CURVETOPIA: A Journey into the World of Curves

July 15, 2024

Welcome to Curvetopia, where we bring order and beauty to the world of 2D curves! This project will guide you through identifying, regularizing, and beautifying various types of curves. Let's dive in and explore.

Objective: Our mission is to identify, regularize, and beautify curves in 2D Euclidean space. We'll start by focusing on closed curves and progressively work with more complex shapes. This project will also cover symmetry and curve completion techniques.

Problem description: Ideally we want a end to end process where we take a PNG (raster) image of a line art and output a set of curves, where the curves are defined as a connected sequence of cubic Bézier curves.

However to begin, we are allowed to make the following simplification. Instead of PNG as input, we will present line art in the form of *polylines*, which is defined as sequence of points. To be precise, let a path be a finite sequence of point $\{p_i\}_{1 \leq i <= n}$ from $\mathbb{R}^2$, and $\mathcal{P}$ be the set of all paths in $\mathbb{R}^2$. The input to our problem is finite subset of paths from $\mathcal{P}$. The expected output is another set of paths with the necessary properties of *regularization*, *symmetry* and *completeness* as defined in next sections.

For visualization, we may use SVG format that can be rendered on a browser, and the output curve can be in the form of cubic Bézier instead of polylines. The principal challenge is divided in the following sections.[1]

1 Regularize Curves

Identify regular shapes on a given set of curves. Task can be broken down into the following primitives:

1. **Straight lines.**
2. **Circles and Ellipses**. Look for curves where all points are equidistant from a center (circle) or have two focal points (ellipse).

[1] Converting an image of a line art to *polyline* is an optional challenge.

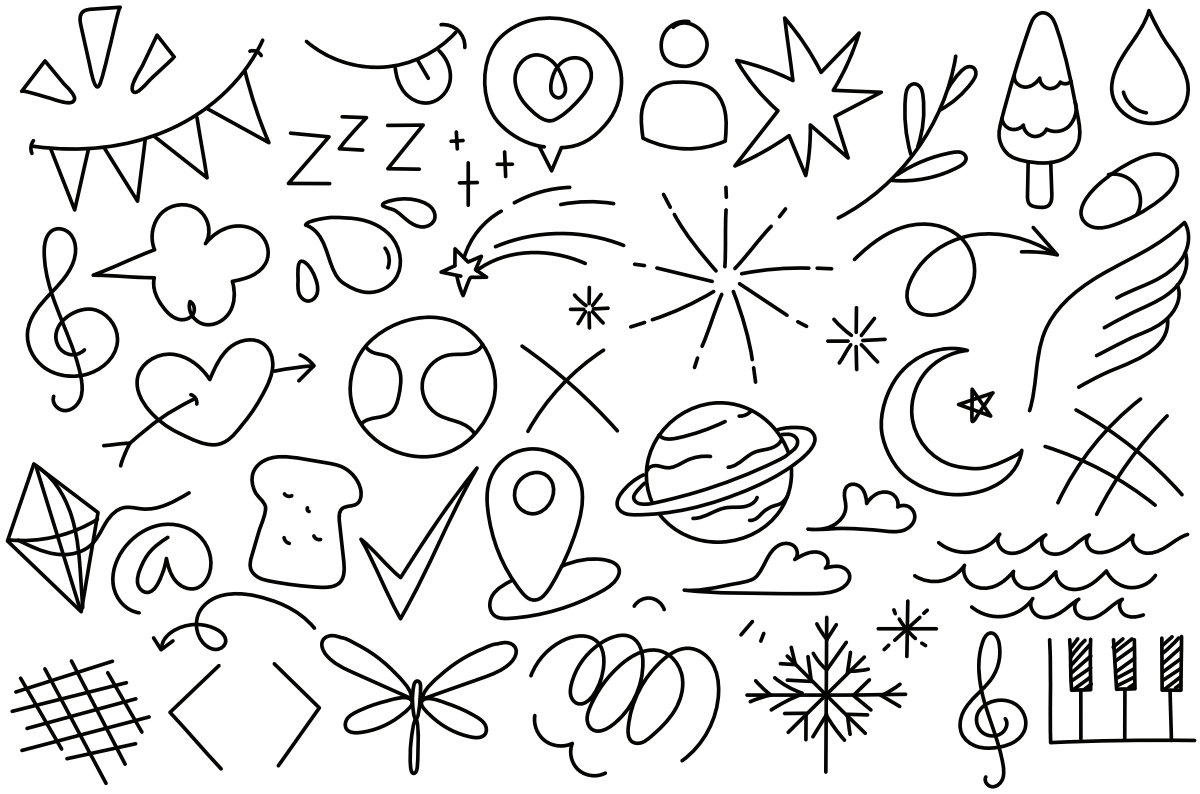


Figure 1: A doddle

3. **Rectangles and Rounded Rectangles.** Identify rectangles and distinguish them from rectangles with curved edges. Identify rectangles and distinguish them from rectangles with curved edges.

4. **Regular Polygons**. Identify polygons with equal sides and angles.

5. **Star Shapes**. Identify a star shape from various curves. Look for a central point with multiple radial arms.

Activity: The target use cases for this activity are hand drawn shapes and doodles. It is important to consider that not all shapes can be regularised. Your algorithm should be able to distinguish between them. Test your algorithms with different images containing various shapes and verify the results.

2 Exploring Symmetry in Curves

Consider closed shapes. Identify the presence of symmetry in them. We begin with reflection symmetries. Explore how symmetries are defined, for example, check for lines of

symmetry where the shape can be divided into mirrored halves for reflection symmetry.

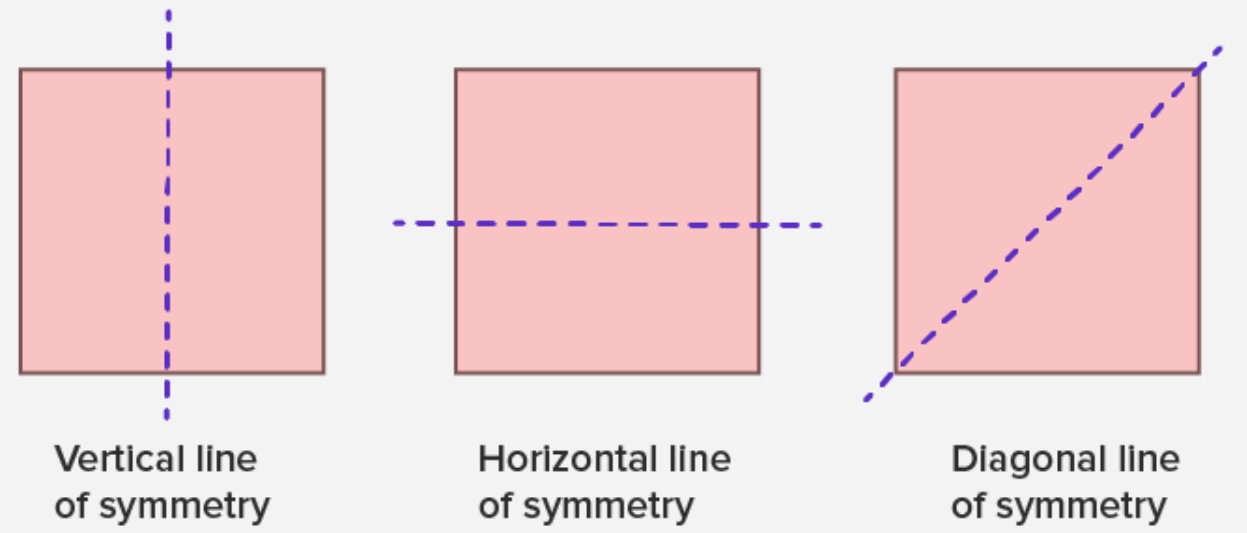


Figure 2: Reflection symmetries

Symmetry hunt: Observe that an identical looking curve may be created using different sequence of Bézier curves. The task is to identify symmetry, it makes sense to transform the presentation as set of points. In the first stage, identify symmetry under the assumptions, and then try to fit identical Bézier curves on points that are symmetric.

3 Completing Incomplete Curves

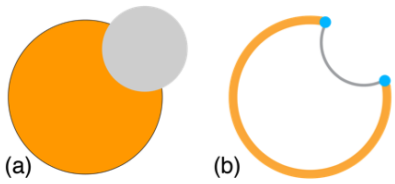
Imagine a set of 2D curves that have been "planarized," meaning any overlapping portions have been removed, leaving gaps or partial holes in the curves. Your task is to complete these curves in a natural way.

Example. Create algorithms using computer vision techniques to identify and naturally complete incomplete 2D curves that have gaps due to occlusion removal. The input will be one or more curves with another curve that shares some boundary with the curves, which need to be modified to complete the curve.

Figure 3: One circle partially occluding another.

Guide. Identify different levels of shape occlusion with increasing level of complexity.

- Fully contained. e.g. one circle is completely inside another.



- Partially contained but the closed curve is still connected, say half a circle is occluded by another circle.
- The curve is disconnected. For example a circle being disconnected by a long rectangle on top of it. In such case the input are two portions of the circle on either side of the rectangle.

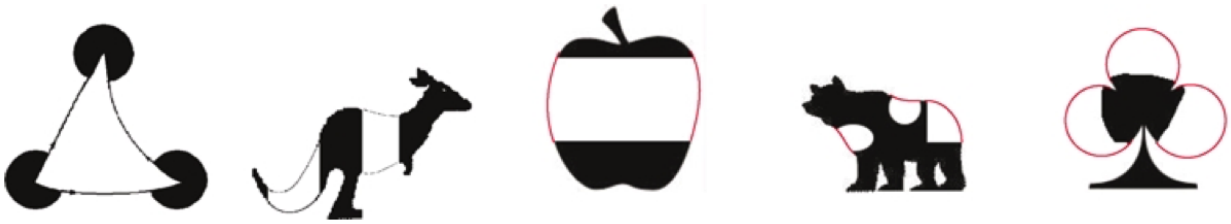


Figure 4: Shapes to complete.

Activity. The task is to explore the notion of curve completion (not object). The completion algorithm may be guided by smoothness, regularity and symmetry.

4 Attachment

Please refer to the attached zip file for examples. In a general setting, shapes can be represented by any SVG curve primitive (Bezier, line, arcs). For the sake of uniform presentation, the examples contain polyline approximations of the curves. These polylines are saved as CSV files. Use the following code to read the CSV files:

```
import numpy as np

def read_csv(csv_path):
    np_path_XYs = np.genfromtxt(csv_path, delimiter=',')
    path_XYs = []
    for i in np.unique(np_path_XYs[:, 0]):
        npXYs = np_path_XYs[np_path_XYs[:, 0] == i][:, 1:]
        XYs = []
        for j in np.unique(npXYs[:, 0]):
            XY = npXYs[npXYs[:, 0] == j][:, 1:]
            XYs.append(XY)
        path_XYs.append(XYs)
    return path_XYs
```

This code will yield a list containing each shape. A shape is encoded as a list of paths, where each path is a numpy array of points defining the polyline. Use the following code to visualize the shapes. Use the following code to visualize:

```
import numpy as np
import matplotlib.pyplot as plt

def plot(paths_XYs):
    fig, ax = plt.subplots(tight_layout=True, figsize=(8, 8))
    for i, XYs in enumerate(path_XYs):
        c = colours[i % len(colours)]
        for XY in XYs:
            ax.plot(XY[:, 0], XY[:, 1], c=c, linewidth=2)
    ax.set_aspect('equal')
    plt.show()
```

4.1 Expected results

The sets of input polyline for curve regularization and symmetry can be classified the following category:

- *Isolated.* This is the simple form of input curves where each shape is one single polyline. Several isolated curves s well as expected solution are presented in:
  - Input: `exmaples/isolated.csv`, Expected output: `exmaples/isolated_sol.csv`
- *Fragmented.* A shape is build from a collection of polylines. The challenge is to find regular shapes and symmetry in a subset of polylines. The examples are presented in the following paths:
  - Input: `examples/frag0.csv`, `examples/frag1.csv`, Expected output: `examples/frag01_sol.csv`
  - Input: `examples/frag2.csv`, Expected output: `examples/frag2.csv`

For shape completion, we will consider closed curves. Each closed curve defined by a single (embeded) polyline. When we consider one such curve, we expect all the neighbouring curves (curves that share boundary) to be completed.

- *Connected Occlusion*. The occluding shape partially blocks the occluded shape, hence the shape is still connected and can be represented by a single polyline.
  - Input: `examples/occlusion1.csv`. Expected output: `examples/occlusion1_sol.csv`
- *Disconnected Occlusion*. The occluding shape fragments the occluded shape into multiple closed paths (polylines).
  - Input: `examples/occlusion2.csv`, Expected output: `examples/occlusion2_sol.csv`

4.2 Evaluation

Regularization and symmetry will be evaluated based on the number of regular geometric shapes and the symmetry found in the input. For example, in `examples/frag0.csv` and `examples/frag1.csv`, we have two regular geometric shapes: a rectangle with two lines of symmetry, a circle with radial symmetry, and two other curves, each with one line of symmetry.

Observe that we can define the same shape using different SVG primitives and polylines. Therefore, for the evaluation of occlusion, we will use rasterization of the SVG to ascertain the correctness of a solution. We will use filled paths and rasterize them with aliasing. Use the below code for rasterization.

```
import svgwrite
import cairosvg


def polylines2svg(paths_XYs, svg_path):
    W, H = 0, 0
    for path_XYs in paths_XYs:
        for XY in path_XYs:
            W, H = max(W, np.max(XY[:, 0])), max(H, np.max(XY[:, 1]))
    padding = 0.1
    W, H = int(W + padding * W), int(H + padding * H)

    # Create a new SVG drawing
    dwg = svgwrite.Drawing(svg_path, profile='tiny',
                           shape_rendering='crispEdges')
    group = dwg.g()

    for i, path in enumerate(paths_XYs):
        path_data = []
        c = colours[i % len(colours)]
        for XY in path:
            path_data.append(("M", (XY[0, 0], XY[0, 1])))
            for j in range(1, len(XY)):
                path_data.append(("L", (XY[j, 0], XY[j, 1])))
            if not np.allclose(XY[0], XY[-1]):
                path_data.append(("Z", None))
        group.add(dwg.path(d=path_data, fill=c,
                           stroke='none', stroke_width=2))
    dwg.add(group)
    dwg.save()

    png_path = svg_path.replace('.svg', '.png')
    fact = max(1, 1024 // min(H, W))
    cairosvg.svg2png(url=svg_path, write_to=png_path,
                     parent_width=W, parent_height=H,
                     output_width=fact*W, output_height=fact*H,
                     background_color='white')
    return
```