Efficient Computation of Crossing Components and Shortcut Hulls

Zohaib Aslam - za08134
M Mansoor Alam - ma08322

Problem

Simplifying complex polygon shapes efficiently while preserving their outer structure.

Why It Matters

Essential in computer graphics, GIS, motion planning, and visibility analysis.

Challenges

Finding non-overlapping shortcuts among polygon edges is computationally expensive.

Real World Examples

- Simplifying maps for mobile devices.
- Visualizing geographic shapes efficiently.

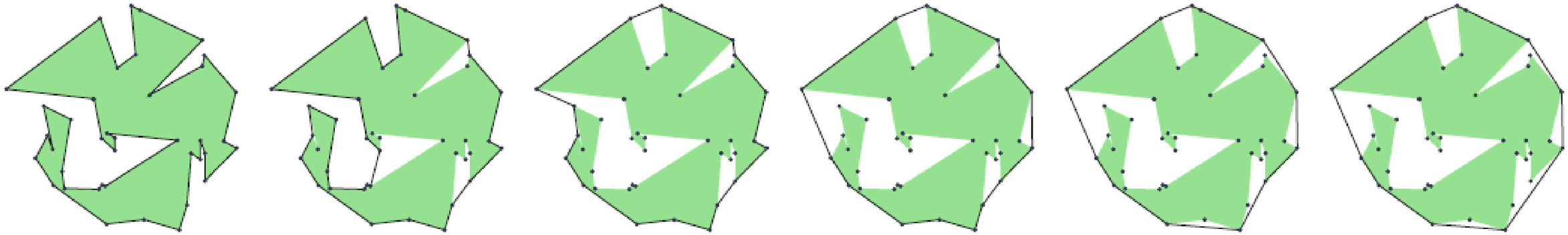


Fig. 1. Input polygon (green) and shortcut hulls (black outline) with varying level of detail. The right one coincides with the convex hull of the polygon. (Color figure online)

Problem Overview: Shortcut Hull

Inputs:

- A polygon with n vertices
- A set of valid shortcut edges (non-intersecting, from exterior visibility graph)

Goal:

- Compute the "Shortcut Hull" — a simplified polygon that fully contains the original shape

Constraints:

- Hull must not cross the original polygon’s boundary
- Only valid shortcuts from the exterior visibility graph can be used

Related Algorithms

Earlier Approaches

- Edge intersection detection: Bentley & Ottmann (1979), Chazelle & Edelsbrunner (1992).
- Visibility graphs: Used in robot motion planning and rendering.

Limitations

- Naive method for crossing component computation is slow (O(n^4))
- Assumes crossing components are precomputed.

How This Work Differs

- Designed an algorithm that reports all k crossings in O(n + m + k)
- Proposed a new method for crossing component computation that speeds up the running time from O(n^4) to O(n^2)
- Faster and efficient computation than previous algorithms

Proposed Solution

What's Novel About The Approach

- Introduced algorithms that speed up edge crossing detection and identification of crossing component
- New Data Structure - Crossing Component Hierarchy - Efficiently partitions the polygon

Core Idea

- Use matrix structure to reduce checks - O(n + m + k)
- Replace full intersection graph with a lightweight pseudo-graph - Reduced to O(n^2)
- Use a hierarchical tree to organize components and speed up region processing
- Break polygon into smaller "pockets" using convex hull → solve independently

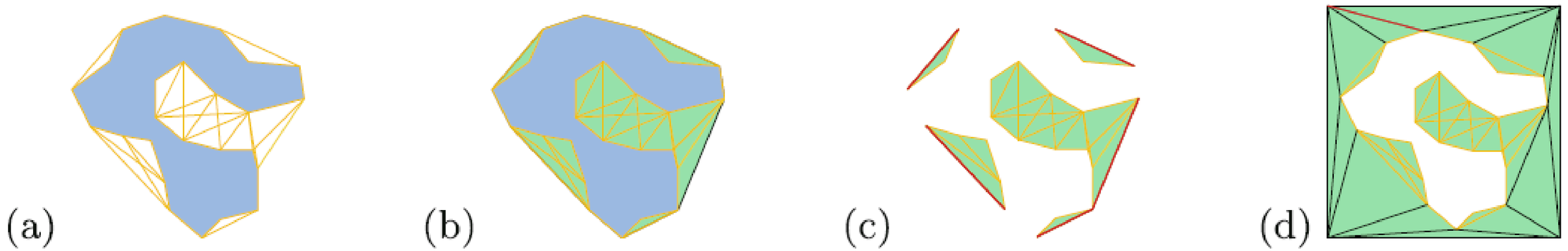
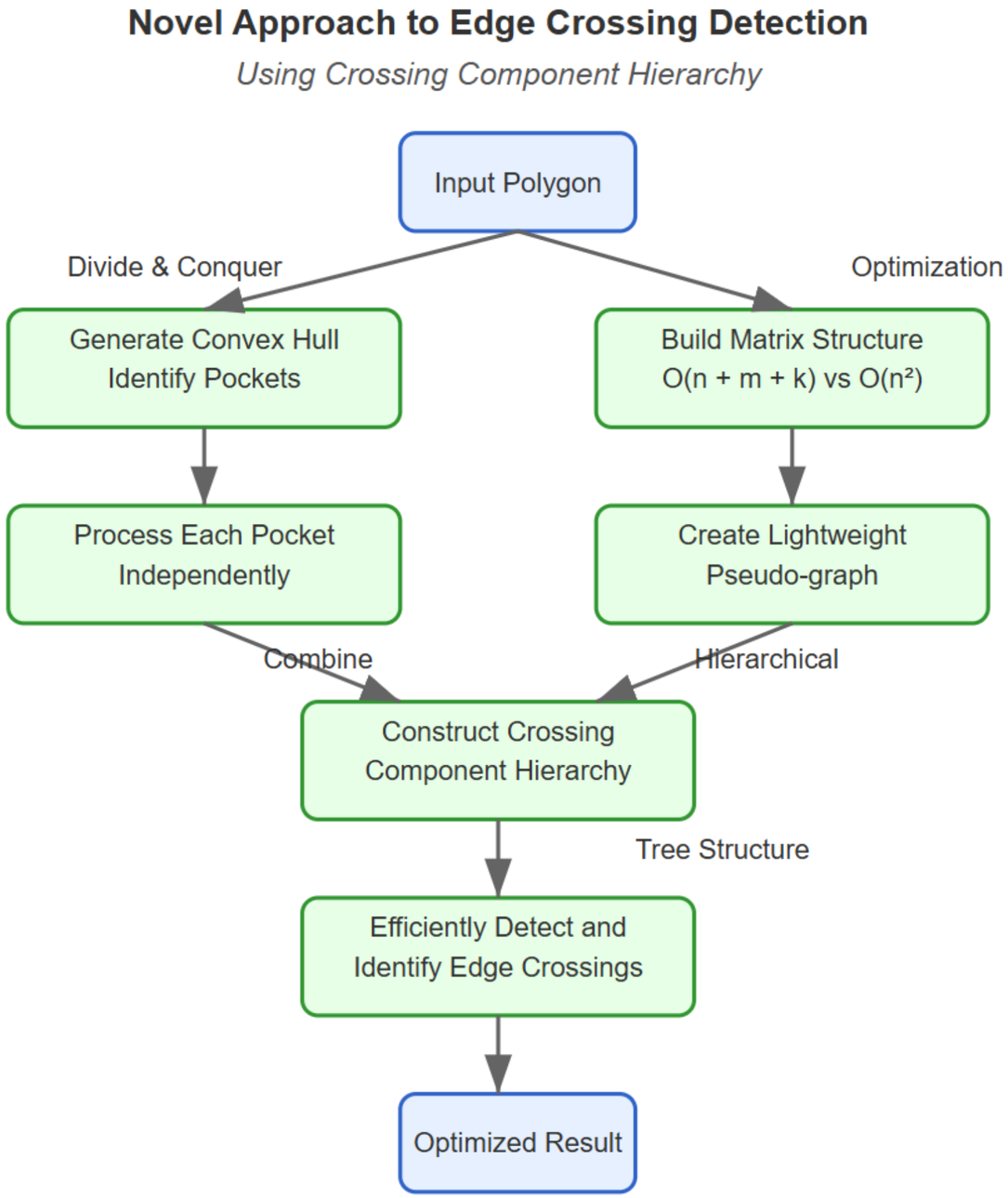


Fig. 2. Segmentation of a polygon P into smaller problems. (a) P with shortcut candidates $\mathcal{C}$. (b) Convex hull of P. (c) Our approach: pockets of the convex hull. (d) Sliced donut approach from [5] for comparison.

Flow Chart


Novel Approach to Edge Crossing Detection
Using Crossing Component Hierarchy
Input Polygon
Divide & Conquer
Optimization
Generate Convex Hull
Identify Pockets
Build Matrix Structure
O(n + m + k) vs O(n²)
Process Each Pocket
Independently
Create Lightweight
Pseudo-graph
Combine
Hierarchical
Construct Crossing
Component Hierarchy
Tree Structure
Efficiently Detect and
Identify Edge Crossings
Optimized Result

Algorithm Overview

Efficient Shortcut Hull Computation using Crossing Components

Step-by-Step Breakdown

1. Input Preparation
   - Start with a polygon P and a set of shortcut candidates C (from the visibility graph).
2. Edge Crossing Detection
   - Traverse an implicit matrix of edge pairs.
   - Use vertex order and adjacency lists to detect crossings efficiently.
   - Time Complexity: $O(n + m + k)$
   - n = vertices, m = candidate shortcuts, k = number of crossings

Step-by-Step Breakdown

3. Build Pseudo-Intersection Graph (GP)
   - Don't build a full graph of all intersecting edge pairs.
   - Instead, build a simpler graph that still captures all connected components of crossings.
4. Extract Crossing Components
   - Use DFS or BFS on GP to group edges that intersect.
5. Form Regions
   - For each crossing component, create a minimal enclosing polygonal region.
   - Use convex chains and visibility rules to avoid overlaps in boundaries.
6. Construct Crossing Component Hierarchy
   - Build a tree structure based on how regions are nested.
   - This helps organize the final shortcut selection process.

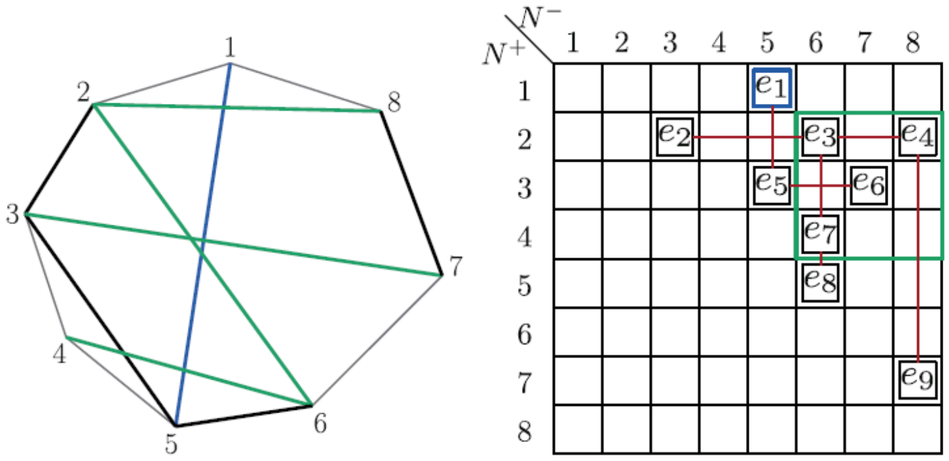


Fig. 3. Subset of a polygon's visibility graph with corresponding adjacency matrix

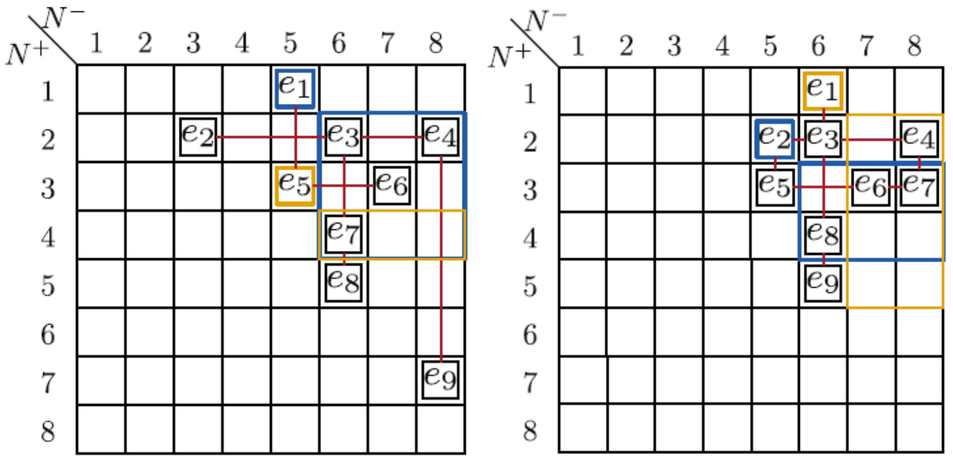
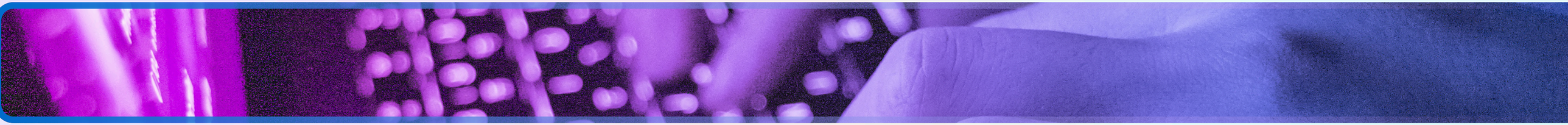


Fig. 4. Adjacency matrix with rectangles that allow for optimization of the algorithm

Step-by-Step Breakdown

7. Dynamic Programming for Final Shortcut Hull
   - Optimize the final set of shortcuts (non-crossing) using DP.
   - Goal: Minimize the cost function
   - cost = $\lambda \cdot$ perimeter + $(1 - \lambda) \cdot$ area
   - where $\lambda \in [0, 1]$ balances shape compactness vs. area.

Complexity Analysis



Aspect	Previous Algorithm	New Algorithm	Improvement
Edge Crossings	$O(m^3 / n^2)$	$O(n + m + k)$	Significant reduction
Crossing Components	$O(n^4)$	$O(\min\{n + m + k, n^2\})$	Reduced to quadratic or better

Experimental Setup

Environment:

- Input: Simple polygons, predefined shortcut candidates.
- Output: Shortcut hulls.

Tools:

- Implemented and analyzed with geometric test cases

Metrics:

- Running time, correctness.

Results

Findings:

- Significantly better than naive crossing component computation (n^4 to n^2)
- Faster in computing edge crossings as well (m^3 / n^2 to n + m + k)
- Still guarantees optimal shortcut hulls

Strengths:

- Scales well even with large number of vertices (n) and many crossings (k)
- Better memory overhead due to construction of a pseudo-intersection graph

DEMO

Limitations & Future work

Limitations:

- Assumes simple polygons - Doesn't handle polygons with self-intersections
- Fixed λ parameter - Algorithm needs to re run if different λ value has to be tried. No dynamic tuning built in

Future Work:

- Extend to polygons with holes or self-intersections
- Implement dynamic changing of λ parameter
- Implement and test on real-world datasets for GIS, robotics, and graphics.

Conclusion

Recap:

- We addressed the bottleneck in shortcut hull computation.
- Proposed faster algorithms for edge crossings and component detection.

Key Takeaway:

- You can simplify polygons efficiently now, even at scale.

References:

Efficient Computation of Crossing Components and Shortcut Hulls (Bonerath et al. 2023)

Thank You.

Thank You.

Thank You.