Curvetopia: A Comprehensive Guide to Curve Regularization and Beautification

Welcome to Curvetopia, where we transform and beautify 2D curves. This document provides a detailed guide to identifying, regularizing, and enhancing curves in 2D space. Our goal is to convert line art into smooth, regularized cubic Bezier curves, exploring symmetry, and completing incomplete curves.
This guide provides a step-by-step approach to regularizing shapes from images using OpenCV. We will cover reading polylines from a CSV file, converting them to images, detecting and classifying shapes, simplifying contours, and detecting symmetries. The guide includes instructions for running the code in different environments, such as Google Colab and local machines.

Project Overview

Objective

The objective of this project is to identify, regularize, and beautify curves in 2D Euclidean space. We start with closed curves and work towards more complex shapes, ultimately converting line art into cubic Bezier curves. Initially, we will work with polylines, simplifying our task by focusing on sequences of points rather than full raster images.

Problem Description

Given a set of polylines representing paths in a 2D plane, our goal is to process these paths to produce another set with the following properties:

1. Regularization: Smooth and uniform representation.
2. Symmetry: Identification of symmetry lines.
3. Completion: Filling in gaps in incomplete curves.

The final output will be visualized in SVG format using cubic Bezier curves instead of polylines.

Principal Challenges

1. Regularization of Curves
2. Exploring Symmetry in Curves
3. Completing Incomplete Curves

1. Environment Setup

- Google Colab
  - If you're using Google Colab, you'll need to use cv2_imshow() from the google.colab.patches module for displaying images. Ensure that you have installed the necessary packages.
  - Installation:
    !pip install opencv-python-headless matplotlib numpy

  - Importing for Colab:
    from google.colab.patches import cv2_imshow
    import cv2
    import numpy as np
    import matplotlib.pyplot as plt

- Local Environment
  - For local environments, use cv2.imshow() for displaying images. Ensure you have OpenCV and other dependencies installed.

  - Installation:
    pip install opencv-python matplotlib numpy

  - Importing for Local:
    import cv2
    import numpy as np
    import matplotlib.pyplot as plt

2. Reading and Plotting CSV Data

This section describes how to read polylines from a CSV file and plot them.

```
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
def plot(paths_XYs, ax, title=None, show_axis=True):
    colours = ['black']
    for i, XYs in enumerate(paths_XYs):
        c = colours[i % len(colours)]
        for XY in XYs:
            ax.plot(XY[:, 0], XY[:, 1], c=c, linewidth=2)
    ax.set_aspect('equal')
    if title:
        ax.set_title(title)
    if not show_axis:
        ax.axis('off')
```

Displaying Images:

- **Google Colab:**

  ```
  from google.colab.patches import cv2_imshow
  cv2_imshow(img)  # img is the image to display
  ```

- **Local Environment:**

  ```
  cv2.imshow('Image', img)  # img is the image to display
  cv2.waitKey(0)
  cv2.destroyAllWindows()
  ```

3. Shape Detection and Regularization

- **Circle Detection:**

```
def is_circle(contour, approx, circularity_tolerance=0.3, area_ratio_tolerance=0.3):
    area = cv2.contourArea(contour)
    perimeter = cv2.arcLength(contour, True)
    if perimeter == 0:
        return False
    circularity = 4 * np.pi * area / (perimeter ** 2)
    (x, y), radius = cv2.minEnclosingCircle(contour)
    enclosing_circle_area = np.pi * (radius ** 2)
    area_ratio = area / enclosing_circle_area
    is_circular = (1 - circularity_tolerance <= circularity <= 1 + circularity_tolerance)
    is_area_close = (1 - area_ratio_tolerance <= area_ratio <= 1 + area_ratio_tolerance)
    return is_circular and is_area_close
```

- **Star Detection:**

```
def is_star(approx):
    if len(approx) == 10:
        angles = []
        for i in range(len(approx)):
            pt1 = approx[i][0]
            pt2 = approx[(i + 2) % len(approx)][0]
            angle = np.arctan2(pt2[1] - pt1[1], pt2[0] - pt1[0])
            angles.append(angle)
        angle_diff = np.diff(angles)
        if np.all(np.abs(angle_diff) > 0.5):
            return True
```

```
    return False
```

- **Line Simplification:**

```
def is_nearly_straight_line(pt1, pt2, pt3, threshold=0.3):
    vec1 = np.array(pt1) - np.array(pt2)
    vec2 = np.array(pt3) - np.array(pt2)
    angle = np.arccos(np.clip(np.dot(vec1, vec2) / (np.linalg.norm(vec1) * np.linalg.norm(vec2)), -1.0, 1.0))
    return np.abs(angle - np.pi) < threshold

def merge_collinear_points(approx, threshold=0.3):
    new_approx = []
    num_points = len(approx)
    i = 0
    while i < num_points:
        pt1 = approx[i][0]
        pt2 = approx[(i + 1) % num_points][0]
        pt3 = approx[(i + 2) % num_points][0]
        if is_nearly_straight_line(pt1, pt2, pt3, threshold):
            new_approx.append(approx[(i) % num_points])
            new_approx.append(approx[(i + 2) % num_points])
            i += 2
        else:
            new_approx.append(approx[i])
            i += 1
    return np.array(new_approx)
```

4. Contour Properties and Similarity Check

- **Contour Properties:**

```
def contour_properties(approx):
    x, y, w, h = cv2.boundingRect(approx)
    center = (x + w // 2, y + h // 2)
    aspect_ratio = float(w) / h
    return center, aspect_ratio, w, h
```

- **Similarity Check:**

```
def is_similar(contour1_props, contour2_props):
    center1, aspect_ratio1, w1, h1 = contour1_props
    center2, aspect_ratio2, w2, h2 = contour2_props
    center_dist = np.sqrt((center1[0] - center2[0]) ** 2 + (center1[1] - center2[1]) ** 2)
    aspect_ratio_similar = abs(aspect_ratio1 - aspect_ratio2) < 10
    dimension_similar = abs(w1 - w2) < 100 and abs(h1 - h2) < 100
    center_similar = center_dist < 10
    return aspect_ratio_similar and dimension_similar and center_similar
```

5. Symmetry Detection

- **Symmetry Check:**

```
def detect_symmetries(contour, image, tolerance=0.02, angles=np.arange(0, 360, 3)):
    mask = np.zeros(image.shape[:2], dtype=np.uint8)
    cv2.drawContours(mask, [contour], -1, 255, thickness=cv2.FILLED)
    def resize_mask(m, size):
        return cv2.resize(m, (size[1], size[0]), interpolation=cv2.INTER_NEAREST)
```

```
def check_symmetry(m1, m2):
    if m1.shape != m2.shape:
        m2 = resize_mask(m2, m1.shape)
    return np.mean(np.abs(m1 - m2)) < tolerance * 255
symmetries = 0
mask_h, mask_w = mask.shape
flip_h = cv2.flip(mask, 0)
flip_v = cv2.flip(mask, 1)
if check_symmetry(mask, flip_h):
    symmetries += 1
if check_symmetry(mask, flip_v):
    symmetries += 1
flip_d1 = cv2.transpose(mask)
flip_d1 = cv2.flip(flip_d1, 1)
flip_d2 = cv2.transpose(mask)
flip_d2 = cv2.flip(flip_d2, 0)
if check_symmetry(mask, flip_d1):
    symmetries += 1
if check_symmetry(mask, flip_d2):
    symmetries += 1
for angle in angles:
    M = cv2.getRotationMatrix2D((mask_w / 2, mask_h / 2), angle, 1)
    rotated_mask = cv2.warpAffine(mask, M, (mask_w, mask_h), flags=cv2.INTER_NEAREST)
    if check_symmetry(mask, rotated_mask):
        symmetries += 1
return symmetries
```

5. Complete Incomplete Curve

Objective

To complete incomplete curves by drawing additional shapes or extending existing contours to ensure that curves are fully formed.

Procedure

1. **Input Processing**:
   - Convert the image to binary and detect contours.

2. **Curve Completion**:
   - Draw additional shapes or extend contours to fill gaps.

3. **Output**:
   - Save and visualize the completed curves.

Detailed Steps

- **Read Image and Process Contours:**

```
def read_and_process_image(image_path):
    img = cv2.imread(image_path)
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(img_gray, 240, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    return img, contours
```

- **Complete Curves:**

```
def complete_curves(img, contours):
    img_output = np.ones_like(img) * 255
    for contour in contours:
        approx = cv2.approxPolyDP(contour, 0.01 * cv2.arcLength(contour, True), True)
        cv2.drawContours(img_output, [approx], 0, (0, 0, 0), 5)
    center = (250, 250)
    radius = 100
    color = (0, 0, 255)
    thickness = 2
    cv2.circle(img_output, center, radius, color, thickness)
    return img_output
```

- **Save and Display Completed Image:**

```
def save_and_display_completed_image(img_output, output_path):
```

```
cv2.imwrite(output_path, img_output)
cv2.imshow('Completed Curves', img_output)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

7. Troubleshooting

- **Google Colab Display Issues:** Ensure you use `cv2_imshow()` from the `google.colab.patches` module for displaying images.
- **Local Environment Issues:** Ensure OpenCV is correctly installed and paths are correct. Use `cv2.imshow()` and ensure you call `cv2.waitKey(0)` and `cv2.destroyAllWindows()` to manage the display window.

Conclusion

This document outlines the detailed approach to regularizing, analyzing symmetry, and completing curves from line art. The provided code snippets serve as a foundation to accomplish these tasks effectively, with the output visualized in SVG format and as completed images.

Team Members:

1. Vaibhav Rai (+91 8171633565)
2. Utakrsh Jain (+91 7425094343)