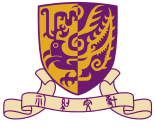
IEEE/ACM ICCAD 2025 INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN 44th Edition

G-Contour: GPU Accelerated Contour Tracing For Large-Scale Layouts

Shuo Yin[1], Jiahao Xu[1], Jiaxi Jiang[1], Mingjun Li[1], Yuzhe Ma[2], Tsung-Yi Ho[1], Bei Yu[1]

[1]The Chinese University of Hong Kong
[2]Hong Kong University of Science and Technology (Guangzhou)
{syin22}@cse.cuhk.edu.hk

October 27, 2025

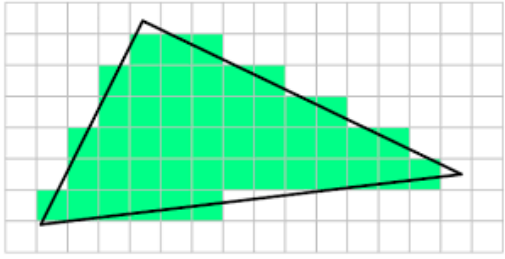


Outline

Introduction

Rasterization

The process of converting a vector graphics image, composed of shapes, into a 2D image.



The rasterization of a triangle.

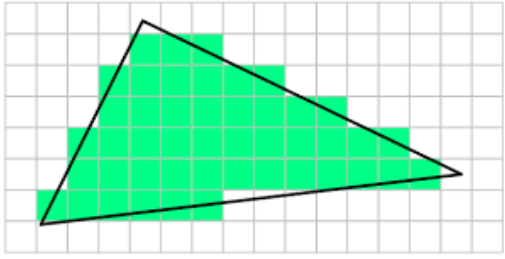
Mask rasterization for ILT: [DAC'15][1]

What is the inverse process of rasterization?

[1] **Yixiao Ding, Chris Chu, and Xin Zhou (2015). "An efficient shift invariant rasterization algorithm for all-angle mask patterns in ILT".** In: *Proceedings of the 52nd Annual Design Automation Conference*, pp. 1–6.

Rasterization

The process of converting a vector graphics image, composed of shapes, into a 2D image.



The rasterization of a triangle.

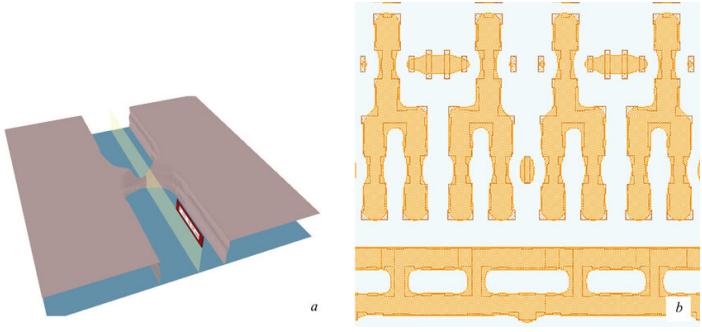
Mask rasterization for ILT: [DAC'15][1]

What is the inverse process of rasterization? Contour Tracing

[1] **Yixiao Ding, Chris Chu, and Xin Zhou (2015). "An efficient shift invariant rasterization algorithm for all-angle mask patterns in ILT".** In: *Proceedings of the 52nd Annual Design Automation Conference*, pp. 1–6.

Why Contour Tracing is Important for DFM?

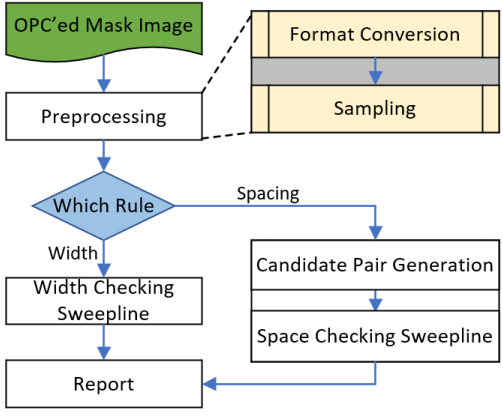
1. Print image should be in GDSII format.



(a) 3D resist computation using "full" simulation; (b) resist contour computation using compact models (Mentor Graphics Calibre CAD).

Why Contour Tracing is Important for DFM?

2. Mask Rule Checking (MRC) should take GDSII as input.



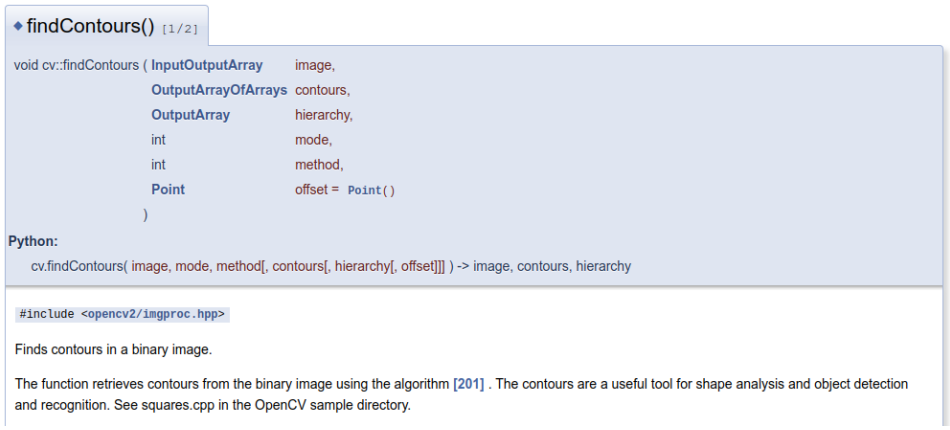
Overall flow of mask rule checking.

Contour Tracing Algorithms

Most contour tracing algorithms are based on BFS (Breadth-First Search) walking.

- Square Tracing Algorithm
- Moore-Neighbor Tracing
- Radial Sweep
- Theo Pavlidis' Algorithm

Contour Tracing on CPU



◆ findContours() [1/2]

```
void cv::findContours ( InputOutputArray  image,
                        OutputArrayOfArrays  contours,
                        OutputArray  hierarchy,
                        int  mode,
                        int  method,
                        Point  offset = Point()
                      )
```

Python:

```
cv.findContours( image, mode, method[, contours[, hierarchy[, offset]]] ) -> image, contours, hierarchy
```

```
#include <opencv2/imgproc.hpp>
```

Finds contours in a binary image.

The function retrieves contours from the binary image using the algorithm [201] . The contours are a useful tool for shape analysis and object detection and recognition. See squares.cpp in the OpenCV sample directory.

The `cv::findContour` API in OpenCV.[2]

[2] **Satoshi Suzuki et al. (1985). "Topological structural analysis of digitized binary images by border following".** In: *Computer vision, graphics, and image processing* **30.1, pp. 32–46**.

The Needs of Contour Tracing on GPU

- Extract optimized mask for later manufacturing
  1. Most ILT algorithms perform on GPU: [DATE'21][3], [ICCAD'20][4], [DAC'18][5], ...
  2. We can avoid data transfer between CPU and GPU.
- Preserving lithography simulation results
  1. The full-chip simulation result is too large to preserve in image format.
  2. Use KLayout[6] for efficient handling and visualization.
- More reliable wafer image evaluation (EPE, L_2, ...), mask rule checking

[3] **Ziyang Yu et al. (2021). "A GPU-enabled Level Set Method for Mask Optimization".** In: *Proc. DATE*.

[4] **Bentian Jiang et al. (2020). "Neural-ILT: Migrating ILT to Nerual Networks for Mask Printability and Complexity Co-optimizaton"".** In: *Proc. ICCAD*.

[5] **Haoyu Yang et al. (2018). "GAN-OPC: Mask Optimization with Lithography-guided Generative Adversarial Nets".** In: *Proc. DAC*, 131:1–131:6.

[6] ***KLayout* (2024).** `https://www.klayout.de/`.

Polygon Representation

Left Hand Rule

When traversing the edges of a polygon, the foreground will always be on the left side of the traversal direction.

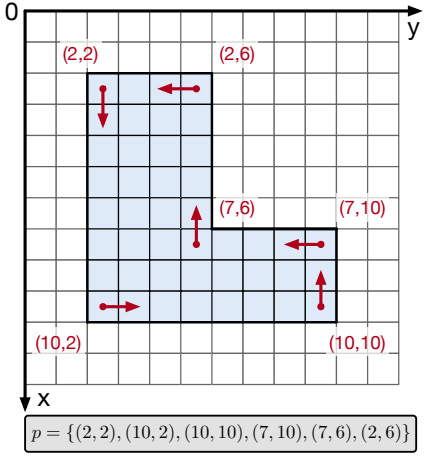


Illustration of polygon representation in the context of contour tracing.

G-Contour Framework

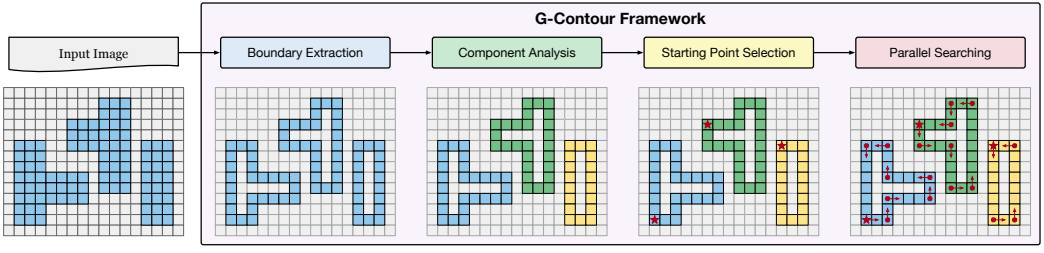
Contour Tracing

Given a layout image $\boldsymbol{I}$, the goal of contour tracing is to identify all the contours in the counterclockwise direction of the patterns within the layout, represented as the set $\mathcal{C}$.

G-Contour Framework

Contour Tracing

Given a layout image $\boldsymbol{I}$, the goal of contour tracing is to identify all the contours in the counterclockwise direction of the patterns within the layout, represented as the set $\mathcal{C}$.



The overall flow of G-Contour.

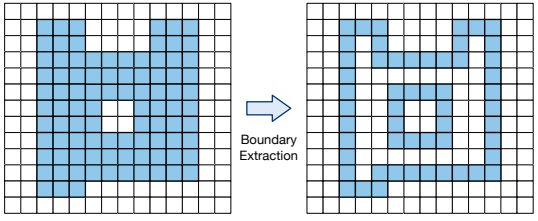
Boundary Extraction

Boundary

The boundary of a layout image $\boldsymbol{I}$ is defined as the set of pixels that separate the foreground and background of the image.

$$\mathcal{B} = \{I(x,y)| \quad \exists|\delta x + \delta y| = 1, s.t.\ I(x+\delta x, y+\delta y) = 0\} \tag{1}$$

$$\partial \boldsymbol{I} = \nabla f(\boldsymbol{I}) = \begin{cases} \partial I(x,y) < T, \forall I(x,y) \notin \mathcal{B} \\ \partial I(x,y) \geq T, \forall I(x,y) \in \mathcal{B} \end{cases} \tag{2}$$



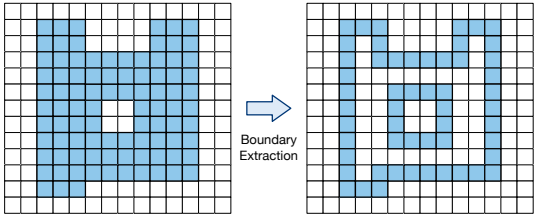
Boundary Extraction

Boundary

The boundary of a layout image $\boldsymbol{I}$ is defined as the set of pixels that separate the foreground and background of the image.

$$\mathcal{B} = \{I(x,y)| \quad \exists|\delta x + \delta y| = 1, s.t.\ I(x+\delta x, y+\delta y) = 0\} \tag{1}$$

$$\partial \boldsymbol{I} = \nabla f(\boldsymbol{I}) = \begin{cases} \partial I(x,y) < T, \forall I(x,y) \notin \mathcal{B} \\ \partial I(x,y) \geq T, \forall I(x,y) \in \mathcal{B} \end{cases} \tag{2}$$



We launch a 2D kernel for each pixel to extract the boundary image, $\partial \boldsymbol{I}$, in parallel.

Parallel Component Analysis

Connected Component

A connected component is defined as a set of pixels forming a foreground, where for any two pixels (x_i, y_i) and (x_j, y_j) in this set, there exists a path composed of pixels that also belong to the set.

Each boundary in the boundary image $\partial \boldsymbol{I}$ is a connected component.

The union-find structure maintains a label image $\partial \boldsymbol{I}'$, where each pixel in $\partial \boldsymbol{I}'$ is assigned a unique label.

- `find` (x_i, y_i) : Returns the label of the component to which the pixel (x_i, y_i) belongs.
- `union` $((x_i, y_i),\ (x_j, y_j))$: Joins the components containing (x_i, y_i) and (x_j, y_j) with the smaller label.

Parallel Component Analysis

Input: Boundary image $\partial \boldsymbol{I}$ with shape $[H, W]$.
Output: Labeled image $\partial \boldsymbol{I'}$ with shape $[H, W]$.

```
1: // Merge two labels under atomic lock
2: function union(∂I', (x_i, y_i), (x_j, y_j))
3:     done ← false;
4:     while done = false do
5:         label_i ← find(∂I', (x_i, y_i));
6:         label_j ← find(∂I', (x_j, y_j));
7:         if label_i < label_j then
8:             x_j, y_j ← Index^{-1}(label_j);
9:             ∂I'[x_j][y_j] ← atomicMin(∂I'[x_j][y_j], label_i);
10:        else if label_i > label_j then
11:            x_i, y_i ← Index^{-1}(label_i);
12:            ∂I'[x_i][y_i] ← atomicMin(∂I'[x_i][y_i], label_j);
13:        else
14:            done ← true;
```

```
1: function kernel(∂I, ∂I')
2:     id_x ← blockIdx.x * blockDim.x + threadIdx.x;
3:     id_y ← blockIdx.y * blockDim.y + threadIdx.y;
4:     if id_x < H and id_y < W then
5:         ∂I'[id_x][id_y] ← Index(id_x, id_y); // Initialize the
   label
6:         cudaSyncThreads();
7:         for (x, y) ∈ neighbors of (id_x, id_y) do
8:             if ∂I[x][y] = ∂I[id_x][id_y] then
9:                 union(∂I', (id_x, id_y), (x, y));
```

Inflection Point

An **inflection point** is defined as a pixel on the boundary, where the direction of its neighbors changes.

An inflection point on the component is a vertex on the contour.

$$\forall\, |\delta x + \delta x'| + |\delta y + \delta y'| = 0$$
$$s.t. \quad \partial \boldsymbol{I}'(x + \delta x, y + \delta y) \neq \partial \boldsymbol{I}'(x + \delta x', y + \delta y'),$$

where $(\delta x, \delta y)$ is the unit displacement of one direction, and $(\delta x', \delta y')$ is the unit displacement of the opposite direction.

Starting Point Selection

Inflection Point

An **inflection point** is defined as a pixel on the boundary, where the direction of its neighbors changes.

An inflection point on the component is a vertex on the contour.

$$\forall\, |\delta x + \delta x'| + |\delta y + \delta y'| = 0$$
$$s.t. \quad \partial \boldsymbol{I}'(x + \delta x, y + \delta y) \neq \partial \boldsymbol{I}'(x + \delta x', y + \delta y'),$$

where $(\delta x, \delta y)$ is the unit displacement of one direction, and $(\delta x', \delta y')$ is the unit displacement of the opposite direction.

We use atomic operations to select one inflection point as the starting point for each component in parallel. (1D kernel)

Rotation

The **rotation** ϕ of the foreground is defined as a circular transformation, where the source moves clockwise from one directional edge of the inflection point, across the foreground, to the other directional edge.

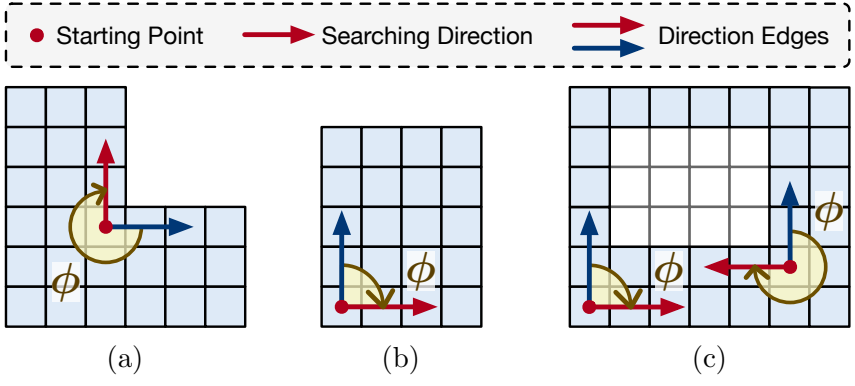


Illustration of the rotation ϕ of the foreground and the initial searching direction.

Parallel Searching

```
1:  function kernel(I, ∂I′, L, P)
2:      id ← blockIdx.x ∗ blockDim.x + threadIdx.x;
3:      label ← L[id];
4:      if id < N then
5:          Initial: p ← {P[id]}; visited ← {P[id]};
6:          cur ← P[id]; next_x, next_y ← −1, −1;
7:          direction ← init_direction(I, ∂I′, P[id], label);
8:          while next_x ≠ P[id]_x or next_y ≠ P[id]_y do
9:              for d ∈ counterclockwise directions do
10:                 new_x, new_y ← cur_x + d_x, cur_y + d_y;
11:                 if ∂I′[n_x][n_y] = label and new ∉ visited then
12:                     next_x, next_y ← new_x, new_y;
13:                     visited ← visited ∪ next;
14:                     direction′ ← d;
15:                     break;
16:             if direction′ ≠ direction then
17:                 p ← p ∪ (next_x, next_y);
18:                 direction ← direction′;
19:             cur_x, cur_y ← next_x, next_y;
20:         C ← C ∪ p;
```

- Each boundary component is assigned to a thread for contour tracing in parallel.
- The initial searching direction is defined using ϕ.
- Hole contours are traced in a clockwise direction, while outer contours are traced counterclockwise.

Experimental Results

G-Contour v.s. OpenCV

1. **Platform**: 1× NVIDIA A800 GPU
2. **Implementation**: C++/CUDA (four kernels)
3. **Benchmarks**: Optimized masks using FuILT[7]

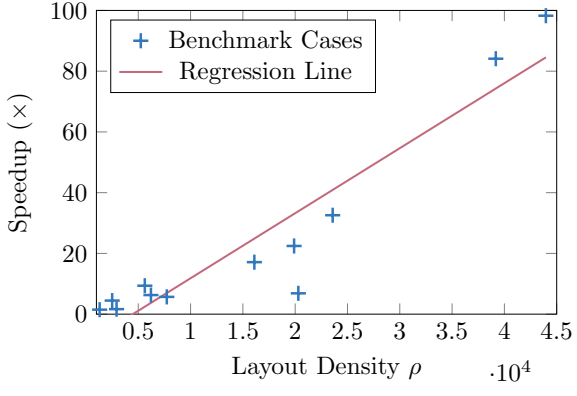
Benchmark	Layer	Layout Size (um^2)	#Pattern	Avg. Degree	Contour Tracing Runtime (s)		
					OpenCV	G-Contour	Speedup
gcd	Metal	35.70	9722	20.66	2.05	0.22	9.36×
	Via	35.53	8171	10.90	0.86	0.19	4.47×
	Poly	34.97	4385	23.33	0.91	0.55	1.65×
	Activate	35.59	2709	17.20	0.70	0.47	1.48×
ibex	Metal	249.45	89150	56.81	42.29	6.16	6.86×
	Via	255.33	394404	10.42	168.90	9.86	17.12×
	Poly	255.33	81776	24.13	12.08	2.12	5.69×
	Activate	254.72	277139	21.68	454.42	13.95	32.58×
riscv	Metal	176.29	663180	11.69	1566.19	15.94	98.26×
	Via	152.57	170602	5.56	11.32	1.80	6.29×
	Poly	147.11	670730	8.59	1193.35	14.19	84.11×
	Activate	147.25	433960	6.75	136.74	6.09	22.45×
Average					26.64	**2.46**	**10.83**×

Table: '#Pattern' refers to the number of patterns in the layout, and 'Avg. Degree' refers to the average number of vertices in the patterns.

[7] **Shuo Yin et al. (2024). "Fuilt: Full chip ilt system with boundary healing".** In: *Proceedings of the 2024 International Symposium on Physical Design*, pp. 13–20.

Pattern Complexity v.s. Runtim

$$\rho = \frac{\#\text{Pattern} \times \text{Avg. Degree}}{\text{Layout Size}}. \tag{3}$$



The relationship between the layout density ρ and the speedup of G-Contour across all benchmarks.

Conclusion

Summary

- We introduce G-Contour, the *first* GPU-accelerated framework addressing the contour tracing problem for large-scale EDA layouts.
- G-Contour incorporates a novel parallel component analysis algorithm and an efficient search-based method for contour extraction on GPUs.
- Experimental results show that G-Contour achieves significant speedups over the state-of-the-art CPU implementation (OpenCV), reaching up to 10× on average and 98× in peak cases.
- Beyond its primary application in EDA, G-Contour serves as a general-purpose tool applicable to broader tasks in image processing and computer graphics.

Q&A