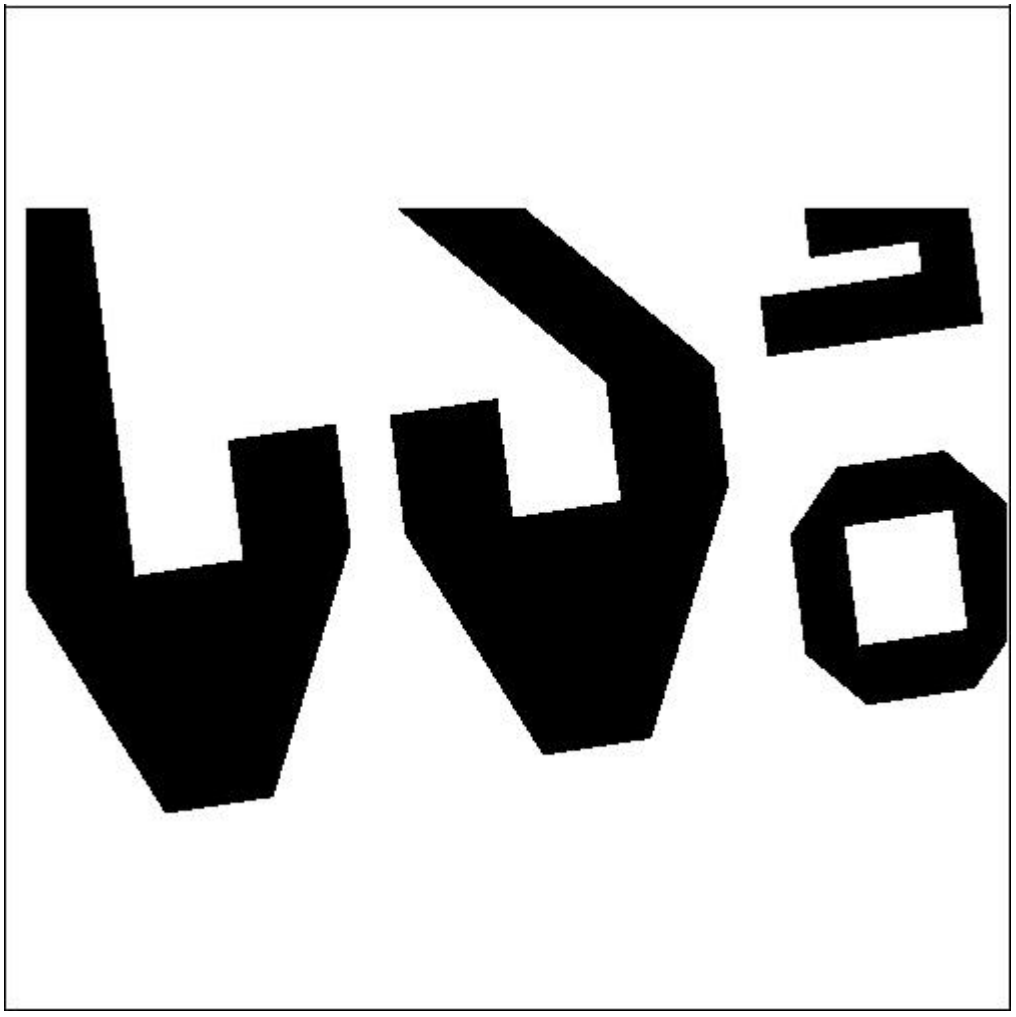
./A5 -f hw5_1.ps -a 0 -b 0 -c 499 -d 499 -j 0 -k 0 -o 499 -p 499 -s 0.5 -m 10 -n 20 -r 10 > out5.xpm



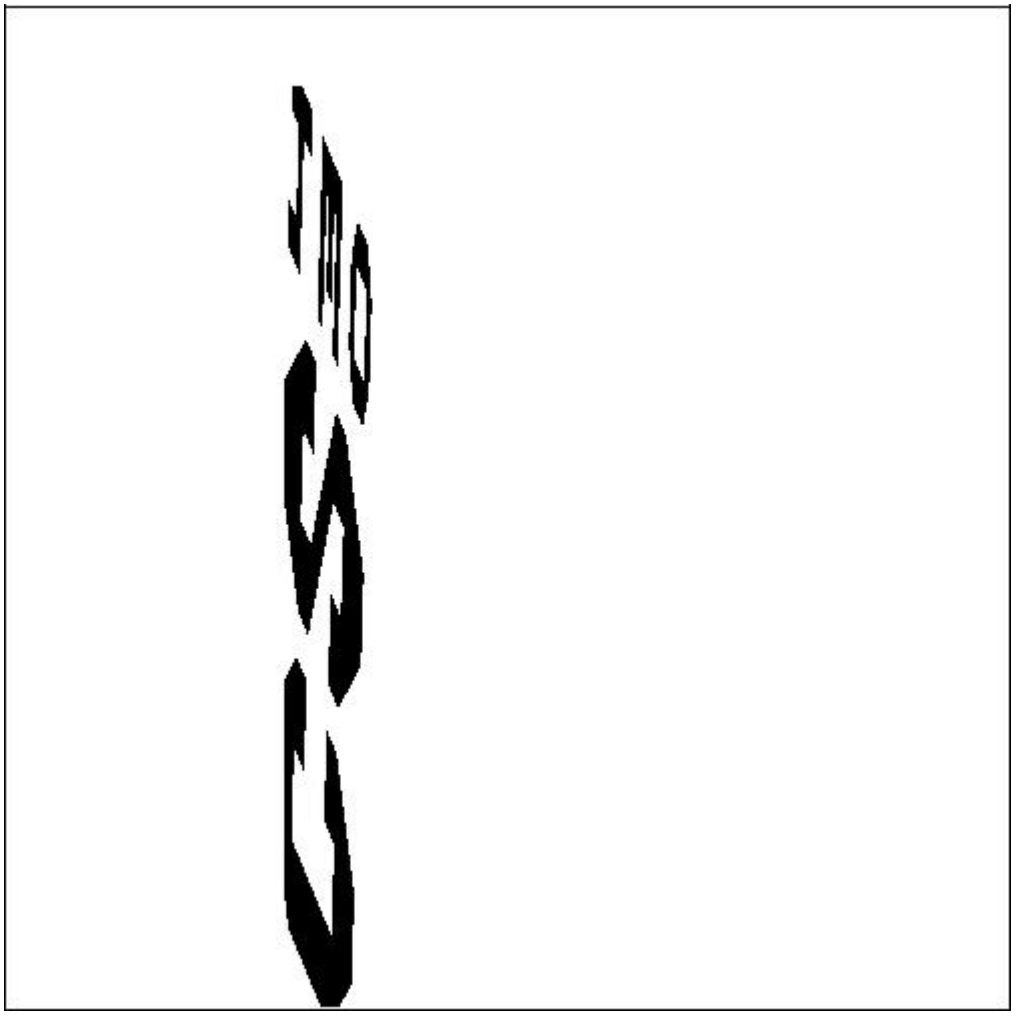
./A5 -f hw5_1.ps -a 50 -b 0 -c 325 -d 500 -j 0 -k 110 -o 480 -p 410 -s 1 -m 0 -n 0 -r 0 > out6.xpm



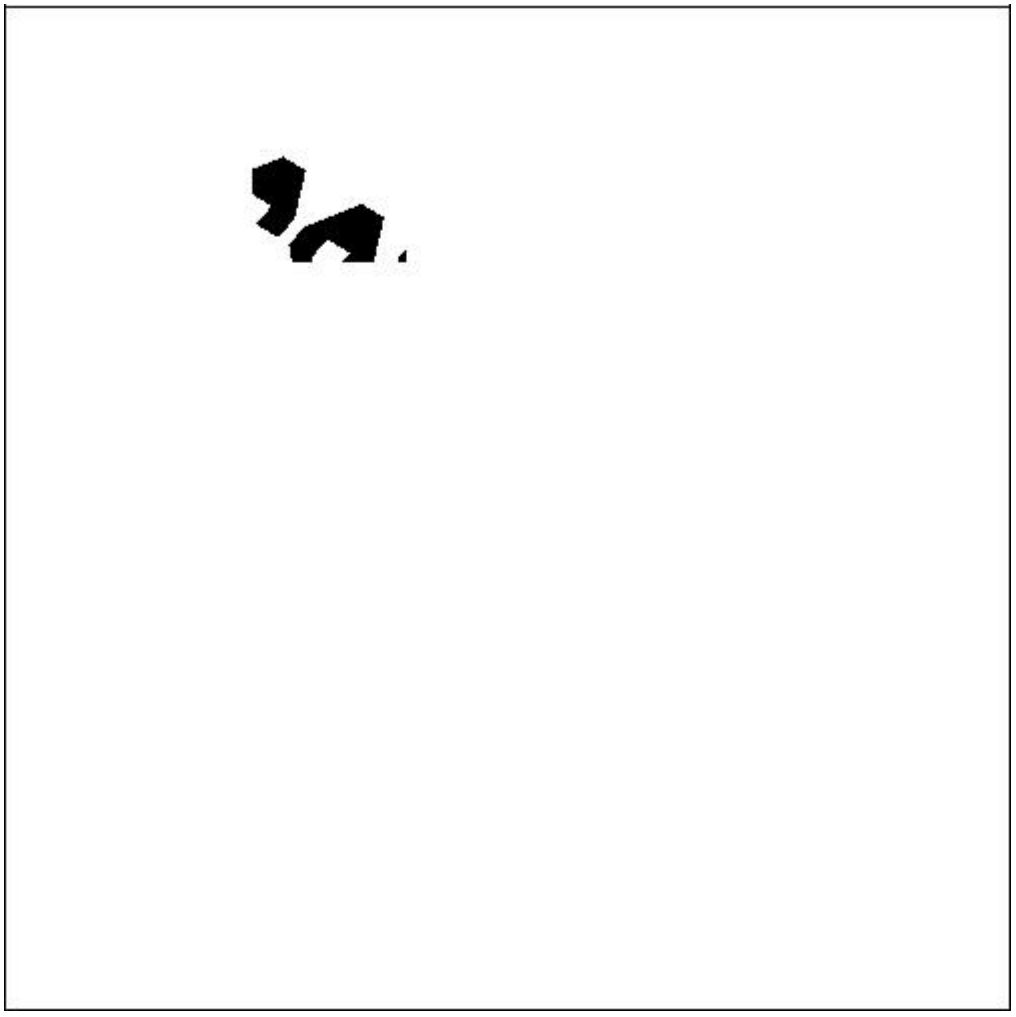
./A5 -f hw5_1.ps -a 10 -b 10 -c 550 -d 400 -j 10 -k 10 -o 499 -p 400 -s 1.2 -m 6 -n 25 -r 8 > out7.xpm



./A5 -f hw5_1.ps -b 62 -c 500 -d 479 -r 75 -j 139 -0 404 -p 461 -s .85 -m 300 > out8.xpm



./A5 -f hw5_1.ps -a 275 -b 81 -c 550 -d 502 -r -37 -j 123 -k 373 -p 467 > out9.xpm



./A5 -f hw5_1.ps -a -135 -b -53 -c 633 -d 842 -m -23 -j 101 -p 415 -s 3.6 > out10.xpm