Real-time Corner and Polygon Detection System on FPGA

Chunmeng Bi and Tsutomu Maruyama
University of Tsukuba

Outline

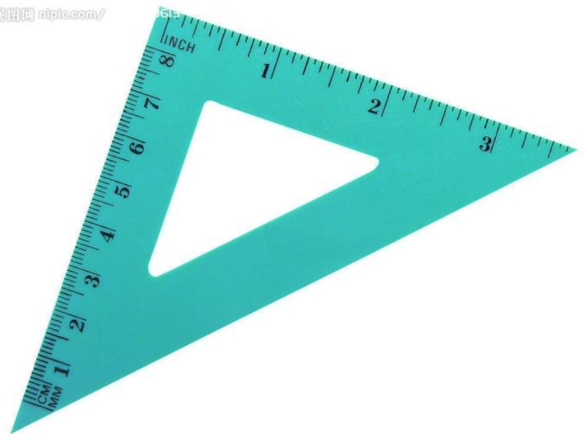
- Introduction
- Algorithms
- FPGA Implementation
- Experimental Results
- Conclusions and Future Work

Introduction

- Corner detection is a basic step frequently used in many image processing applications such as object recognition.
- Polygon detection can further help the recognition of the objects in the images.

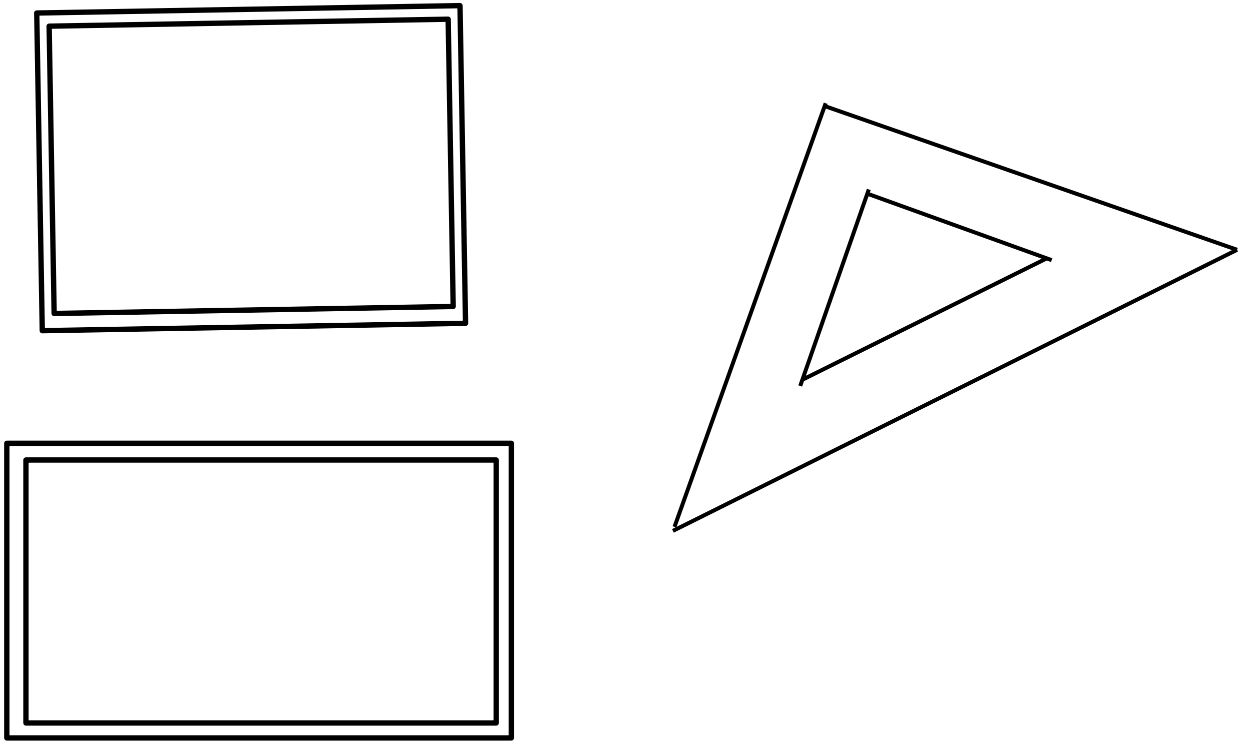
Introduction

- Polygon Detection



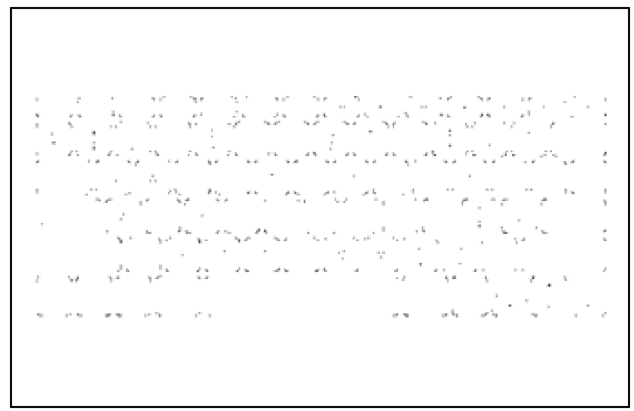
Introduction

- Polygon Detection



Algorithms of Corner Detection

- SUSAN: weak in the blurred image

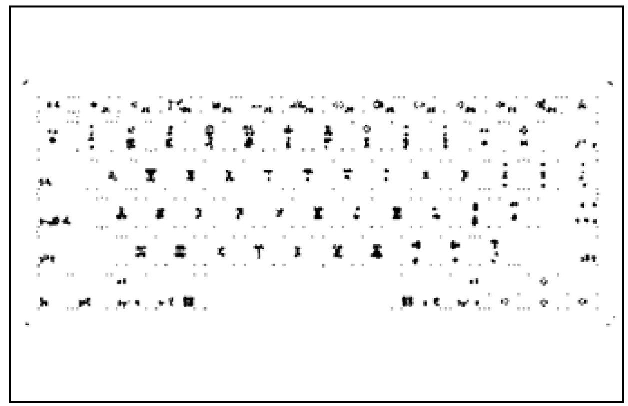


input image SUSAN

Algorithms of Corner Detection

- SUSAN: weak in the blurred image
- Harris: low localization rate

input image



Harris(13143)

Algorithms of Polygon Detection

- Hough Transform: requires significant computation and large memory, and it is difficult to achieve real-time processing.
- Shape Matching: uses the shape description for detecting polygons but it is difficult to define a descriptor which works well for all images.

Our Algorithm

edge detection

↓

edge thinning

↓

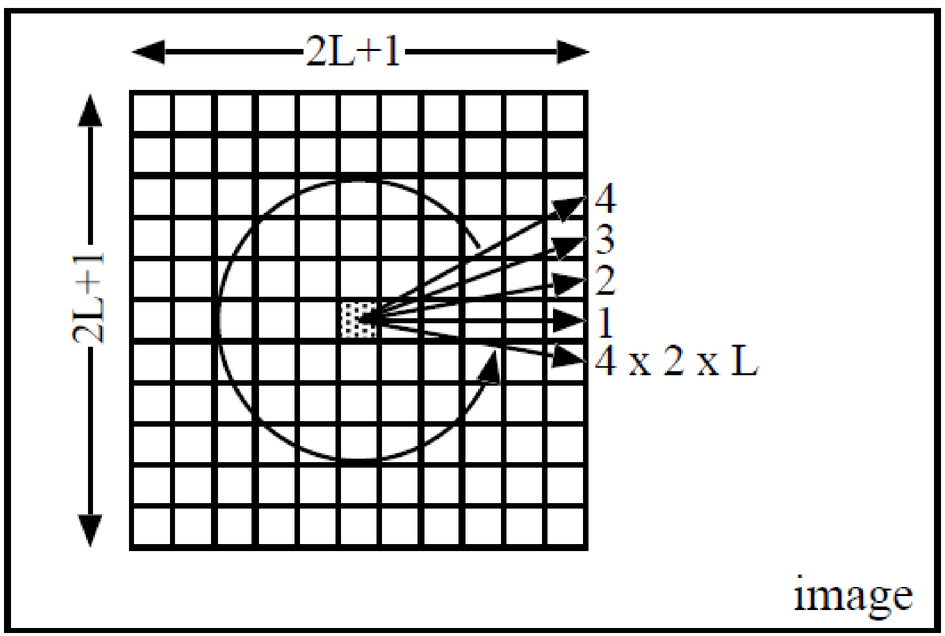
corner detection

↓

polygon detection

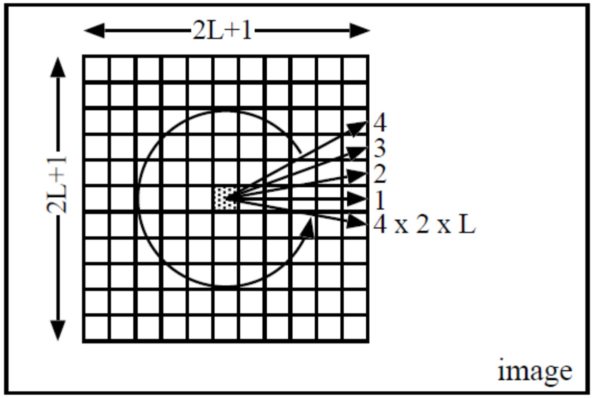
Our Corner Detection Algorithm

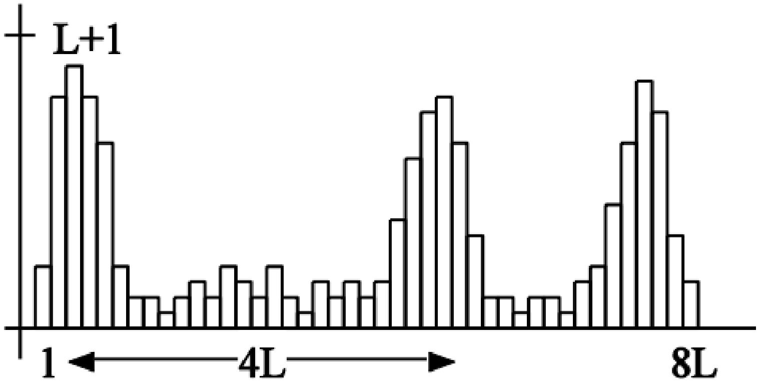
1. For all pixels in the image, a filter whose size is (2L+1) × (2L+1) is applied. In this filter, the number of the edges are counted up along the radial direction and 8L sums are calculated in total.



Our Corner Detection Algorithm

1. For all pixels in the image, a filter whose size is (2L+1) × (2L+1) is applied. In this filter, the number of the edges are counted up along the radial direction and 8L sums are calculated in total.
2. A histogram is obtained by calculating the 8L sums.

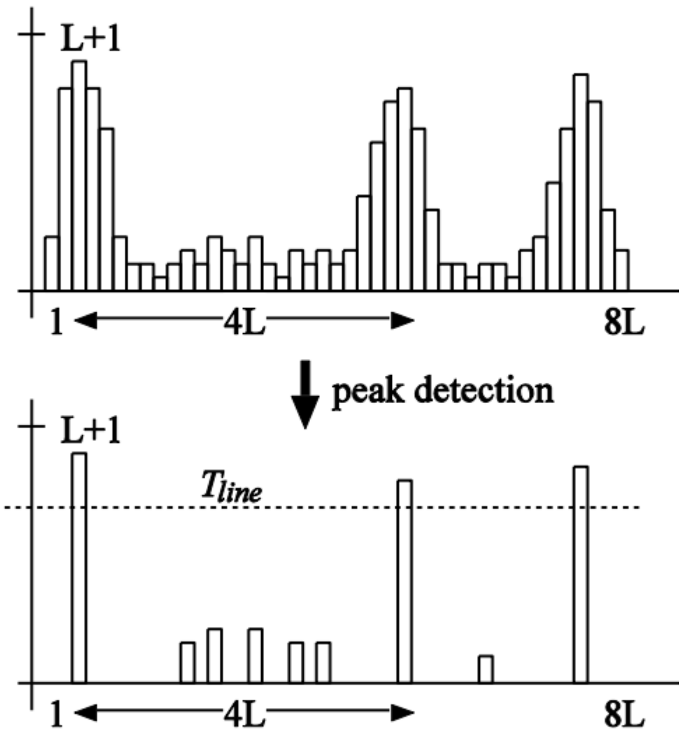




Our Corner Detection Algorithm

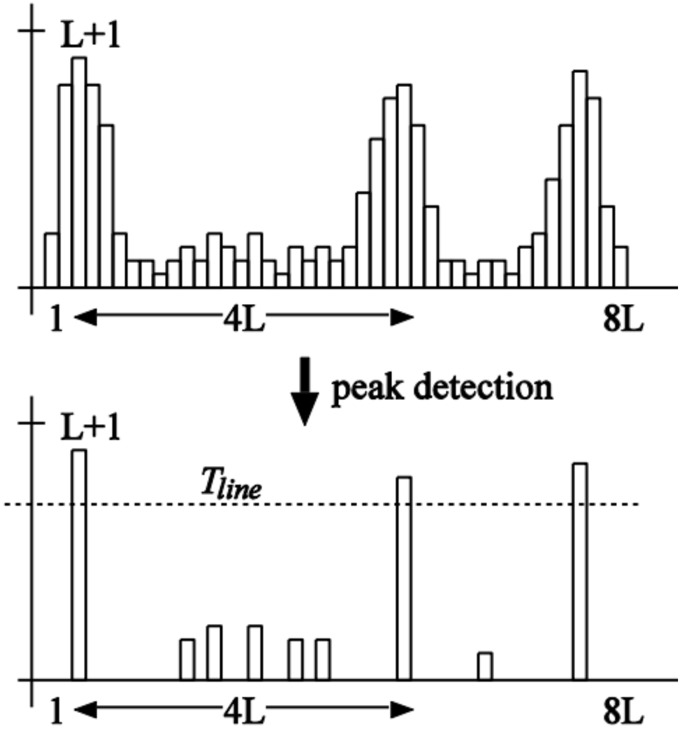
2. A histogram is obtained by calculating the 8L sums.

3. The peaks in the histogram are detected.



Our Corner Detection Algorithm

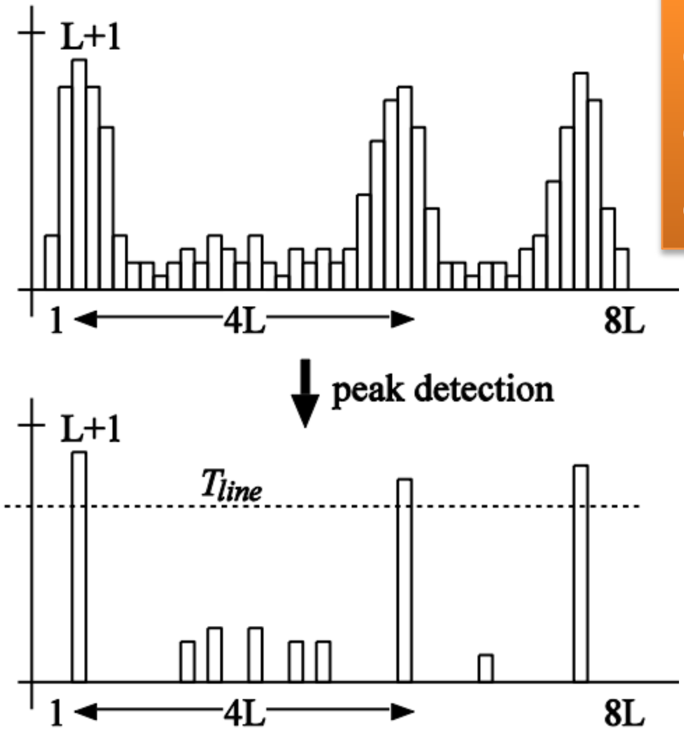
3. The peaks in the histogram are detected.
4. If the peak is larger than a given threshold T_{line}, it is considered that there exists a line segment on the radial direction.

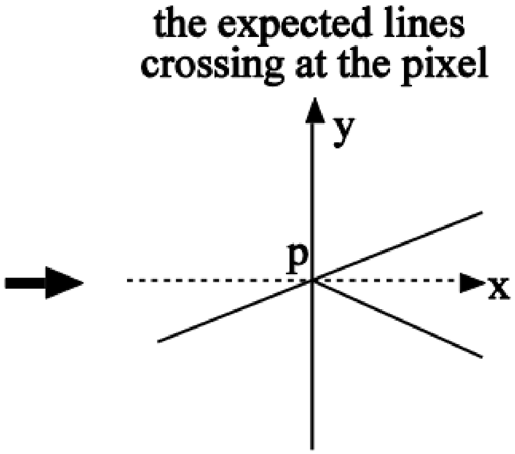


Our Corner Detection Algorithm

3. The peaks in the histogram are detected.
4. If the peak is larger than a given threshold T_{line}, it is considered that there exists a line segment on the radial direction.

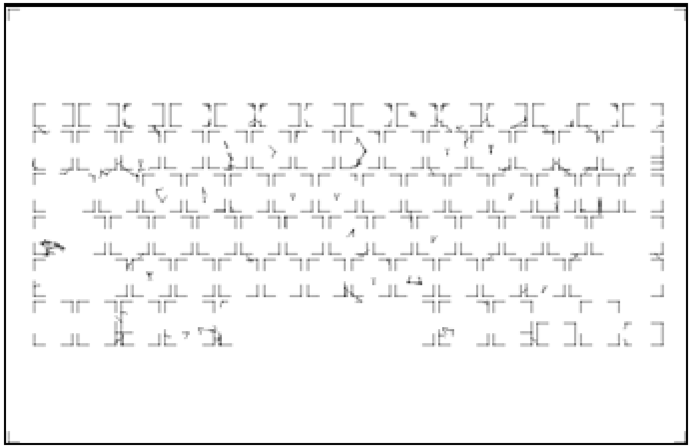
From the histogram for a pixel p, it is expected that two line segments (one of them passes through p and another one stops at p) are crossing at p.





Our Corner Detection Algorithm

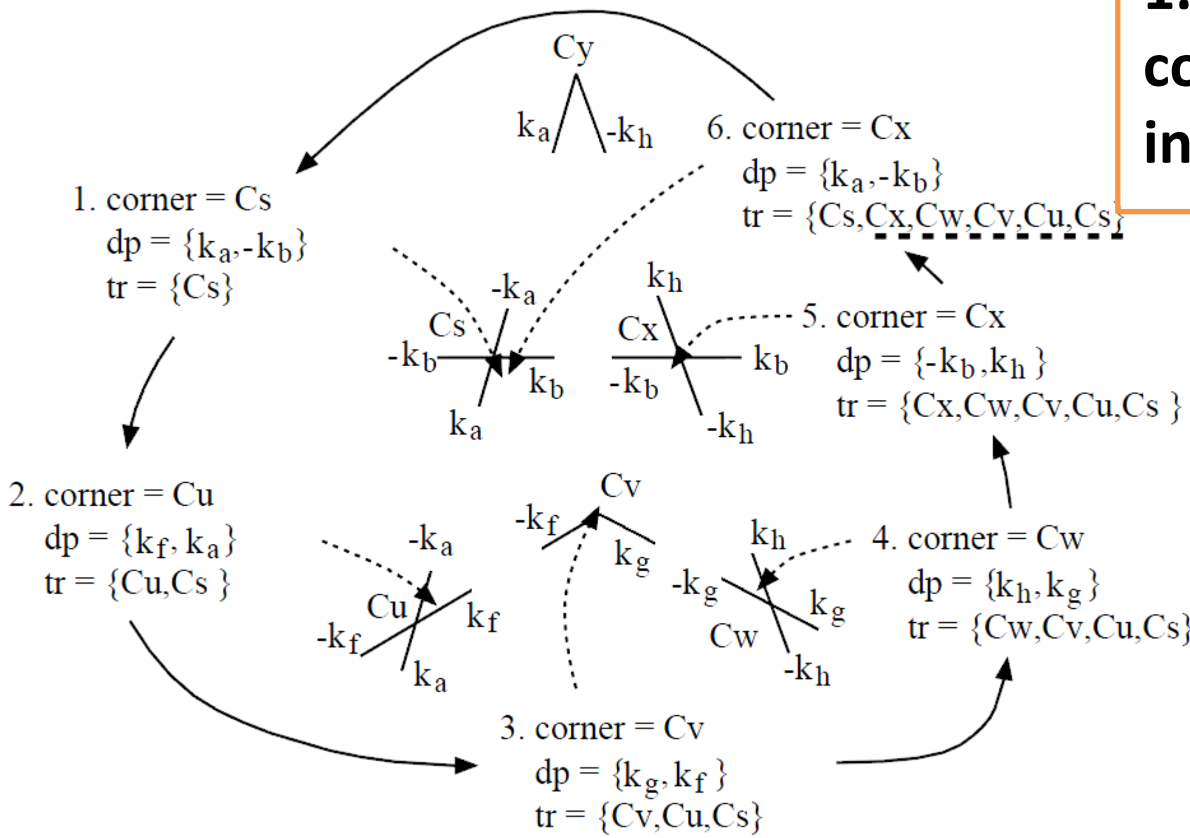
input image



our algorithm(473)

Our approach detects fewer corners and can tell the exact position of the corners as well as the gradients of all the line segments crossing at the corners.

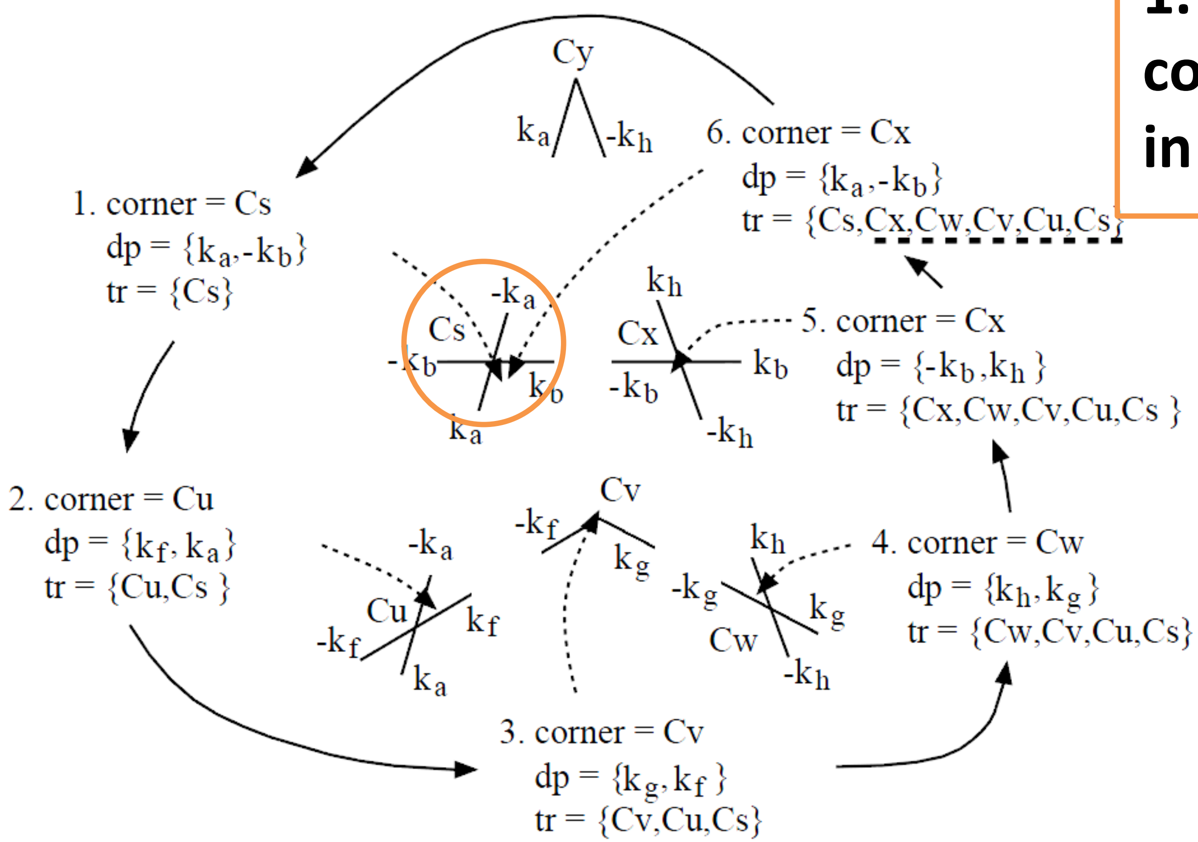
Our Polygon Detection Algorithm



1. Let S_c be the set of the corners and a corner(C_s) in S_c is chosen.

S_c : C_s, C_u, C_v, C_w, C_x, C_y

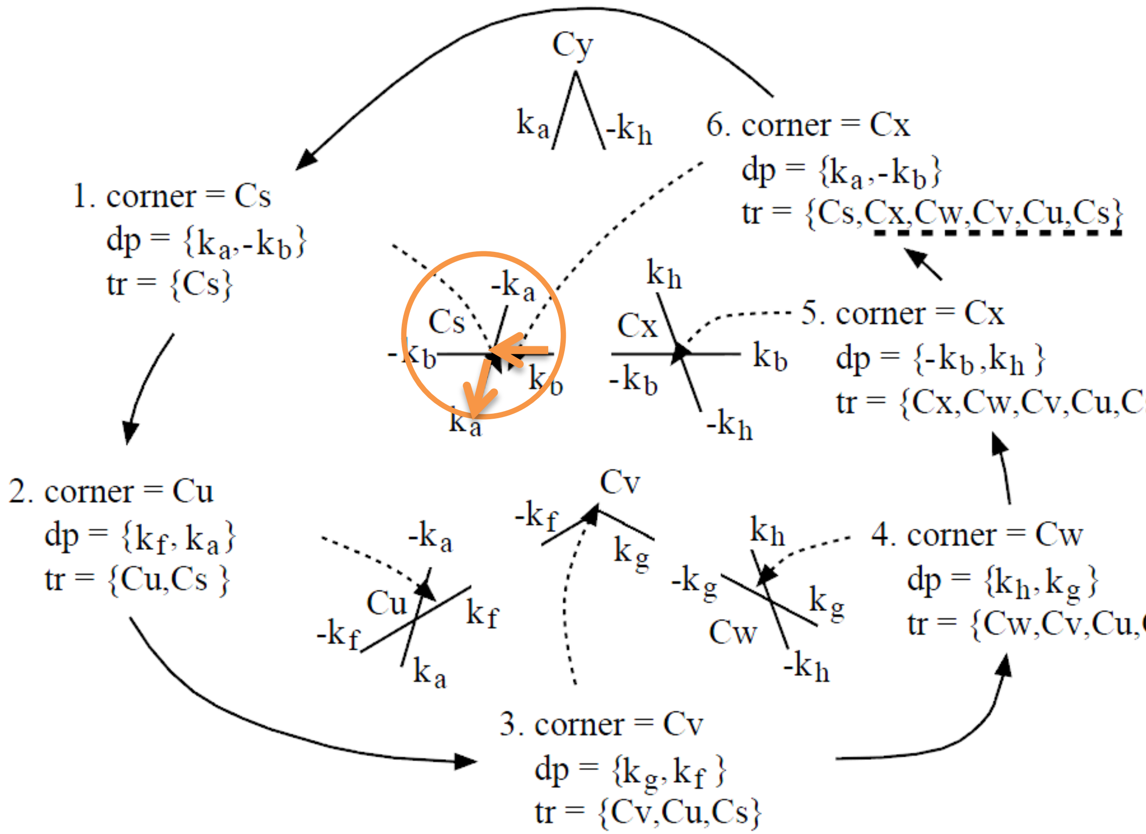
Our Polygon Detection Algorithm



1. Let S_c be the set of the corners and a corner(C_s) in S_c is chosen.

S_c : C_s, C_u, C_v, C_w, C_x, C_y

Our Polygon Detection Algorithm



1. Let S_c be the set of the corners and a corner(C_s) in S_c is chosen.

2. Find the corners which can be matched with (k_a, k_b).

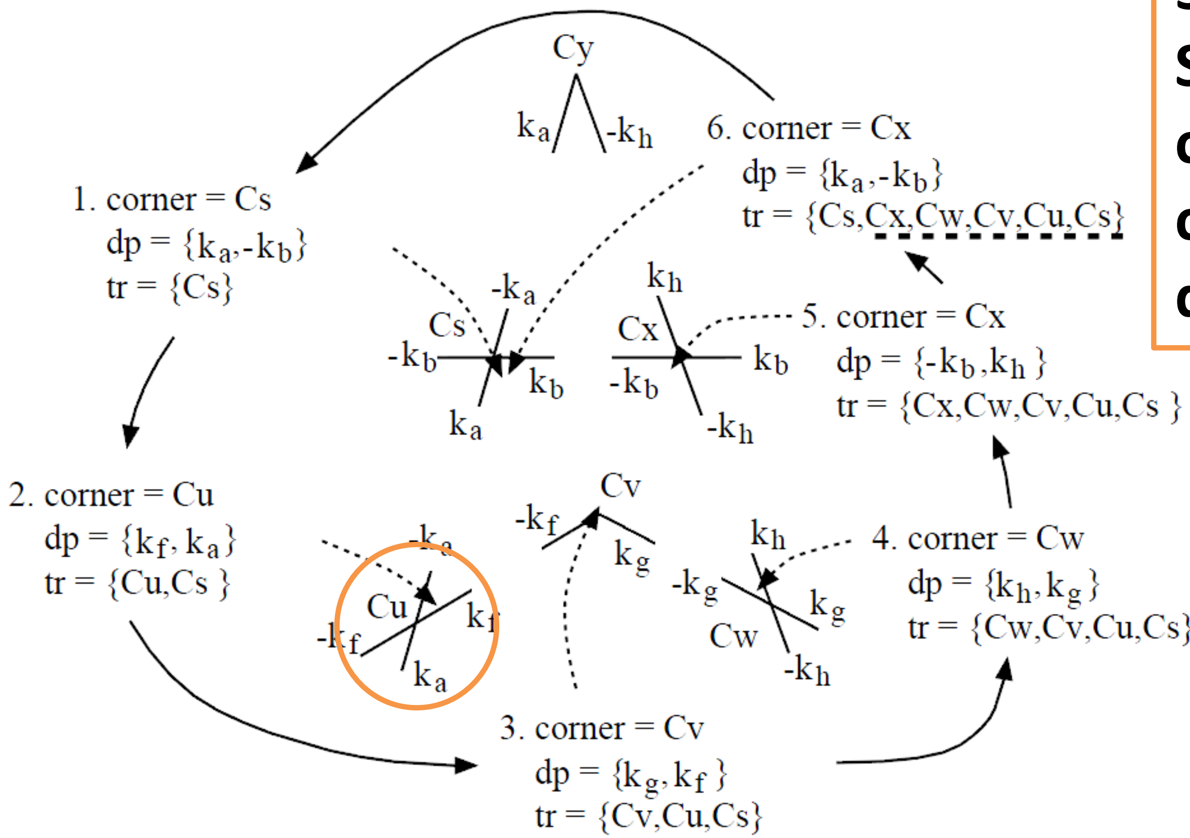
C_s: k_a, k_b, $-k_a$, $-k_b$

d_p (direction pair) : {k_a, $-k_b$}

t_r (trace) :{C_s}

S_c : C_s, C_u, C_v, C_w, C_x, C_y

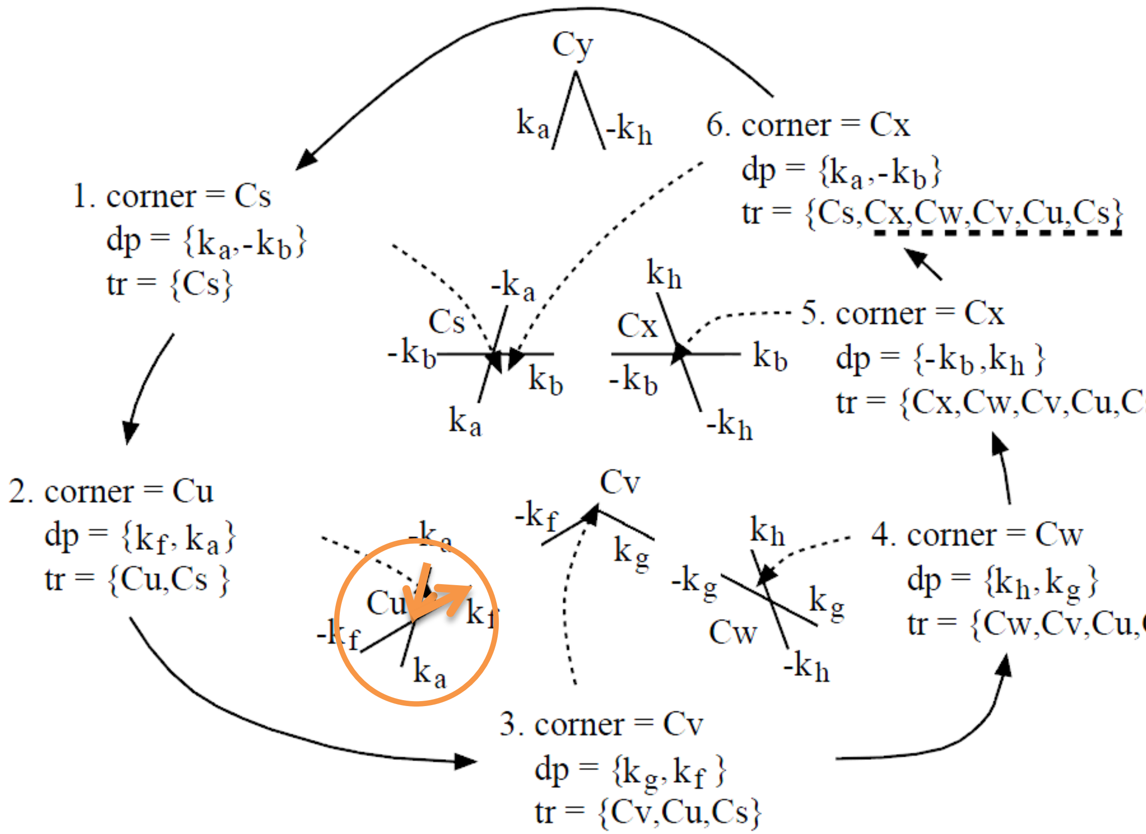
Our Polygon Detection Algorithm



3.Scan the corners in S_c and choose the closest connectable corner (C_u) to C_s in distance from the list.

S_c : C_s, C_u, C_v, C_w, C_x, C_y

Our Polygon Detection Algorithm



$S_c : C_s, C_u, C_v, C_w, C_x, C_y$

3.Scan the corners in S_c and choose the closest connectable corner (C_u) to C_s in distance from the list.

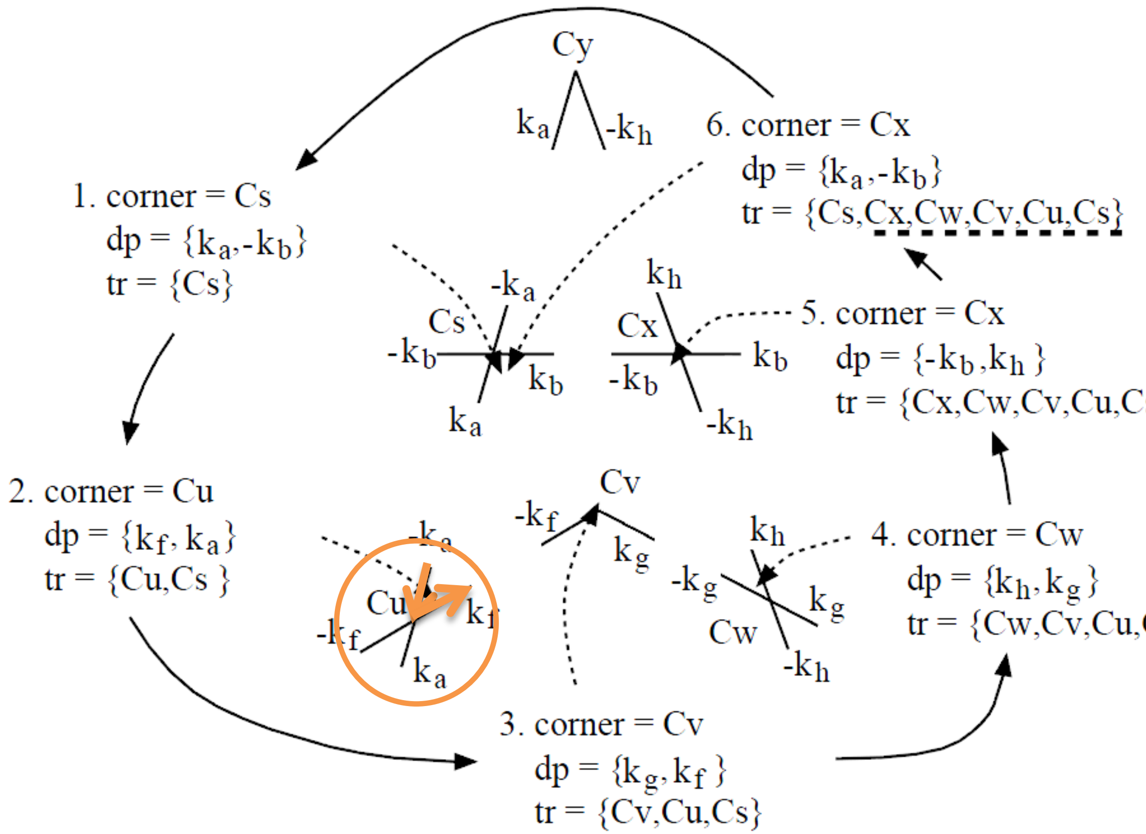
4. C_u has four directions but k_f is chosen, because it is the closest to $-k_a$ and k_b.

C_u : k_a, k_f, $-k_a$, $-k_f$

d_p :{k_f, k_a}

t_r :{C_u, C_s}

Our Polygon Detection Algorithm



3.Scan the corners in S_c and choose the closest connectable corner (C_u) to S_c in distance from the list.

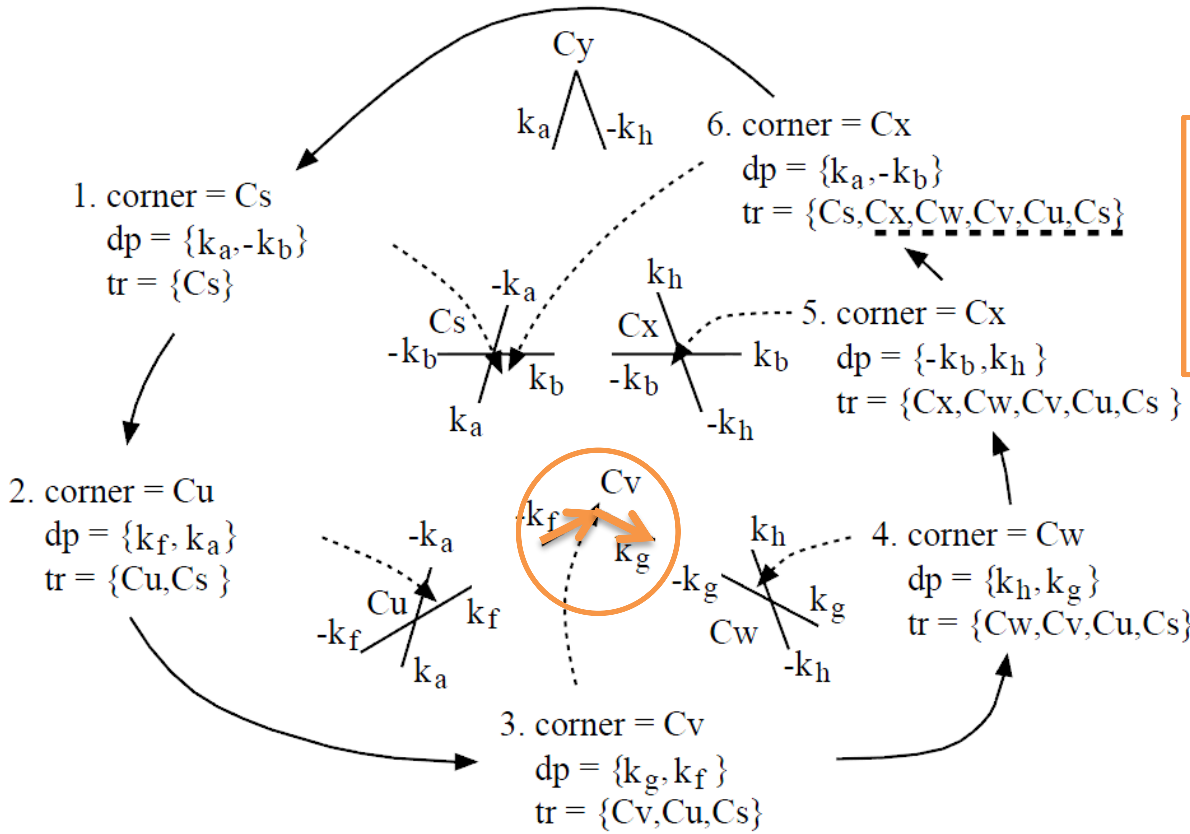
4. C_u has four directions but k_f is chosen, because it is the closest to - k_a and k_b.

C_u : k_a, k_f, $-k_a$, $-k_f$
d_p :{k_f, k_a}
t_r :{C_u, C_s}

5. This tracking is continued for this new d_p from step 3.

S_c : C_s, C_u, C_v, C_w, C_x, C_y

Our Polygon Detection Algorithm



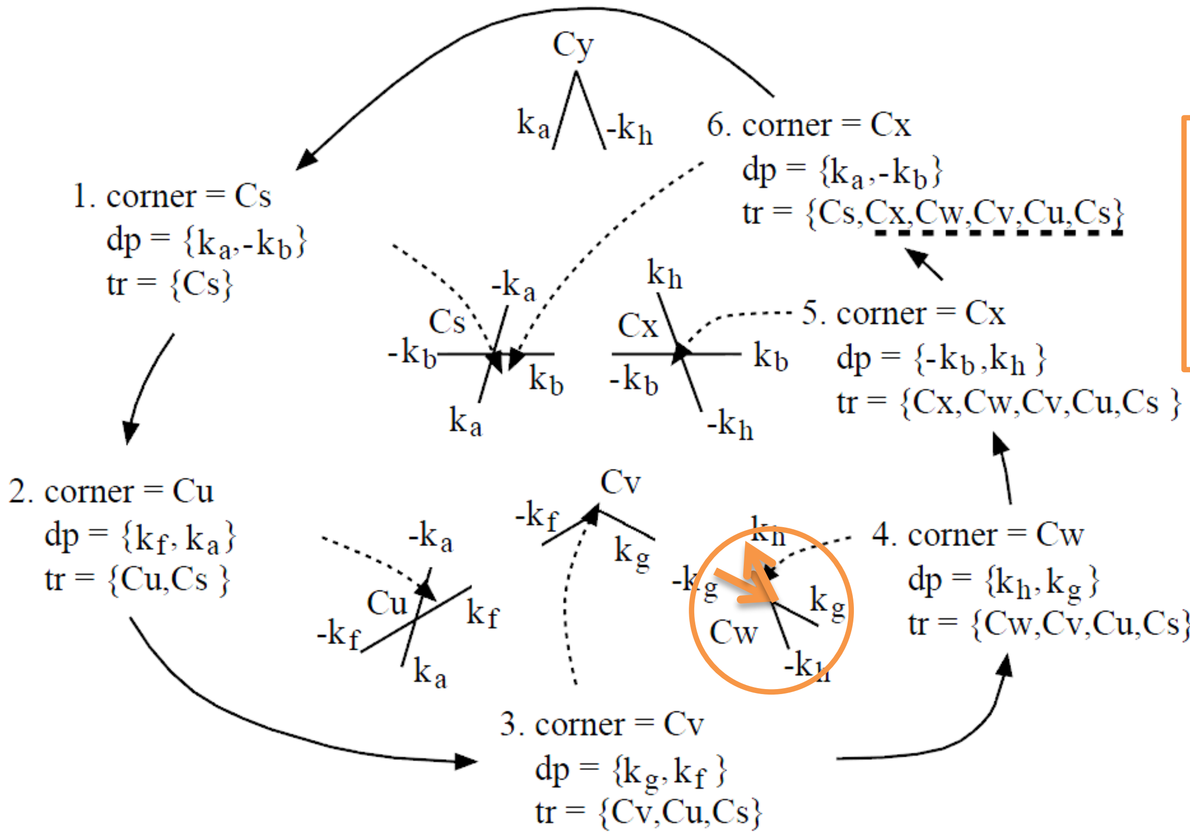
C_v :$-k_f$, k_g

d_p :{k_g, k_f}

t_r :{C_v, C_u, C_s}

S_c : C_s, C_u, C_v, C_w, C_x, C_y

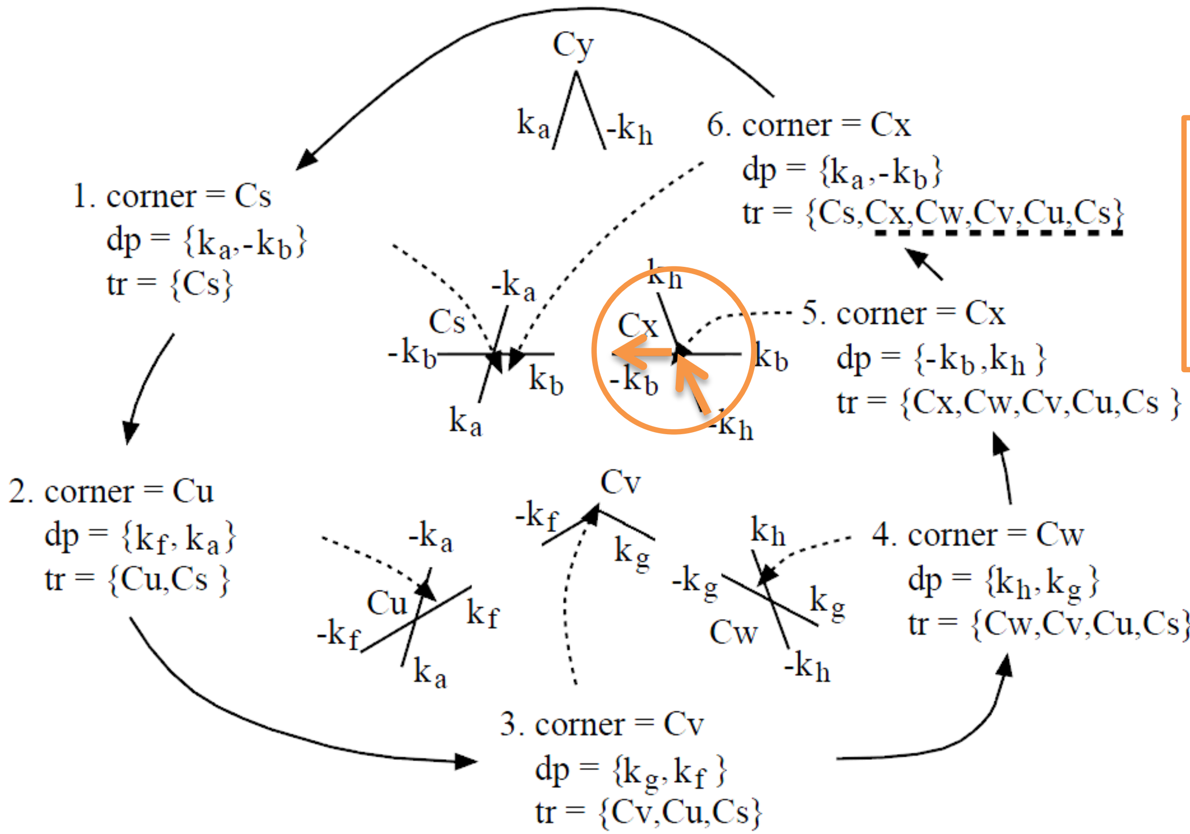
Our Polygon Detection Algorithm



C_w: k_g, k_h, $-k_g$, $-k_h$
d_p :{k_h, k_g}
t_r :{C_w, C_v, C_u, C_s}

S_c : C_s, C_u, C_v, C_w, C_x, C_y

Our Polygon Detection Algorithm



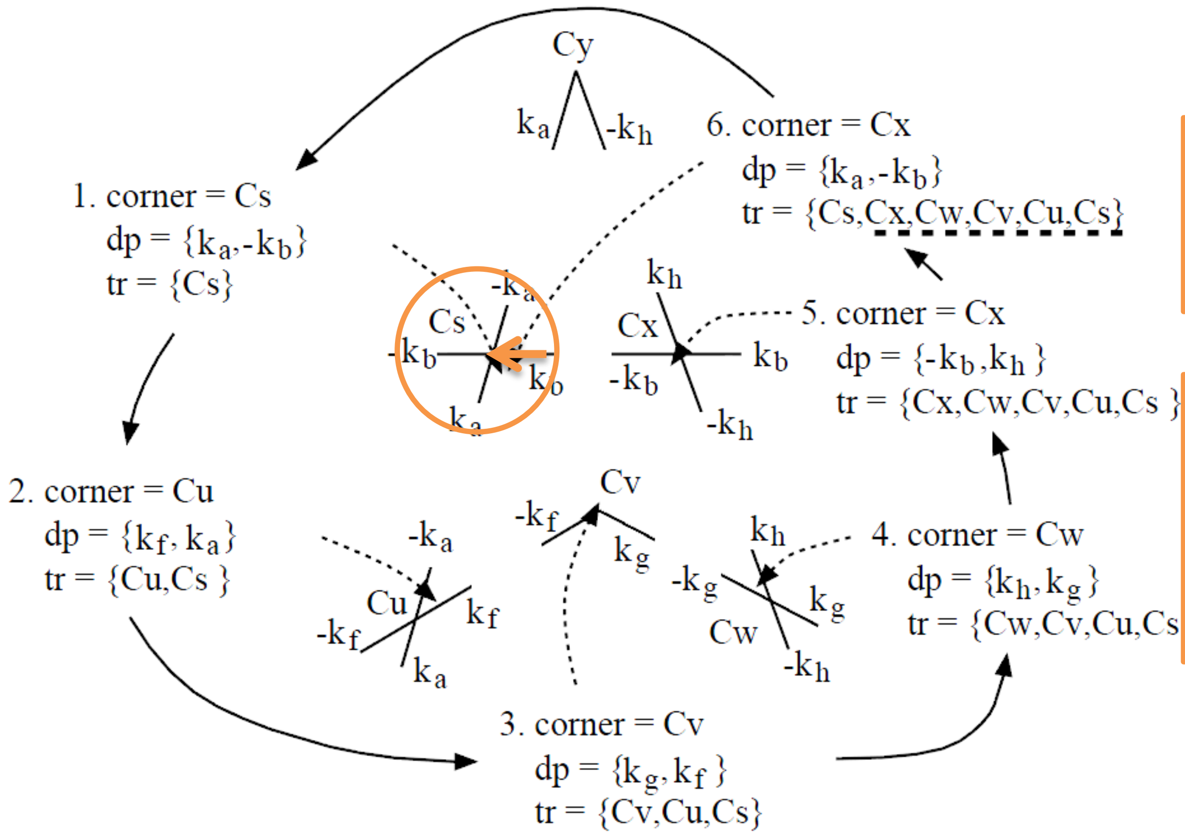
C_x : k_b, k_h, $-k_b$, $-k_h$

d_p :{$-k_b$, k_h}

t_r :{C_x, C_w, C_v, C_u, C_s}

S_c : C_s, C_u, C_v, C_w, C_x, C_y

Our Polygon Detection Algorithm

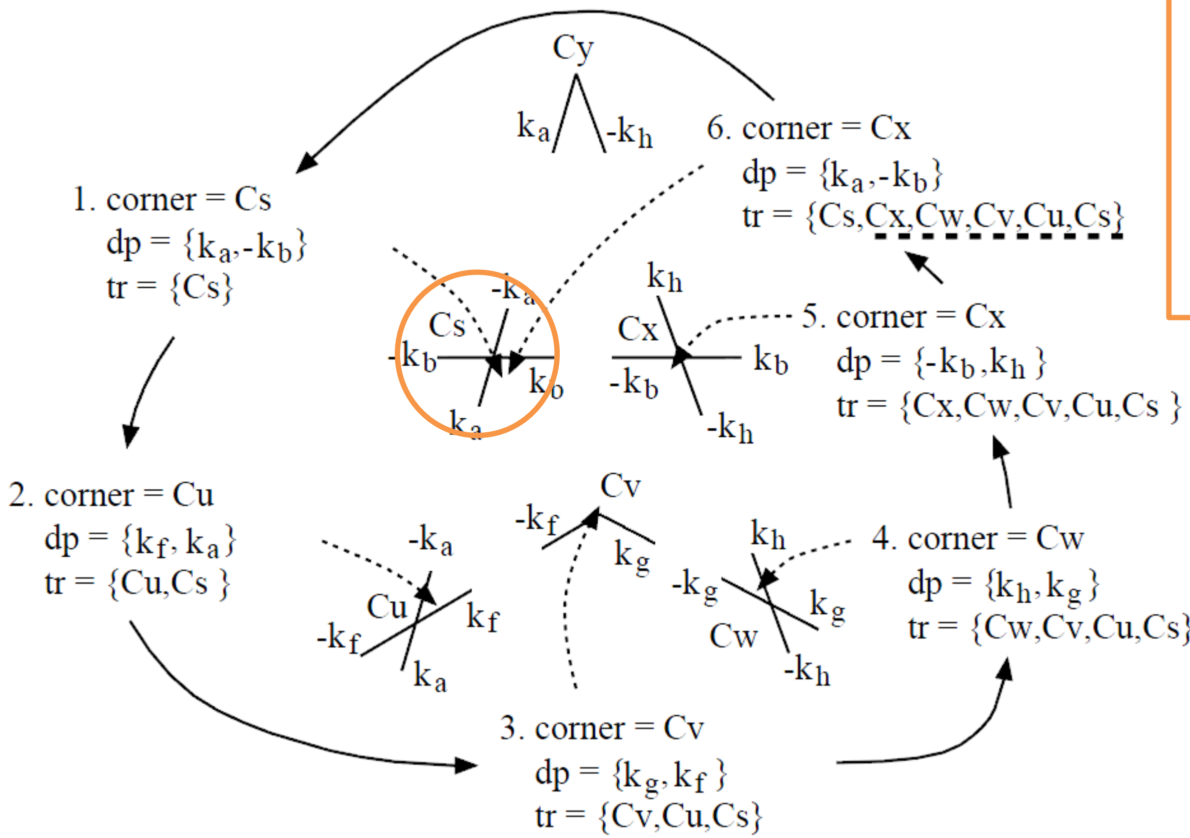


C_s : k_a, k_b, $-k_a$, $-k_b$
d_p :{k_a, $-k_b$}
t_r :{C_s, C_x, C_w, C_v, C_u, C_s}

5.Finally, C_s is chosen again. Then, a polygon is detected and the corners in t_r fix the shape of the polygon.

S_c : C_s, C_u, C_v, C_w, C_x, C_y

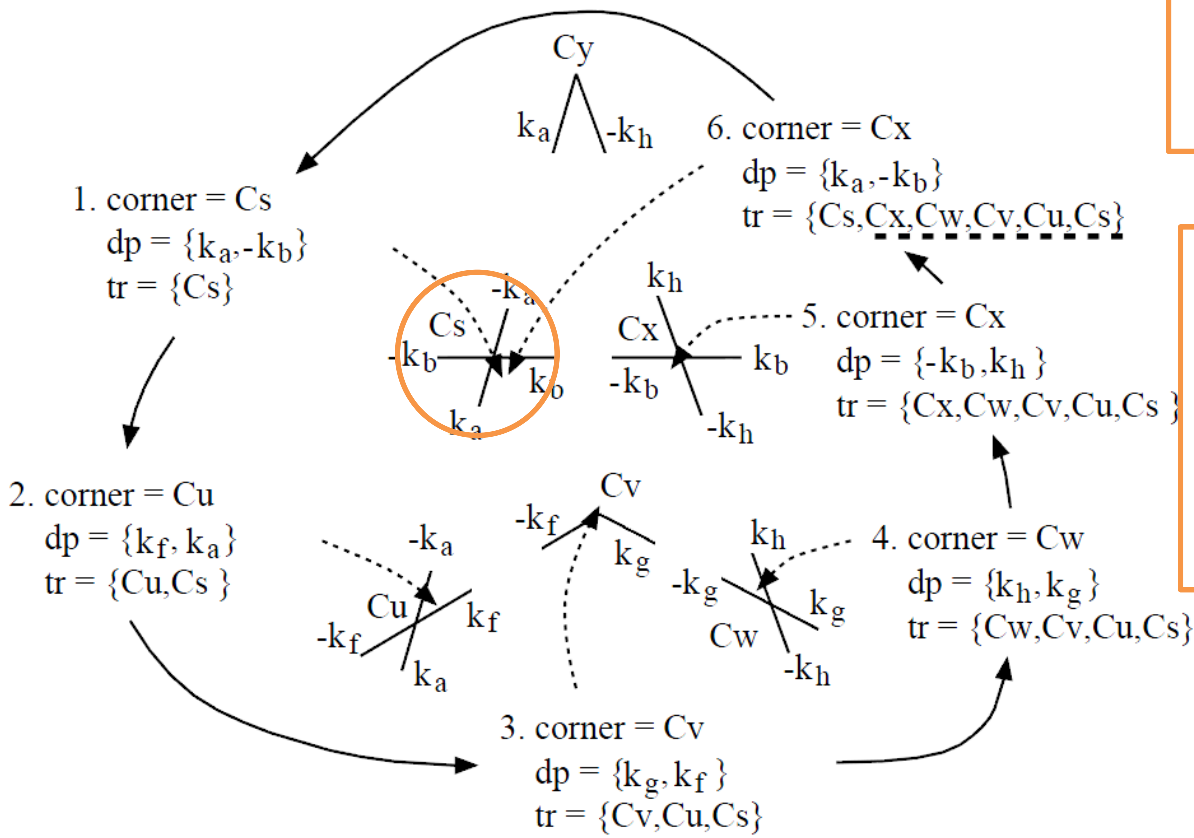
Our Polygon Detection Algorithm



6. Repeat the same procedure for other angles of C_s(eg.(k_a, $-k_b$) ($-k_a$, $-k_b$)).

S_c : C_s, C_u, C_v, C_w, C_x, C_y

Our Polygon Detection Algorithm

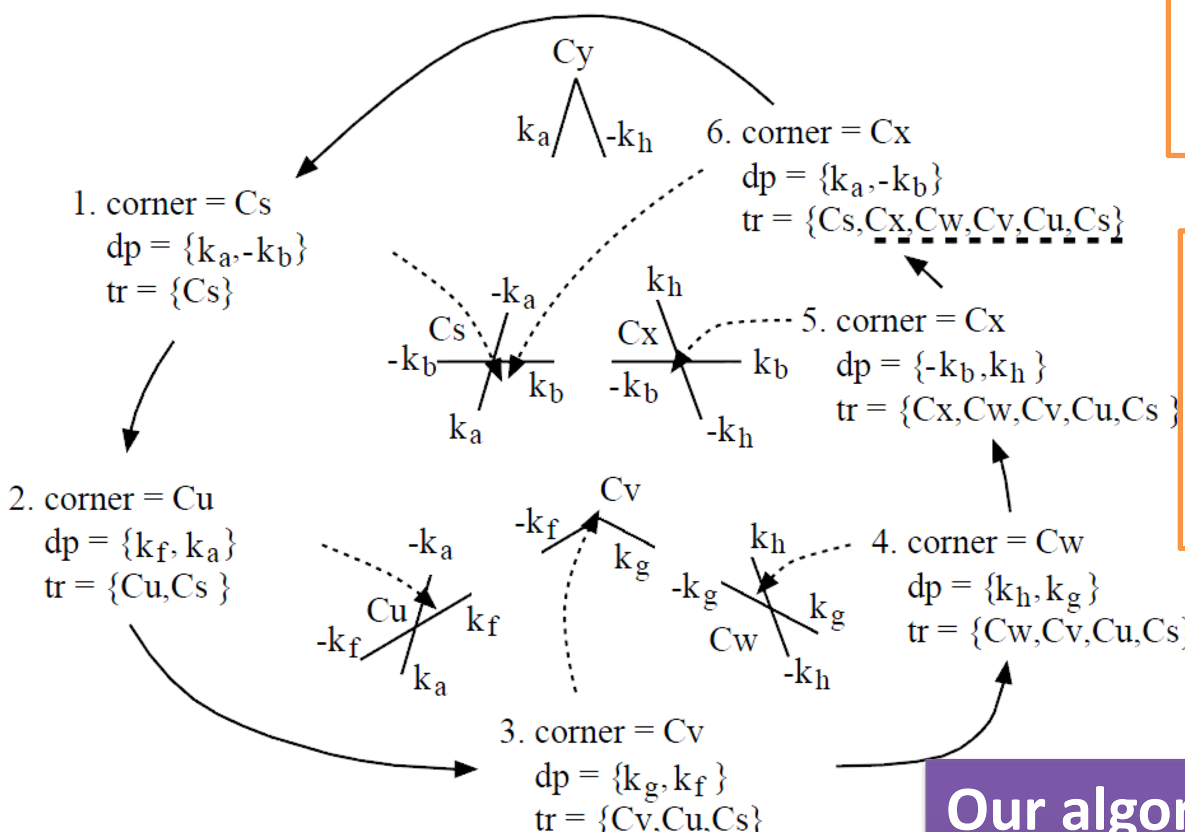


6. Repeat the same procedure for other angles of C_s(eg.(k_a, -k_b) (-k_a, -k_b)).

7. C_s is deleted from S_c to prevent multiple detection of the same polygon.

S_c : C_u, C_v, C_w, C_x, C_y

Our Polygon Detection Algorithm



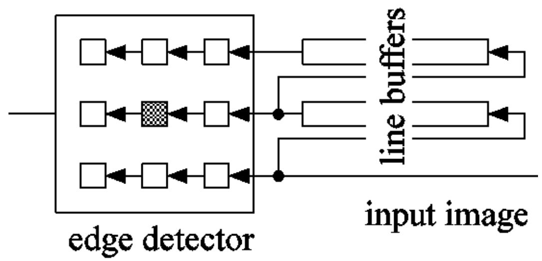
6. Repeat the same procedure for other angles of C_s(eg.(k_a, $-k_b$) ($-k_a$, $-k_b$)).

7. C_s is deleted from S_c to prevent multiple detection of the same polygon.

Our algorithm can detect polygons of any shapes and any number of corners.

FPGA Implementation

Edge Detection(Prewitt filter)



Horizontal direction

$$hx = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Vertical direction

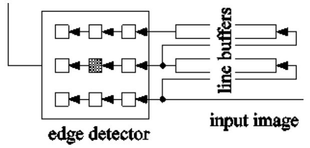
$$hy = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Horizontal& Vertical

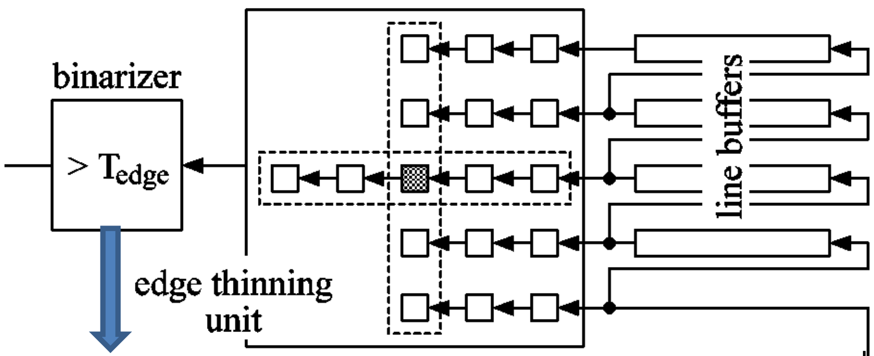
$$hxy = (hx^2 + hy^2)^{\frac{1}{2}}$$

FPGA Implementation

edge detection



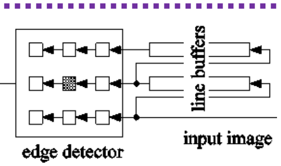
FPGA Implementation



the edges are binarized:

$$e_b(x,y) = \begin{cases} 1 & \text{if } e(x,y) \geq T_{edge} \text{ (a given threshold)} \\ 0 & \text{otherwise} \end{cases}$$

edge detection

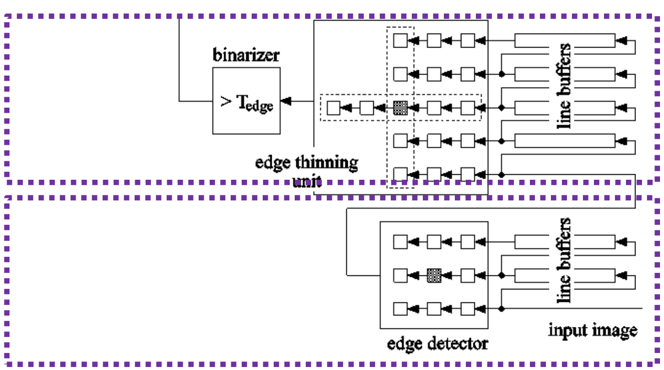


Edge thinning : Each edge is compared with its left, right, up and down pixels and if the edge is not the peak along the x nor y axis, its value is masked to 0.

FPGA Implementation

edge thinning

edge detection

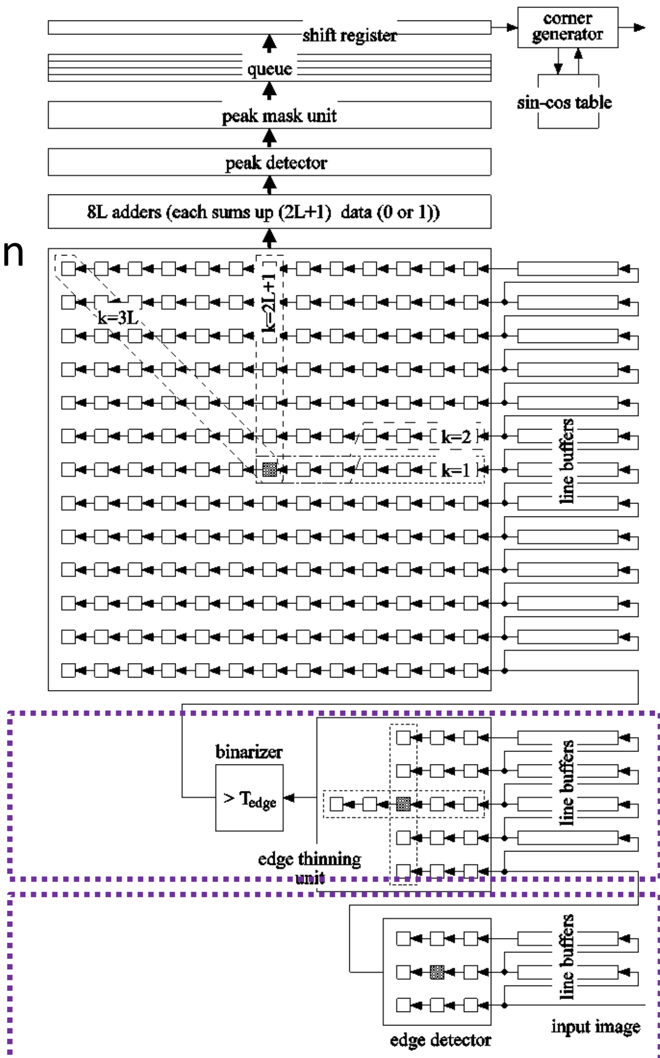


FPGA Implementation

corner detection

edge thinning

edge detection

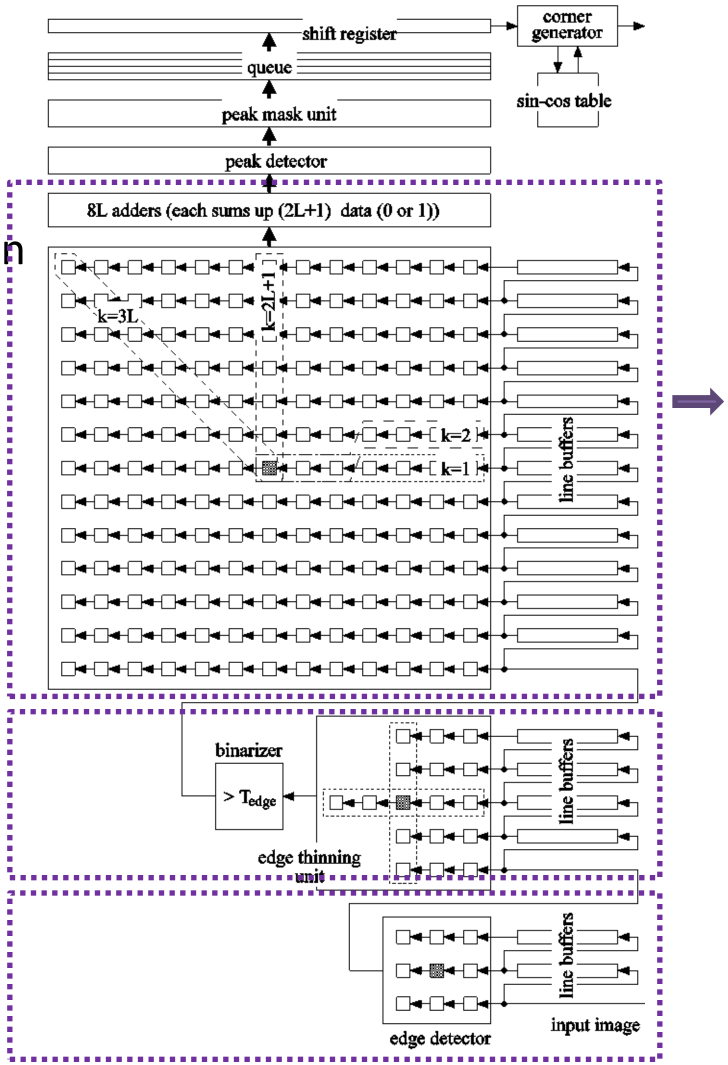


FPGA Implementation

corner detection

edge thinning

edge detection



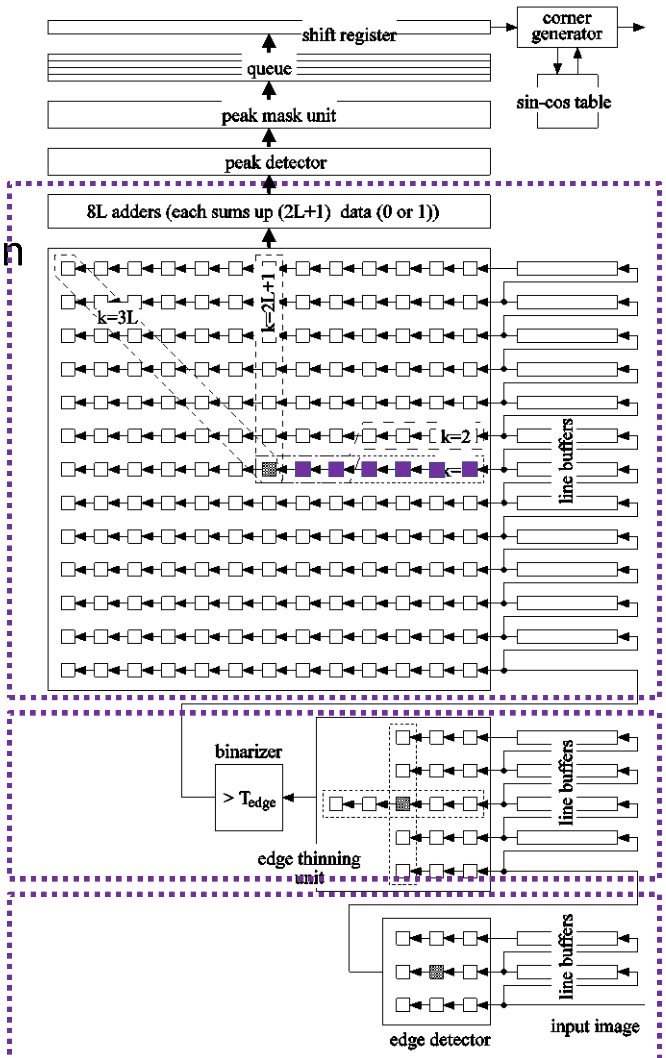
The binarized edges are held on a register array whose size is (2L+1) × (2L+1) and summed up along the radial lines which are drawn from the center to the 8L pixels on the edges.

FPGA Implementation

corner detection

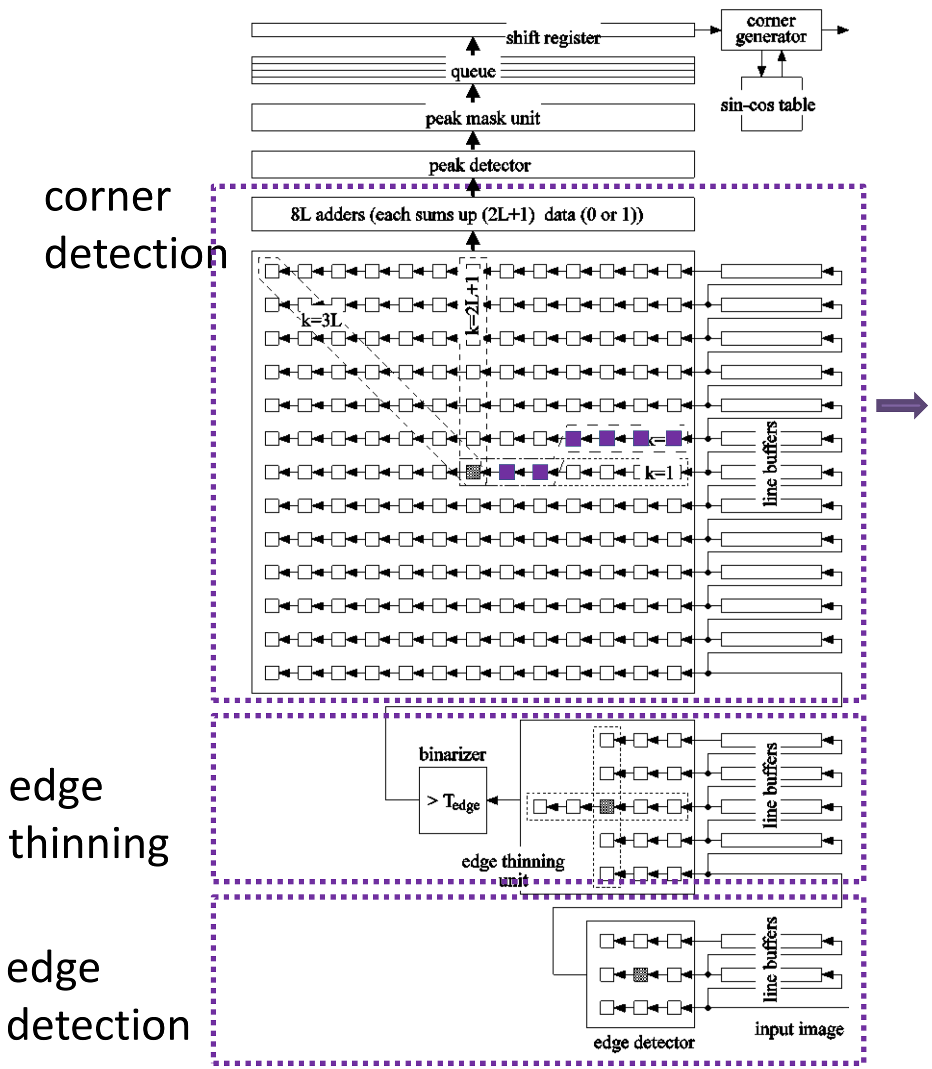
edge thinning

edge detection



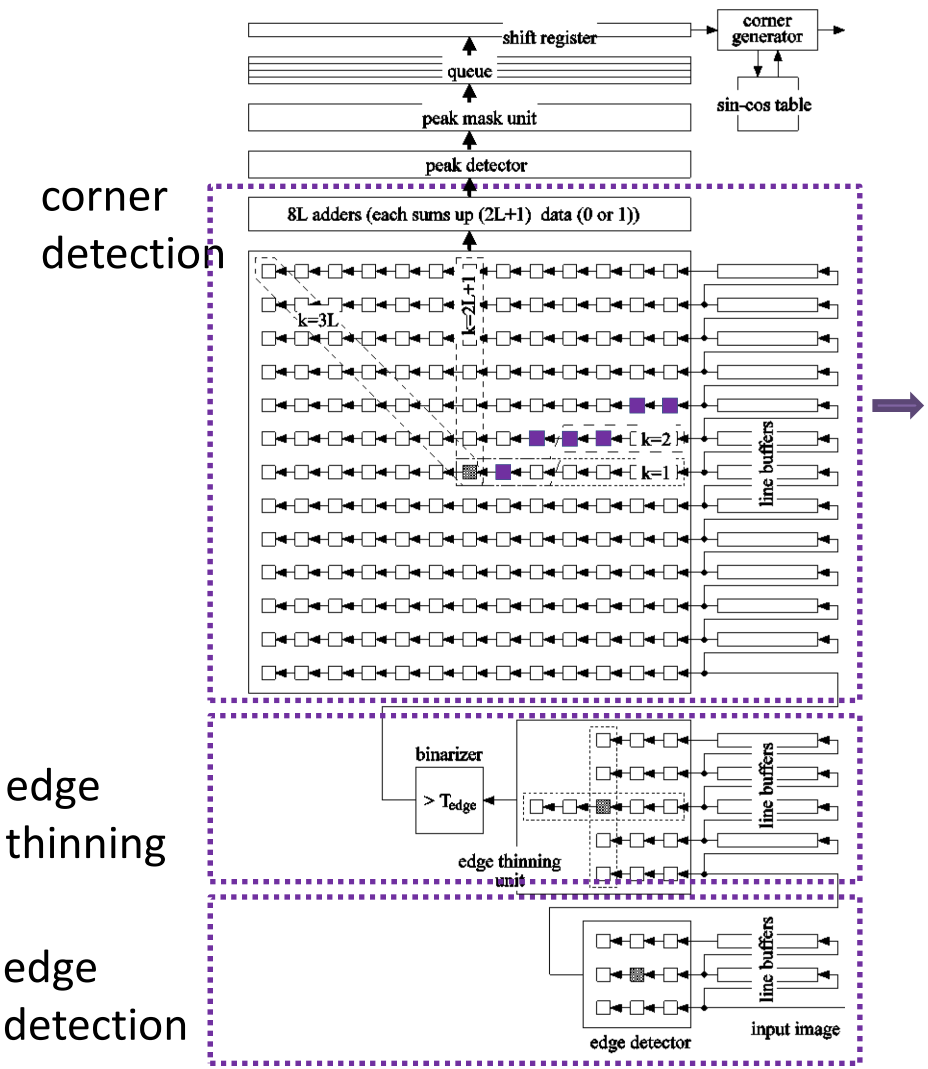
The binarized edges are held on a register array whose size is (2L+1) × (2L+1) and summed up along the radial lines which are drawn from the center to the 8L pixels on the edges.

FPGA Implementation



The binarized edges are held on a register array whose size is (2L+1) × (2L+1) and summed up along the radial lines which are drawn from the center to the 8L pixels on the edges.

FPGA Implementation



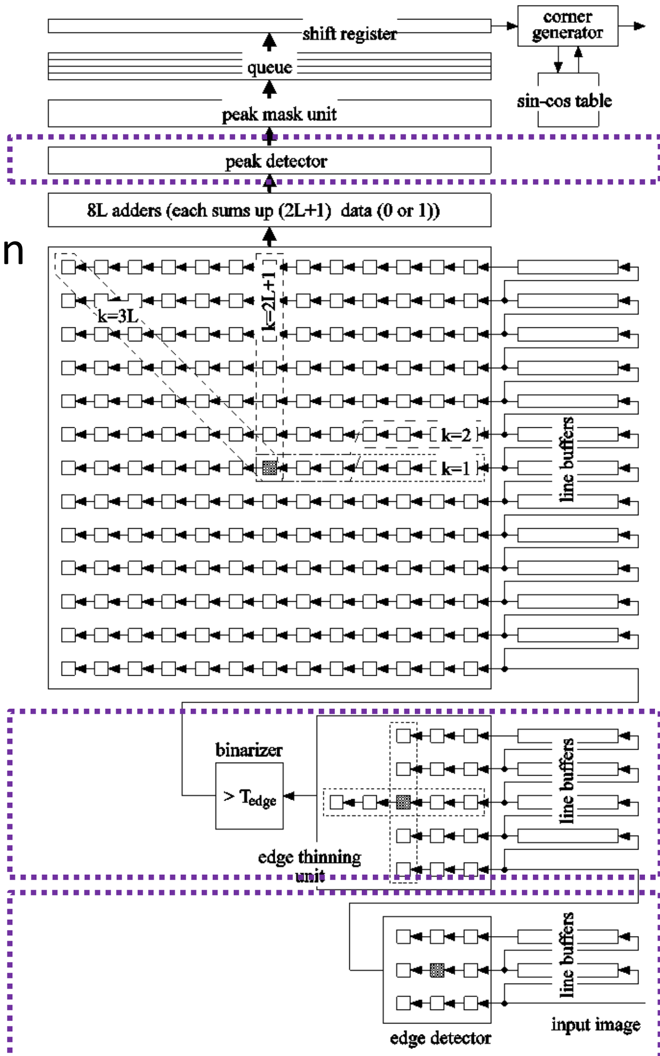
The binarized edges are held on a register array whose size is (2L+1) × (2L+1) and summed up along the radial lines which are drawn from the center to the 8L pixels on the edges. 8L data are counted up at the same time.

FPGA Implementation

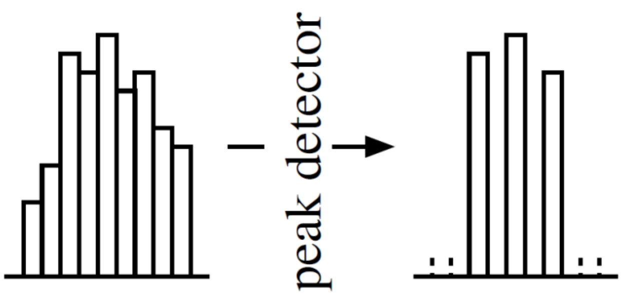
corner detection

edge thinning

edge detection



peak detector

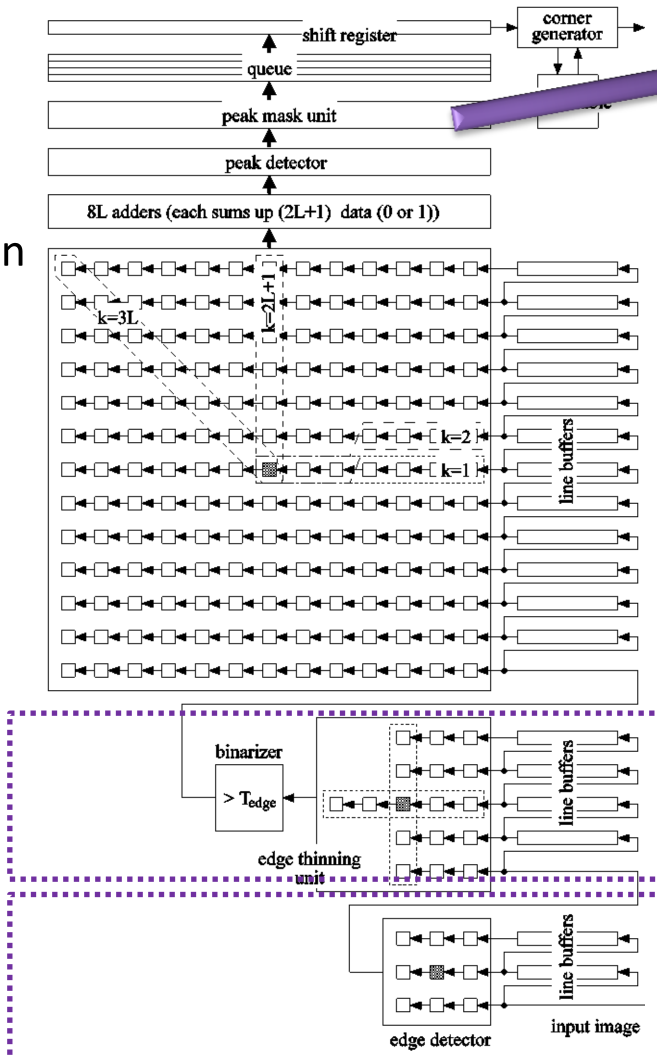


FPGA Implementation

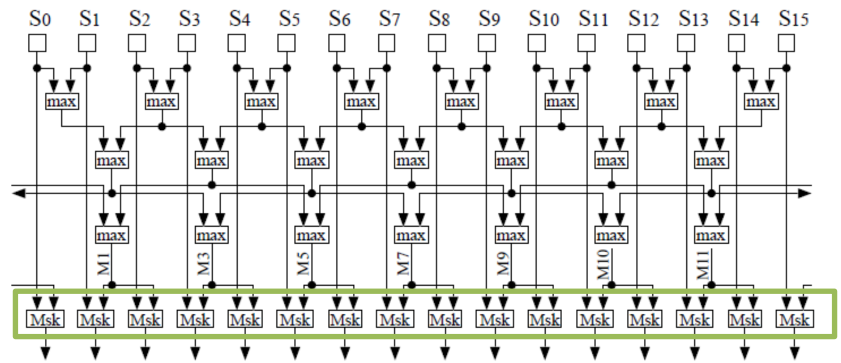
corner detection

edge thinning

edge detection



peak mask unit

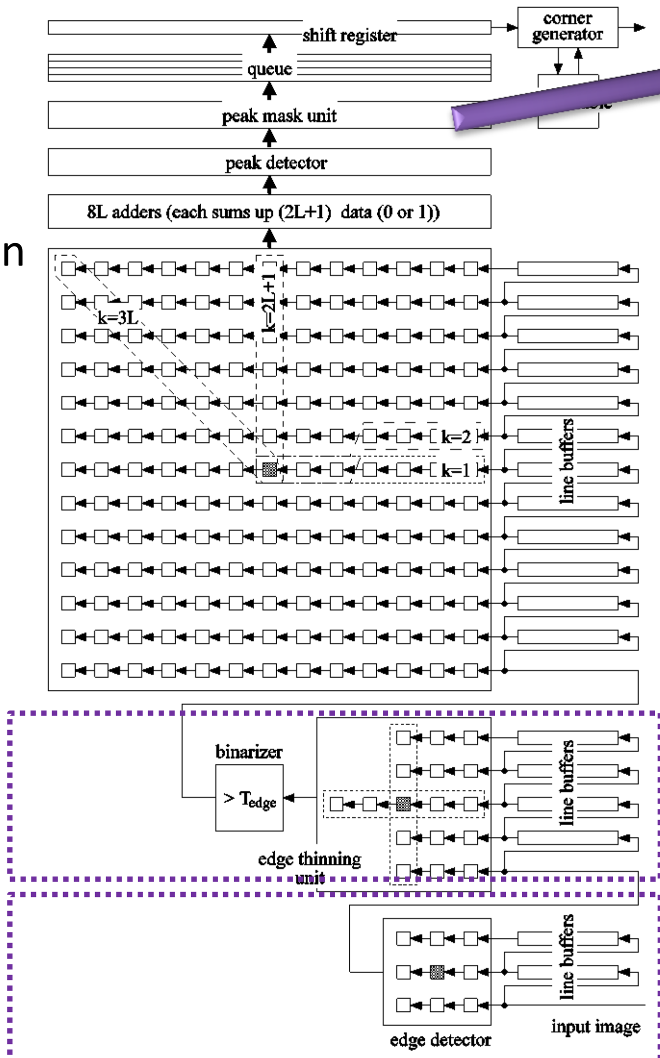


FPGA Implementation

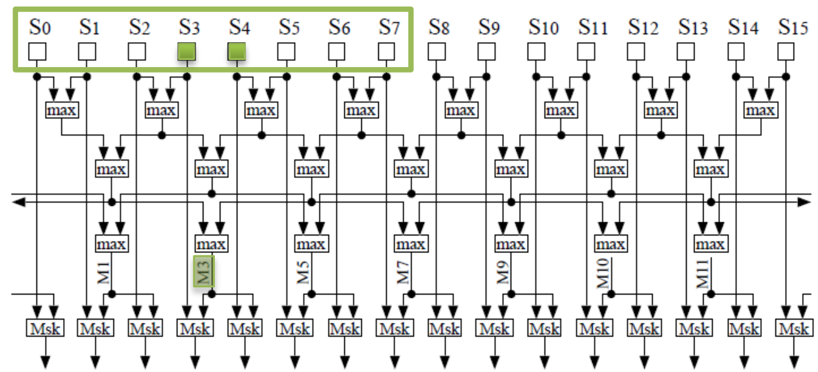
corner detection

edge thinning

edge detection



peak mask unit

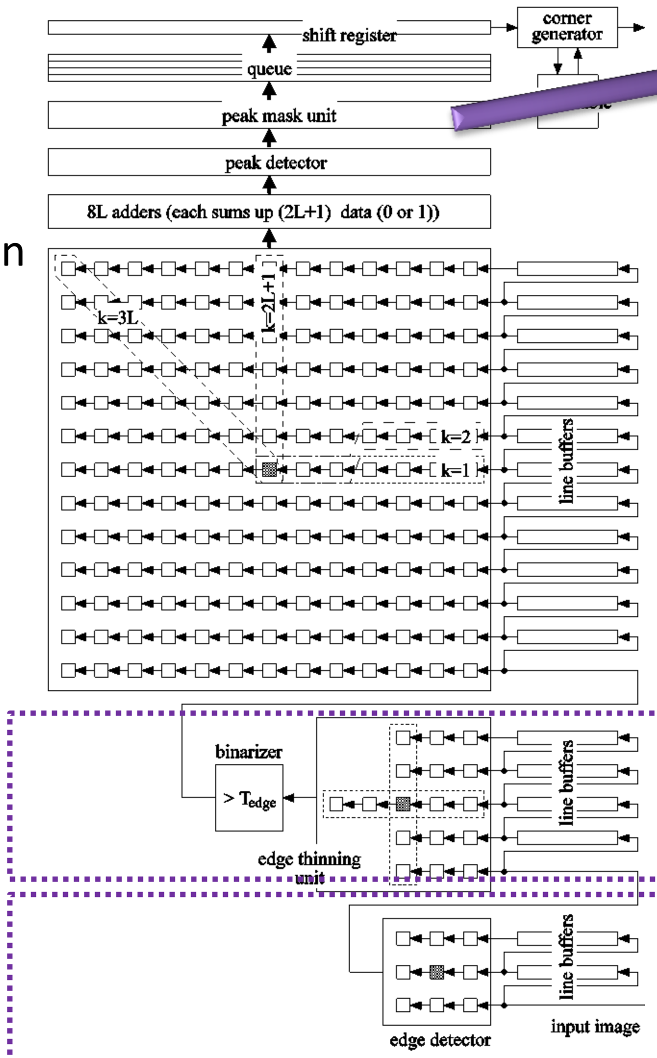


FPGA Implementation

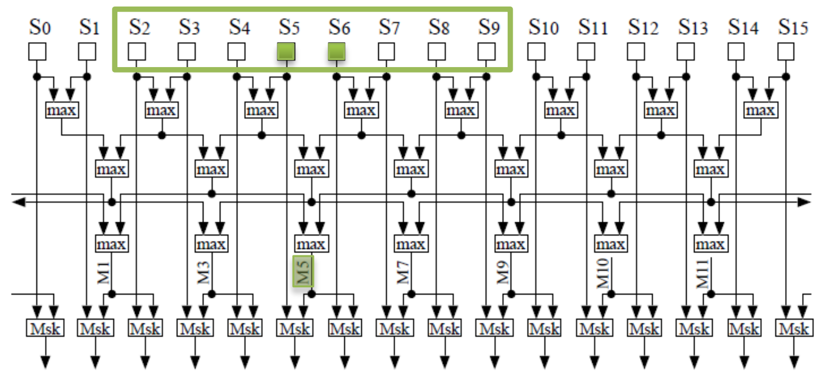
corner detection

edge thinning

edge detection



peak mask unit

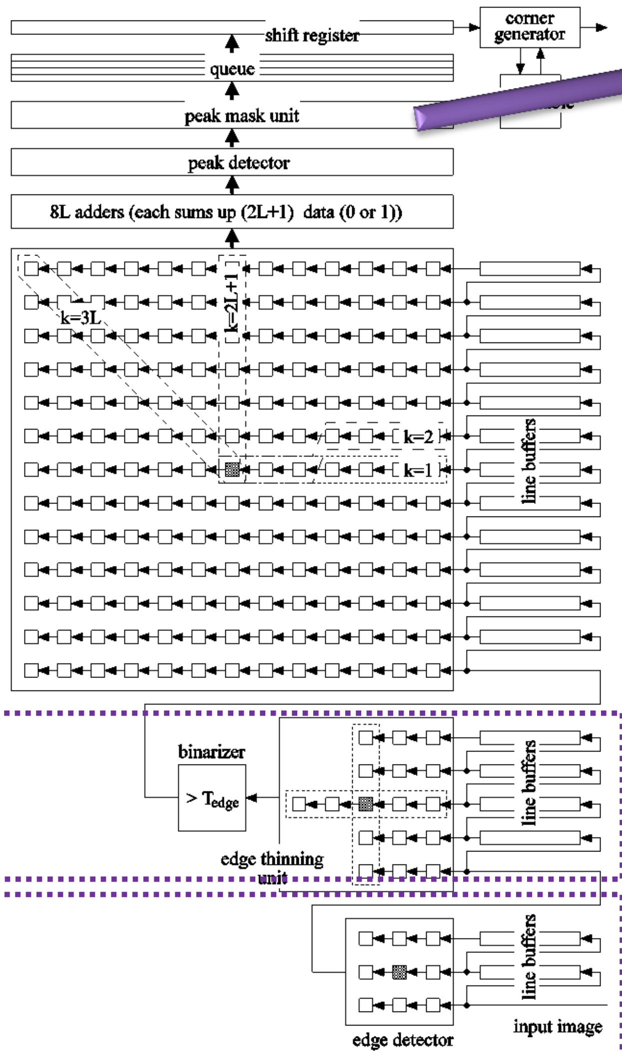


FPGA Implementation

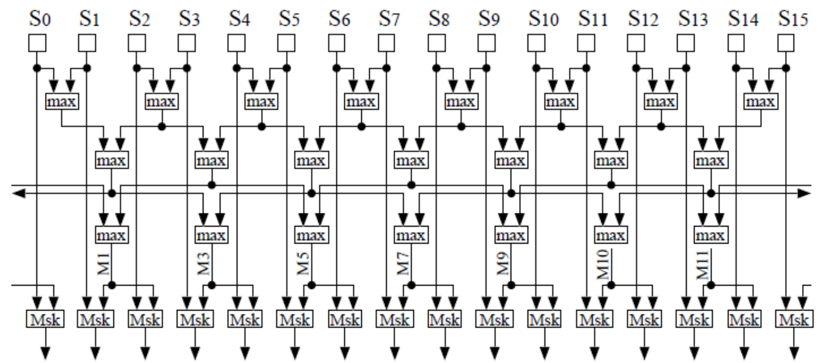
corner detection

edge thinning

edge detection



peak mask unit



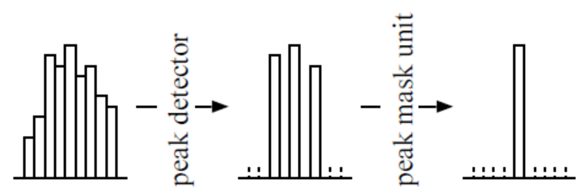


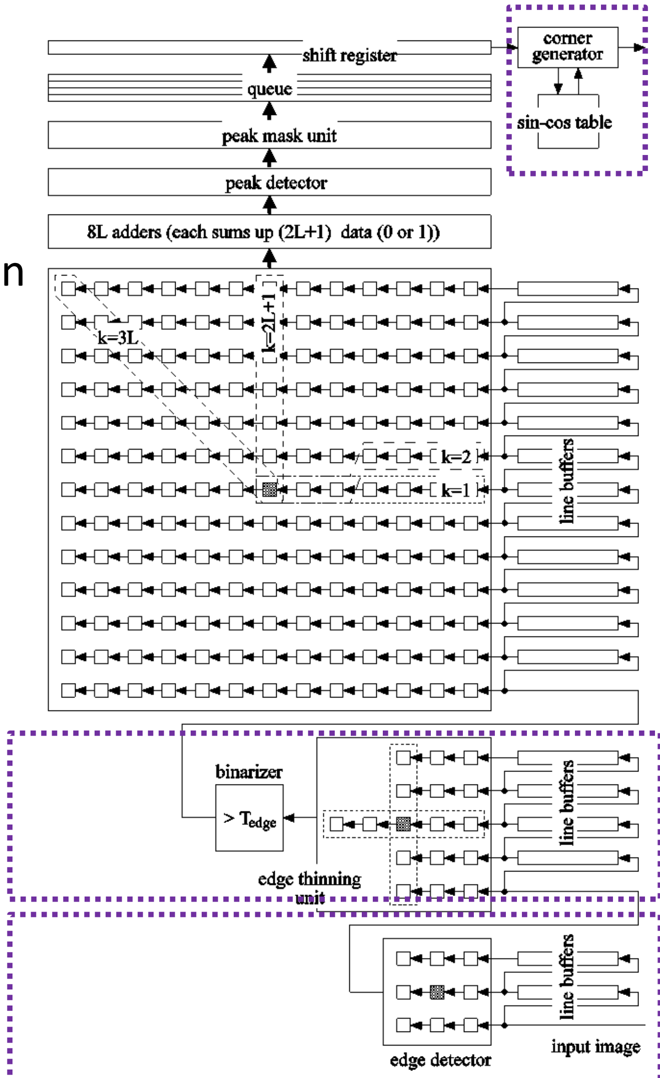
Fig. 7. An example of the peak detection and masking

FPGA Implementation

corner detection

edge thinning

edge detection



corner generator

(x, y)	(k_0, r_0)	(k_1, r_1)	(k_2, r_2)	(k_3, r_3)

(x,y): coordinates

k_i: gradient

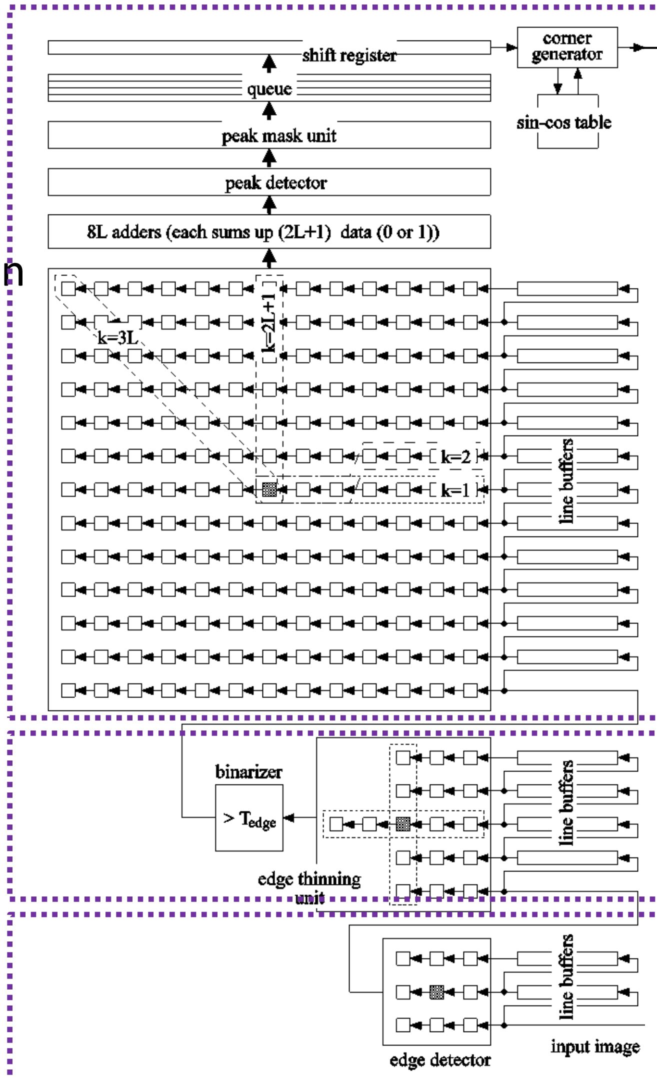
r_i : the distance from the origin to the line whose gradient is k_i, and which goes through (x, y).

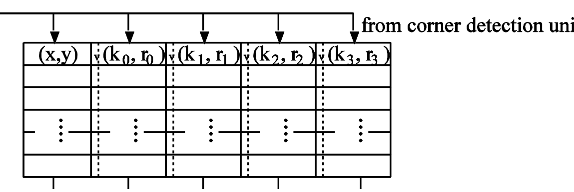
FPGA Implementation

corner detection

edge thinning

edge detection





(x, y)	(k_0, r_0)	(k_1, r_1)	(k_2, r_2)	(k_3, r_3)

(x,y): coordinates

k_i: gradient

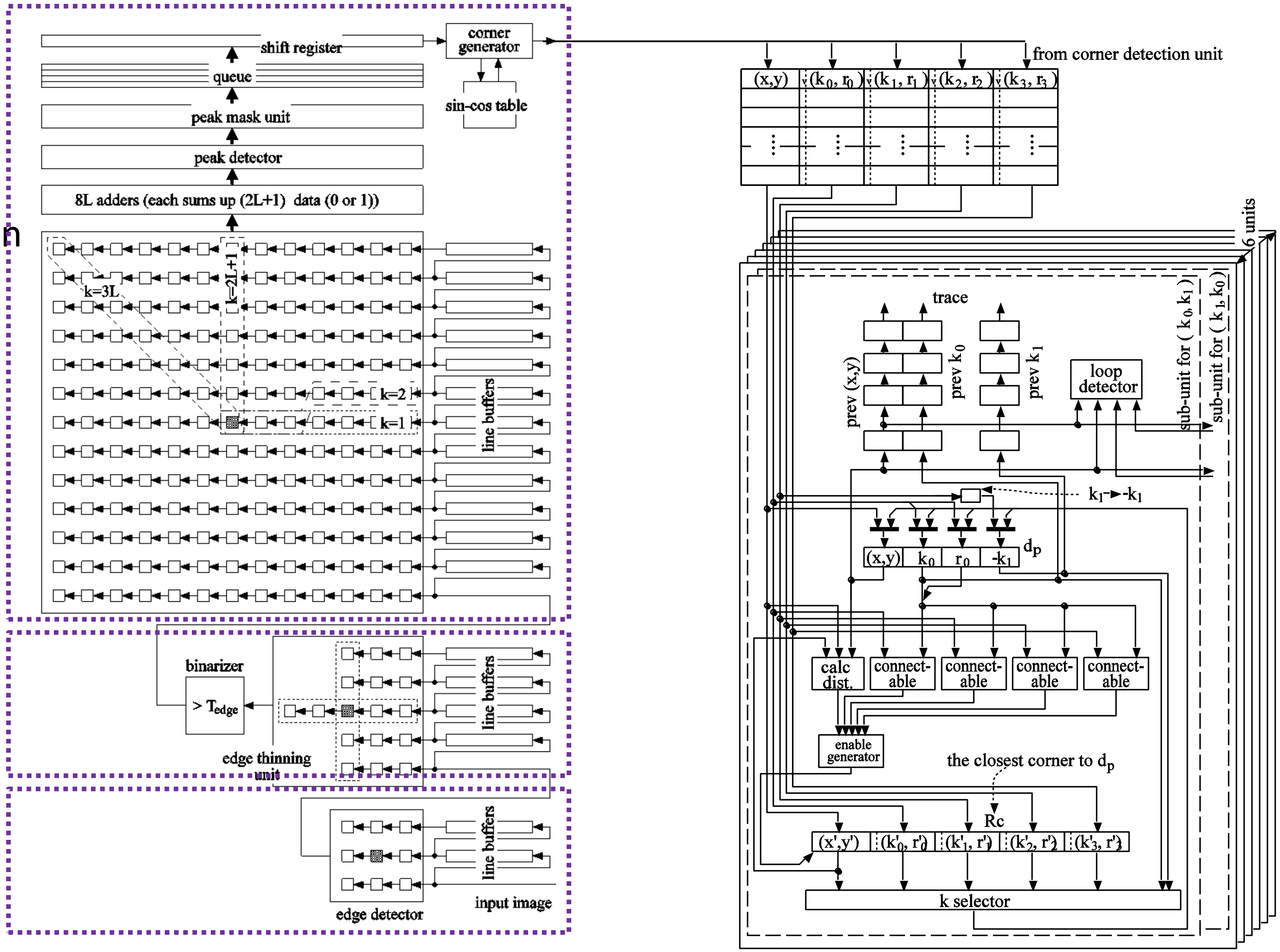
r_i : the distance from the origin to the line whose gradient is k_i, and which goes through (x, y).

FPGA Implementation

corner detection

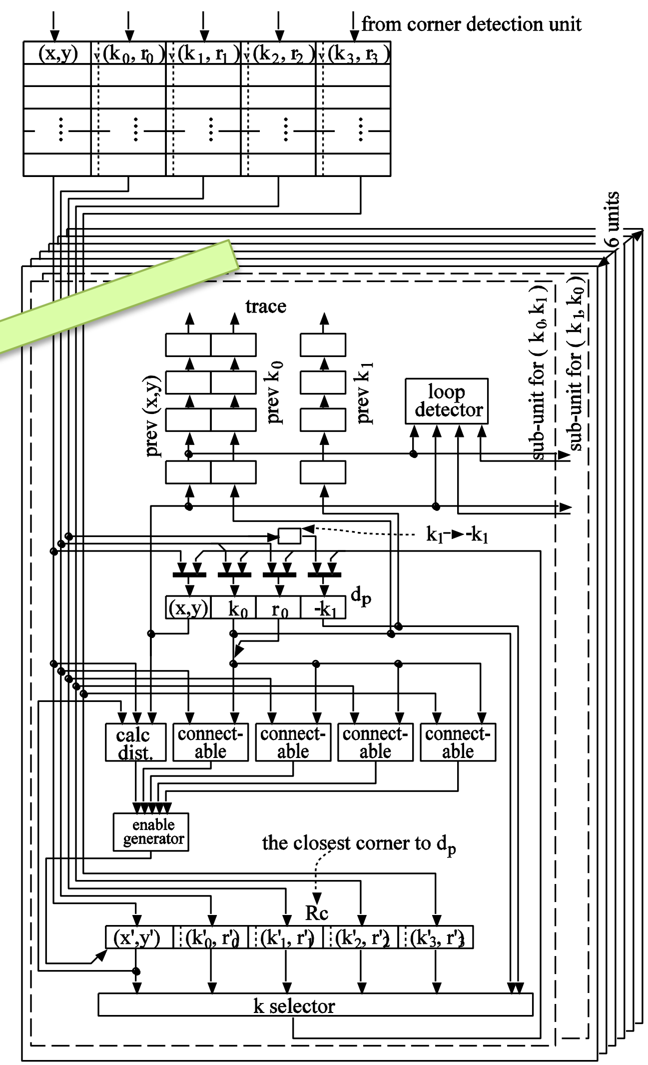
edge thinning

edge detection



polygon detection

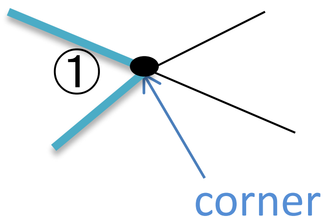
FPGA Implementation



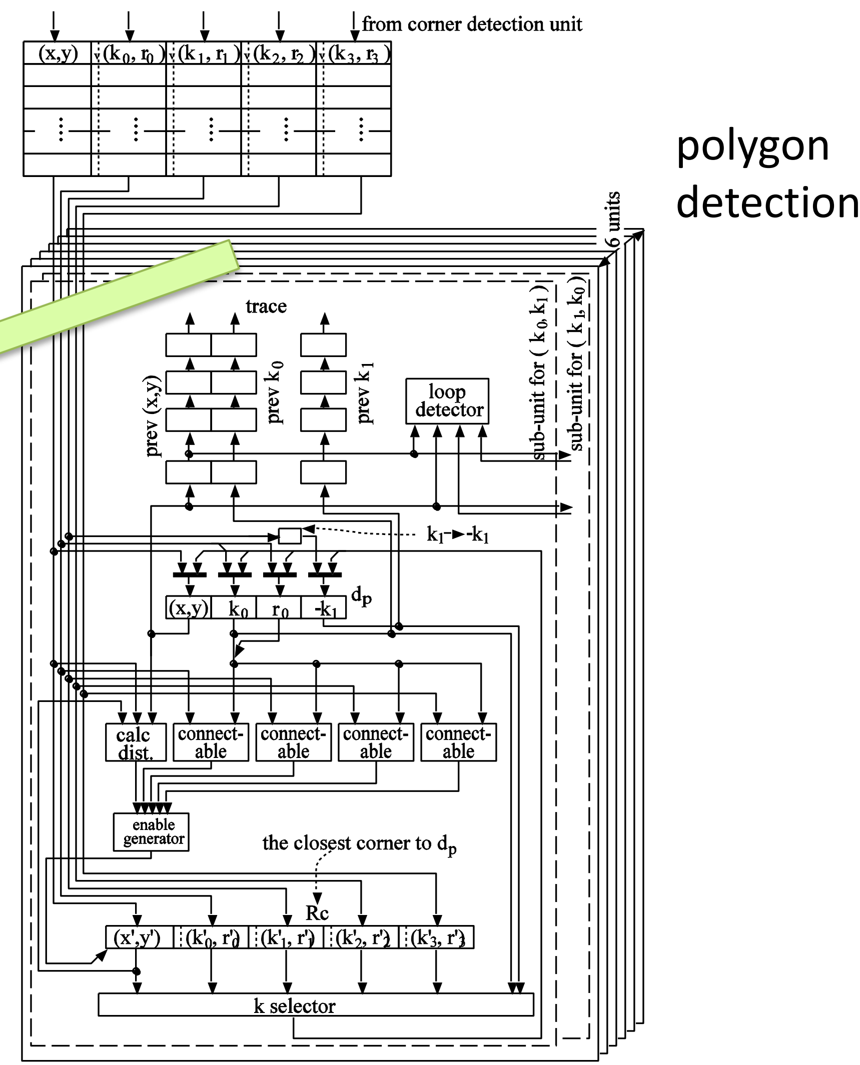
polygon detection

We use 6 units to trace all directions(up to 6) of the line segments crossing at the corner at the same time.

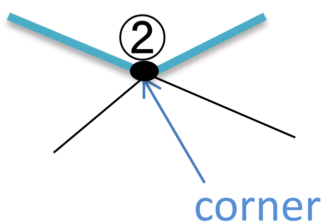
FPGA Implementation



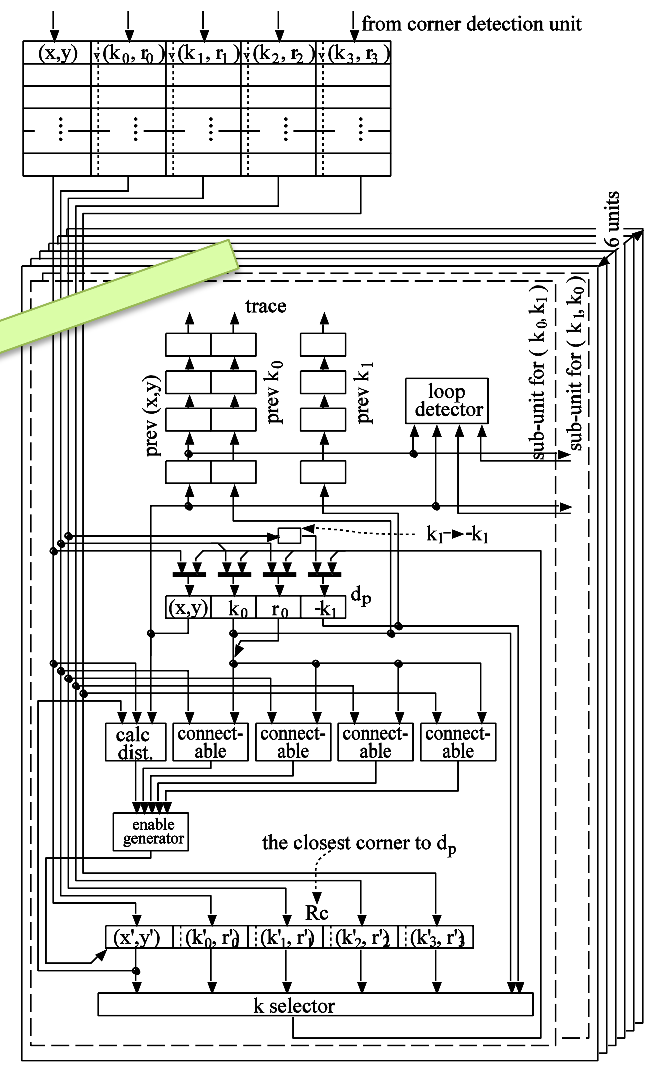
We use 6 units to trace all directions(up to 6) of the line segments crossing at the corner at the same time.



FPGA Implementation

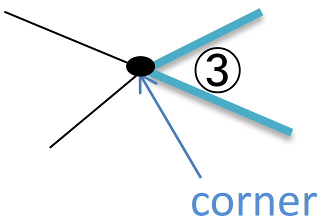


We use 6 units to trace all directions(up to 6) of the line segments crossing at the corner at the same time.

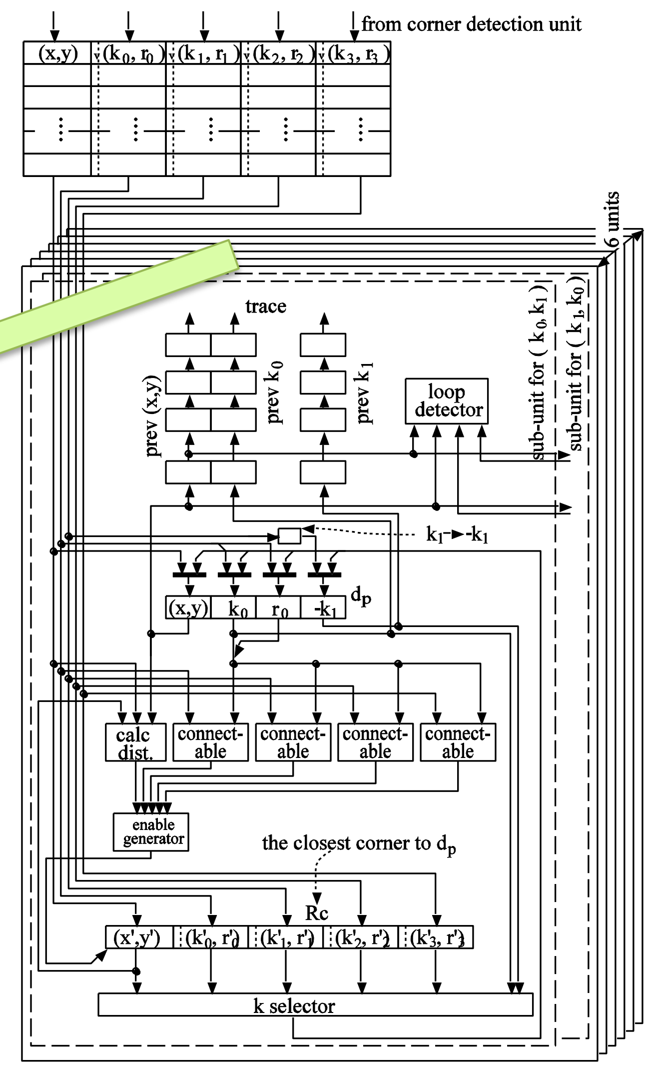


polygon detection

FPGA Implementation

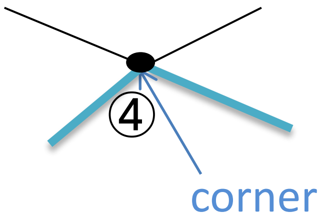


We use 6 units to trace all directions(up to 6) of the line segments crossing at the corner at the same time.

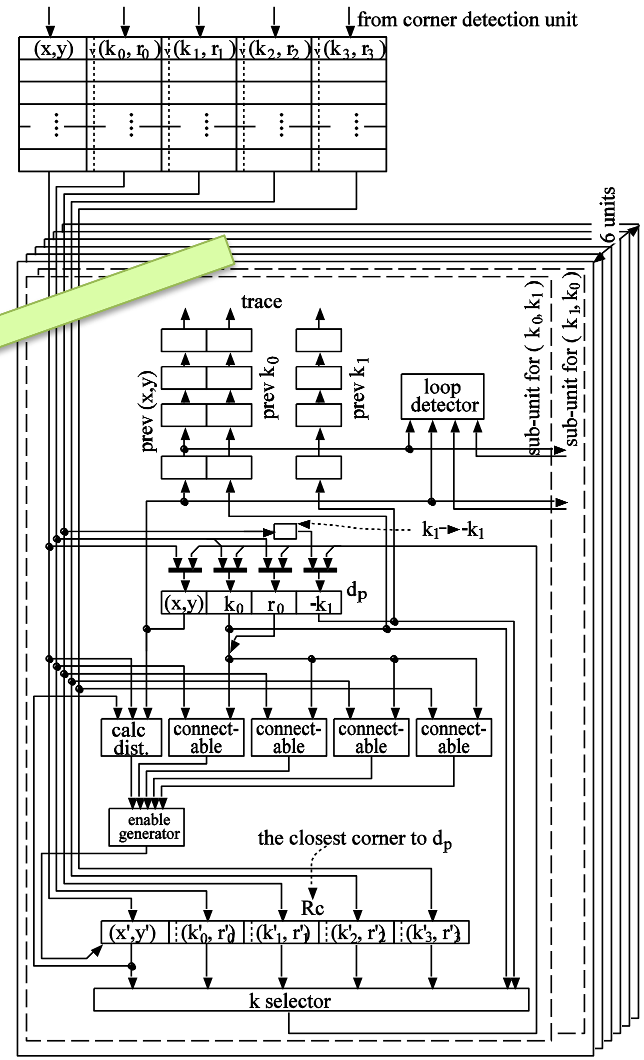


polygon detection

FPGA Implementation



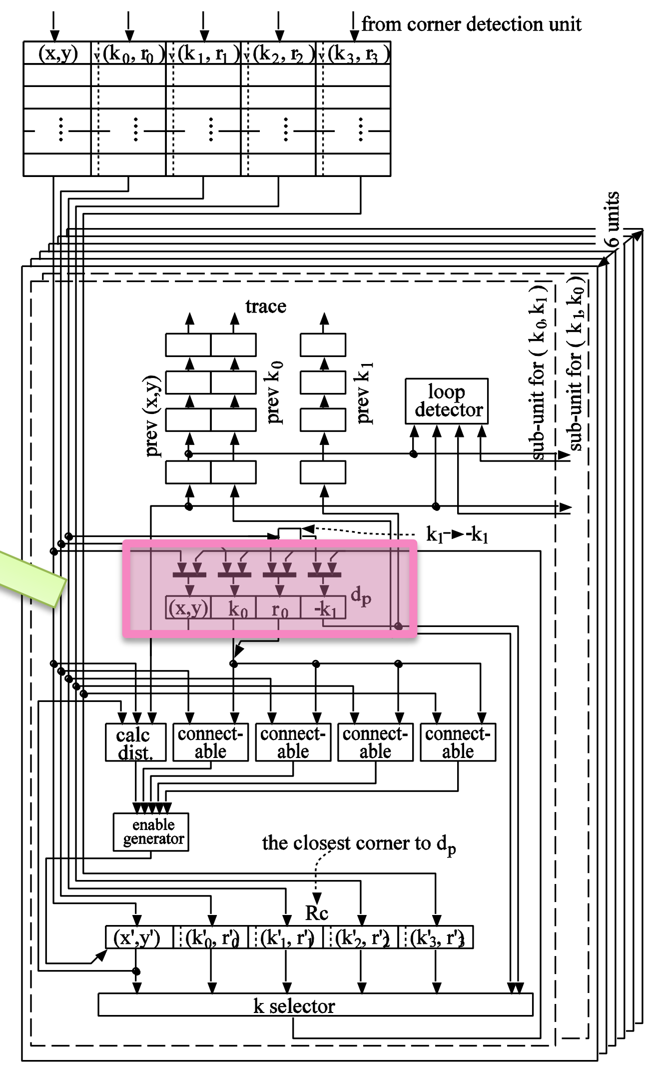
We use 6 units to trace all directions(up to 6) of the line segments crossing at the corner at the same time.



polygon detection

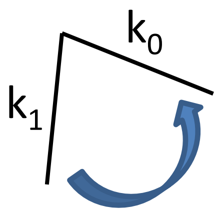
FPGA Implementation

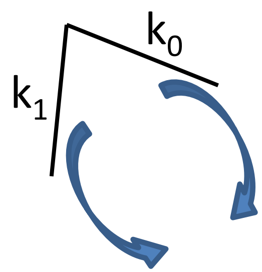
Data for a corner are read out from the buffer and put into the register set d_p in the 6 units.



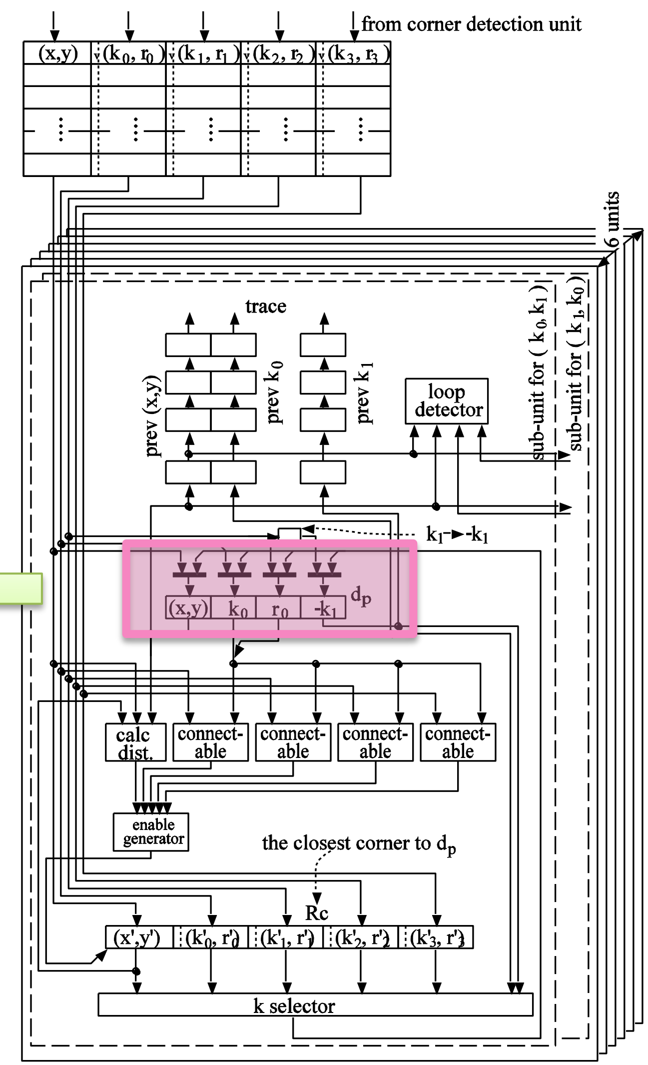
polygon detection

FPGA Implementation





In another sub-unit in this unit, k_1 and $-k_0$ are put to d_p instead of k_0 and $-k_1$ to trace the other side of the corner.



polygon detection

FPGA Implementation

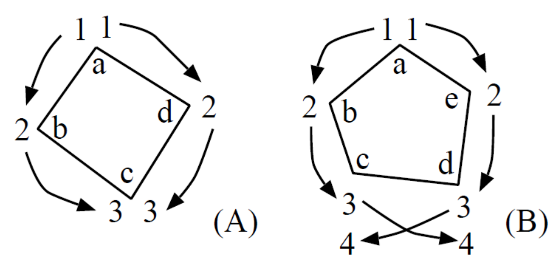
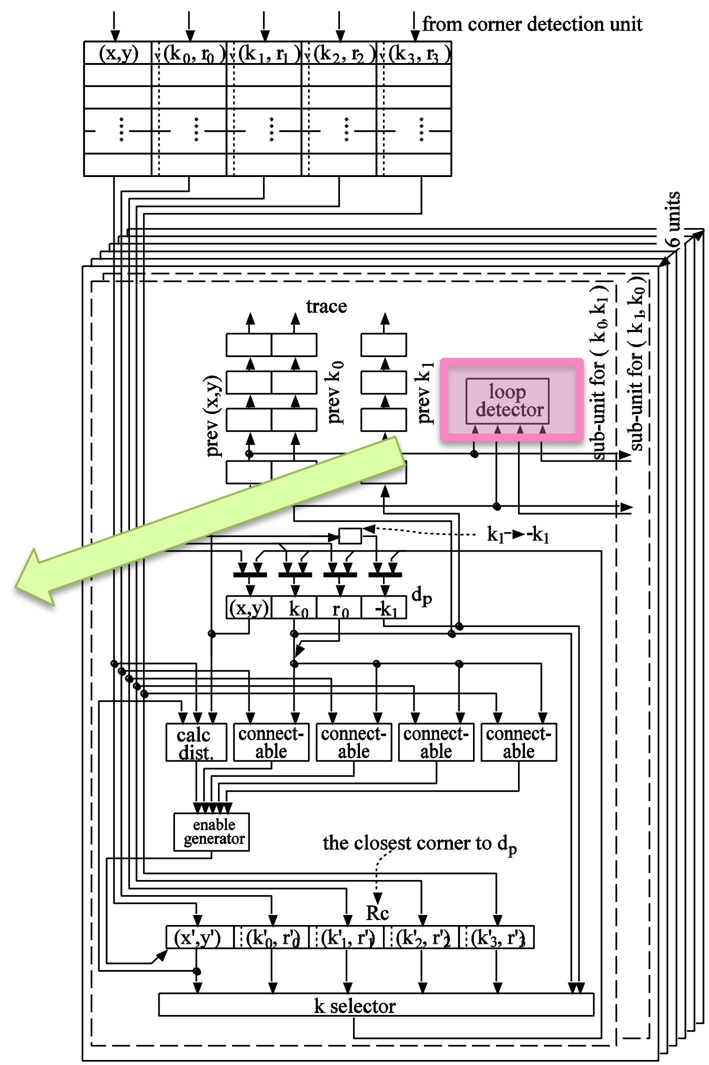


Fig. 9. Finding Connectable corners

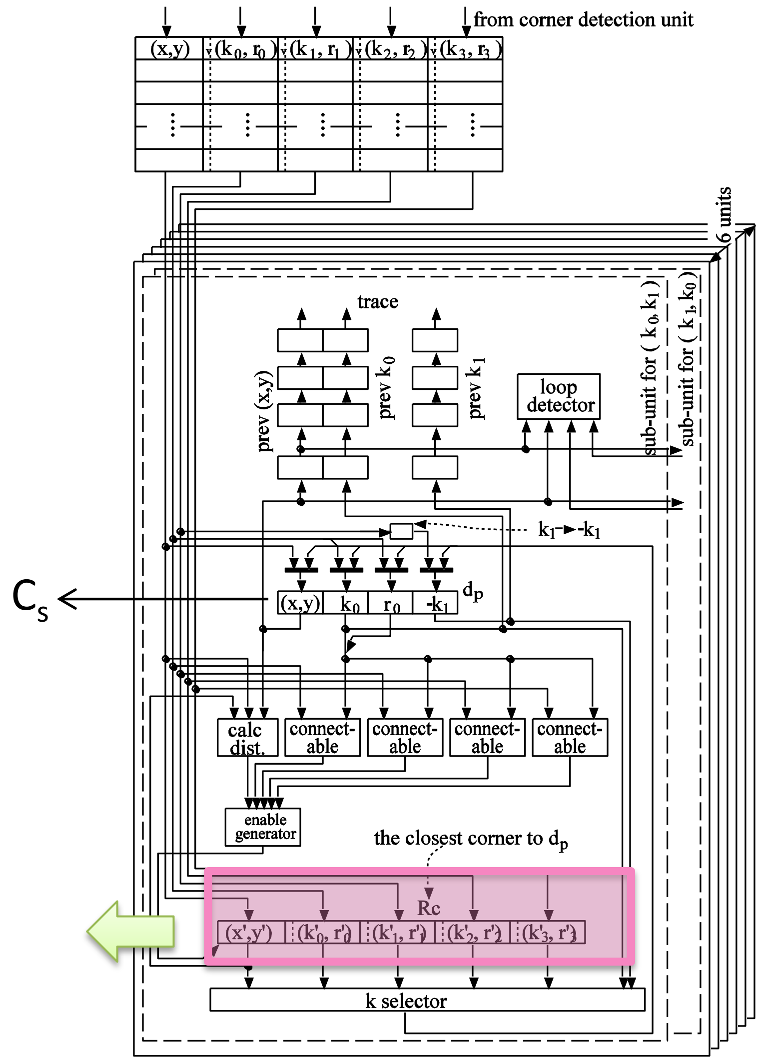


polygon detection

A: Comparing the addresses of the new corners detected in both sub-units by the loop detector.

B: The address of the new corner is also compared with the previous one in another sub-unit.

FPGA Implementation



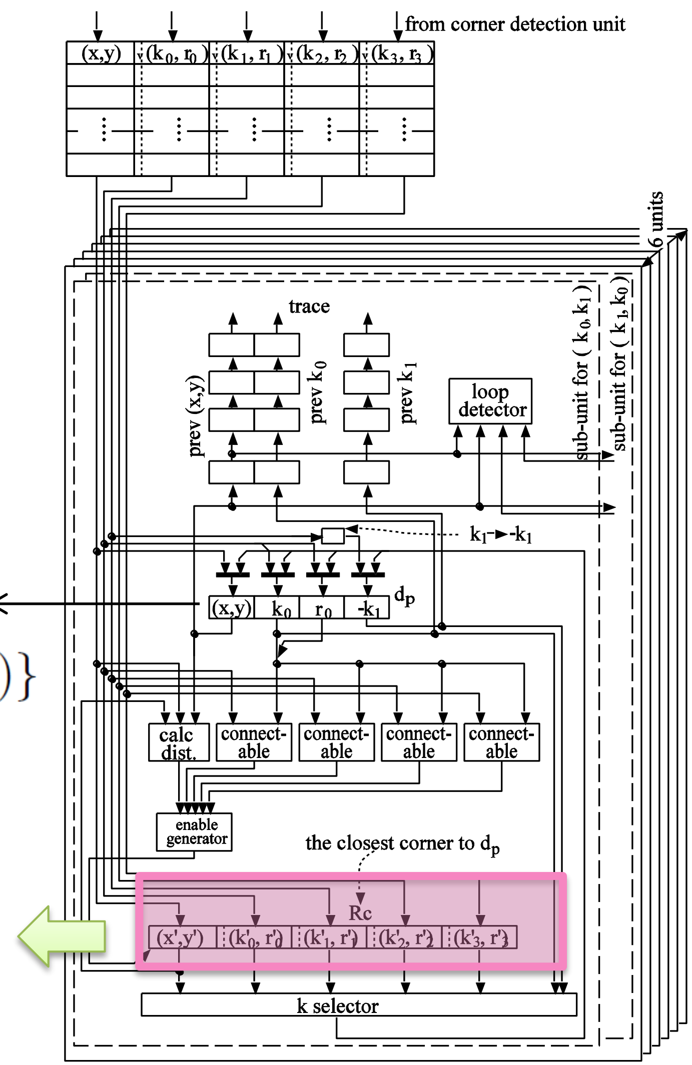
polygon detection

Register set R_c :
keep the closest corner to C_s

FPGA Implementation

polygon detection

C_s



$$C_c = \{(x', y'), (k'_0, r'_0), (k'_1, r'_1), (k'_2, r'_2), (k'_3, r'_3)\}$$

Suppose that a corner C_c is now held on the register set R_c as the closest connectable corner to C_s of the corners that have been detected already.

FPGA Implementation

$$C_c = \{(x', y'), (k_0', r_0'), (k_1', r_1'), (k_2', r_2'), (k_3', r_3')\}$$

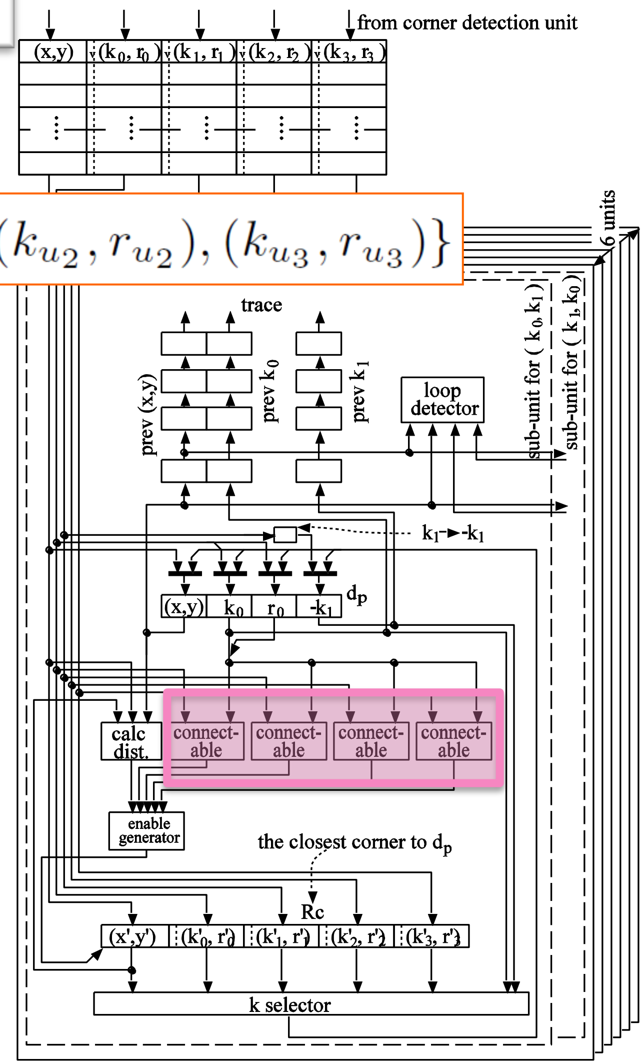
$$C_u = \{(x_u, y_u), (k_{u0}, r_{u0}), (k_{u1}, r_{u1}), (k_{u2}, r_{u2}), (k_{u3}, r_{u3})\}$$

C_c is replaced by C_u if the following conditions are satisfied

① ‘connectable’unit: connectable to C_s

$$|r_0 - r_{u_i}| \leq T_{dist}$$

$$|k_0 - -k_{u_i}| \leq T_{grad}$$



polygon detection

FPGA Implementation

$$C_c = \{(x', y'), (k'_0, r'_0), (k'_1, r'_1), (k'_2, r'_2), (k'_3, r'_3)\}$$

$$C_u \;=\; \{(x_u, y_u), (k_{u0}, r_{u0}), (k_{u1}, r_{u1}), (k_{u2}, r_{u2}), (k_{u3}, r_{u3})\}$$

C_c is replaced by C_u if the following conditions are satisfied

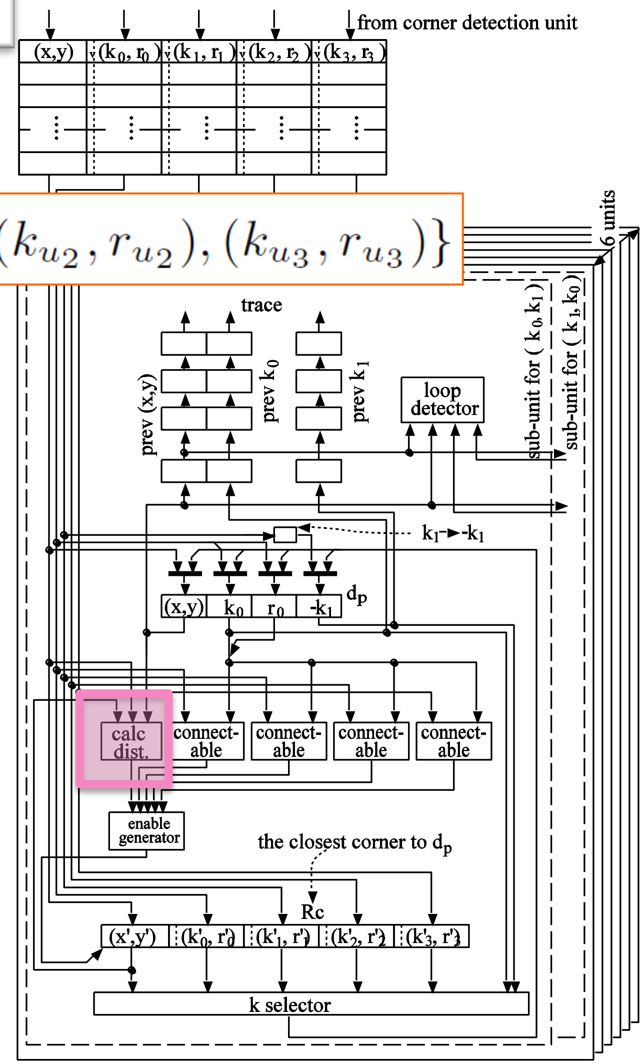
① 'connectable' unit: connectable to C_s

$$|r_0 - r_{u_i}| \leq T_{dist}$$

$$|k_0 - -k_{u_i}| \leq T_{grad}$$

② 'calc.dist' unit: choose the closest one

$$|x - x_u| + |y - y_u| < |x - x'| + |y - y'|$$



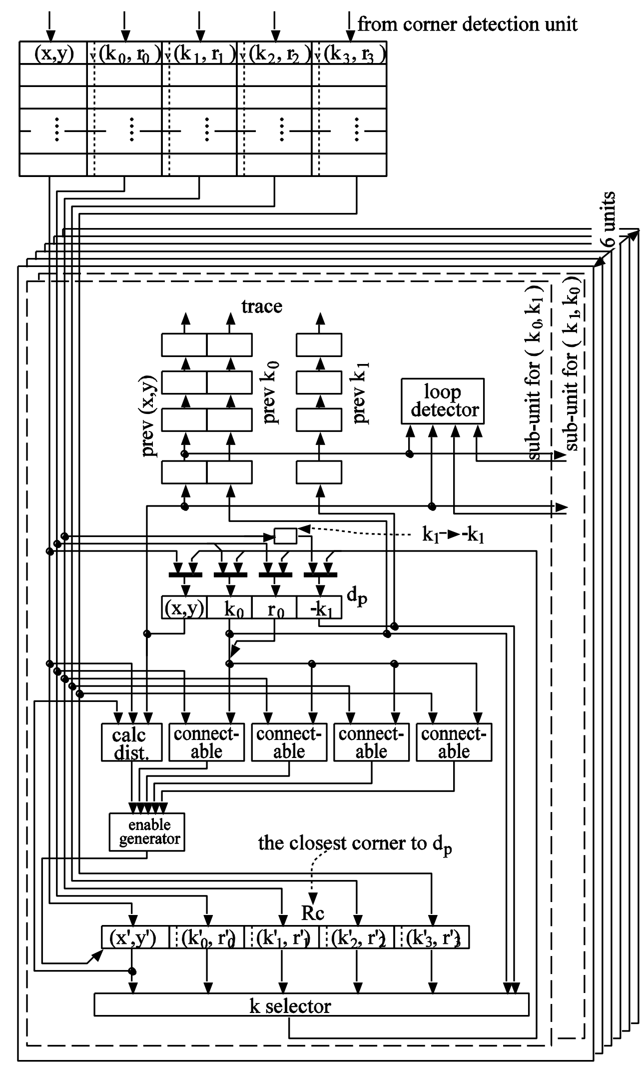
FPGA Implementation

In our method, all corners in the buffer are scanned, and all angles in each corner is compared with the angle on d_p in parallel.

By managing the corners using their k_i or r_i, we can reduce the number of the corners which should be compared with d_p, but we have chosen not to use those sophisticated approaches, because of the following two reasons.

1. We can still achieve more than 30fps for 640 × 480 pixel images if the number of the corners is less than 1000.

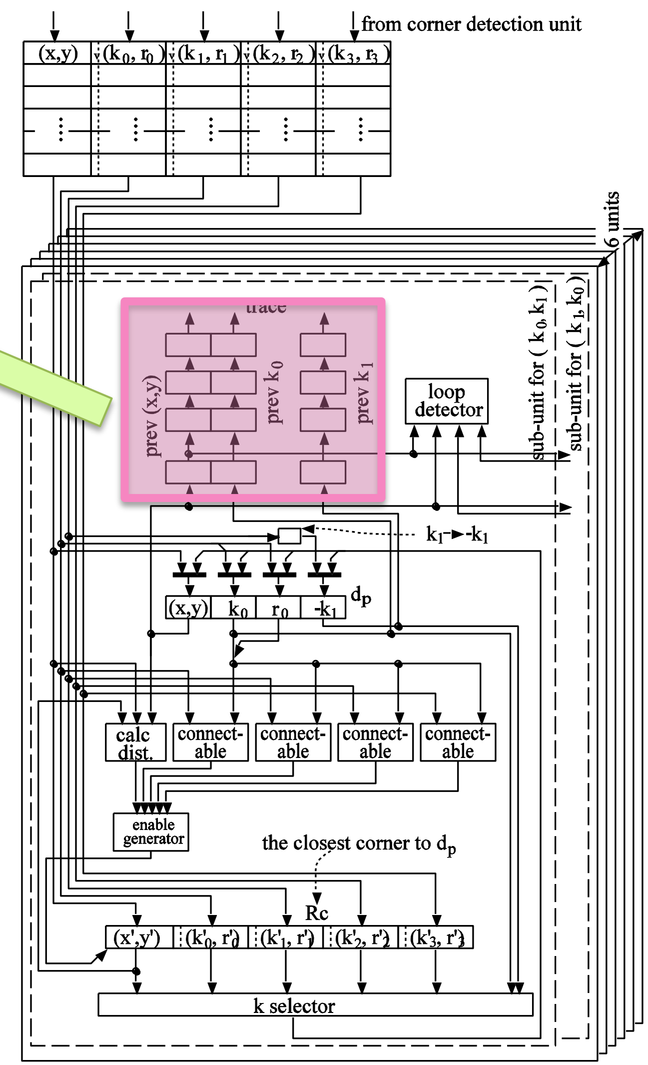
2. With our corner detector, the corners which are unrelated to the polygons are rarely detected.



polygon detection

FPGA Implementation

After comparing d_p with all angles of each other, one of the angles of the corner on R_c is chosen according to the two angles on d_p and put into d_p. At the same time, the angle on d_p is moved to the shift register to keep the trace of the tracking.



polygon detection

Experimental Results -1

- Experimental environment：Xilinx XC4VLX160
- Image size： X≦1024
- Circuit size and the performance

	L	LUTs(K)	BRAMs	Freq.(MHz)
corner detection	16	5.3	6	262.4
	32	23.3	8	219.1
polygon detection		6.5	5	101.0

Experimental Results -2

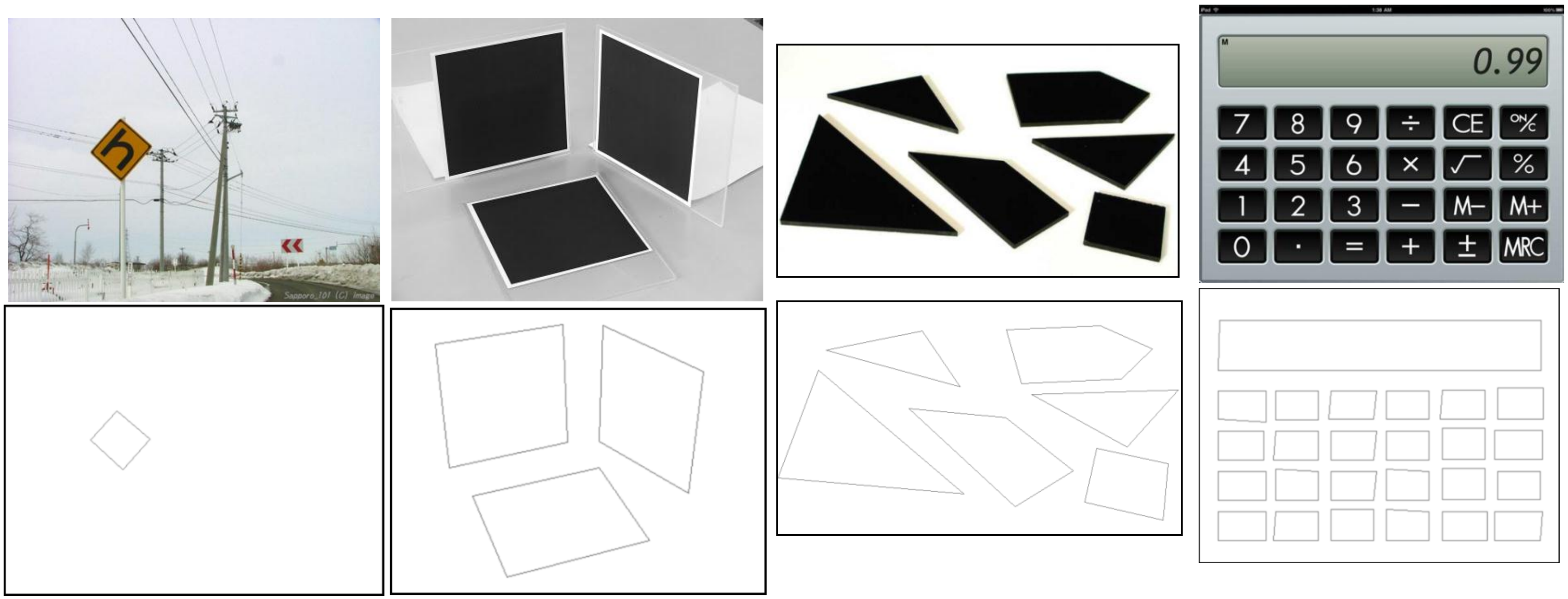


Table 2. Performance

		corner		polygon	
image	size	fps	#c	fps	#p
traffic sign	512×385	464.9	34	453.9	1
panels	400×300	746.0	48	708.9	3
pieces	640×364	391.5	43	382.3	6
calculator	480×360	527.3	284	251.1	26

Conclusions and Future Work

- We have described a corner detection algorithm, and the polygon detection system based on the algorithm.
- We can reduce the number of the corners in the images with our algorithm and obtain the correct angles of the corners. This makes it possible to detect polygons on hardware systems efficiently.

Conclusions and Future Work

- Future Work: Develop a technique for excluding false corners which are detected around shadows.

Thank you for your kind attention.