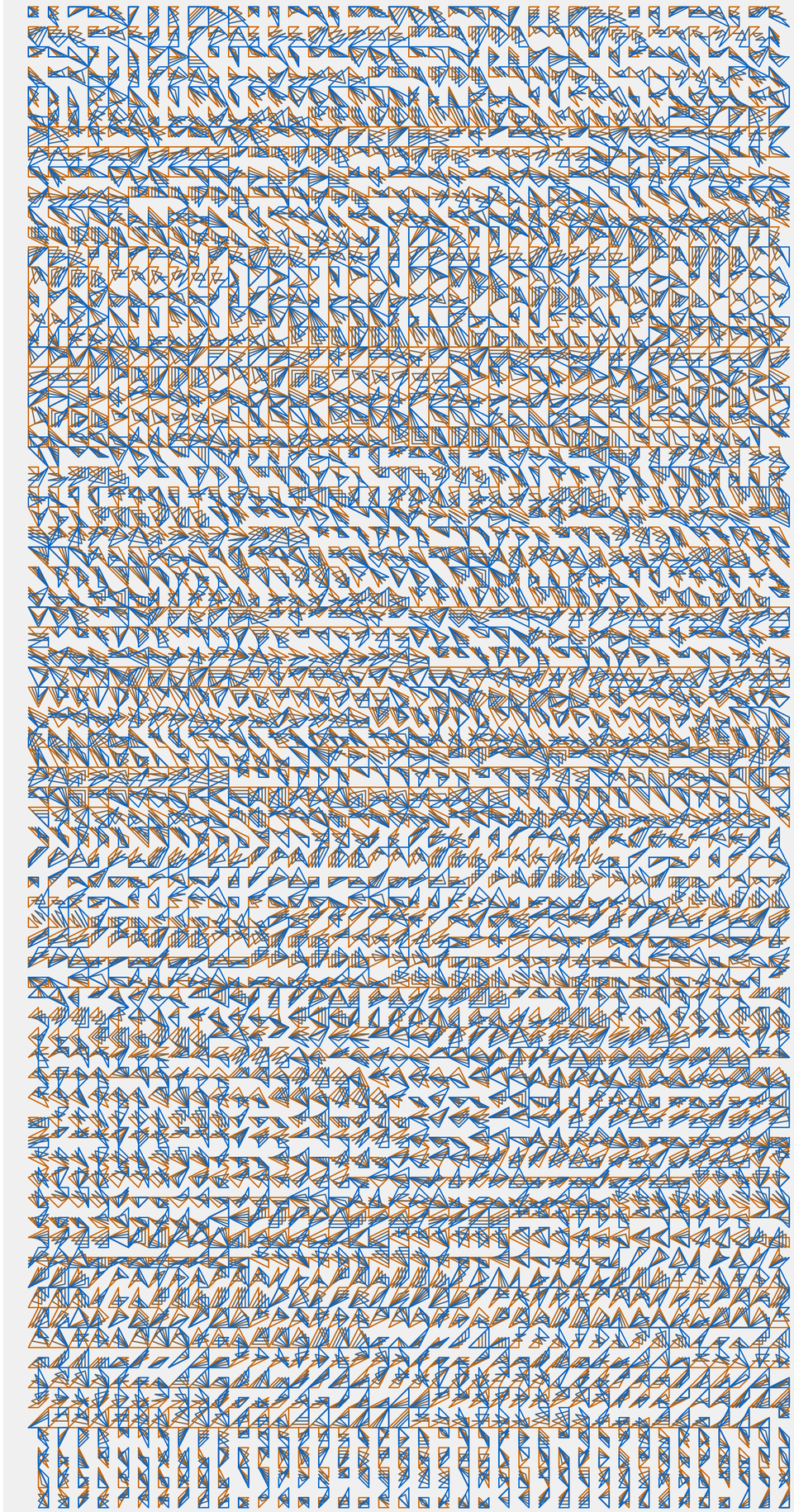

```
"""
Title: 2850 Triangle Interpolations v3
Author: Alexandre B A Villares - https://abav.lugaralgum.com/
License: Creative Commons Attribution 4.0 International License
https://creativecommons.org/licenses/by/4.0/

Submitted to Internet & Sociedade http://revista.internetlab.org.br
in 2019-06-30. Coded with Processing Python Mode
https://py.processing.org (Processing 3.5.3 + Python Mode 3056)
"""

add_library('pdf')
from itertools import product, combinations

SPACE, BORDER = 16, 20

def setup():
    """Prepare screen or SVG and geometry."""
    global two_triangle_combos, W, H
    # size(1240, 648)  # used to debug on screen
    size(648, 1240, PDF, "2850_T_I_v3.pdf") # export
    strokeJoin(ROUND)
    # Calculate all 3-point combinations on a 3x3 grid
    grid_points = product((-1, 0, 1), repeat=2)
    point_triples = combinations(grid_points, 3)
    # Identify triangles (discard colinear points)
    triangles = []
    for pt in point_triples:
        area = (pt[1][0] * (pt[2][1] - pt[0][1]) +
                pt[2][0] * (pt[0][1] - pt[1][1]) +
                pt[0][0] * (pt[1][1] - pt[2][1]))
        if area != 0:
            triangles.append(pt)
    println("Number of possible triangles: {}"
            .format(len(triangles)))
    # Calculate the 2-triangle combinations
    two_triangle_combos = list(combinations(triangles, 2))
    println("Number of 2-triangle combinations: {}"
            .format(len(two_triangle_combos)))
    # Calculate the display grid dimensions
    W = (width - BORDER * 2) // SPACE
    H = (height - BORDER * 2) // SPACE
    println("Cols: {} Rows: {}"
            .format(W, H))


def draw():
    """Draw geometry."""
    background(240)
    i = 0
    for y in range(H):
        for x in range(W):
            if i < len(two_triangle_combos):
                pushMatrix()
                translate(BORDER + SPACE / 2 + SPACE * x,
                          BORDER + SPACE / 2 + SPACE * y)
                draw_combo(two_triangle_combos[i])
                popMatrix()
                i += 1
    exit()


def draw_combo(combo):
    """Draw a combination of 2 triangles, interpolating 2 others."""
    t0, t3 = combo[0], combo[1]
    t1, t2 = lerp_poly(t0, t3, 0.33), lerp_poly(t0, t3, 0.66)
    triangles = (t0, t1, t2, t3)
    # Colors for the triangles
    c0, c3 = color(200, 100, 0), color(0, 100, 200)
    c1, c2 = lerpColor(c0, c3, .33), lerpColor(c0, c3, .66)
    colors = (c0, c1, c2, c3)
    # For each triangle, draw it in a different stroke color.
    noFill()
    half_combo = SPACE * .5  # this size lets the combinations touch
    for i, t in enumerate(triangles):
        stroke(colors[i])
        draw_poly(scale_poly(t, half_combo))


def draw_poly(p_list, closed=True):
    """Draw a polygon from a list of points (vectors or tuples)."""
    beginShape()
    for p in p_list:
        if len(p) == 2 or p[2] == 0:
            vertex(p[0], p[1])
        else:
            vertex(*p)
    if closed:
        endShape(CLOSE)
    else:
        endShape()


def lerp_poly(p0, p1, t):
    """Create interpolated version of poly - using tuples for points """
    return [tuple(lerp(c0, c1, t) for c0, c1 in zip(sp0, sp1))
            for sp0, sp1 in zip(p0, p1)]


def scale_poly(p_list, s):
    """Return a scaled version of a list of points (as tuples)."""
    return [(p[0] * s, p[1] * s) for p in p_list]
```