

Polygonal Approximation of objects' contours

Image Processing Project

Sofia Dobra

Department of Computer Science
Technical University of Cluj-Napoca

June 1, 2025

Overview

1. Introduction
2. Contour Representation in Images
3. Polygonal Approximation Algorithms
4. General Approach
5. Future Directions
6. Implementation

Introduction

Polygonal approximation is a technique used in computer vision and image processing to simplify the representation of object contours while maintaining essential shape features. The goal is to approximate an object's contour with a series of polygonal line segments, reducing complexity while preserving accuracy within a defined precision threshold.

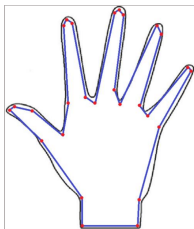


Figure: contour;
vertices; edges.
[ResearchGate].

Goal

Given a curve $C : f(x, y) = 0 \rightarrow$ polygon that closely approximates C with error smaller than ϵ and having a number of vertices as small as possible.

Contour Representation in Images

Contours are the boundaries of objects detected in an image. Typically, contours are extracted using edge detection techniques such as:

- **Canny Edge Detection:** Identifies strong edges in an image.
- **Sobel and Prewitt Filters:** Detect edges using gradients.
- **Thresholding and Morphological Operations:** Used in binary images to extract object boundaries.



Figure: Original image

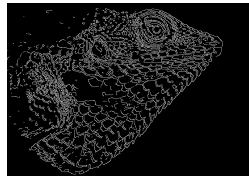


Figure: Contour image

Polygonal Approximation Algorithms

Several algorithms exist for approximating contours with polygonal segments. Some common methods include:

- Ramer-Douglas-Peucker (RDP) Algorithm
- Visvalingam-Whyatt Algorithm
- Minimum Perimeter Polygon (MPP)

RDP Algorithm

- A widely used recursive approach for simplifying polylines.
- Works by identifying critical points that deviate significantly from a straight line.
- Process:
 1. Find the straight line connecting the first and last points of the contour.
 2. Identify the point that deviates the most from this line.
 3. If the deviation is greater than a user-defined threshold, the contour is split at this point.
 4. Repeat recursively on the resulting sub-contours until all deviations are within the threshold.

Pseudocode [Wikipedia contributors, 2024a]

```
function DouglasPeucker(PointList[], epsilon)
# Find the point with the maximum distance
dmax = 0
index = 0
end = length(PointList)
for i = 2 to (end - 1) {
    d = perpendicularDistance(PointList[i], Line(PointList
        [1], PointList[end]))
    if (d > dmax) {
        index = i
        dmax = d
    }
}

ResultList[] = empty;
```

Pseudocode - contd.

```
# If max distance is greater than epsilon, recursively
simplify
if (dmax > epsilon) {
    # Recursive call
    recResults1[] = DouglasPeucker(PointList[1...index],
                                    epsilon)
    recResults2[] = DouglasPeucker(PointList[index...end],
                                    epsilon)

    # Build the result list
    ResultList[] = {recResults1[1...length(recResults1) - 1],
                    recResults2[1...length(recResults2)]}
} else {
    ResultList[] = {PointList[1], PointList[end]}
}
return ResultList[]
```


Visvalingam-Whyatt Algorithm

- Points are assigned an importance based on local conditions, and points are removed from the least important to most important.
- In Visvalingam's algorithm, the importance is related to the triangular area added by each point.

Detailed Approach

Given a chain of 2D points $\{p_i\} = \left\{ \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right\}$, the importance of each interior point is determined by the **triangle area** formed with its neighbors, computed via a matrix determinant or: $A_i = \frac{1}{2} |x_{i-1}y_i + x_iy_{i+1} + x_{i+1}y_{i-1} - x_{i-1}y_{i+1} - x_iy_{i-1} - x_{i+1}y_i|$. The **least important point** p_i is removed, updating A_{i-1} and A_{i+1} , until the required number of points remains or further removal is unjustified.

MPP Algorithm [Wikipedia contributors, 2024b]

- The algorithm aims to **minimize the total perimeter** of the approximated polygon, unlike methods such as Ramer-Douglas-Peucker, which focus on error minimization.
- A **greedy** or **dynamic programming** approach can be used to iteratively remove points while checking the effect on the perimeter.
- *Iterations*: Remove the point that contributes the least to the perimeter while keeping the error within a predefined threshold. Use heuristics such as **triangle area minimization** or **distance from line segments** to decide which point to remove.
- *Stopping condition*: Continue until removing another point would exceed the user-defined approximation error.

Pseudocode

```
function MPP_Approx(P[], threshold)
    while true {
        min_inc, idx = Inf, -1
        for i = 2 to length(P) - 1 {
            p1, p2, p3 = P[i-1], P[i], P[i+1]
            if (dist(p1, p2) + dist(p2, p3) - dist(p1, p3) <
                min_inc
                and error(P, p1, p2, p3) <= threshold) {
                min_inc, idx = dist(p1, p2) + dist(p2, p3) -
                    dist(p1, p3), i
            }
        }
        if idx == -1 break
        remove(P, idx)
    }
    return P
```

General Approach

Step 1: Preprocess the Image

- Convert the image to grayscale if necessary.
- Apply edge detection (e.g., Canny) to extract contours.
- Use contour detection methods (e.g., OpenCV's `cv.findContours()`).

Step 2: Apply a polygonal approximation algorithm

- Choose an appropriate algorithm (e.g., RDP for efficient approximation).
- Set a user-defined precision threshold to control the level of simplification.
- Approximate the contour using the selected method.

General Approach - contd.

Step 3: Evaluate the approximation

- Compare the approximated contour with the original contour.
- Measure accuracy using metrics like:
 - **Hausdorff Distance**: Measures the worst-case deviation.
 - **Frechet Distance**: Evaluates similarity between curves.
 - **Vertex Count Reduction**: Compares the number of points before and after approximation.

Step 4: Visualize and Analyze Results

- Overlay the approximated contour on the original image.
- Allow user interaction to adjust the precision threshold dynamically.

Future Directions

- Generalization the idea of the idea of linear approximation (polygonal edges) to quadratic/spline approximation, Bézier curves etc.



Figure: Bézier curves used in representing the alphabet letters. [source]

Implementation Overview

The notebook implements and compares three polygonal approximation algorithms. It includes:

- Loading and preprocessing grayscale contour images.
- Interactive display for adjusting simplification parameters.
- Individual implementations for:
 - Douglas-Peucker (RDP)
 - Visvalingam-Whyatt (VW)
 - Minimum Perimeter Polygon (MPP)
- Visualization of results with color-coded overlays.

Contour Preprocessing: Border Trace Algorithm

Before applying polygonal approximation, a custom contour extraction function is used to highlight object borders:

Moore Neighbor Tracing

A classic **8-connected Moore-neighbor following** algorithm is used to trace the first black (foreground) contour in a binary image:

- Searches for the first black pixel → starting point.
- Follows the border by rotating clockwise through 8-neighbor offsets.
- The traced contour is drawn in white on a black background.

```
dst[p_curr] = 255 # mark traced pixels as white
if p_curr == p1 and p_prev == p0 and n >= 2:
    break # closed contour found
```

This process creates simplified contour images, stored in the `contour_imgs/` folder, which are later used for polygonal approximation.

Douglas-Peucker Implementation

A recursive function retains points that are farther than ϵ from the line segment:

```
def douglas_peucker(pts, eps):  
    ...  
    if dmax > eps:  
        left = douglas_peucker(pts[:index+1], eps)  
        right = douglas_peucker(pts[index:], eps)  
        return left[:-1] + right  
    else:  
        return [start, end]
```

This method preserves significant corner-like features.

Visvalingam-Whyatt with Ratio

A min-heap ranks points by triangle area. Points are removed until a percentage of the original points remains:

```
def visvalingam_whyatt(pts, ratio):  
    ...  
    while remaining > target:  
        point = heapq.heappop(heap)  
        if not removed[point.index]:  
            removed[point.index] = True
```

This ensures a consistent simplification degree across all images.

Minimum Perimeter Polygon (MPP)

Points are removed if doing so significantly reduces the perimeter and their perpendicular error is within tolerance:

```
def minimum_perimeter_polygon(pts, eps):  
    ...  
    if (gain < min_gain) and (error <= eps):  
        min_gain, idx = gain, i
```

The loop continues until no more removable points are found.

Interactive Visualization

Users can explore simplifications dynamically:

```
widgets.interactive_output(  
    interactive_display,  
    {'image_index': image_slider, 'epsilon': epsilon_slider, '  
    ratio': ratio_slider}  
)
```

Each algorithm is drawn in a separate color:

- Red: RDP
- Green: VW
- Blue: MPP

Interactive Interface

- **Image Slider** – Selects test image.
- **Epsilon Slider** – Sets ϵ for:
 - Douglas-Peucker (DP)
 - MPP
- **VW Ratio Slider** – Sets r for:
 - Visvalingam-Whyatt (VW)

Image: 0
Epsilon: 21.00
VW Ratio: 0.17
File: horizontal_ellipse.bmp | Epsilon: 21.0 | Ratio: 0.17
horizontal_ellipse.bmp
Red: DP ($\epsilon=21.0$) | Green: VW ($r=0.17$) | Blue: MPP ($\epsilon=21.0$)

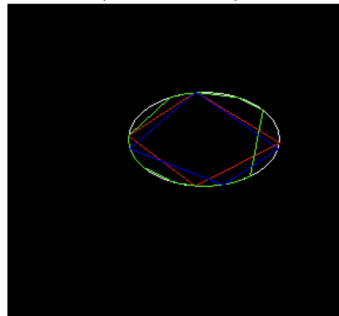


Figure: Interactive display

Results

- A small value of the number of points ratio gives poor results. On the other hand, a smaller ϵ gives more accurate results (see Fig. 6)
- VW finds a very good approximation by keeping only half of the points (Fig. 7).
- Polygonal objects (Fig. 8): For a quite big value of ϵ RDP and MPP find very good approximating polygons; in contrast, VW does not find the optimal points.

object_holes.bmp
Red: DP ($\epsilon=13.5$) | Green: VW ($r=0.01$) | Blue: MPP ($\epsilon=13.5$)



Figure: Irregular Object

skew_ellipse.bmp
Red: DP ($\epsilon=13.5$) | Green: VW ($r=0.54$) | Blue: MPP ($\epsilon=13.5$)

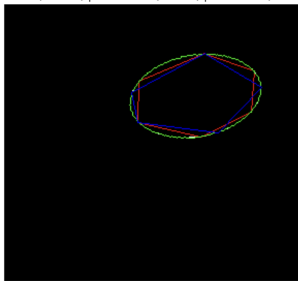


Figure: Ellipse

star_R90.bmp
Red: DP ($\epsilon=18.0$) | Green: VW ($r=0.17$) | Blue: MPP ($\epsilon=18.0$)

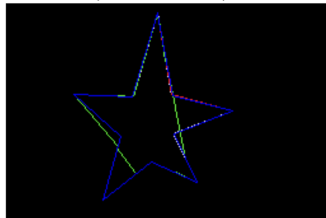
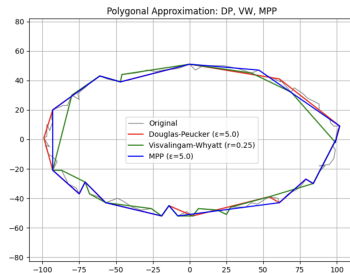


Figure: Star


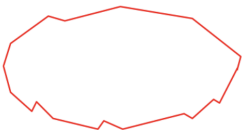

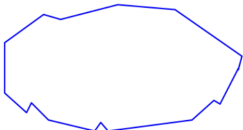
Results - contd.

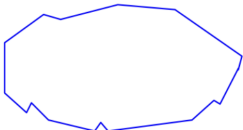
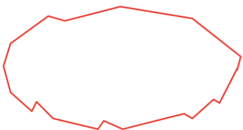
Contour generation

A synthetic noisy ellipse was generated using 100 sample points with added Gaussian noise. Each simplification algorithm was applied using common parameters: $\epsilon = 5.0$ for RDP and MPP; Retention Ratio = 0.25 for VW. The results were visualized side by side to assess performance.



Polygonal Simplification Algorithms

Original (100 pts)	Douglas-Peucker ($\epsilon=5.0$) 20 pts	VW ($r=0.25$) 25 pts	MPP ($\epsilon=5.0$) 18 pts
			



References



Wikipedia contributors (2024a).

Ramer–douglas–peucker algorithm — Wikipedia, the free encyclopedia.

[Online; accessed 24-March-2025].



Wikipedia contributors (2024b).

Visvalingam–whyatt algorithm — Wikipedia, the free encyclopedia.

[Online; accessed 24-March-2025].