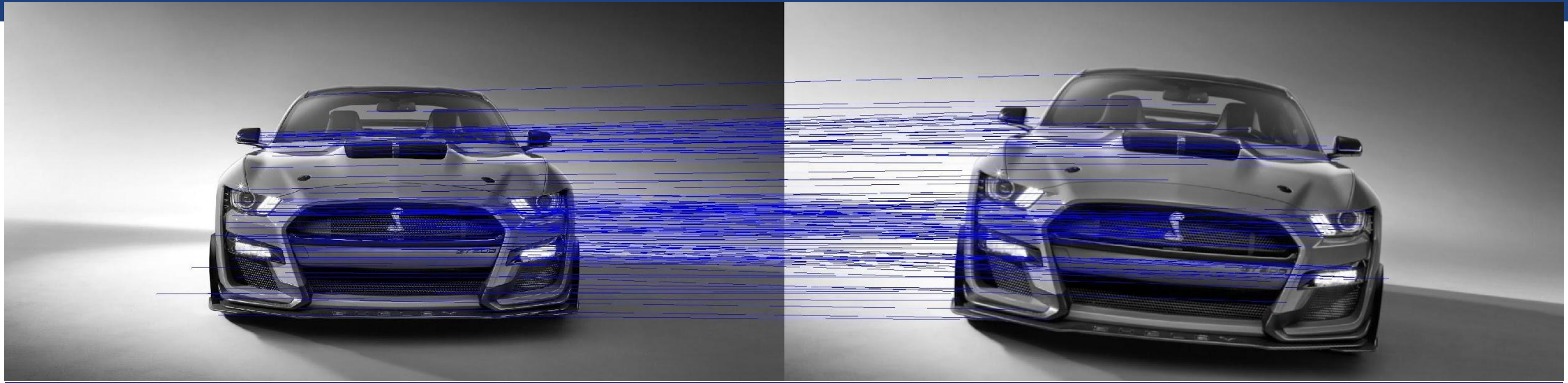
ETH zürich

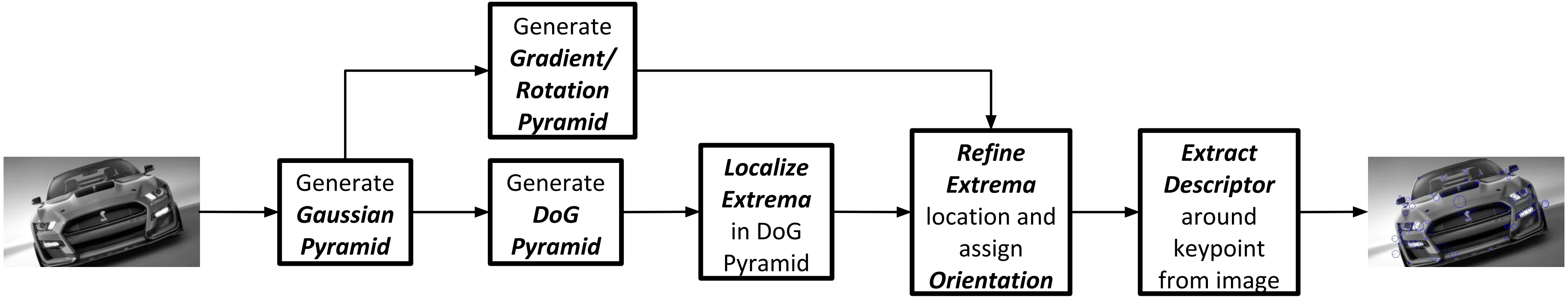


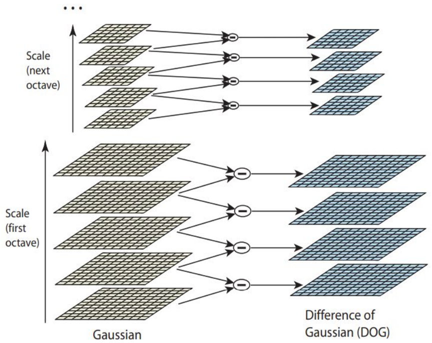
A SIFT Descriptor for Feature Matching

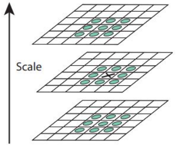
Advanced Systems Lab

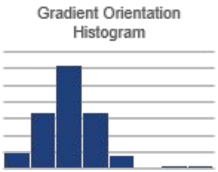
Nicolas Hafner, Costanza Maria Improta, Zsombor Kalotay, Jan Leutwyler

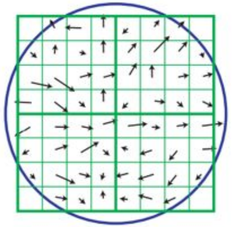
Our Algorithm





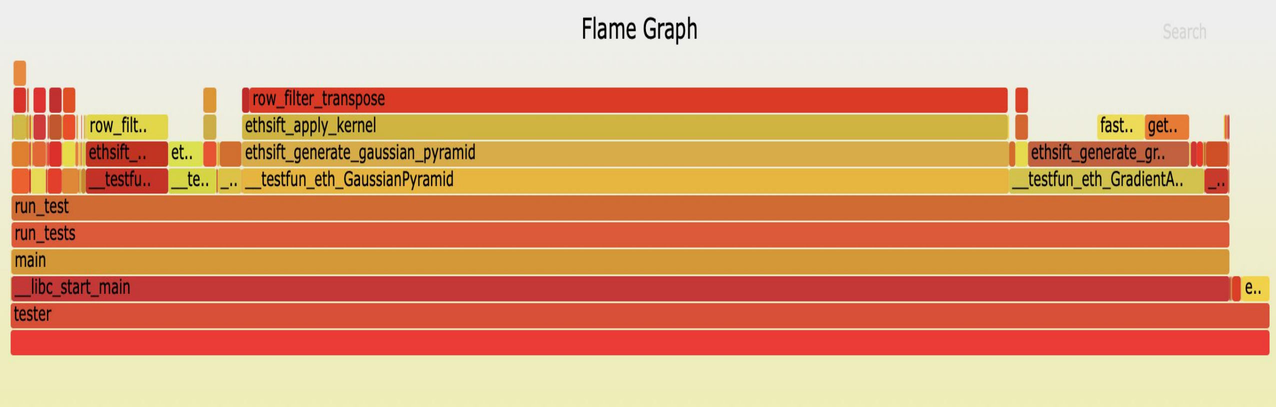


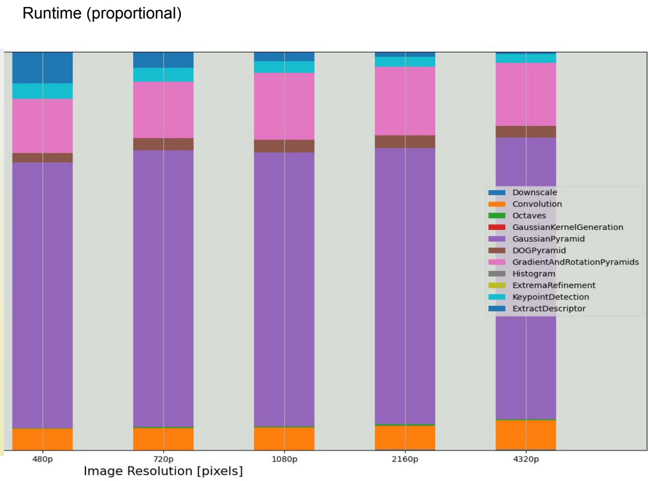




Cost Analysis and Profiling

- ***Flop count*** and
  memory accesses

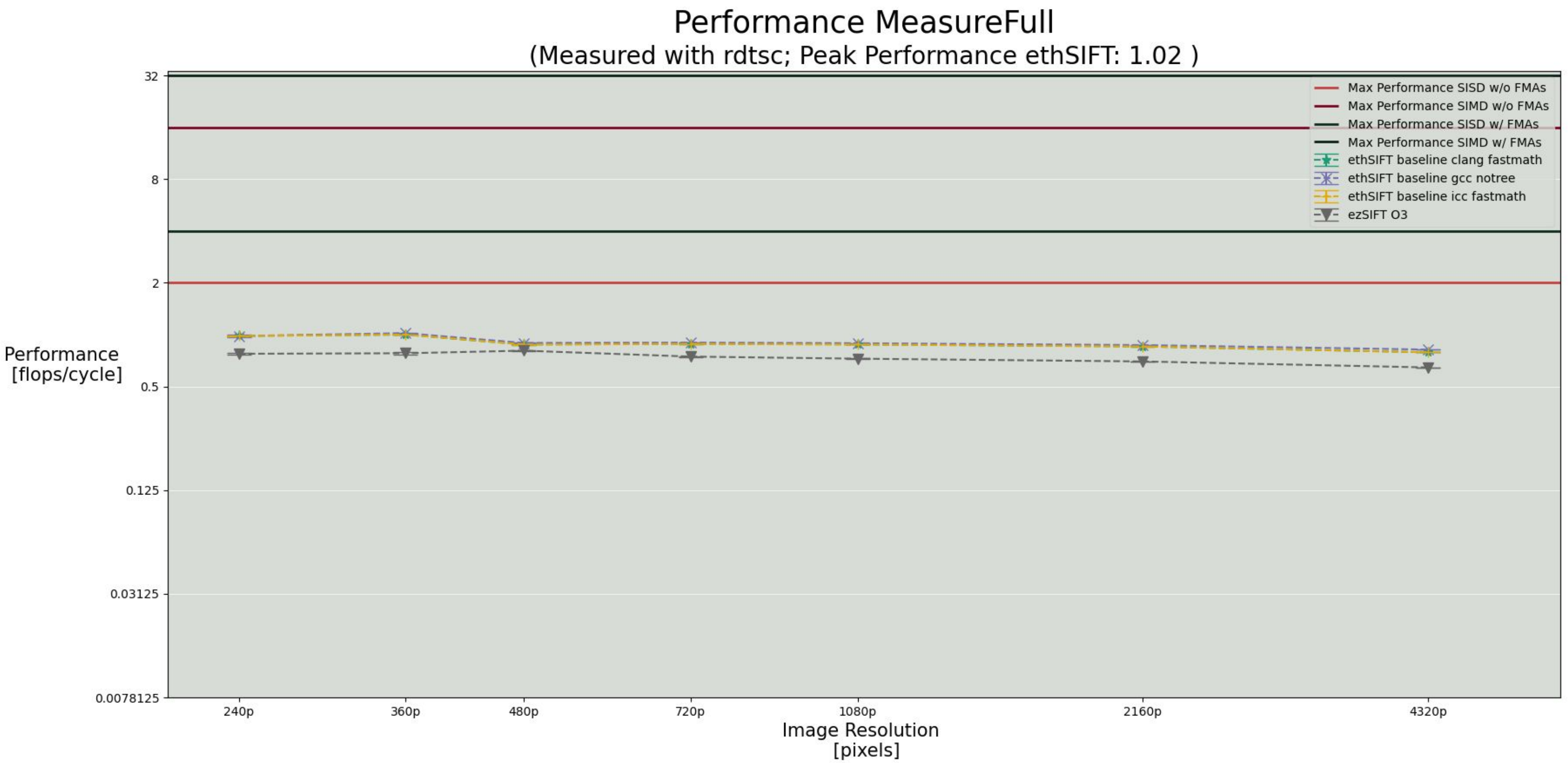




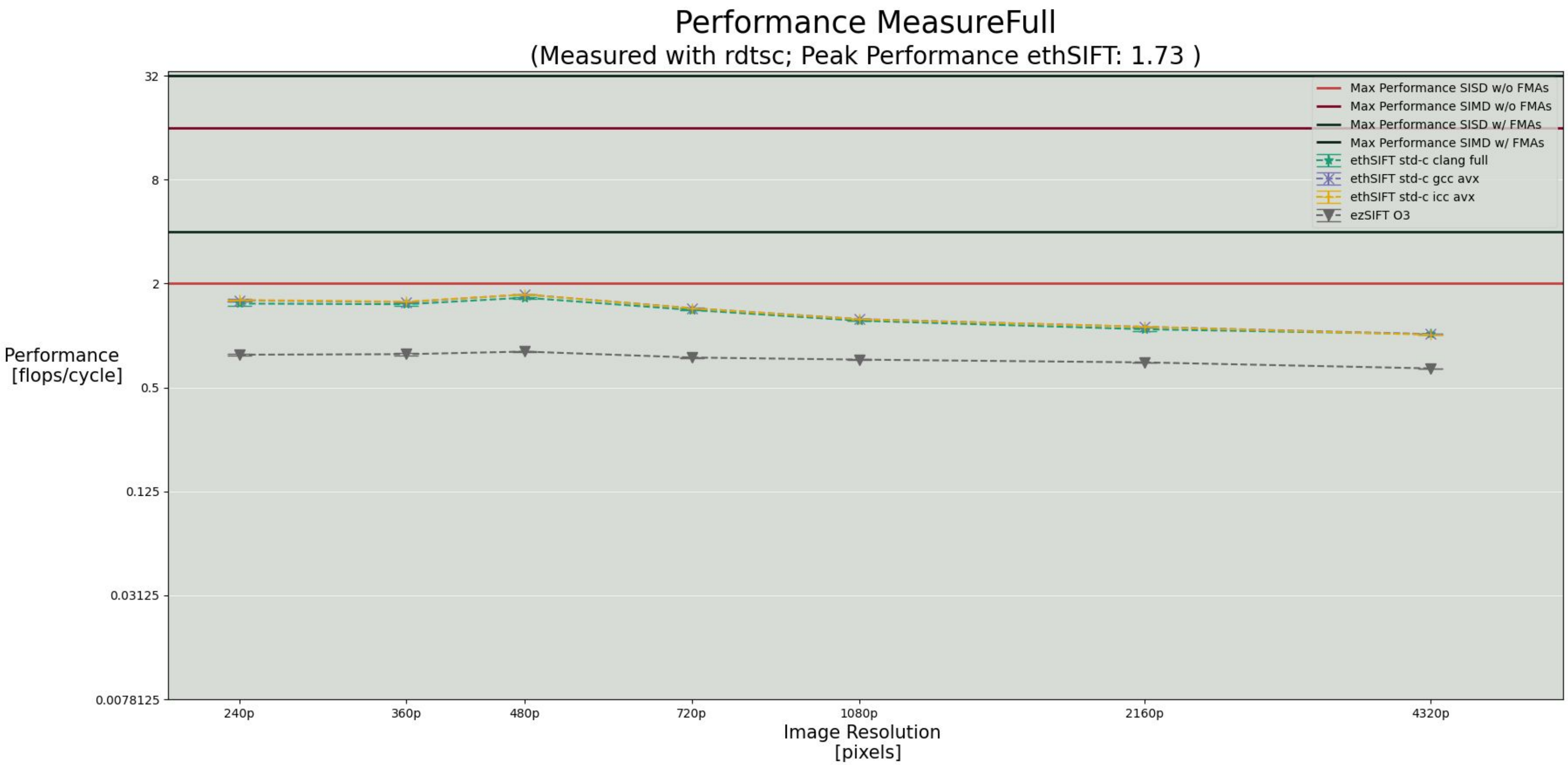
- ***Bottlenecks:***
  - Gaussian Pyramid
  - Gradient and Rotation Pyramids

- ***Reference Implementation:***
  - ezSIFT

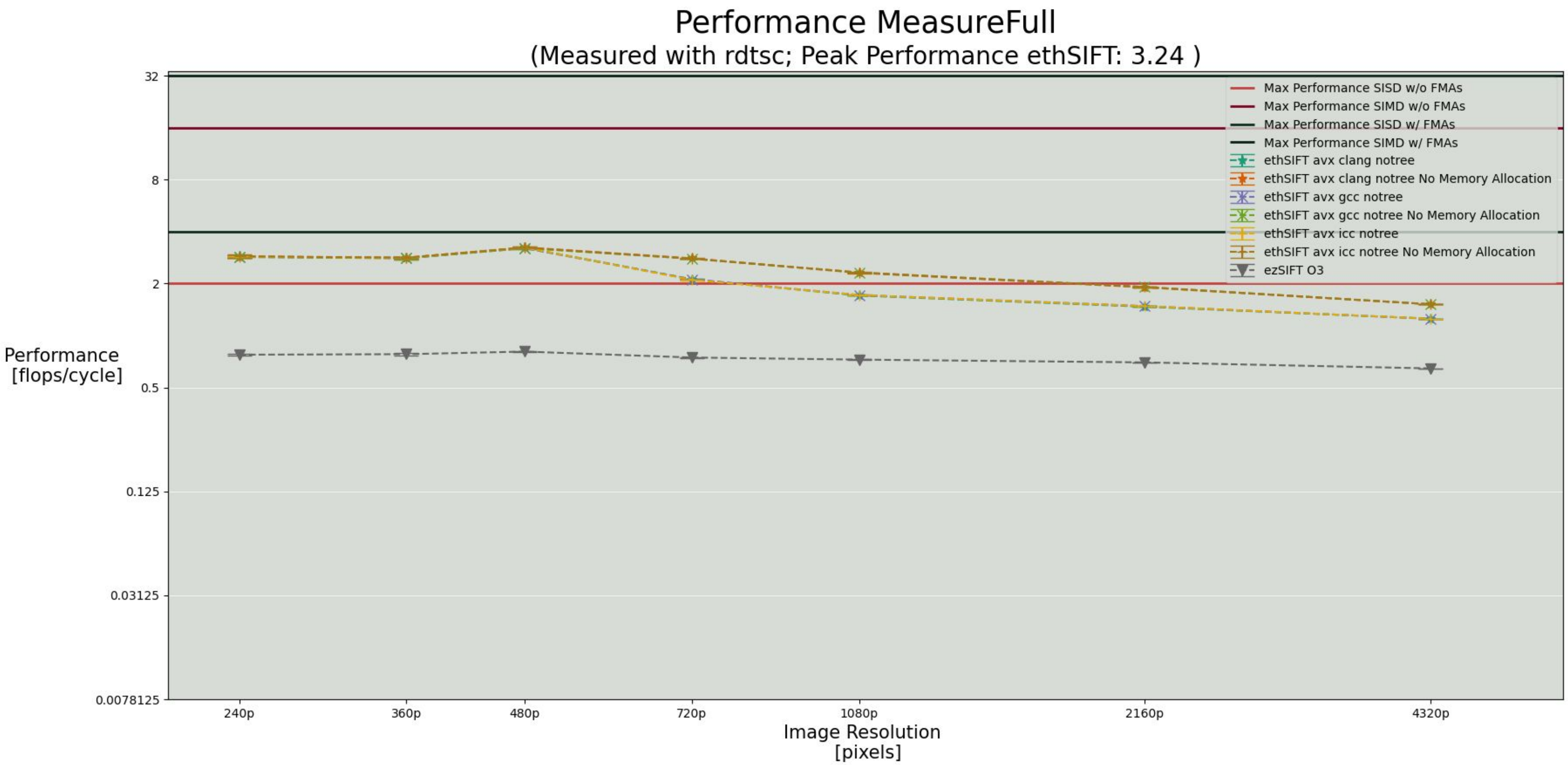
Baseline Implementation (Intel i7-8700K @ 4.4GHz)



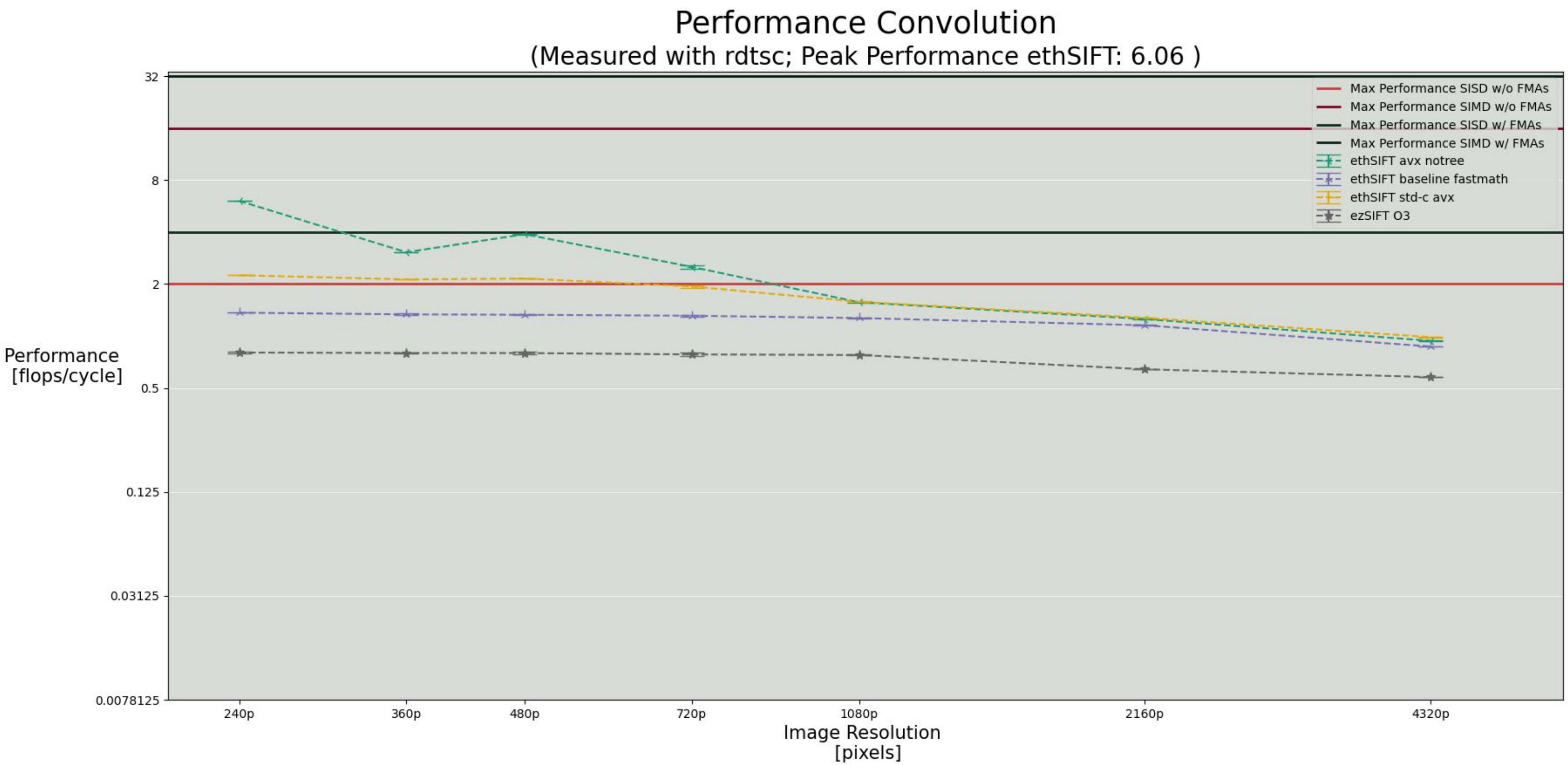
Standard C optimisations (Intel i7-8700K @ 4.4GHz)



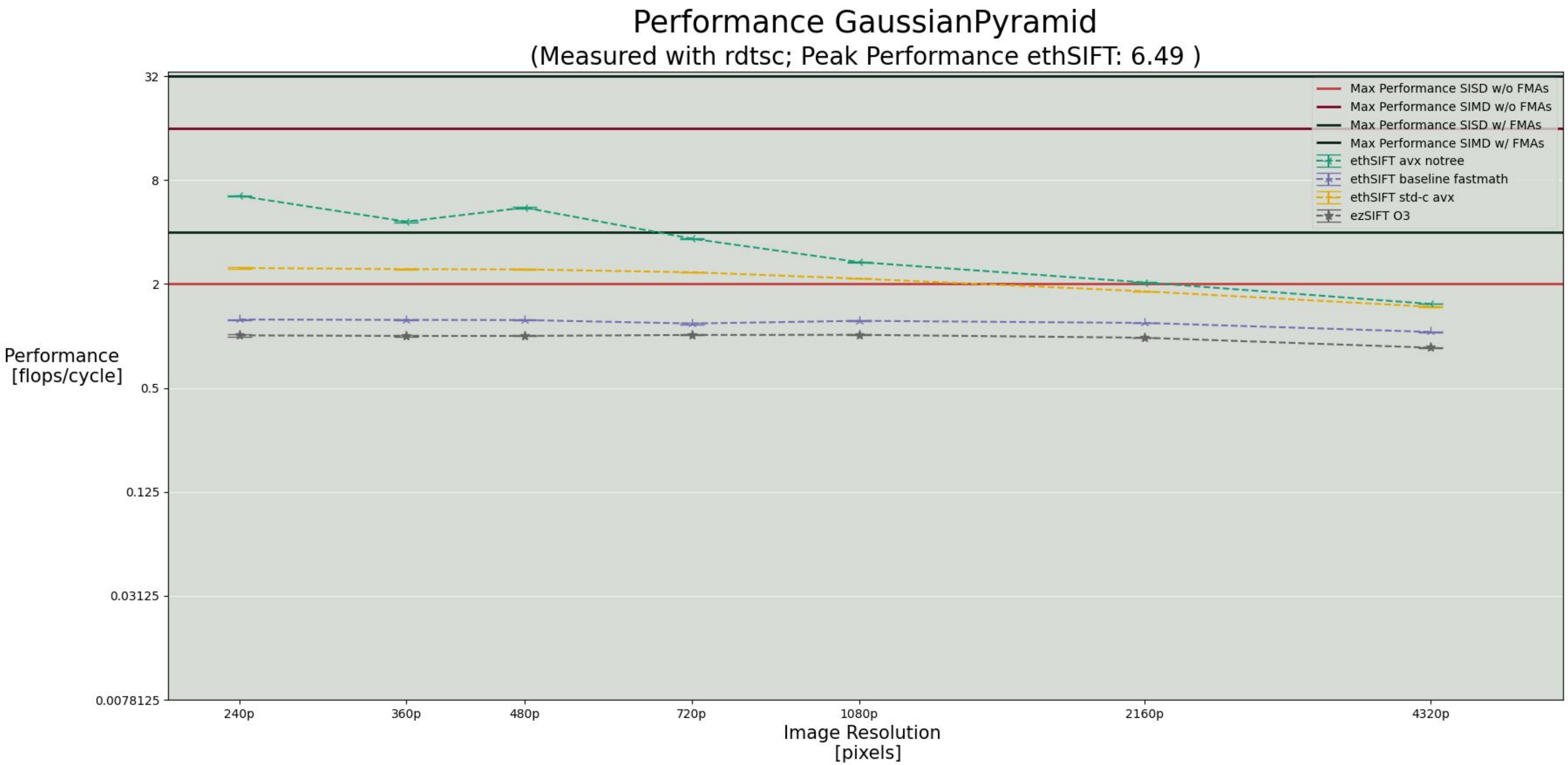
AVX optimisations (Intel i7-8700K @ 4.4GHz)



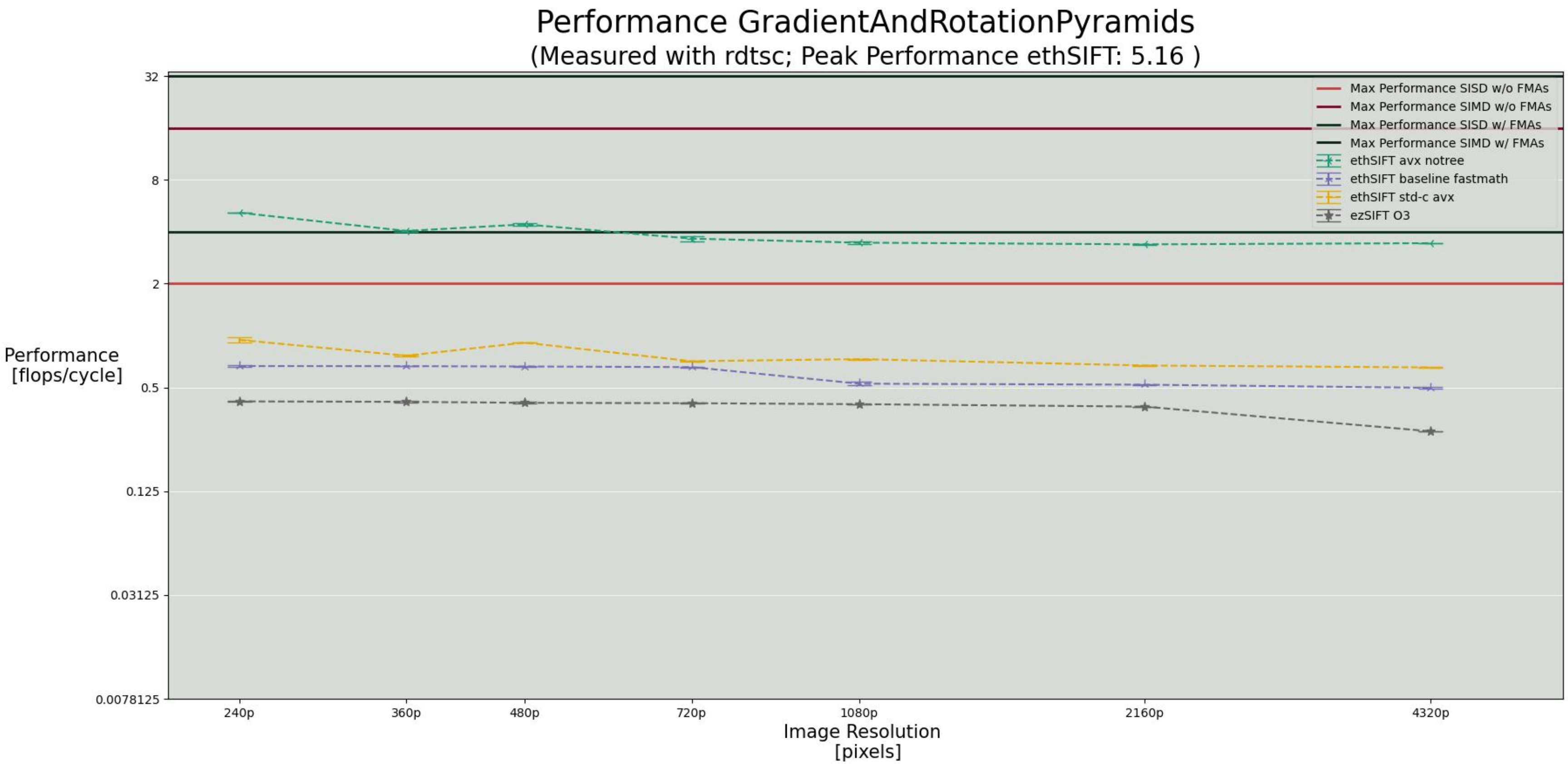
Some interesting plots (Intel i7-8700K @ 4.4GHz with ICC)



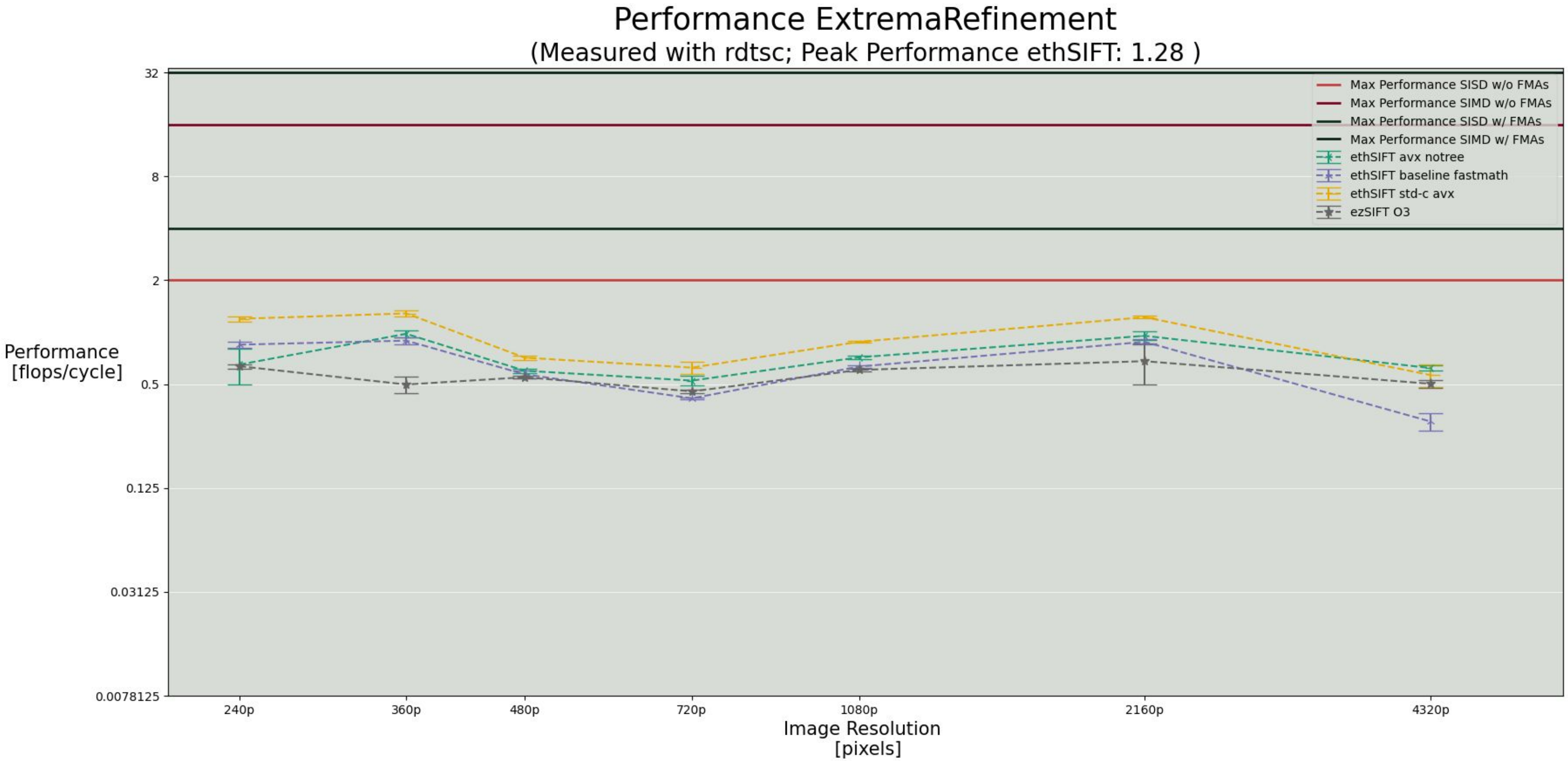
Some interesting plots (Intel i7-8700K @ 4.4GHz with ICC)



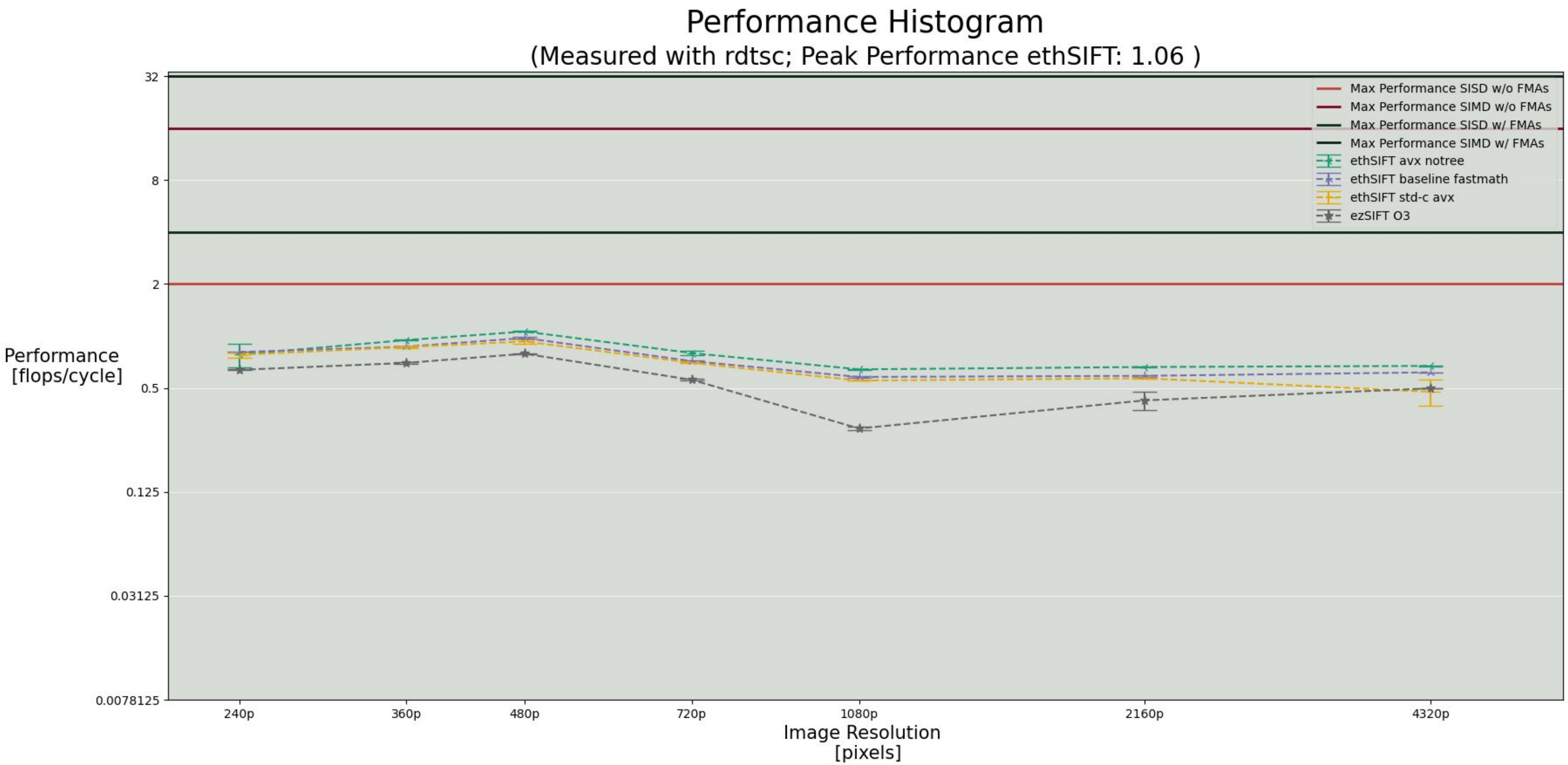
Some interesting plots (Intel i7-8700K @ 4.4GHz with ICC)



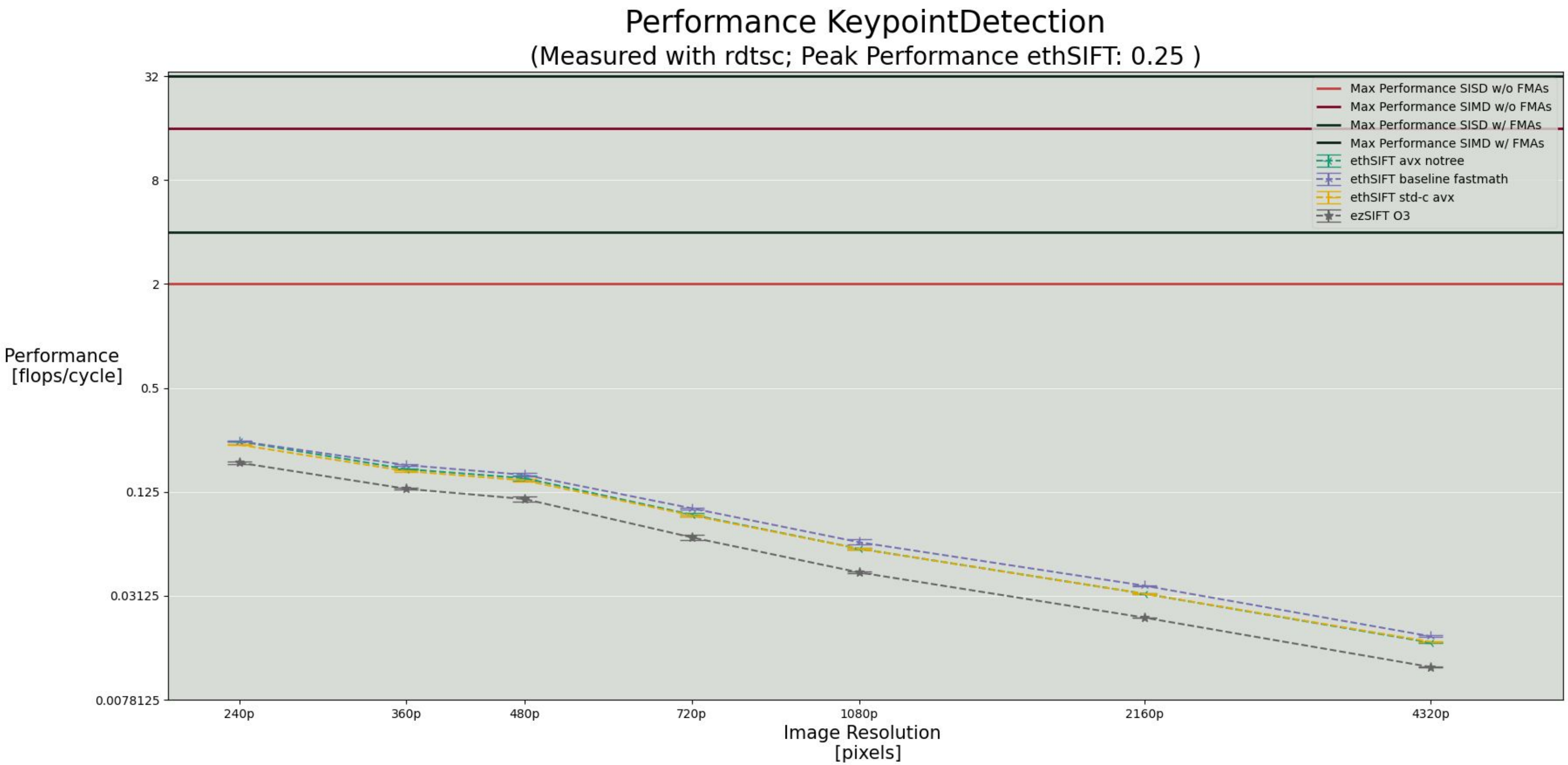
Some interesting plots (Intel i7-8700K @ 4.4GHz with ICC)



Some interesting plots (Intel i7-8700K @ 4.4GHz with ICC)

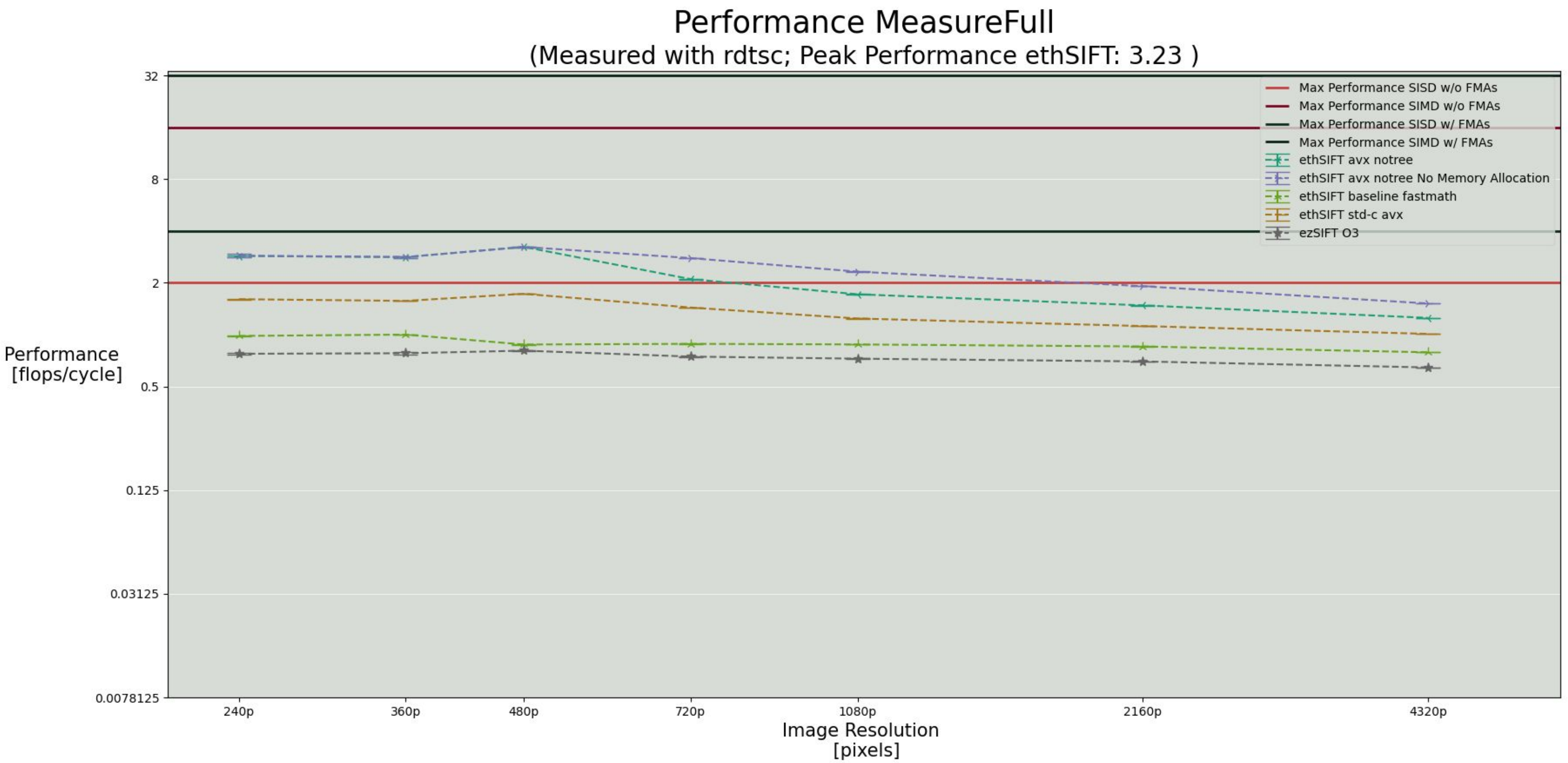


Some interesting plots (Intel i7-8700K @ 4.4GHz with ICC)



Experimental Results (Intel i7-8700K @ 4.4GHz with ICC)

Overall Speedup: ~3x



Interesting Findings

- ★ **ezSIFT leaks** memory by default (manually fixed)
- ★ **Helped** performance:
  - Variable **declarations close to point of use**
  - Declaring **const**
- ★ **Worsened** performance:
  - **size_t** instead of int
  - Forcing **memory** to be **paged** when allocated
- ★ **Sometimes** helped sometimes hurt**:**
  - **Prefetching** memory
  - Compiler **flags**

Without optimisations

1) GCC
2) Clang
3) ICC

With optimisations

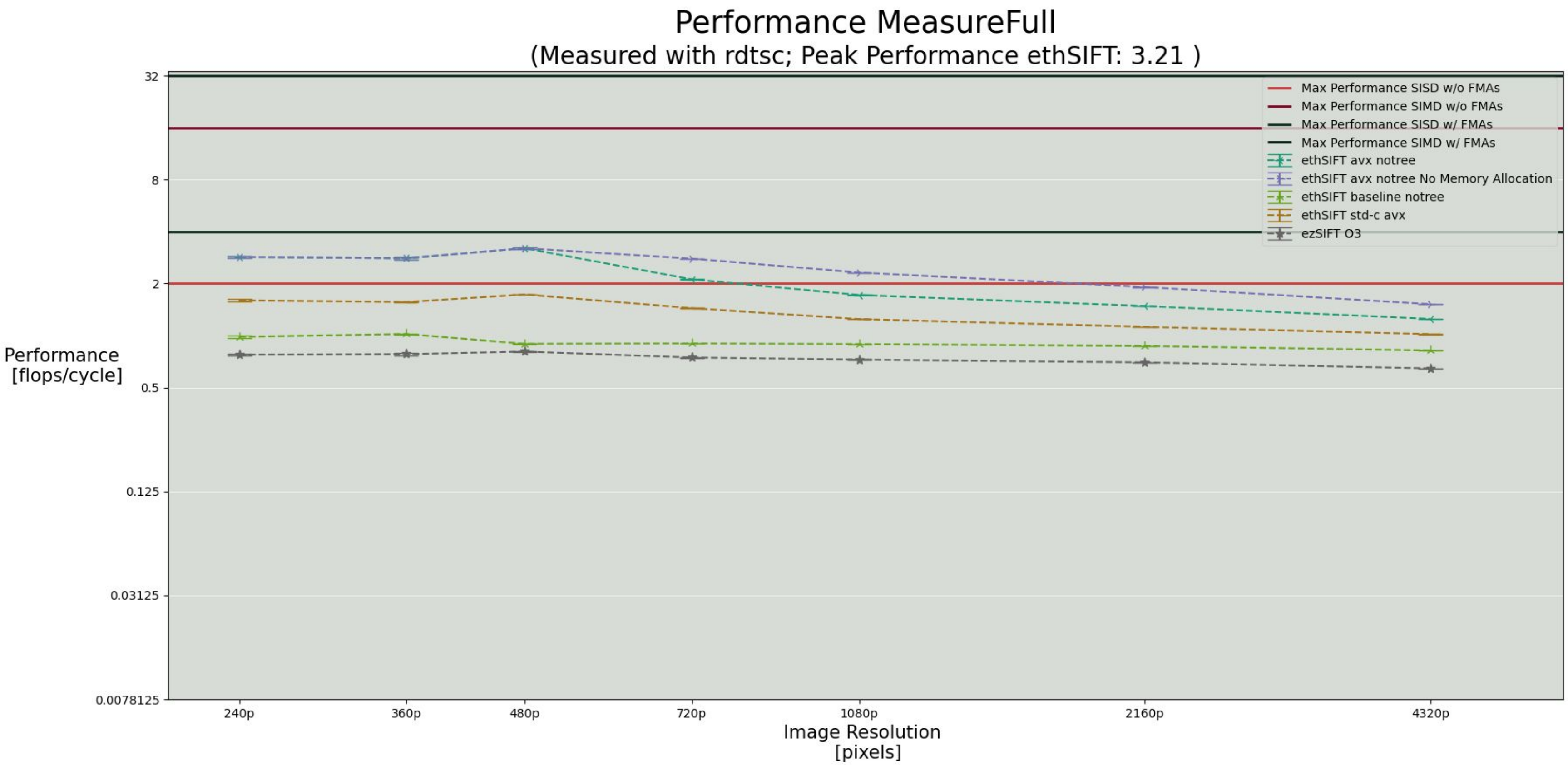
1) ICC
2) Clang
3) GCC

In Conclusion

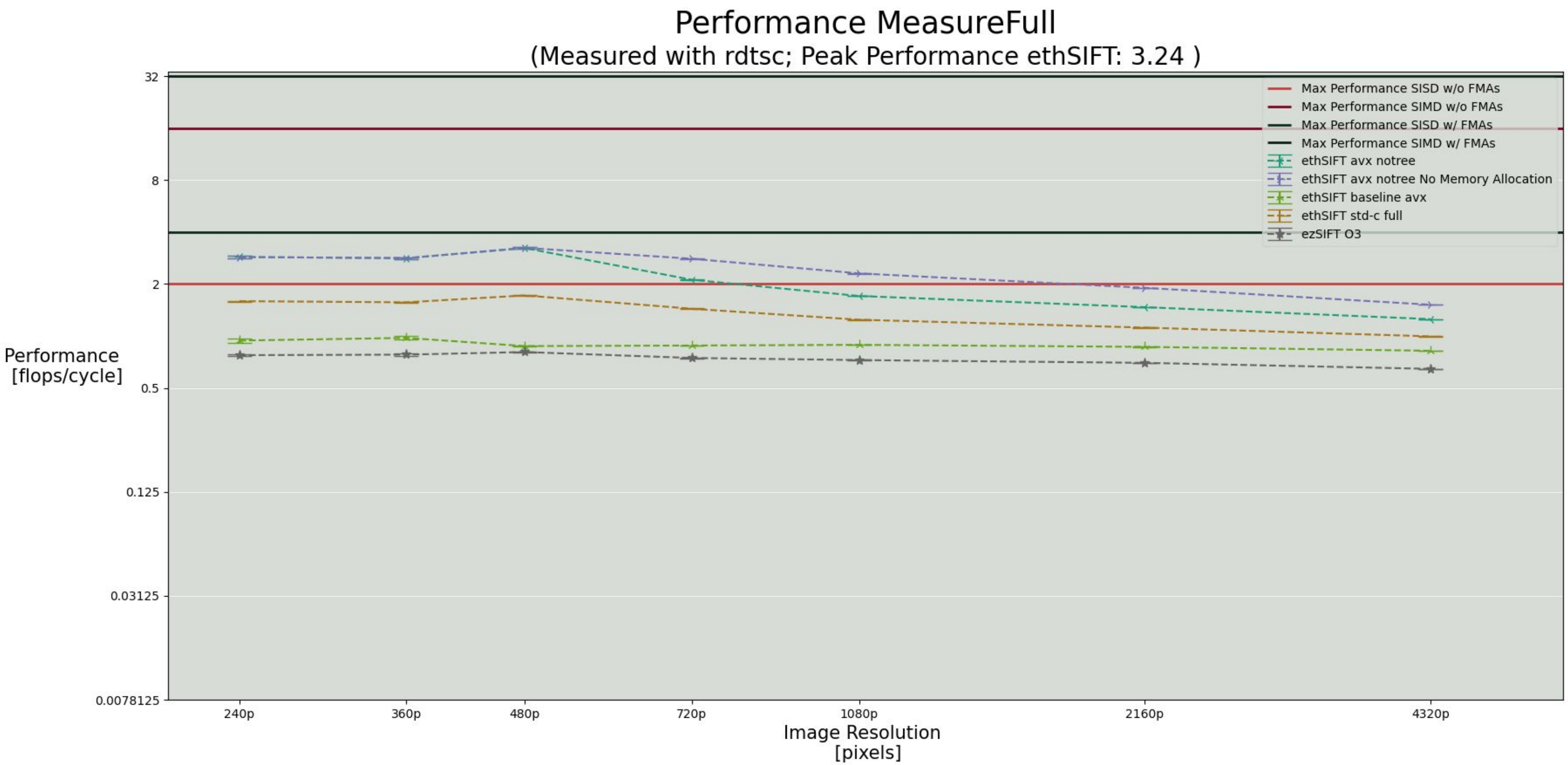
- **SIFT is difficult to optimise**
- **Gaussian** kernel convolution quickly becomes the **biggest bottleneck**
- In parts achieved **up to 10x** speedup over the reference
- **Overall** achieved **~3x** speedup

Appendix

Experimental Results (Intel i7-8700K @ 4.4GHz with GCC)



Experimental Results (Intel i7-8700K @ 4.4GHz with CLANG)



STEP 1: Scale-space Extrema Detection

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- Layers in Gaussian Pyramid

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$$

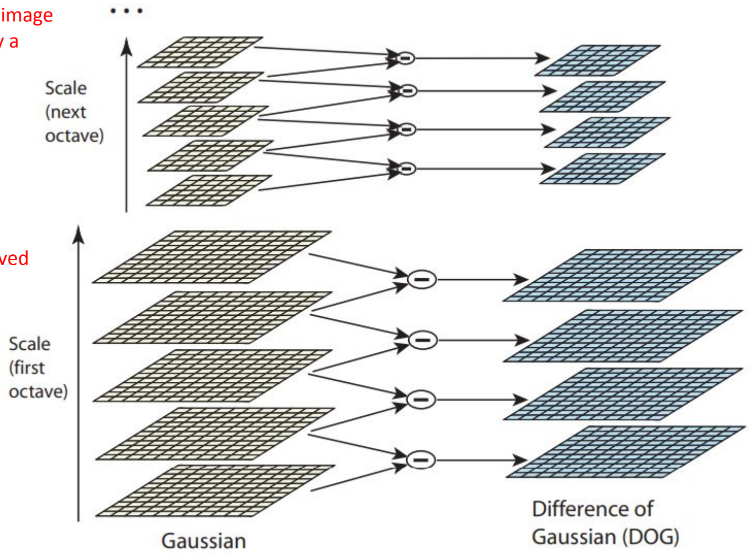
- Layers in Difference of Gaussians

$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma)$$

Constant multiplicative factor k

After each octave, image is downsampled by a factor of 2

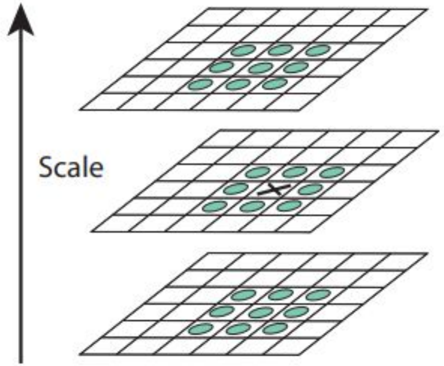
Repeatedly convolved with Gaussians



STEP 1: Scale-space Extrema Detection

Find local extrema in DoG pyramid by comparing a pixel to its 8 neighbour pixels in the same scale, as well as to the 9 pixels in each of the adjacent scales.

It is only selected if it is either larger or smaller than all of these other pixel values.



STEP 2: Keypoint Localization

Refinement of potential keypoint candidates:

- Determine interpolated location of extrema, using Taylor Expansion of scale-space function $D(x, y, \sigma)$

$$x = (x, y, \sigma)^T \qquad D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \qquad \hat{x} = -\frac{\partial^2 D}{\partial x}^{-1} \frac{\partial D}{\partial x}$$

- Where $\hat{x}$ is the derived location of the extremum. Remove a keypoint if $|D(\hat{x})| < "threshold"$

- For stability, edge responses get eliminated. This is done by first computing the Hessian H:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

- And use the Hessian to check if ratio of principal curvatures is below some curvature threshold r

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

STEP 3: Orientation Assignment

Next, Orientation is assigned to each keypoint to achieve invariance to image rotation.

- Calculation of gradient magnitude $m(x,y)$, and orientation $\Theta(x,y)$, using pixel differences:

$$m(x,y) = \sqrt{\left(L(x+1,y) - L(x-1,y)\right)^2 + \left(L(x,y+1) - L(x,y-1)\right)^2}$$
$$\Theta(x,y) = \tan^{-1}\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}$$

- An orientation histogram is formed from gradient orientations of sample points within a region around the keypoint. It has 36 bins covering the 360° range of orientations.

- Each sample added to histogram is weighted by its gradient magnitude.

- Peaks in the orientation histogram correspond to dominant directions of local gradients.

STEP 4: Extraction of Keypoint Descriptor

Last step is to create a descriptor for the local image region:

- We consider a 16x16 pixel patch around the keypoint
- The patch is subdivided in 16 blocks of size 4x4, for each of these blocks an 8 bin orientation histogram is created
- This gives us a total of 128 bin values. These values are represented as a vector to form the keypoint descriptor

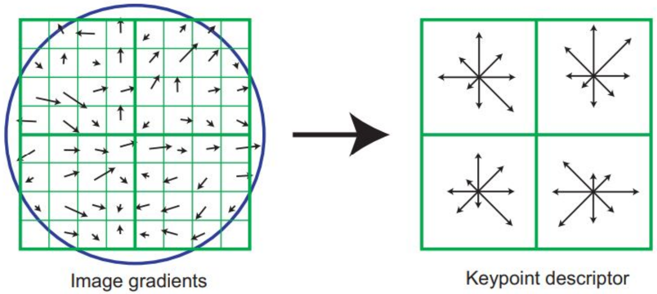
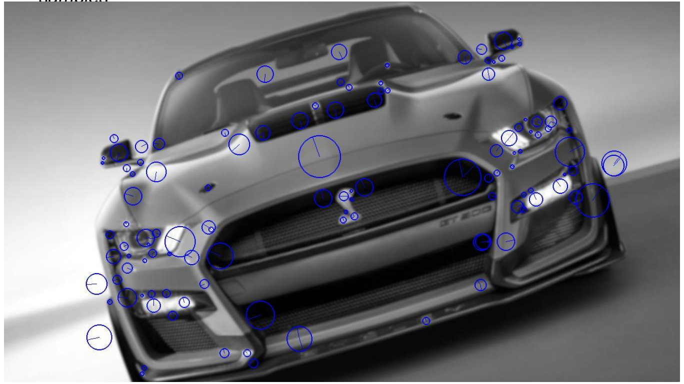


Figure: Shows a 2x2 descriptor array computed from an 8x8 set of samples



References

- Distinctive Image Features from Scale-Invariant Keypoints [Lowe, 2004]
- Guohui Wang, ezSIFT: an easy-to-use standalone SIFT library, 2013