Faster and better: a machine learning approach to corner detection.

Ed Rosten

Los Alamos
NATIONAL LABORATORY
EST. 1943

What is interest point detection?

- Visually ‘salient’ features.
- Localized in 2D.
- Sparse.
- High ‘information’ content.
- Repeatable between images.

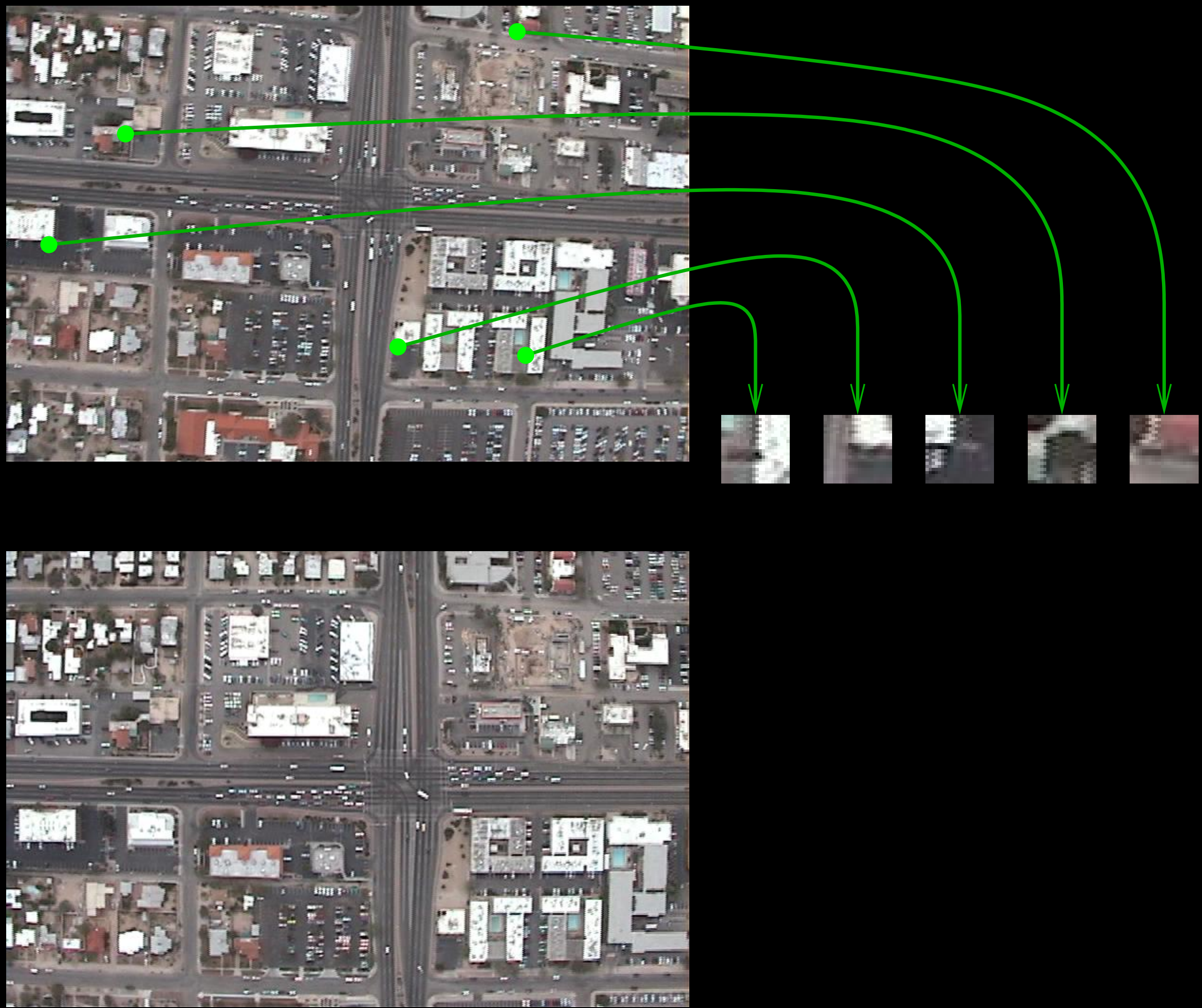
Useful for:

- 2D tracking, 3D tracking, SLAM, object recognition, …

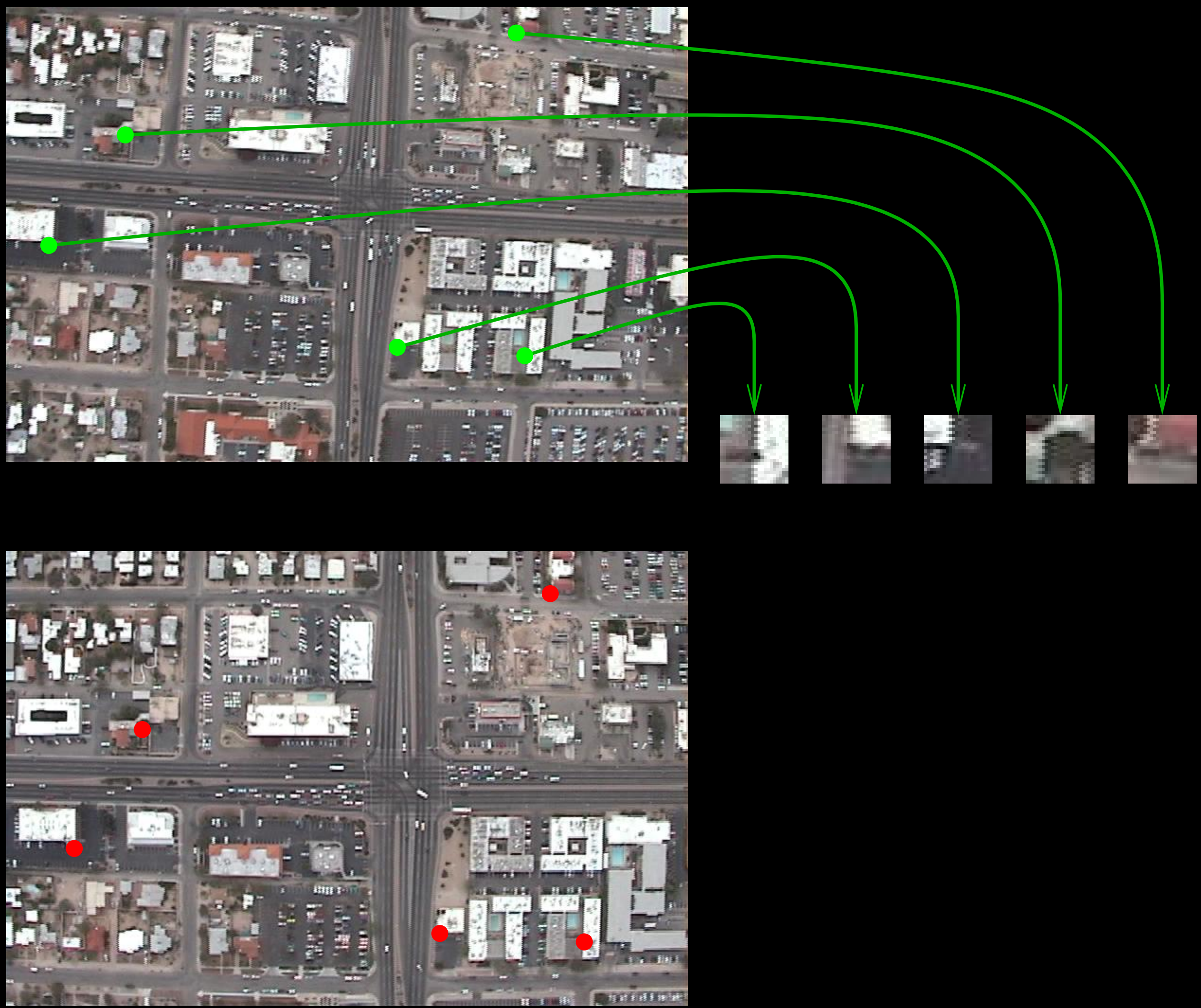
Example: registration

Example: registration

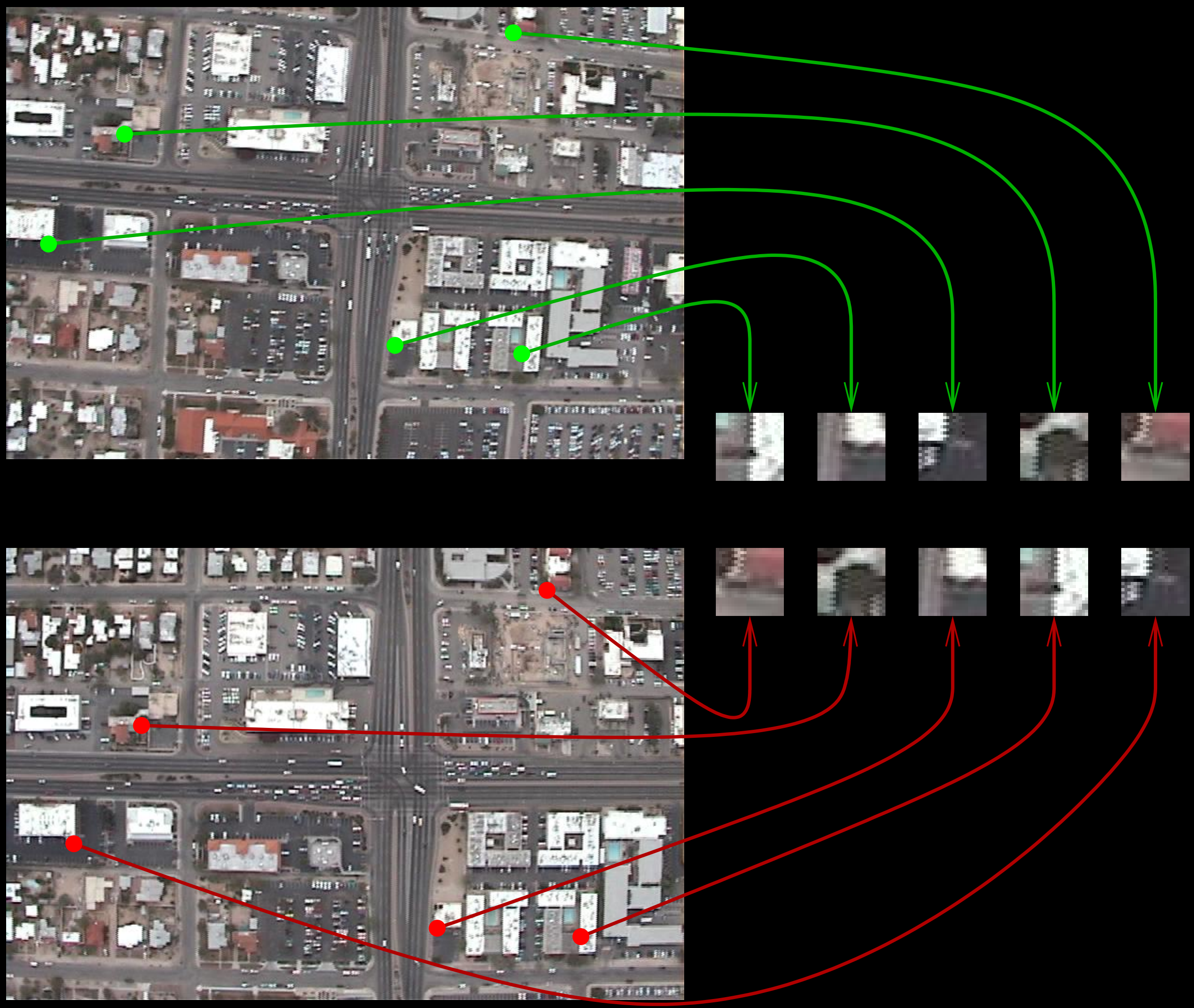
Example: registration



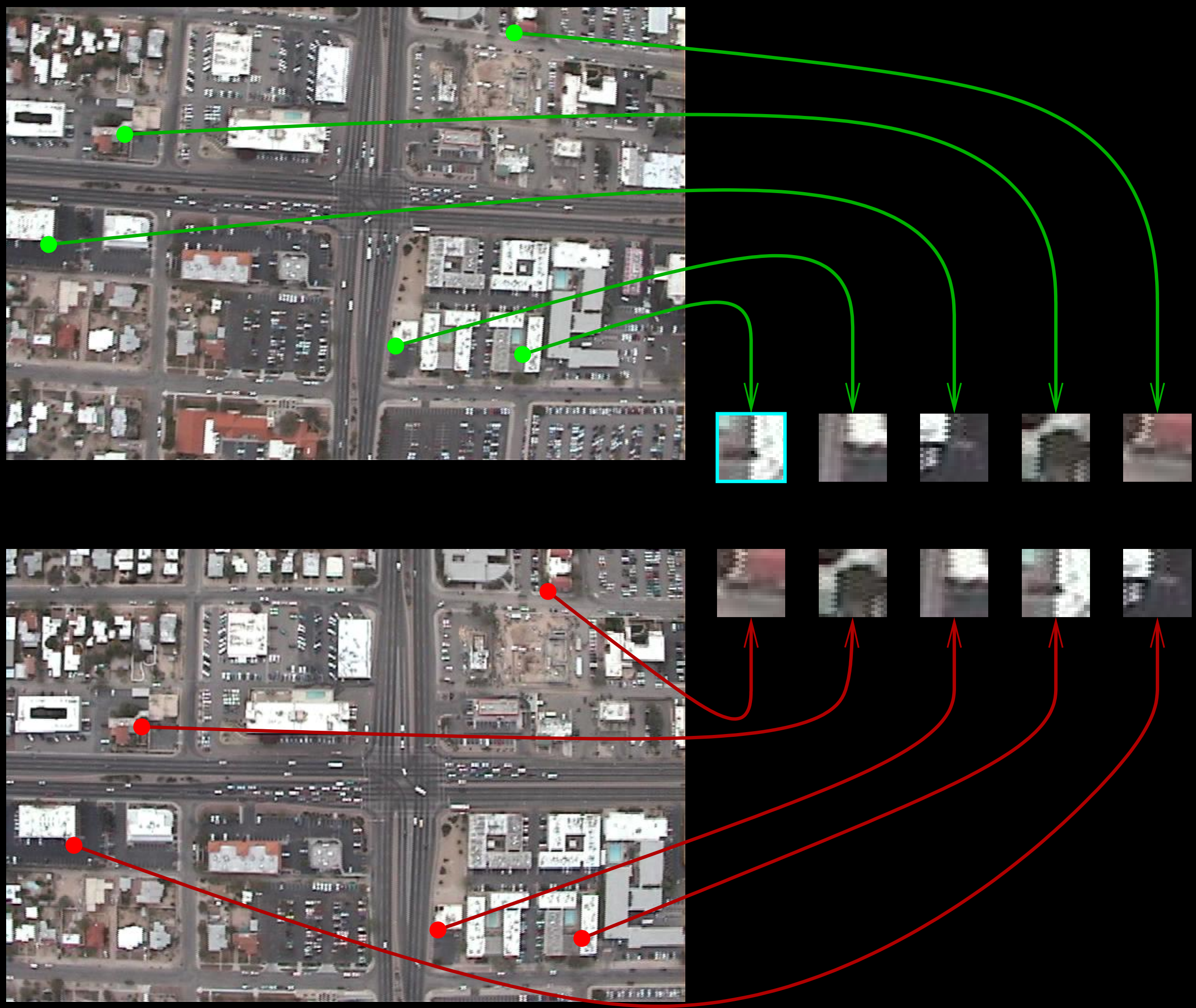
Example: registration



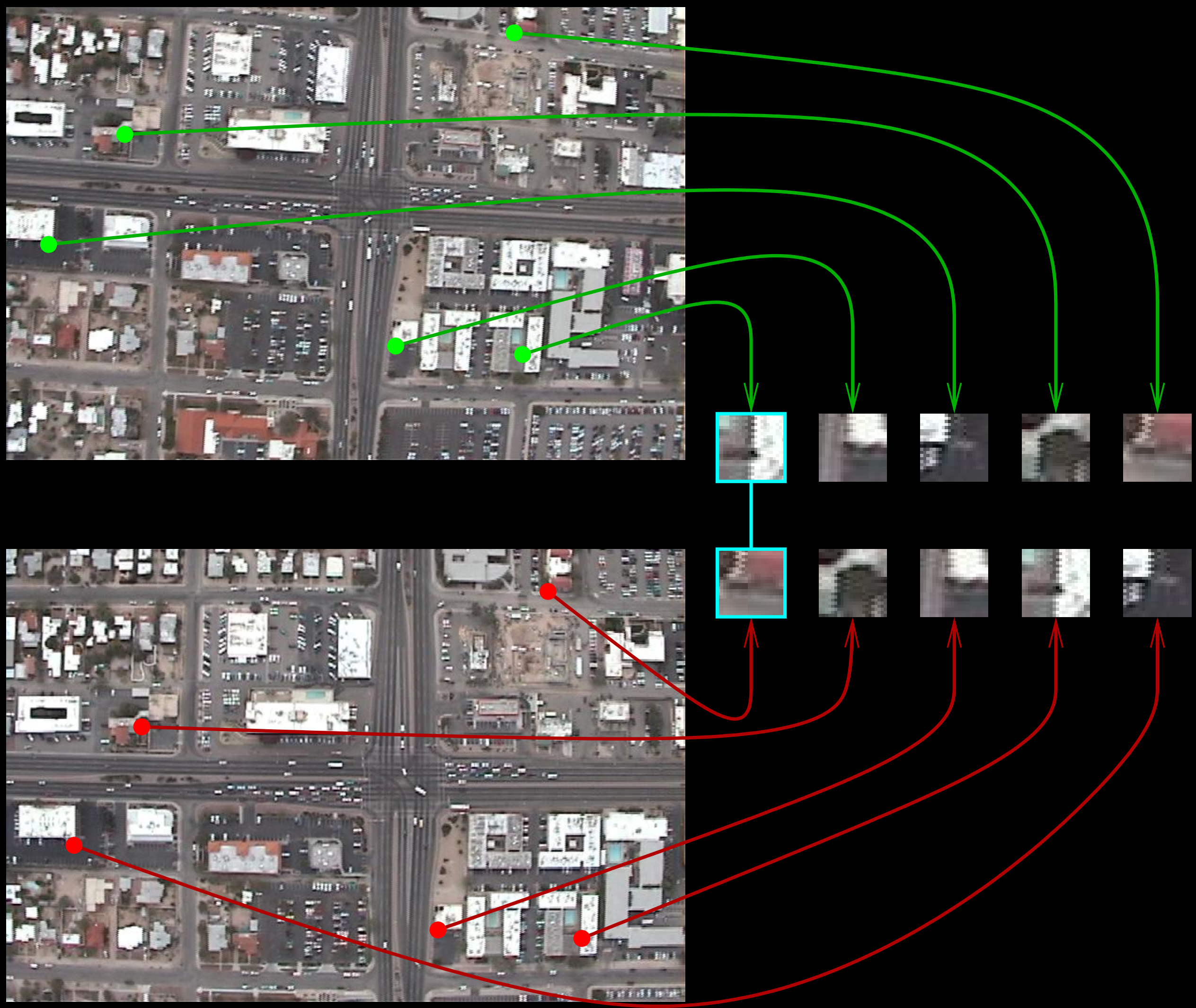
Example: registration



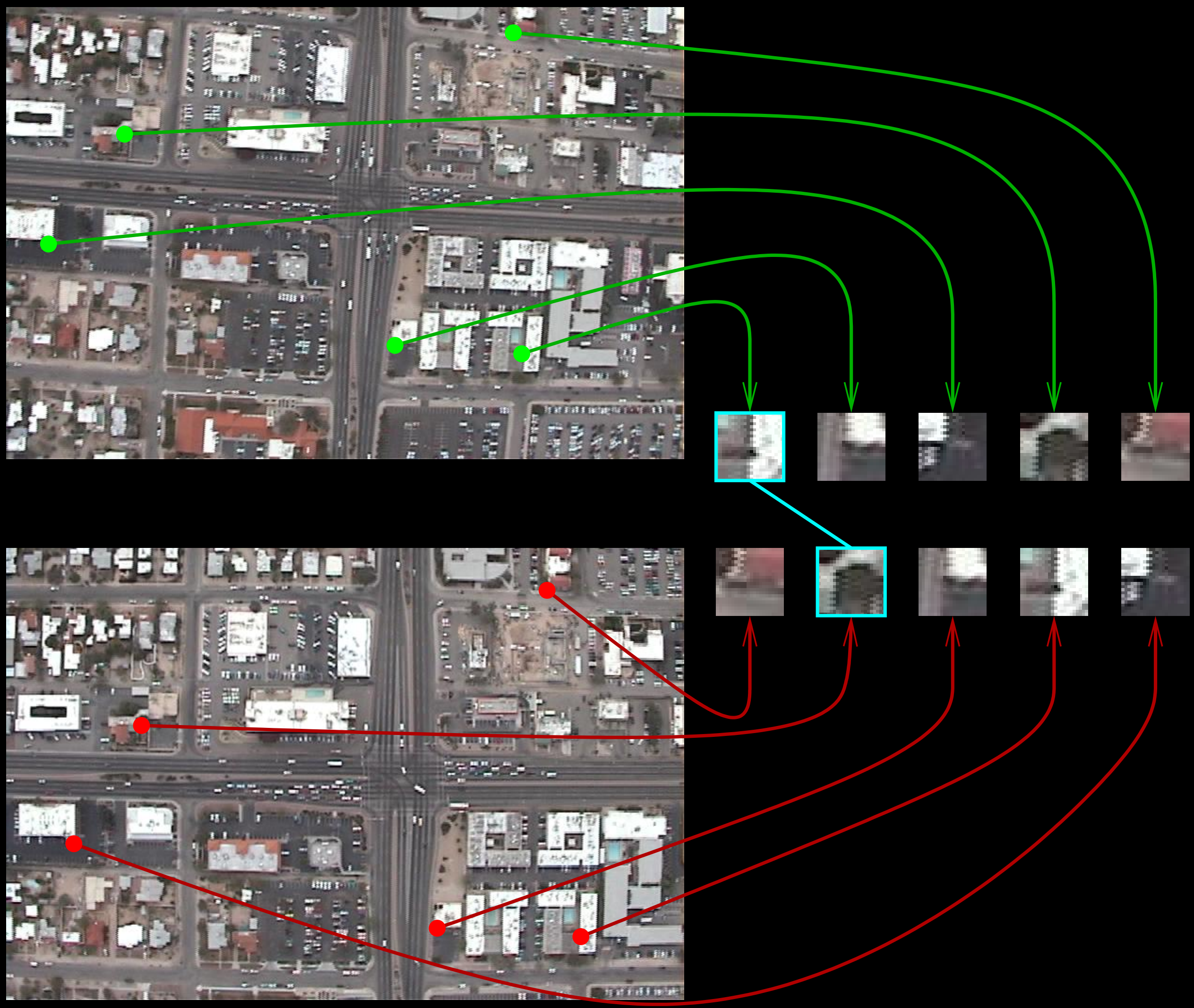
Example: registration



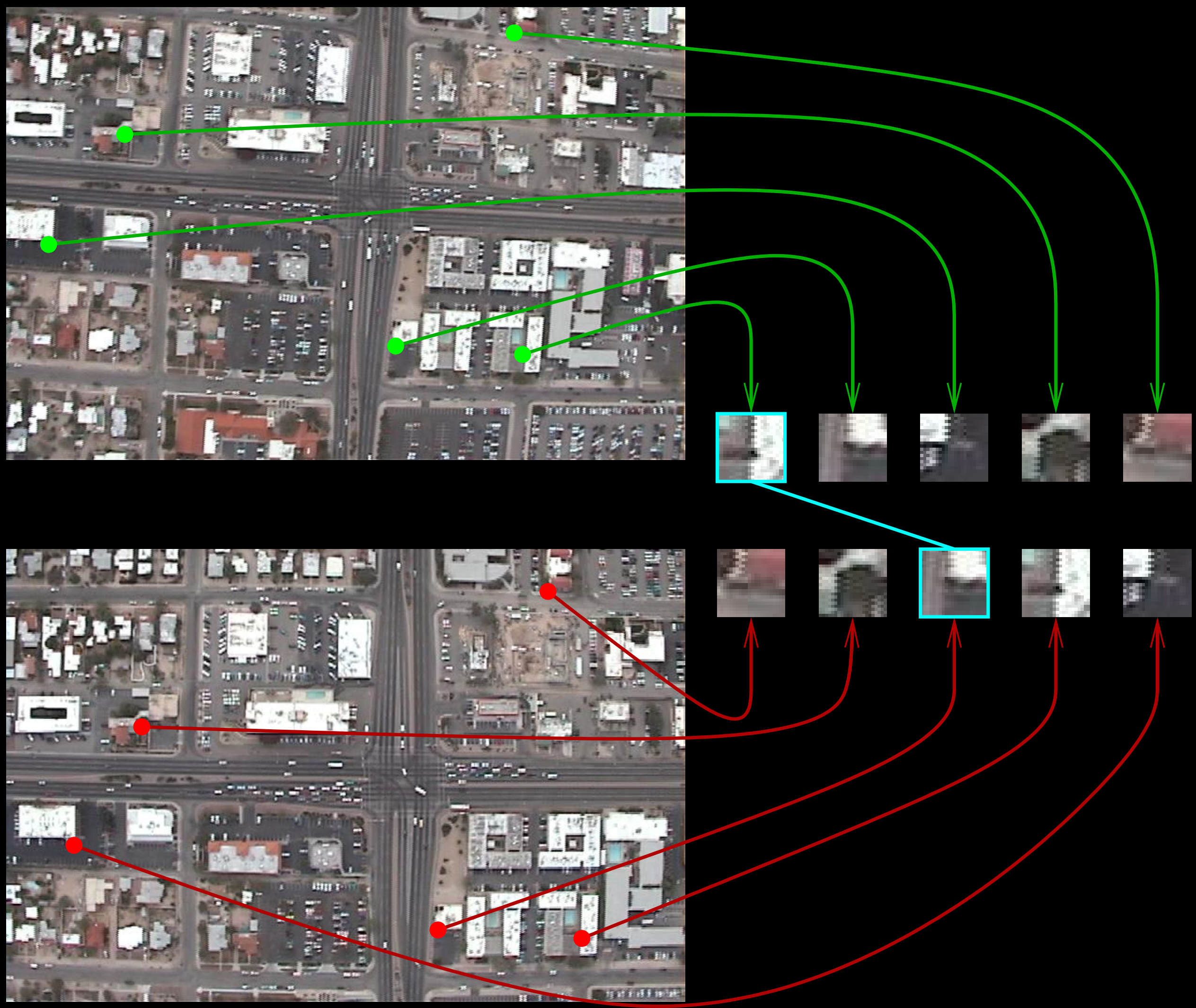
Example: registration



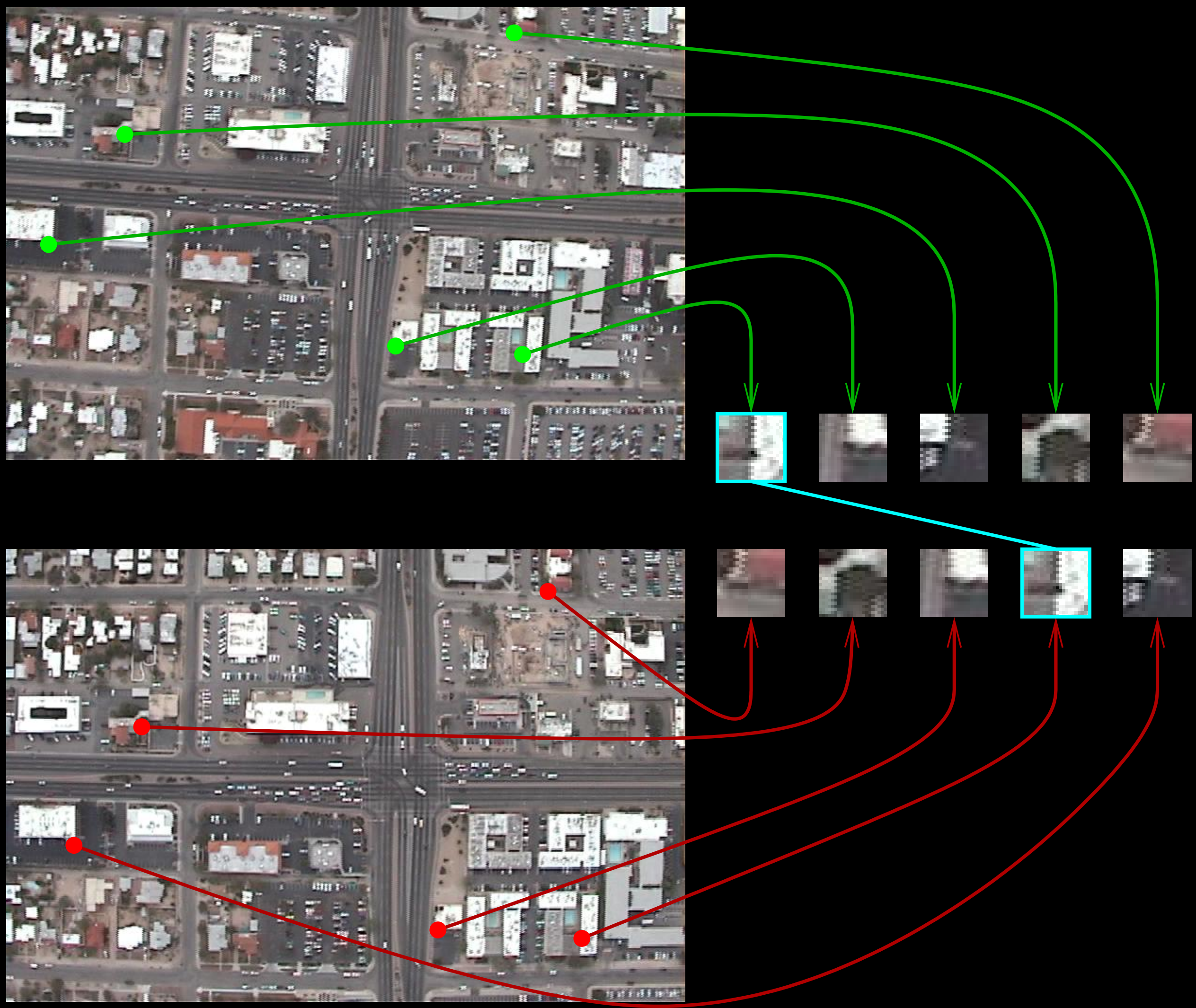
Example: registration



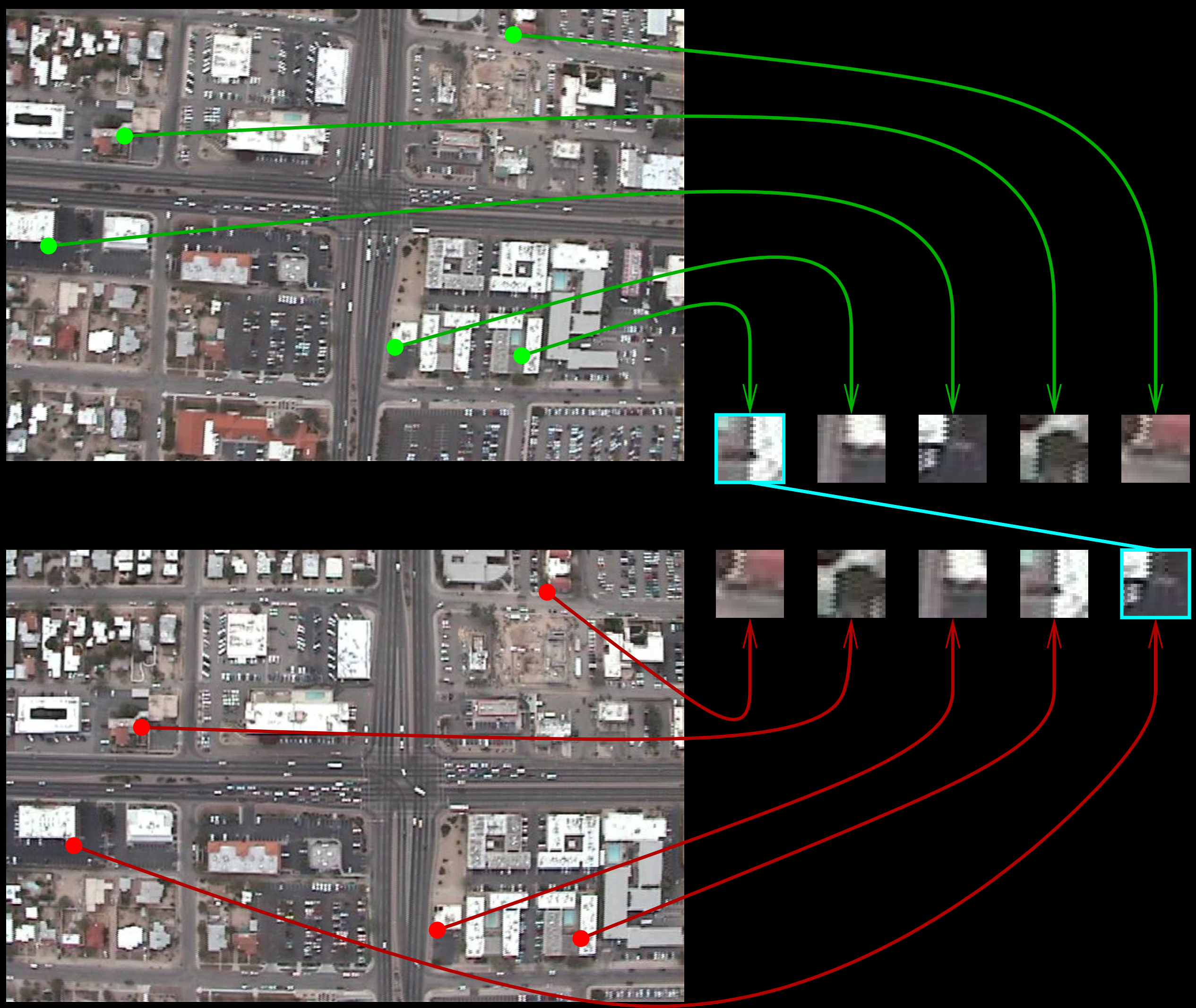
Example: registration



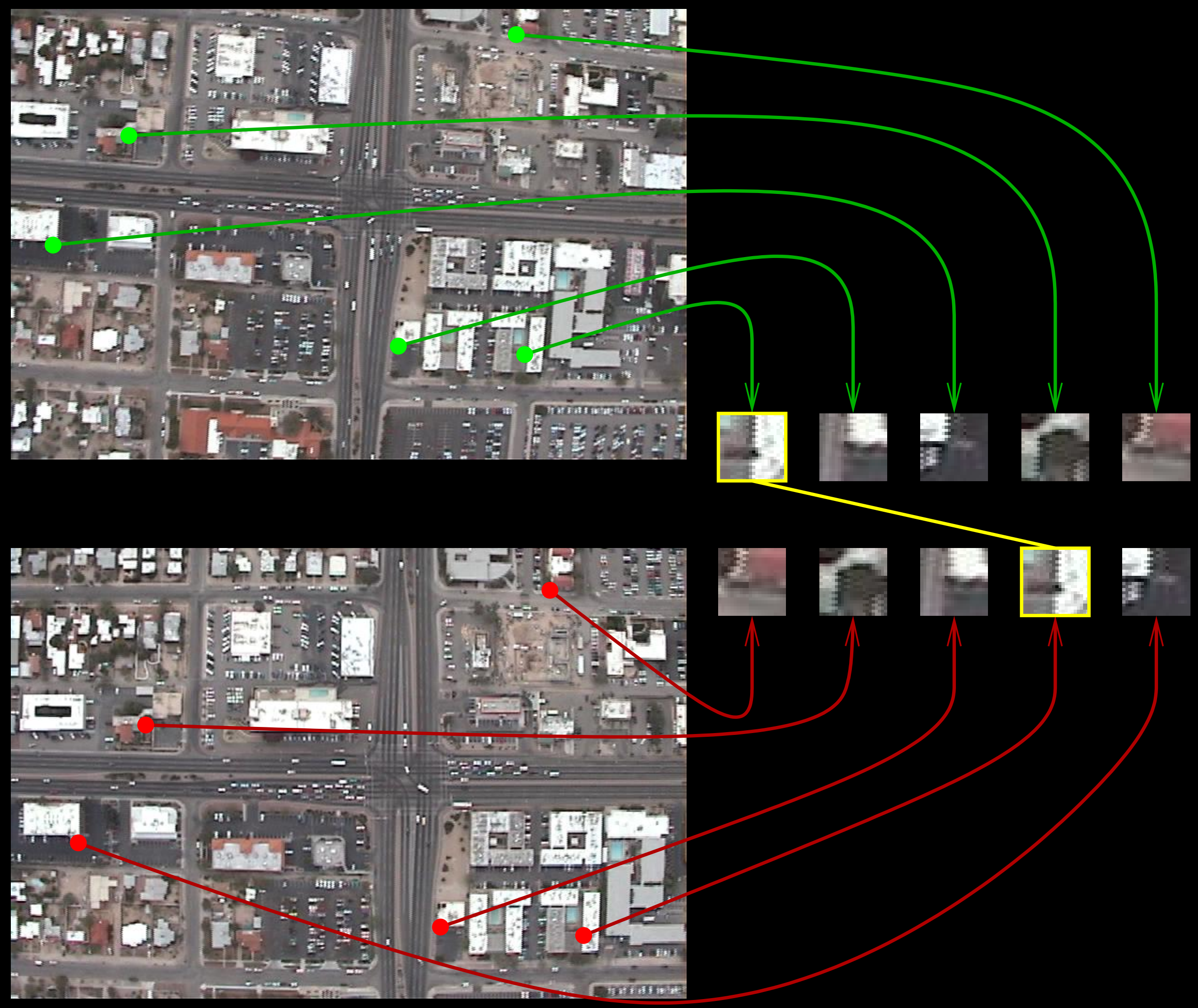
Example: registration



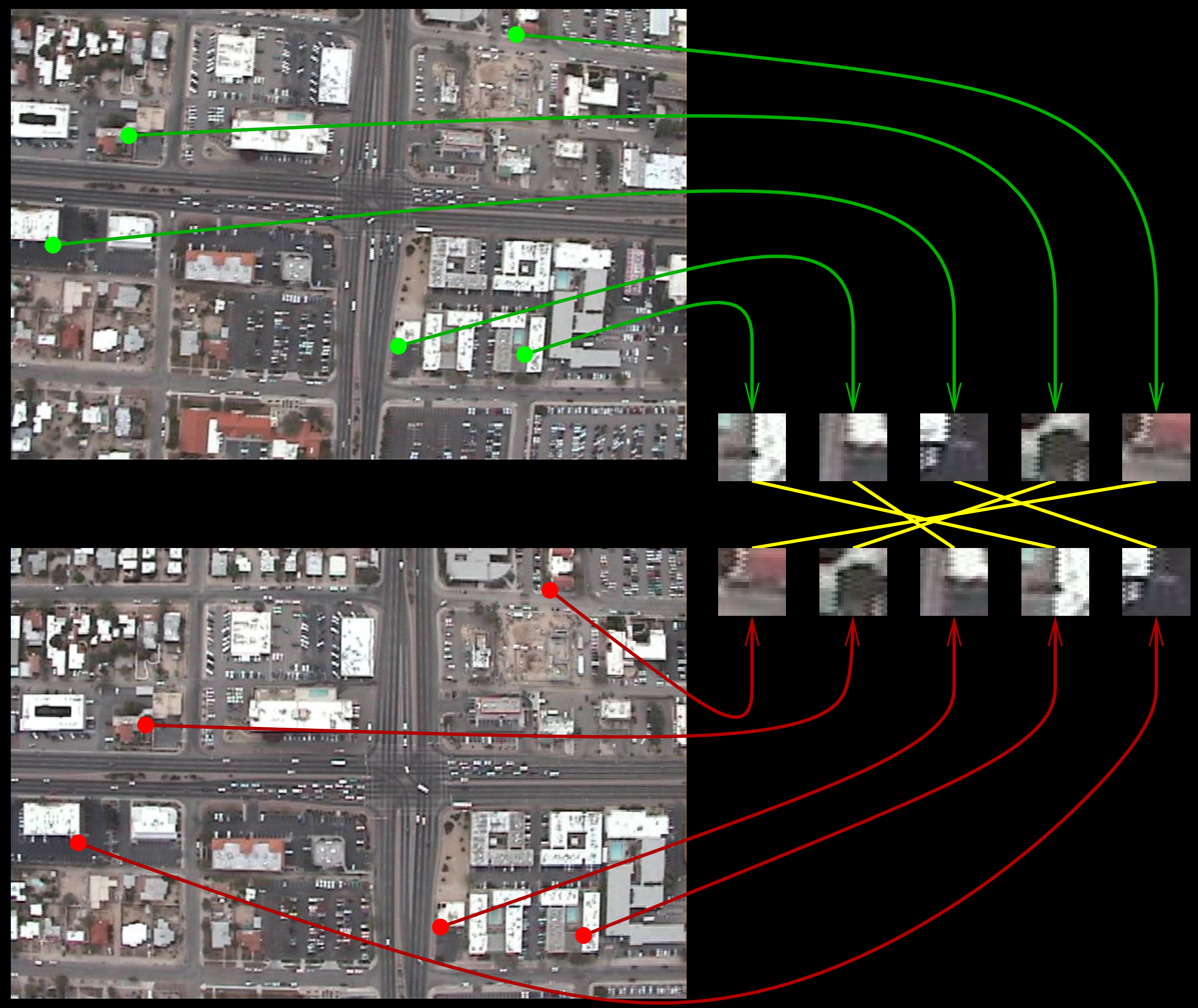
Example: registration



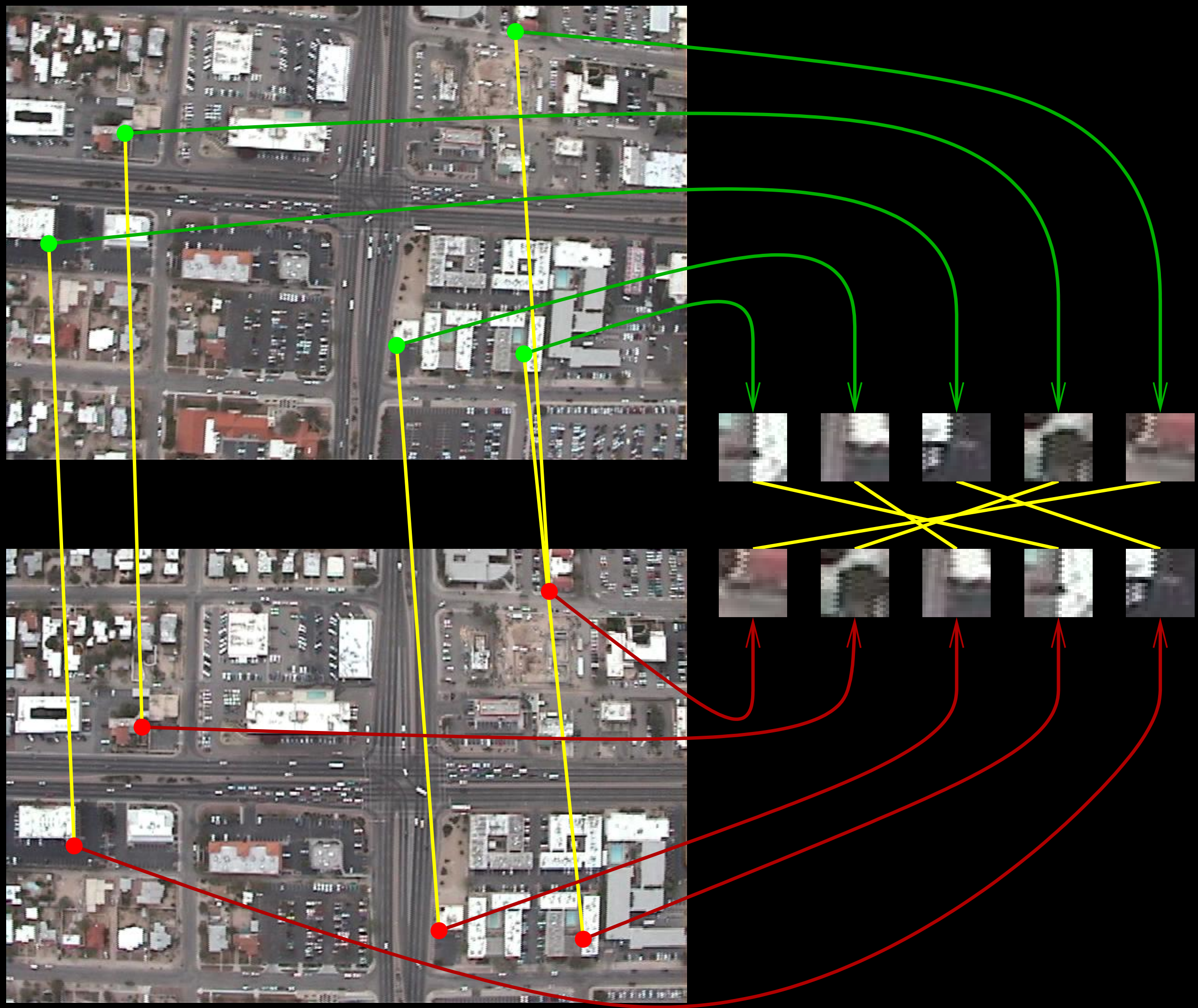
Example: registration



Example: registration



Example: registration



Example: registration

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \\ x_5 & y_5 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \\ u_3 & v_3 \\ u_4 & v_4 \\ u_5 & v_5 \\ \vdots & \vdots \end{bmatrix}$$

Example: registration

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \\ x_5 & y_5 \\ \vdots & \vdots \end{bmatrix} \quad \begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \\ u_3 & v_3 \\ u_4 & v_4 \\ u_5 & v_5 \\ \vdots & \vdots \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} \approx W\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)$$

Typical processing pipeline

Typical processing pipeline

Saliency
function

Typical processing pipeline

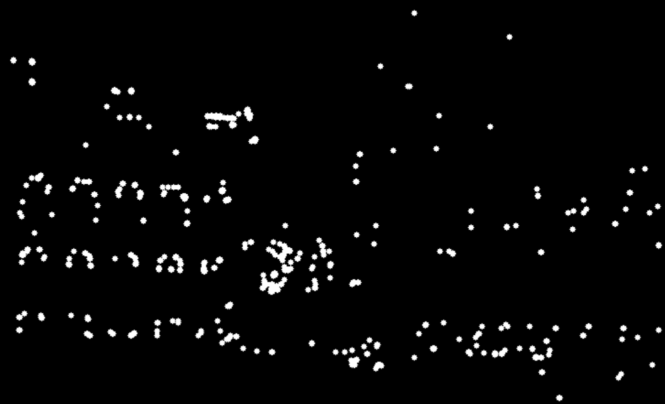
Saliency function →

Threshold ↓

Typical processing pipeline

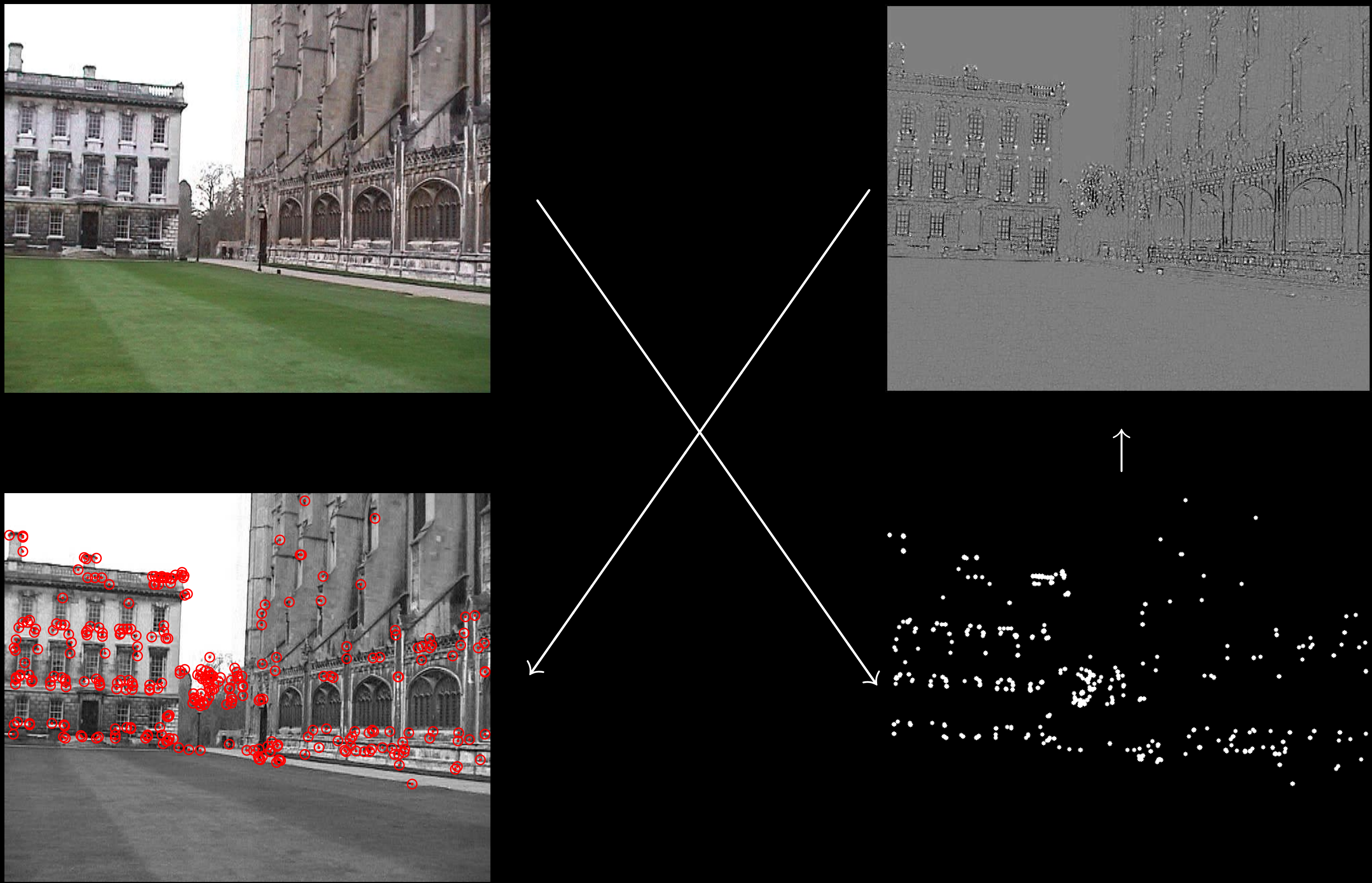
Saliency function →

Threshold ↓

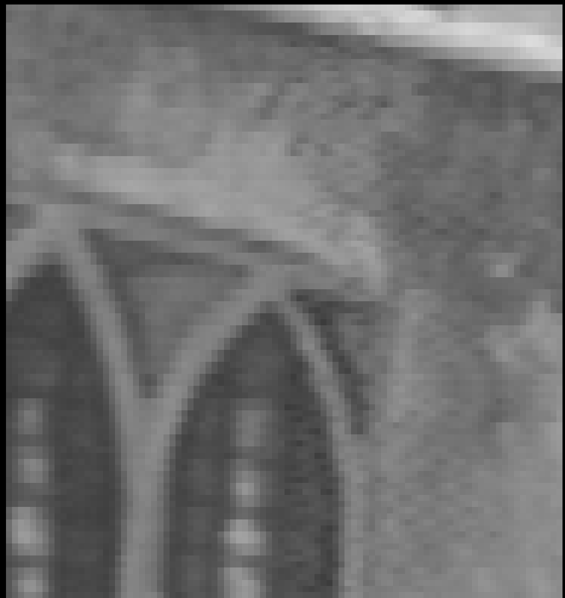


Local maxima ←

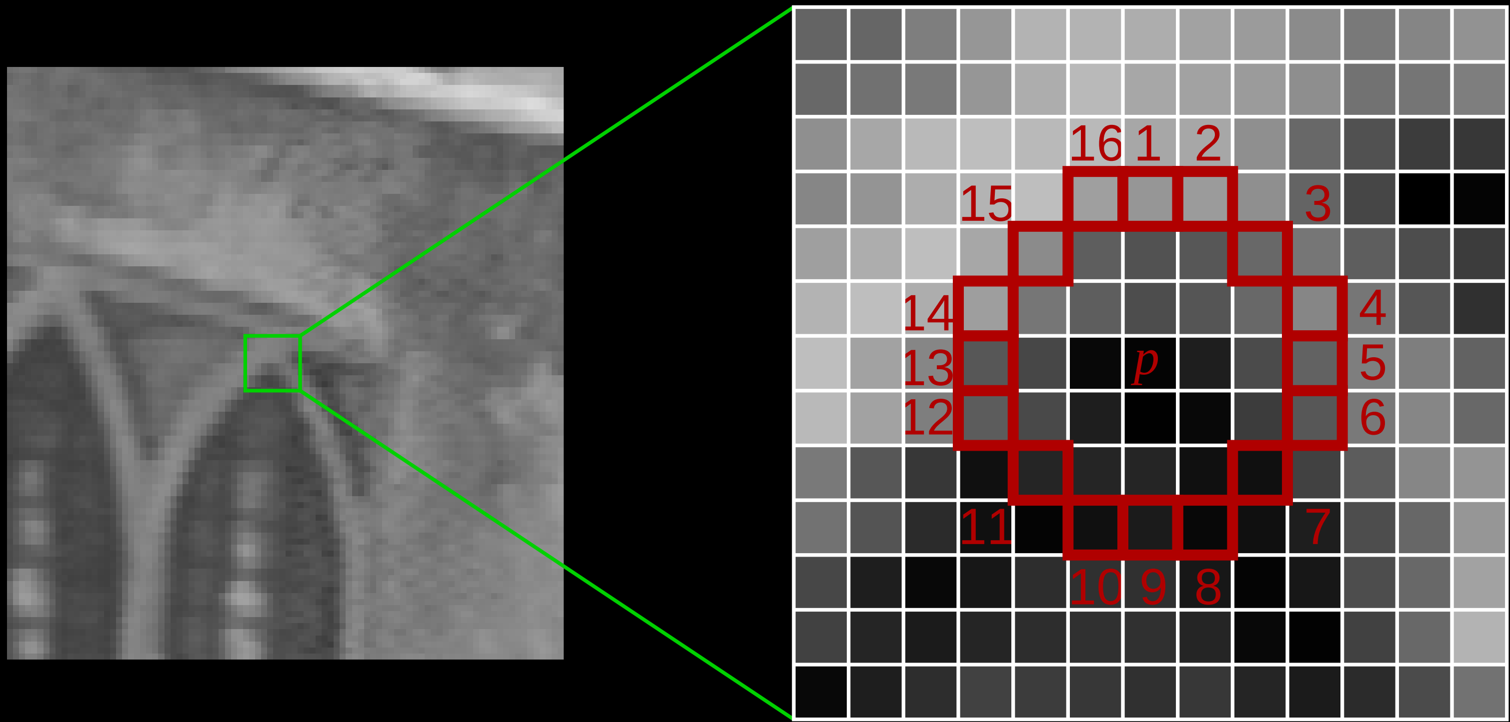
Typical processing pipeline



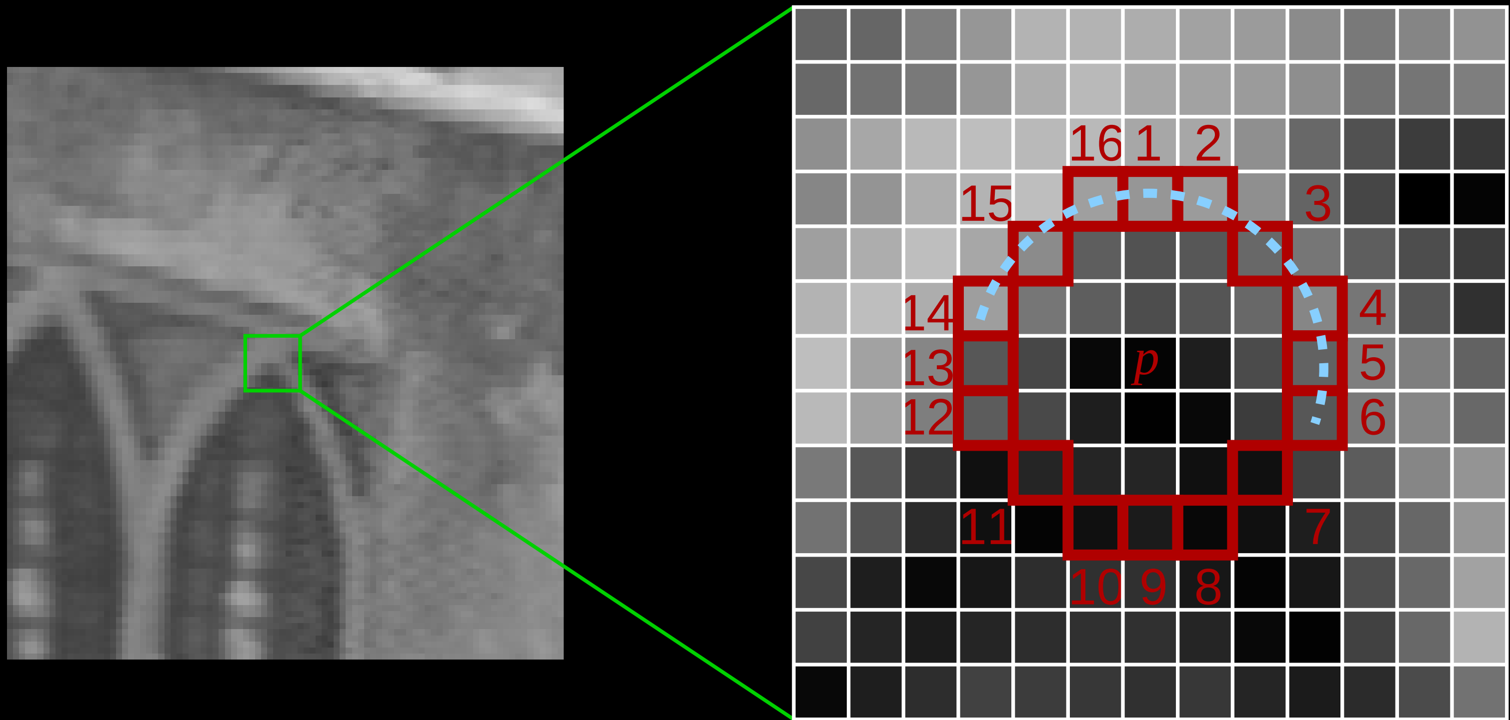
The segment-test detector



The segment-test detector



The segment-test detector



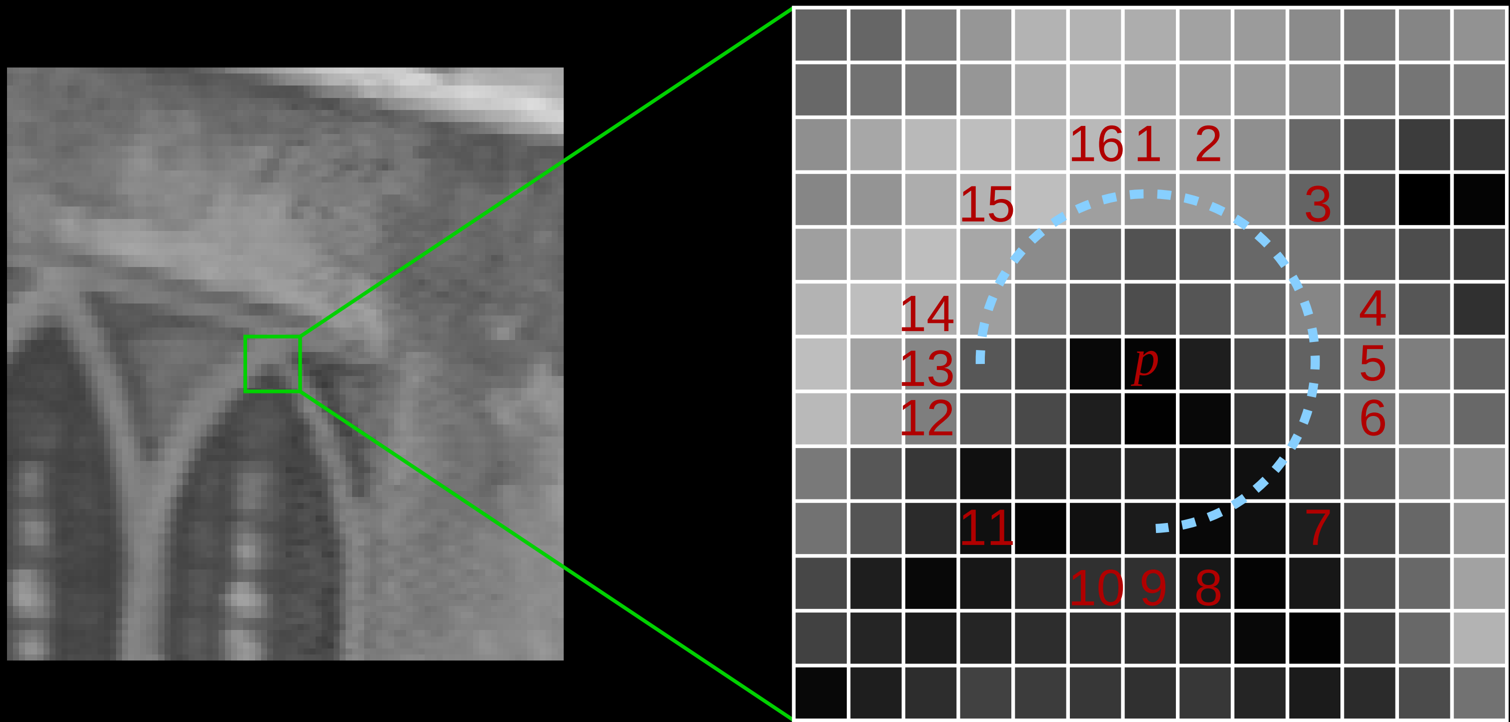
Contiguous arc of N or more pixels:

- All much brighter than p (brighter than $p + t$).

or

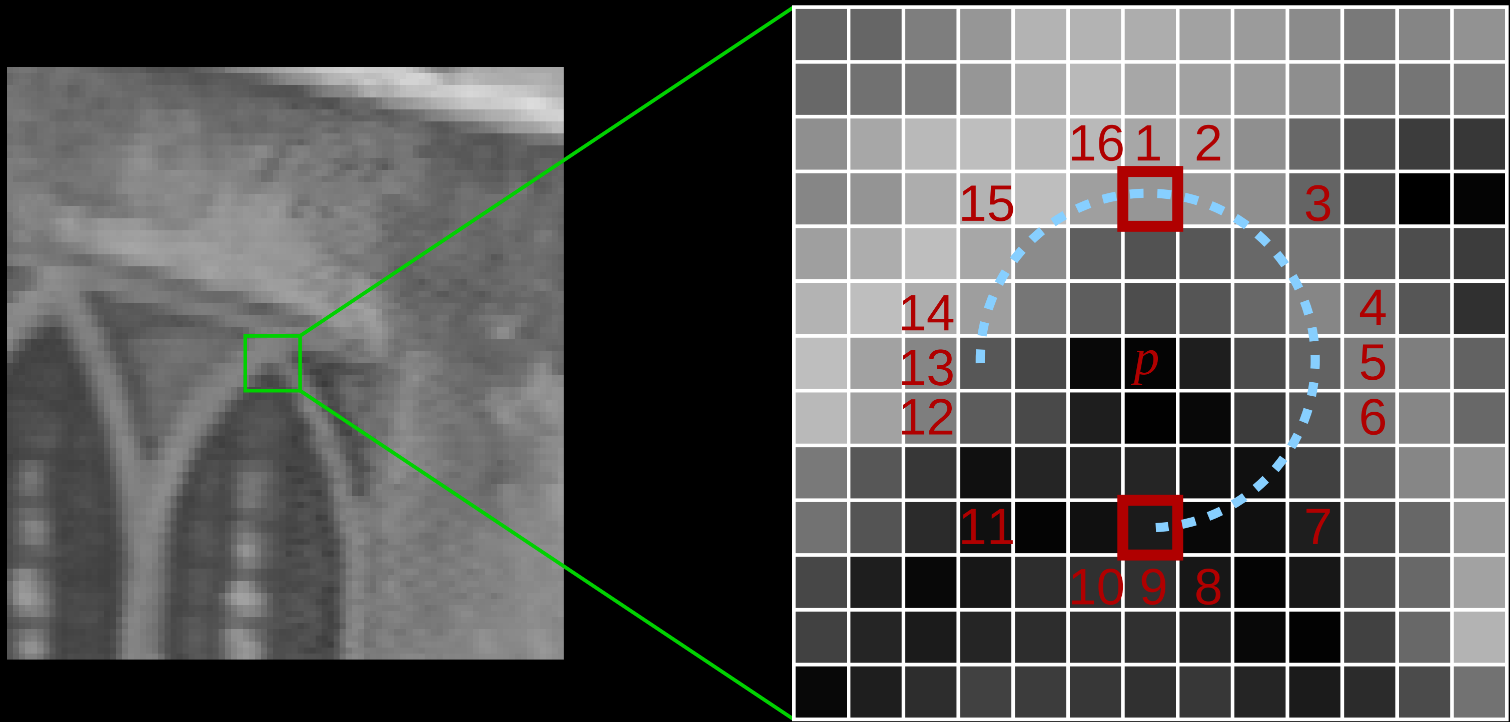
- All much darker than p (darker than $p - t$).

The FAST detector (version 1)



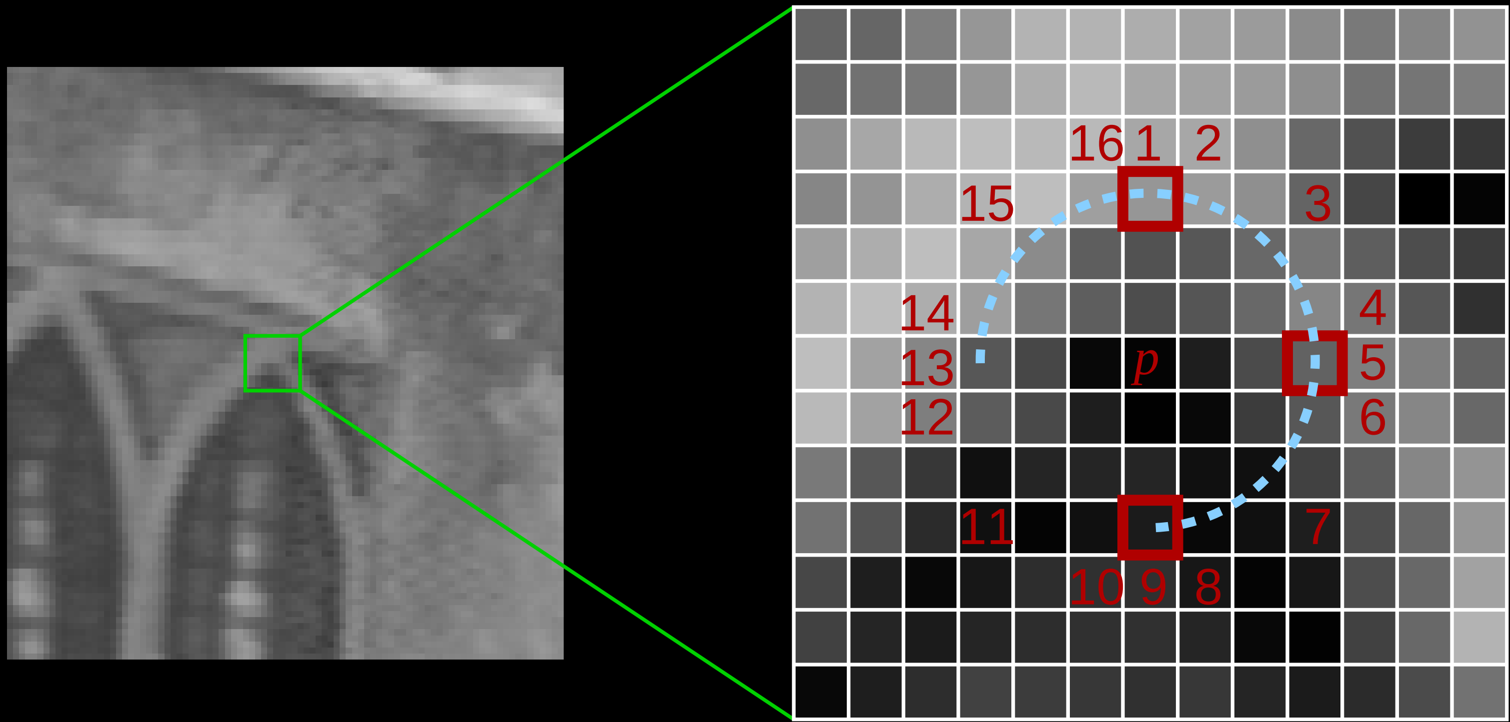
FAST—Features from Accelerated Segment Test

The FAST detector (version 1)



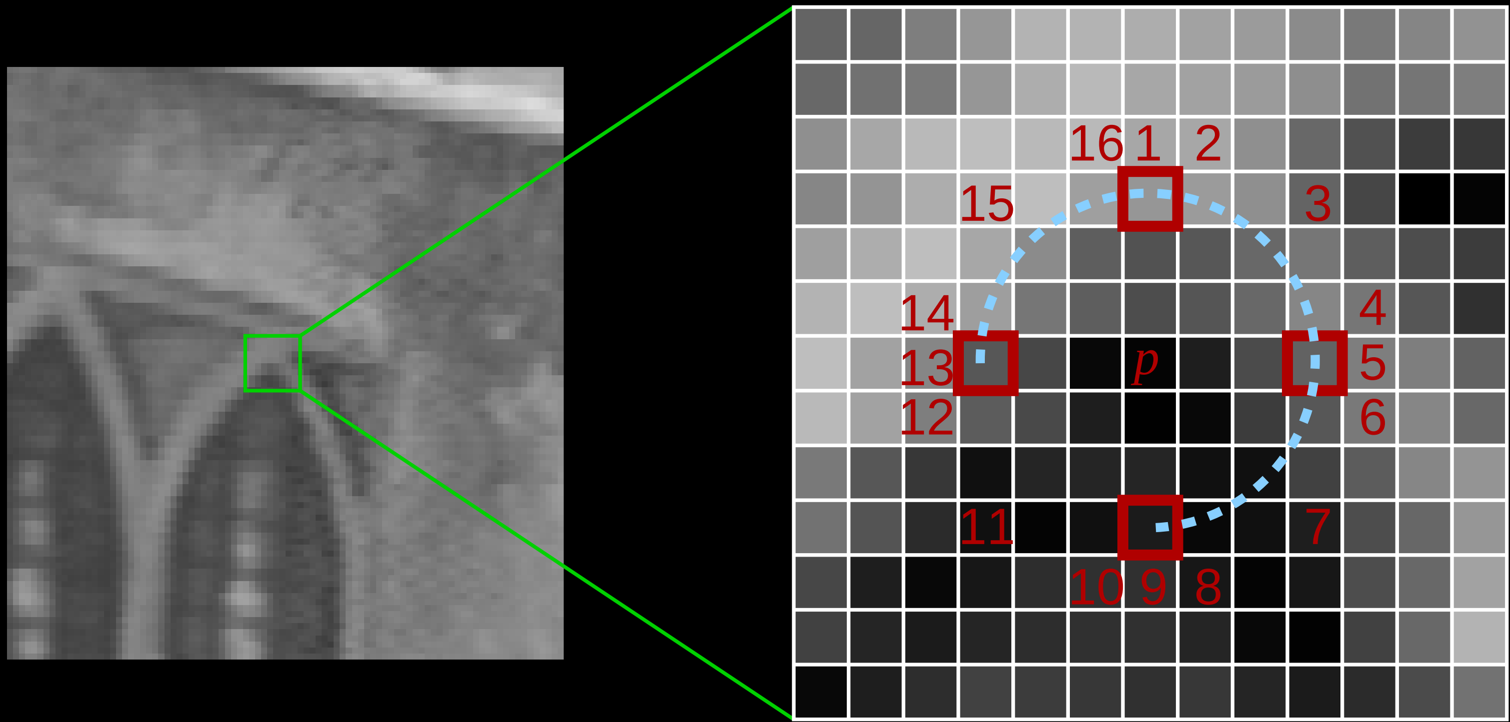
- Test pixels 1 and 9

The FAST detector (version 1)



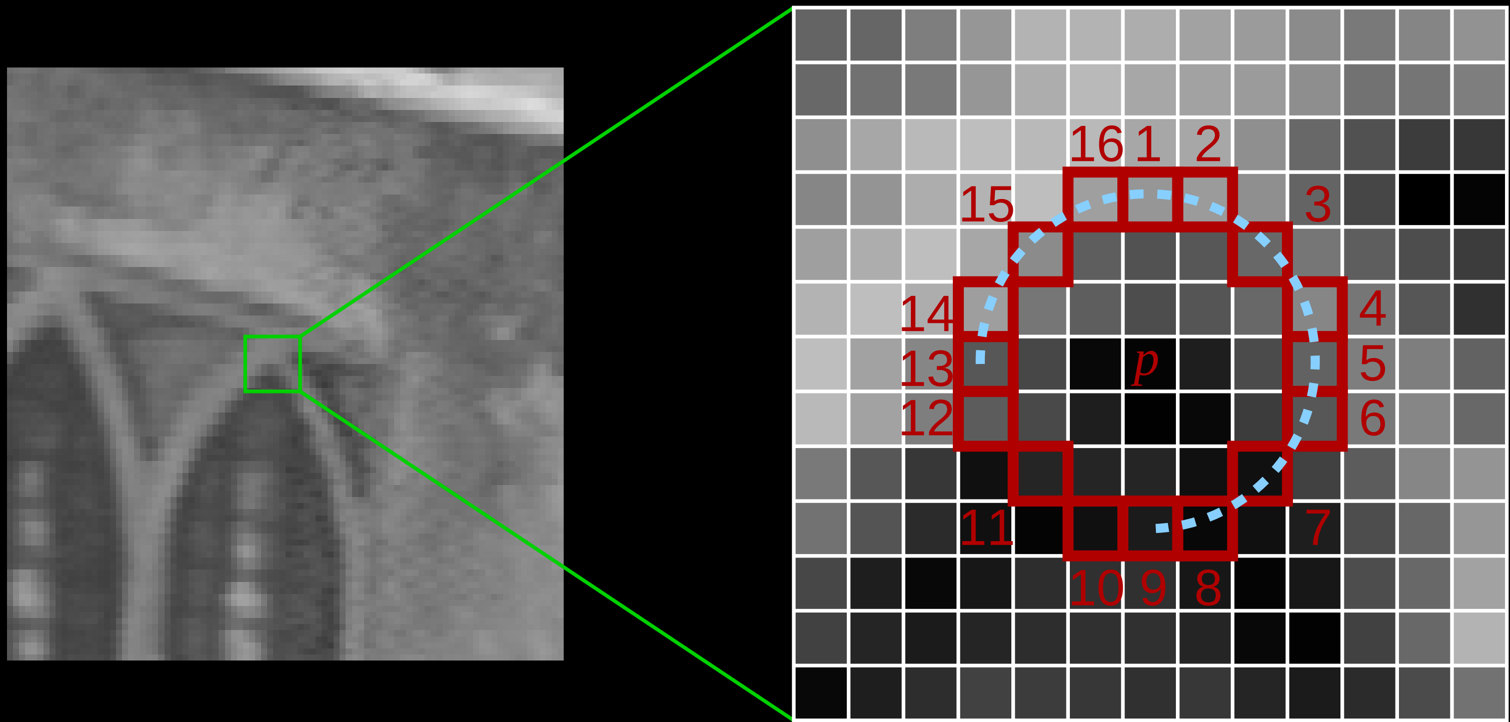
- Test pixels 1 and 9
- Test pixel 4

The FAST detector (version 1)



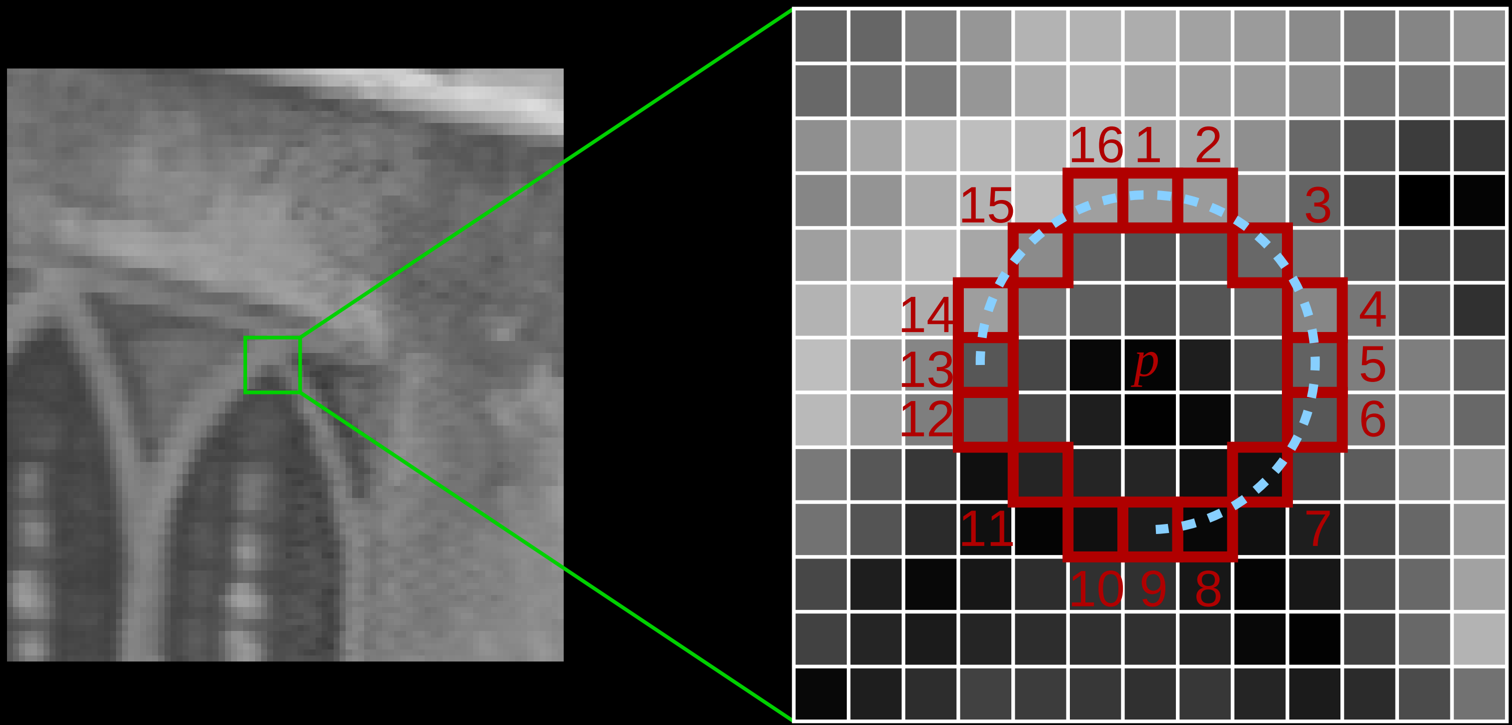
- Test pixels 1 and 9
- Test pixel 4
- Test pixel 12

The FAST detector (version 1)



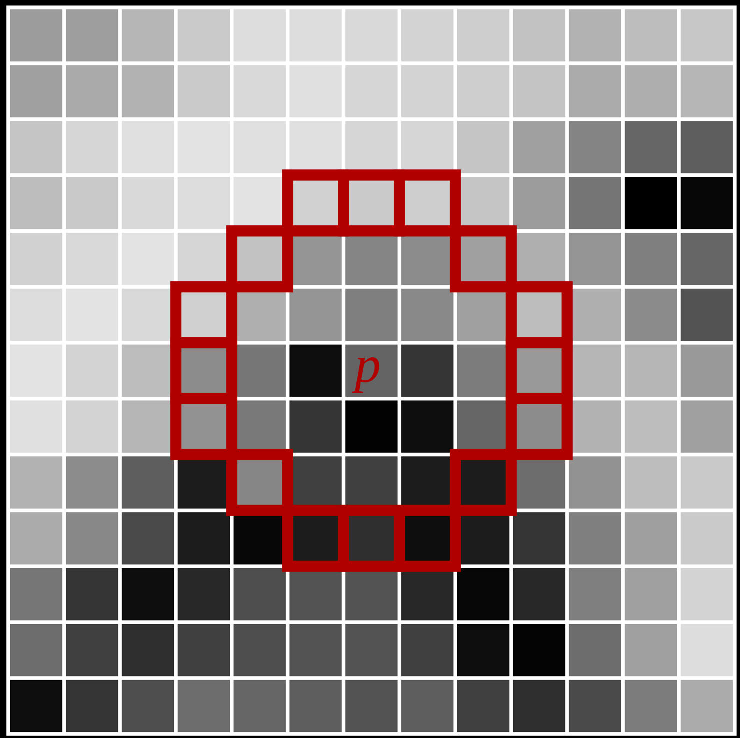
- Test pixels 1 and 9
- Test pixel 4
- Test pixel 12
- Perform complete segment test

FAST saliency

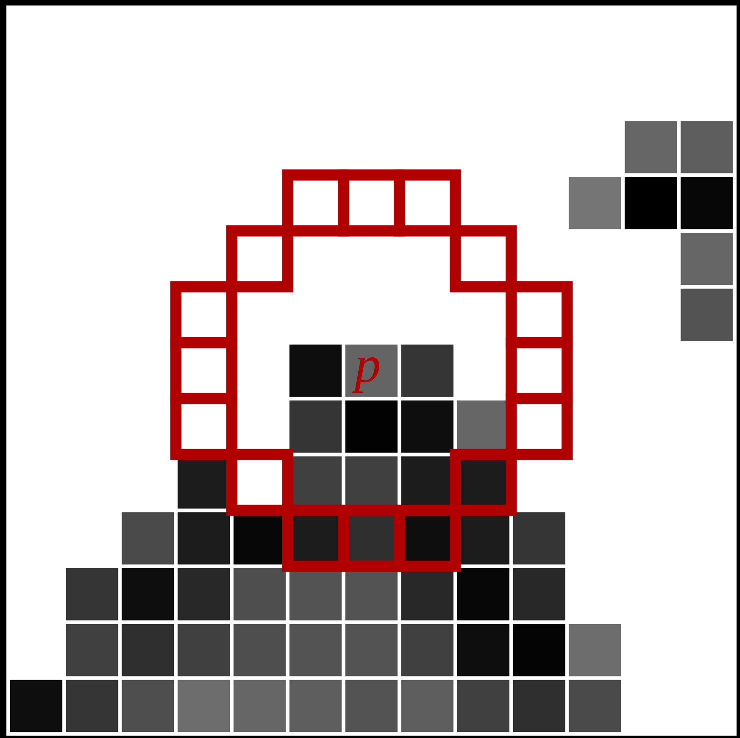


- Highest t for which point is a corner.
- Find using bisection over t.
  - 8 iterations required.
  - Very small subset of points.

FAST feature detection (version 2)

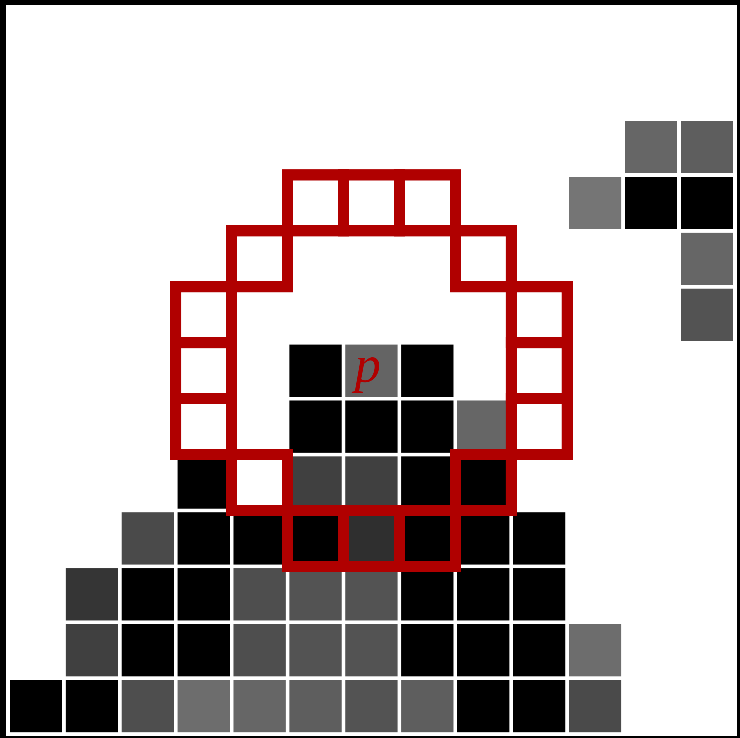


FAST feature detection (version 2)



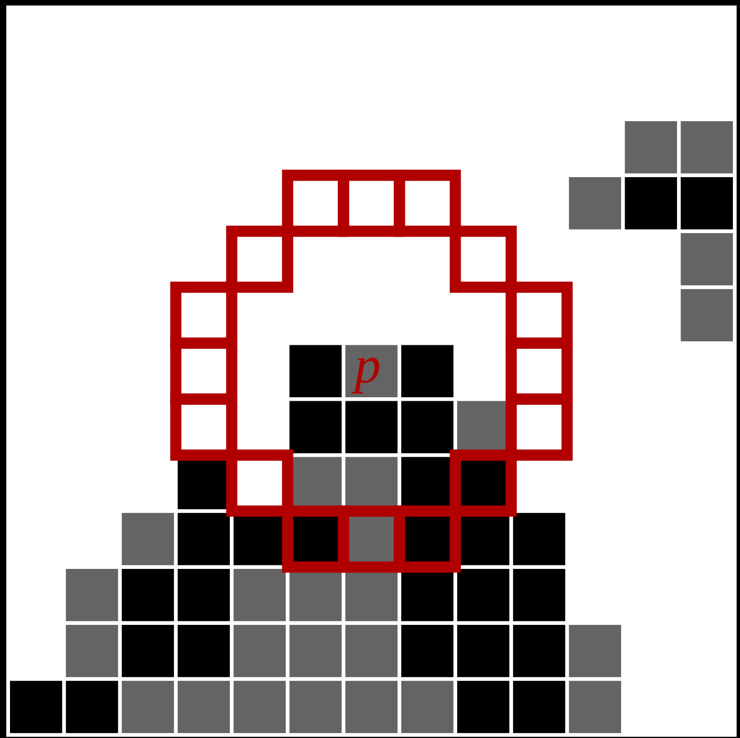
- Pixels are either:
  - Much brighter.

FAST feature detection (version 2)



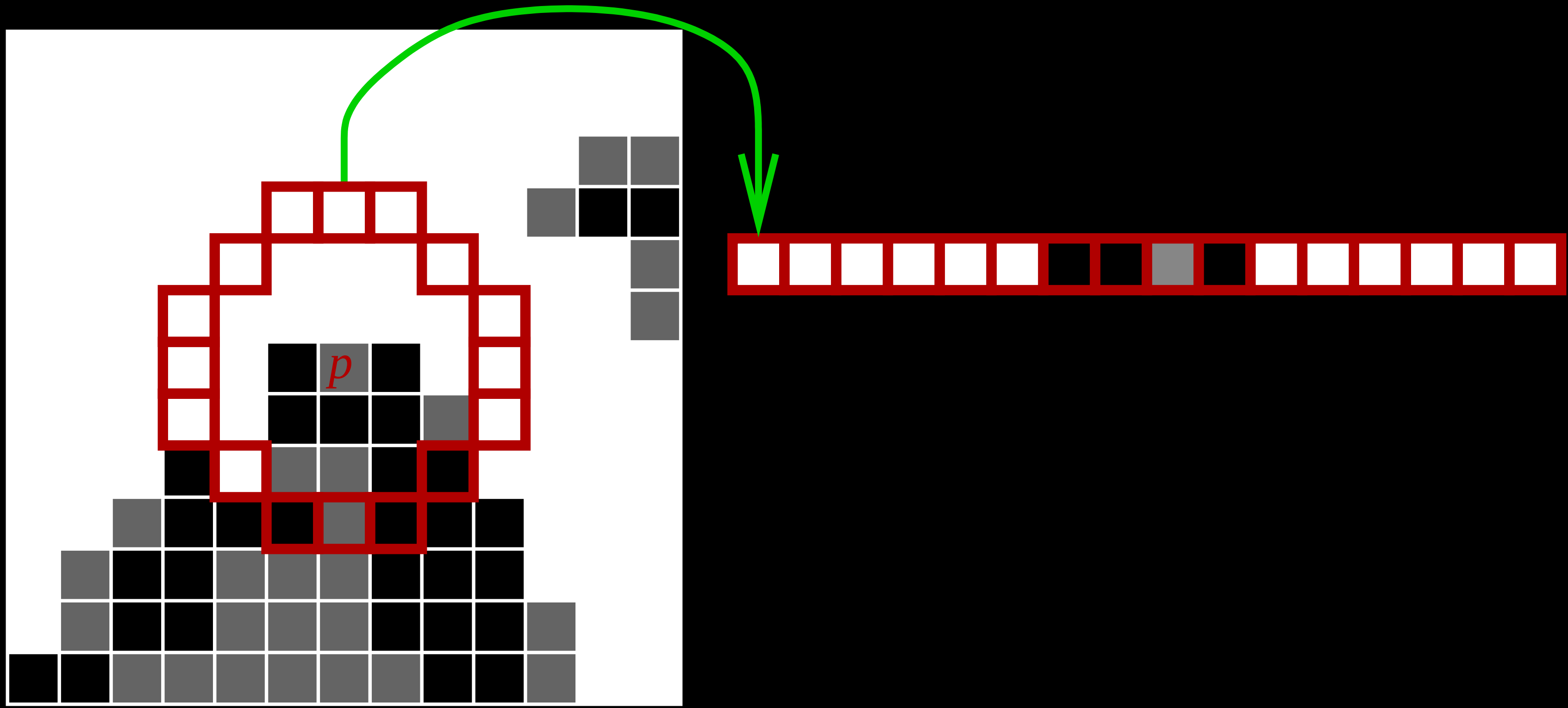
- Pixels are either:
  - Much brighter.
  - Much darker.

FAST feature detection (version 2)



- Pixels are either:
  - Much brighter.
  - Much darker.
  - Similar.

FAST feature detection (version 2)



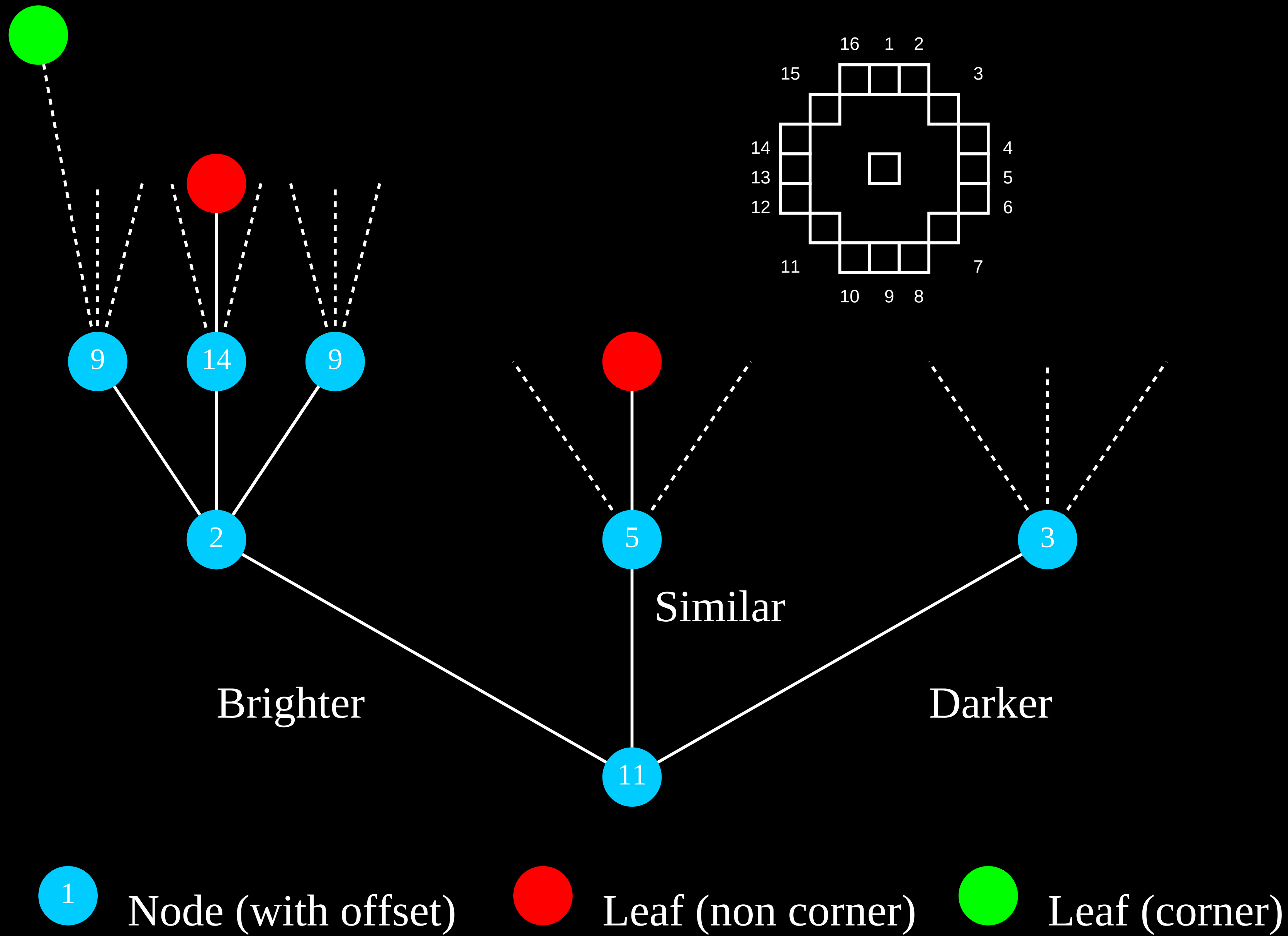
- Pixels are either:
  - Much brighter.
  - Much darker.
  - Similar.
- Represent ring as a ternary vector.
- Classify vectors using segment test.

Train a classifier

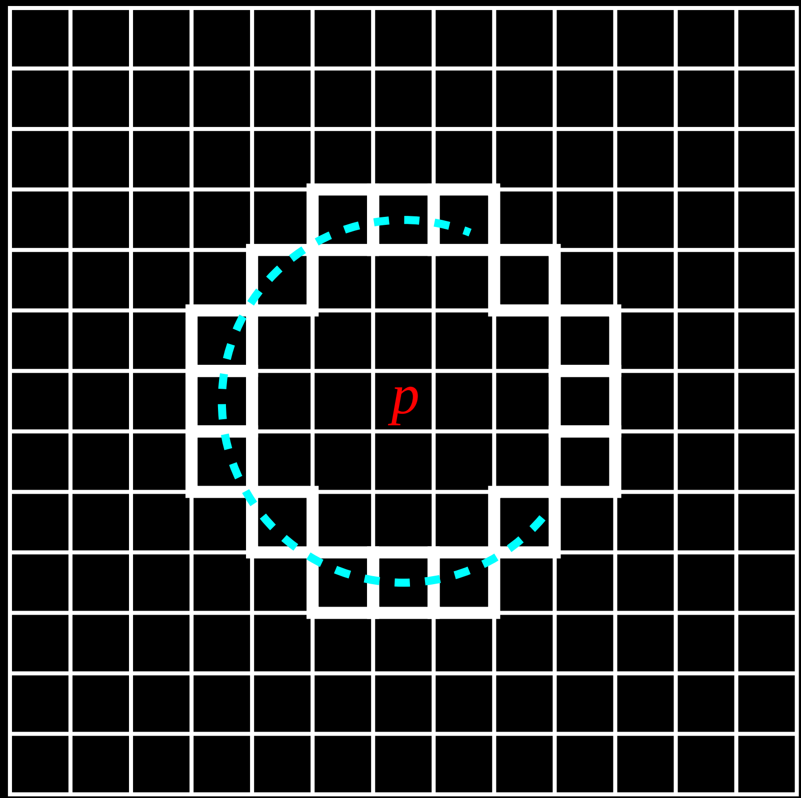
- Decision tree classifiers are very efficient.
- Ask: “What is the state of pixel x?”
- Question splits list in to 3 sublists.
- Query each sublist.
- Recurse until list contains all features or all non features.
- Choose questions to minimize entropy (ID3).

- Use questions on new feature.
- Works for *any* N.

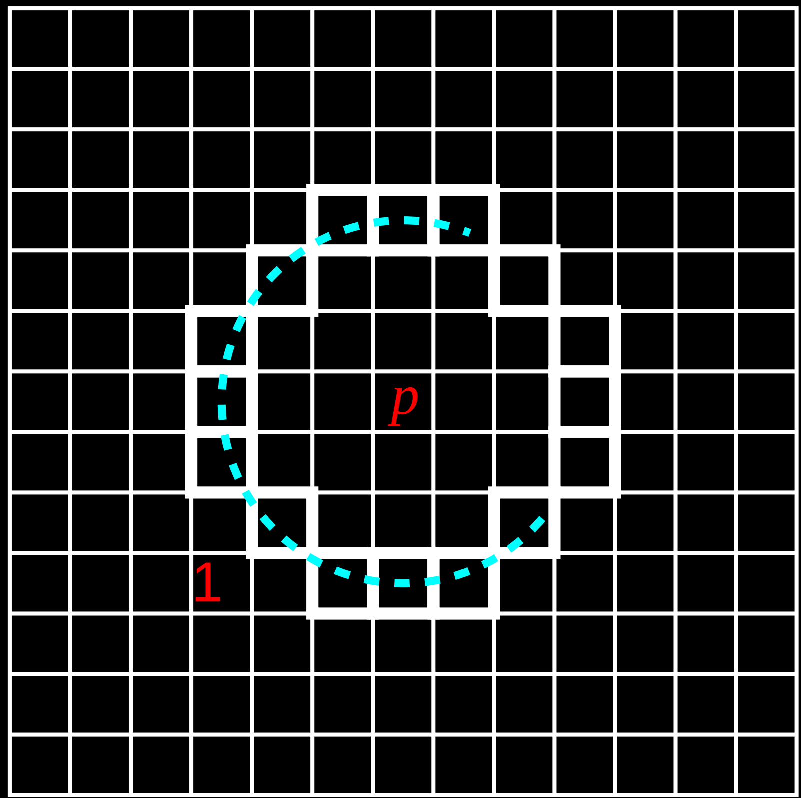
Example tree



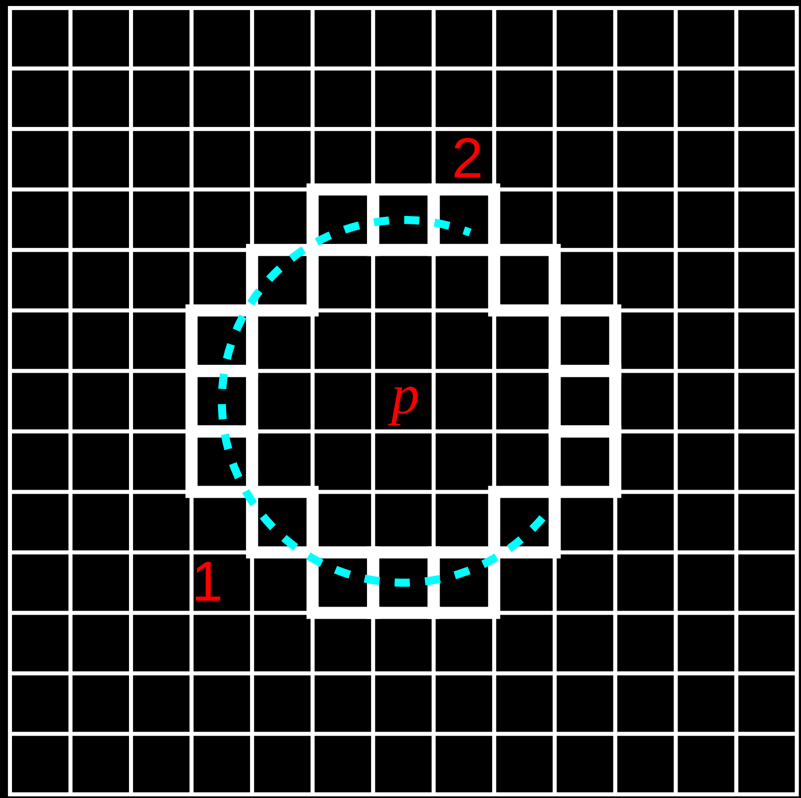
Example tree



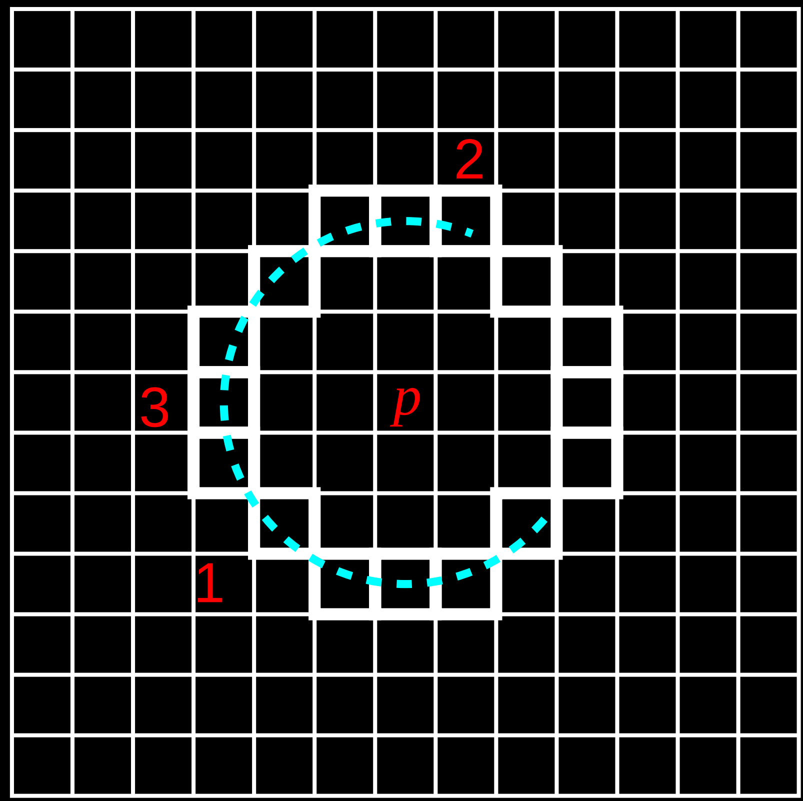
Example tree



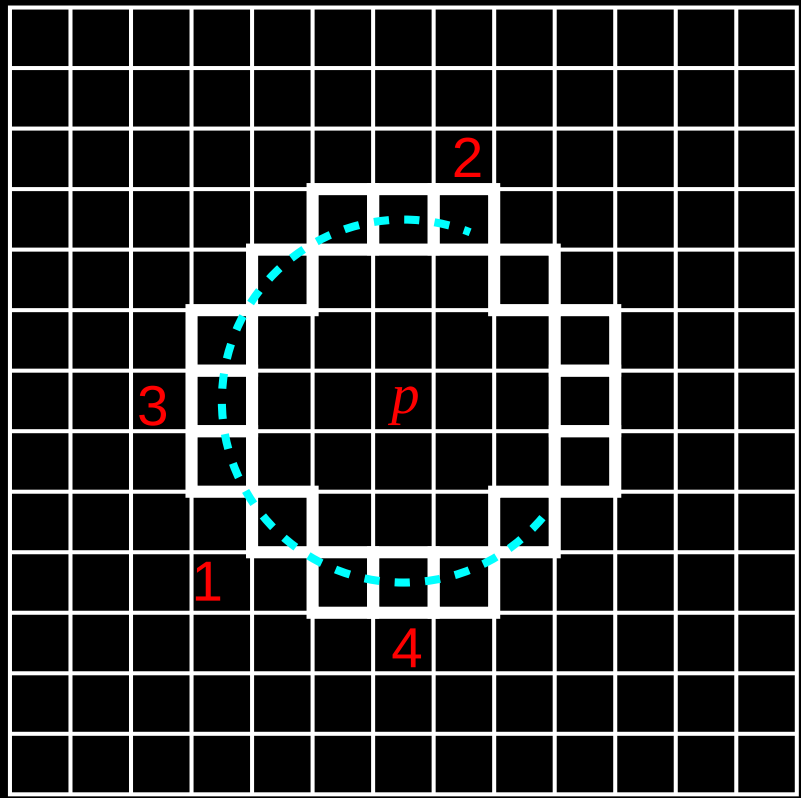
Example tree



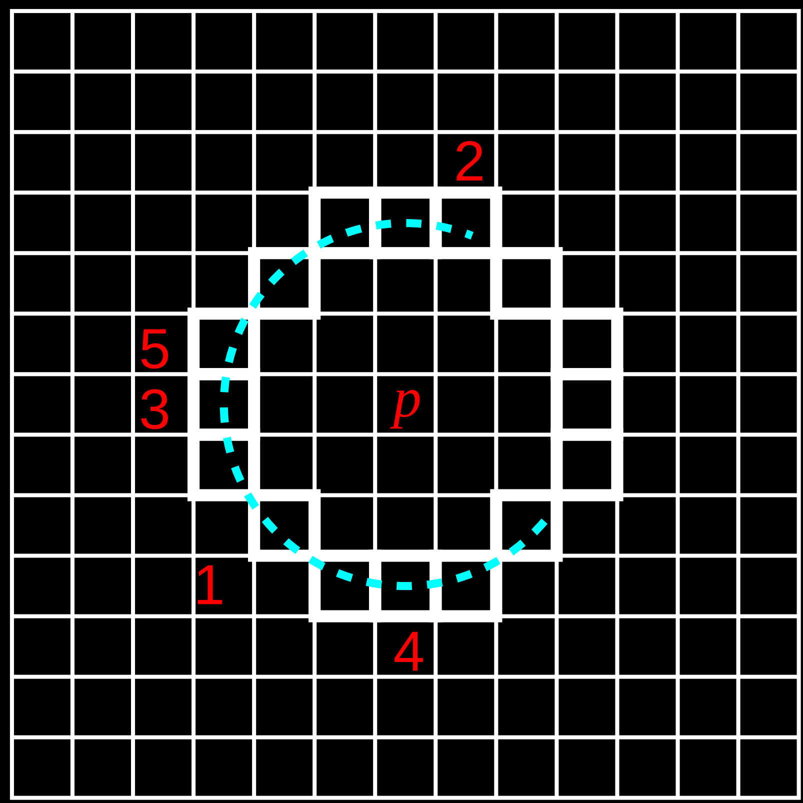
Example tree



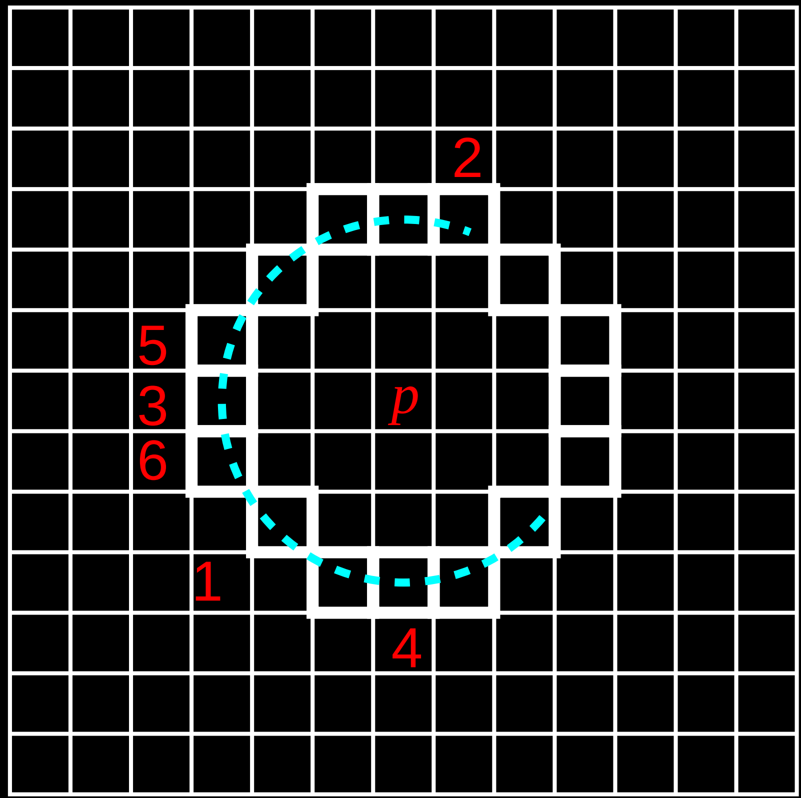
Example tree



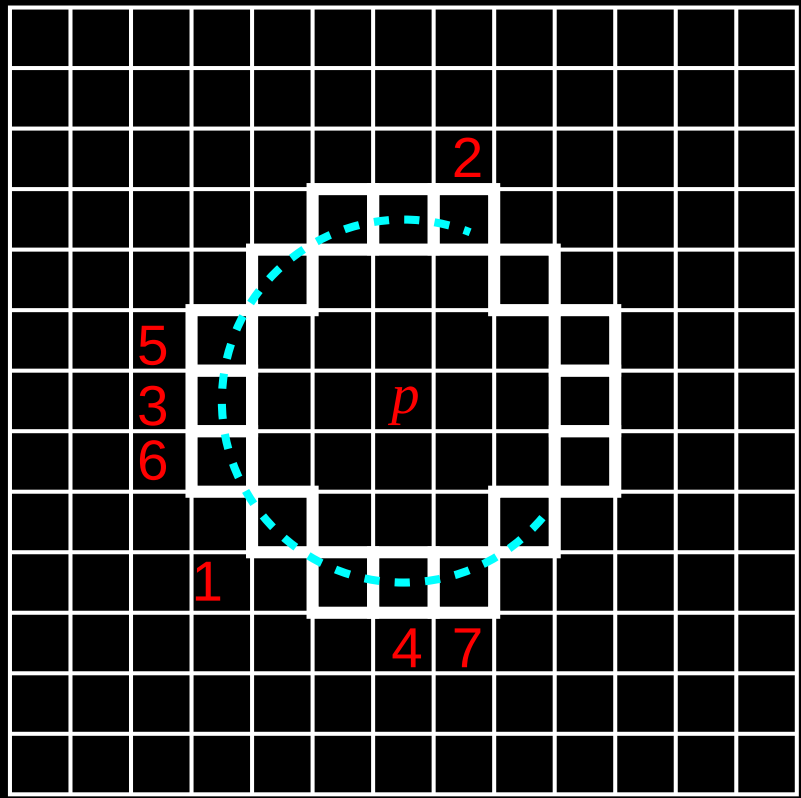
Example tree



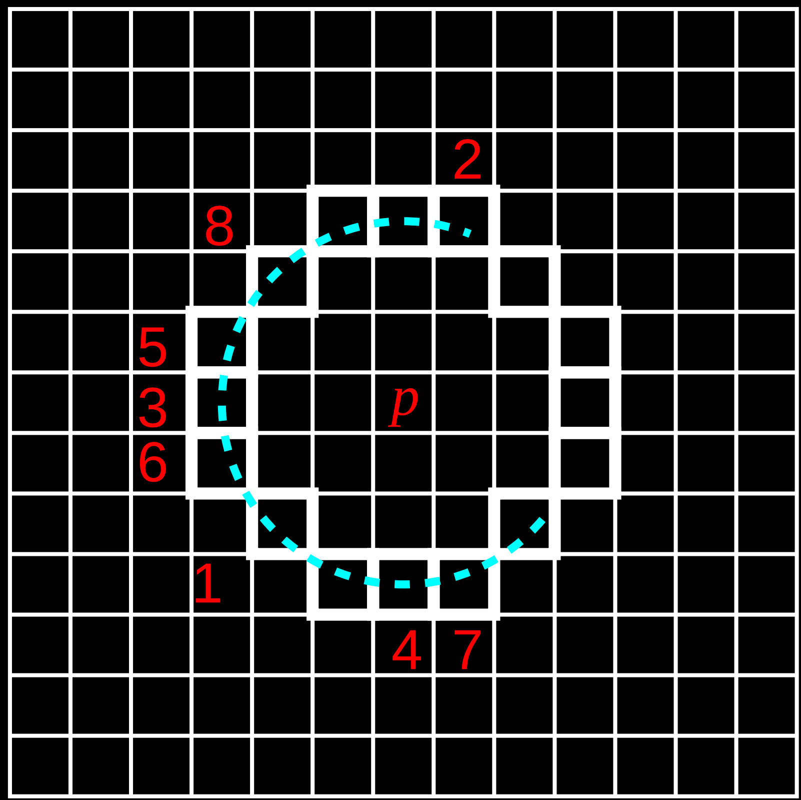
Example tree



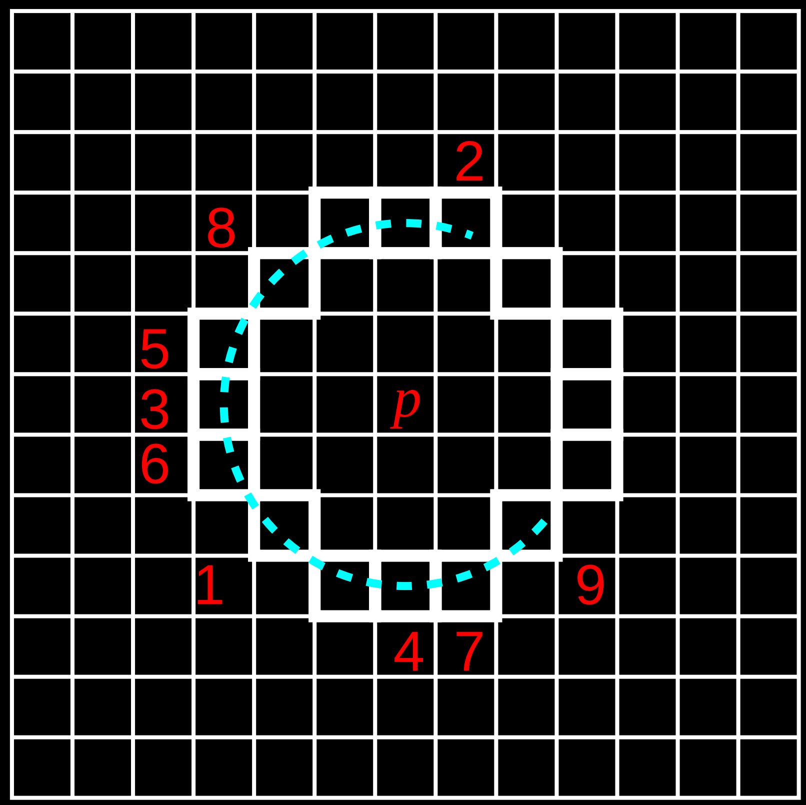
Example tree



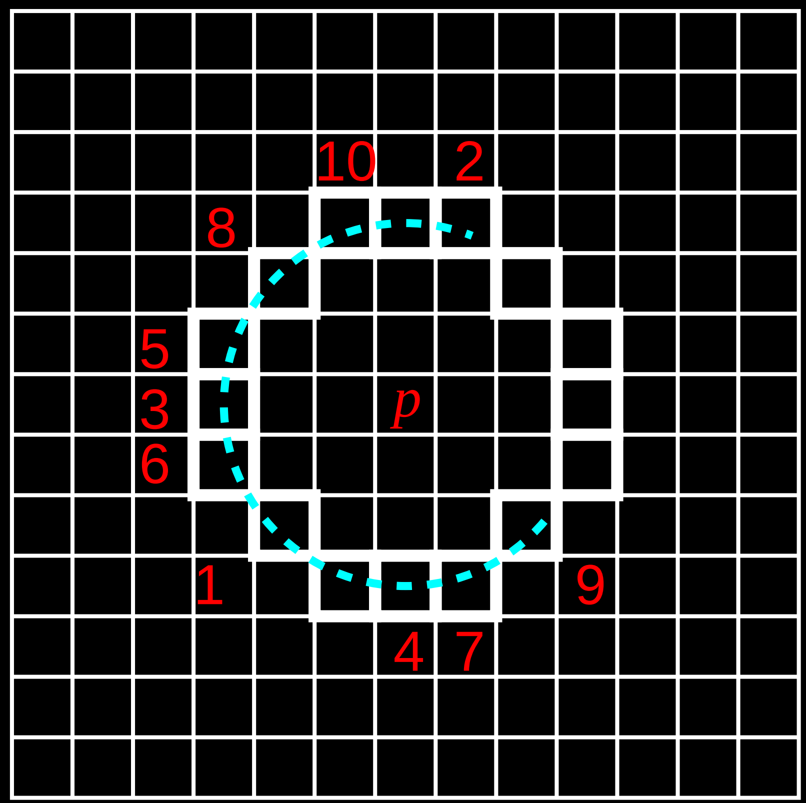
Example tree



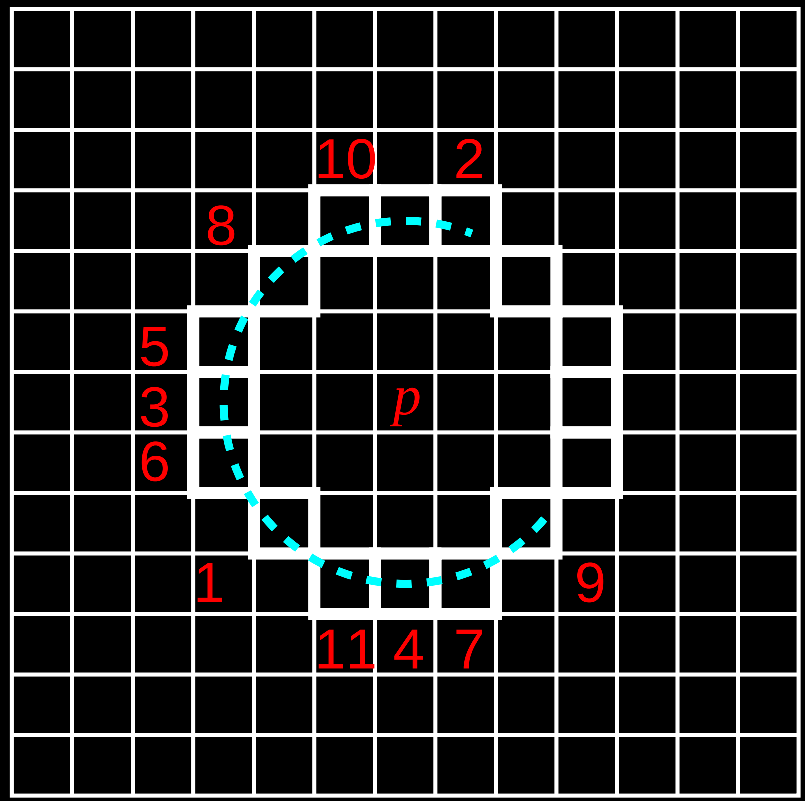
Example tree



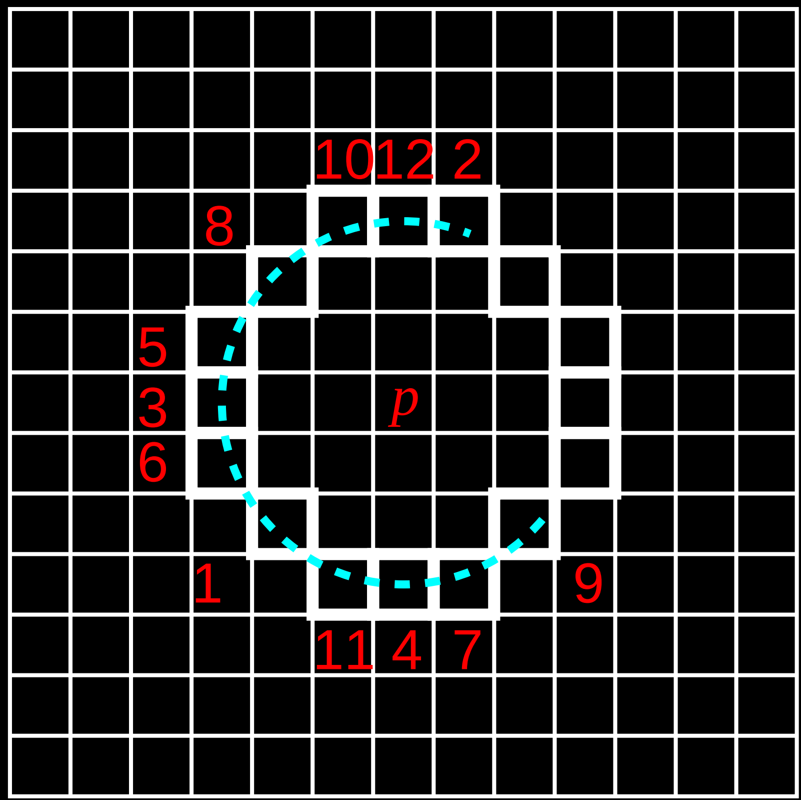
Example tree



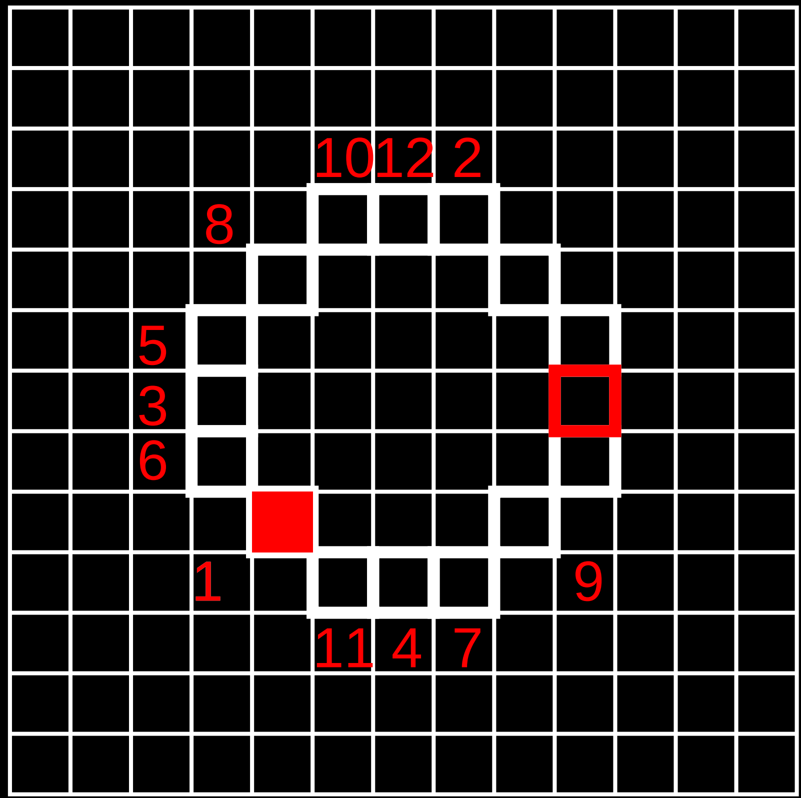
Example tree



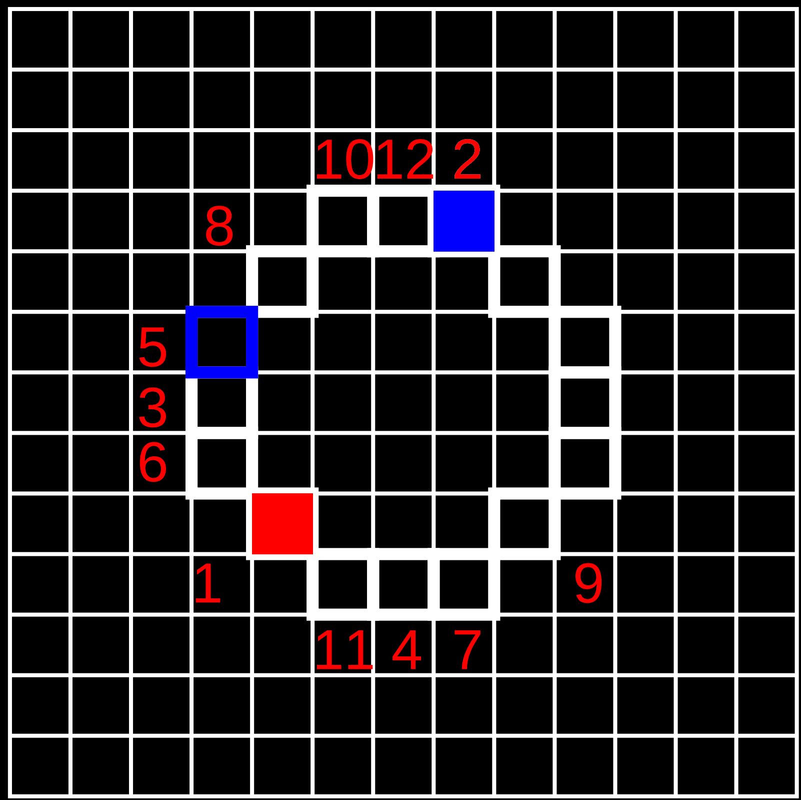
Example tree



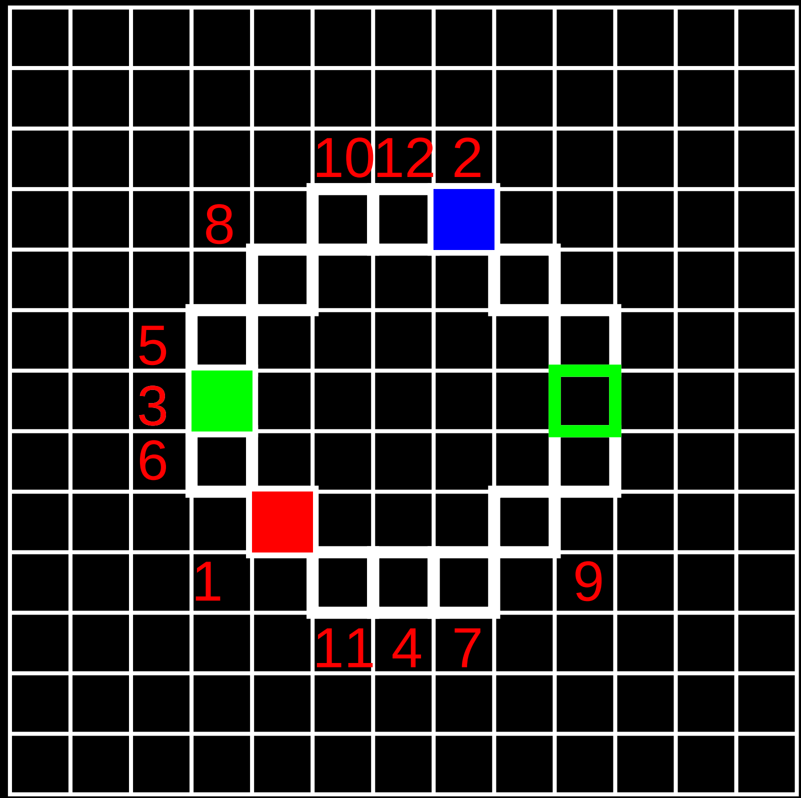
Example tree



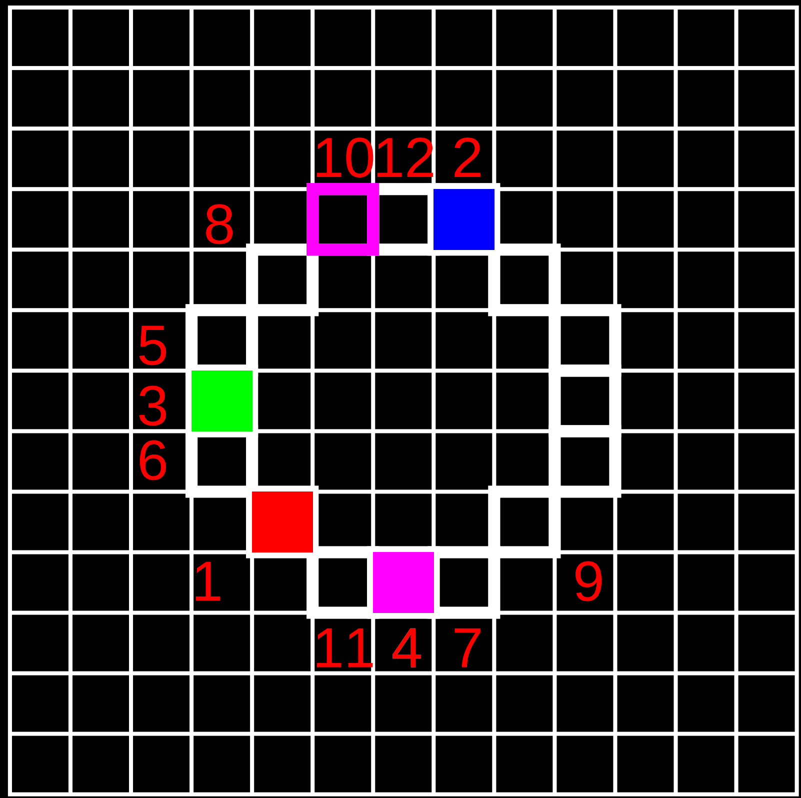
Example tree



Example tree

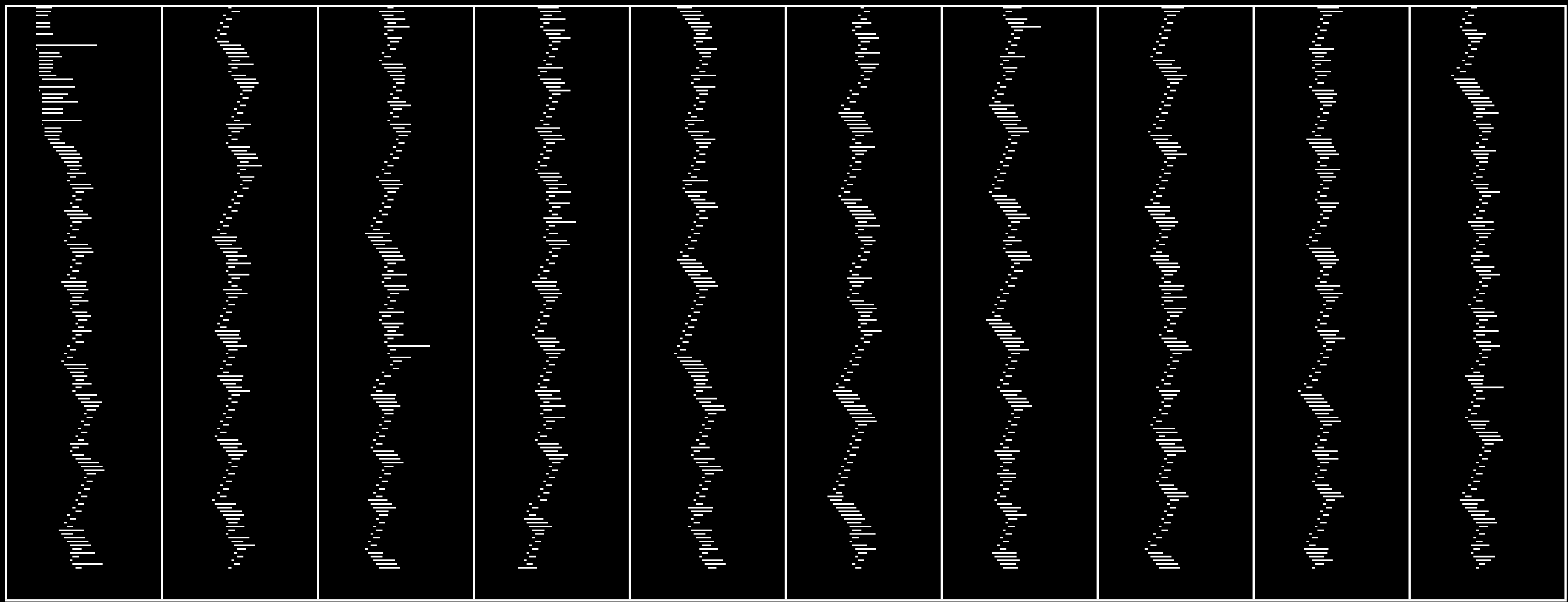


Example tree



Output C++ code

A long string of nested if-else statements:



…which continues for 2 more pages.

How FAST? (very)

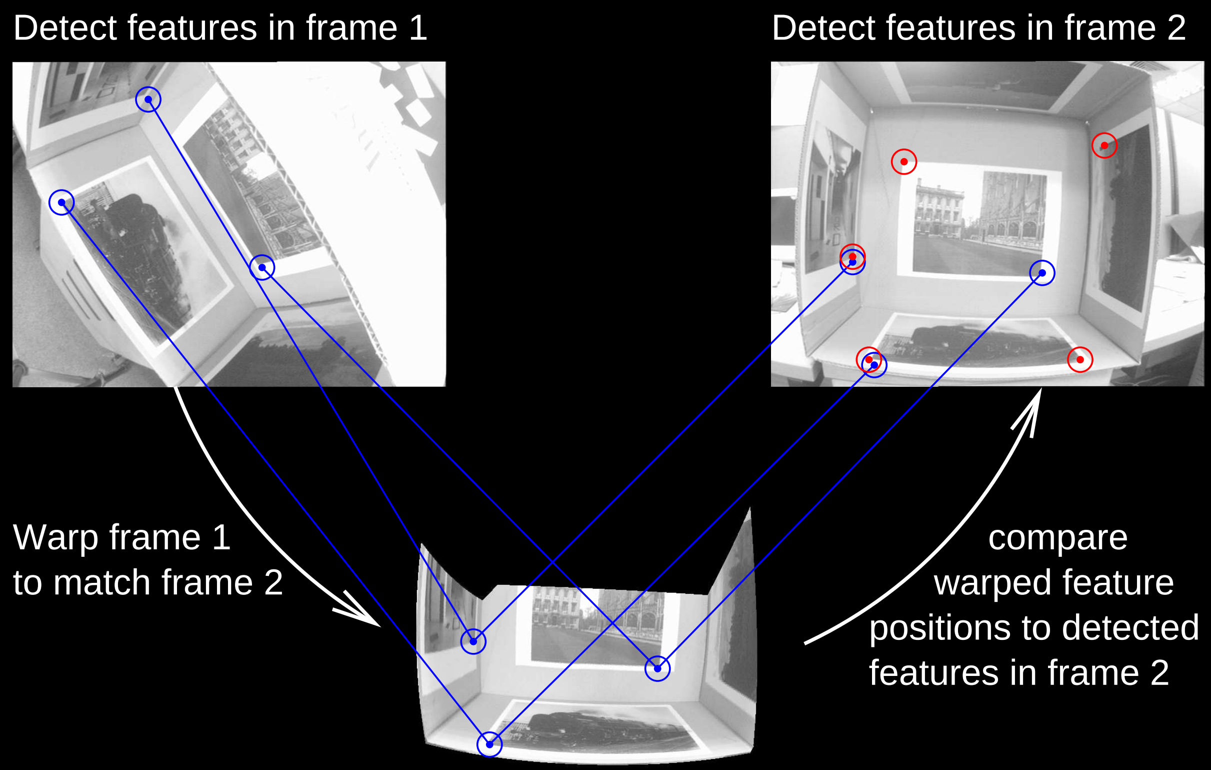
Detector	Set 1		Set 2	
	Pixel rate (MPix/s)	%	MPix/s	%
FAST $n = 9$	188	4.90	179	5.15
FAST $n = 12$	158	5.88	154	5.98
Original FAST ($n = 12$)	79.0	11.7	82.2	11.2
SUSAN	12.3	74.7	13.6	67.9
Harris	8.05	115	7.90	117
Shi-Tomasi	6.50	142	6.50	142
DoG	4.72	195	5.10	179

- 3.0GHz Pentium 4
- Set 1: 992×668 pixels.
- set 2: 352×288 (quarter-PAL) video.
- Percentage budget for PAL, NTSC, DV, 30Hz VGA.

Is it any good?

Repeatability

Is the same real-world 3D point detected from multiple views?



Repeat for all pairs in a sequence

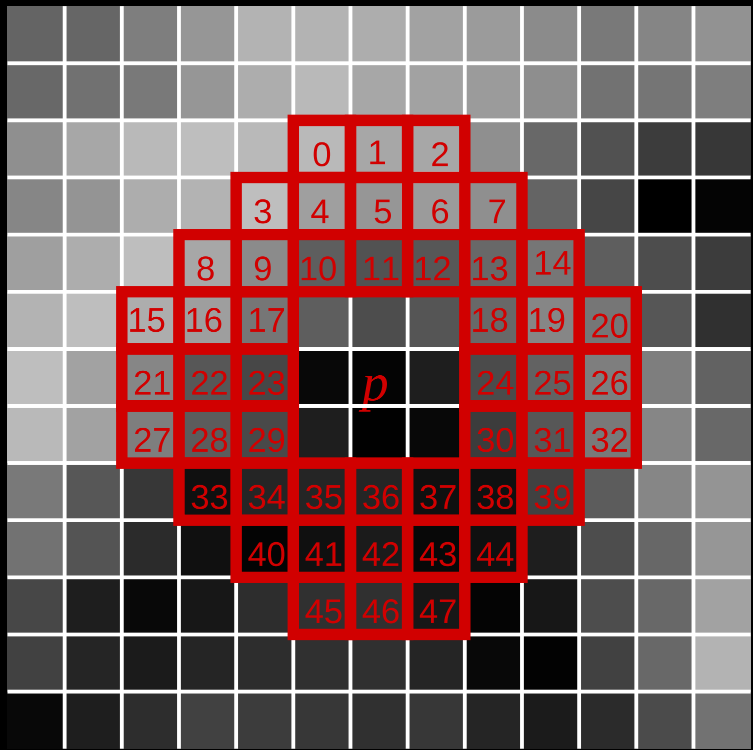
FAST-ER: Enhanced Repeatability

- Define feature detector as:

  A decision tree which detects points with a high repeatability.

- To evaluate repeatability:
  1. Detect features in all frames.
  2. Perform non-maximal suppression.
  3. Compute repeatability.
- Repeatability is a non-convex function of the tree configuration.
- Optimize tree using simulated-annealing.
- Use more offsets than FAST.

FAST-ER: Enhanced Repeatability



- Use more offsets than FAST.

Cost function

1. Higher repeatability is better.
2. Every pixel is a feature $\Rightarrow$ repeatability is 100%.
3. A single detected feature can have 100% repeatability.

Multi-objective optimization needed:

$$cost = (k_R + R^{-2})(k_N + N^2)(k_S + S^{-2})$$

R = Repeatability.
N = Number of detected features.
S = Size of tree.

Cost function

1. Higher repeatability is better.
2. Every pixel is a feature $\Rightarrow$ repeatability is 100%.
3. A single detected feature can have 100% repeatability.

Multi-objective optimization needed:

$$cost = (k_R + R^{-2})(k_N + N^2)(k_S + S^{-2})$$

R = Repeatability.
N = Number of detected features.
S = Size of tree.

Cost function

1. Higher repeatability is better.
2. Every pixel is a feature $\Rightarrow$ repeatability is 100%.
3. A single detected feature can have 100% repeatability.

Multi-objective optimization needed:

$$cost = (k_R + R^{-2})(k_N + N^2)(k_S + S^{-2})$$

R = Repeatability.
N = Number of detected features.
S = Size of tree.

Cost function

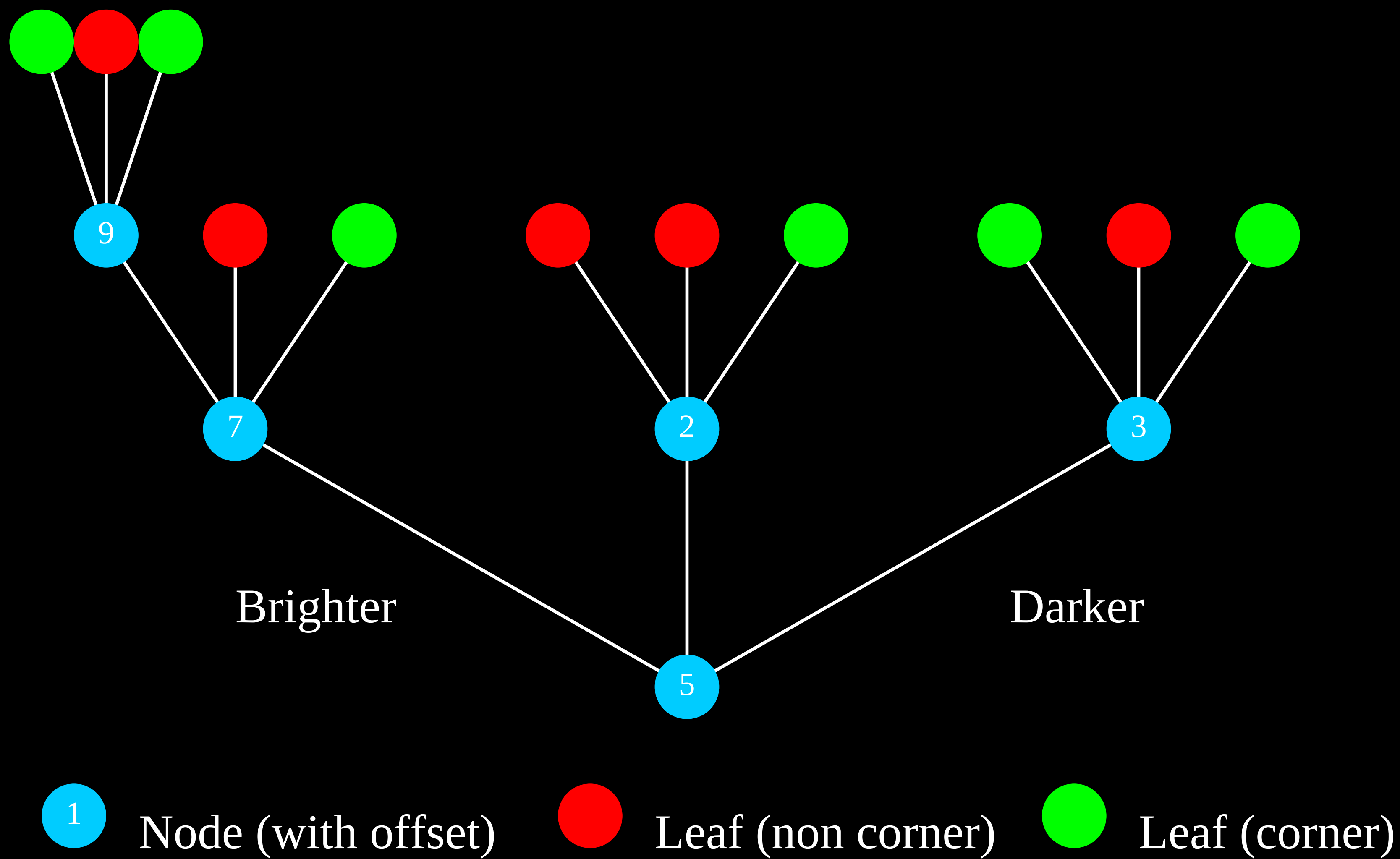
1. Higher repeatability is better.
2. Every pixel is a feature $\Rightarrow$ repeatability is 100%.
3. A single detected feature can have 100% repeatability.

Multi-objective optimization needed:

$$cost = (k_R + R^{-2})(k_N + N^2)(k_S + S^{-2})$$

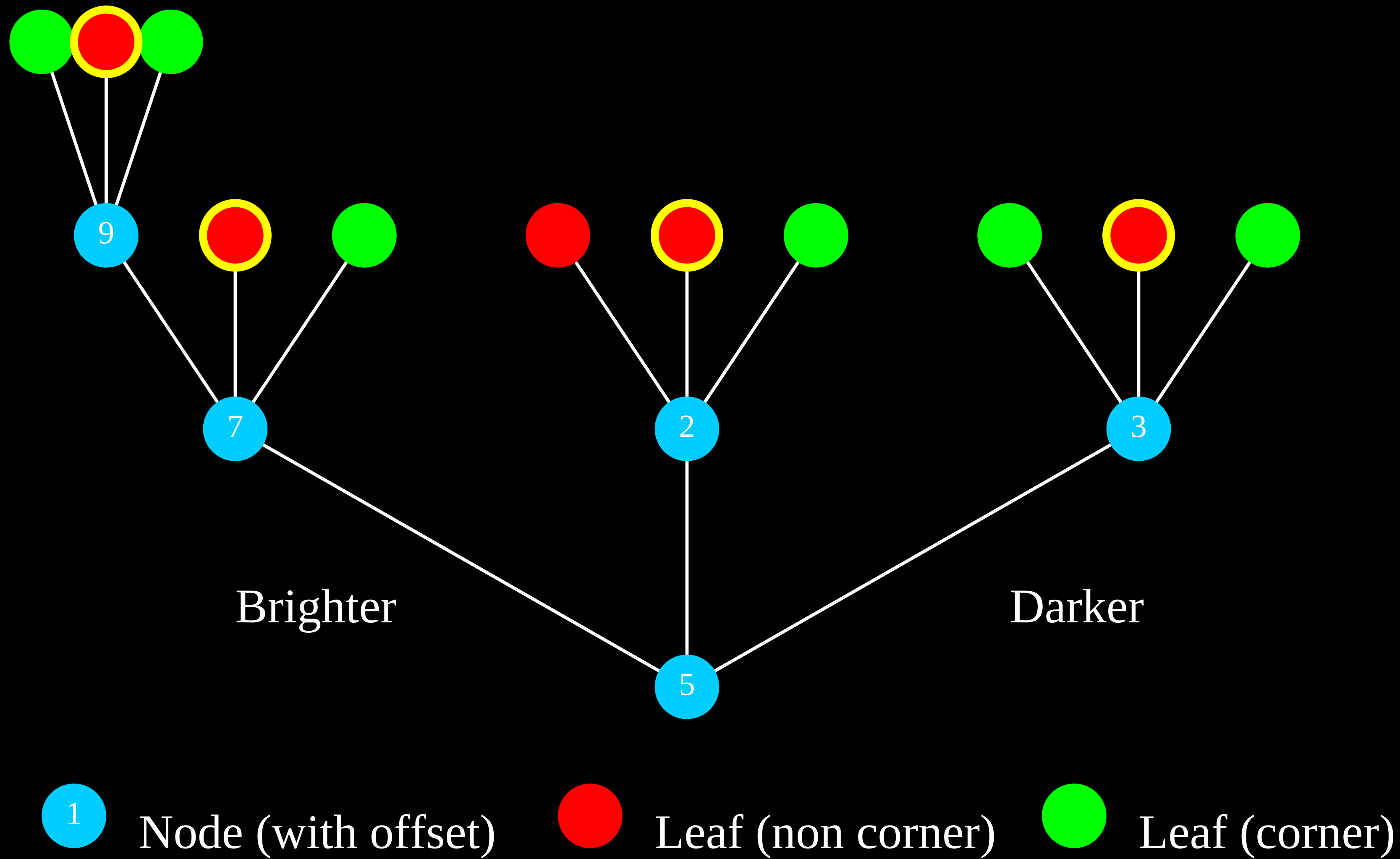
R = Repeatability.
N = Number of detected features.
S = Size of tree.

Operations

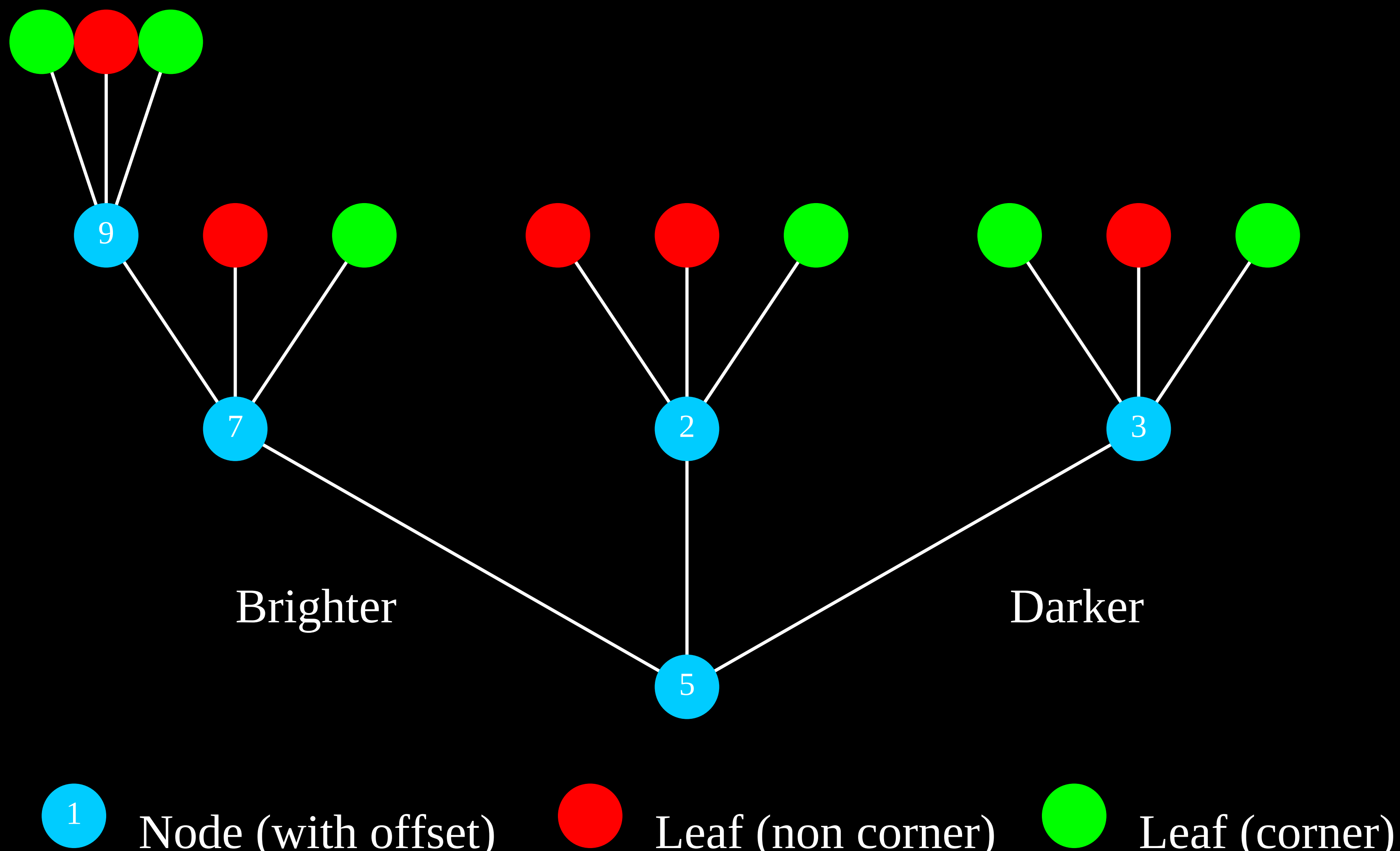

9
7
2
3
5
Brighter
Darker
1
Node (with offset)
Leaf (non corner)
Leaf (corner)

Operations

'Similar' lead nodes are constrained.

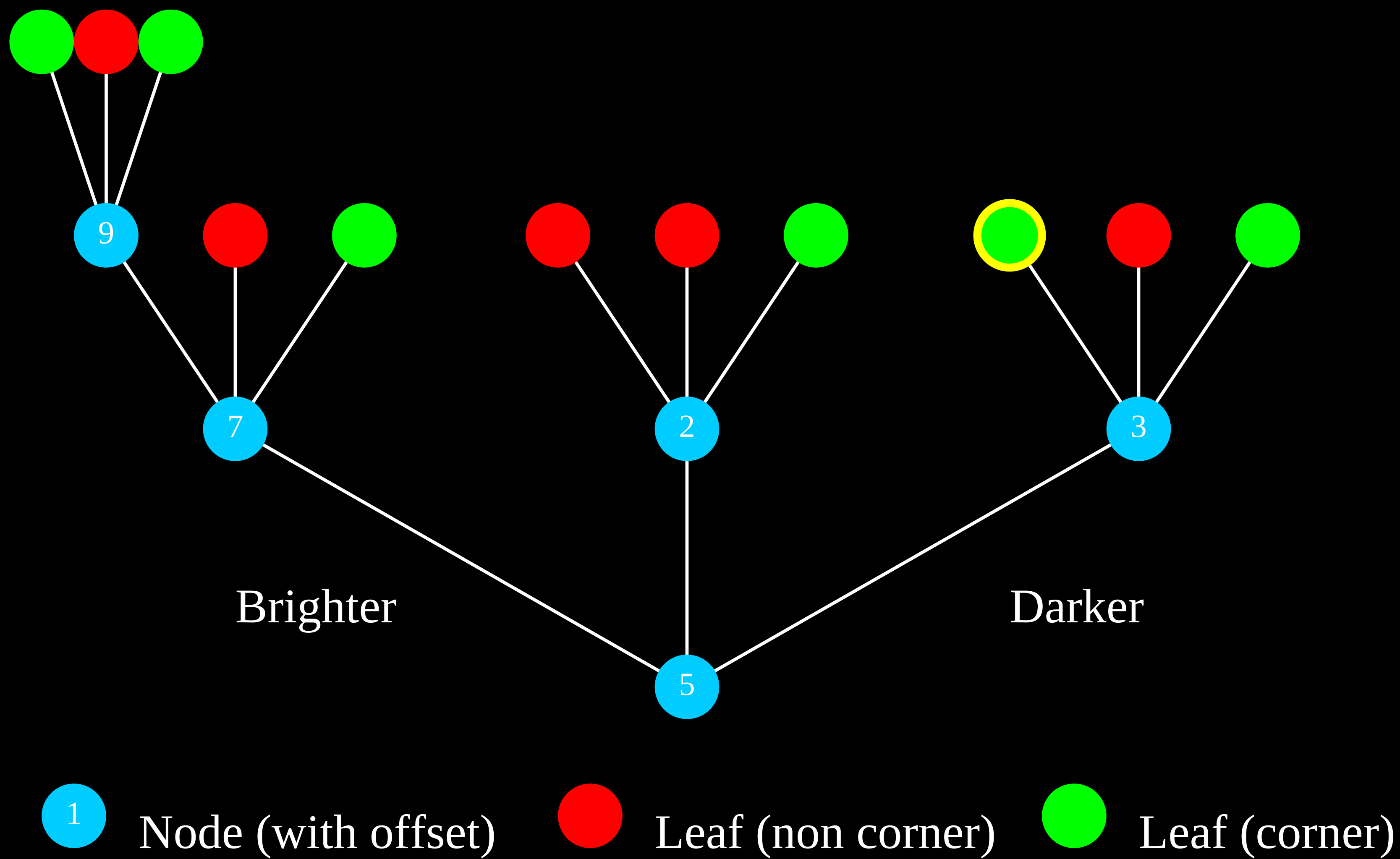


Operations


9
7
2
3
5
Brighter
Darker
1
Node (with offset)
Leaf (non corner)
Leaf (corner)

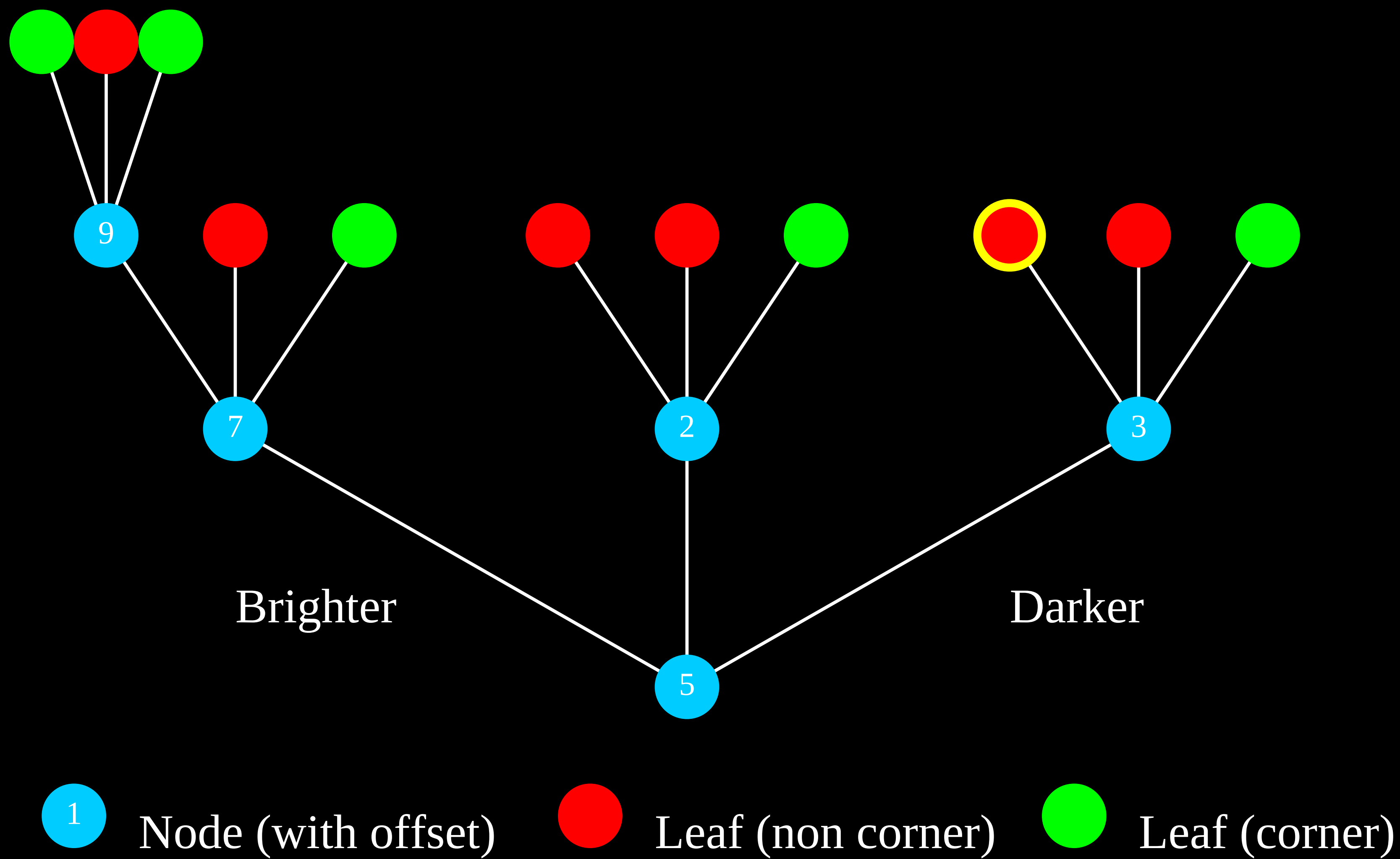
Operations

Select a random node. If node is a leaf:



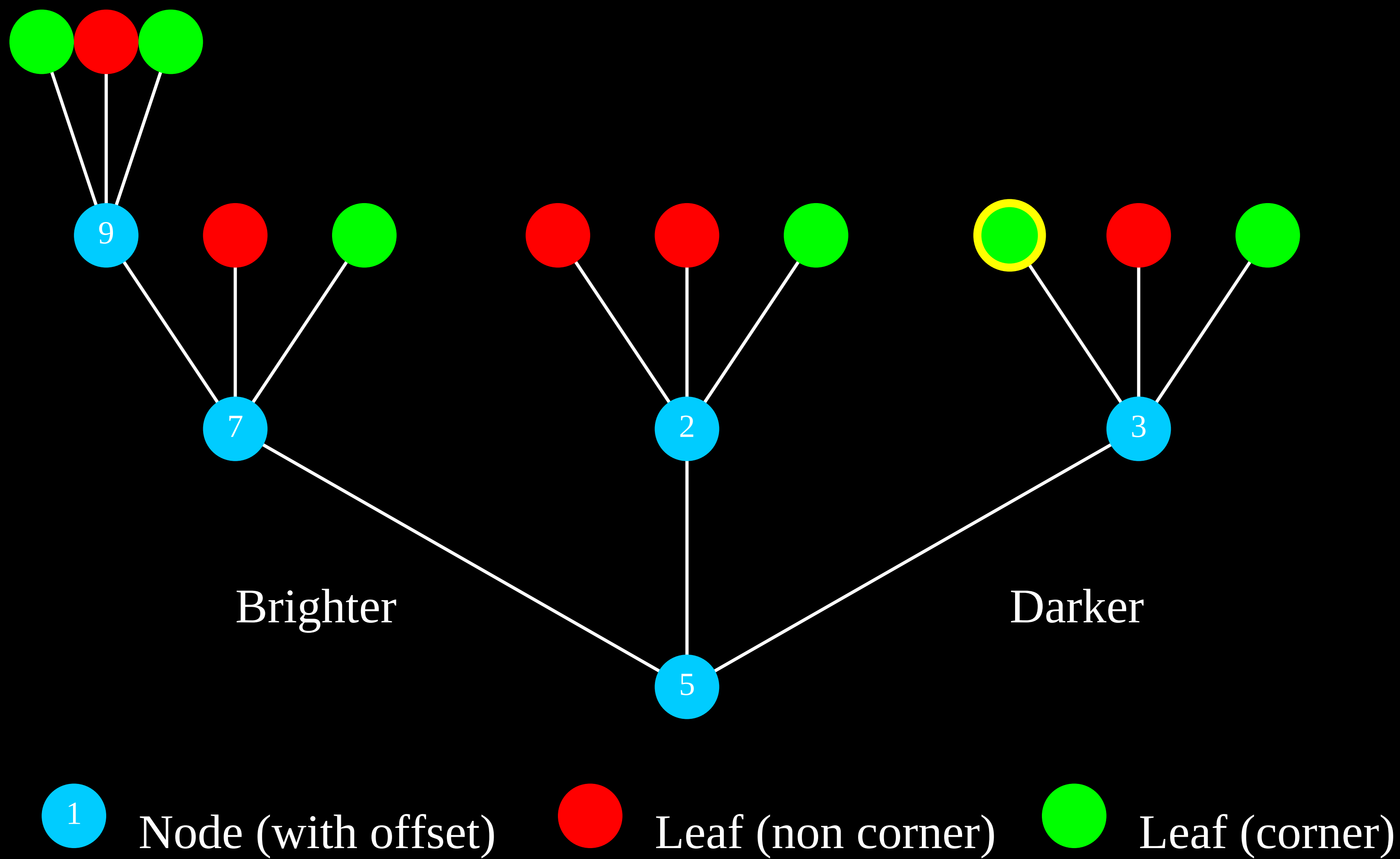
Operations

flip the class (if possible), . . .



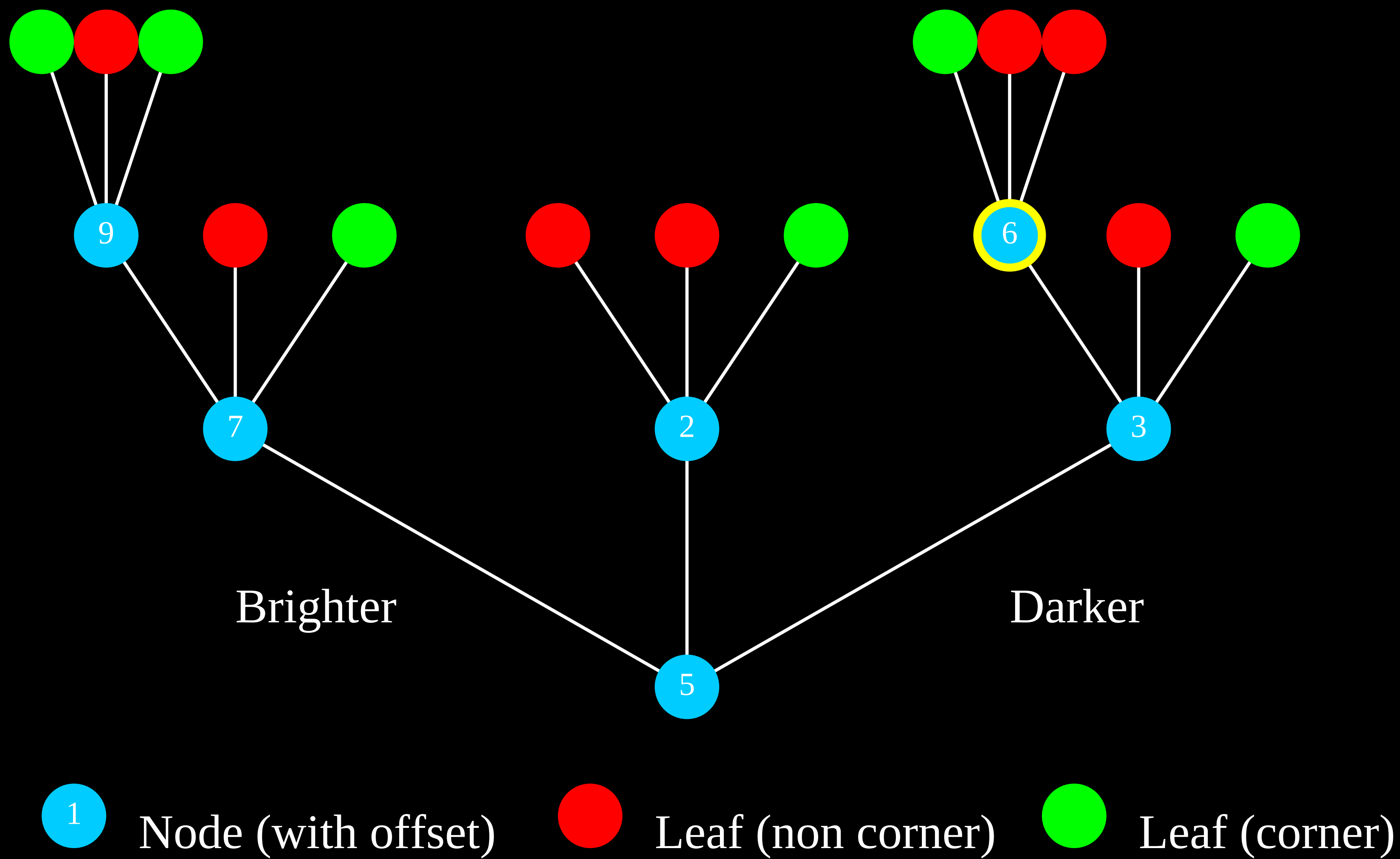
Operations

. . . or . . .



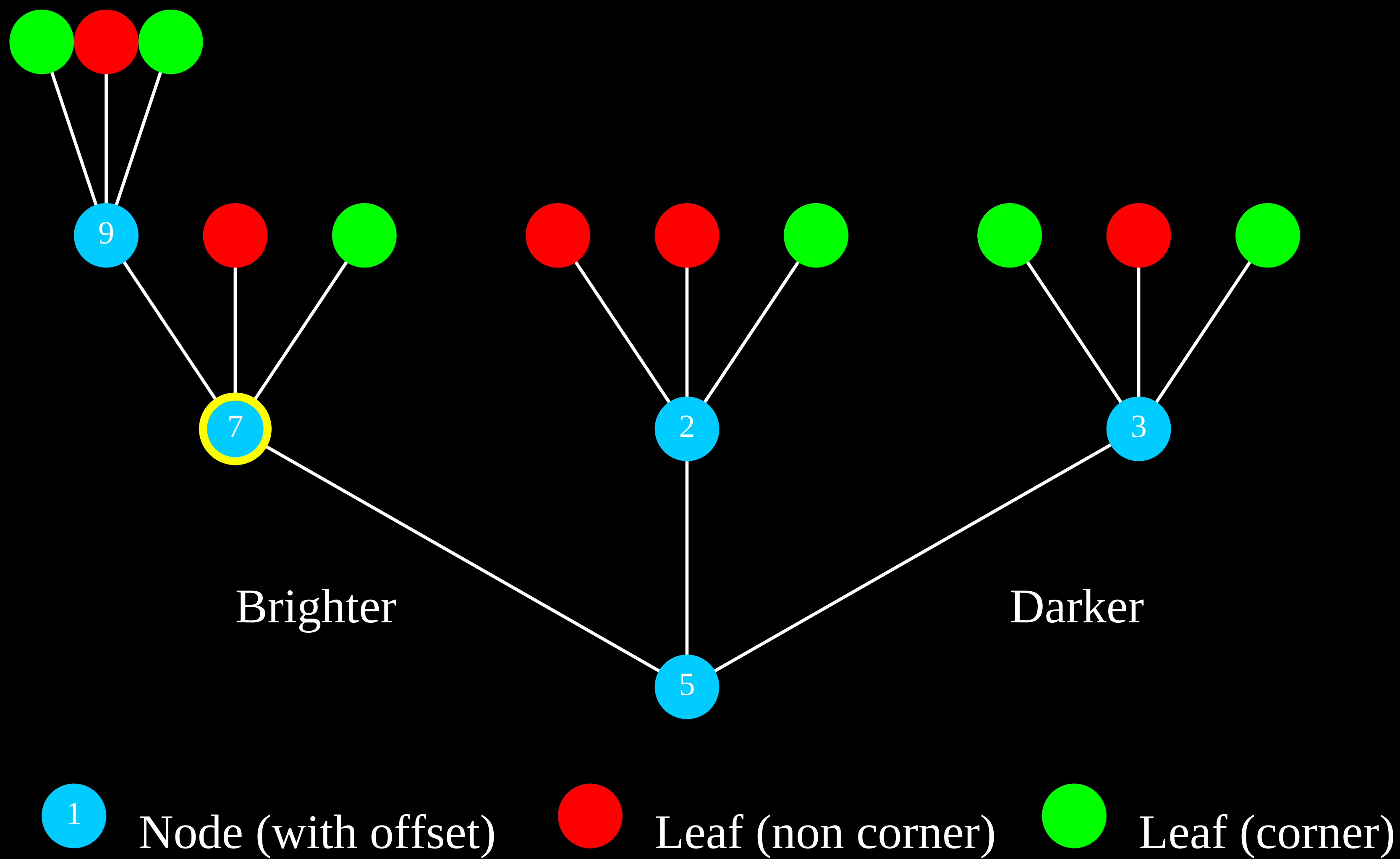
Operations

grow a random subtree.



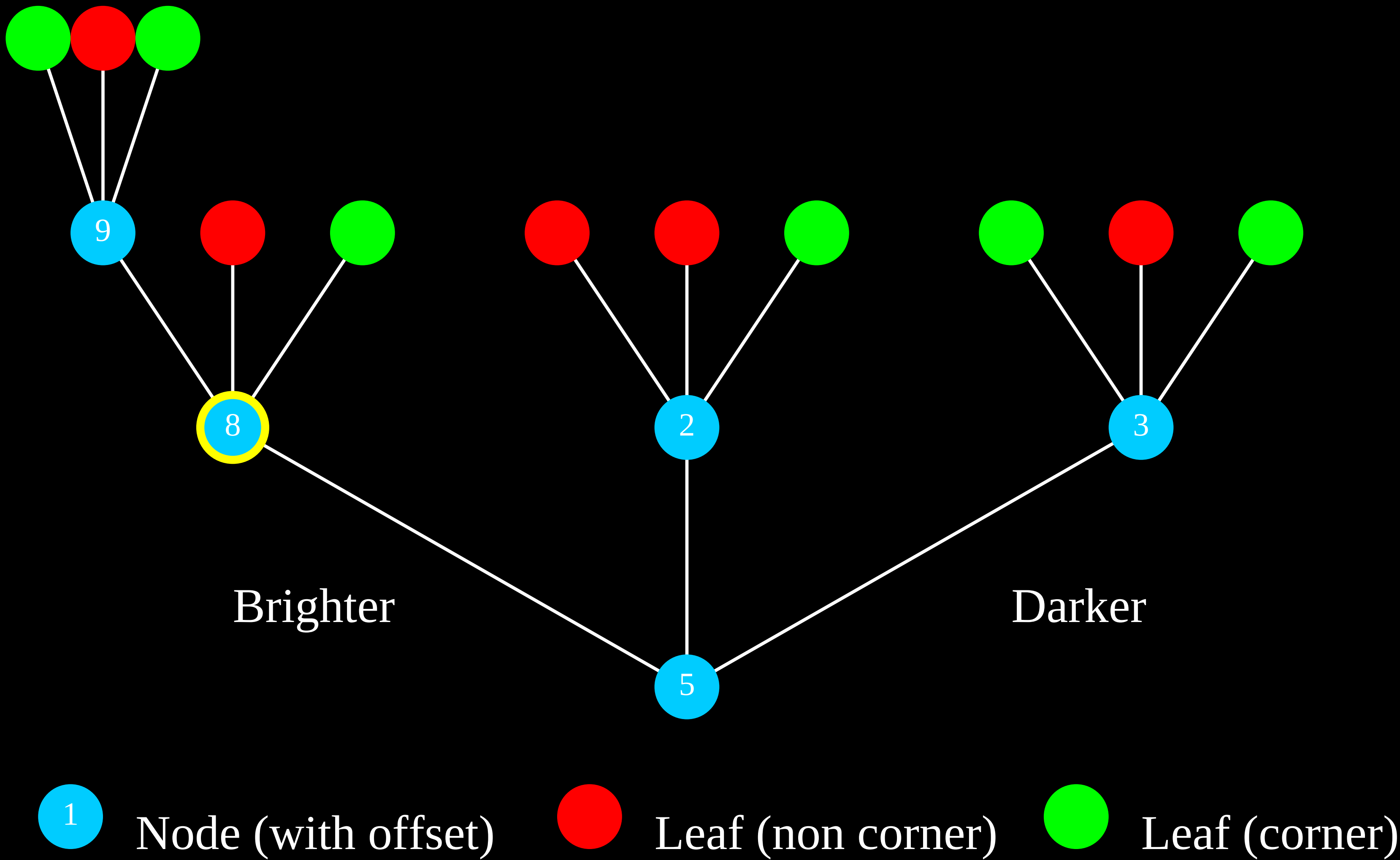
Operations

If node is a non-leaf:



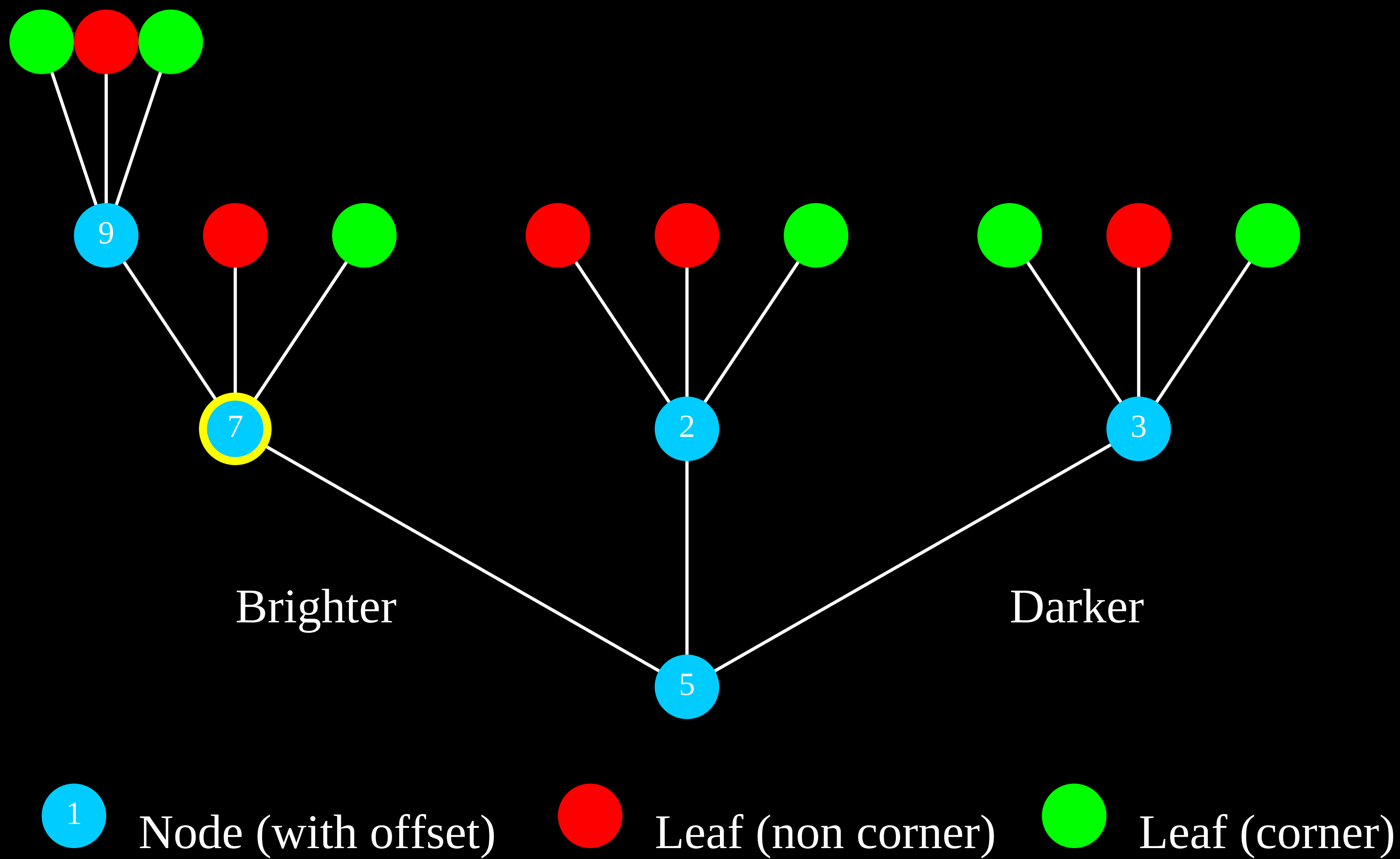
Operations

randomize the offset, . . .



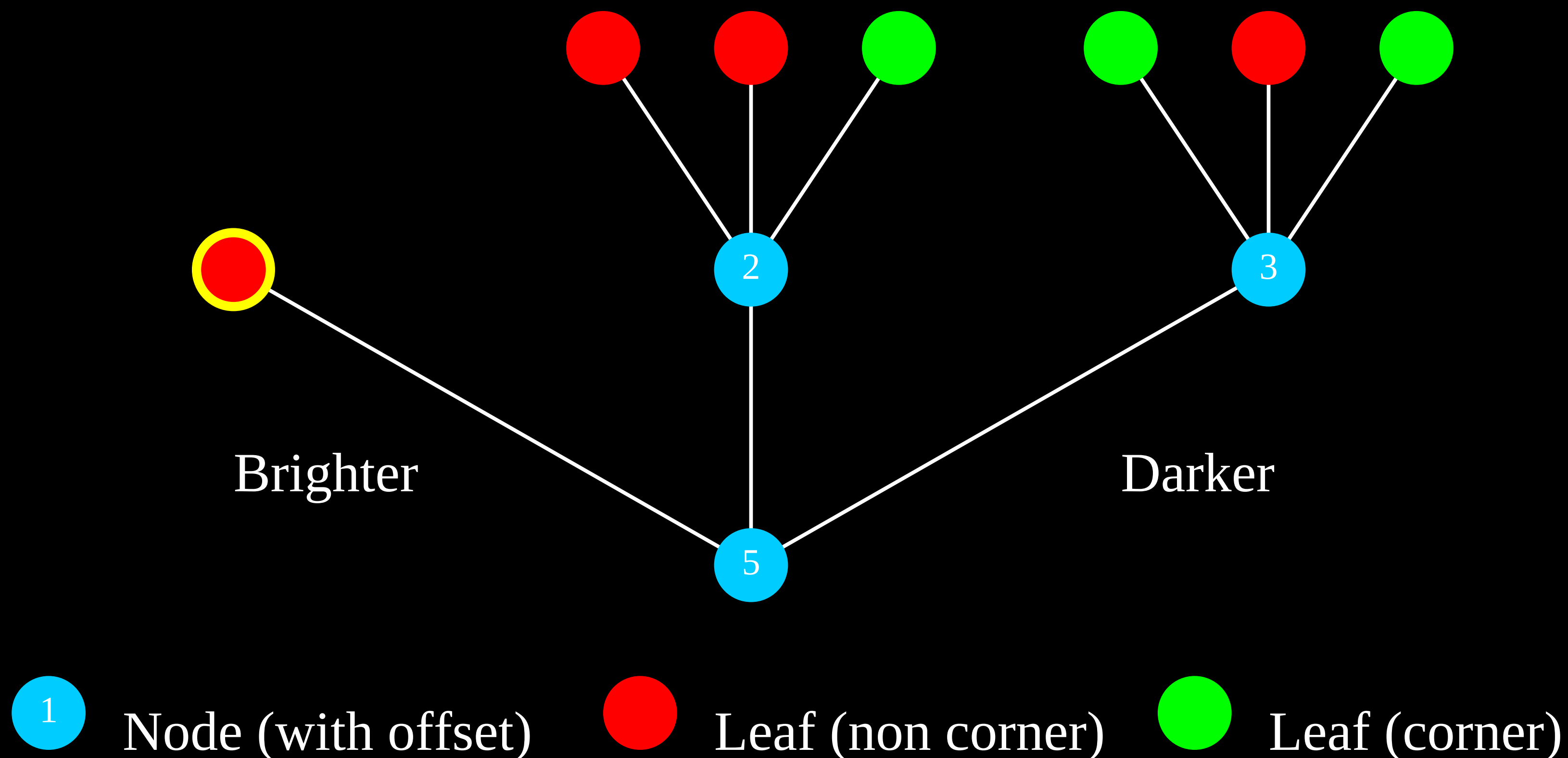
Operations

. . . or . . .



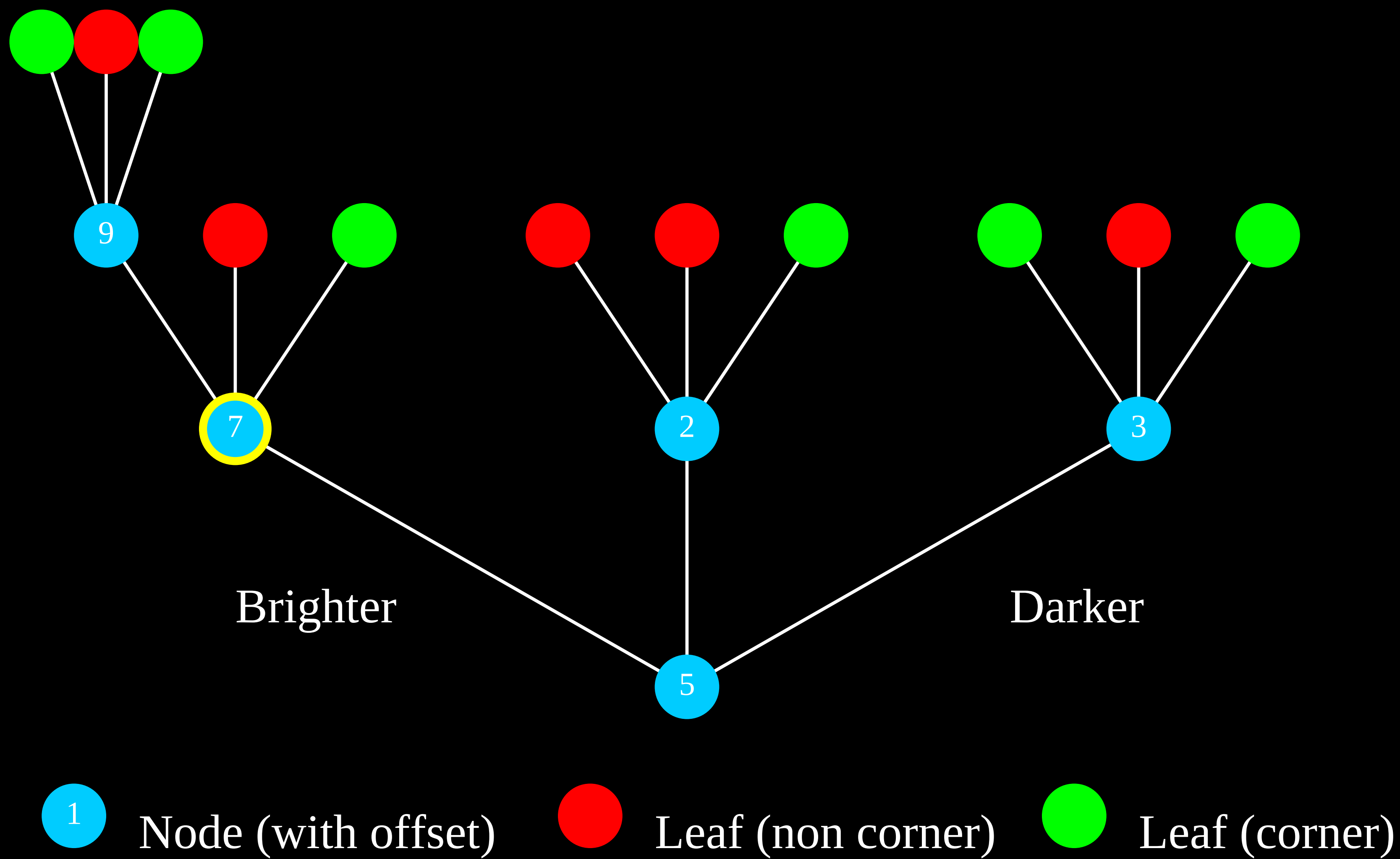
Operations

replace node with a leaf, . . .



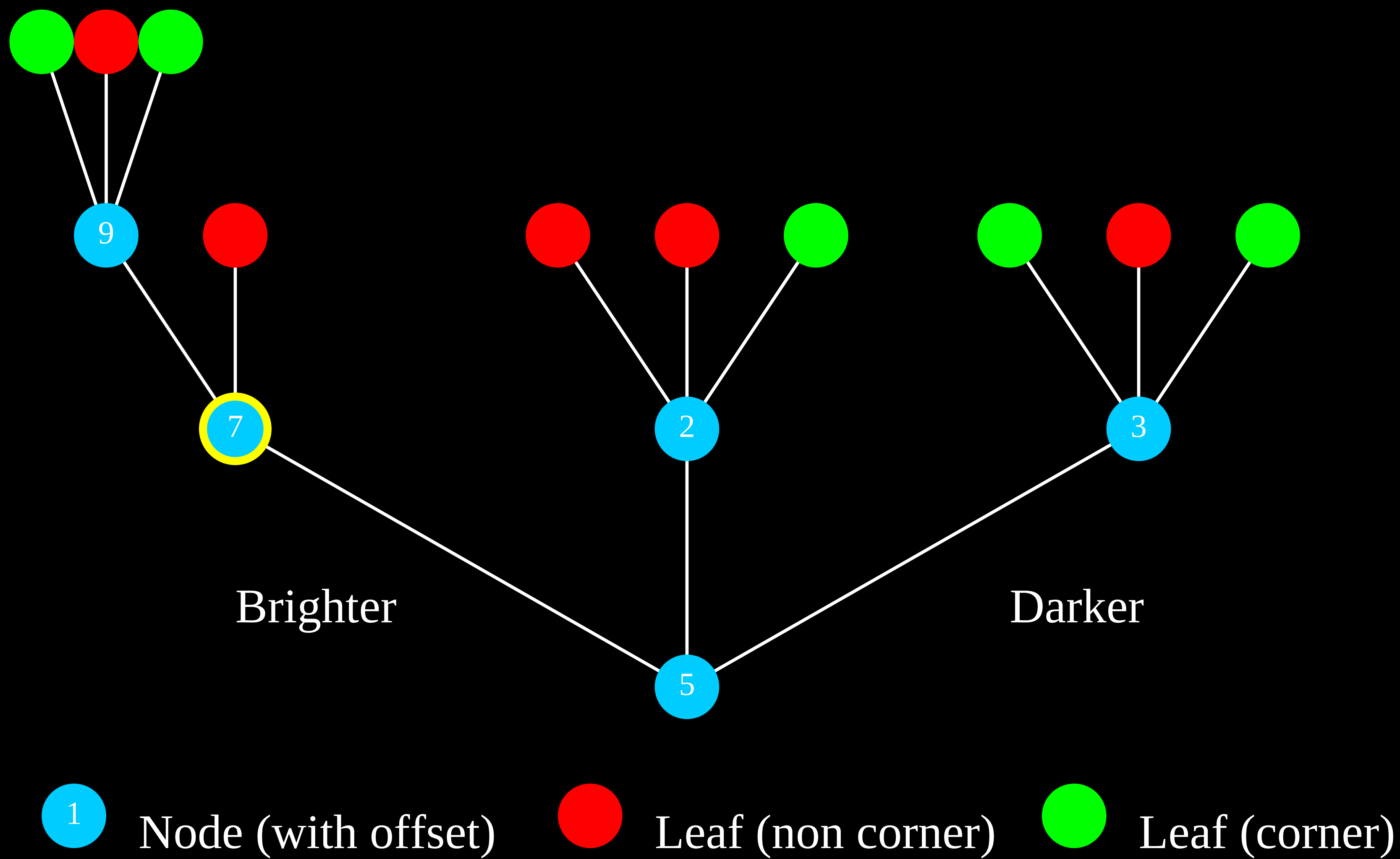
Operations

. . . or . . .



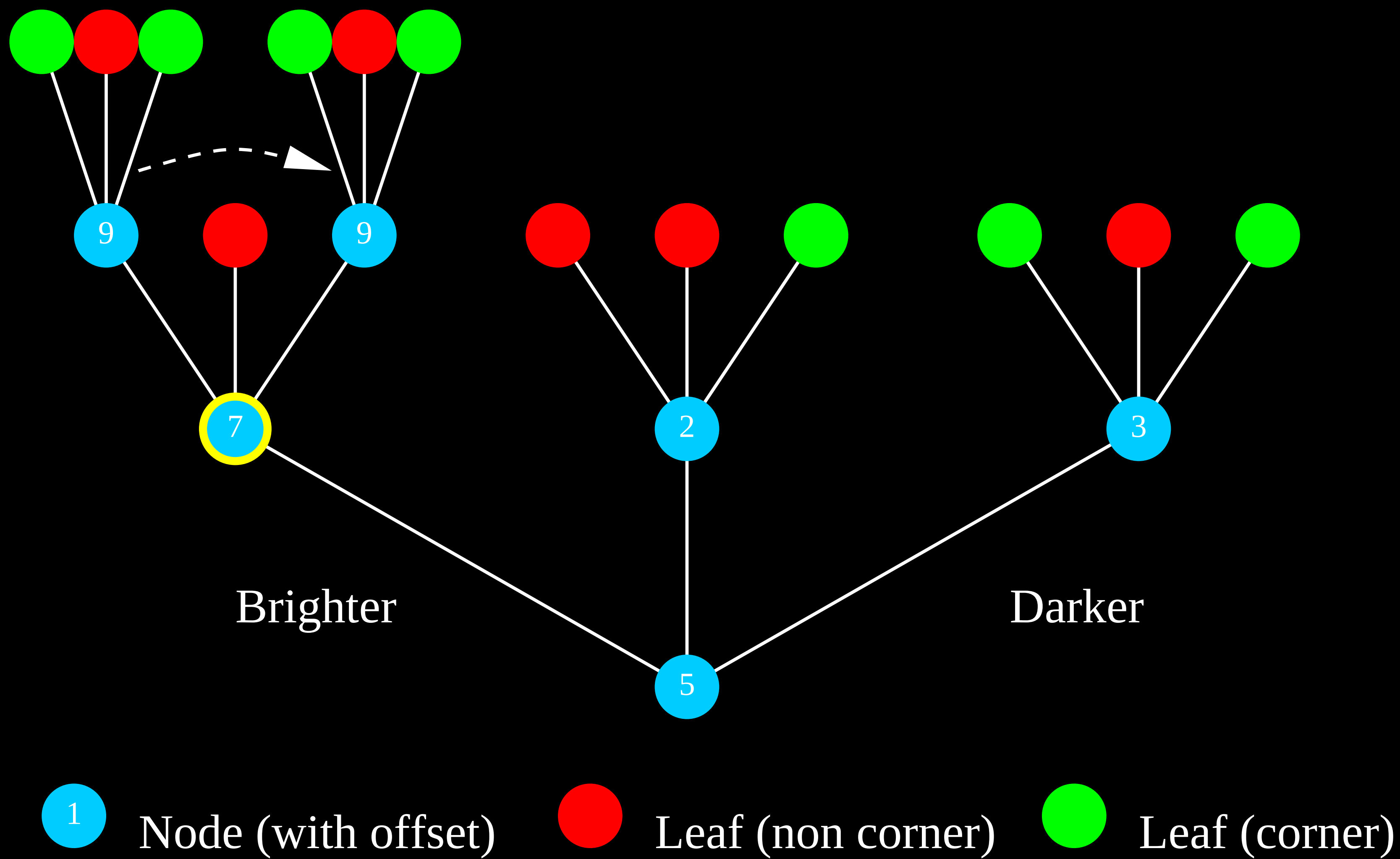
Operations

delete one subtree



Operations

and replace it with a copy of another subtree.



Reducing the burden on the optimizer

Corners should be invariant to:

- Rotation.
- Reflection.
- Intensity inversion.

There are 16 combinations:

- 4 simple rotations (multiples of 90°).
- 2 reflections.
- 2 intensity inversions.

Run the detector in all combinations.

Iteration scheme

For 100,000 iterations:

1. Randomly modify tree.
2. Output as code.
3. Detect features and perform nonmax suppression.
4. Compute repeatability.
5. Evaluate cost.
6. Keep the modification if:

$$e^{\frac{\text{oldcost}-\text{cost}}{\text{temp}}} > \text{rand}$$

7. Reduce the temperature.

Now repeat that 100 times (200 Hours required).

Training data for repeatability



- Change in scale.
- Mostly affine warping.
- Varied texture.

Optimizing FAST-ER for speed

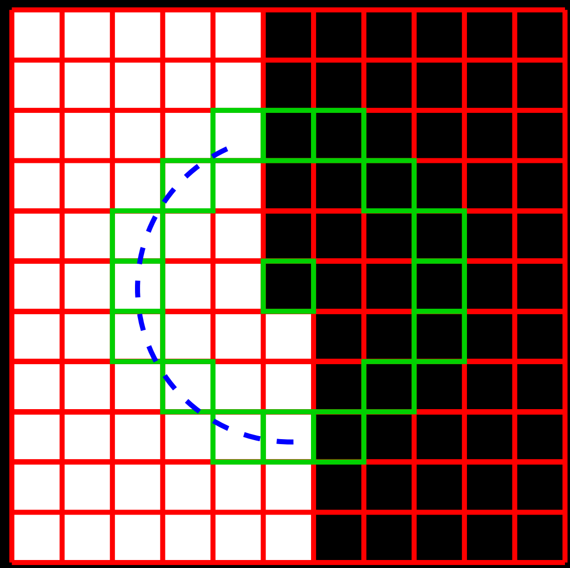
- Tree is applied 16 times at each pixel
- Use repeatability optimized FAST-ER to gather training data:
  1. Detect points in images.
  2. Extract ternary vector of surrounding pixels available to FAST-ER.
- Train single decision tree using ID3.
- Output tree as C code.

Results

Comparisons

- FAST detectors
  - Which N is best?
  - Which of the 200 FAST-ER detectors is best?
- Other detectors
  - Harris.
  - Shi-Tomasi
  - DoG (Difference of Gaussians)
  - Harris-Laplace
  - SUSAN
- What parameters should these detectors use?

Comparisons



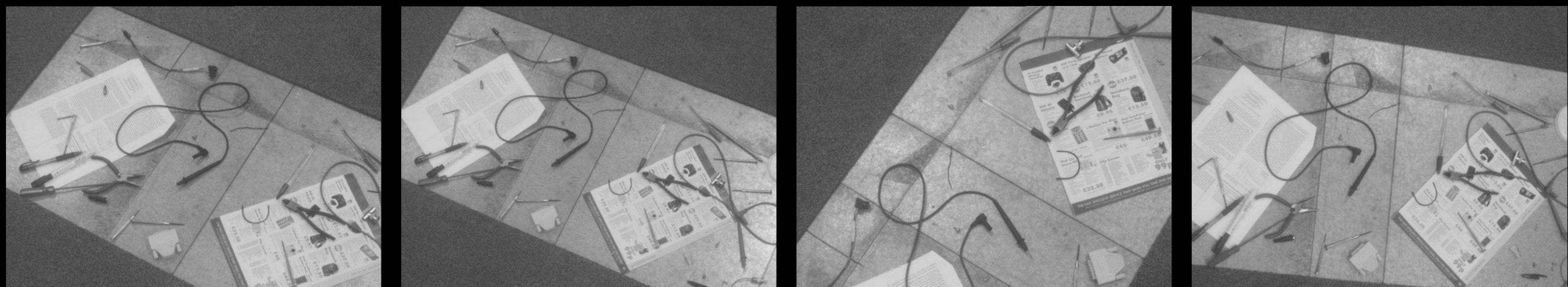
- FAST detectors
  - Which N is best?
  - Which of the 200 FAST-ER detectors is best?
- Other detectors
  - Harris.
  - Shi-Tomasi
  - DoG (Difference of Gaussians)
  - Harris-Laplace
  - SUSAN
- What parameters should these detectors use?

Evaluation: Datasets (3D Models)

14 images:

15 images:

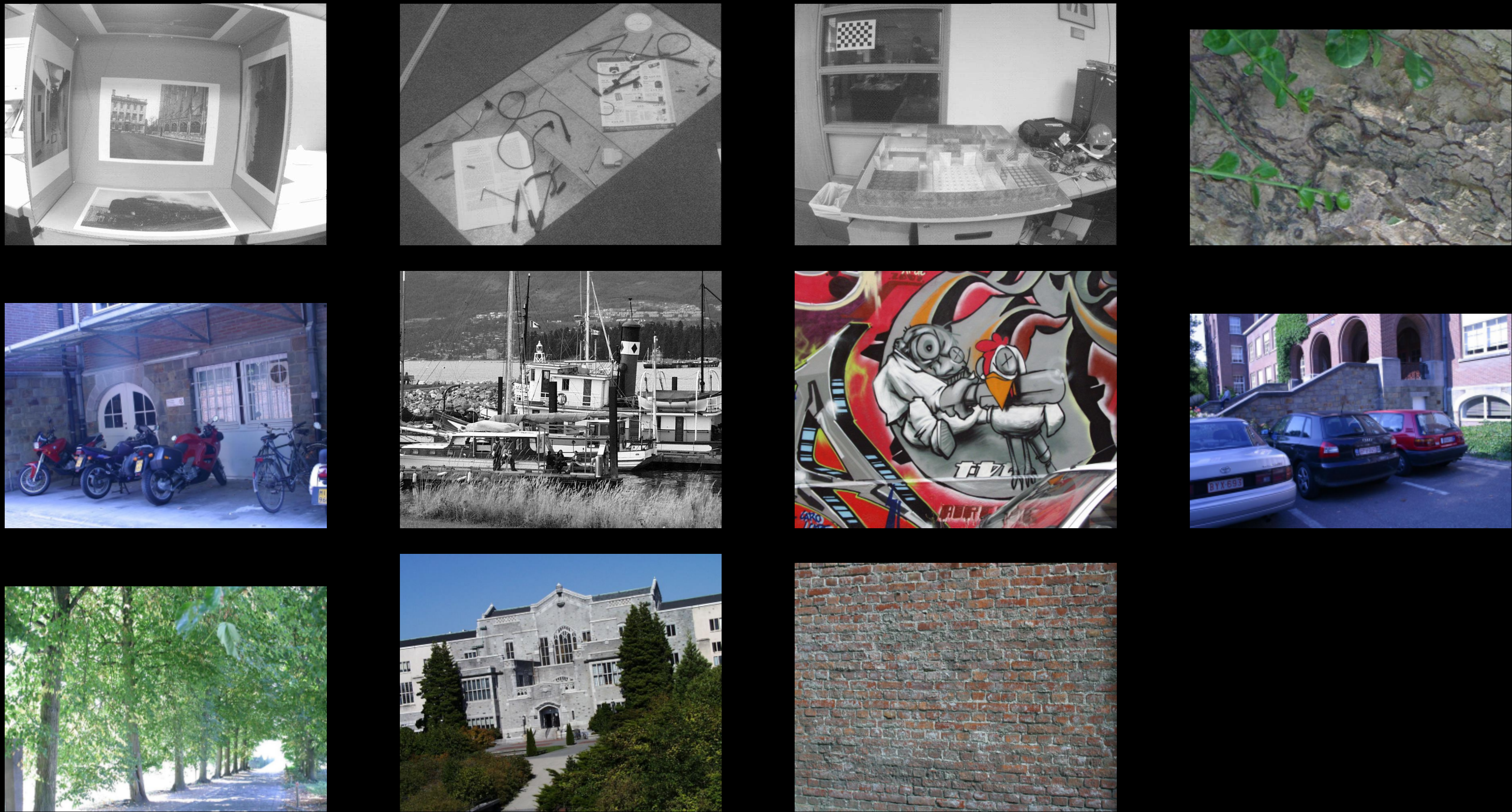
8 images:



Evaluation: Homographies

6 images per set:

Results: repeatability curves



Aggregate results

Detector	A
FAST-ER	1313.6
FAST-9	1304.57
DoG	1275.59
Shi & Tomasi	1219.08
Harris	1195.2
Harris-Laplace	1153.13
FAST-12	1121.53
SUSAN	1116.79
Random	271.73

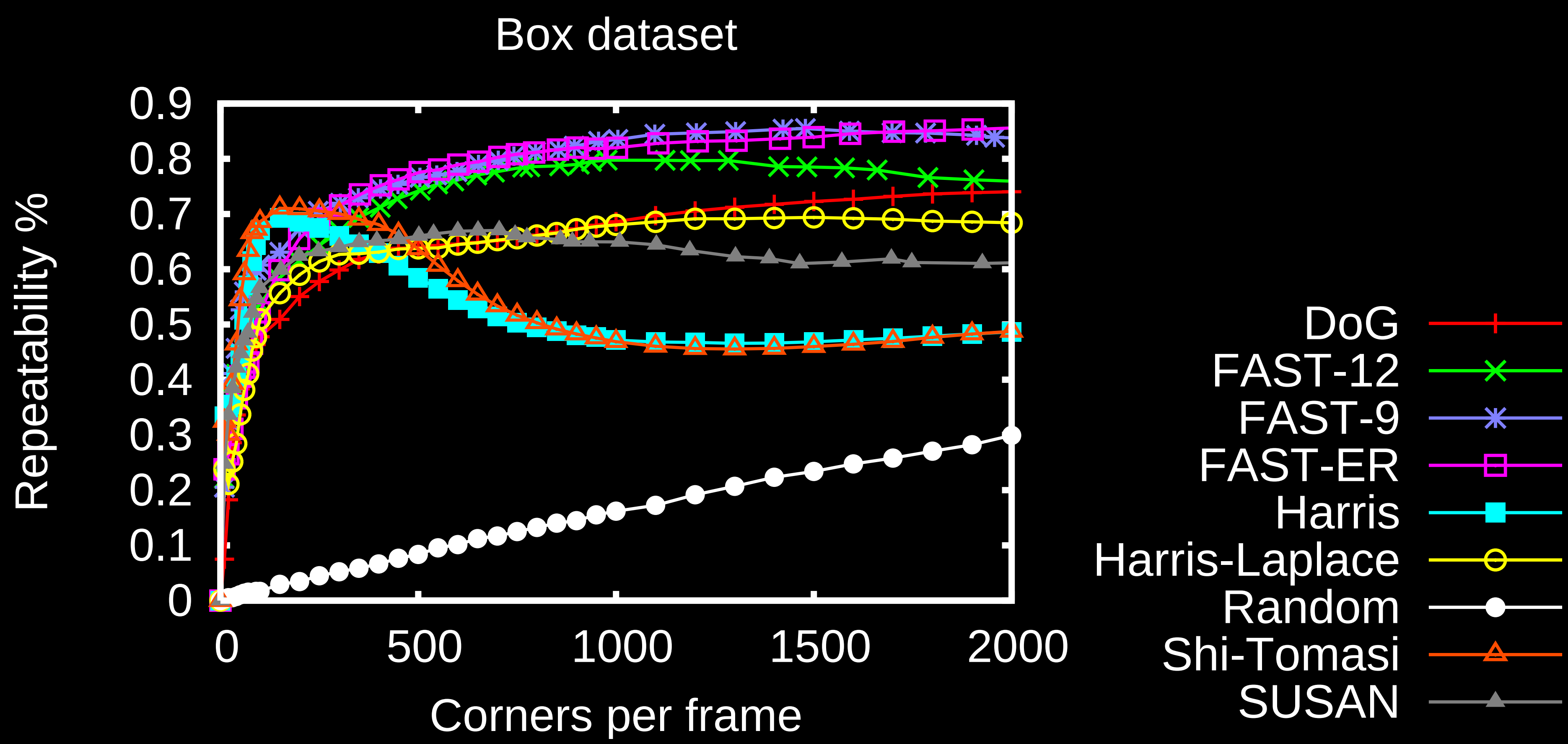
Conclusions

What do the results say?

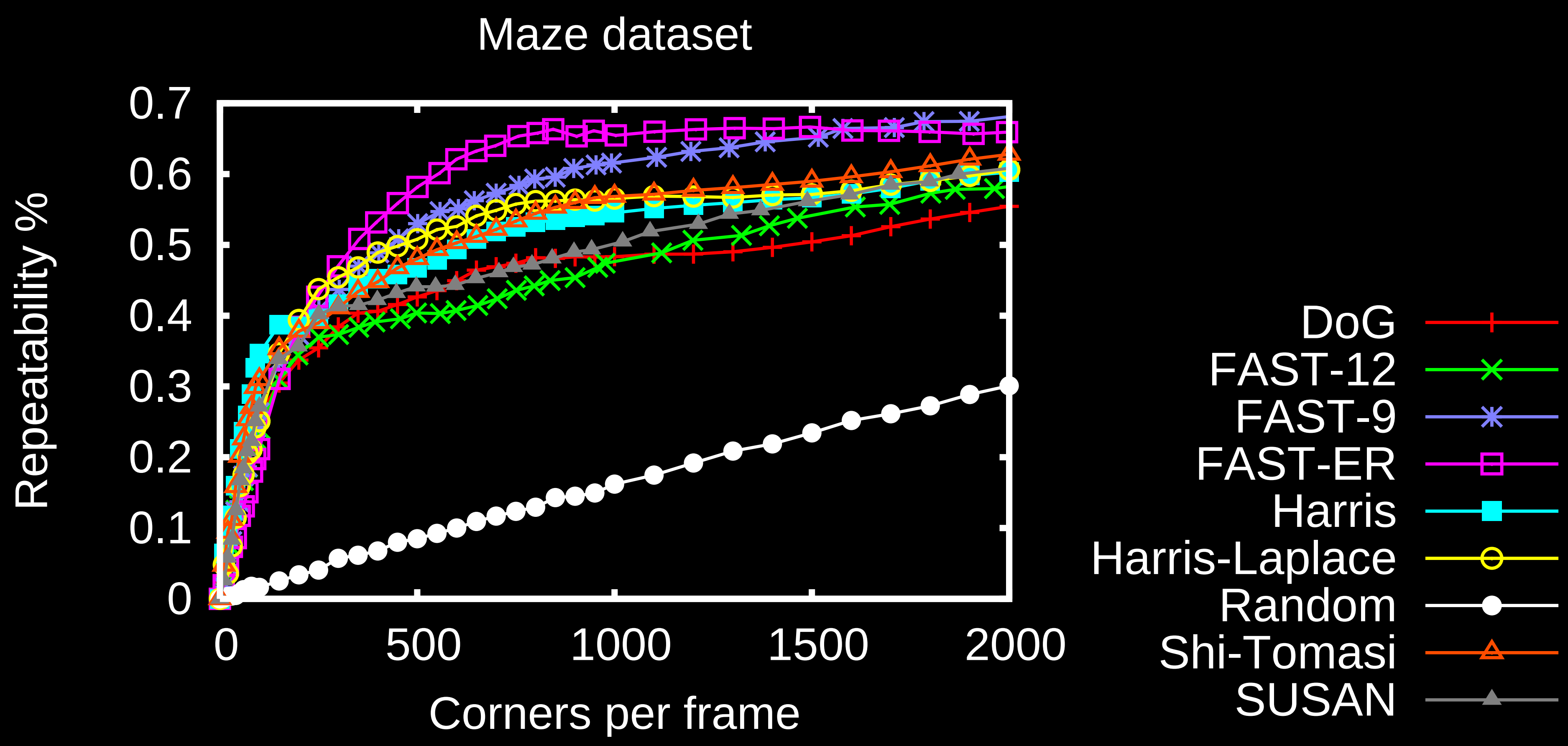
- FAST is suprisingly good.
- FAST-ER is better but slower.

More results

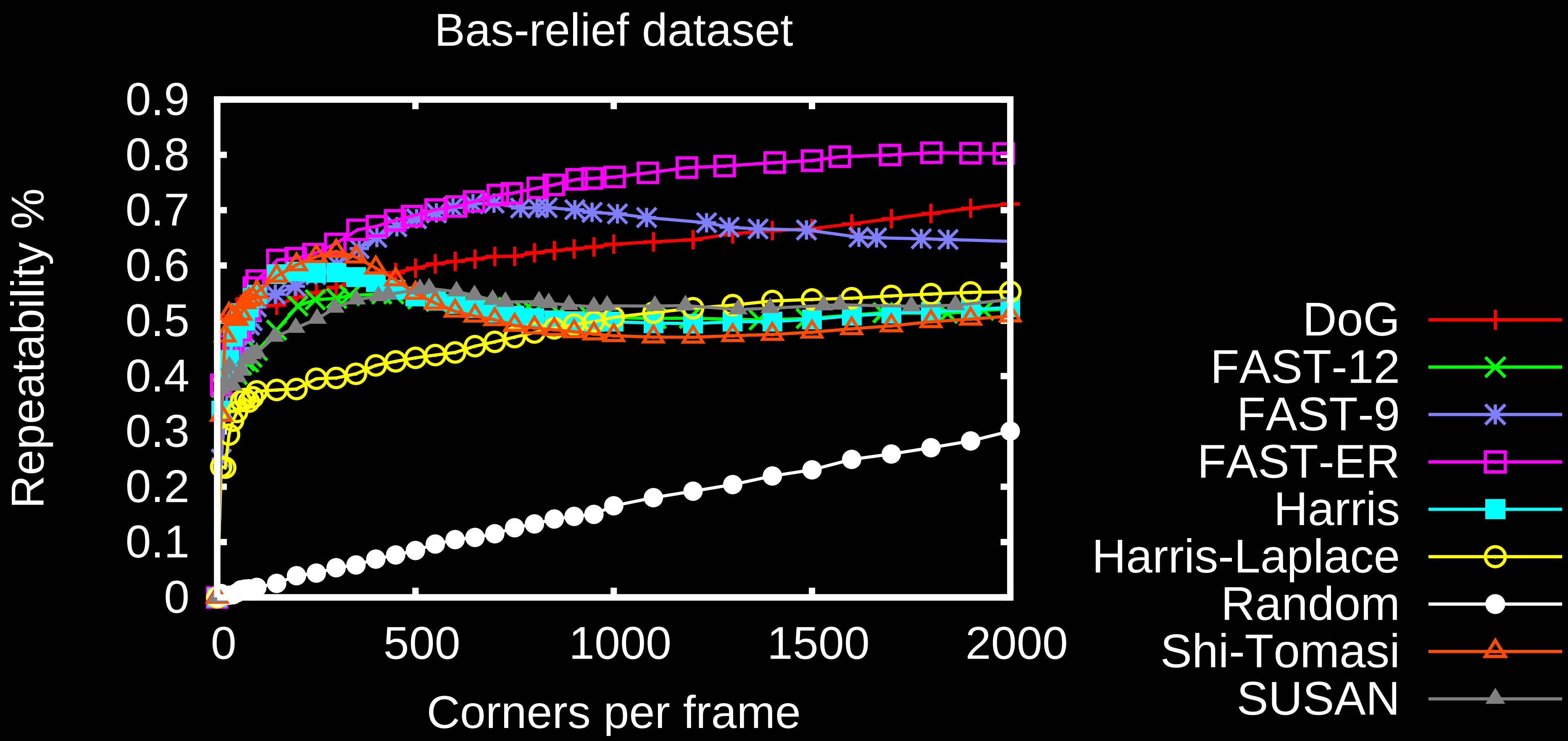
Results: Perspective (box) dataset



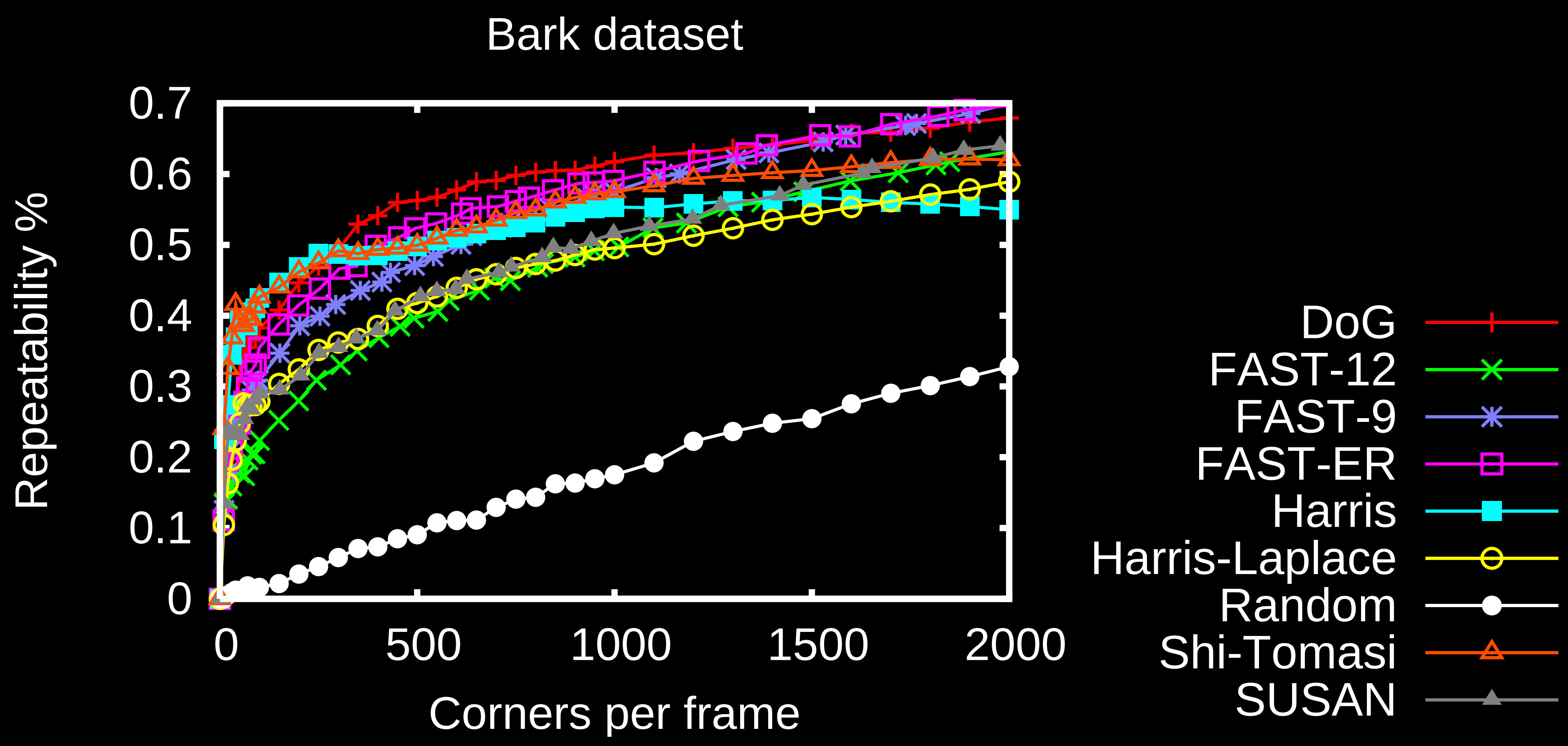
Results: Geometric dataset



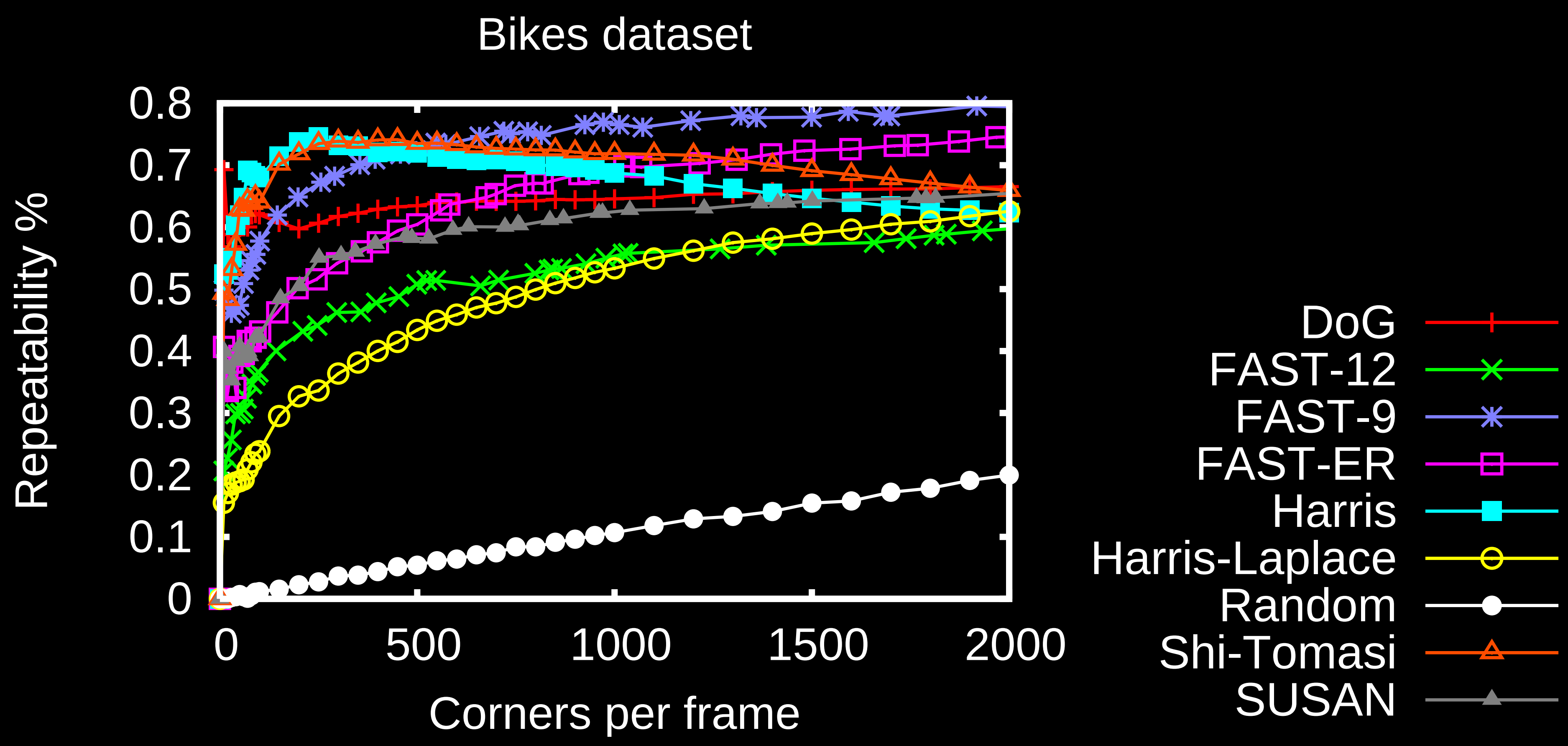
Results: Bas-relief dataset



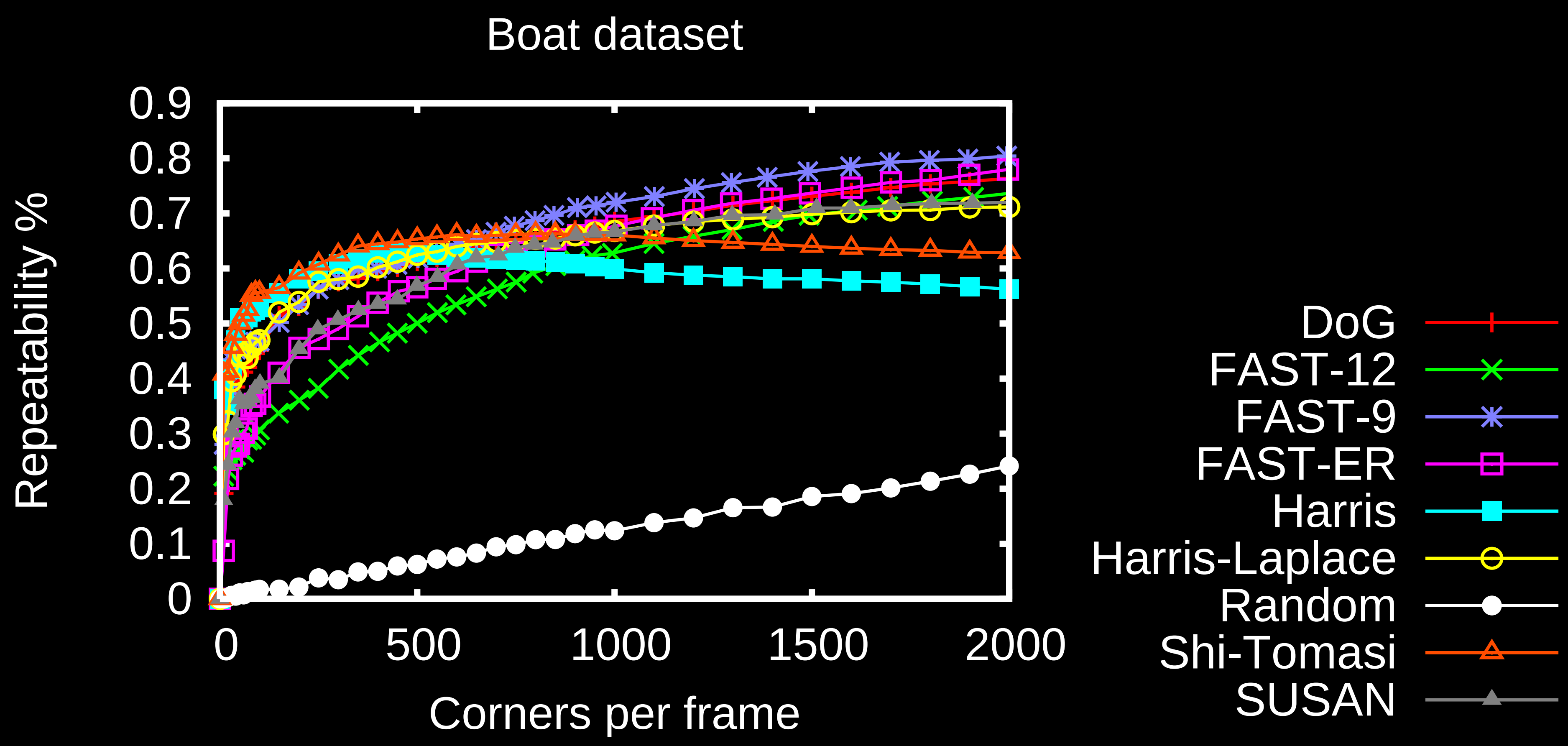
Results: Scale and rotation (bark) dataset



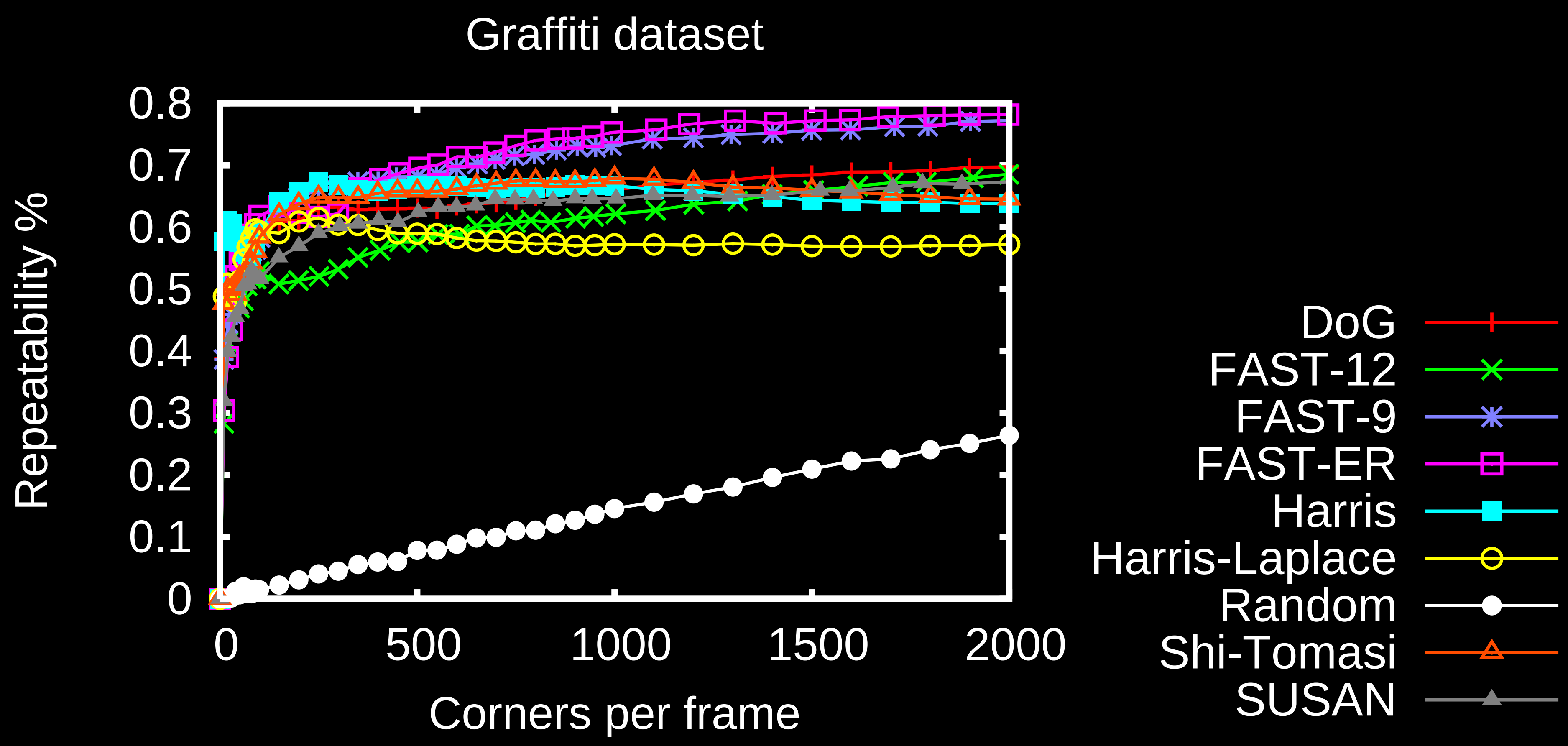
Results: Blur (bikes) dataset



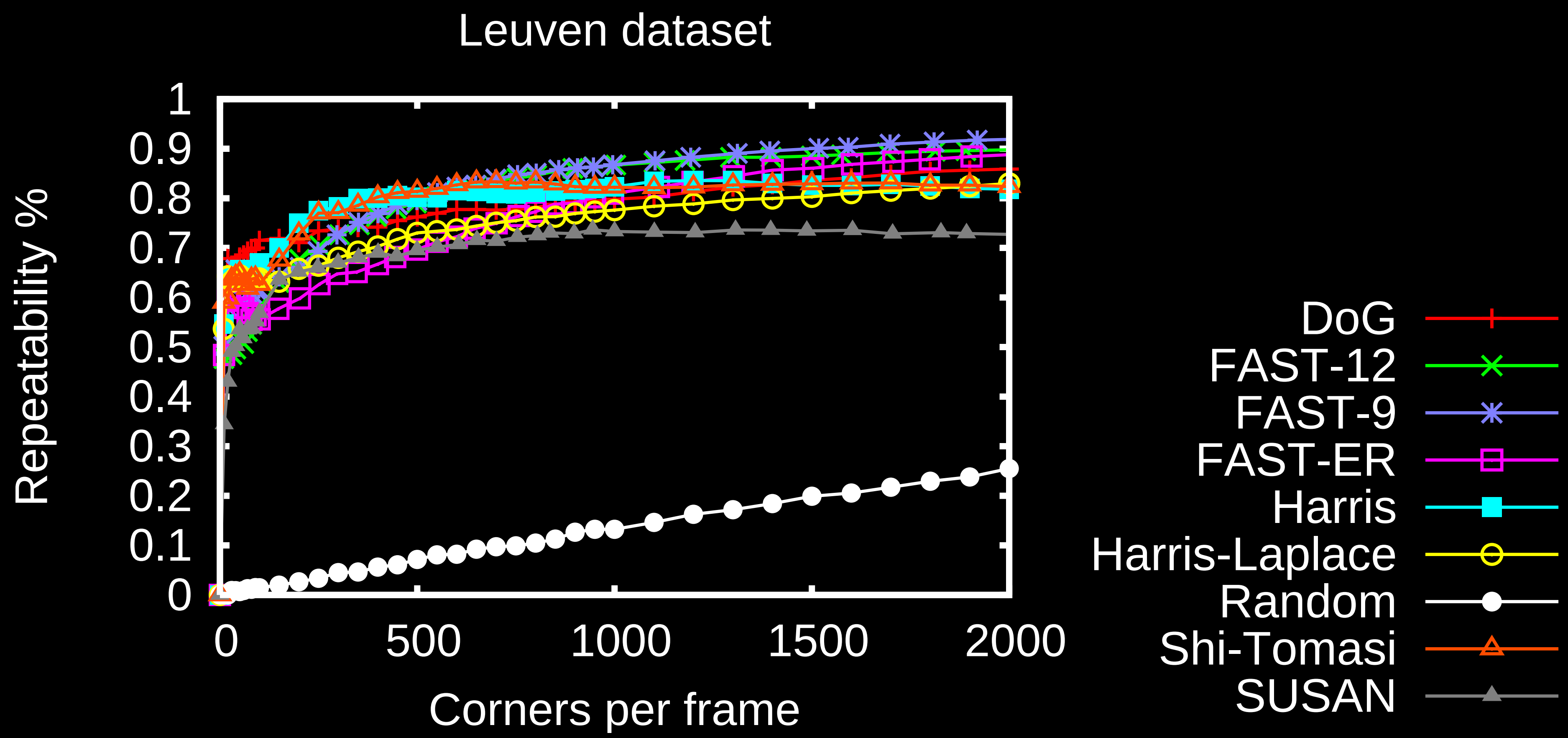
Results: Scale and rotation (boat) dataset



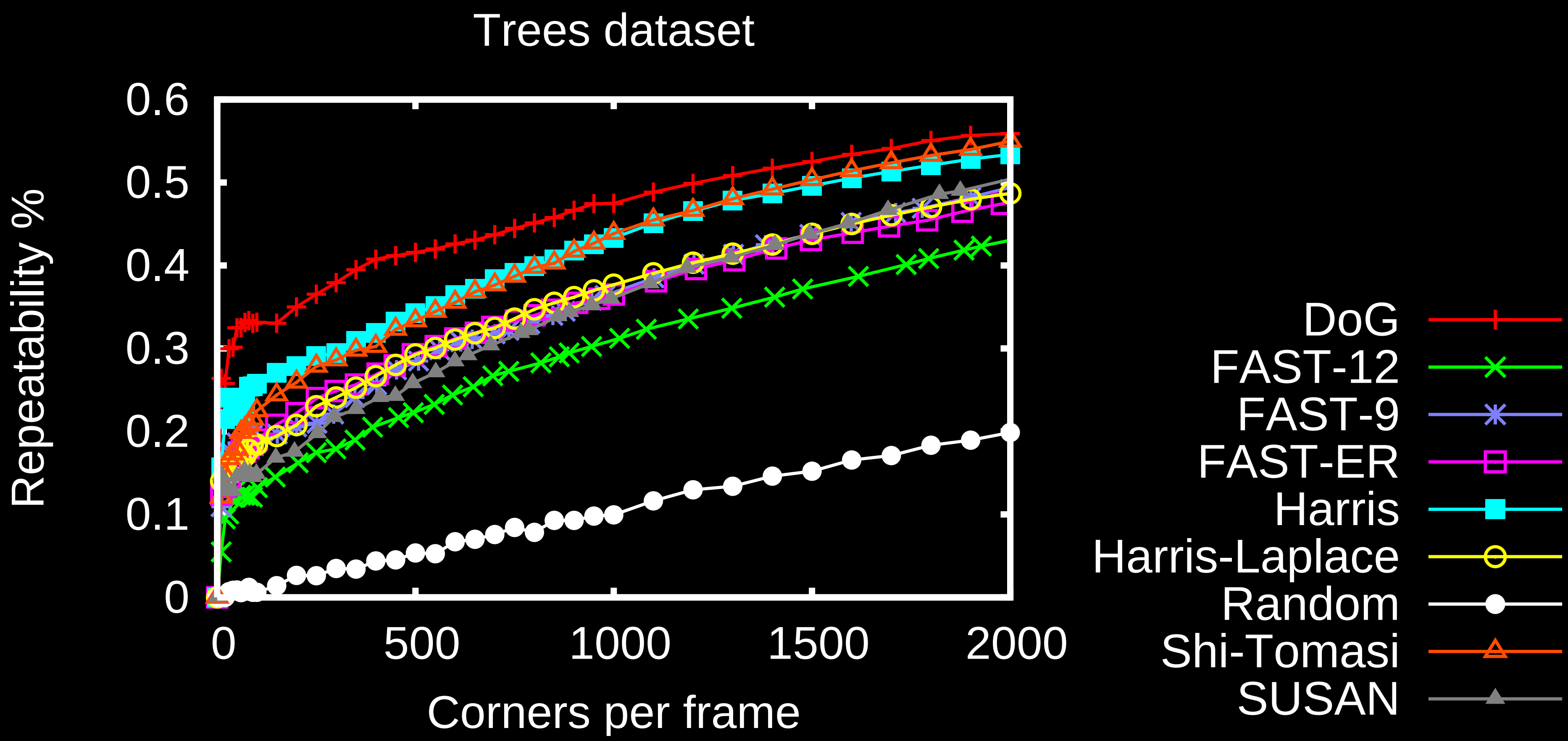
Results: Perspective (graffiti) dataset



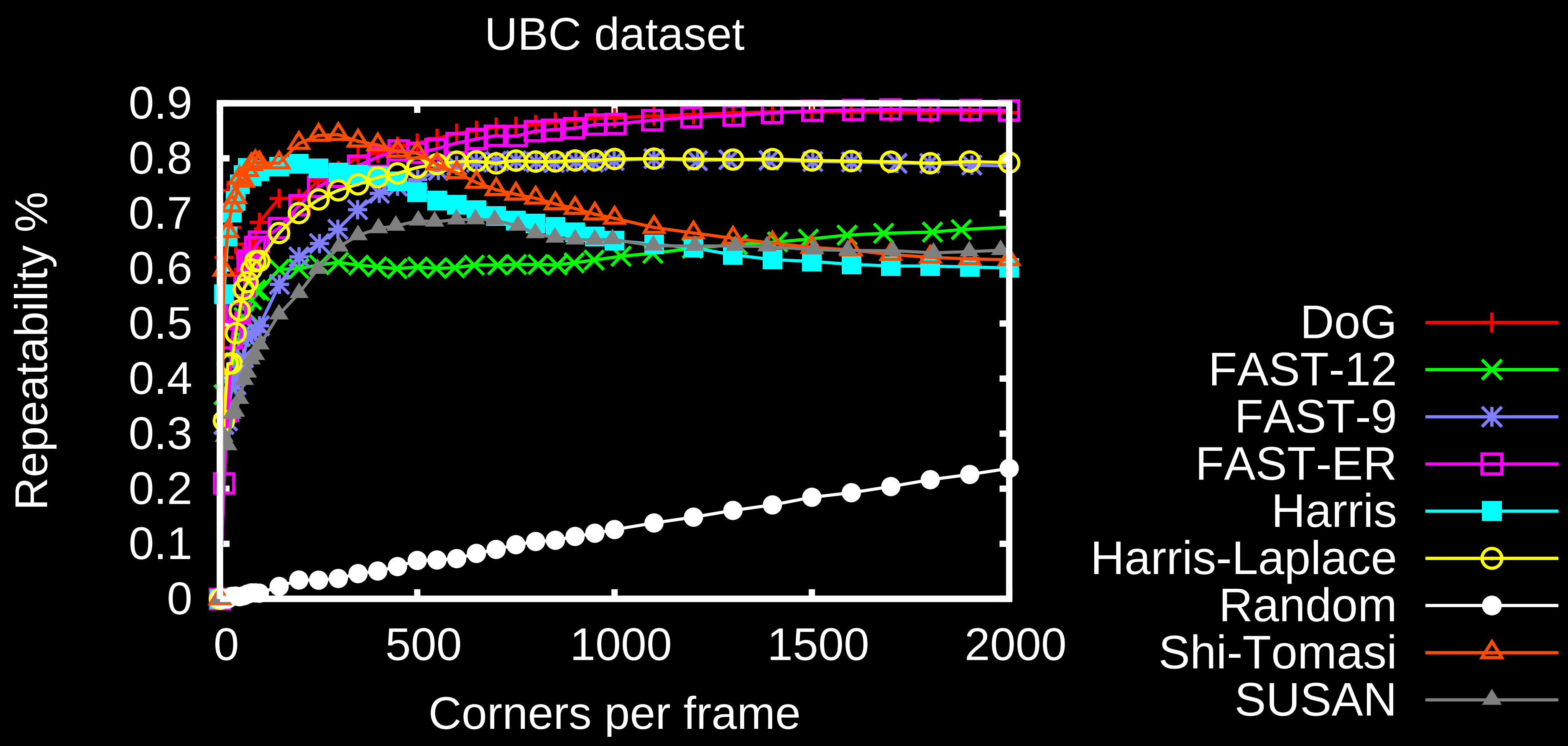
Results: Lighting dataset



Results: Blur (trees) dataset



Results: JPEG compression dataset



Results: Perspective (wall) dataset