ADOBE
GENSOLVE

Curvetopia

Create the future. **Own the outcome.** **Raise the bar.** **Be genuine.**

Presented by:

Yashwardhan Khanna
Balaji P
Johan Mathew Joseph

SRM Institute of Technology, Kattankulathur

Table of Contents

Identification & Regularization of Shapes

The **RDP algorithm** is pivotal in our code. It works by recursively breaking down a polyline into segments and eliminating points that do not significantly affect the shape of the curve. Here's how it operates:

Distance Calculation:

The algorithm measures how far each point deviates from the straight line connecting the polyline's endpoints. This perpendicular distance helps determine which points can be removed. Mathematically, for a point PPP and a line segment defined by points AAA and BBB, the distance is:

$$d = \frac{|(B_y - A_y) \cdot P_x - (B_x - A_x) \cdot P_y + B_x \cdot A_y - B_y \cdot A_x|}{\sqrt{(B_y - A_y)^2 + (B_x - A_x)^2}}$$

Here, d represents the perpendicular distance, and this formula helps in assessing how well a point fits on the straight line segment.

Recursive Simplification:

The RDP algorithm removes points based on their perpendicular distance from the line segment. If any point's distance exceeds a threshold ϵ\epsilonϵ, it's kept, and the simplification is applied recursively to the adjacent segments. This process continues until only the points essential for shape accuracy remain, resulting in a simpler polyline.

Straightening Polylines: Post-simplification, polylines are straightened by sorting points within each segment to ensure smoothness and continuity, thus aligning the segments more coherently.

Contour Classification

The code leverages contour approximation to classify shapes. Using **cv2.approxPolyDP**, it simplifies contours by determining how closely the points on the contour fit to a straight line segment defined by the endpoints. The parameter epsilon, proportional to the contour's arc length, is crucial for this approximation. Shapes are classified based on the number of vertices and geometric properties:

- **Triangles and Squares:** Directly identified by the number of vertices.
- **Rectangles and Circles:** Distinguished by the aspect ratio and circularity. Circularity is computed using the formula:

$$\text{Circularity} = \frac{4\pi \times \text{Area}}{\text{Perimeter}^2}$$

where a higher value indicates a shape closer to a circle.

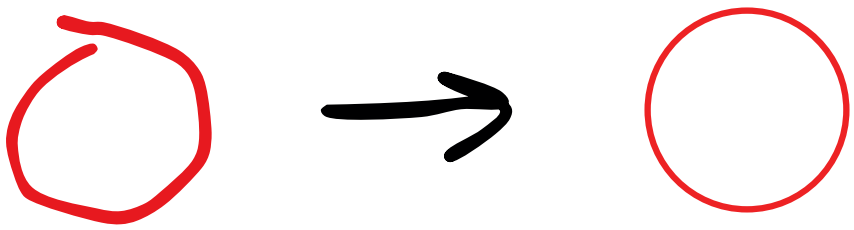
Drawing Regular Shapes

For identified shapes like circles, squares, and polygons, the code computes and draws idealized shapes:

- **Circles:** Derived from the contour's centroid and area, using:

$$\text{Radius} = \sqrt{\frac{\text{Area}}{\pi}}$$

- **Squares and Rectangles:** Constructed from the bounding box dimensions.
- **Stars and Polygons:** Calculated based on vertex count and specific geometric formulas, ensuring accurate representation of the shape.



Optional Task

Converting Image to set of Polylines

Contour Extraction:

- The code first converts the image to grayscale and applies Canny edge detection to find edges.
- Contours are extracted from these edges using **cv2.findContours**.

Saving Contours:

- Contours are saved to a CSV file with coordinates adjusted for image height to match the correct orientation.

Visualization:

- The CSV data is read back and plotted using Matplotlib.
- Each contour is visualized as a polyline, with different colors for clarity.

Identification of lines of symmetry in Shapes

The main idea is to check whether the shape inside a contour is symmetric about various axes: vertical, horizontal, and the two diagonals. This is done by dividing the shape into two halves and comparing these halves.

1

Vertical Symmetry:

- The shape is divided into a left half and a right half.
- The right half is flipped horizontally to align with the left half.
- The symmetry score is calculated as the sum of absolute differences (cv2.absdiff) between the corresponding pixels of the two halves. If the difference is minimal (i.e., less than a threshold like 1e5), the shape is considered vertically symmetric.

2

Horizontal Symmetry:

- The shape is divided into a top half and a bottom half.
- The bottom half is flipped vertically to align with the top half.
- The symmetry score is calculated similarly to the vertical symmetry.

3

Diagonal Symmetry:

- The shape is transposed (cv2.transpose), which effectively rotates it by 90 degrees, and then flipped along the other diagonal axis.
- This process is repeated for both the major and minor diagonals, and symmetry is checked in the same manner as before.

Key Functions

- **cv2.absdiff:** This function computes the absolute difference between two arrays (images in this case), pixel by pixel. The sum of these differences gives a quantitative measure of how similar (or different) the two halves of the shape are.
- **Flipping and Transposing:**
- Flipping (cv2.flip) allows for aligning one half of the shape with the other, facilitating direct comparison.
- Transposing (cv2.transpose) and then flipping the shape allow for checking diagonal symmetries, as it rotates the shape in a way that aligns diagonally opposed halves.

Mathematical Representation

Let I1 and I2 be the two halves of the image (or shape) that we want to compare for symmetry. The absolute difference DDD between these two halves is given by:

$$D(x, y) = |I_1(x, y) - I_2(x, y)|$$

Where:

- (x,y) are the coordinates of a pixel in the image.
- I1(x,y) and I2(x,y) represent the pixel intensity values at coordinate (x,y) in the two halves.

Symmetry Score: The overall symmetry score S is computed by summing the absolute differences over all pixels:

$$S = \sum_{x,y} D(x, y) = \sum_{x,y} |I_1(x, y) - I_2(x, y)|$$

If S is close to zero (or below a specified threshold ϵ), the halves I1 and I2 are considered symmetric.

Curve Completion

For Circles

Circle Detection (Hough Transform)

- **Intuition:** Detecting circles in an image by identifying the center (a,b) and radius r using edge points.
- cv2.HoughCircles detects circles by finding local maxima in a transformed space.

$$(x-a)^2 + (y-b)^2 = r^2$$

Finding Missing Arc

- Identifying the angles of the detected circle contour to find gaps.
- **Formulas:**
  - **Angle Calculation:** For each contour point (x,y)

$$\theta = \text{atan2}(y-b, x-a)$$

  - **Angle Conversion:** Convert to degrees:

$$\theta_{\text{degrees}} = \theta \times \frac{180}{\pi}$$

  - **Normalize Angles:** Ensure angles are within [0,360)

$$\theta_{\text{mod}} = (\theta_{\text{degrees}} + 360) \mod 360$$

Completing the Missing Part

- Drawing the missing arc between the identified start and end angles.
- cv2.ellipse draws the missing arc using the calculated angles

Curve Completion

For Squares

The code detects and completes a square in an image by leveraging edge and line detection techniques. It starts by converting the image to grayscale and then uses the Canny edge detection algorithm to identify edges based on intensity changes. The Hough Transform is then employed to detect straight lines within the edge-detected image. By examining the endpoints of these detected lines, the code determines the extremities of the shape and draws lines to complete the square if any sides are missing.

Canny Edge Detection:

- **Gradient Magnitude:** Magnitude

$$\sqrt{G_x^2 + G_y^2}$$

- **Gradient Direction:**

$$\arctan\left(\frac{G_y}{G_x}\right)$$

Non-Maximum Suppression and Thresholding are used to finalize edge detection.

Hough Line Transform:

- Line Representation: $\rho = x\cos(\theta) + y\sin(\theta)$
- The accumulator space tracks potential lines by voting in the parameter space (ρ, θ) for each edge point.

And that’s a Wrap.