MECE 4606 DIGITAL MANUFACTURING

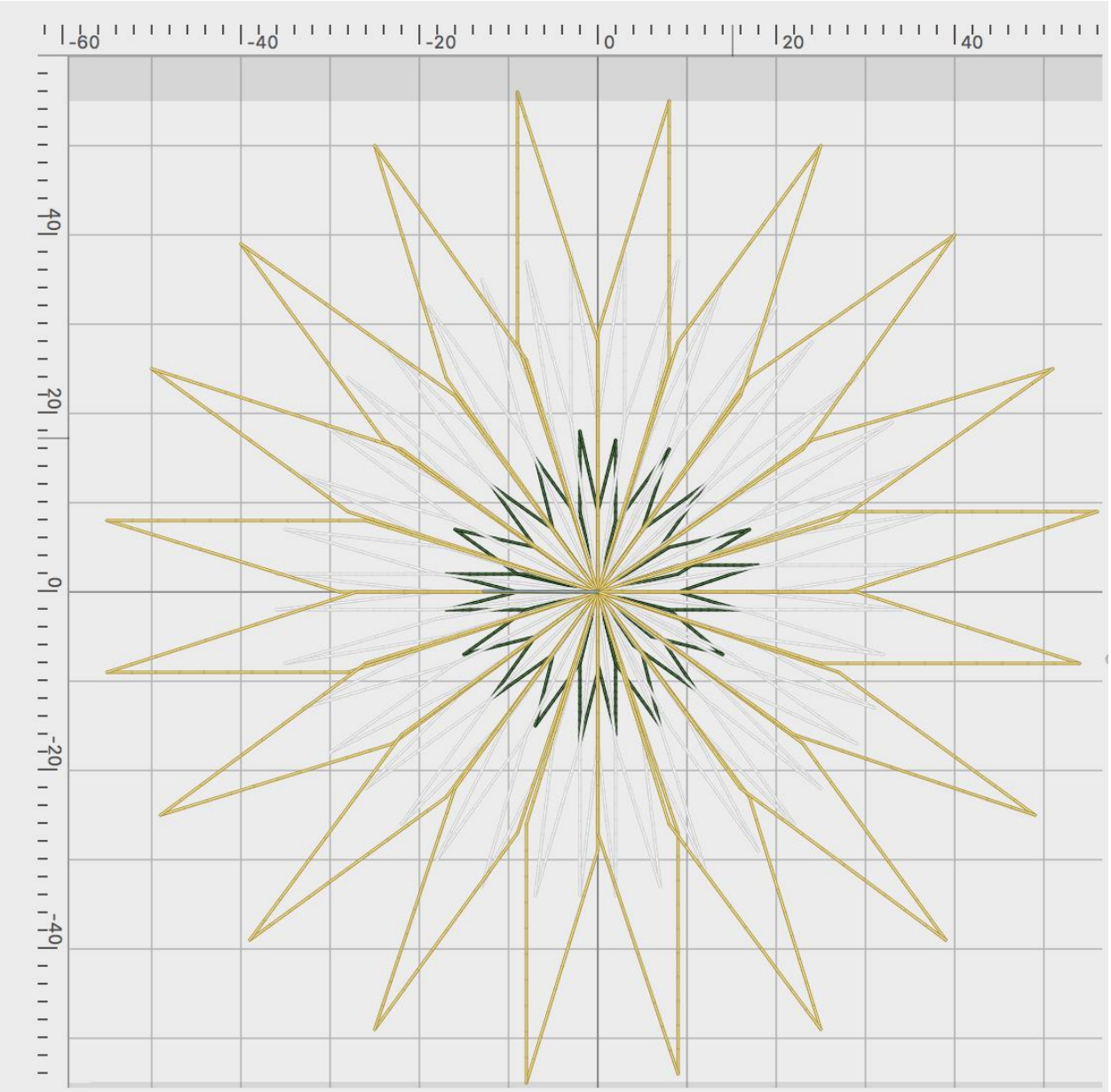
PROF. HOD LIPSON

Programmable Embroidery

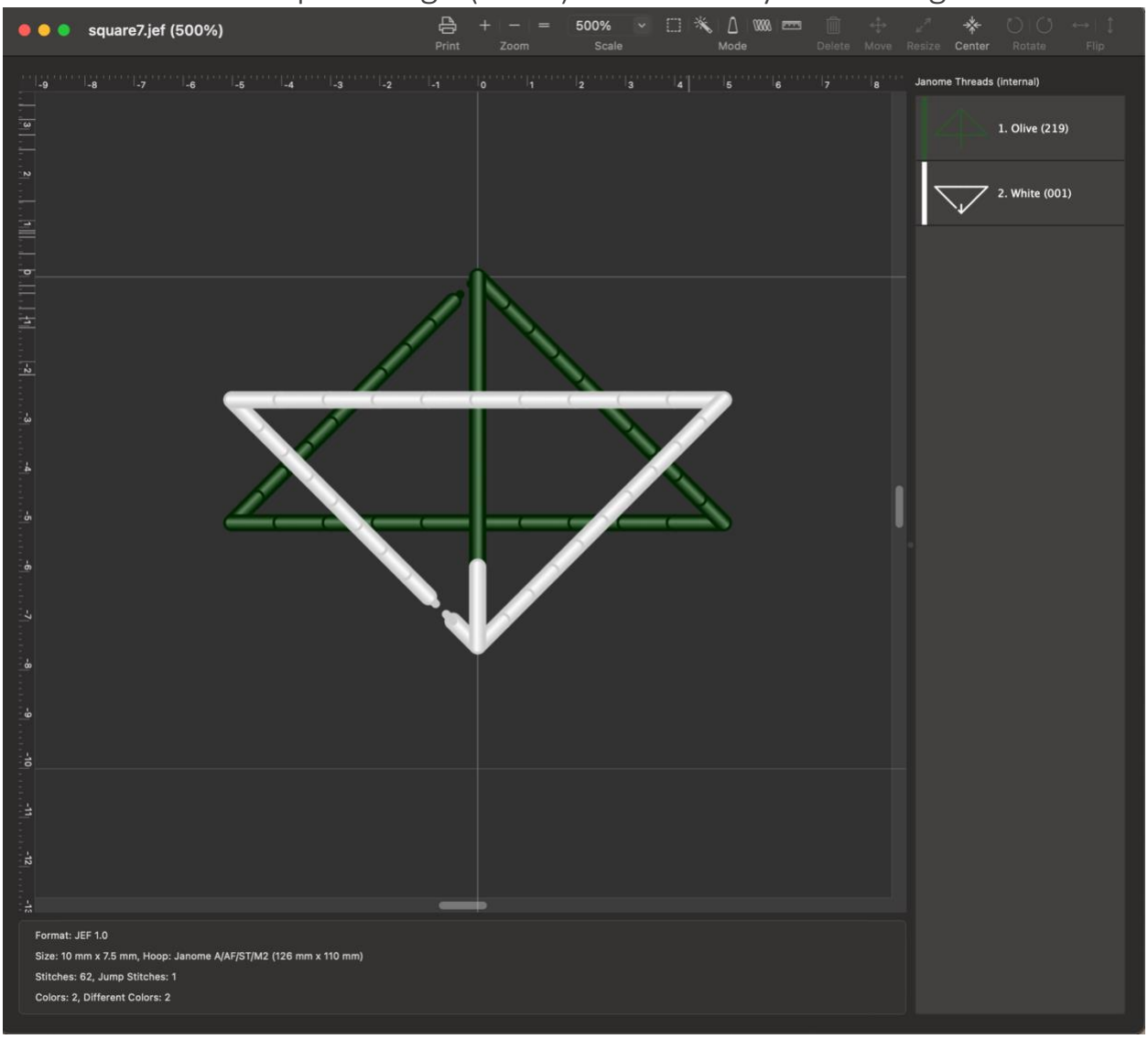
Rohan Sahu (rzs2120), Brock Taylor (btt2115)

March 13th: 9:30 PM

218 Grace Hours Before Submission, +2.5 Hours Gained, 220.5 Hours After Submission

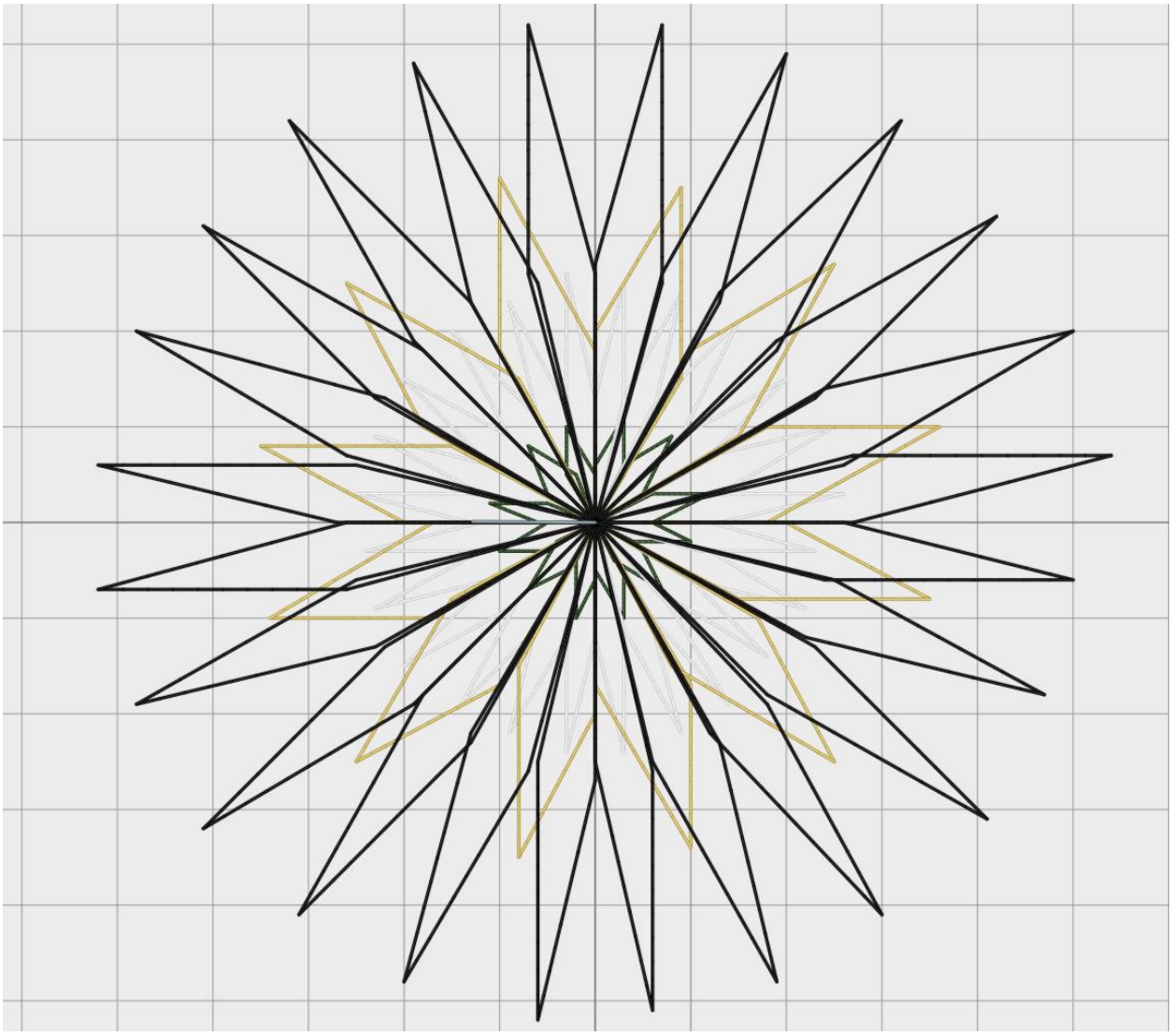


II. Simple Design (Draft): StitchBuddy Rendering

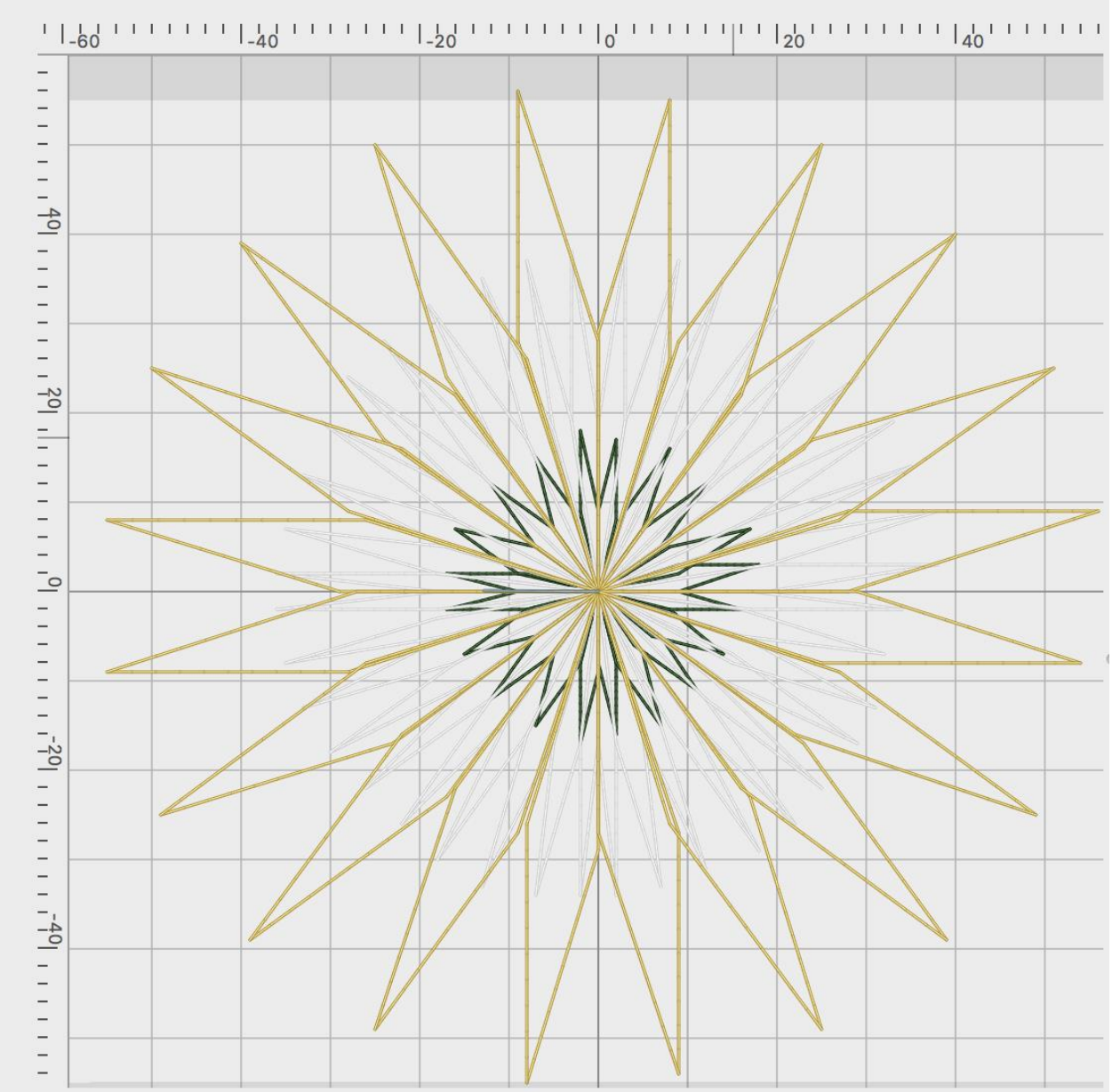


III. Photos of Simulated Embroidered Designs

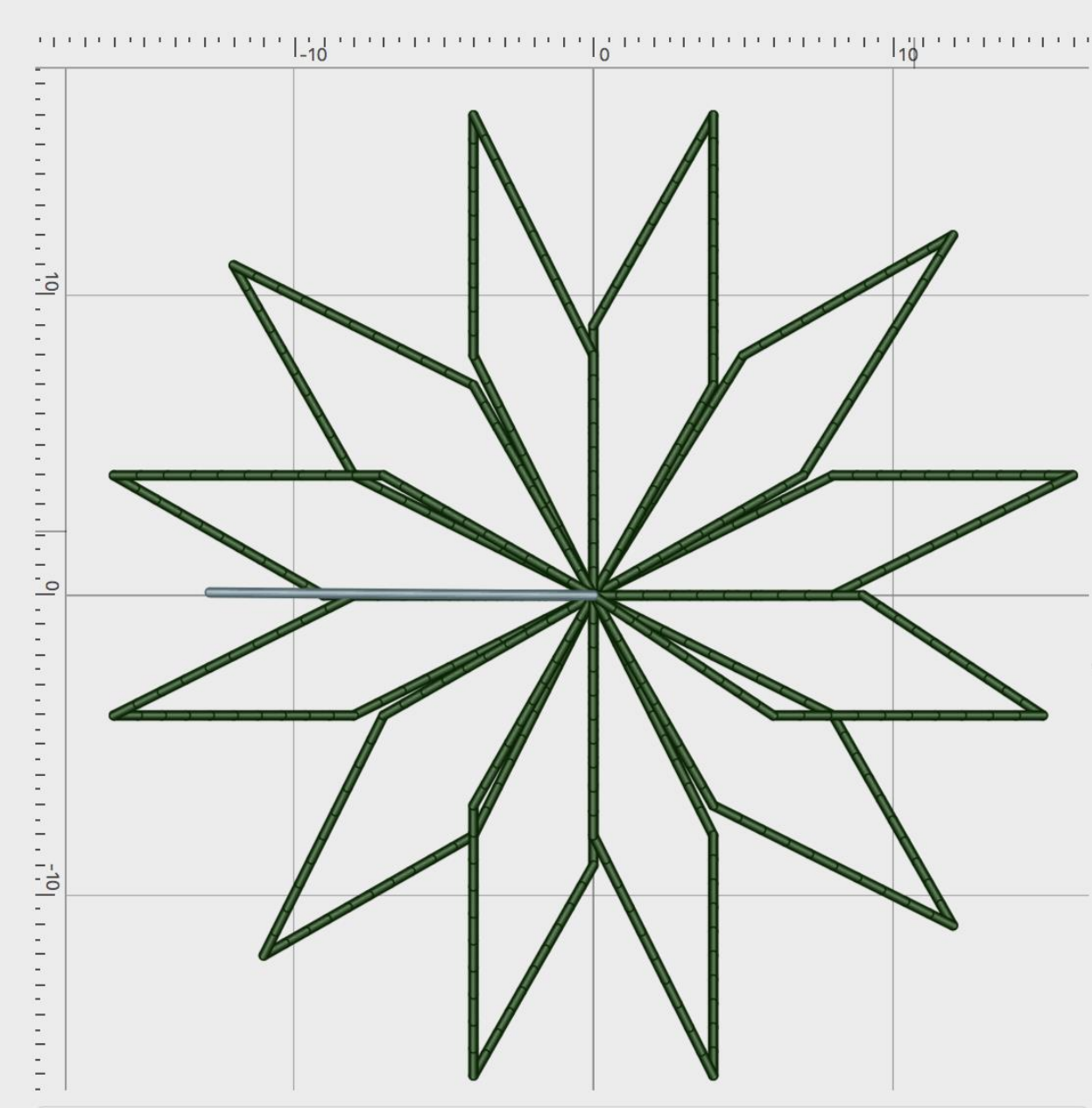
4 Flowers, Base Length 70, 24 Petals



3 Flowers, Base Length 70, 20 Petals



1 Flower, Base Length 100, 12 Petals



IV. Code Description

This embroidery program creates a parametric fractal flower. The scale and number of fractal expansions is specified by the user as inputs for number of petals, number of flowers, and the size of a petal. The base idea of the program is to create one diamond as a petal for the flower, then rotate that petal by a given angle until the full flower is created. The base petal is made according to the following formulae

x = length * math.cos(angle)
y = length * math.sin(angle)

a = [x, y]
b = [x, 0]
c = [-x, -y]
d = [-x, 0]

In which the vectors a through b are denoted as clockwise rotation around the diamond. This vector combination is fairly simple: positive diagonal displacement followed by positive horizontal displacement then by negative diagonal displacement and negative horizontal displacement. Each of these segments are broken up into 10 stitches for implementation. After drawing, the following rotation matrix is applied to each of the vectors in order to rotate them by a given angle.

$$R\mathbf{v} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

This process is then repeated until the full flower is created. This functionality is captured in a function called flower, which is called multiple times according to user input and takes length and the number of petals as parameters. The colour of thread is changed for each flower. Please refer to the above pictures in section III for examples of varying each parameter. Further documentation can be found in the code appendix.

V. Code Appendix

```
# Function to create stitch sequence

import math



global num_flowers



def negate(val):

 #also rounds

 a = val

 if (val < 0):

 a = 256 - int((abs(val) / 10))

 else:

 a = int(val / 10)

 if(a == 256):

 a = 0

 return a

def flower(stitches, number, length):



 #find angle between each petal

 angle = math.pi / (number / 2)

 #x and y values for initial petal

 x = length * math.cos(angle)

 y = length * math.sin(angle)



 #create vectors for each segment of the petal

 a = [x, y]

 b = [x, 0]
```

```
c = [-x, -y]

d = [-x, 0]

for n in range(0, int((2 * math.pi) / angle)):

#iterate for the number of petals

ar = [0, 0]

br = [0, 0]

cr = [0, 0]

dr = [0, 0]



#If vector component is negative, set it to be 1- abs(component).

#This function also divides each vector by 10 and rounds to int

ar = [negate(a[0]), negate(a[1])]

br = [negate(b[0]), negate(b[1])]

cr = [negate(c[0]), negate(c[1])]

dr = [negate(d[0]), negate(d[1])]



#Make 10 stitches for each vector

for i in range(0, 10):

stitches += ar

for i in range(0, 10):

stitches += br

for i in range(0, 10):

stitches += cr

for i in range(0, 10):

stitches += dr



#Apply rotation matrix to each vector

#Note that this is done on the unrounded values
```

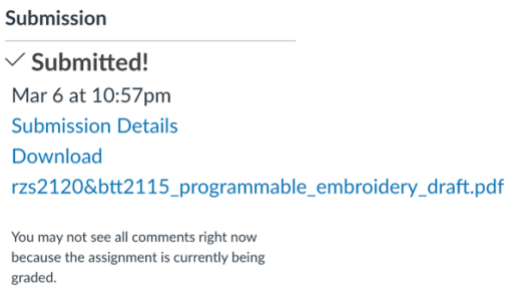
```
 a = [int(a[0] * math.cos(angle) - a[1] * math.sin(angle)), int(a[0] * math.sin(angle) + a[1] *
math.cos(angle))]

 b = [int(b[0] * math.cos(angle) - b[1] * math.sin(angle)), int(b[0] * math.sin(angle) + b[1] *
math.cos(angle))]

 c = [int(c[0] * math.cos(angle) - c[1] * math.sin(angle)), int(c[0] * math.sin(angle) + c[1] *
math.cos(angle))]

 d = [int(d[0] * math.cos(angle) - d[1] * math.sin(angle)), int(d[0] * math.sin(angle) + d[1] *
math.cos(angle))]




def getStitchSequence():



 #user input for number of flowers, length of each petal, and the number of petals per flower

 global num_flowers

 num_flowers = int(input("Please input the number of Flowers: "))

 length = int(input("Please input the base length: "))

 num_pedals = int(input("Please input the number of pedals per flower: "))



 # stitches = [128, 2] # 128 = escape_character , 2=Move

 stitches = [0, 0]     # followed by 8 bit displacement X,Y



 #iterate for each flower

 for i in range(1, num_flowers + 1):

 stitches += [0, 0]

 #Call flower function with input parameters.

 #length multiplies each time

 #Number of petals alternates between num_pedals and num_pedals / 2

 flower(stitches, int((num_pedals / (i%2 + 1))), length * i)



 #switch colors for each flower

 stitches += [128, 1]
```

```
 stitches += [128, 1]

 stitches += [0, 0]




 return stitches

# Function to create JEF file header

def getJefHeader(num_stitches, num_colors):

 jefBytes = [ 128, 0, 0, 0, # The byte offset of the first stitch

 10, 0, 0, 0, # unknown command

 ord("2"), ord("0"), ord("2"), ord("1"), #YYYY

 ord("0"), ord("2"), ord("2"), ord("4"), #MMDD

 ord("1"), ord("5"), ord("2"), ord("1"), #HHMM

 ord("0"), ord("0"), 99, 0, #SS00

 num_colors, 0, 0, 0, # Thread count nr. (nr of thread changes)

 (num_stitches) & 0xff, (num_stitches) >> 8 & 0xff, 0, 0, # Number of stitches

 3, 0, 0, 0, # Sewing machine Hoop

 # Extent 1

 50, 0, 0, 0, # Left boundary dist from center (in 0.1mm)

 50, 0, 0, 0, # Top boundary dist from center (in 0.1mm)

 50, 0, 0, 0, # Right boundary dist from center (in 0.1mm)

 50, 0, 0, 0, # Bottom boundary dist from center (in 0.1mm)

 # Extent 2

 50, 0, 0, 0, # Left boundary dist from center (in 0.1mm)

 50, 0, 0, 0, # Top boundary dist from center (in 0.1mm)

 50, 0, 0, 0, # Right boundary dist from center (in 0.1mm)

 50, 0, 0, 0, # Bottom boundary dist from center (in 0.1mm)

 # Extent 3

 50, 0, 0, 0, # Left boundary dist from center (in 0.1mm)
```

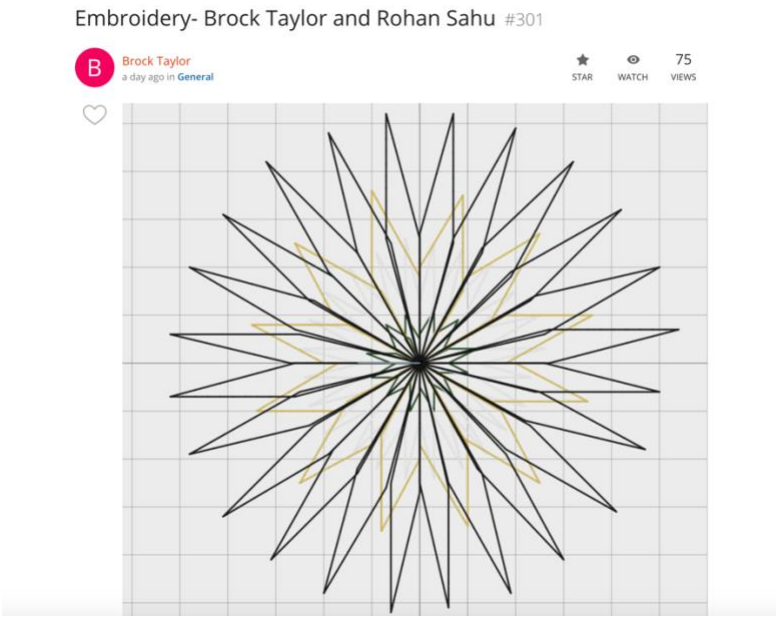
```
 50, 0, 0, 0, # Top boundary dist from center (in 0.1mm)
 50, 0, 0, 0, # Right boundary dist from center (in 0.1mm)
 50, 0, 0, 0, # Bottom boundary dist from center (in 0.1mm)
 # Extent 4
 50, 0, 0, 0, # Left boundary dist from center (in 0.1mm)
 50, 0, 0, 0, # Top boundary dist from center (in 0.1mm)
 50, 0, 0, 0, # Right boundary dist from center (in 0.1mm)
 50, 0, 0, 0, # Bottom boundary dist from center (in 0.1mm)
 # Extent 5
 50, 0, 0, 0, # Left boundary dist from center (in 0.1mm)
 50, 0, 0, 0, # Top boundary dist from center (in 0.1mm)
 50, 0, 0, 0, # Right boundary dist from center (in 0.1mm)
 50, 0, 0, 0, # Bottom boundary dist from center (in 0.1mm)
 5, 0, 0, 0, # Thread Color (white)
 2, 0, 0, 0, # Thread Color (white)
 13, 0, 0, 0, # Thread type (unknown)
 ]
 return jefBytes
#Main program combines headers and stich sequence
def main():
 stitchseq = getStitchSequence()
 header = getJefHeader(len(stitchseq)//2, num_flowers)
 data = bytes(header) + bytes(stitchseq)
 with open("square7.jef", "wb") as f:
 f.write(data)


if __name__ == '__main__':
 main()
```

VI. Rubrics Attempted

1. 10pts Cover page correct and complete (pp. 1)
2. 10pts Report neatly organized and formatted (pp. 1-13)
3. 10pts Circle pattern submitted a week before the deadline (show screenshot)



4. 10pt A parametric fractal shape embroidered (pp. 3-6)
5. 10pt Complexity/Aesthetics of the best pattern (pp. 3-5)
6. 10pt Quality of the stitch (over-stitching, wide stitches) (pp. 3-5)
7. 10pt Number of input parameters in software interface (pp. 6)
8. 10pt A description of the software you wrote – calculation steps, formulas, conditions. (pp. 6-11)
9. 10pt A fractal shape that is not a tree (pp. 3-5)
10. 10pt Multiple threads used (at least two) (pp. 3-6)
11. 10pt Multiple threads colors used (excluding bobbin thread) (pp 3-6)
12. 10pt Embroidery posted on Ed at least 24h before deadline (show screenshot)



Total: 12*10 = 120 points