MONASH University

Engineering

High speed matching and tracking

Prof Tom Drummond, Monash University

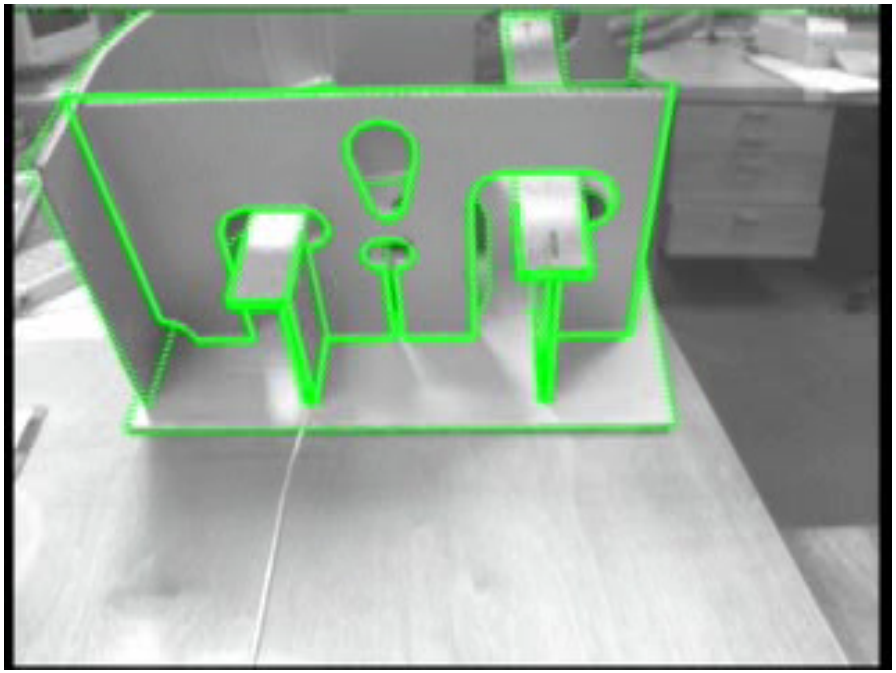
In 1999, I created a 3D tracker

Using a 150 MHz SGI O2
(that also made this video)

Monochrome camera
768x576 @ 25Hz

= 13.5 clock cycles
per pixel

How??



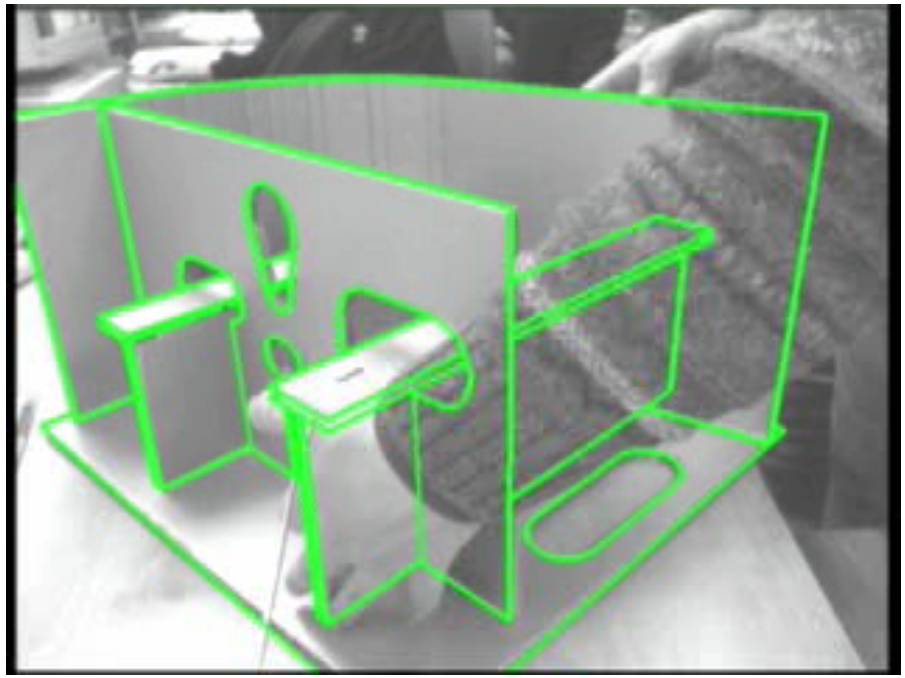
In 1999, I created a 3D tracker

Using a 150 MHz SGI O2
(that also made this video)

Monochrome camera
768x576 @ 25Hz

= 13.5 clock cycles
per pixel

How??



Rule 1: Pixels are expensive

The tracker shown did not look at every pixel

Instead it rendered a surface model and sampled the visible edges (~400 samples)

It searched the neighbourhood of each sample for intensity discontinuities

(+/- 20 pixels in direction perpendicular to edge)

Rule 2: Edges are great!

Edges vs Points

Thought experiment: find the corner of the black region given that it is somewhere in the blue square

Option 1: use corner detection

Cost is proportional to area of blue square (=WxH)

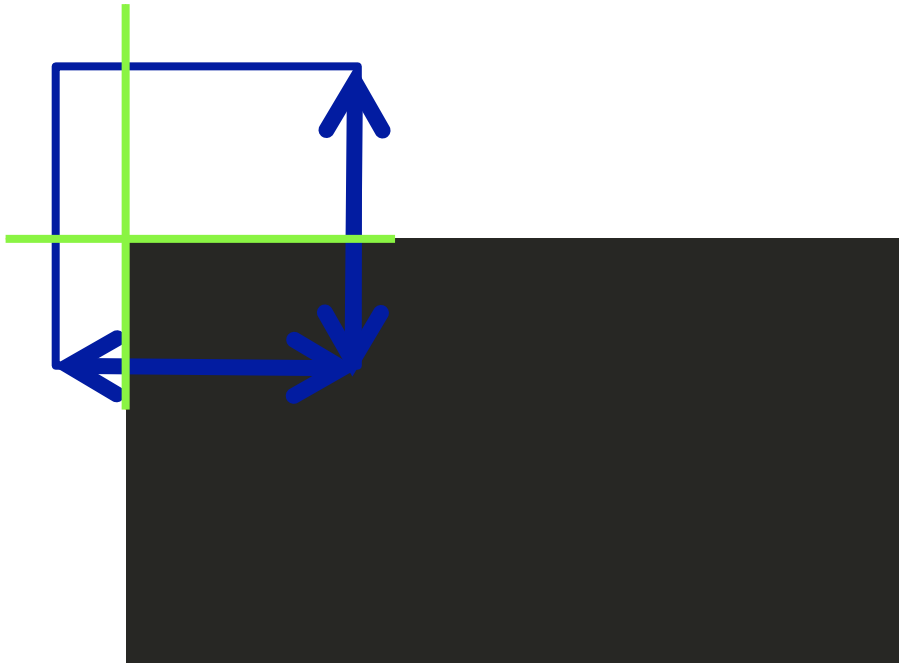
e.g. 40x40 = 1600 pixels

Option 2: use edge detection

Search edges of box for intensity discontinuities

Cost is proportional to width + height

e.g. 40+40 = 80 pixels



Rule 2b: Edge processing is easy

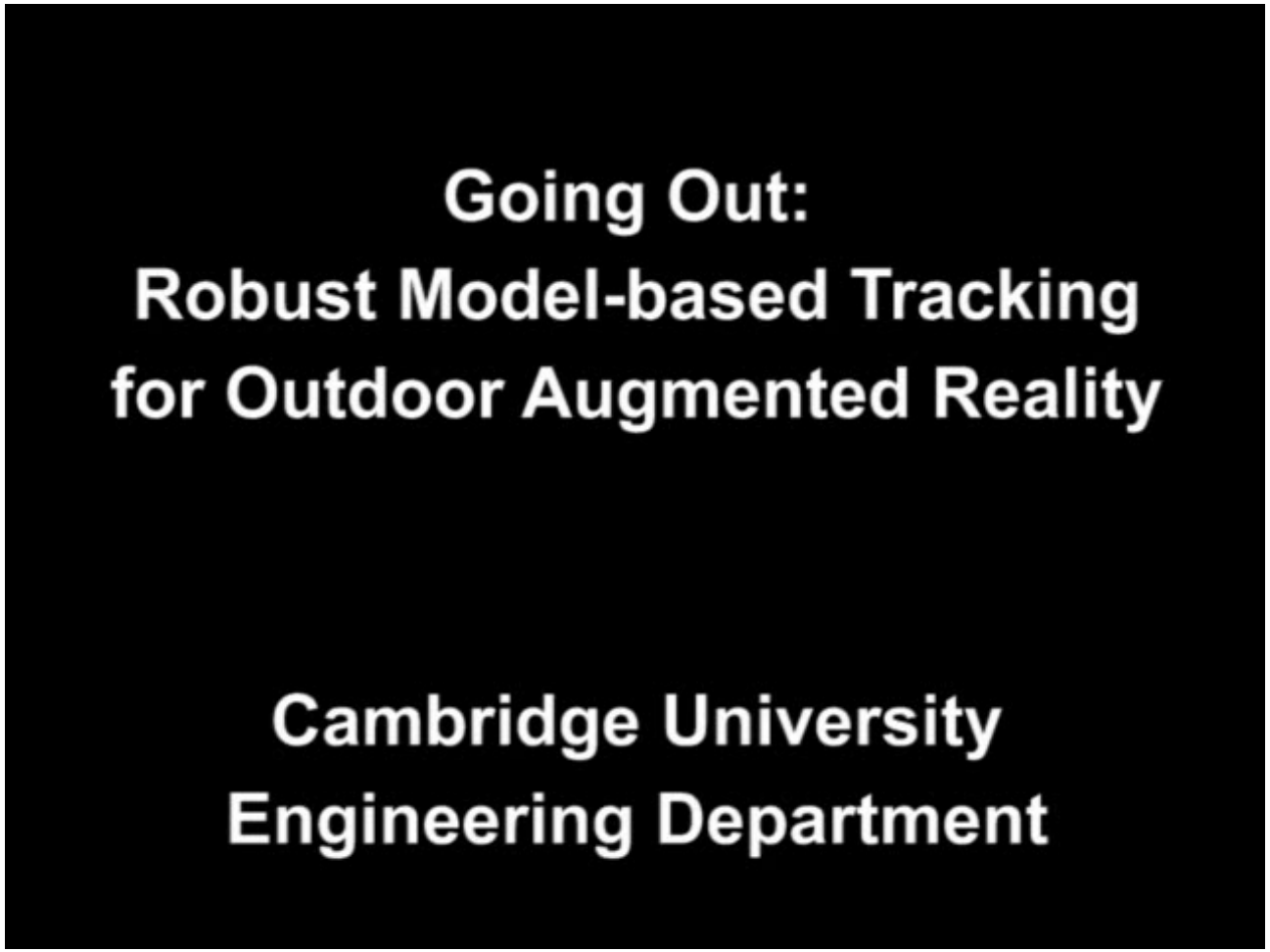
Can do simple edge processing

- search for discontinuities

Or more complex processing

- cross correlate a 1D texture match
- find the texture change point

Example: Going Out



Rapidly constructed appearance models

Build models from video sequence

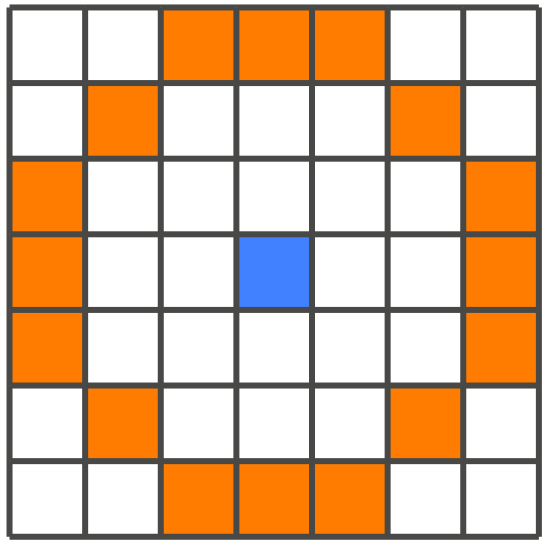
1) Do SLAM/PTAM on sequence to obtain camera poses
2) Extract keyframes
3) User labels planar regions
4) Compute parameters of each plane
5) Extract edge features from each plane
6) Track edge features in live video

Rapidly constructed appearance models

FAST corner detector
Rule 3: If you must use pixels then do as little work as possible per pixel

Original version is now equivalent to FAST-12

blue pixel is a corner iff a contiguous set of 12 pixels in ring are brighter than blue pixel by a threshold (or darker by a threshold)

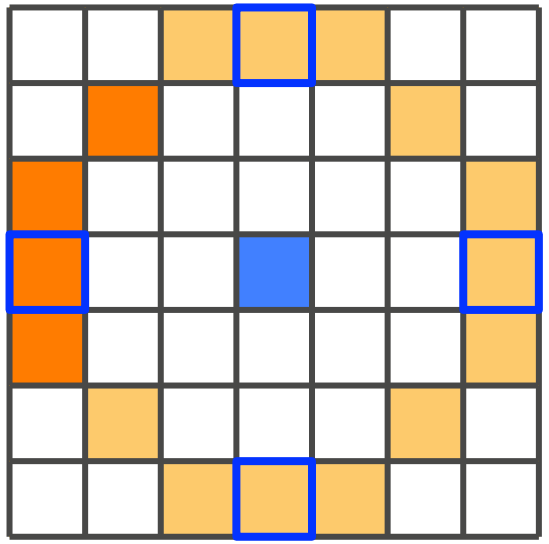


FAST corner detector

Trick: any contiguous set of 12 must include 3 of the 4 pixels at the compass points

Can check those first if any 3 of 4 are brighter than centre + threshold or darker than centre – threshold then
check complete ring

Ed Rosten optimised this with a decision tree and found FAST-9 was best



HIPS feature matcher
Rule 4: Binary feature descriptors are fast

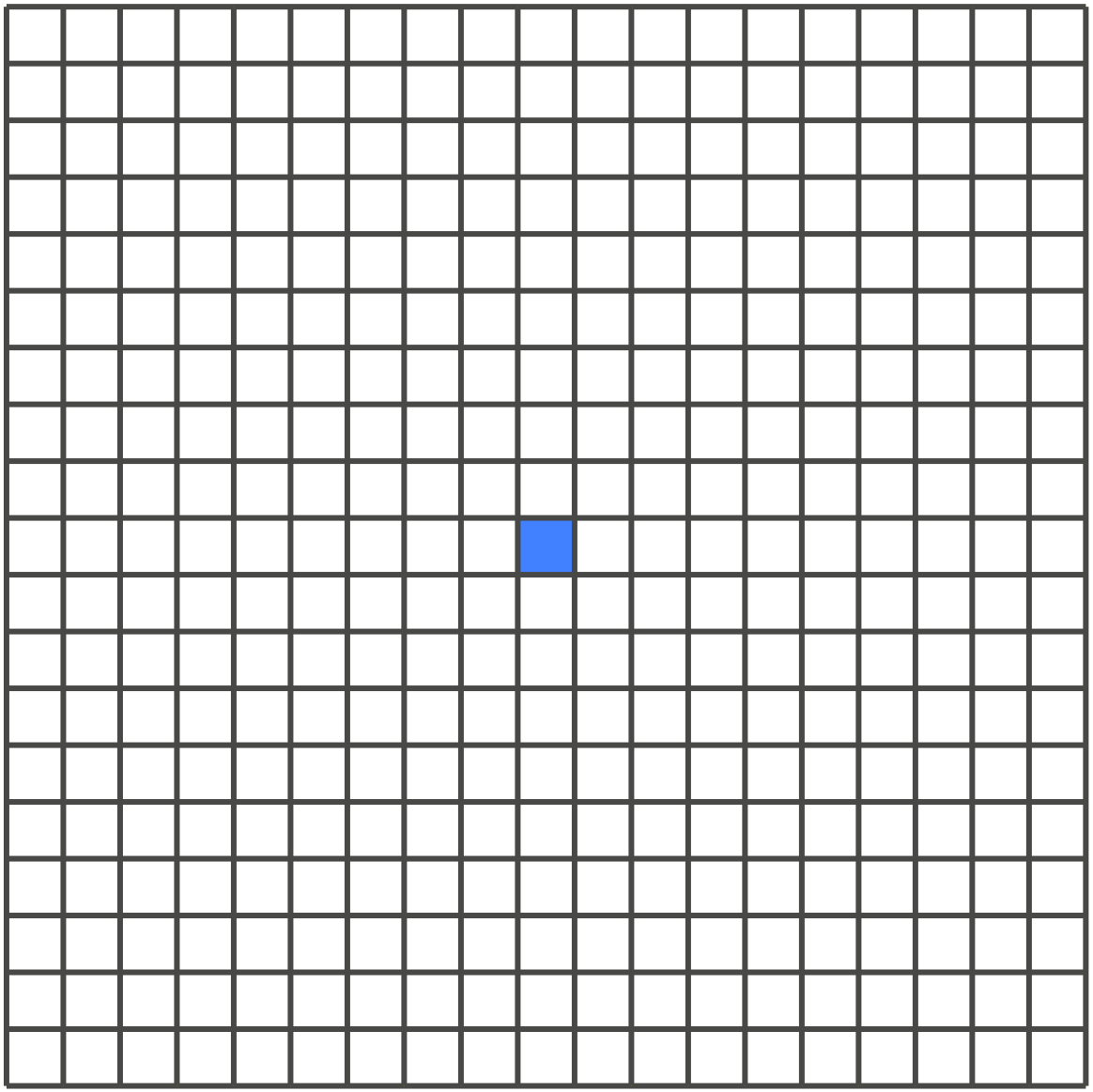
Want to match FAST corners between images.

Descriptors like SIFT are slow(ish) to compute (many multi-scale images with expensive computations per feature)

SIFT is also slow to match (128-D distances)

HIPS

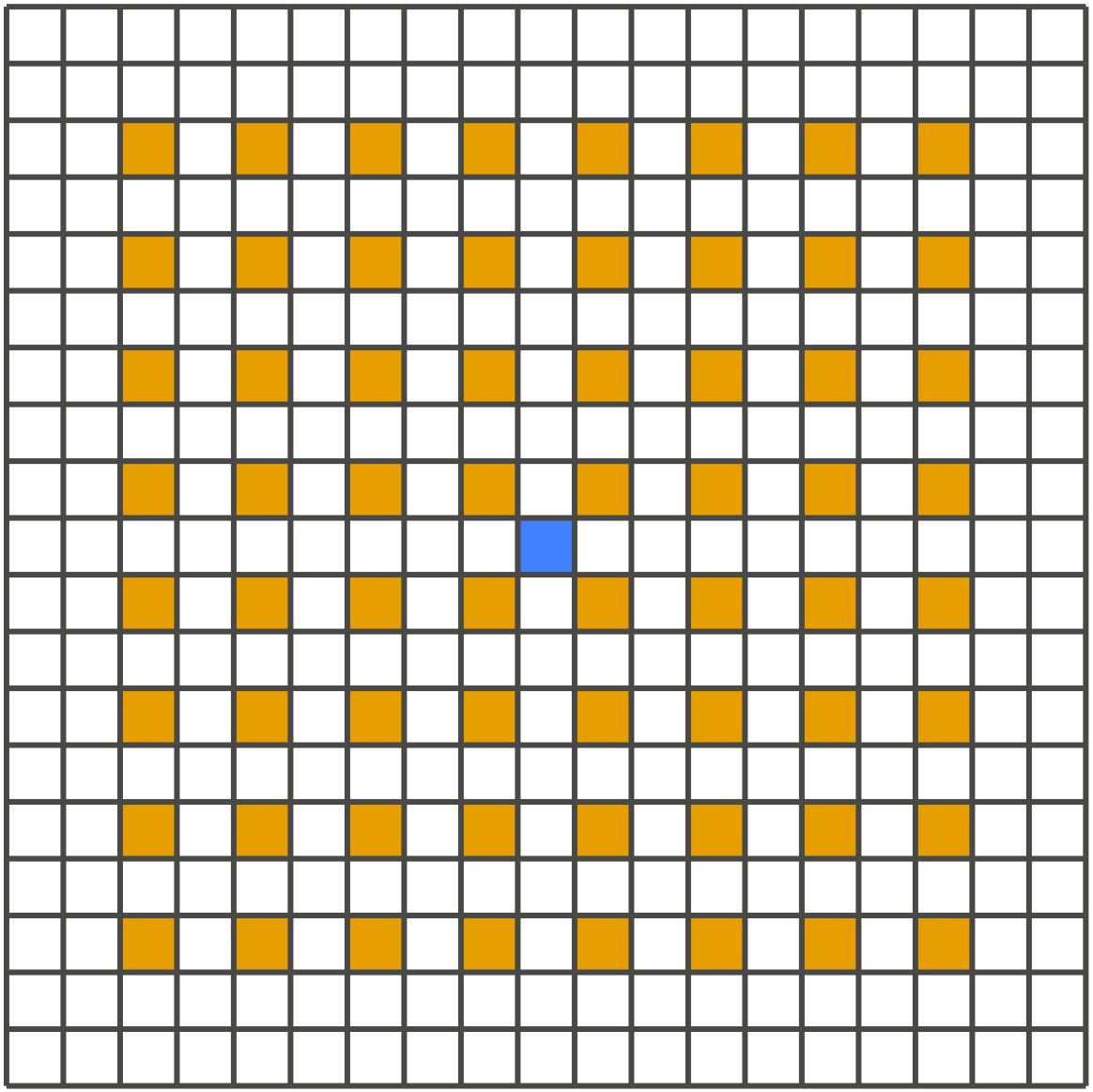
Locate FAST features



HIPS

Locate FAST features

Sample 8x8 grid of pixels

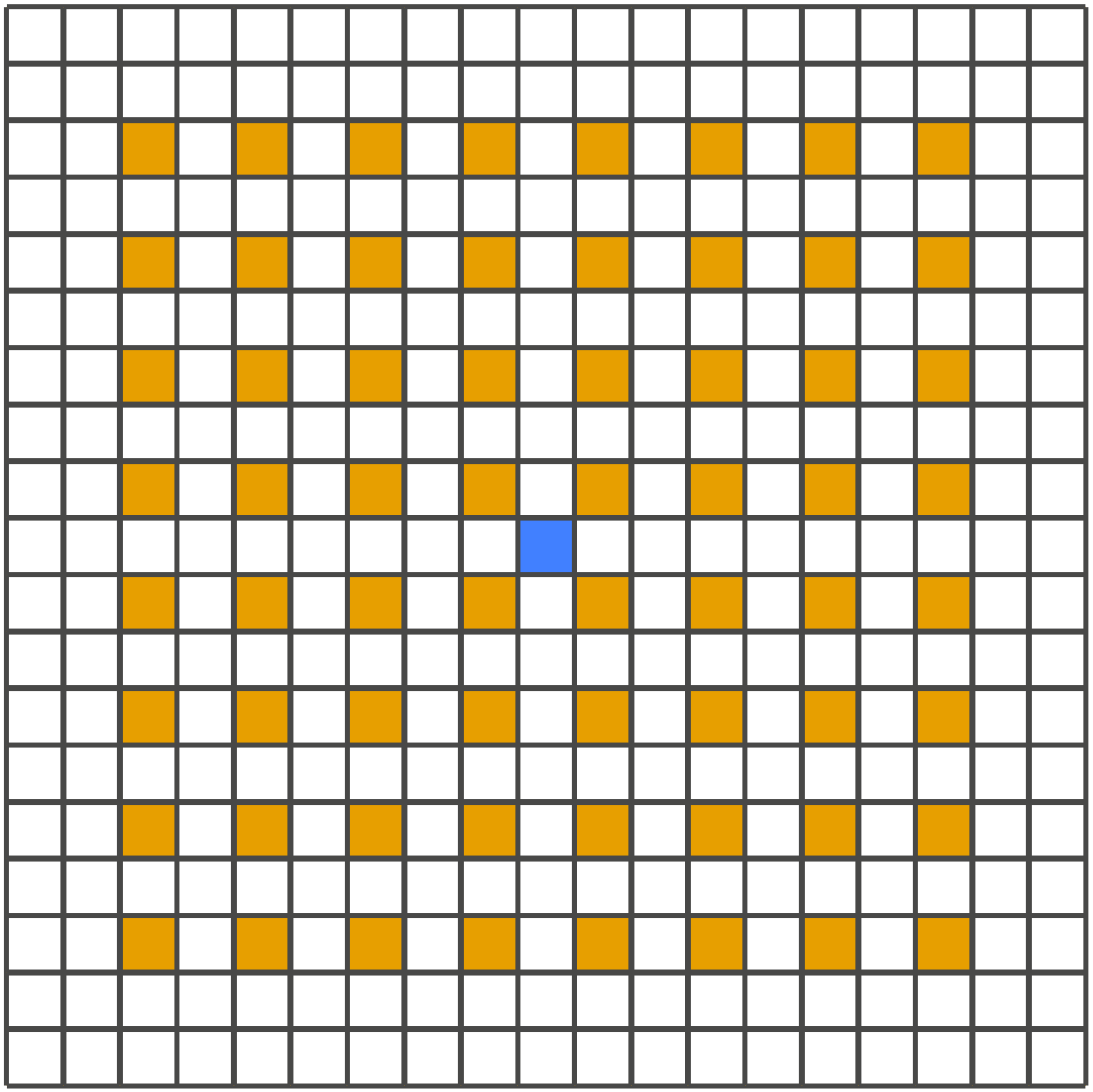


HIPS

Locate FAST features

Sample 8x8 grid of pixels

Compute μ, σ



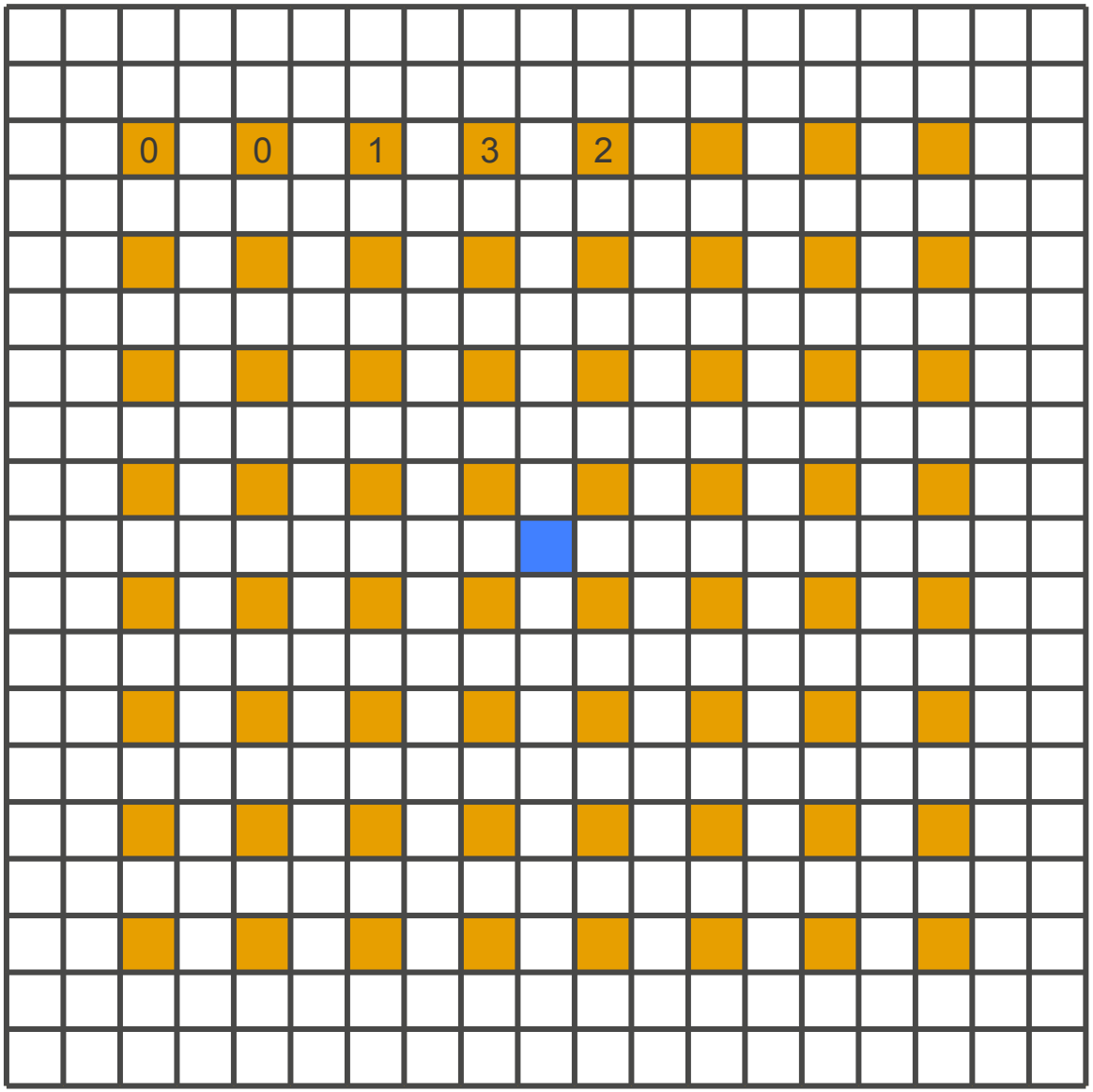
HIPS

Locate FAST features

Sample 8x8 grid of pixels

Compute μ, σ

Quantize into 4 (or 5) levels



HIPS

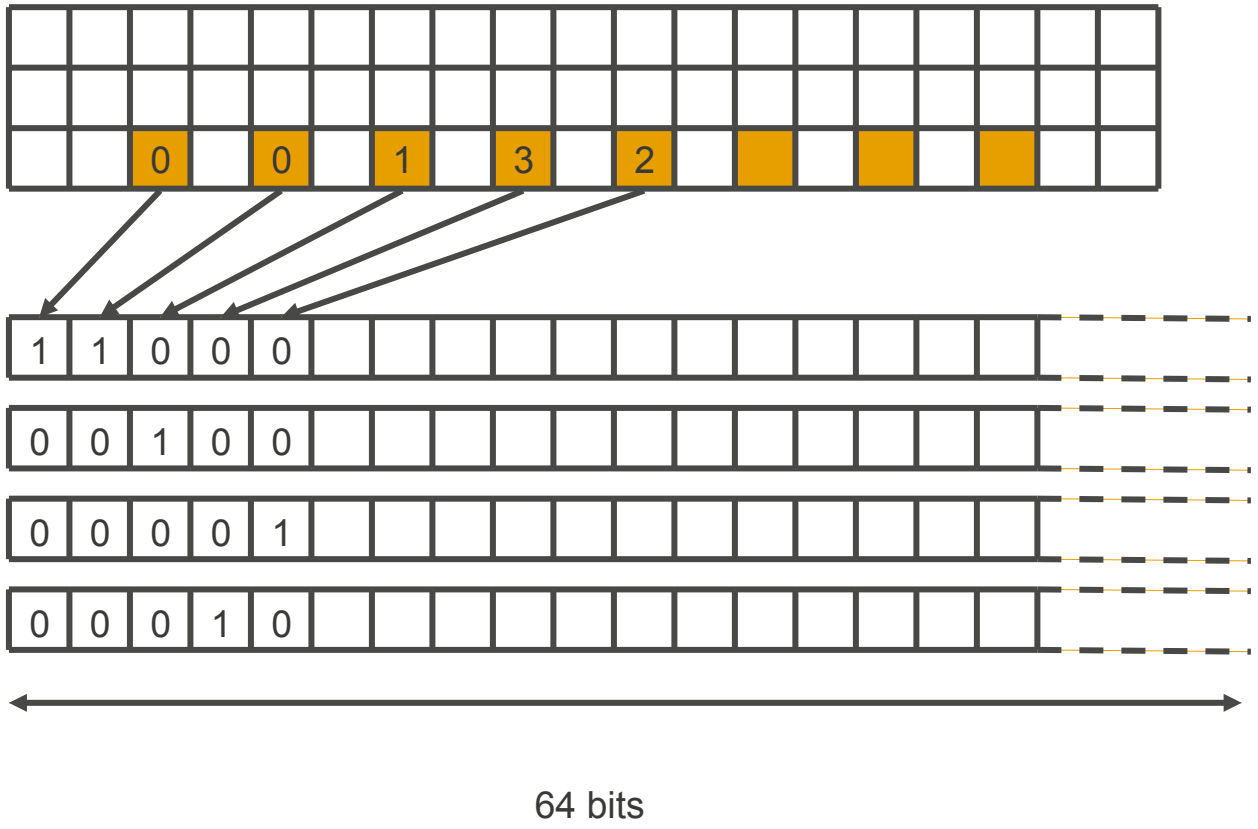
Locate FAST features

Sample 8x8 grid of pixels

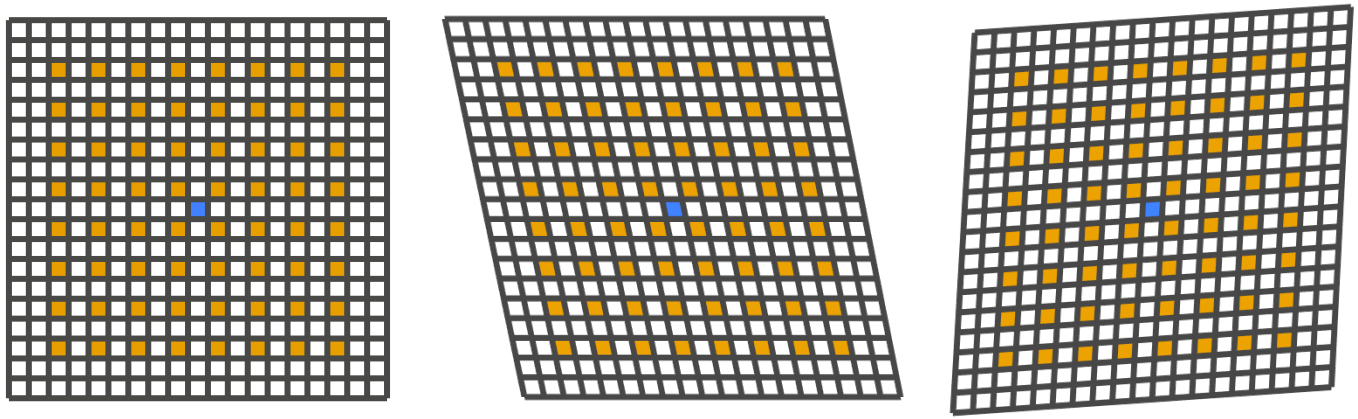
Compute μ, σ

Quantize into 4 (or 5) levels

Build descriptors



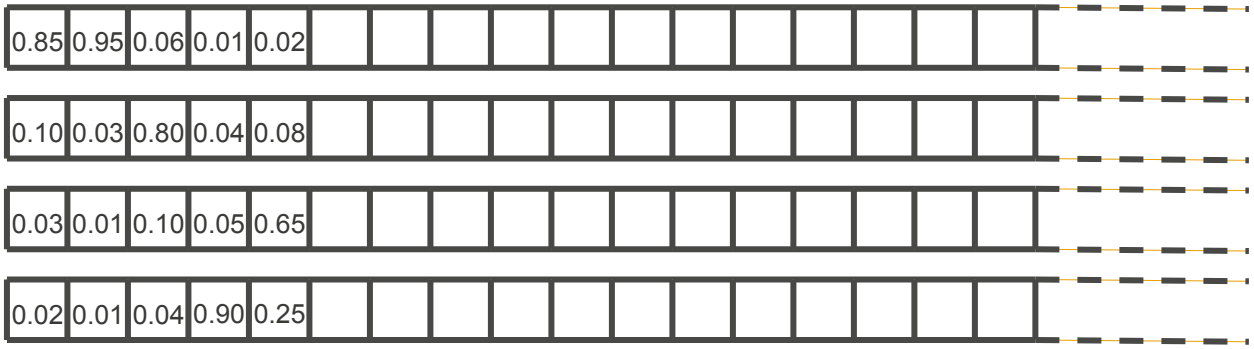
HIPS



To build reference descriptor, sample many affine warps and calculate average descriptor

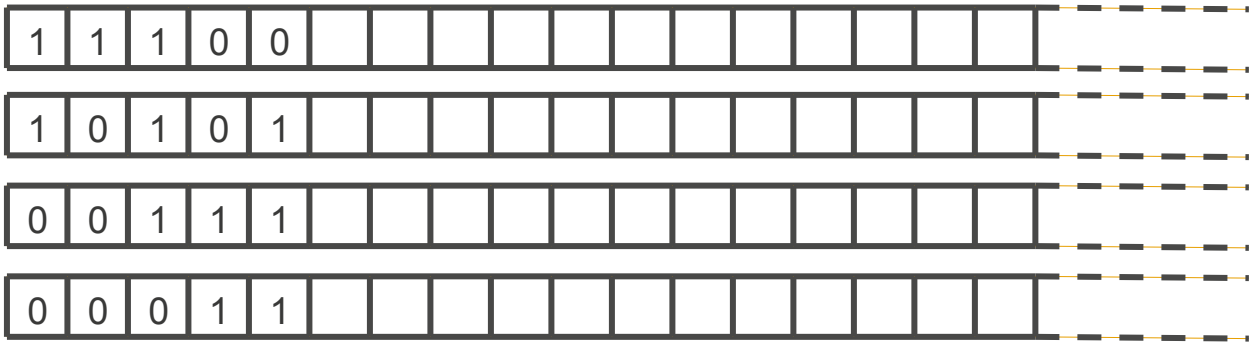
0.85	0.95	0.06	0.01	0.02											
0.10	0.03	0.80	0.04	0.08											
0.03	0.01	0.10	0.05	0.65											
0.02	0.01	0.04	0.90	0.25											

HIPS



threshold at 0.05:

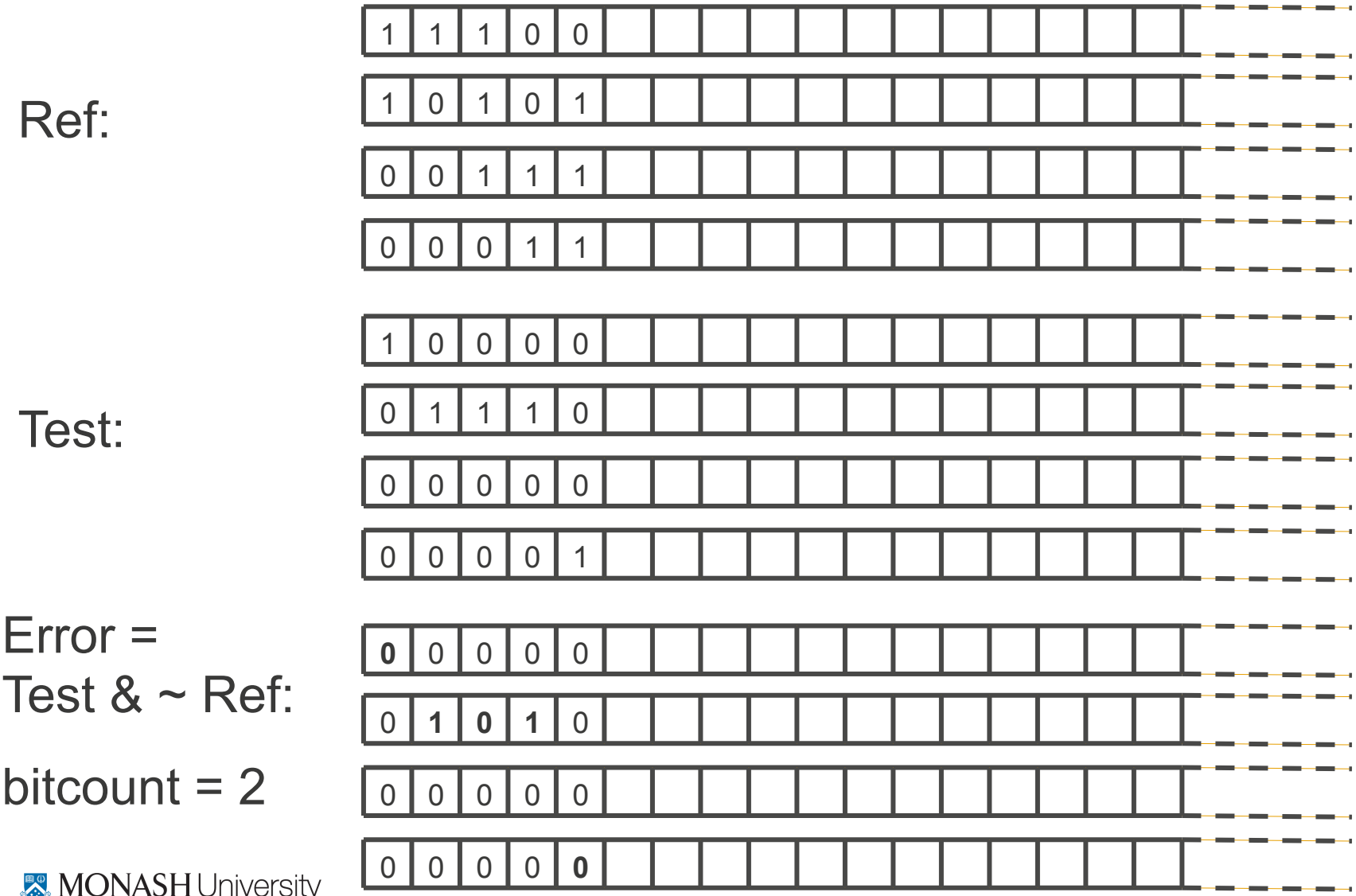
Reference descriptor:



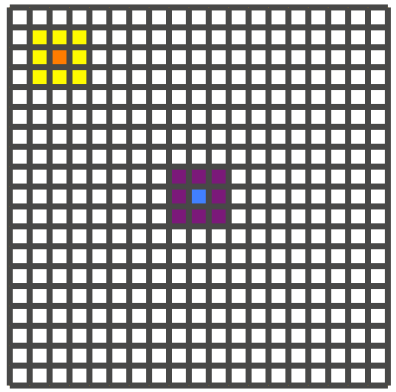
A ‘1’ means likely to occur, ‘0’ means rarely occurs

(NB – this is the inverse of the original paper)

To match count how many rare bits occur in ‘Test’:



Rule 5: Near enough is good enough



Replace warps with 9 sampled descriptors

Compute descriptors for 8 neighbours of feature point

simply OR all 9 descriptors together

This means that each pixel in the descriptor is also sampled from its 8 neighbouring positions

This is a close approximation to the places it would be sampled from with the warping approach

Because the descriptor treats each pixel independently, this is enough

Rule 6: Trees are good

To build an index of many reference descriptors, observe that:

bitcount (test & ~ (ref1 | ref2)) < bitcount (test & ~ ref1)

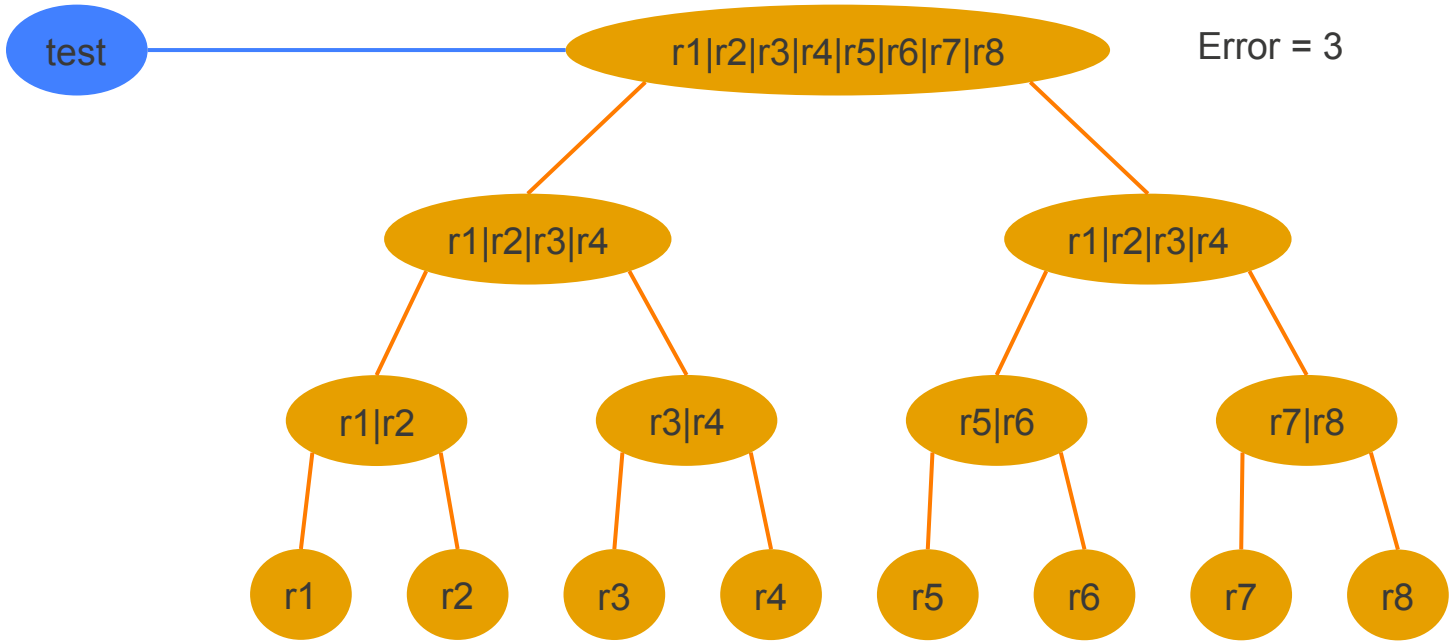
ie ORing two reference descriptors together creates a new descriptor for which the error of a test descriptor is a lower bound on the errors for ref1 and ref2.

This means that if bitcount (test & ~ (ref1 | ref2)) exceeds a threshold then it is not necessary to compute the error for either ref1 or ref2.

Rule 6: Trees are good

Can use this to build a tree by pairing similar descriptors

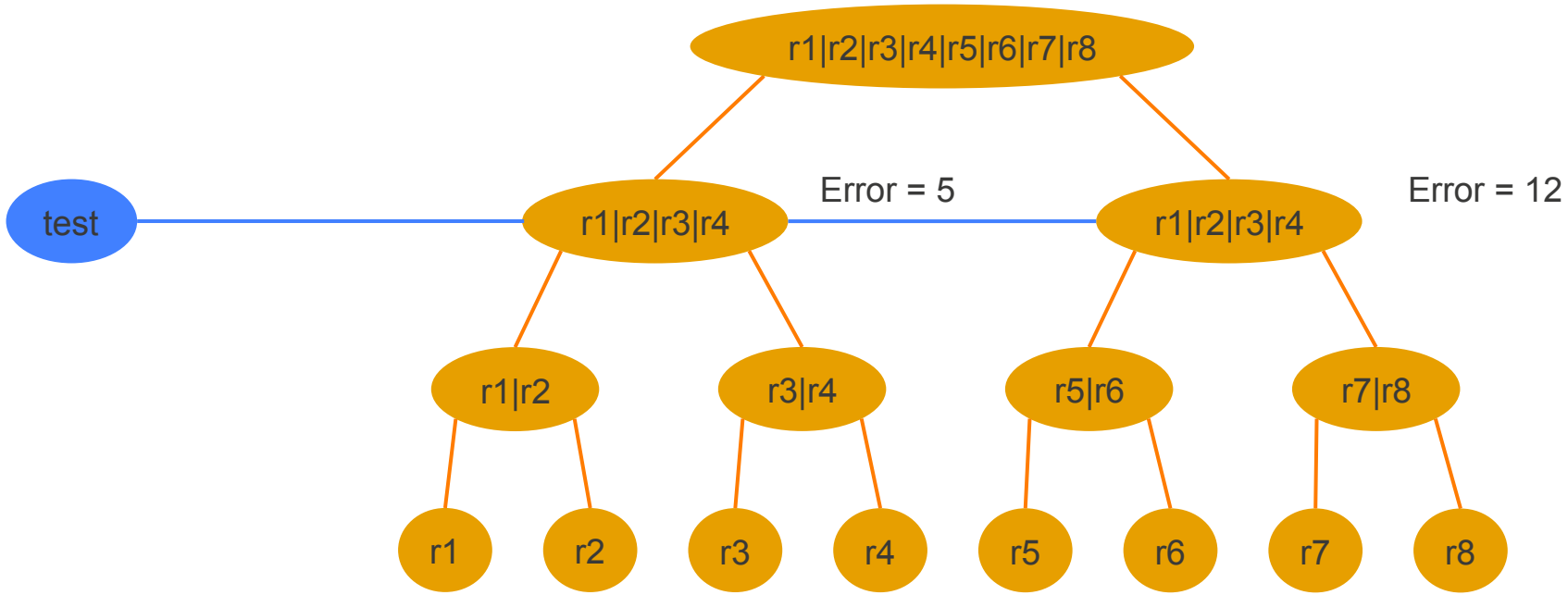
Find a match by starting at the top (max error = 10)



Rule 6: Trees are good

Can use this to build a tree by pairing similar descriptors

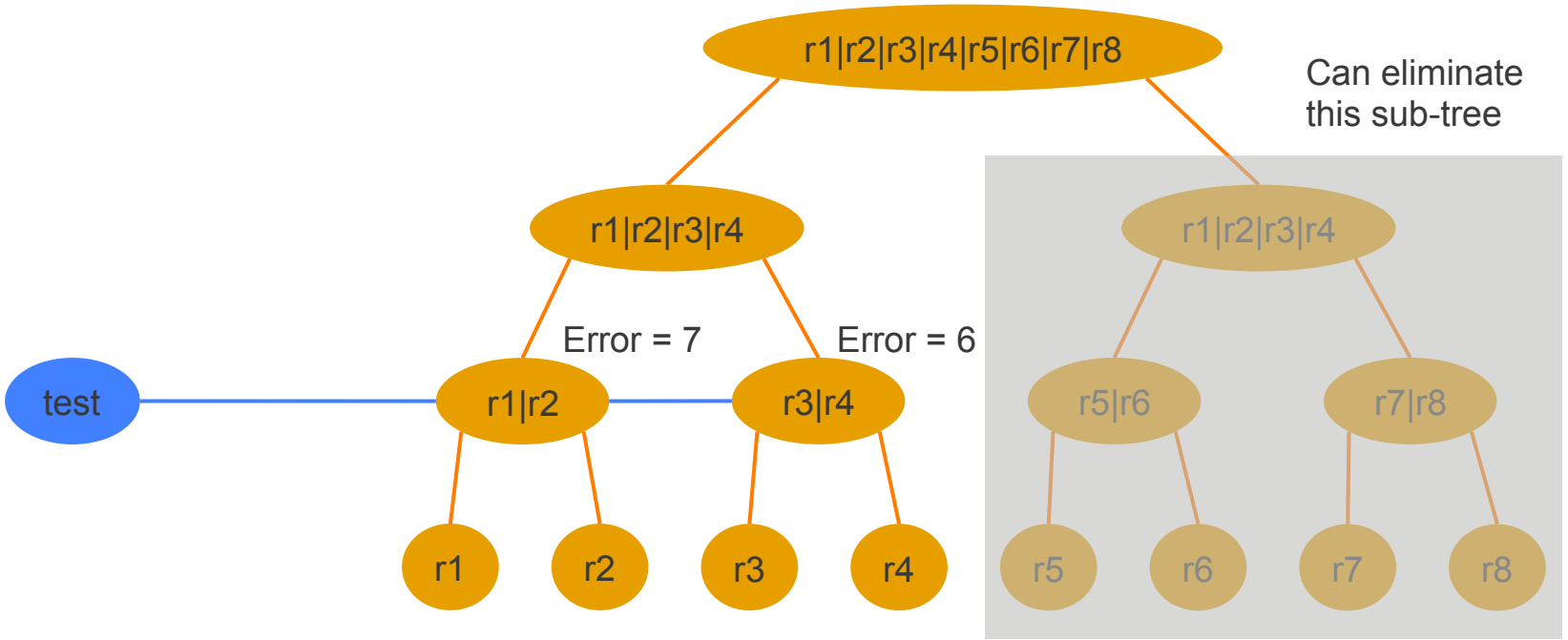
Find a match by starting at the top (max error = 10)



Rule 6: Trees are good

Can use this to build a tree by pairing similar descriptors

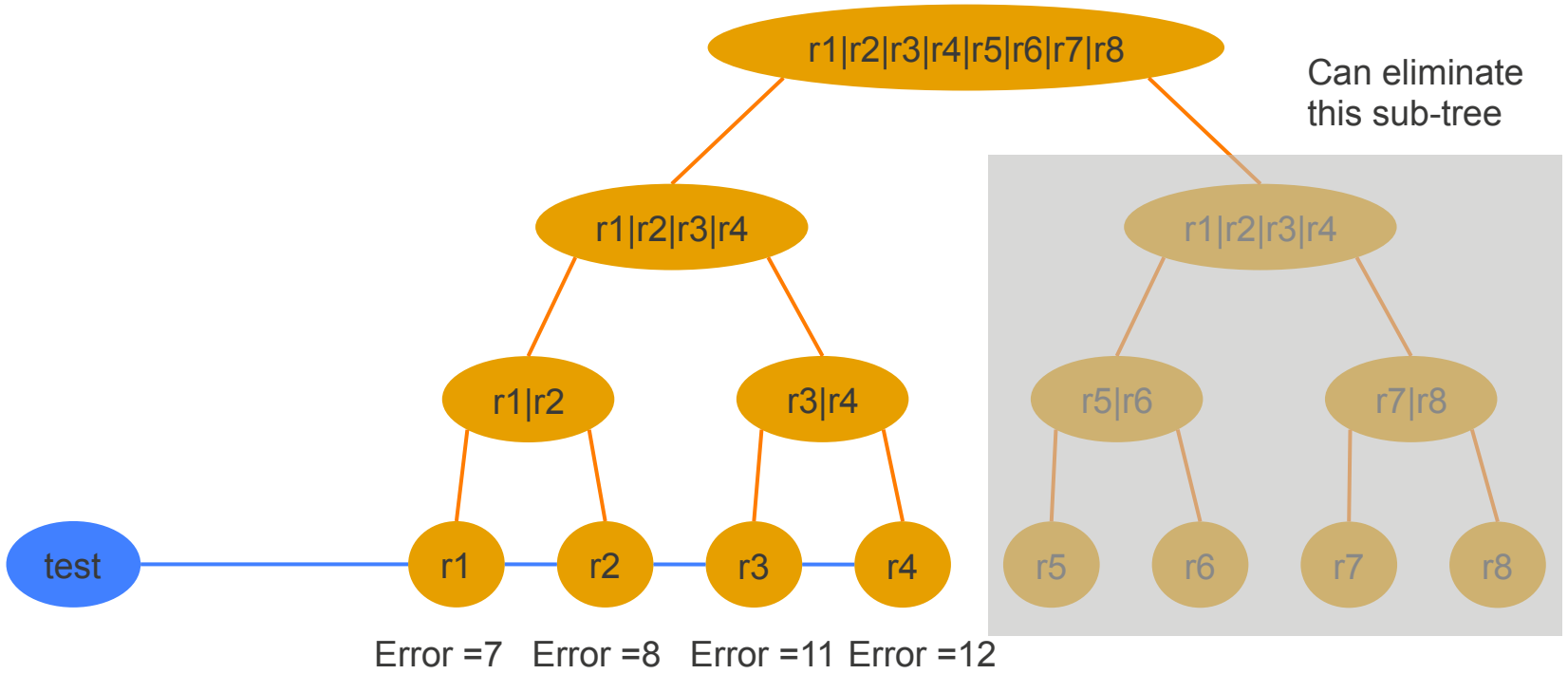
Find a match by starting at the top (max error = 10)



Rule 6: Trees are good

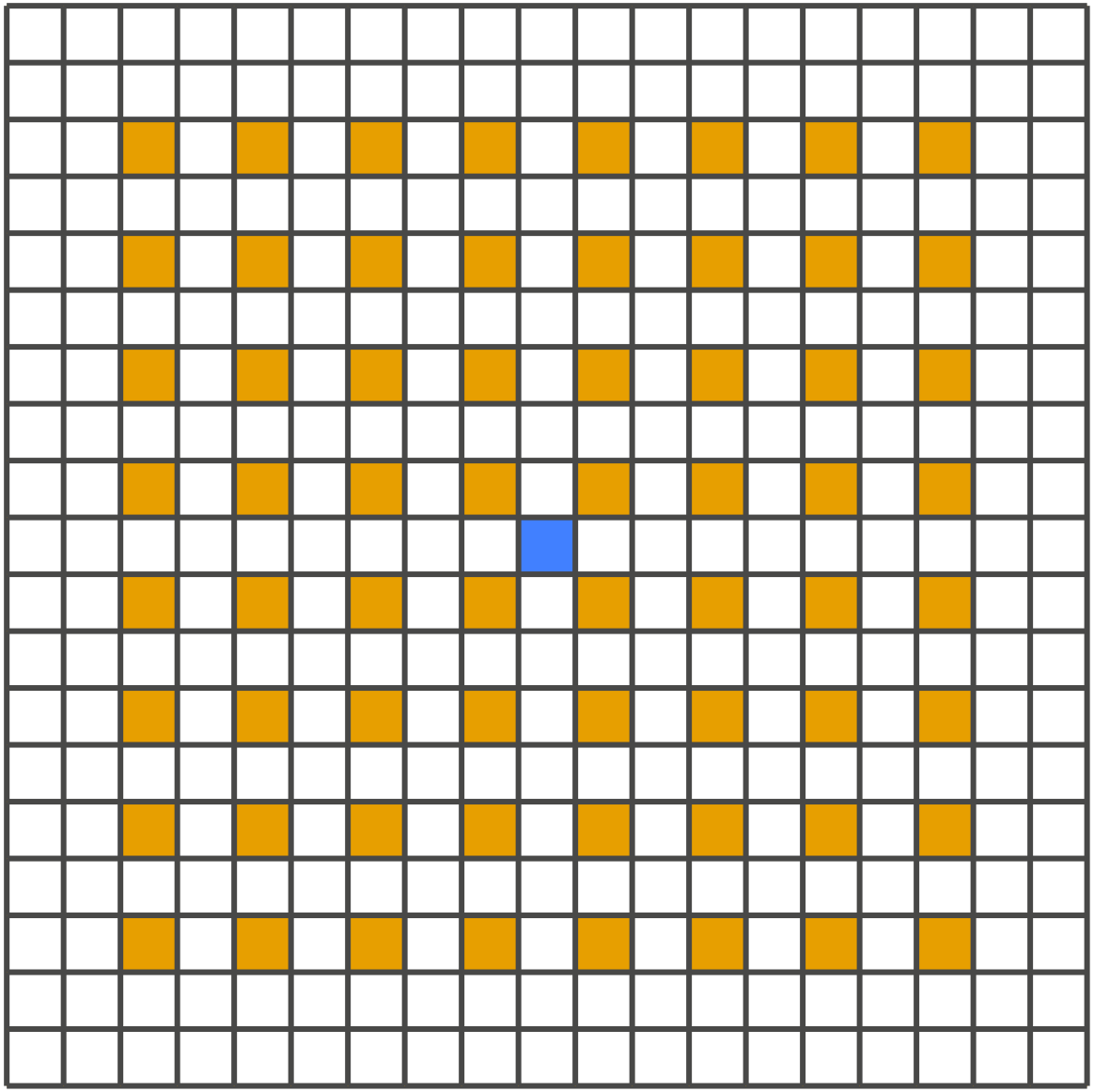
Can use this to build a tree by pairing similar descriptors

Find a match by starting at the top (max error = 10)



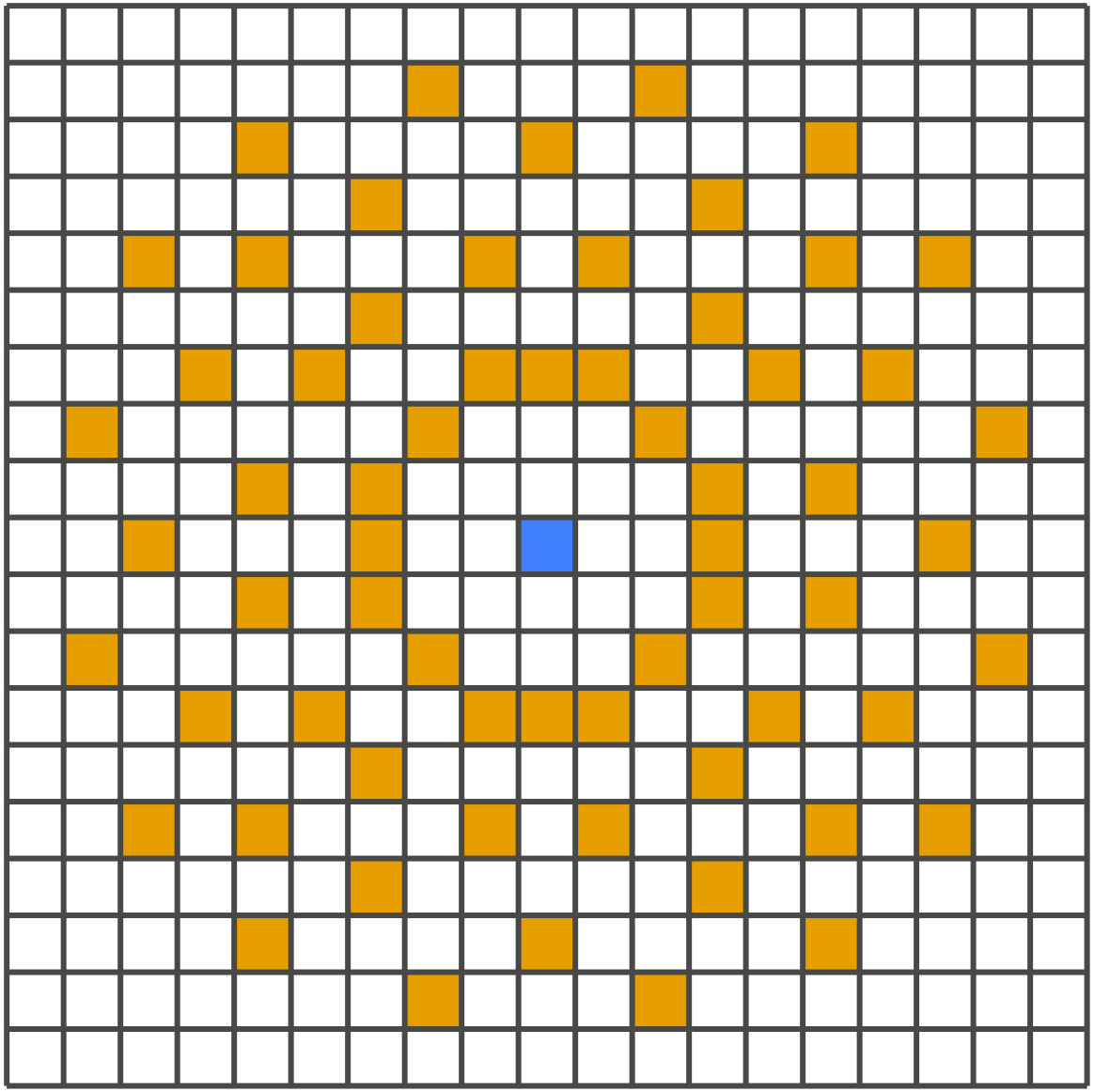
Rule 7: Rotations can be expensive

Replace standard pattern:



Rule 7: Rotations can be expensive

RHIPS introduces a circular pattern for HIPS

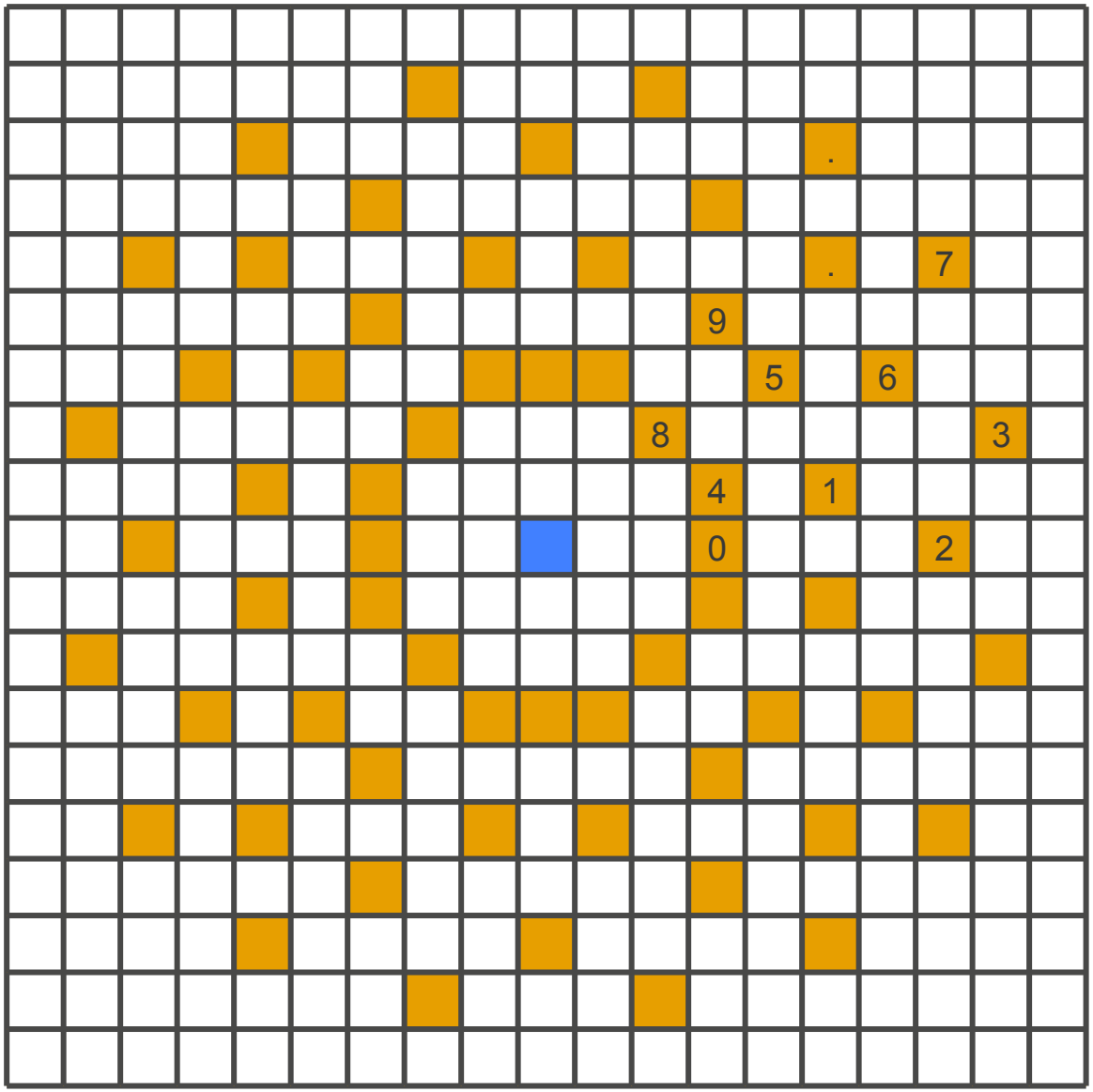


Rule 7: Rotations can be expensive

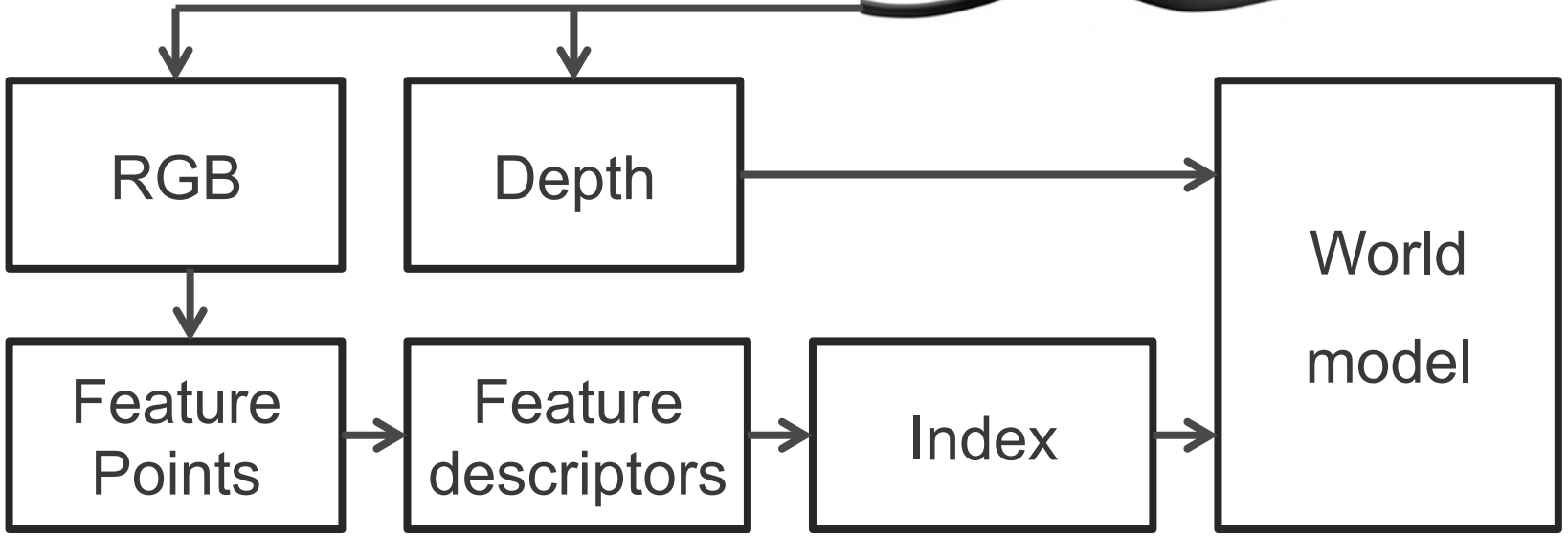
RHIPS introduces a circular pattern for HIPS

Sampled in order shown:

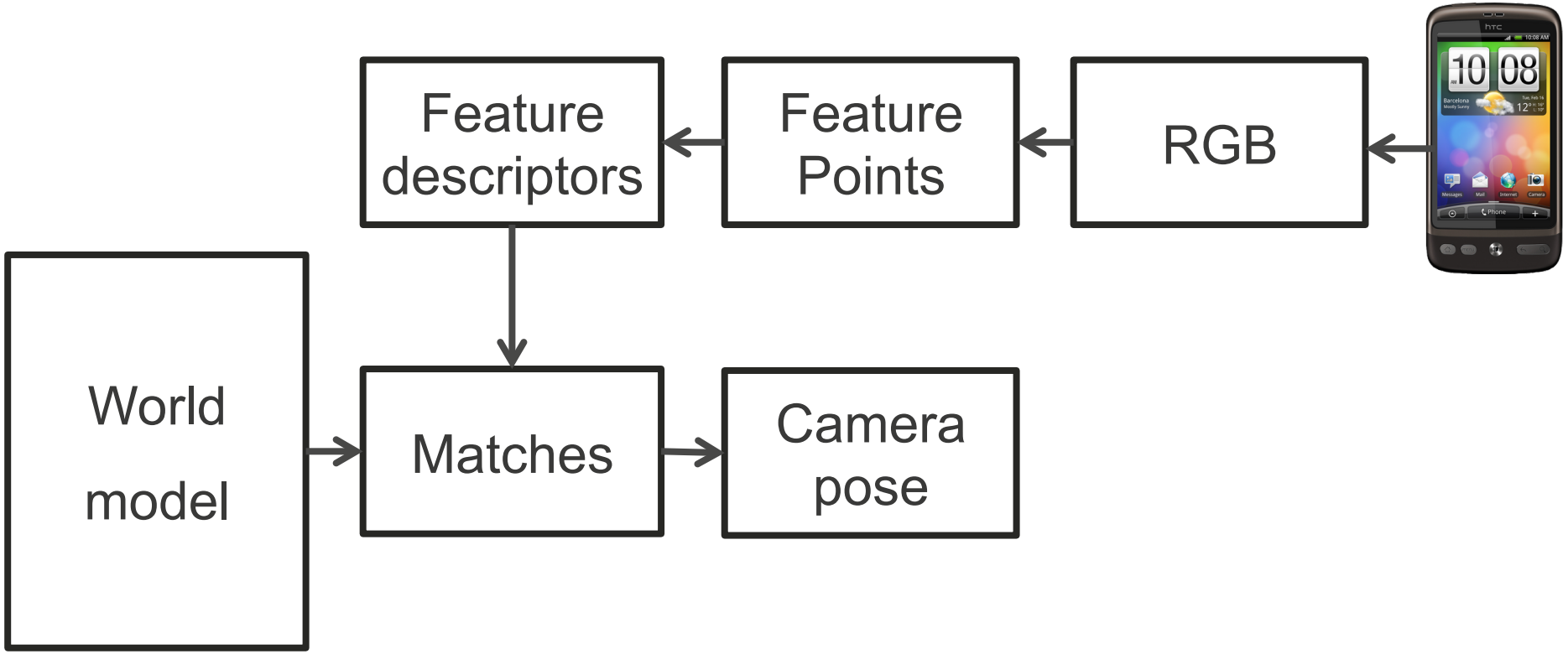
Barrel shift of descriptor by 4 bits = rotation by $\pi/8$



Visual localisation of a robot from an external RGB-D sensor

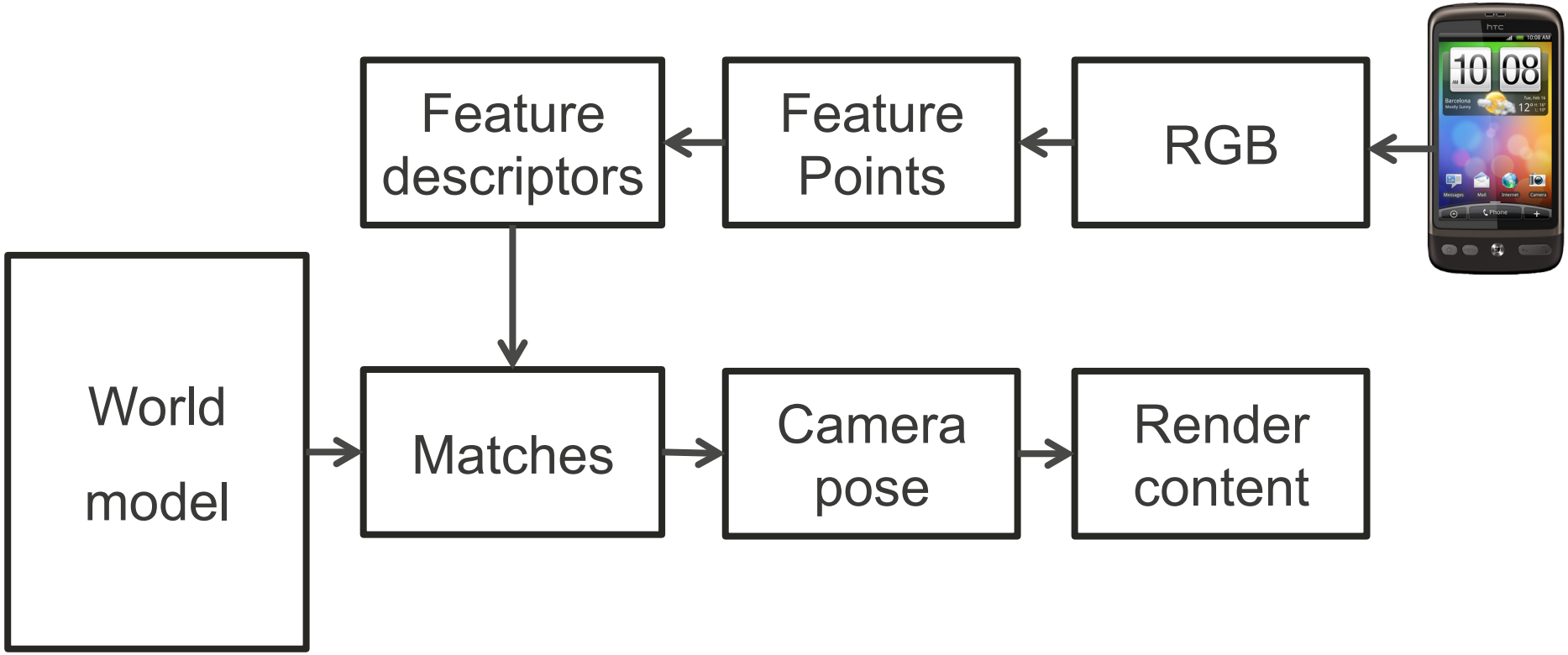


Using the world model



Example:

Use kinect model for Augmented Reality



Rule 8: Inverse depth coordinates can be handy

Represent changes in coordinate frame by a matrix acting on $(x\ y\ z\ 1)^T$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} R & t \\ 000 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Can treat this as acting on projective coordinates (any non zero multiple of a vector is treated as being the same vector)

$$\begin{pmatrix} \alpha x' \\ \alpha y' \\ \alpha z' \\ \alpha \end{pmatrix} \equiv \begin{bmatrix} R & t \\ 000 & 1 \end{bmatrix} \begin{pmatrix} \lambda x \\ \lambda y \\ \lambda z \\ \lambda \end{pmatrix}$$

Inverse depth coordinates

Can choose $\lambda = 1/z$

$$\begin{pmatrix} \alpha x' \\ \alpha y' \\ \alpha z' \\ \alpha \end{pmatrix} \equiv \begin{bmatrix} R & t \\ 000 & 1 \end{bmatrix} \begin{pmatrix} x/z \\ y/z \\ 1 \\ 1/z \end{pmatrix}$$

But x/z = u (normalized camera x coordinate)
and y/z = v (normalized camera y coordinate)
also write q = 1/z (inverse depth)

$$\begin{pmatrix} u' \\ v' \\ 1 \\ q' \end{pmatrix} \equiv \begin{bmatrix} R & t \\ 000 & 1 \end{bmatrix} \begin{pmatrix} u \\ v \\ 1 \\ q \end{pmatrix}$$

So transformation matrix can apply to coordinates expressed in terms of u v and q

Computing pose between kinect and mobile camera

These come from coordinates of feature in mobile camera image

These come from coordinates of feature in Kinect RGB image

$$\begin{pmatrix} u' \\ v' \\ 1 \\ q' \end{pmatrix} \equiv \begin{bmatrix} R & t \\ 000 & 1 \end{bmatrix} \begin{pmatrix} u \\ v \\ 1 \\ q \end{pmatrix}$$

This is unknown, so only 2 constraints per correspondence

This comes directly from Kinect depth image

Rule 9: Naïve RANSAC sucks
(unless you have a GPU)

In this example we used a simple RANSAC implementation.

This is by far the slowest part of the code and should be replaced by PROSAC or DETSAC

Although we have a GPU implementation that is very fast because the naïve blindness (lack of conditional code) of the algorithm means that it maps well onto OpenCL

So Rule 9 is really that Rules 1-8 often don't apply if you are using a GPU (pixels can be cheap, FAST is implemented very differently, etc)

Application: Robot localisation

ACRA 2011 - Video Submission for paper titled

Visual Localisation of a Robot with an External RGB-D Sensor

Winston Yii
Nalika Damayanthi
Wai Ho Li
Tom Drummond

Monash University

Contact
{winston.yii, nalika.dona, wai.ho.li, tom.drummond}@monash.edu

Application: Augmented Reality