# Curvilinear Optical Proximity Correction via Cardinal Spline


Su Zheng[1], Xiaoxiao Liang[2], Ziyang Yu[1], Yuzhe Ma[2], Bei Yu[1], Martin Wong[3]

[1]Chinese University of Hong Kong, [2]Hong Kong University of Science and Technology (GZ), [3]Hong Kong Baptist University


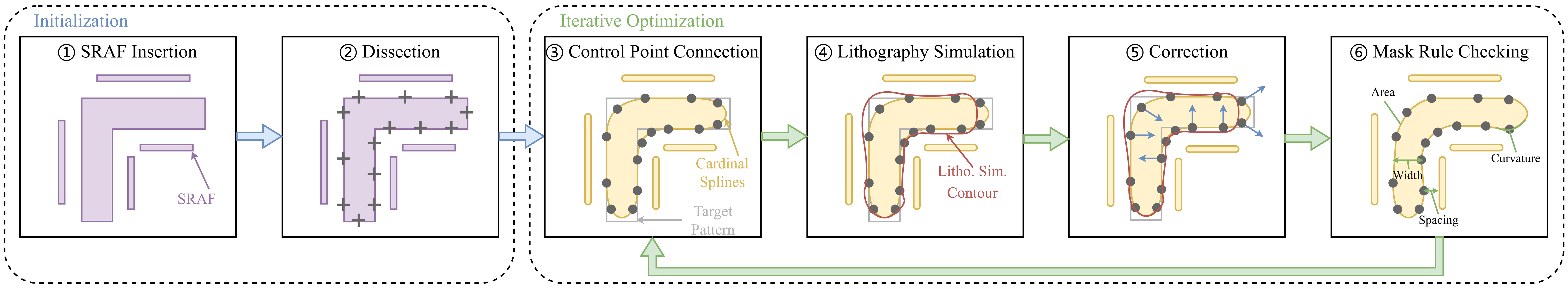


Figure 1. Overview of our CardOPC framework.

## Motivation

- **Curvilinear OPC enables flexible shapes + mask rule check**

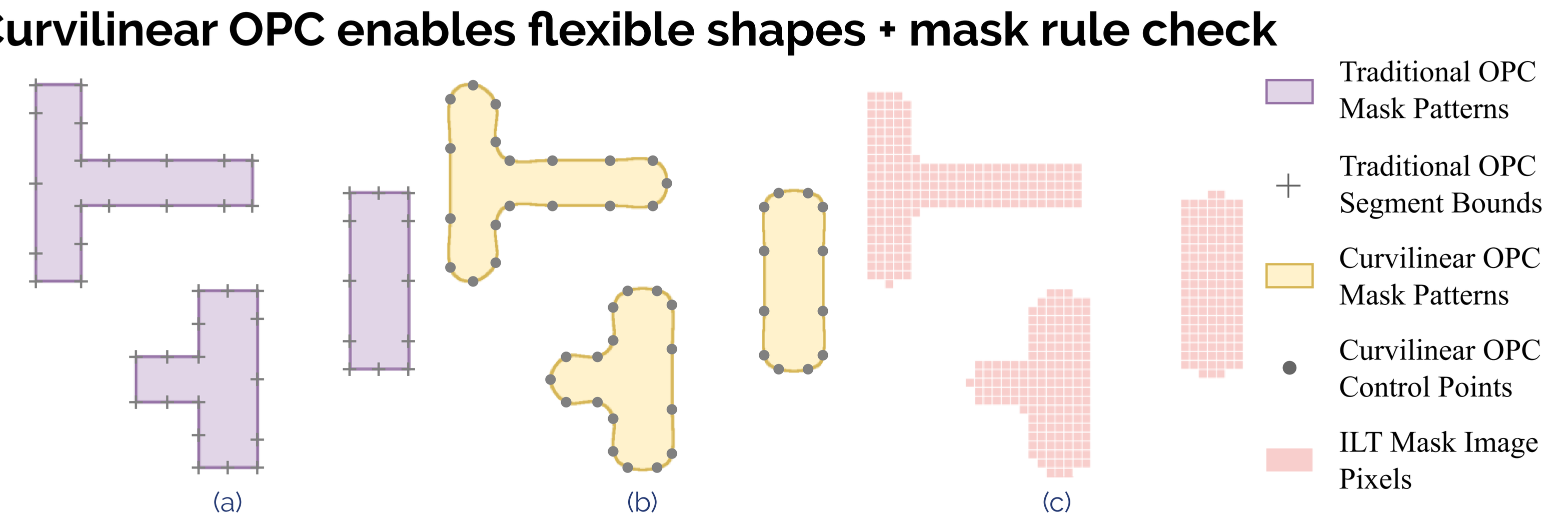


Figure 2. Illustration of the mask pattern representation in (a) traditional OPC; (b) curvilinear OPC; (c) ILT. Curvilinear OPC combines the reduced number of variables from traditional OPC and the flexibility of curvilinear shapes from ILT.

## Initialization Phase

### ① ② Initialization Phase: OPC Only

- SRAFs are placed at a certain distance from the main pattern
- Dissect the layout polygons into segments
- Generate the control points for cardinal splines

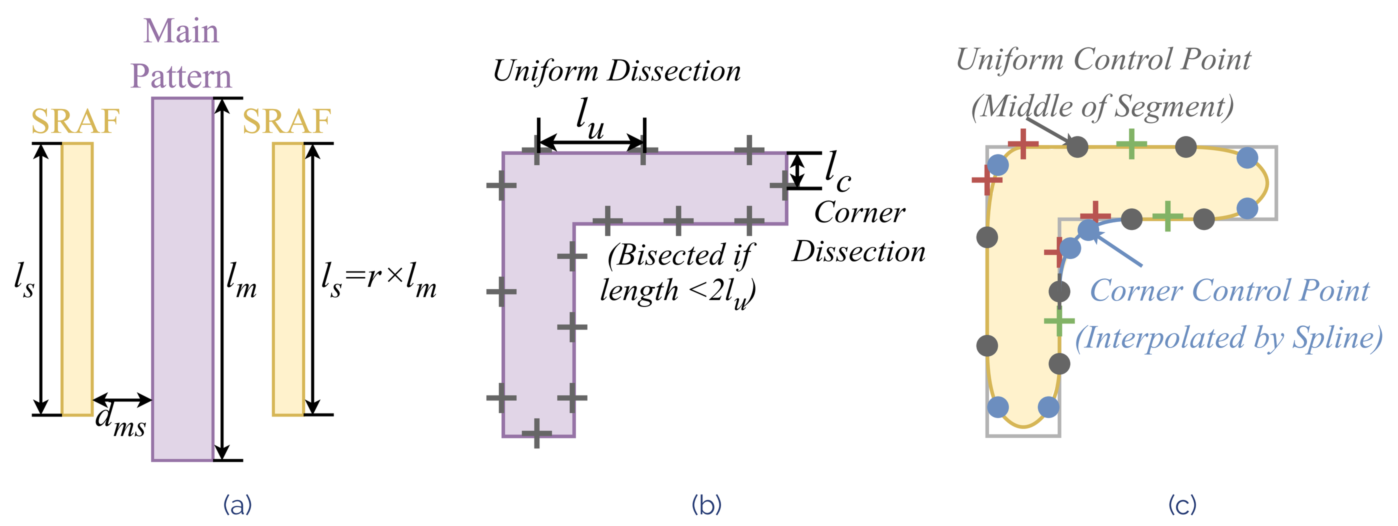


Figure 3. Illustration of the initialization phase of our curvilinear OPC flow.

### ① ② Initialization Phase: Initialized by ILT

**Algorithm 1** Fitting Method for ILT-Optimized Masks

**Input:** ILT-optimized mask image $M$, cardinal spline function $F(\cdot)$, ratio $r_Q$ & $r_R$, learning rate $\alpha$.

$Q = \emptyset$, $R = \emptyset$;
**for** each shape $S_i$ in the mask image $M$ **do**
  Extract the boundary points of $S_i$, denoted by $P_i$;
  Sample $r_Q|P_i|$ points from $P_i$ evenly, add them to $Q$;
  Sample $r_R|P_i|$ points from $P_i$ evenly, add them to $R$;
**for** $k \in \{1, 2, ......, K\}$ **do** //$K$ iterations in total;
  Interpolate $Q$ with $F(\cdot)$ to have $|R|$ points;
  Compute the loss $L(Q, R) = \|F(Q) - R\|^2$;
  Optimize $Q$ with $Q \leftarrow Q - \alpha\frac{\partial L(Q,R)}{\partial Q}$;
**return** $Q$;

### ③ Cardinal Spline v.s. Bézier spline

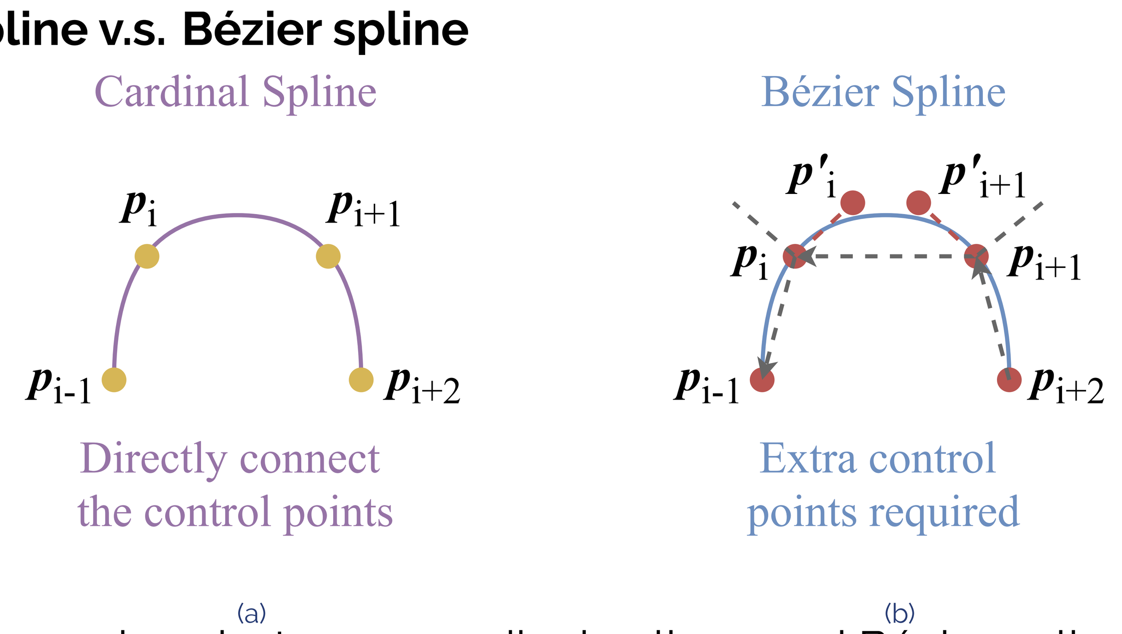


Figure 4. Comparison between cardinal splines and Bézier splines in curvilinear OPC. Using cardinal splines, the curve can pass through all the control points $p_{i-1}$, $p_i$, $p_{i+1}$, and $p_{i+2}$. However, when using Bézier splines, two extra control points $p'_i$ and $p'_{i+1}$ should be generated to ensure that the curve can pass through $p_i$ and $p_{i+1}$. This introduces non-negligible computational overhead.

## Optimization Phase

### ④ Lithography Simulation

Hopkins model—the aerial image $I$ is obtained by applying a set of optical kernels $H$ to the mask $M$ with:

$$I(x,y) = \sum_{k=1}^{N_h} w_k \left|M(x,y) \otimes h_k(x,y)\right|^2. \quad (1)$$

### ⑤ Correction

- Mathematical formulation:

$$f_{EPE}(M_{\tau+1}) \approx f_{EPE}(M_\tau) + f'_{EPE}(M_\tau) \times \Delta\text{Positions}_\tau. \quad (2)$$

- Move control points to minimize EPE:

$$\Delta\text{Positions}_\tau = -f'_{EPE}(M_\tau)^{-1} \times f_{EPE}(M_\tau). \quad (3)$$

- Smooth the moving distances:

$$\overline{\Delta d_i} = \sum_{k=-W}^{W} w_k \Delta d_k. \quad (4)$$

### ⑥ MRC Violation Resolving

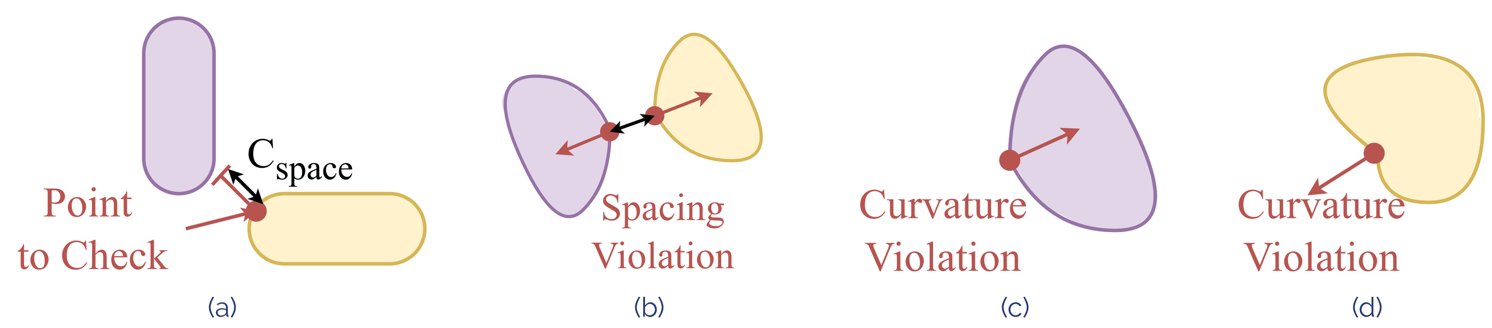


Figure 5. Illustration of mask rule checking and MRC violation resolving. (a) shows how to create a line segment to test the spacing rule violation. (b) moves the control points to resolve spacing rule violations. (c) and (d) resolve curvature rule violations by moving the control points in and out, respectively.

## Results

Table 1. Large-scale OPC result comparison on EPE violations and PVB ($\mu m^2$)

| Testcase | #Tiles ($30 \times 30\mu m^2$) | Calibre EPE | Calibre PVB | SimpleOPC EPE | SimpleOPC PVB | CardOPC EPE | CardOPC PVB |
|---|---|---|---|---|---|---|---|
| gcd | 1 | 3657 | 35.6651 | 3454 | 37.3002 | 3507 | 34.2606 |
| aes | 144 | 2722 | 27.7226 | 2571 | 29.4301 | 2578 | 27.3485 |
| dynamicnode | 144 | 2088 | 26.1663 | 1941 | 27.1213 | 1923 | 25.5011 |
| Average | - | 2409 | 26.9746 | 2260 | 28.3069 | **2255** | **26.4519** |
| Ratio | - | 100.0% | 100.0% | 93.8% | 104.9% | **93.6%** | **98.1%** |

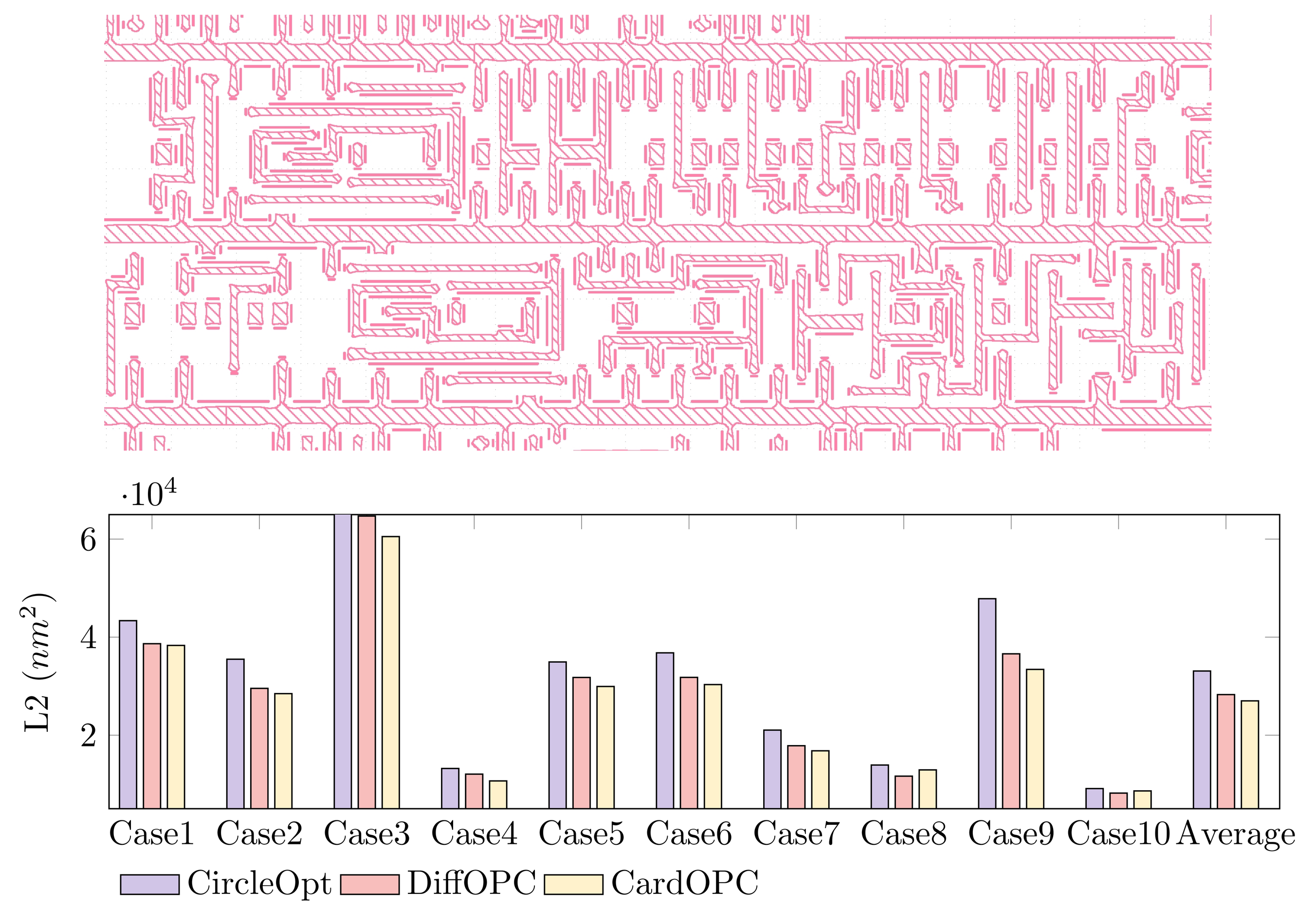


Figure 6. Comparison between our ILT-OPC hybrid approach and SOTA methods on ICCAD-13 benchmark. Additionally, our approach achieves 1.4 EPE violations on average, outperforming CircleOpt (3.9) and DiffOPC (2.2).