Fast and Accurate Gaussian Pyramid Construction by Extended Box Filtering

Silvère Konlambigue, Jean-Baptiste Pothin, Paul Honeine, Abdelaziz Bensrhair

FAST AND ACCURATE GAUSSIAN PYRAMID CONSTRUCTION BY EXTENDED BOX FILTERING

Silvère KONLAMBIGUE[⋆†], *Jean-Baptiste POTHIN*[⋆], *Paul HONEINE*[†], *Abdelaziz BENSRHAIR*[†]

[⋆]DATAHERTZ
R&D, Troyes - FRANCE
silvere.konlambigue@datahertz.fr
jean-baptiste.pothin@datahertz.fr

[†]University of Rouen - INSA Rouen
LITIS, Rouen - FRANCE
paul.honeine@univ-rouen.fr
abdelaziz.bensrhair@insa-rouen.fr

ABSTRACT

Gaussian Pyramid (GP) is one of the most important representations in computer vision. However, the computation of GP is still challenging for real-time applications. In this paper, we propose a novel approach by investigating the extended box filters for an efficient Gaussian approximation. Taking advantages of the cascade configuration, tiny kernels and memory cache, we develop a fast and suitable algorithm for embedded systems, typically smartphones. Experiments with Android NDK show a 5x speed up compared to an optimized CPU-version of the Gaussian smoothing.

Index Terms— Gaussian pyramid, extended box filters, computer vision, SIFT.

1. INTRODUCTION

The Scale-Invariant Feature Transform (SIFT) algorithm [1] performs keypoints that are invariant to scale, rotation and (partially) viewpoint. The good distinctiveness of the generated features made the SIFT method a reference in the computer vision field. It is widely used for instance for object detection or recognition [2, 3] to name a few. However, due to its high computation-time, the SIFT method cannot be used in real-time applications. Although several computer vision algorithms have been proposed to reduce the computational time, they rather favor the speed and sacrifice the quality of the extracted features.

This paper deals with software speed-up of the SIFT method while keeping the same quality of the extracted features. This choice over dedicated hardware, such as in [4, 5, 6], is motivated by the high popularity and low cost mobile devices such as smartphones. Huang *et al.* show in [4] that the most time-consuming in the SIFT method is the construction of the Gaussian Pyramid (GP), as it represents about 80% of the whole processing.

The GP can be viewed as a stack of blurred-images, where one layer is obtained by filtering the original image with a Gaussian filter. Several methods exist to speed-up this step. The fastest approximate the Gaussian smoothing by a convolution with iterative box filters [7] or binomial filters [8]. The main drawback is that they fail to approximate Gaussians with an arbitrary standard deviation σ. Recursive filters [9] are more sophisticated as they allow a very good approximation of Gaussians with large σ values. However, the approximation is less good for small values. Therefore, they are not efficient for GP.

In this paper, we revisit the construction of GP in the light of the recent survey in [10] and recent developments on the extended box (ebox) filters. Ebox is an extension of box filters that allows a fractional radius [11]. They combine simplicity/efficiency, while providing a much better approximation to Gaussian convolution than iterative box filters. We propose an improvement of ebox, that takes advantage of tiny kernels and makes the method faster than the original one in [11]. In addition, we propose an efficient "cache-friendly" implementation that reduces significantly the delay caused by data accessing. For the sake of clarity, we restrict the presentation to the implementation in CPU-systems; Extensions to GPU-systems will be considered in future works.

This paper is organized as follows. We first review the GP in Section 2. In Section 3, we give a primer of the extended box method before we present the new method. More details about the implementation are given in Section 3.2. Experiments and results are presented in Section 4, followed by a conclusion and future works.

2. SIFT GAUSSIAN PYRAMID

This section briefly reviews the scale-space pyramid generation in the SIFT method. Details about keypoint detection and descriptor generation are available in [1].

The Gaussian scale-space of an image is formed by convolution of the original image $I(x, y)$ and Gaussians

functions G_σ of variable standard deviation σ, that is:

$$(G_\sigma * I)(x,y) = \iint_{\mathbb{R}^2} G_\sigma(x-u, y-v) I(u,v) du dv, \quad (1)$$

with

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp\Big(-\frac{x^2+y^2}{2\sigma^2}\Big). \quad (2)$$

The result of the convolution is a Gaussian-blurred image at scale σ. In practice, the operation has to be discretized. The kernel G_σ is truncated and normalized to unity. Its width W is usually chosen as:

$$W = 2\,\mathrm{round}(K\sigma) + 1. \quad (3)$$

where $\mathrm{round}(x)$ denotes the closest integer to x. In the SIFT method, the scale σ is discretized according to:

$$\sigma_{s,o} = \sigma_0\, 2^{o+(s/S)}, \quad (4)$$

where s is the *scale index*, o the *octave index*, S the *scale resolution*, and $\sigma_0 \in \mathbb{R}^+$ the *base scale offset*. Using the values proposed in [1], i.e., $\sigma_0 = 1.6$, $S = 3$ and $o_{\min} = -1$ (which means the original image is doubled in both width and height before processing), the discrete set of scales can be grouped in *octave* as:

$$\{1.6\delta_o, 2.02\delta_o, 2.54\delta_o, 3.2\delta_o, 4.03\delta_o, 5.8\delta_o\} \quad (5)$$

with $\delta_o = 2^o$ representing the inter-pixel grid for the o-th octave, for $o = -1, 0, 1, 2, \ldots$.

Looking at (3) and (4), one can see that the size of the Gaussian kernels grows very fast with o and s, making basic processing inefficient. To overcome this limitation, Lowe [1] proposed two tricks: 1) the image that has blurred twice the initial value of σ is downsampled by two and used for the next octave, 2) the images in an octave are obtained by cascade filtering. The whole process of GP construction is represented in Fig. 1.

The first trick is based on the *scale invariance property* of the Gaussian kernels. The downsampling is done easily by taking second pixel in each row and column.

The cascade approach exploits the *semigroup property* that allows to decompose the filter G_σ as two (smaller) filters, i.e.,:

$$G_{\sigma_1} * G_{\sigma_2} = G_{\sqrt{\sigma_1^2+\sigma_2^2}}. \quad (6)$$

The next image in an octave is obtained by filtering the previous one with a Gaussian filter parametrized by:

$$\sigma_{\mathrm{cascade}}(i) = \sigma_0 \sqrt{2^{2i/S} - 2^{2(i-1)/S}}, \quad i \geq 1. \quad (7)$$

For the set (5), this yields to "effective" filter-values:

$$\sigma_{cascade} \in \{1.2263, 1.545, 1.9466, 2.4525, 3.09\}. \quad (8)$$

With the typical $K = 3$ and recommended values for the SIFT method, the width of Gaussian kernels used in the cascade approach are successively 9, 11, 13, 15 and 19.

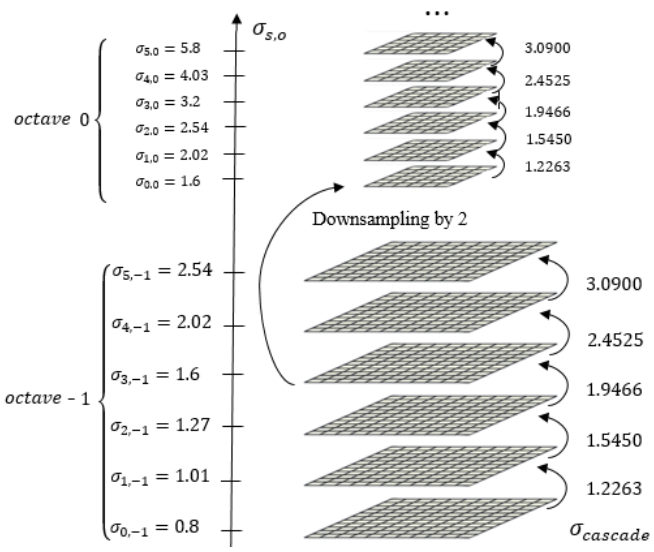

Fig. 1: The SIFT GP construction ($\sigma_0 = 1.6$, $S = 3$ and $o_{\min} = -1$). The scales $\sigma_{s,o}$ are achieved by cascade filtering with $\sigma_{\mathrm{cascade}}$ values on the right.

3. PROPOSED METHOD

In this section, we investigate the iterative ebox filters for fast and accurate Gaussian-smoothing approximation.

Let us first consider the one dimensional (1D) discrete ebox kernel, E_Λ, defined in [11]:

$$E_\Lambda(k) = \begin{cases} \frac{1}{\Lambda} & \text{if } \ -r \leq k \leq r \\ \frac{\alpha}{\Lambda} & \text{if } \ k \in \{\pm(r+1)\} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

with $k \in \mathbb{Z}$, $r \in \mathbb{N}_0$, $0 \leq \alpha < 1$ and $\Lambda = 2r + 1 + 2\alpha$.

Gaussian convolution with standard deviation σ can be approximated by d-passes of E_Λ with [11]:

$$r = \Big\lfloor \frac{1}{2}\sqrt{\frac{12\sigma^2}{d} + 1} - \frac{1}{2} \Big\rfloor, \quad (10)$$

$$\alpha = (2r+1)\frac{r(r+1) - \frac{3\sigma^2}{d}}{6\Big(\frac{\sigma^2}{d} - (r+1)^2\Big)}, \quad (11)$$

where $\lfloor x \rfloor$ denotes the so-called floor function. In practice, each pass can be efficiently implemented using a 'sliding-window' algorithm. After the first value of a row has been computed, the convolution $u_i = (E_\Lambda * f)_i$ at sample i is given by:

$$u_i = u_{i-1} + c_1(f_{i+r+1} - f_{i-r-2}) + c_2(f_{i+r} - f_{i-r-1}) \quad (12)$$

where $c_1 = \alpha/\Lambda$ and $c_2 = (1-\alpha)/\Lambda$. Thus, the computational complexity is independent from the length of the ebox. By the well-known *separability property*, image filtering can be accomplished using two 1D convolutions,

one in the horizontal and the other in the vertical direction. Eqn. (12) requires only 2 multiplications and 4 additions per sample. Hence, for an image, the method costs $4d$ multiplications and $8d$ additions per pixel.

3.1. Proposed method: Fast ebox

For small values of σ, like those involved in the GP construction, we propose an implementation faster than (12). To this end, we introduce the *non*-normalized counterpart of E_Λ, noted here $\tilde{E}_{r(\Lambda)}$, defined by:

$$\tilde{E}_{r(\Lambda)}(k) = \Lambda E_\Lambda(k) = \begin{cases} 1 & \text{if } -r \le k \le r \\ \alpha & \text{if } k \in \{\pm(r+1)\} \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

Its convolution with a signal f, at sample i, gives:

$$(\tilde{E}_{r(\Lambda)} * f)_i = \alpha(f_{i-r-1} + f_{i+r+1}) + \sum_{m=-r}^{r} f_{i+m}. \quad (14)$$

Iterating this kernel d-times gives:

$$\tilde{E}^d_{r(\Lambda)} := \underbrace{(\Lambda E_\Lambda) * \cdots * (\Lambda E_\Lambda)}_{d-\text{times}} = \Lambda^d E^d_\Lambda. \quad (15)$$

Dividing $\tilde{E}^d_{r(\Lambda)}$ by Λ^d allows to recover the iterative ebox E^d_Λ. Since Λ^d is constant over the passes, it can be propagated to the very last pass. In 2D, this means we multiply by $1/\Lambda^{2d}$ the result of the last pass at last direction.

Eqn. (14) needs $2(r+1)$ additions and 1 multiplication. Iterating d-times the same filter in horizontal and vertical directions yields $4(r+1)d$ additions and $2d+1$ multiplications (including the normalization that has to be done at the end) per pixel; See Table 1.

Inner ebox length	Multiplications	Additions	Memory access (Read)
$r = 0$ [Eqn. (14)]	$2d+1$	$4d$	3
$r = 1$ [Eqn. (14)]	$2d+1$	$8d$	5
$r \ge 2$ [Eqn. (12)]	$4d$	$8d$	5

Table 1: Number of operations per pixel for the proposed fast ebox filtering.

Using (10) and noting that the argument in the floor function is a strictly increasing function of σ, one can show that:

$$r = 0 \Leftrightarrow \sigma \in [0; \sqrt{2d/3}[, \quad (16)$$

and

$$r = 1 \Leftrightarrow \sigma \in [\sqrt{2d/3}; \sqrt{2d}[. \quad (17)$$

These relations can be used to justify the computational advantage in considering (14) instead of (12) in the SIFT applications. As an example, for $d = 4$, the cases $\sigma_{\text{cascade}} < \sqrt{8/3}$ and $\sqrt{8/3} \le \sigma_{\text{cascade}} < \sqrt{8}$ occur respectively two times per octave, giving a reduction in number of operations for 4 filters out of 5 (per octave).

Code 1: Fast Iterative Ebox ($r = 0$)

```
1  void ebox_conv_r0(float* I, float* J, int d,
2    float* tmp, float* tmprows, int width,
3    int height, float alpha, float coeff) {
4   float *rin, *rout;
5   /***** convolution along x-direction *****/
6   for (int y = 0; y < height; ++y) {
7    int iter = d;
8    rin = &I[y*width];  rout = tmprows;
9    while (--iter > 0) {
10     scanline_r0(rin,rout,alpha,width-1);
11     rin = rout;     rout += width;
12   }
13  scanline_r0(rin,tmp,alpha,width-1,height,y);
14  }
15  /***** convolution along y-direction *****/
16  for (int y = 0; y < width; ++y) {
17   int iter = d;
18   rin = &tmp[y*height]; rout = tmprows;
19   while (--iter > 0) {
20     scanline_r0(rin,rout,alpha,height-1);
21     rin = rout;    rout += height;
22   }
23   scanline_r0(rin,J,alpha,height-1,width, y);
24  }
25  for(int i=0, wh=width*height; i < wh; ++i)
26    J[i] *= coeff; // normalization
27 }
28
29 void scanline_r0(float* in, float* out,
30    float w, int xmax) {   int x;
31  out[0] = in[0] + w * (in[1] + in[1]);
32  for (x=1; x < xmax; ++x)
33    out[x] = w * (in[x-1] + in[x+1]) + in[x];
34  out[x] = w * (in[x-1] + in[x-1]) + in[x];
35 }
```

3.2. Cache-aware implementation

A "cache-friendly" algorithm generally exploits the principle of temporal and spatial locality [12], which states that data located close together in address-space are also referenced close together in time.

We propose a cache-friendly algorithm, in the same spirit as the one in [13] for reconstructing X-ray images. The C++ implementation (see Code 1) starts with an horizontal convolution. For the first $d-1$ iterations, it works iteratively (lines 9-12) using the function *scanline_r0* to get advantages of prefetched data in the cache. This function handles boundaries by symmetry (lines 31 and 34) outside the loop in order not to waste time for unnecessary checks. The last iteration along x-direction (line 13) performs the convolution and on-the-fly transposition - code not listed here due to lack of space. The result is stored in a temporary array re-used as input for the vertical convolution (lines 16-24) based on the same row-by-row process. The transposed result (line 23) is stored in J (the proper orientation) and a final normalization is done (lines 25-26).

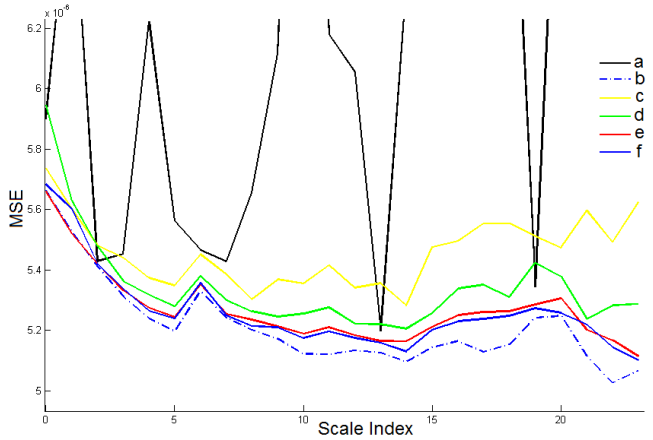

Fig. 2: MSE for test image 640x480 with $o_{\min} = -1$: a) box filtering with $d = 6$, b) Gaussian, the proposed ebox filtering with c) $d = 2$, d) $d = 3$, e) $d = 4$, f) $d = 5$.

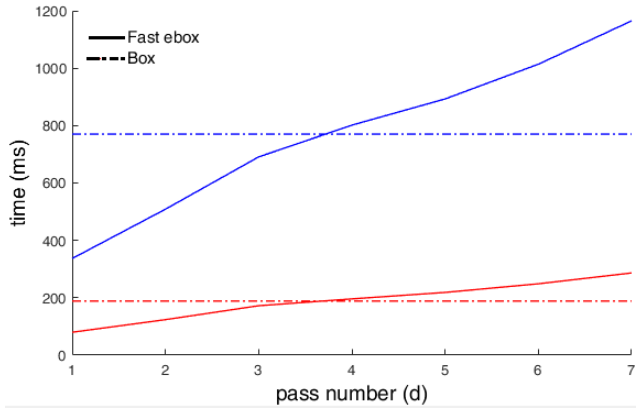

Fig. 3: Running-time of the GP construction for QVGA (320x240) in red (bottom) and VGA (640x480 pixels) in blue (top), by iterative ebox with different passes d. 'Box' stands for the 6-iterative box filters.

4. EXPERIMENTS AND RESULTS

In this evaluation, the Gaussian filtering and the proposed fast ebox method were implemented for Android NDK in C++. The test images were recorded by the Android app on a Samsung Galaxy A5 (Qualcomm Snapdragon 410 1.2 GHz) using the camera preview mode. For one image test from the camera preview, the Android app created the GP from each method and stored the associated blurred-images.

4.1. Accuracy of the iterative ebox filtering

We first investigated the accuracy of iterative ebox filters. The blurred-images generated by the Android implementations were compared to a Matlab implementation, based on the Mean Square Error defined as:

$$MSE(s) = \frac{1}{|L(s)|} \sum_x \sum_y \left(L(x,y;s) - L^*(x,y;s)\right)^2,$$

where L^* is the "reference" blurred-image, L the approximated image, and $|L(s)|$ the number of pixels at scale index s. Fig. 2 shows the MSE for one test image. Using ℓ_∞ operator norm, curves are similar (not shown here due to lack of space). In Fig. 2, one can remark that biggest variations are given by iterative box filters, despite the high number ($d = 6$) of iterations. The small difference between Matlab and Android for Gaussian comes from the precision level (double vs floating-point). Iterative ebox gives a better approximation than iterative box. For $d \geq 4$, there is almost no difference between iterative ebox and Gaussian filtering.

Box filters introduce noise on edges, known in image processing as ringing artifacts. To quantify this, we used the Blur Measure (BM), defined in [14] as :

$$BM(I) = \frac{\sum_{(x,y)\in E} \sqrt{\sum_{(x',y')\in N_{xy}} (I(x,y) - I(x',y'))^2 / |N_{xy}|}}{\sum_{(x,y)\in E} I(x,y)},$$

where E is the set of edge pixels, obtained using a Sobel operator, and N_{xy} a set of 8-neighborhood of a pixel $I(x,y)$. Table 2 shows the Normalized Absolute Error (NAE), $|BM(L) - BM(L^*)|/BM(L^*)$, for the 1st octave. Higher value of NAE BM-score means there is higher change in intensity along the edges which in turn means image has more artifacts. As can be seen, the NAE BM-score of Fast ebox is lower, which means that ebox is better than box near the edges.

Scale index	0	1	2	3	4	5
Box ($d = 6$)	2.31	3.70	0.50	1.90	3.22	1.88
Fast ebox	0.37	0.30	0.36	0.91	0.42	0.47

Table 2: NAE BM-score (%).

4.2. Running-time

Fig. 3 shows the (average) running-time of the GP construction step with our Android implementation; the dashdot lines represents box filters with $d = 6$, which is the recommended value in [15]. As can be seen, the running time of ebox is piecewise linear in d. This can be explained by analysis in Section 3. From Fig. 2 and 3, $d = 4$ is a good choice, because it is as fast as iterative box filters while being a very good approximation of the Gaussian. In Table 3, we compare the running time with CPU-only implementations of GP reported in [4, 16]. Although our smartphone is not as fast as those used by the authors, our algorithm is faster in both cases QVGA

and VGA. To get insight the speed up improvement, let's get back to a common clock rate by normalization. The process is based on the well-known CPU performance equation :

$$\text{CPU time} = \frac{\text{Instruction Counts} \times \text{CPI}}{\text{Clock Rate}}, \qquad (18)$$

where CPI represents the Cycles Per Instruction. Assuming the same program (i.e. no changes in the compiler), we can apply a change in the clock rate as follows :

$$\text{CPU time}_{\text{new}} = \frac{\text{CPU time}_{\text{old}} \times \text{Clock Rate}_{\text{old}}}{\text{Clock Rate}_{\text{new}}}. \qquad (19)$$

Results for a clock rate of 1 GHz are shown in italic in Table 3. Comparing the normalized running-times, our implementation gives a 5x speed up.

	Input	Time (ms)	CPU Freq (GHz)
Rister *et al.* (see Table 1 on CPU, [16])	QVGA	734 *1101*	1.5 1
Huang *et al.* (see Table 2, [4])	VGA	2100 *4389*	2.09 1
Fast ebox ($d = 4$)	QVGA	185 *222*	1.2 1
	VGA	754 *905*	1.2 1

Table 3: Comparison of running times. The values in italic are obtained by normalization.

5. SUMMARY

In this paper, we proposed fast extended box filters for Gaussian Pyramid construction. Experiments on a single CPU and Android NDK showed the efficiency of the proposal, in terms of accuracy and running-time. In ongoing research, we will study parallel implementation of the proposed methods, based on multi-core CPUs or hardware such as GPUs. We will also study ebox for fast and accurate approximations of Gaussian second-derivatives, as they are widely used in signal processing for applications such as the Laplacian or the Hessian detectors. Finally, by optimizing the rest of its steps, we will implement the entire SIFT method on an Android device for a real-time localization application.

References

[1] David G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[2] David G. Lowe, "Object recognition from local scale-invariant features," in *The proceedings of the 7th IEEE International Conference on Computer Vision*, 1999, vol. 2, pp. 1150–1157.

[3] Matthew Brown and David G. Lowe, "Recognising panoramas," in *The proceedings of the 9th International Conference on Computer Vision*, 2003, pp. 1218–1225.

[4] Feng-Cheng Huang, Shi-Yu Huang, Ji-Wei Ker, and Yung-Chang Chen, "High-performance SIFT hardware accelerator for real-time image feature extraction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 3, pp. 340–351, 2012.

[5] Liu Ke, Jun Wang, Xijun Zhao, and Fan Liang, "Fast-gaussian SIFT and its hardware architecture for keypoint detection," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2016, pp. 436–439.

[6] Jie Jiang, Xiaoyang Li, and Guangjun Zhang, "SIFT hardware implementation for real-time image feature extraction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 7, pp. 1209–1220, 2014.

[7] William M. Wells, "Efficient synthesis of gaussian filters by cascaded uniform filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 2, pp. 234–239, 1986.

[8] Matthew Aubury and Wayne Luk, "Binomial filters," in *Journal of VLSI Signal Processing*, 1995, pp. 1–8.

[9] Rachid Deriche, *Recursively implementating the Gaussian and its derivatives*, Ph.D. thesis, INRIA, 1993.

[10] Pascal Getreuer, "A survey of gaussian convolution algorithms," *Image Processing On Line*, pp. 286–310, 2013.

[11] Pascal Gwosdek, Sven Grewenig, Andrés Bruhn, and Joachim Weickert, "Theoretical foundations of gaussian convolution by extended box filtering," in *International Conference on Scale Space and Variational Methods in Computer Vision*, 2011, pp. 447–458.

[12] Peter J. Denning, "The locality principle," *Communications of the ACM*, vol. 48, no. 7, pp. 19–24, 2005.

[13] Hüseyin Cüneyt Aka, "Reconstruction of X-ray images," M.S. thesis, İzmir Institute of Technology, 1997.

[14] Kanjar De and V. Masilamani, "A new no-reference image quality measure for blurred images in spatial domain," *Journal of Image and Graphics*, vol. 1, no. 1, pp. 39–42, 2013.

[15] Peter Kovesi, "Fast almost-gaussian filtering," in *The proceedings of Inter. Conf. on Digital Image Computing: Techniques and Application*, 2010, pp. 121–125.

[16] Blaine Rister, Guohui Wang, Michael Wu, and Joseph R. Cavallaro, "A fast and efficient SIFT detector using the mobile GPU," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 2674–2678.