Hardware-Efficient Guided Image Filtering For Multi-Label Problem

Longquan Dai[1]   Mengke Yuan[2]   Zechao Li[1]   Xiaopeng Zhang[2]   Jinhui Tang[1*]

1. School of Computer Science and Engineering, Nanjing University of Science and Technology
2. National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Science

{dailongquan, zechao.li, jinhuitang}@njust.edu.cn, {mengke.yuan, xpzhang}@nlpr.ia.ac.cn

Abstract

*The Guided Filter (GF) is well-known for its linear complexity. However, when filtering an image with an n-channel guidance, GF needs to invert an $n \times n$ matrix for each pixel. To the best of our knowledge existing matrix inverse algorithms are inefficient on current hardwares. This shortcoming limits applications of multichannel guidance in computation intensive system such as multi-label system. **We need a new GF-like filter that can perform fast multichannel image guided filtering.** Since the optimal linear complexity of GF cannot be minimized further, the only way thus is to bring all potentialities of current parallel computing hardwares into full play. In this paper we propose a hardware-efficient Guided Filter (HGF), which solves the efficiency problem of multichannel guided image filtering and yields competent results when applying it to multi-label problems with synthesized polynomial multichannel guidance. Specifically, in order to boost the filtering performance, HGF takes a new matrix inverse algorithm which only involves two hardware-efficient operations: element-wise arithmetic calculations and box filtering. In order to break the linear model restriction, HGF synthesizes a polynomial multichannel guidance to introduce nonlinearity. Benefiting from our polynomial guidance and hardware-efficient matrix inverse algorithm, HGF not only is more sensitive to the underlying structure of guidance but also achieves the fastest computing speed. Due to these merits, HGF obtains state-of-the-art results in terms of accuracy and efficiency in the computation intensive multi-label systems.*

1. Introduction

Since 2010, GF has been applied to many computer vision and graphics problems such as image retargeting [6], color transfer [1] and video dehazing [33]. Among them, multi-label system maybe one of the most suitable applications for GF to make full use of its efficiency and effectiveness because the heavy computation in multi-label system

---

* Corresponding author.

is in urgent need of a fast filtering tool. A typical multi-label system [12] records the costs c for choosing a label l at coordinates x and y in a cost volume ($V(x, y, l) = c$) according to the input data. Then WTA (Winner-Takes-All) label selection strategy is exploited to determine the final label for each pixel (x, y) after the aggregation (*i.e.* guided image smoothing) step operated on each slice of the cost volume. Besides applying GF to the aggregation step of the multi-label system, we can also incorporate it into MRF models [28, 30] since edge-aware filters adopted in these MRF models can be taken place by GF directly.

Due to the linear complexity and edge-preserving ability, GF is consider as the best choice [12] among all candidate filters for the multi-label system. However, a blemish of GF is that the color image guided filtering algorithm is not very efficient. More specifically, matrix inverse is a time-consuming operation, but GF needs to invert a 3×3 matrix for each pixel to calculate both coefficients $\vec{w}_{\mathfrak{p}}$ of Eq (1) and filtering result $\mathbf{Z}(\mathfrak{q})$ according to the ith channel $\boldsymbol{I}_i(\mathfrak{q})$

$$\boldsymbol{Z}(\mathfrak{q}) = \sum_{i=1}^{3} \vec{w}_{\mathfrak{p}}(i)\boldsymbol{I}_i(\mathfrak{q}) + \vec{w}_{\mathfrak{p}}(0) \tag{1}$$

of a color input guidance $\boldsymbol{I}$. Fig 1a plots the run time of inverting 10^6 matrices with increasing matrix size, where the matrix inverse algorithm is the built-in LU algorithms of OpenCV. We can observe that the run time increases with the size of matrix dramatically. Hence it is inefficient to apply GF to multi-label system with a multi-channel guidance, especially for a large channel number.

To decrease the execution time, the most straightforward method for GF is to launch a set of threads to invert matrices simultaneously. However this strategy is not very efficient on current hardwares. This is because 1, both CPU and GPU rely on SIMD (Single Instruction Multiple Data) architecture to boost performance; 2, branch instructions are inevitable for traditional matrix inversion methods such as LU algorithm [13]; 3, SIMD architecture cannot achieve the fastest speed to run branch instructions as these instructions require decomposing each vector into elements and processing them sequentially on the architecture. Fig 1a also illustrates the run time of inverting 10^6 matrices simultane-
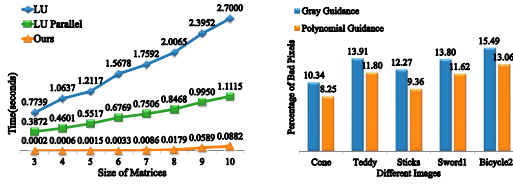
(a) Matrix Inverse (b) Stereo Matching

Figure 1: **Matrix Inverse** & **Stereo Matching:** (a) demonstrates the run time of inverting 10^6 matrices with increasing size (b) illustrates the stereo matching results of GF with different guidances.

ously. The reported data is unsatisfactory. To avoid branch instructions, we can invert matrices according to the analytic solution of matrix inverse [11]. The fastest OpenCV implementation of GF takes this strategy to invert 3×3 matrices and successfully reduces the run time of inverting 10^6 matrices to less than $100ms$. But the implementation complexity of the analytic solution increases with matrix size heavily. When the size of matrix becomes large, the method is no longer implementable manually. This explains why we miss the run time of this method in Fig 1a.

We propose a hardware-efficient matrix inverse algorithm to exploit the parallel computing power of hardwares to its full potential. The ability of our algorithm stems from two hardware-efficient operations: *element-wise arithmetic calculations* and *box filtering*. To compare with the built-in LU algorithm adopted by OpenCV, we collect run times of our algorithm implemented by Intel OpenCL and plot them in Fig 1a. The result proves that our method gets a substantial performance boost on Intel platform.

We introduce nonlinearity to GF by synthesizing a polynomial multi-channel guidance from the input guidance to conquer the drawback of the linear model of GF. The linear model usually underfits the input data due to its simplicity and thus is apt to produce over-smoothed results. Fig 1b shows the stereo matching results under the guidance of a gray image and our synthesized multi-channel image. Obviously, the synthesized guidance produces better results. Note that the extra run time in accompany with the nonlinear model can be reduced to an acceptable level by our hardware-efficient matrix inverse algorithm.

Taking proposed polynomial guidance and new matrix inverse algorithm, we construct a hardware-efficient guided filter (HGF) for multi-label problem. For presentation clarity, we organize the rest of this paper in following way. Section 2 is devoted to introduce related work. In the next section, we present how to invert matrix efficiently and its application in computing regression coefficients. Section 4, 5 discuss HGF in details, including the hardware-efficient implementation and the method to synthesis polynomial guidances. Finally, we conduct comprehensive experiments to verify the superiority of HGF in section 6.

2. Related Work

Acceleration: People have already proposed many image guided filters such as the Bilateral Filter (BF) [27] and the Median Filter (MF) [34] for edge-preserving smoothing. However, the benefit is not free. For example, the naive implementation of BF usually requires several minutes to filter a typical megapixel image. Several methods have been developed to accelerate the computation of bilateral filtering [7, 21, 23, 32, 8, 5] and as a result people obtained several orders of magnitude speedup. Similar to BF, MF can also effectively filter images while not strongly blurring edges. Although there is little work for speeding up weighted MF [19, 34], unweighted MF has several solutions [2, 15, 22, 29]. Generally, the main idea of above accelerating methods focuses on reducing the computational complexity per pixel to the optimal $O(1)$. However, the minimal linear complexity of GF leaves no room to reduce the computational complexity further. To accelerate GF, He *et al*. [9] remind that GF can be sped up by subsampling to subset the amount of data processed. But this strategy inevitably introduces approximation error. A different way is to implement the algorithm of GF on VLSI directly [14]. But Kao *et al*. [14] only give a gray guidance based GF implementation as existing matrix inverse algorithms are too complicated to implement on hardware. Different from other algorithms, our hardware-efficient matrix inverse algorithm not only can fully exploit current parallel processors but also can be easily implemented.

Improvement: Due to nice visual quality, fast speed, and ease of implementation, GF since its birth has received much effort to make it much better. In 2012, Lu *et al*. [18] pointed out that the box window adopted by GF is not geometric-adaptive and design a adaptive support region to take place of it. However, Tan *et al*. [26] argued that the new designed adaptive support region depends on the scanning order and put forward a further improvement for the support region. In addition, they introduced a quadratic spatial regularization term to GF. But the filtering speed is very slow because they employ traditional algorithms to invert 5×5 matrices. Later, Dai *et al*. [4] successfully incorporated spatial similarity into GF without significant performance degeneration. Unlike previous methods, Li *et al*. [17] modified GF to adapt it to suppress halo. Through careful investigation, we can find that all these improvements do not make any modification for the gray/color guidance or incorporate more complicated nonlinear model to GF to achieve better results. This is because these modifications will increase the matrix size and therefore make the run time unacceptable. Note that even though all of them only take into account the linear model, their speeds are still slower than GF [4]. Benefiting from our hardware-efficient matrix inverse algorithm, we realizes these modifications in HGF without sacrificing efficiency.

3. Technology Brief

In this section, we outline our matrix inverse algorithm and its application in computing $\vec{w}_\mathfrak{p}$ in Eq (2). Before the formal discussion, we kindly remind readers to refer to Table 1 for symbol conventions.

The key step of HGF is to calculate the vector $\vec{w}_\mathfrak{p}$ according to Eq (2) which contains the matrix inverse operation $(\lambda \boldsymbol{E} + \boldsymbol{X}_\mathfrak{p}^T \boldsymbol{X}_\mathfrak{p})^{-1}$. Here $\boldsymbol{E}$ denotes an identity matrix

$$\vec{w}_\mathfrak{p} = (\lambda \boldsymbol{E} + \boldsymbol{X}_\mathfrak{p}^T \boldsymbol{X}_\mathfrak{p})^{-1} \boldsymbol{X}_\mathfrak{p}^T \vec{c}_{n+1,\mathfrak{p}} \qquad (2)$$

and $\boldsymbol{X}_\mathfrak{p} = [\vec{c}_{0,\mathfrak{p}}, \cdots, \vec{c}_{n,\mathfrak{p}}]$. As for the vector $\vec{c}_{i,\mathfrak{p}}$, we put $\vec{c}_{i,\mathfrak{p}} = [\boldsymbol{G}_i(\mathfrak{q}_1), \cdots, \boldsymbol{G}_i(\mathfrak{q}_{|\Omega_\mathfrak{p}|})]^T (0 \le i \le n+1)$ to record values $\boldsymbol{G}_i(\mathfrak{q}_k)$, where $\mathfrak{q}_k \in \Omega_\mathfrak{p}$, $\Omega_\mathfrak{p}$ presents a neighborhood centered at the pixel $\mathfrak{p}$, $|\Omega_\mathfrak{p}|$ indicates the total number of pixels in $\Omega_\mathfrak{p}$ and $\boldsymbol{G}_i$ ($0 \le i \le n+1$) is an image.

To invert $\lambda \boldsymbol{E} + \boldsymbol{X}_\mathfrak{p}^T \boldsymbol{X}_\mathfrak{p}$ efficiently, we will substitute $\lambda \boldsymbol{E} + \boldsymbol{X}_\mathfrak{p}^T \boldsymbol{X}_\mathfrak{p}$ with $\lambda \boldsymbol{E} + \sum_{i=0}^n \vec{c}_{i,\mathfrak{p}} \vec{c}_{i,\mathfrak{p}}^T$ by reformulating Eq (2) as Eq (3). This is because $(\lambda \boldsymbol{E} + \sum_{i=0}^n \vec{c}_{i,\mathfrak{p}} \vec{c}_{i,\mathfrak{p}}^T)^{-1}$

$$\begin{aligned} \vec{w}_\mathfrak{p} &= \boldsymbol{X}_\mathfrak{p}^T (\lambda \boldsymbol{E} + \boldsymbol{X}_\mathfrak{p} \boldsymbol{X}_\mathfrak{p}^T)^{-1} \vec{c}_{n+1,\mathfrak{p}} \\ &= [\vec{c}_{0,\mathfrak{p}}^T, \cdots, \vec{c}_{n,\mathfrak{p}}^T]^T (\lambda \boldsymbol{E} + \sum_{i=0}^n \vec{c}_{i,\mathfrak{p}} \vec{c}_{i,\mathfrak{p}}^T)^{-1} \vec{c}_{n+1,\mathfrak{p}} \end{aligned} \quad (3)$$

can be interpreted as a linear combination of outer products $\vec{c}_{i,\mathfrak{p}} \vec{c}_{i,\mathfrak{p}}^T$ (4) according to following Proposition 1.

Proposition 1. *If* $\lambda \boldsymbol{E} + \sum_{i=0}^n \vec{c}_{i,\mathfrak{p}} \vec{c}_{i,\mathfrak{p}}^T$ *is invertible, then*

$$(\lambda \boldsymbol{E} + \sum_{i=0}^n \vec{c}_{i,\mathfrak{p}} \vec{c}_{i,\mathfrak{p}}^T)^{-1} = \lambda^{-1} \boldsymbol{E} + \sum_{i,j=0}^n \alpha_{ij,\mathfrak{p}} \vec{c}_{i,\mathfrak{p}} \vec{c}_{j,\mathfrak{p}}^T \quad (4)$$

where $\alpha_{ij,\mathfrak{p}} = \alpha_{ij,\mathfrak{p}}^n$ *which can be iteratively computed by*

$$\alpha_{ij,\mathfrak{p}}^\kappa = \begin{cases} \gamma_\mathfrak{p}^\kappa F_{ij,\mathfrak{p}}^\kappa + \alpha_{ij,\mathfrak{p}}^{\kappa-1} & i < \kappa, j < \kappa \\ \lambda^{-1} \gamma_\mathfrak{p}^\kappa (\sum_{n=0}^{\kappa-1} \alpha_{in,\mathfrak{p}}^{\kappa-1} G_{nj,\mathfrak{p}}) & i < \kappa, j = \kappa \\ \lambda^{-1} \gamma_\mathfrak{p}^\kappa (\sum_{m=0}^{\kappa-1} \alpha_{mj,\mathfrak{p}}^{\kappa-1} G_{im,\mathfrak{p}}) & i = \kappa, j < \kappa \\ \lambda^{-2} \gamma_\mathfrak{p}^\kappa & i = j = \kappa \end{cases} \quad (5)$$

from $\kappa = 1$ *to* $\kappa = n$ *with* $\alpha_{00,\mathfrak{p}}^0 = -(\lambda + G_{00,\mathfrak{p}})^{-1}$, $G_{ij,\mathfrak{p}} = \vec{c}_{i,\mathfrak{p}}^T \vec{c}_{j,\mathfrak{p}}$, $F_{ij,\mathfrak{p}}^\kappa = \sum_{m,n=0}^{\kappa-1} \alpha_{im,\mathfrak{p}}^{\kappa-1} \alpha_{nj,\mathfrak{p}}^{\kappa-1} G_{m\kappa,\mathfrak{p}} G_{\kappa n,\mathfrak{p}}$ *and* $\gamma_\mathfrak{p}^\kappa = -(1 + \lambda^{-1} G_{\kappa\kappa,\mathfrak{p}} + \sum_{m,n=0}^{\kappa-1} \alpha_{mn,\mathfrak{p}}^{\kappa-1} G_{\kappa m,\mathfrak{p}} G_{n\kappa,\mathfrak{p}})^{-1}$.

Proof. The recursive formula is derived from the Sherman-Morrison formula [25]. Constrained by maximal 8 pages constraint, we leave derivation details to supplementary materials. ∎

Further, putting Eq (4) into Eq (3), we are able to transform the kth element $\vec{w}_\mathfrak{p}(k)$ of $\vec{w}_\mathfrak{p}$ to a linear combination

$\mathcal{F}$	Calligraphic means that $\mathcal{F}$ is a function.
i	Italics emphasizes that the variable i is a scalar.
$\mathfrak{p}$	Fraktur denotes $\mathfrak{p}$ is an image pixel.
$\vec{v}$	Arrow $\vec{}$ imlies v is a vector.
$\boldsymbol{X}$	Bold (disregarding case) indicates $\boldsymbol{X}$ is a matrix.
$i_\mathfrak{p} \backslash \vec{v}_\mathfrak{p} \backslash \boldsymbol{X}_\mathfrak{p}$	The subscript $_\mathfrak{p}$ suggests $i_\mathfrak{p} \backslash \vec{v}_\mathfrak{p} \backslash \boldsymbol{X}_\mathfrak{p}$ is the scalar\vector\matrix located at $\mathfrak{p}$.
$\vec{v}_{i,\mathfrak{p}} \backslash \boldsymbol{X}_{i,\mathfrak{p}}$	The subscript $_{i,\mathfrak{p}}$ suggests $\vec{v}_{i,\mathfrak{p}} \backslash \boldsymbol{X}_{i,\mathfrak{p}}$ is the ith vector\matrix of $\mathfrak{p}$.
$\vec{v}(j) \backslash \vec{v}_\mathfrak{p}(j) \backslash \vec{v}_{i,\mathfrak{p}}(j)$	$\vec{v}(j) \backslash \vec{v}_\mathfrak{p}(j) \backslash \vec{v}_{i,\mathfrak{p}}(j)$ is the jth element of $\vec{v} \backslash \vec{v}_\mathfrak{p} \backslash \vec{v}_{i,\mathfrak{p}}$.
$\boldsymbol{X}(\mathfrak{p}) \backslash \boldsymbol{X}_i(\mathfrak{p})$	$\boldsymbol{X}(\mathfrak{p}) \backslash \boldsymbol{X}_i(\mathfrak{p})$ is the element of $\boldsymbol{X} \backslash \boldsymbol{X}_i$ located at $\mathfrak{p}$.

Table 1: Symbol Conventions

$$\begin{aligned} \vec{w}_\mathfrak{p}(k) &= \vec{c}_{k,\mathfrak{p}}^T (\lambda \boldsymbol{E} + \sum_{i=0}^n \vec{c}_{i,\mathfrak{p}} \vec{c}_{i,\mathfrak{p}}^T)^{-1} \vec{c}_{n+1,\mathfrak{p}} \\ &= \vec{c}_{k,\mathfrak{p}}^T (\lambda^{-1} \boldsymbol{E} + \sum_{i,j=0}^n \alpha_{ij,\mathfrak{p}} \vec{c}_{i,\mathfrak{p}} \vec{c}_{j,\mathfrak{p}}^T) \vec{c}_{n+1,\mathfrak{p}} \\ &= \lambda^{-1} I_{k\,n+1,\mathfrak{p}} + \sum_{i,j=0}^n \alpha_{ij,\mathfrak{p}} I_{ki,\mathfrak{p}} I_{j\,n+1,\mathfrak{p}} \qquad (6) \end{aligned}$$

of $G_{ij,\mathfrak{p}}$ as illustrated in Eq (6). According to Proposition 2, $G_{ij,\mathfrak{p}}$ equals to the box filtering result $\boldsymbol{G}_{ij}(\mathfrak{p})$ of the element-wise production image $\boldsymbol{G}_i \boldsymbol{G}_j$. Then $\vec{w}_\mathfrak{p}(k)$ can be synthesized by composing a set of box filtering results.

Proposition 2. *if the neighborhood $\Omega_\mathfrak{p}$ of $\mathfrak{p}$ is a box window, we have $G_{ij,\mathfrak{p}} = \boldsymbol{G}_{ij}(\mathfrak{p})$ which is the box filtering result of the element-wise production image $\boldsymbol{G}_i \boldsymbol{G}_j$ in $\Omega_\mathfrak{p}$.*

Proof. We have the dot production $G_{ij,\mathfrak{p}} = \vec{c}_{i,\mathfrak{p}}^T \vec{c}_{j,\mathfrak{p}} = \sum_{k=1}^{|\Omega_\mathfrak{p}|} \boldsymbol{G}_i(\mathfrak{q}_k) \boldsymbol{G}_j(\mathfrak{q}_k) = \sum_{\mathfrak{q} \in \Omega_\mathfrak{p}} \boldsymbol{G}_i(\mathfrak{q}) \boldsymbol{G}_j(\mathfrak{q})$ and the box filtering result $\boldsymbol{G}_{ij}(\mathfrak{p}) = \sum_{\mathfrak{q} \in \Omega_\mathfrak{p}} \boldsymbol{G}_i \boldsymbol{G}_j(\mathfrak{q}) = \sum_{\mathfrak{q} \in \Omega_\mathfrak{p}} \boldsymbol{G}_i(\mathfrak{q}) \boldsymbol{G}_j(\mathfrak{q})$. So we conclude $G_{ij,\mathfrak{p}} = \boldsymbol{G}_{ij}(\mathfrak{p})$. ∎

Above all, the kth element $\vec{w}_\mathfrak{p}(k)$ of $\vec{w}_\mathfrak{p}$ can be figured out in two steps:

1. applying box filter to element-wise production image $\boldsymbol{G}_i \boldsymbol{G}_j$ for $0 \le i, j \le n+1$ to produce $\boldsymbol{G}_{ij}$;

2. computing $\vec{w}_\mathfrak{p}(k)$ according to the linear combination (6) of $\boldsymbol{G}_{ij}$.

Then applying the procedure to each $\vec{w}_\mathfrak{p}(k)$, we will compose the vector $\vec{w}_\mathfrak{p}$. At last, we note that above two steps computing procedure is only composed by arithmetic calculation and box filtering. The matrix inverse operation is completely eliminated in the computation procedure.
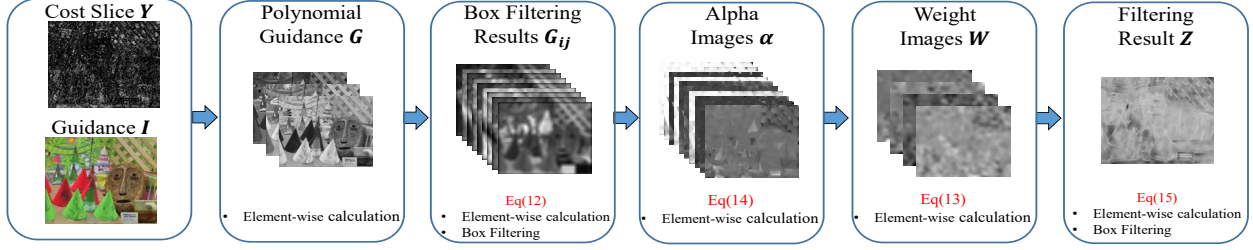
Figure 2: **Filtering Flowchart of HGF with the Input from a Multi-Label System**. At first, we exploit the input guidance to synthesize a polynomial guidance. Then the input cost slice together with the polynomial guidance are used to produce box filtering images G_{ij} (12). After that we compute α_{ij} and W_i from G_{ij} according to Eq (14) (13). At last, the filtering result is yielded by Eq (15). Note that each step only involves element-wise arithmetic computation or box filtering.

4. Hardware-Efficient Guided Filter

4.1. Defintion

In this section, we define HGF and present its hardware-efficient implementation in detail. Fig 2 plots the overall computing procedure to give a general idea for our HGF.

The key assumption of HGF is the generalized linear model (7) between the synthesized n-channel guidance G and the output Z, where $\vec{w}_{\mathfrak{p}}$ is constant in the window $\Omega_{\mathfrak{p}}$.

$$Z(\mathfrak{q}) = \sum_{i=1}^{n} \vec{w}_{\mathfrak{p}}(i) G_i(\mathfrak{q}) + \vec{w}_{\mathfrak{p}}(0), \ \forall \mathfrak{q} \in \Omega_{\mathfrak{p}} \quad (7)$$

According to the local multipoint filtering framework [16], HGF comprises two major steps: 1) *multipoint estimation:* calculating the estimates for a set of points within a local support, and 2) *aggregation:* fusing multipoint estimates available for each point. Specifically, in the first step, HGF estimates the coefficients $\vec{w}_{\mathfrak{p}}$ of model (7) by minimizing the linear ridge regression (8), where Y denotes an input image and the closed-form solution of the minimizer $\vec{w}_{\mathfrak{p}}$ is Eq (2). In the second step, HGF puts the

$$\min_{\vec{w}_{\mathfrak{p}}} \lambda \|\vec{w}_{\mathfrak{p}}\|_2^2 + \sum_{\mathfrak{q} \in \Omega_{\mathfrak{p}}} \left(Y(\mathfrak{q}) - \sum_{i=1}^{n} \vec{w}_{\mathfrak{p}}(i) G_i(\mathfrak{q}) - \vec{w}_{\mathfrak{p}}(0) \right)^2 \ (8)$$

minimizer $\vec{w}_{\mathfrak{p}}$ of optimization (8) into Eq (7) and obtains a set of values $Z'_{\mathfrak{p}}(\mathfrak{q}) = \sum_{i=1}^{n} \vec{w}_{i,\mathfrak{p}} G_i(\mathfrak{q}) + \vec{w}_{0,\mathfrak{p}}, \mathfrak{q} \in \Omega_{\mathfrak{p}}$ for a given window $\Omega_{\mathfrak{p}}$. So each pixel $\mathfrak{q}$ has $|\Omega_{\mathfrak{p}}|$ values. HGF aggregates these values together and considers their mean $\frac{1}{|\Omega_{\mathfrak{q}}|} \sum_{\mathfrak{p} \in \Omega_{\mathfrak{q}}} Z'_{\mathfrak{p}}(\mathfrak{q})$ as final filtering result $Z(\mathfrak{q})$. So we have

$$\begin{aligned} Z(\mathfrak{q}) &= \frac{1}{|\Omega_{\mathfrak{q}}|} \sum_{\mathfrak{p} \in \Omega_{\mathfrak{q}}} \sum_{i=1}^{n} \vec{w}_{\mathfrak{p}}(i) G_i(\mathfrak{q}) + \vec{w}_{\mathfrak{p}}(0) \\ &= \sum_{i=1}^{n} \vec{w}_{\mathfrak{q}}^a(i) G_i(\mathfrak{q}) + \vec{w}_{\mathfrak{q}}^a(0) \end{aligned} \quad (9)$$

where $\vec{w}_{\mathfrak{q}}^a = \frac{1}{|\Omega_{\mathfrak{q}}|} \sum_{\mathfrak{p} \in \Omega_{\mathfrak{q}}} \vec{w}_{\mathfrak{p}}$ is the average of $\vec{w}_{\mathfrak{p}}$ in $\Omega_{\mathfrak{q}}$.

4.2. Polynomial Guidance

GF exploits the linear model (1) to estimate filtering results from an input guidance. However, it is unreasonable to expect that the linear model works well under all circumstances as the linear models (1) of GF are apt to underfits data due to its simplicity and thus produces over-smoothing filtering results. We address the problem by synthesizing a polynomial guidance from a raw image.

The polynomial guidance is designed to introduce nonlinearity to the generalized linear model (7). For example, Eq (10) shows the polynomial model with a gray input

$$Z(\mathfrak{q}) = \sum_{i=1}^{d} \vec{w}_{\mathfrak{p}}(i) I^i(\mathfrak{q}) + \vec{w}_{\mathfrak{p}}(0), \ \forall \mathfrak{q} \in \Omega_{\mathfrak{p}} \quad (10)$$

guidance I, where d is the degree of the polynomial function. Compared with the linear model (7), we can find the equivalence between the linear model (7) and the polynomial model (10) if we assume $G_i = I^i$. This inspires us to introduce nonlinearity to HGF by synthesizing a polynomial guidance, where each channel of the synthesized polynomial guidance is produced by a polynomial function operated on a channel of the input guidance.

This method can be easily extended to the multichannel guidance. In the case when the input guidance is multichannel, it is straightforward to apply the map procedure $G_{(i-1)d+j} = I_i^j$ to each channel independently, where I_i denotes the ith channel of the multichannel guidance I and n is the channel number of I. After that we stack produced results of each channel of the input multichannel guidance I to synthesize our polynomial guidance. Mathematically, the linear model (7) in this situation is equivalent to following nonlinear polynomial model (11). So we successfully endow the nonliearity to the generalized linear model (7) of HGF.

$$Z(\mathfrak{q}) = \sum_{i=1}^{n} \sum_{j=1}^{d} \vec{w}_{\mathfrak{p}}((i-1)d+j) I_i^j(\mathfrak{q}) + \vec{w}_{\mathfrak{p}}(0) \quad (11)$$

4.3. Hardware-Efficient Implementation

Here we present the matrix inverse algorithm tailored to $(\lambda \boldsymbol{E} + \boldsymbol{X}_{\mathfrak{p}}^T \boldsymbol{X}_{\mathfrak{p}})^{-1}$ to make full use of the parallel computing ability of hardwares. HGF in the first step calculates the minimizer (2) of the linear ridge regression (8). However traditional matrix inverse algorithms cannot fully take advantages of the parallel computing hardwares. Fortunately, Eq (6) sheds light on a hardware-efficient way to compute $\vec{w}_{\mathfrak{p}}$ because it guarantees the kth element $\vec{w}_{\mathfrak{p}}(k)$ of $\vec{w}_{\mathfrak{p}}$ is the linear combination of the box filtering result $\boldsymbol{G}_{ij}(\mathfrak{p})$. Specifically, putting $\mathcal{B}(\boldsymbol{X})$ denote the box filtering result of an image $\boldsymbol{X}$, $\boldsymbol{W}_i$ and $\boldsymbol{\alpha}_{ij}$ record all values of $\vec{w}_{\mathfrak{p}}(i)$ and $\alpha_{ij,\mathfrak{p}}$ for arbitrary $\mathfrak{p}$ in the image domain (*i.e.* $\boldsymbol{W}_i(\mathfrak{p}) = \vec{w}_{\mathfrak{p}}(i)$, $\boldsymbol{\alpha}_{ij}(\mathfrak{p}) = \alpha_{ij,\mathfrak{p}}$), we can generalize Eq (6) to following element-wise arithmetic calculations (13) of box filtering results $\boldsymbol{G}_{ij}$ (12). Here $\boldsymbol{G}_0$ denotes matrix of ones, $\boldsymbol{G}_i(1 \leq i \leq n)$ stand for the ith channel of the synthesized polynomial n-channel guidance $\boldsymbol{G}$, $\boldsymbol{G}_{n+1}$ be the alias of the input image $\boldsymbol{Y}$. Similarly, the updat-

$$\boldsymbol{G}_{ij} = \mathcal{B}(\boldsymbol{G}_i \boldsymbol{G}_j) \tag{12}$$

$$\boldsymbol{W}_i = \lambda^{-1}\boldsymbol{G}_{\kappa n+1} + \sum_{i,j=0}^{n} \boldsymbol{\alpha}_{ij}\boldsymbol{G}_{\kappa i}\boldsymbol{G}_{jn+1} \tag{13}$$

ing formulate of $\alpha_{ij,\mathfrak{p}}^{\kappa}$ can also be modified to the element-wise arithmetic calculation of matrices (14), where $\boldsymbol{\alpha}_{00}^0 =$

$$\boldsymbol{\alpha}_{ij}^{\kappa} = \begin{cases} \boldsymbol{F}_{\mathfrak{p}}^{\kappa} + \boldsymbol{\alpha}_{ij}^{\kappa-1} & i < \kappa, j < \kappa \\ \lambda^{-1}\boldsymbol{\gamma}^{\kappa}\sum_{n=0}^{\kappa-1}\boldsymbol{\alpha}_{in}^{\kappa-1}\boldsymbol{G}_{n\kappa} & i < \kappa, j = \kappa \\ \lambda^{-1}\boldsymbol{\gamma}^{\kappa}\sum_{m=0}^{\kappa-1}\boldsymbol{\alpha}_{mj}^{\kappa-1}\boldsymbol{G}_{\kappa m} & i = \kappa, j < \kappa \\ \lambda^{-2}\boldsymbol{\gamma}^{\kappa} & i = j = \kappa \end{cases} \tag{14}$$

$-(\lambda + \boldsymbol{G}_{00})^{-1}$, $\boldsymbol{F}^{\kappa} = \sum_{m,n=0}^{\kappa-1}\boldsymbol{\alpha}_{im}^{\kappa-1}\boldsymbol{\alpha}_{nj}^{\kappa-1}\boldsymbol{G}_{m\kappa}\boldsymbol{G}_{\kappa n}$ and $\boldsymbol{\gamma}^{\kappa} = -(1 + \lambda^{-1}\boldsymbol{G}_{\kappa\kappa} + \sum_{m,n=0}^{\kappa-1}\boldsymbol{\alpha}_{mn}^{\kappa-1}\boldsymbol{G}_{\kappa m}\boldsymbol{G}_{n\kappa})^{-1}$.

HGF in the second step computes filtering results $\boldsymbol{Z}(\mathfrak{p})$ according to the average of coefficients $\vec{w}_{\mathfrak{p}}$. Define the average operator $\mathcal{A}(\boldsymbol{X}) = \mathcal{B}(\boldsymbol{X})/\mathcal{B}(\boldsymbol{G}_0)$, we can formulate the element-wise arithmetic calculation form of Eq (9) as

$$\boldsymbol{Z} = \sum_{i=1}^{n} \mathcal{A}(\boldsymbol{W}_i)\boldsymbol{G}_i + \mathcal{A}(\boldsymbol{W}_0) \tag{15}$$

Taking a closer look at Eq (12) (13) (14) (15), we find that all equations only involve two computation types (Fig 2 sums up computation types in each step and overall computation flowchart): one is element-wise arithmetic calculations of matrices, the other is box filtering for images. Both computations can easily exert the parallelism power of current hardware. More importantly, there are highly optimized libraries for CPU and GPU. In the next, we present how to implement them in a hardware-efficient way.

Element-wise arithmetic calculations is a typical data parallel task because it distributes the operation over ele-

ments of a matrix. In parallel computing literatures, it belongs to the map pattern [20] which applies an elemental function to an actual collection of input data. Current CPU and GPU do well in these calculations as the SIMD instructions of CPU and GPU can benefit from same computations done on different pieces of data in the data parallel task. Although inverting matrices simultaneously belongs to the map pattern too, it cannot be as efficient as our method because inverting operation is not directly supported by hardware and thus can not benefit from SIMD instructions. In contrast the arithmetic computations can be directly allocated to a core of CPU or thread of GPU for parallel computing. Finally, we note that element-wise arithmetic calculations are supported by many softwares or libraries such as such as Matlab, ViennaCL [24] and Arrayfire [31].

Box filtering produces a smoothing image of which has a value equal to the sum of its neighboring pixels in the input image and can be computed in linear time from the Summed Area Table [3]. Different from element-wise calculation, box filtering belongs to the stencil pattern [20] that is a generalization of the map pattern in which an elemental function can access not only a single element in an input collection but also a set of neighbors. A detailed description on how to implement the stencil pattern efficiently on parallel devices can be found in the book [20]. Fortunately, we do not need to implement the box filtering manually since two libraries OpenCV of Intel and NPP of Nvidia already have offered.

5. More Discussion for HGF

In this section, we clarity the connection and differences between HGF and GF. Moreover, the performance and implementation concerns of HGF will also be discussed.

5.1. Connection with GF

Our matrix inverse algorithm can also be used to speed up GF. Specifically, GF minimizes optimization (16) to determine the coefficient $\vec{w}_{\mathfrak{p}}$, where $\vec{w}_{\mathfrak{p}}$ is a 4 elements vector. Let $\vec{c}_{i,\mathfrak{p}}$ be a n element vector extracted from $\boldsymbol{I}_i$ in

$$\min_{\vec{w}_{\mathfrak{p}}} \lambda \sum_{i=1}^{3} \vec{w}_{\mathfrak{p}}^2(i) + \sum_{\mathfrak{q}\in\Omega_{\mathfrak{p}}} \left(\boldsymbol{Y}(\mathfrak{q}) - \sum_{i=1}^{3}\vec{w}_{\mathfrak{p}}(i)\boldsymbol{I}_i(\mathfrak{p}) - \vec{w}_{\mathfrak{p}}(0)\right)^2 \tag{16}$$

$\Omega_{\mathfrak{p}}$, $\boldsymbol{X}_{\mathfrak{p}} = [\vec{c}_{1,\mathfrak{p}}, \vec{c}_{2,\mathfrak{p}}, \vec{c}_{3,\mathfrak{p}}]$, $\vec{\mathbf{1}}$ denote the identity vector, $\vec{x}_{\mathfrak{p}} = \frac{1}{n}\boldsymbol{X}^T\vec{\mathbf{1}}$, $c_{n+1,\mathfrak{p}} = \frac{1}{n}\vec{c}_{n+1,\mathfrak{p}}^T\vec{\mathbf{1}}$, $\boldsymbol{X}_{\mathfrak{p}}'^T = \boldsymbol{X}_{\mathfrak{p}}^T - \vec{\mathbf{1}}\vec{x}_{\mathfrak{p}}^T$ and $\vec{c}_{n+1,\mathfrak{p}}' = \vec{c}_{n+1,\mathfrak{p}} - c_{n+1,\mathfrak{p}}\vec{\mathbf{1}}$, then $\vec{w}_{\mathfrak{p}}(0) = c_{n+1,\mathfrak{p}} - \vec{w}_{\mathfrak{p}}^T\vec{x}_{\mathfrak{p}}$. and the last three terms of $\vec{w}_{\mathfrak{p}}$ can be computed by Eq (17).

$$[\vec{w}_{\mathfrak{p}}(1), \vec{w}_{\mathfrak{p}}(2), \vec{w}_{\mathfrak{p}}(3)]^T = (\lambda\boldsymbol{E} + \boldsymbol{X}_{\mathfrak{p}}'^T\boldsymbol{X}_{\mathfrak{p}}')^{-1}\boldsymbol{X}_{\mathfrak{p}}'^T\vec{c}_{n+1,\mathfrak{p}}' \tag{17}$$

The form of Eq (17) is same to Eq (2). Hence we can apply the same technique to invert the matrix.

5.2. Differences From GF

HGF is similar to GF but not same to it. Simply put, there are three major differences between the two filters:

The guidance is different. GF directly takes a gray/color image as its guidance I. Contrarily, HGF synthesizes a multi-channels polynomial guidance G from a gray or color input image. With increasing channel number, the guidance carries more and more information.

The cost function is different. The regularization term of GF does not punish $\vec{w}_{\mathfrak{p}}(0)$. In contrast, HGF treats all elements of $\vec{w}_{\mathfrak{p}}$ equally in the cost function (8) as the minimizer (3) can be computed in a hardware-efficient way and the final result of HGF in the multi-label system is much better according to our experiments.

The matrix inverse algorithm is different. GF does not take any special measure to accelerate the speed of inverting matrices. On the contrary, HGF designs an effective matrix inverse algorithm to fully utilize the parallel ability of current hardwares. As a result, the computational procedure of HGF only comprises two hardware-efficient operations: element-wise arithmetic calculations and box filtering.

5.3. Performance Evaluation

Our experiments are conducted on a Laptop with an Intel i7 CPU and GTX 960 GPU. In order to compare the running performance fairly and comprehensively, we exploit two C++ libraries (*i.e.* OpenCV and Arrayfire [31]) to implement GF and HGF, where the Summed Area Table [3] accelerated box filtering and the three matrix inverse algorithms are incorporated into the two filters. Here the three matrix inverse algorithms are:

- **Built-in LU matrix inverse algorithm:** We use the built-in matrix inverse function "inv" of OpenCV to compute final filtering results of GF. The built-in function takes a highly optimized LU matrix inverse algorithm and the input matrix can be arbitrary size.

- **Analytic solution of matrix inverse algorithm:** The closed-form solution of each invertible matrix can be expressed by adjugate matrices. It is an efficient way to calculate the inverse of small matrices but is inefficient for large matrices. The built-in function "Guided-Filter" of OpenCV takes it to compute color-guidance

Algorithms	3	5	7	9
GF_1	$0.85s$	$1.36s$	$2.01s$	$2.83s$
GF_2	$0.17s$	$0.35s$	/	/
HGF	$0.08s$	$0.16s$	$0.25s$	$0.43s$

Table 2: Execution time to filter $10^3 \times 10^3$ gray images with different guidances on CPU, where the channel number of guidance varies from 3 to 10.
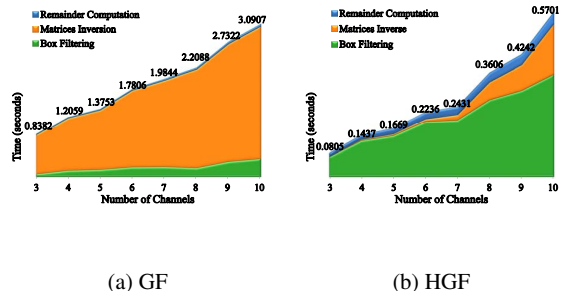

(a) GF   (b) HGF

Figure 3: **Execution Time Anatomy:** (a) (b) illustrate run times of the box filtering in preparing $\lambda E + X_{\mathfrak{p}}^T X_{\mathfrak{p}}$, inverting this matrix and remainder computation of GF and HGF, where the green, orange and blue regions denote the run time of box filtering, matrix inverse and the remainder computation.

results of GF. We only extend it to inverse 5×5 matrices because the matrix inverse code becomes much complex when the matrix size is large than 5 and thus it is nearly impossible to write the code of inverting 7×7 and 9×9 matrices by hand (the analytic solutions of 7×7 and 9×9 matrix inverse are listed in the supplemental material for demonstration).

- **Our hardware-efficient matrix inverse algorithm:** We incorporate it with our HGF to produce final results. Note that unlike the first algorithm that can invert arbitrary matrices and the second algorithm that only suits to a small matrix, our method can invert a arbitrary size matrix only if the matrix is invertible and can be presented by $\lambda E + \sum_{i=0}^{n} \vec{c}_{i,\mathfrak{p}} \vec{c}_{i,\mathfrak{p}}^T$.

OpenCV is taken to reimplement GF according to its public Matlab code, where the C++ code is basically a line-by-line translation of the Matlab code and the matrix inverse algorithm is provided by the built-in "inv" function of OpenCV and our implementation according to the analytic solution of matrix inverse. We denote GF adopting the two implementations as GF_1 and GF_2. Since both the algorithm of GF and its Matlab code published by He *et al.* [10] do not consider parallelism or SIMD instructions, we compile the C++ code of GF with default complier option. Similarly, we do not take tricks to speed up HGF: the code of HGF is also compiled by default compiler option and all parallel computing infrastructure is provided by Arrayfire.

The GF_1 and HGF rows of Table 2 report execution times of GF and HGF for a $10^3 \times 10^3$ guidance with increasing channel number on CPU. According to our experiments, HGF is much faster than GF. Due to the smaller running cost of our matrix inverse algorithm, it is more efficient even if channel number of guidance is large. We also test the GF code using analytic solution of matrix inverse. The experimental data for $3/5$-channel guidances are reported in the

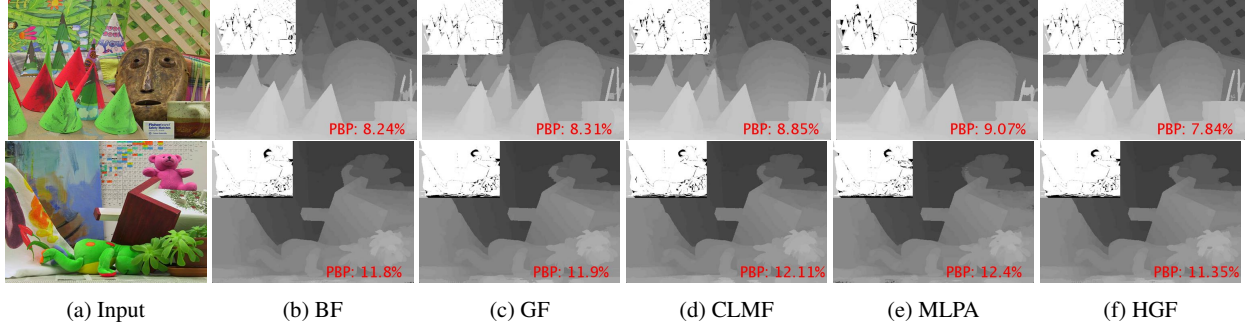| | | | | | | |
|---|---|---|---|---|---|
| (a) Input | (b) BF | (c) GF | (d) CLMF | (e) MLPA | (f) HGF |

Figure 5: **Stereo Matching Results:** (a) is input color image. (b) (c) (d) (e) and (f) illustrate the stereo matching results of BF, GF, CLMF, MLPA and HGF, respectively. In the left upper corners of (b) (c) (d) (e) and (f), we show comparisons with the respective groundtruth (error larger than one pixel in nonoccluded are denoted by black and in occlude regions are denoted by gray) and the Percentage of Bad Pixel (PBP) scores are listed in the bottom of each image.

Size/n	3	5	7	9
250×250	$3.9ms$	$10.7ms$	$39.9ms$	$110.2ms$
500×500	$3.7ms$	$11.1ms$	$40.1ms$	$112.4ms$
1000×1000	$4.1ms$	$10.9ms$	$41.3ms$	$115.9ms$
2000×2000	$4.3ms$	$11.2ms$	$42.7ms$	$121.8ms$

Table 3: Execution time of HGF on Nvidia GPU with increasing guidances channel number n, where the input is a gray image with increasing sizes, the channel number of guidance varies from 3 to 9.

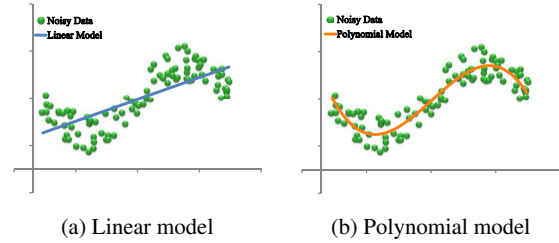| (a) Linear model	(b) Polynomial model

Figure 4: **Fitting curves of linear model and nonlinear polynomial model on noisy data.** According to the fitting curves, the polynomial model is more flexible than the linear model.

GF$_2$ row. We can observe the speed of the analytic solution is faster than generalized matrix inverse function "inv" function of OpenCV, but is still slower than our matrix inverse algorithm. In short, the analytic solution achieves a relative fast speed by paying an extremely heavy cost to translate it into element-wise calculation.

Box filtering takes up most of the run time of HGF. The area graph in Fig 3 displays the execution time of box filtering in preparing matrices $\lambda \boldsymbol{E} + \boldsymbol{X}_\mathfrak{p}^T \boldsymbol{X}_\mathfrak{p}$ (2), inverting these matrices and remainder computation in GF and HGF, where the green denotes the box filtering time, the orange presents matrix inverse time and the horizontal axis is the channel number of guidance. We can observe that box filtering in HGF consumes more than half of the time. This inspires us to precompute the box filtering results $\boldsymbol{G}_{ij} = \mathcal{B}(\boldsymbol{G}_i \boldsymbol{G}_j)$ of all pair-wise production image $\boldsymbol{G}_i \boldsymbol{G}_j$ in Eq (13) (14) (15) to accelerate the computational speed of HGF further. Unlike HGF, matrix inverse operation in GF spends most of the time. So GF cannot take the same strategy to reduce its run time. At last, we note that an unignorable part of the overall execution time of HGF is spent by the "remainder" part denoted by the blue in Fig (3) and most time of this part is payed for box filtering as Eq (15) involves n box filters. Here n is the channel number of the guidance.

To demonstrate the actual power of our algorithm, we accelerate HGF by precomputing all-pair box filtering results $\boldsymbol{G}_{ij} = \mathcal{B}(\boldsymbol{G}_i \boldsymbol{G}_j)$. Table 3 reports the execution time tested on GPU with increasing guidance channel number. For guidance images of which the channel number is smaller than 7, the performance is realtime. It is also worth noting that the performance of HGF scales well with the size of input image (*i.e.* the execution time does not change with the data size). We contributes this ability to the parallel computing ability of HGF.

6. Experiments and Applications

To demonstrate the ability of our HGF, we apply it to three classic multi-label problems: stereo matching, optical flow and image segmentation. To show the robustness of our method, we use following, the same constant parameter setting to generate our results: ($\lambda = 0.05, r = 7$), where r indicates the radius of box window and λ balances the two terms in Eq (8). In the following experiments, each channel $\boldsymbol{G}_i$ of the polynomial guidance $\boldsymbol{G}$ is synthesized by the mapping $\boldsymbol{G}_{(i-1)d+j} = \boldsymbol{I}_i^j$ ($1 \leq j \leq 2$) with a color

(a) Input   (b) BF   (c) GF   (d) CLMF   (e) MLPA   (f) HGF
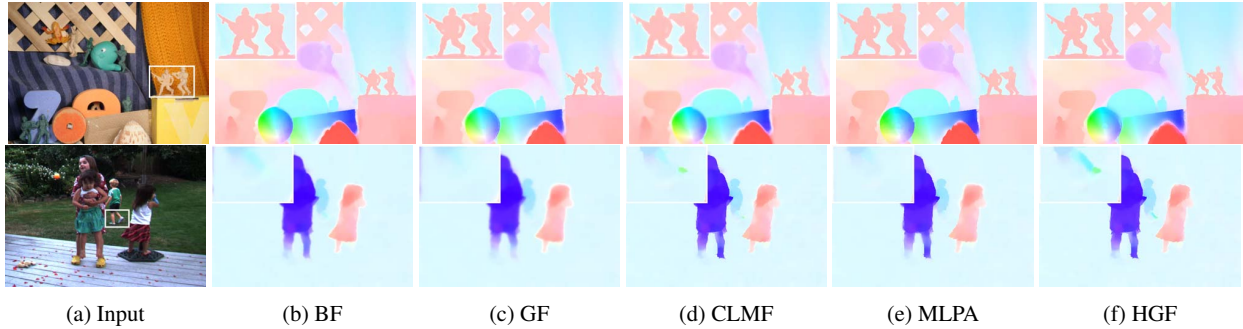
Figure 6: **Optical Flow Estimation:** (a) is input color image. (b) (c) (d) (e) and (f) demonstrate the stereo matching results of BF, GF, CLMF, MLPA and HGF, respectively, where the white box denotes the close-up region for visual comparison.

input guidance I and total five edge-aware filters including BF [27], GF [10], CLMF [18], MLPA [26] and our HGF are taken to perform comparison. For each application, we incorporate above five filters into the same framework to procedure final results. Experiments disclose our filter achieves the best result in terms of accuracy.

Nonlinearity We own the achievement of HGF to the synthesized polynomial guidance which introduces nonlinearity to our HGF. Fig 4 compares the polynomial model and linear model on an artificial dataset, which consists of a curve extracting from a nature image and strong noise added to every fifth datapoints. From the plotted fitting curves of linear model, we can observe that the linear model fails in the dataset. We thus believe that linear model will fail too while smoothing highly corrupted cost slice in the following experiments

Stereo Matching We conduct experiments based on the cost volume filtering framework [12]. This framework comprises cost volume computation, cost aggregation, disparity computation and post processing. Above five filters are employed for cost aggregation. According to the results reported in Fig 5, our method outperforms other aggregation filters and the run time is less than $800ms$.

Optical Flow Xiao *et al*. [30] separate the traditional one-step variational updating model into a two-step filtering-based updating model for optical flow estimation, where BF is proposed to substitute the original anisotropic diffusion process. In this framework, we substitute original BF with GF, CLMF, MLPA and HGF to test the performance of these filters. According to Fig 6, our method can detect tiny structure in the optical flow images within $900ms$.

Image Segmentation We also take the cost volume filtering framework [12] to show that HGF performs well for image segmentation, where the labels encode whether a pixel belongs to the foreground or background and cost computation is same to Hosni *et al*. [12]. Fig 7 proves that our HGF is able to distinguish foreground from background much
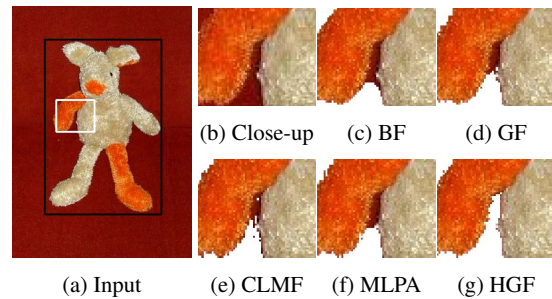

Figure 7: **Segmentation results:** (a) is input image, where black box denotes bounding box and white box presents the close-up region. (b) is the close-up of (a). (c) (d) (e) (f) (g) are the segmentation results of BF, GF, CLMF, MLPA and HGF, respectively.

better and the run time is $120ms$.

7. Conclusion

This paper presented an effective guided image filter for multi-label problem. We own the power to the nonlinearity introduced by the synthesized polynomial guidance. A side effect of the nonlinearity model is that it inevitably increases the running cost as we have to invert a bulk of large matrices. Fortunately, our new designed hardware-efficient matrix inverse algorithm can reduce the run time significantly by the help of our effective matrix inverse technique. We believe our filter will greatly profit building efficient computer vision systems in other multi-label problems.

8. Acknowledgments

References

[1] A. Y.-S. Chia, S. Zhuo, R. K. Gupta, Y.-W. Tai, S.-Y. Cho, P. Tan, and S. Lin. Semantic colorization with internet images. *ACM Transactions on Graphics*, 30(6):156:1–156:8, Dec 2011. 1

[2] D. Cline, K. B. White, and P. K. Egbert. Fast 8-bit median filtering based on separability. In *IEEE International Conference on Image Processing*, volume 5, pages V – 281–V – 284, Sept 2007. 2

[3] F. C. Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH Computer Graphics*, 18(3):207–212, Jan. 1984. 5, 6

[4] L. Dai, M. Yuan, F. Zhang, and X. Zhang. Fully connected guided image filtering. In *IEEE International Conference on Computer Vision*, pages 352–360, Dec 2015. 2

[5] L. Dai, M. Yuan, and X. Zhang. Speeding up the bilateral filter: A joint acceleration way. *IEEE Transactions on Image Processing*, 25(6):2657–2672, June 2016. 2

[6] Y. Ding, J. Xiao, and J. Yu. Importance filtering for image retargeting. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 89–96, June 2011. 1

[7] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics*, 21(3):257–266, July 2002. 2

[8] B. K. Gunturk. Fast bilateral filter with arbitrary range and domain kernels. *IEEE Transactions on Image Processing*, 20(9):2690–2696, Sept 2011. 2

[9] K. He and J. Sun. Fast guided filter. *CoRR*, abs/1505.00996, 2015. 2

[10] K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397–1409, June 2013. 6, 8

[11] R. A. Horn and C. R. Johnson, editors. *Matrix Analysis*. Cambridge University Press, New York, NY, USA, 1986. 2

[12] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(2):504–511, Feb 2013. 1, 8

[13] S. Hougardy and J. Vygen. *Algorithmic Mathematics*. Springer International Publishing, 1 edition, 2016. 1

[14] C. C. Kao, J. H. Lai, and S. Y. Chien. Vlsi architecture design of guided filter for 30 frames/s full-hd video. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(3):513–524, March 2014. 2

[15] M. Kass and J. Solomon. Smoothed local histogram filters. *ACM Transactions on Graphics*, 29(4):100:1–100:10, July 2010. 2

[16] V. Katkovnik, A. Foi, K. Egiazarian, and J. Astola. From local kernel to nonlocal multiple-model image denoising. *International Journal of Computer Vision*, 86(1):1–32, Jan. 2010. 4

[17] Z. Li, J. Zheng, Z. Zhu, W. Yao, and S. Wu. Weighted guided image filtering. *IEEE Transactions on Image Processing*, 24(1):120–129, Jan 2015. 2

[18] J. Lu, K. Shi, D. Min, L. Lin, and M. N. Do. Cross-based local multipoint filtering. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 430–437, June 2012. 2, 8

[19] Z. Ma, K. He, Y. Wei, J. Sun, and E. Wu. Constant time weighted median filtering for stereo matching and beyond. In *IEEE International Conference on Computer Vision*, pages 49–56, Dec 2013. 2

[20] M. McCool, J. Reinders, and A. Robison. *Structured Parallel Programming: Patterns for Efficient Computation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012. 5

[21] S. Paris and F. Durand. A fast approximation of the bilateral filter using a signal processing approach. *International Journal of Computer Vision*, 81(1):24–52, Jan. 2009. 2

[22] S. Perreault and P. Hebert. Median filtering in constant time. *IEEE Transactions on Image Processing*, 16(9):2389–2394, Sept 2007. 2

[23] F. Porikli. Constant time o(1) bilateral filtering. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008. 2

[24] K. Rupp, F. Rudolf, and J. Weinbub. ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs. In *International Workshop on GPUs and Scientific Applications*, pages 51–56, 2010. 5

[25] J. Sherman and W. J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 03 1950. 3

[26] X. Tan, C. Sun, and T. D. Pham. Multipoint filtering with local polynomial approximation and range guidance. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2941–2948, June 2014. 2, 8

[27] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *IEEE International Conference on Computer Vision*, pages 839–, Washington, DC, USA, 1998. IEEE Computer Society. 2, 8

[28] V. Vineet, J. Warrell, and P. H. Torr. Filter-based mean-field inference for random fields with higher-order terms and product label-spaces. *International Journal of Computer Vision*, 110(3):290–307, Dec. 2014. 1

[29] B. Weiss. Fast median and bilateral filtering. *ACM Transactions on Graphics*, 25(3):519–526, July 2006. 2

[30] J. Xiao, H. Cheng, H. S. Sawhney, C. Rao, and M. A. Isnardi. Bilateral filtering-based optical flow estimation with occlusion detection. In *European Conference on Computer Vision*, volume 3951, pages 211–224. Springer, 2006. 1, 8

[31] P. Yalamanchili, U. Arshad, Z. Mohammed, P. Garigipati, P. Entschev, B. Kloppenborg, J. Malcolm, and J. Melonakos. ArrayFire - A high performance software library for parallel computing with an easy-to-use API, 2015. 5, 6

[32] Q. Yang, K. H. Tan, and N. Ahuja. Real-time o(1) bilateral filtering. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 557–564, June 2009. 2

[33] J. Zhang, L. Li, Y. Zhang, G. Yang, X. Cao, and J. Sun. Video dehazing with spatial and temporal coherence. *The Visual Computer*, 27(6-8):749–757, 2011. 1

[34] Q. Zhang, L. Xu, and J. Jia. 100+ times faster weighted median filter (wmf). In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2830–2837, June 2014. 2