The creative art of algorithmic embroidery

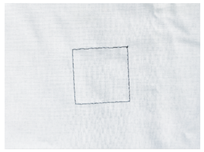
Marie Roald & Yngve Mardal Moe

TurtleThread is a Python package inspired by Turtlestitch and powered by pyembroidery that lets you create embroidery patterns with turtle commands and context managers. This poster shows some examples that use loops, variables and recursion to create patterns inspired by traditional embroidery and mathematics

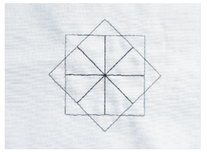
```python
import turtlethread

needle = turtlethread.Turtle()
with needle.running_stitch(30):
    needle.forward(300)

needle.save("line.jef")
```



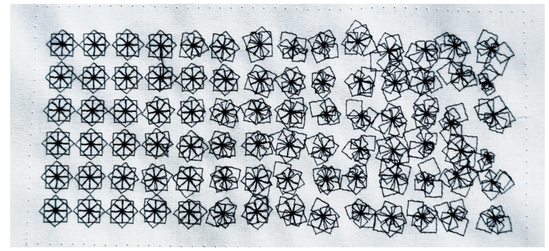
```python
import turtlethread

def draw_square(needle, length):
    for side in range(4):
        needle.forward(length)
        needle.right(90)

needle = turtlethread.Turtle()
with needle.running_stitch(30):
    draw_square(needle, 400)

needle.save("square.jef")
```

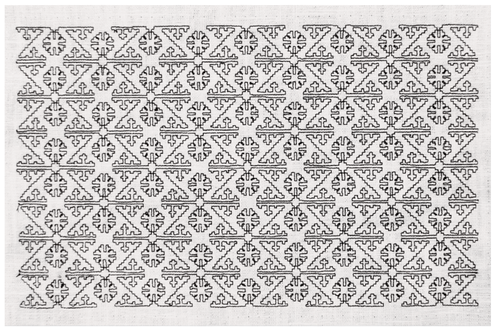


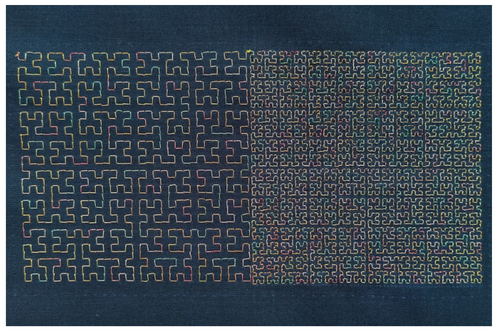
```python
import turtlethread
from example_2 import draw_square

def draw_flower(needle, length):
    for petal in range(8):
        square(needle, length)
        needle.right(45)

needle = turtlethread.Turtle()
with needle.running_stitch(30):
    draw_flower(needle, 400)

needle.save("flower.jef")
```

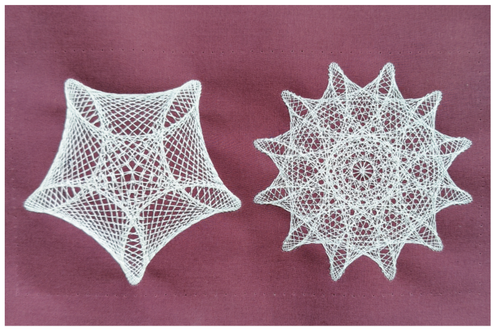


```python
import turtlethread
from random import gauss
from example_2 import draw_square

def random_flower(needle, length, noise):
    for petal in range(8):
        # Choose petal size and angle
        l = gauss(length, noise*length)
        a = gauss(45, noise*45)

        draw_square(needle, l)
        needle.right(a)

needle = turtlethread.Turtle()
for x in range(0, 1951, 150):
    # Set noise level
    n = 0.4 ** (x / 1950)**1.4
    for y in range(0, 751, 150):
        # Choose random offset and rotation
        dx = gauss(0, 50*n)
        dy = gauss(0, 50*n)
        da = gauss(0, 90*n)

        # Move to random offset
        with needle.jump_stitch():
            needle.goto(x + dx, y + dy)
            needle.setheading(da)

        # Embroider random flower
        with needle.running_stitch(25):
            random_flower(needle, 50, n)

needle.save("field_schotter.jef")
```









Painting by Hans Holbein the Younger

Blackwork is a counted thread embroidery where a black thread is used on white linen fabric. It was prevalent among British royals in the late 15th-16th century.

Early blackwork is very geometric, and this code example forms a pattern inspired by that on Jane Seymour's cuff. Three motifs are repeated and mirrored to create the design: a staircase a cross and a flower.

```python
import turtlethread

# The following imports are available on GitHub
from blackwork_utils import (
    draw_cross_line,
    draw_staircase,
    draw_flower
)


def draw_column(pen, unit, d, length):
    for n in range(length):
        # Draw a flower after iteration 1
        if n > 0:
            pen.forward(unit*2)
            draw_flower(pen, unit, d)
            pen.forward(unit*2)

        # Mirror symmetry
        for m in [1, -1]:
            direction = d*m

            # Straight line
            pen.forward(12*unit)
            pen.left(-90*direction)
            pen.forward(unit)
            pen.left(-90*direction)

            # Staircase and crosses
            draw_cross_line(
                pen, unit, direction
            )
            draw_staircase(
                pen, unit, direction
            )
            draw_cross_line(
                pen, unit, -direction
            )

            # Straight line
            pen.left(90*direction)
            pen.forward(unit)
            pen.left(90*direction)
            pen.forward(12*unit)

            # Flower separator
            if m == 1:
                pen.forward(unit*2)
                draw_flower(pen, unit, -direction)
                pen.forward(unit*2)

unit = 12

pen = turtlethread.Turtle()
pen.left(-90)
for offset in range(4):
    if offset > 0:
        with pen.jump_stitch():
            pen.goto(0 + 2 * offset * 14 * unit, 0)

    # Draw left column
    with pen.running_stitch(unit):
        draw_column(pen, unit, 1, 5)

    # Move to top
    with pen.jump_stitch():
        pen.goto(unit + 2 * offset * 14 * unit, 0)

    # Draw right column
    with pen.running_stitch(unit):
        draw_column(pen, unit, -1, 5)

pen.save("blackwork.jef")
```

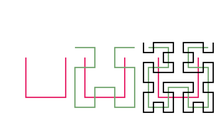


Illustration by Quef/Wikimedia Commons (modified)

The **Hilbert curve** is a recursive curve that fills a square without crossing itself. It has several interesting properties and is used in various applications in computer science.

To construct the Hilbert curve, the code uses a recursive function that replaces each line with a new Hilbert curve. The embroidery pattern consists of recursion levels 5 (left) and 6 (right).

```python
import turtlethread

def hilbert(pen, length, n, angle=90):
    if n == 0:
        return

    pen.right(angle)
    hilbert(pen, length, n-1, -angle)
    pen.forward(length)

    pen.left(angle)
    hilbert(pen, length, n-1, angle)
    pen.forward(length)
    hilbert(pen, length, n-1, angle)
    pen.left(angle)

    pen.forward(length)
    hilbert(pen, length, n-1, -angle)
    pen.right(angle)


pen = turtlethread.Turtle()

# Draw first, low resolution, curve
with pen.running_stitch(2*18):
    hilbert(pen, 2*18, 5)

# Add space
pen.forward(18)

# Draw second, high resolution, curve
with pen.running_stitch(18):
    hilbert(pen, 18, 6)

pen.save("hilbert.jef")
```

Photo (right) by Starker Moe

White shapes embroidered on white cloth characterise **Norwegian whitework**, which is found, among other places, on decorative cloth and the shirts of the Norwegian national dress,

This example is inspired by a shirt cuff and a motif popular in Norway: eight parallelograms flipped and rotated to form a flower. The code repeats square corners to make the border and repeats a tight zigzag pattern to fill the parallelograms.

```python
import turtlethread

# The following import is available on GitHub
from whitework_utils import draw_parallelogram


# Border
def forward_and_left(pen, a, l):
    pen.forward(l)
    pen.left(a)

def step(pen, w, h, d):
    forward_and_left(pen, -d*90, w)
    forward_and_left(pen, d*90, h)
    forward_and_left(pen, d*90, w)
    forward_and_left(pen, -d*90, h)

def walk(pen, w, h, steps, d):
    for i in range(steps):
        step(pen, w, h, d)

def border(pen, w, h, steps, n_triangles, d):
    x0, y0 = pen.position()

    for row in range(steps):
        # Adjust row starting position
        with pen.jump_stitch():
            x = x0 + row*w
            y = y0 + d*row*h
            pen.goto(x, y)

        for tri in range(n_triangles):
            # Make row of border
            with pen.running_stitch(w):
                walk(pen, w, h, steps-row, d)

            # If we're not on final triangle
            if tri != n_triangles-1:
                with pen.jump_stitch():
                    pen.forward(w*2*row)

        # End row with a short line
        with pen.running_stitch(w):
            pen.forward(w)

# Rose
def rose(pen, offset, scale):
    x0, y0 = pen.position()
    for n in range(4):
        pen.right(90)
        for mirror in [1, -1]:
            # Add offset to petals
            with pen.jump_stitch():
                pen.right(mirror*17)
                pen.forward(offset)
                pen.right(-mirror*17)

            # Draw a petal as a parallellogram
            draw_parallelogram(
                pen, scale, mirror
            )

            # Go back to center again
            with pen.jump_stitch():
                pen.right(mirror*17)
                pen.forward(-offset)
                pen.right(-mirror*17)

pen = turtlethread.Turtle()

# Border
for y, d in [[-750, 1], [750, -1]]:
    pen.goto(-972, y)
    border(pen, 24, 18, 10, 4, d)

# Roses
for x in [-732, 0, 732]:
    pen.goto(x, 0)
    rose(pen, 20, 100)

pen.save("whitework.jef")
```

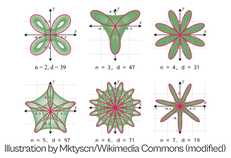


Illustration by Mktycsn/Wikimedia Commons (modified)

The **Maurer rose**, named after Peter M. Maurer, who first introduced it, is a polar graph made up of lines connecting points on a rose curve, forming a rose-like pattern. By adjusting the rotation angle and scaling, we can get a variety of expressions demonstrating how simple mathematical operations can lead to beautiful and intricate designs.

The embroidery above shows roses with n=5, d=97 (left) and n=6, d=71 (right).

```python
import turtlethread
from math import pi, sin, cos


def draw_rose(pen, n, d, r0):
    x0, y0 = pen.position()
    for theta in range(0, 361, 1):
        k = theta * d * pi / 180
        r = r0 * sin(n * k)
        x = r * cos(k)
        y = r * sin(k)
        needle.goto(x0 + x, y0 + y)

needle = turtlethread.Turtle()

# Draw first rose
with needle.running_stitch(50):
    draw_rose(needle, 5, 97, 570)

# Move a bit more than two radii
with needle.jump_stitch():
    needle.forward(1150)

# Draw second rose
with needle.running_stitch(50):
    draw_rose(needle, 6, 71, 570)

needle.save("maurer_rose.jef")
```

Find more examples and the documentation at:
turtlethread.com
All code examples from this poster are available at
github.com/TurtleThread/PyConUS23_poster

REFERENCES

Batsford Encyclopaedia of Embroidery Techniques. Swift, Gay (1984)

Blackwork Embroidery. Geddes, Elisabeth & McNeill, Moyra (1976)

The Art of Blackwork Embroidery. Drysdale, Rosemary (1975)

A Rose is a Rose... Maurer, Peter M (1987). DOI: 10.2307/2322215

Space-Filling Curves. Bader, Michael (2013). DOI: 10.1007/978-3-642-31046-1