國 立 交 通 大 學

電子工程學系　電子研究所碩士班

碩 士 論 文

即時的積分直方圖基準之聯合雙邊濾波演算法分析與設計

Analysis and Design of Real-time Integral Histogram Based Joint Bilateral Filtering

研究生：許博雄

指導教授：張添烜 博士

中華民國　九十九 年　八 月

即時的積分直方圖基準之聯合雙邊濾波演算法分析與設計

Analysis and Design of Real-time Integral Histogram Based Joint Bilateral Filtering

研 究 生：許博雄　　Student: Po-Hsiung Hsu

指導教授：張添烜 博士　　Advisor: Dr. Tian-Sheuan Chang

國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of Master of science
in
Electronics Engineering
August 2010
Hsinchu, Taiwan, Republic of China

中華民國 九十九 年 八 月

即時的積分直方圖基準之聯合雙邊濾波演算法分析與設計

研究生: 許博雄　　　　　　　　　　　　指導教授: 張添烜 博士

國立交通大學

電子工程學系電子研究所碩士班

摘　要

雙邊濾波演算法和聯合雙邊濾波演算法已經被廣泛運用在許多影像處理的領域中，例如去除雜訊、色調處理、甚至是立體的相關應用和 MPEG 標準。它雖然可以用快速演算法中的積分直方圖方法加速，但針對需要即時處理的應用，仍然遭受高運算複雜度，高記憶體使用量的問題。要解決這些問題，VLSI 實現是個必要的方法。本篇研究針對積分直方圖基準之(聯合)雙邊濾波演算法提出一個有效率的硬體架構，其中包含三個自提的記憶體減量方法和可大量平行運算的單元。

這些自提的記憶體減量方法包含動態更新方法，條狀切割方法，和積分起點位移方法。其中動態更新方法是在運算期間，利用演算法循序逐列掃描計算的特性，移除不再使用的資料。而條狀切割方法則進一步將每一張畫面切割成許多縱向的條狀區域並作為逐列掃描計算的單位；每個條狀區域的寬度比畫面寬度短得多，因此逐列掃描計算只需通過較短的列長，使得資料暫存量大減，不再需要整個畫面寬的記憶體空間。最後，積分起點位移方法利用循序　　動態積分起點的概念，協助原始直方圖演算法的積分過程減少對儲存資料的依賴，使得記憶體使用量得以由整張畫面的尺度，減少至列的尺度。整體來說，這三個方法很容易

結合起來，可以將記憶體使用量減少至原演算法的 0.003%。

另一方面，自提的硬體架構利用延遲暫存資料共用方法和使用查表選擇器，分別解決了積分直方圖運算上高頻寬需求和大量查表的問題;並且利用記憶體的切割來提升內部頻寬的容量。除此之外，它也使用數值(在影像中則為亮度)空間平行方法來有效率地執行大量積分直方圖單元運算，而達到高產出。另外，這個硬體架構的運算模組佈局與參數的選擇無關，因此對於不同參數需求的應用，將不需再重新設計。

最後的硬體實現，在聯華電子 90 奈米製程下，使用 200 MHz 的工作時脈，每秒可以執行 60 張 HD1080p (1920x1080)影像。晶片總共需要 355 K 個邏輯閘和 23 K 個晶片記憶體。

Analysis and Design of Real-time Integral Histogram Based Joint Bilateral Filtering

Student: Po-Hsiung Hsu Advisor: Tian-Sheuan Chang

Department of Electronics Engineering & Institute of Electronics

National Chiao Tung University

Abstract

Bilateral filtering and joint bilateral filtering have been widely used in many image processing fields, such as de-noising, tone-management, and even the 3-D applications and MPEG standard. They can be accelerated by the associated fast algorithm, integral histogram, but still suffer from highly computational complexity and massive memory, especially for real-time applications. To conquer them, VLSI implementation becomes a necessary solution. In the thesis, we design an efficient hardware architecture, which consists of three proposed memory reduction methods, and highly parallel computational components for integral histogram based (joint) bilateral filtering.

The proposed memory reduction methods include runtime updating method (RUM), stripe-based method (SBM), and sliding origin method (SOM). The RUM in runtime takes advantage of progressive raster-scan process of computation to discard unnecessary data. The SBM further divides each frame into vertical stripes and processes them one by one. These stripes are much narrower than a frame; therefore, the raster scan process can traverse along shorter rows and the original frame-wide

memory cost can be significantly reduced. Finally, the SOM uses the concept of progressive sliding integral origin to help the original histogram integration process lessen the dependency on storage data; therefore, the memory requirement can be reduced from frame-scale-magnitude to line-scale-magnitude. On the whole, the three methods can be easily combined to reduce the memory cost to 0.003% of the original requirement.

On the other hand, the proposed hardware architecture solves the integral histogram computational high bandwidth and large table problem by using delay-buffer data-reuse method and table selector, respectively. And use memory banks to enlarge the capacity of internal memory bandwidth. Besides, it uses range (intensity, for image)-space-parallelism methods to process large amount of histogram bins simultaneously to achieve high throughput. What's more, the function block layout of the hardware architecture is invariant to parameter selection; therefore, it doesn't have to be redesigned for applications of different parameter demands.

The final design implemented by UMC 90nm CMOS technology can achieve 60 frames per second for HD1080p (1920x1080) resolution image under 200MHz clock rate. The chip consumes 355 K gate counts and 23 K Bytes on-chip memory.

誌　謝

首先，要感謝我的指導教授—張添烜博士，在研究和修課上讓我能夠自由的發揮；不管是研究本身或是工作應徵，在我遇到瓶頸都給予適切的建議與協助，讓我得以用正確且有效率的方式來解決問題。老師良師益友的形象，深植我心。此外，實驗支援研究的軟硬體充足，電腦設備操作環境寬敞，光線充足；讓我有個舒適的研究環境。

感謝我的口試委員們，清華大學資訊工程系邱瀞德教授與交通大學電子工程系王聖智教授。感謝你們百忙之中抽空前來指導，因為教授們的寶貴意見，讓我的論文更加的充實而完備。

接著我要感謝實驗室的好夥伴們，特別是引我入門的曾宇晟學長，從零開始帶我進入立體相關應用的領域；不斷提供並分析可行的方向；並指導我論文寫作的技巧。也感謝林佑昆，張力中和李國龍等學長經驗的傳承，讓我受用無窮。謝謝蔡宗憲學長，上班之後百忙中還能撥空協助我完成計畫。謝謝陳之悠學長曾力薦我加入該實驗室。也謝謝王國振，許博淵，沈孟維，蔡政君等學長和黃筱珊學姊曾經提供我研究上寶貴的經驗和協助。接著謝謝在國科會計畫中鼎力相助的洪瑩蓉以及吳英佑學弟，沒有你們的即時幫忙，我恐怕無法順利 demo。謝謝 IC 競賽戰友陳奕均，一起硬拼 12 小時拿下佳作，真的很難忘。也謝謝買飯團固定班底廖元歆和陳宥辰，讓我天天不用孤單的走向二餐。此外，實驗室的邱亮齊，溫孟勳，曹克嘉學弟也不能忘，是我平日一起經歷歡笑的好夥伴。

特別感謝生了病卻還不斷為我論文掛心提醒的媽媽，並祈求她早日康復。並感謝永遠支持我的家人們：我的爸爸、弟弟，還有一直陪伴在旁替我加油的女友，你們的支持與鼓勵，是我順利完成學業的最大動力。

在此，謹將本論文獻給所有愛我以及我愛的人。

Contents

List of Figures

List of Tables

1. Introduction

1.1. Background

Bilateral filtering [1] is a special image smoother which can remove small-scale texture or noise while preserving large-scale structure or edges. The judgment to be noise or edge could be determined by an easy-tuning parameter. The ability of easily separating small-scale and large-scale contents makes bilateral filtering be more widely used than a typical smoother, such as joint bilateral filtering. Joint bilateral filtering, which is a variety of bilateral filtering combined with a guidance concept, is associated with more widely applications such as up-sampling [2], adaptive support weight [3], and even 3-D related processing [4] and MPEG standard [5].

The challenge of real time implementation for bilateral filtering is the high computational complexity of its window processing. Many algorithms have been proposed to reduce the complexity. In the thesis, we category them into two approaches: support-pixel-first approach and target-pixel-first approach. In previous work, the support-pixel-first approach was implemented through GPU programming, and achieved real-time speed. However, GPU hardware is general purpose platform and not a dedicated low-cost implementation for embedded applications. Therefore, VLSI hardware implementation is a better solution to minimize hardware cost and achieve real-time speed.

For VLSI hardware implementation, the support-pixel-first approach requires a frame-scale-magnitude memory, but it can not be reduced because of its iterative process by frames. On the other hand, the target-pixel-first approach also suffers from frame-scale-magnitude memory requirement. Nevertheless, the cost is likely to be reduced since its progressive process with pixel-by-pixel order.

1.2. Motivation and contribution

Motivated by the high memory cost in joint bilateral filtering, this thesis proposed efficient hardware architecture based on integral histogram algorithm of the target-pixel-first approach. The goal is to build a dedicated hardware for low memory cost real-time joint bilateral filtering.

The major contributions of this thesis are three.

1. Based on integral histogram based joint bilateral filtering, we proposed three memory reduction methods to significantly reduce the memory cost. This makes integral histogram based joint bilateral filtering suitable for simpler on-chip memory based implementation in ASIC.
2. We propose an efficient hardware architecture which can efficiently process parallel operations and achieve high throughput.
3. We implemented the low memory cost real-time hardware of the proposed architecture with the three proposed memory reduction methods.

1.3. Thesis Organization

Chapter 2 briefly introduces bilateral filtering and its applications. Chapter 3 introduces the acceleration algorithms for bilateral filtering. Chapter 4 discusses the design challenges of integral histogram based joint bilateral filtering. To solve these challenges, Chapter 5 proposes three proposed memory reduction methods, and Chapter 6 proposes an efficient hardware architecture. Finally, Chapter 7 gives the conclusion of this thesis.

2. Introduction of Bilateral Filtering

2.1. Overview

Bilateral filtering (BF) is primary adopted in image processing for de-noising. With BF's de-noising (or smoothing), the object edges and borders of image are preserved. As a result, BF becomes popular because it can provide a no-blur clear result. Moreover, the edge-preserving capability enables us to adapt BF for many advanced applications such as texture editing, tone management, demosaicing, stylization, and optical flow estimation [6].

2.2. Bilateral Filtering

BF, originated by Tomasi and Manduchi [1], is defined as,

$$BF(I)_c = \frac{\sum_{q \in S} f(|c-q|) g(|I_c - I_q|) I_q}{\sum_{q \in S} f(|c-q|) g(|I_c - I_q|)}, \tag{2.1}$$

where c is the target pixel, and q is the support pixel surrounding to c. For ease of computing by typical row-column rectangular image file format, the support pixel q is usually taken from a square window S centered at c. Both the intensities of c and q, I_c and I_q, is in the range domain R from 0 to 255 for gray-level. In this equation, I_q are accumulated and normalized with two weighting kernels, the space kernel f and the range kernel g. Both f and g are usually chosen as low-pass functions with the arguments of space distance $|c-q|$ and intensity difference $|I_c-I_q|$, respectively.

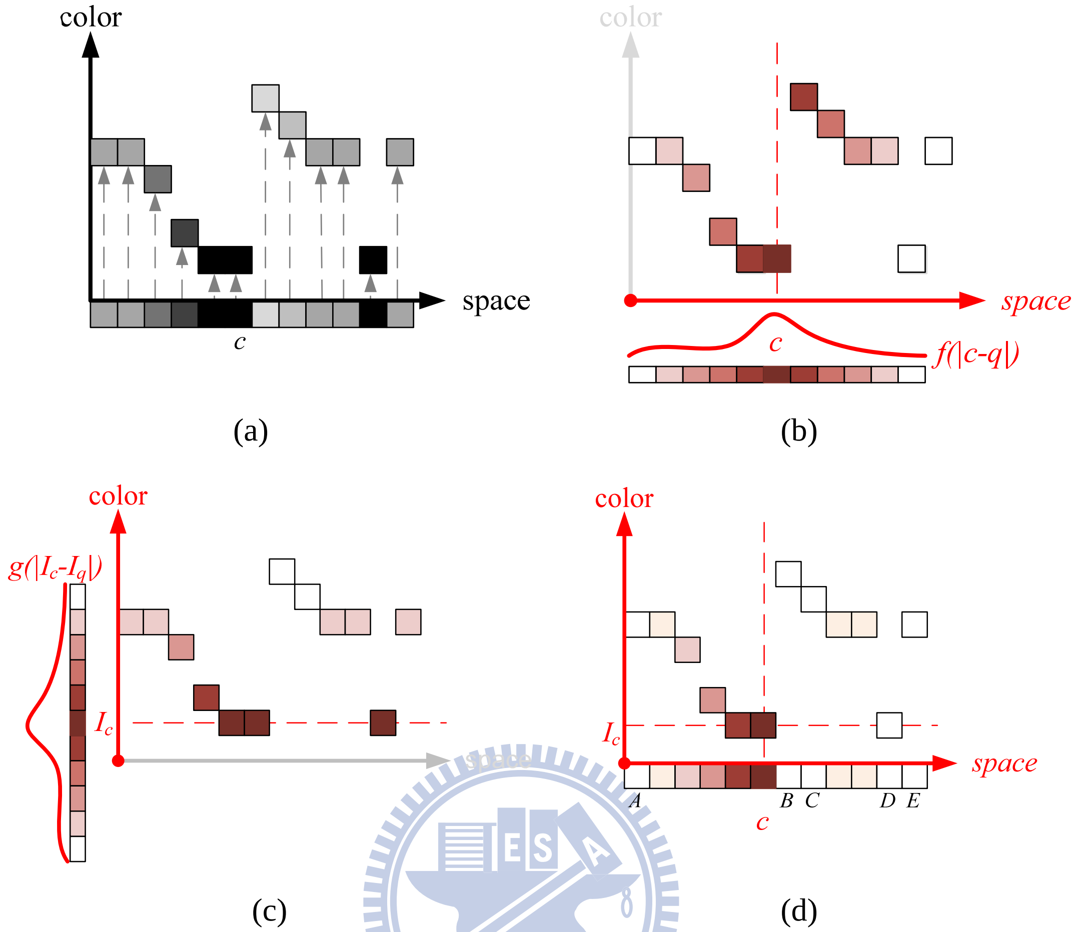
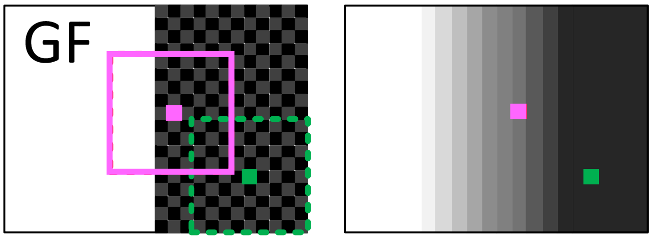
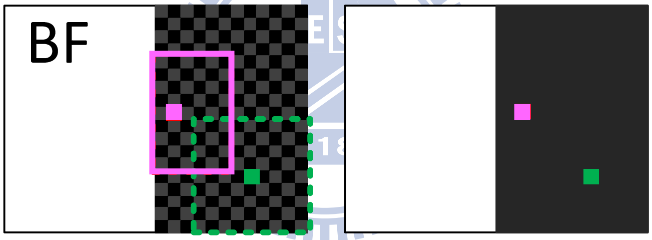


Fig. 2.1. Illustration of space kernel *f* and range kernel *g* of BF
(a) 1-D space-color domain, (b) weighting by *f*, (c) weighting by *g*,(d) combined weighting by *f* and *g*

Fig. 2.1 shows how kernel function *f* and *g* influent the weighting value for support pixel *q*. In Fig. 2.1 (a), for ease of show, we take one-dimension (1-D) image as spatial domain on *x*-axis and project intensity domain *R* onto the *y*-axis. Fig. 2.1 (b) shows that Gaussian function with argument space distance $|c\text{-}q|$ is a low-pass filter; it gives higher weight on near-*c* support pixels and lower weight on farther ones. It is intuitive that the farther *q* is away from target pixel *c*, the smaller its impact should place on the final result. On the other hand, similar weighting mechanism is placed on the intensity difference of *c* and *q*. Fig. 2.1 (c) shows that Gaussian function with argument $|I_c\text{-}I_q|$ gives support pixel higher weight if its intensity is similar to I_c. This is also intuitive to realistic situation: two nearby pixels with similar intensity are likely belongs to the same object. To multiplying the two function's effect as Fig. 2.1 (d) shows: the point *A, B, C, D,* and *E* are regarded as outliers with zero weighting. Especially notes that point *B* is an outlier regarded by kernel *g* though it is adjacent to *c*. Similarly, point *C* is an outlier regarded by kernel *f* though its intensity is I_c. That is to say, either *q* is far away from *c* or I_q is dissimilar to I_c, the impact of *q* will be negligible.



(a)



(b)

Fig. 2.2. Smoothing Results
(a) Gaussian filter, (b) Bilateral filter

Before Tomasi [1] et al. proposed BF, the most typical smoother was Gaussian filtering (GF) or other low pass filtering. The typical smoothers suffered from blur-effect because they only considered space kernel. Many algorithms have proposed to eliminate this effect. Tomasi added a range kernel into GF to be BF; this is a simple but effective method. Fig. 2.2 compares BF with GF to show that the range kernel is the key component for edge-preserving. In Fig. 2.2 (a), GF is used to remove the chessboard-like noise in the dark area of the left image. The right image is its result. It is obvious that GF produces smooth result on the pixel far from the edge (the area around the green pixel), whereas it produces blur effect near the edge. This because GF is blind to entirely different colors across the edge; it still mixes all colors within its window though the window steps across the edge. Therefore, in the output result, it appears a blur area at the both sides of the edge. Fig. 2.2 (b) shows that BF doesn't produce blur effect because its window doesn't step on the both sides of the edge to mix entirely different colors. As shown by the red window, the window of BF is trimmed by the edge because the bright-side pixels, which have entirely different color from the center color, are regarded as outliers by its range kernel.

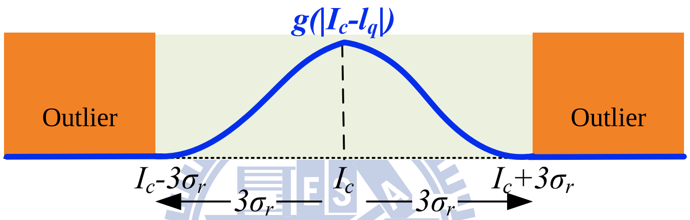


Fig. 2.3. Gaussian kernel's bandwidth

There is a parameter σ_r determining the degree of edge-preserving. It is defined by the Gaussian function equation,

$$g(|I_c - I_q|) = Ae^{-\frac{|I_c - I_q|^2}{2\sigma_r^2}}, \tag{2.2}$$

where A is a constant. As shown in Fig. 2.3, the Gaussian kernel's bandwidth extends by about 3 times of σ_r. Outside the bandwidth, the value of g drops to below 0.01 which is negligible compare with the center weight. Any support pixel q with color outside the bandwidth will be regarded as the outlier. As a result, any edge with lager color difference will be reserved (As last paragraph illustrates, this kind of edge trims kernel.). On the other hand, any edge with smaller color difference is blurred or smoothed as the noise.

Fig. 2.4. Smoothing results of BF with different range parameter σ_r
(a) noisy image, (b) σ_r=25, (c) σ_r=100, (d) σ_r= very large (GF).

Fig. 2.4 shows smoothing results of BF with different parameter σ_r choices. For, the given noisy "Lina" shown by Fig. 2.4 (a), the value 25 is the best choice for σ_r to separate noise and edges. If σ_r becomes larger as Fig. 2.4 (c), more edges are also regarded as noise so that only the image structure is reserved. If σ_r is further set to a very large value, BF will be simplified to GF because the color kernel becomes a constant function. As shown in Fig. 2.4 (d), the blur effect is obvious.

2.3. Application

We will recall applications of BF in this sub-chapter. They are mainly classified into de-noising, texture and illumination separation, and joint BF.

2.3.1. De-noising

De-noising or smoothing is the primary goal of BF. Other than being applied for 2-D image smoothing, it is also adapted for video processing and 3-D mesh smoothing. And many de-noise-related applications, such as flash and no-flash Image correction, are constantly proposed.

For video application, Bennett et al. [7] introduced BF into temporal smoothing. He assumes that the pixel variations in the temporal related same scene point over frames are affected by zero-mean noise. GF is used to reduce the noise level but it produces artifacts on moving object. Using BF instead can avoid these artifacts. For 3-D mesh smoothing, Jones et al. [8] and Fleishman et al. [9] simultaneously presented two similar approaches to adapt BF in the higher-dimension space. In the higher-dimension space, window computations for both kernels become more complex. Geometry properties such as mesh normal, projection, etc., are considered carefully.

On the other hand, in de-noise-related applications, Eisemann and Durand [10] used BF for flash and no-flash image correction. For a no-flash photo of a dark scene, although its illumination is correct, it has low signal-to-noise-ratio (SNR) that leads to inaccurate edge detection. However, a flash photo of the same scene has high SNR and higher discrimination of colors but it suffers from incorrect hard direct illumination. As shown in Fig. 2.5 [10], BF is used to smooth both photos for de-noising and information extraction. BF helps departing their small-scale details and large-scale structure (This will be further discussed in 2.3.2). Finally, information from flash and no-flash photos is combined to form the final result without noise and with correct illumination and structure. Petschnigg et al. [11] also has proposed a similar correction algorithm based on this approach.

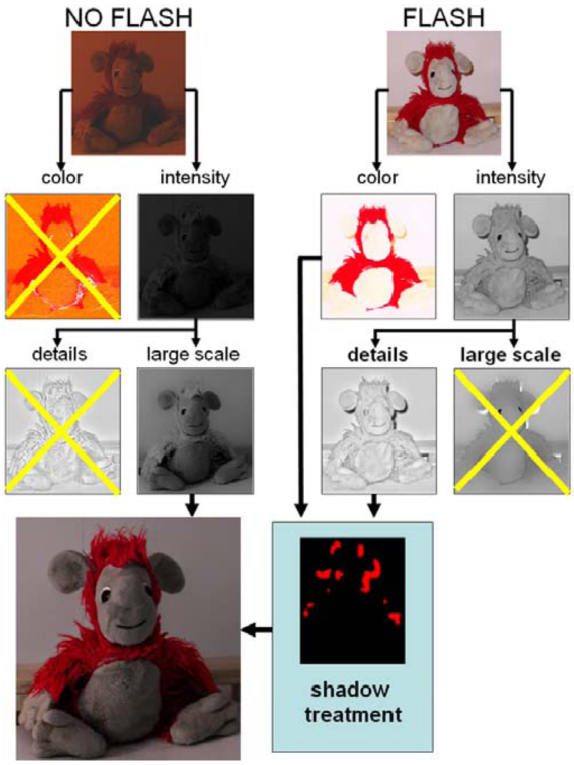


Fig. 2.5. Flow of flash and no-flash image correction [10]

2.3.2. Texture and illumination separation

Oh et al. [12] used BF as a separation algorithm to extract image texture and illumination component. They are motivated by the fact that in typical image, the illumination variation typically occurs at a large scale structure than small scale texture patterns; therefore, they proposed an approach using BF with suitable range kernel g to remove small-scale texture and preserve the large-scale illumination component. Simultaneously, the removed small-scale texture can also be extracted by subtracting the large-scale component from origin image.

With the concept of above separation algorithm, Durand and Dorsey [13] isolated texture component from naïve intensity compression in tone mapping of high-dynamic range (HDR) image for low dynamic range display. This approach prevents the details in small scale texture being removed during compression. Other algorithms addressed in [14] and [15] also use the similar aspect.

2.3.3. Joint Bilateral Filtering

The BF used in the flash and no-flash image correction by Eisemann and Dorsey [10] is defined specially with the following equation,

$$JBF(J)_c = \frac{\sum_{q \in S} f(\|c-q\|) g(\|I_c - I_q\|) J_q}{\sum_{q \in S} f(\|c-q\|) g(\|I_c - I_q\|)}, \tag{2.3}$$

where I is a guidance image, and J is another source image. Through the range kernel g, the guidance image I could identify and suppress outliers for de-noising the source image J. To emphasize that it joints guidance image influence into target source image, this specially defined BF is renamed as joint bilateral filtering (JBF).With this characteristic, JBF has been adopted in another flash and no-flash algorithm [15], image de-nosing [16] and disparity-map fusion [17],[18].

Further extending the applications of JBF, Kopf et al. [2] proposed the joint bilateral up-sampling that employed a high-resolution I to enlarge a low-resolution J for various image processing, such as tone mapping, colorization, disparity maps [19]-[21], demosaicing [22], texture synthesis [23]. A variety of JBF is the adaptive support weight (ADSW), a matching cost aggregation approach, proposed by Yoon and Kweon [3] for disparity estimation in 3D image processing. The disparity estimation is based on matching corresponding pixels in different view frames. To increase matching correctness, disparity estimation uses filter-like convolution to aggregate support matching costs for target pixel. The ADSW employs the space and range kernels into aggregation to deliver better disparity maps than that produced by the traditional box filter. The concept of ADSW is further advanced in the disparity estimation algorithms of [24]-[28], and is also adopted by the developing MPEG standard, 3D Video Coding [5].

2.4. Summary

BF is an edge-preserving filter. Its parameter σ_r in range kernel can determine the discontinuity in images to be either large-scale structure or small-scale texture (noise). The characteristic makes its application more than the primary goal of de-noising such as illumination and texture separation and JBF. Furthermore, with the guidance concept of JBF, BF applicable algorithms can be extended to various fields, such as disparity estimation for stereo process, up-sampling, and even the MPEG standard.

3. Related Work

Within BF applications, stereo processing is increasingly important in recent years. Many 3D-related entertainments, facilities, and industrials are pouring or on the horizon. Under this circumstance, BF and JBF must be ready for its potential real-time requirement of image and video processing. However, the big challenge for BF is its computational complexity in window computation. By brute-force implementation, BF takes extremely long running time on huge operations.

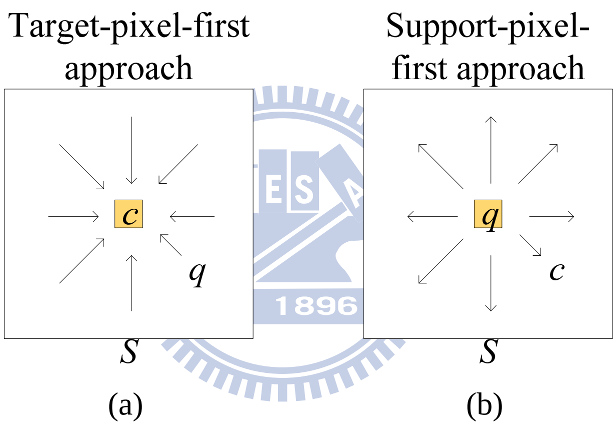


Fig. 3.1. Classification of acceleration approaches

Various acceleration approaches for BF have been proposed, and can be classified into two categories: target-pixel-first approach and support-pixel-first approach, according to their computational characteristics, as illustrated in Fig. 3.1. The target-pixel-first approach is an aggregation process that focuses on a target pixel c and accumulates its support pixels q. On the other hand, the support-pixel-first approach is a diffusion process that regards a support pixel q as a center to diffuse for its target pixels c. With the classification, the milestone algorithms are listed in TABLE. 3-1.

The computational complexity and memory cost of the milestone algorithms are also compared in TABLE. 3-1. Note that the former is shown by amount per pixel and the latter is shown by amount per frame. With this table, it is easy to approximate real amount of computations and memory cost of these algorithms for any size of target image. Take the brute-force implementation for example, referring to (2.1), for each pixel result, BF aggregates support pixels in the window S; therefore, the computational complexity is $O(|S|^2)$ which is associated to window size. This means if it processes an HD1080p image with a 31-pixel window width, the amount of required computations should be at the order of 2 billion (31^2x1920x1080). By software, the computationally expensive implementation takes minutes for a frame.

In the rest of chapter, we introduce the acceleration algorithms. In 3.1 and 3.2, support-pixel-first algorithms and target-pixel-first algorithms are introduced, respectively. Finally, in 3.3, we explain how we select algorithms from them for our proposed architecture design and implementation.

TABLE. 3-1 Comparison of computational complexity and memory cost in related work

Approach			Computational Complexity (per pixel)		Memory Cost (per frame)
	Brute-Force		All	$O(\|S\|^2)$	0
Support Pixel First	Basic		LUT Construction	$O(\|R\|)$	$4MN$
			2-D Conv. by FFT	$O(\|S\|log\|S\|)$	
	Durand and Dorsey [13]	Piecewise-linear Subsampling	LUT Construction	$O(\|R\|/s_r)$	$4MN/s_s^2$
			2-D Conv. by FFT	$O(\|S\|/s_s^2 log(\|S\|/s_s^2))$	
	Yang et al. [29]	Piecewise-linear	LUT Construction	$O(\|R\|/s_r)$	$4MN$
			2-D Conv. by Approx. Gaussian	$O(1)$	
	Paris and Durand [30]	Bilateral Grid	LUT Construction	$O(\|R\|/s_r)$	$MN\|R\|/(s_r s_s^2)$
			3-D Conv. by FFT	$O(\|S\|\|R\|/(s_r s_s^2)log(\|S\|\|R\|/(s_r s_s^2)))$	
Target Pixel First	Pham and Vliet [31]	Separable	1-D Aggre. for Col.	$O(\|S\|)$	0
			1-D Aggre. for Row	$O(\|S\|)$	
	Basic	Histogram	Histogram Calculation	$O(\|R\|\|S\|^2)$	0
			1-D Conv.	$O(\|R\|)$	
	Huang [32]	Extended Histogram	Histogram Calculation	$O(\|R\|\|S\|)$	$\|S\|\|R\|$
			1-D Conv.	$O(\|R\|)$	
	Weiss [33]	Distributed Histogram	Histogram Calculation	$O(\|R\|log\|S\|)$	$\|S\|\|E\|\|R\|$
			1-D Conv.	$O(\|R\|)$	
	Porikli [34]	Integral Histogram	Histogram Calculation	$O(\|R\|/s_r)$	$MN\|R\|/s_r$
			1-D Conv.	$O(\|R\|/s_r)$	

M: frame height, N: frame width, $|S|$: window width, $|R|$: intensity range
s_s: quantization factor for S, s_r: quantization factor for R, E: extension pixel count

3.1. Support-pixel-first Approach

Within support-pixel-first milestone algorithms, Durand and Dorsey's piece-wise linear [13] is the first acceleration algorithm; Young's algorithm [29] and Pairs' algorithm [30] are partially related to it. Yong's algorithm is boost of its constant time speed (independent of window width) and Paris' algorithm proposes a brand-new spatial-intensity space.

3.1.1. Piece-wise linear algorithm and Yong's algorithm

The range kernel makes BF nonlinear to spatial space; therefore, any spatial filter acceleration approach such as Fast Fourier Transform (FFT) doesn't help to speed up BF. Instead of directly using the nonlinear equation of (2.1), Durand and Dorsey [13] approximate BF with a serial of frame-scale look-up tables (LUTs) defined as follows

$$
\begin{aligned}
LUT(j)_c &= \frac{\sum_{q\in S} f(|c-q|)g(|j-I_q|)I_q}{\sum_{q\in S} f(|c-q|)g(|j-I_q|)} \\
&= \frac{\sum_{q\in S} f(|c-q|)H_q^j}{\sum_{q\in S} f(|c-q|)G_q^j}
\end{aligned}, \qquad (3.1)
$$

each of which associates to an intensity j that replaces the I_c of (2.1). The FFT can accelerate the computation of (3.1) since both its numerator and its denominator become linear Gaussian convolution. The overall process includes two steps; at first, for every full-scale intensity j, its LUT is computed; that is, for a typical 8-bit image, 256 LUTs should be computed and stored. Second, for every pixel, its result is picked up from its intensity corresponded LUT by the following equation,

$$
BF(I)_c = LUT(j)_c, \qquad if\ I_c = j \qquad (3.2)
$$

Besides, instead of using full-scale intensities, Durand and Dorsey [13] propose piece-wise linear algorithm to reduce the number of LUT. With a quantization factor s_r, it only computes the LUT corresponds to intensity equals s_r or its multiples. And the result-picking function is rewritten as

$$BF(I)_c = \begin{cases} LUT(j)_c, & if\ I_c = j \\ \frac{j - I_c}{s_r} LUT(j)_c + \frac{I_c - (j - s_r)}{s_r} LUT(j - s_r)_c, & if\ j - s_r < I_c < j \end{cases} \quad . \tag{3.3}$$

With (3.3), for the pixel without intensity corresponded LUT, its result is computed by bilinear interpolation of two LUTs of the most similar intensities.

Durand and Dorsey [13] further introduced a fast piecewise-linear algorithm with spatial space sub-sampling (quantization). The major computational complexity is $O(|(S|/s_s^2)log(|S|/s_s^2))$ per pixel in 2-D FFT, where s_s is a spatial quantization factor. The memory requirement is huge with cost $4MN/s_s^2$ since at least four frame-scale data, H^j, G^j, partial result of LUT(j) and previous result $LUT(j\text{-}s_r)$, are required under the implementation of runtime updating LUT intensity by intensity [13].

Mostly based on piece-wise linear algorithm, Young et al. [29] used Deriche's recursive method [35] to approximate Gaussian convolution of (3.1). They shows that this recursive method is able to run in constant time and the results are visually very similar to the exact. Therefore, the convolution process is reduced to O(1) complexity; and thus the major complexity of BF becomes $O(|R|/s_r)$ of LUT construction.

3.1.2. Bilateral grid

Paris and Durand [30] reformulated gray-level BF with a brand new 3-D space, bilateral grid. By their algorithm, it takes three steps to process BF; they are bilateral gird construction, 3-D Gaussian smoothing, and result extraction.

For bilateral grid construction, given a 2-D image, the first two dimensions of bilateral grid will correspond to the image spatial position *(x,y)* and the third dimension corresponds to the pixel intensity I_c. At the position (x,y,I_c), an non-zero element is constructed. With all elements are constructed, in the second step, BF is computed by a 3-D defined Gaussian smoothing to associate weights w with intensities I and finally store each element with a vector $(\sum wI, \sum I)$. Because in bilateral grid the intensity is defined as an independent dimension, BF is linear for the 3-D Gaussian smoothing. Finally, in the result extraction step, the first two dimensions of bilateral gird correspond back to the position of 2-D image and set intensity there with the value, $\sum wI / \sum I$.

Paris and Durand further reduced the computational effort by down-sampling the three dimensions of bilateral grid with the spatial quantization factor s_s for the first two dimensions (spatial position) and the range quantization factor s_r for the third dimension (intensity). The computational complexity of the algorithm is $O([|S||R|/(s_r s_s^2)][log(|S||R|/(s_r s_s^2))])$ of Gaussian smoothing. The memory cost is $MN|R|/(s_r s_s^2)$ for storing the whole bilateral grid structure.

Following the bilateral grid scheme, Chen [36] further mapped this algorithm to GPU hardware, obtaining real-time processing for several megapixel images. In addition, Adams et al. [37] adopts the Gaussian KD-tree to improve its speed.

3.2. Target-pixel-first Approach

In TABLE. 3-1, the target-pixel-first algorithms can be mainly classified into two kinds of approaches: one is separable approach [31] and the other is histogram-based approach [32]-[34]. The separable approach uses two consequent 1-D BFs to speed up the computation; Histogram-based approach uses range aggregation instead of spatial aggregation by histogram represented BF. Within its acceleration algorithms, integral histogram let the speed of BF implementation be independent of window width $|S|$.

3.2.1. Separable algorithm

Pham et al. [31] proposed this algorithm to approximate 2-D BF by two consequent 1-D BFs which computed by brute-force implementation: pixels within a row (column) are accumulated one by one and finally normalized. At first, by performing 1-D BF to all rows and their results make up a single column; and then, it performs 1-D BF again to the column for the final result. The computational complexity of separable algorithm is reduce to O($|S|$) per pixel because 1-D window with length $|S|$ is used for 1-D BF. Though it is significant faster than the brute-force implementation, the performance degrades linearly with window size. In addition, its axis-aligned 1-D BF makes it not suitable for the target image with complex patterns since its result suffers from the axis-aligned artifact.

3.2.2. Histogram & Huang's algorithm

The histogram-based approach could reduce computation without significant quality degradation. The histogram representation of BF is defined as

$$
\begin{aligned}
BF(I)_c &= \frac{\sum_{q\in S} g(|I_c - I_q|) I_q}{\sum_{q\in S} g(|I_c - I_q|)} \\
&= \frac{\sum_{b\in R}[\sum_{I_q=b} g(|I_c - b|) b]}{\sum_{b\in R}[\sum_{I_q=b} g(|I_c - b|)]} \\
&= \frac{\sum_{b\in R}[g(|I_c - b|) b \sum_{I_q=b} 1]}{\sum_{b\in R}[g(|I_c - b|) \sum_{I_q=b} 1]} = \frac{\sum_{b\in R} g(|I_c - b|) h_c(b) b}{\sum_{b\in R} g(|I_c - b|) h_c(b)}
\end{aligned}
\tag{3.4}
$$

where h_c is the pixel count histogram of the window S centered by c as illustrated in Fig. 3.2. The key point of these approaches is to convert its convolution from the space domain S to the range domain R, as shown in the summation index of (3.4). Thus, its computation includes two parts: histogram calculation and 1-D convolution. In the histogram calculation for h_c, each support pixel q in S is classified by its intensity and accumulated into its corresponding bin b. In other words, $h_c(b)$ refers to the number of support pixels with the intensity b in S. Note that the number of bin N_b is 256 for the exact result of typical 8-bit gray-level. In the 1-D convolution, (3.4) can be calculated with the given h_c. For the basic histogram-based approach, the major computational complexity is $O(|R||S|^2)$ in the histogram calculation.

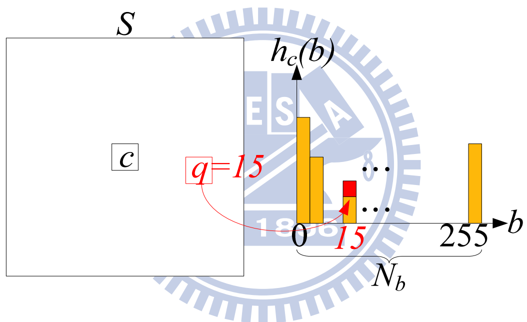


Fig. 3.2. Concept of histogram-based approaches

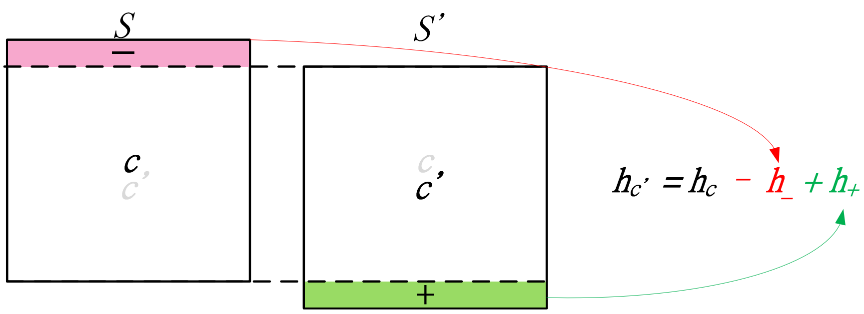


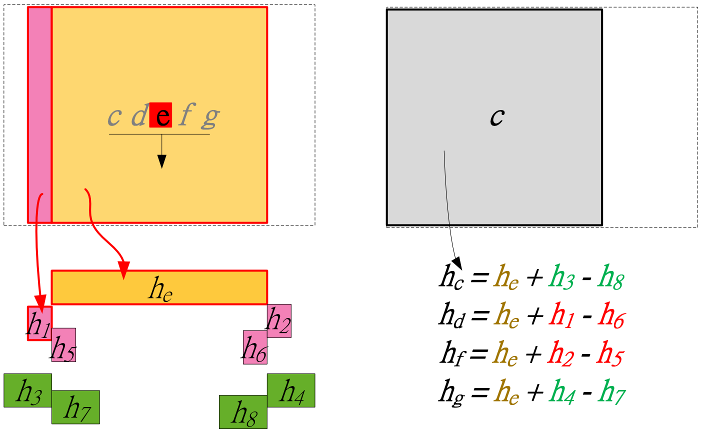
Fig. 3.3. Concept of Huang's algorithm

To speed up the histogram calculation, an early proposed Huang's algorithm [32] can be applied. As shown in Fig. 3.3, windows of two consequently-processed pixels c and c' are almost overlapped each other; therefore, the window histogram $h_{c'}$ can be updated from the processed window h_c by two row histograms. The computational complexity associates to the row histogram is $O(|R||S|)$ which is significantly faster than the basic histogram approach if $|S|$ is large. However, it spends extra memory cost with size $|S||R|$ to store row histograms on overlapped region.

3.2.3. Weiss' Distributed Histogram

Based on Huang's algorithm, Weiss [33] proposed a distributed histogram approach that reassembles the histogram calculation of each row. The approach not only reuses histograms in vertically process direction, but it also reuses data horizontally during processing many column pixels together. Fig. 3.4 (a) illustrates an example of 5-column-parallel process during which Weiss algorithm keeps nine distributed histograms: h_e , which associates to the window of pixel e, and column histograms h_1-h_8. Window histograms associate to targets c, d, f, g are computed from these nine histograms as shown by Fig. 3.4 (b). In horizontal, the approach can be extended for more parallel columns with different set of distributed histograms. On the other hand, in vertical, these distributed histograms update by Huang's algorithm.

Based on distributed histogram approach, Weiss further introduced hierarchical approach. Fig. 3.5 [33] shows an example of hierarchical distributed histogram which has yellow, orange, and red, totally three coarse-to-fine tiers. This hierarchical approach can reduce computational complexity to near $O(|R|log|S|)$. For the memory cost, the approach uses Huang's algorithm so that it also needs memory to store histograms. Furthermore, since histograms are distributed, the memory cost grows larger to $|S||E||R|$, where E associates to how many distributed histograms are used in parallel.



(a) (b)

Fig. 3.4. Concept of Weiss distributed Histogram:
(a) distributed histograms, (b) computations of target histograms

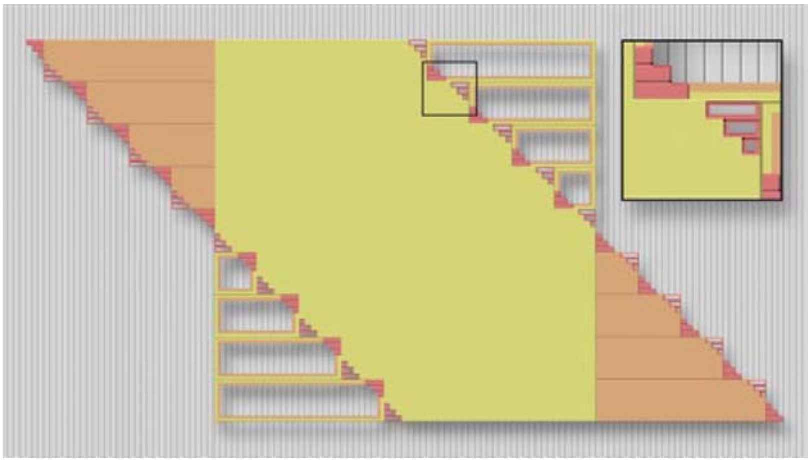
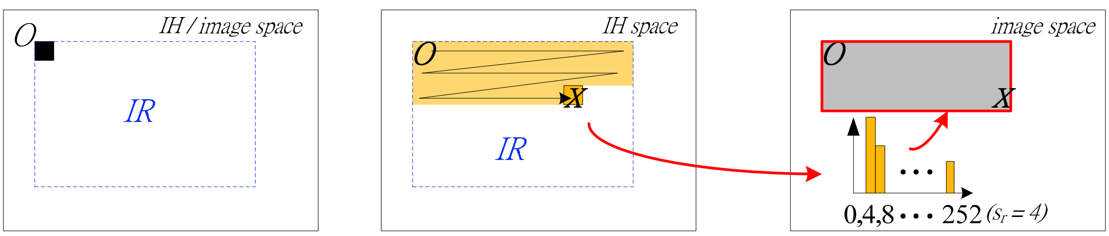


Fig. 3.5. Three-tier hierarchical distributed Histogram [33]

3.2.4. Integral Histogram



(a) (b) (c)

Fig. 3.6. Concept of integral histogram
(a) Integral origin O and integral region (IR), (b) *integration process*, (c) an integral histogram of pixel X of the IH space

Porikli et al. [34] proposed this algorithm to make the computational complexity of histogram calculation independent of window size. The construction of integral histogram (IH) is like a space transformation process from a 2-D image space to a 2-D IH space. Prior to processing the transformation, we have to decide an integral origin O and an integral region (IR) as illustrated in Fig. 3.6 (a). Fig. 3.6 (b) shows that during the transformation with raster scan process from O to the end of IR, each pixel of 2-D IH space is given an IH. Fig. 3.6 (c) illustrates that the given IH at any pixel X is actually a quantized histogram (with quantized factor s_r) for a 2-D image space region stretches from O to X. Porikli et al. showed that quantized histogram doesn't suffer from severe quality degrading for BF result; therefore, the number of histogram bins can be less than the number of intensity levels. In overall, the *integration process*' computational complexity is O($|R|/s_r$) of pure histogram operations. And other details will be further discussed in Chapter 4.1.

In IH space, arbitrary window histogram (as long as the whole window is within the IR) is computed from linearly combination of its four corner integral histograms ;therefore, the computational complexity is reduced to O($|R|/s_r$) that is independent of window width $|S|$. The integral histogram approach can be faster than the brute-force approach when $|R|/s_r$ is smaller than $|S|^2$. That implies this approach is suitable to be applied when BF has large window size. In term of computational complexity, this algorithm is the state-of-art of the target-pixel first approach. But its memory cost is large with amount $MN|R|/s_r$ because of the frame-scale-magnitude process, where MN is the area of the image. Other details of the *extraction process* are also discussed in Chapter 4.1.

3.3. Summary

In comparison, the support-pixel-first algorithms are iterative processed by frames and the target-pixel-first algorithms are progressive processed pixel-by-pixel in raster scan. For computational complexity, Young's algorithm of the former and Porikli's algorithm of the latter achieve constant time of O($|R|/s_r$). They both suffer from high memory cost because of frame-scale-magnitude LUTs and histogram storage, respectively. In terms of implementation, the support-pixel-first approach is more suitable for multi-color-channel computing since they are defined by a multi-dimensional space. For the realization of gray level process, the support-pixel-first has achieved real time in GPU hardware and target-pixel-first approach is implemented by software program. However, we still choose target-pixel-first approach because its memory cost is likely to be reduced and other details are discussed in the next chapter.

4. Analysis of Integral histogram based JBF

The support-pixel-first approach can achieve real time process with GPU hardware. As mentioned before, GPU implementation is a general-purpose hardware. Although it may be implemented in embedded or source-restricted system, it still cost expensive. For a specified low cost implementation, VLSI implementation may be a more proper candidate. In addition, both support-pixel-first and target-pixel-first approaches suffer from high memory cost; however, the cost of the latter is likely to be reduced by taking advantage of its progressive process, whereas the cost of the former must be frame-scale-magnitude because of its iterative process by frames. Therefore, in the thesis, we focus on VLSI implementation of target-pixel-first approach for BF or JBF. Within its algorithms, integral histogram is the state-of-art.

To combine integral histogram and JBF, Ju and Kang [38] modified (3.4) to

$$\begin{aligned} JBF(J)_c &= \frac{\sum_{q\in S} g(|I_c - I_q|)J_q}{\sum_{q\in S} g(|I_c - I_q|)} \\ &= \frac{\sum_{b\in R}[\sum_{I_q=b} g(|I_c - b|)J_q]}{\sum_{b\in R}[\sum_{I_q=b} g(|I_c - b|)]} \\ &= \frac{\sum_{b\in R}[g(|I_c - b|)\sum_{I_q=b} J_q]}{\sum_{b\in R}[g(|I_c - b|)\sum_{I_q=b} 1]} = \frac{\sum_{b\in R} g(|I_c - b|)h'_c(b)}{\sum_{b\in R} g(|I_c - b|)h_c(b)} \end{aligned} \tag{4.1}$$

Different from (3.4), the histogram in the numerator is the pixel intensity histogram h'_c that accumulates the pixel intensity for each bin, instead of the pixel count in h_c. In this chapter, we introduce the integral histogram approach in details, and then analyze the design challenges of integral-histogram-based JBF, which can also be applied to BF.

4.1. Integral histogram based JBF

TABLE. 4-1 Computational flow and complexity analysis for each pixel in the integral histogram based JBF

Process	Complexity (operation)	BW for IH (data)	BW for pixel (data)
Integration process:			
Pixel count histogram h_c			
Loop b=0 to N_b-1			
$IH_O^S(b)=IH_O^Q(b)+IH_O^R(b)-IH_O^P(b)$	ADD: $3N_b$	$4N_b$	
$IH_O^S(I_S)$ += 1	ADD: 1		
Pixel Intensity histogram h'_c			
Loop b=0 to N_b-1			
$IH_O^S(b)=IH_O^Q(b)+IH_O^R(b)-IH_O^P(b)$	ADD: $3N_b$	$4N_b$	
$IH_O^S(I_S)$ += J_s	ADD: 1		2 pixels
Extraction process:			
Pixel count histogram h_c			
Loop $b=0$ to N_b-1			
$h_c(b)=IH_O^D(b)+IH_O^A(b)-IH_O^B(b)-IH_O^C(b)$	ADD: $3N_b$	$4N_b$	
Pixel Intensity histogram h'_c			
Loop $b=0$ to N_b-1			
$h_c(b)=IH_O^D(b)+IH_O^A(b)-IH_O^B(b)-H_O^C(b)$	ADD: $3N_b$	$4N_b$	
Kernel calculation process:			
Loop b=0 to N_b-1	ADD, LUT:		
$G(b) = g(\|I_c-b\|)$	N_b		1 pixel
Convolution process:			
Nu=0, De=0	MUL, ADD:		
Loop b=0 to N_b-1	N_b		
De += $G(b)$ x $h_c(b)$	MUL, ADD:		
Nu += $G(b)$ x $h'_c(b)$	N_b		
Result = Nu / De	DIV: 1		1 pixel
Total	**$17N_b$+3**	**$16N_b$**	**4 pixels**

TABLE. 4-1 presents the computational flow and computational analysis of the integral histogram based JBF to calculate 1-pixel result, which consists of the *integration*, *extraction*, *kernel calculation*, and *convolution processes*. In which, the former two are for the histogram calculation step, and the latter two are for the 1-D convolution step. Especially note that these processes, for each pixel, should compute for all bins of related histograms; therefore, their complexity and bandwidth for integral histogram (bandwidth for IH) are the multiple of the number of bin, N_b.

For ease of explanation, we use the *area view* (image space) to show how this approach operates and the *memory view* (IH space) to show the memory usage, as illustrated in Fig. 4.1 (a). In the *area view*, IH_O^X is a histogram of the rectangular area stretched from the pixel O to X. Thus, the addition and subtraction of IH can be regarded as area merging and cutting, respectively. In the *memory view*, the data of IH_O^X are stored at X, and the gray region represents occupied memory usage. With these representations, Fig. 4.1 (b) and (c) illustrate the *integration* and *extraction processes*.

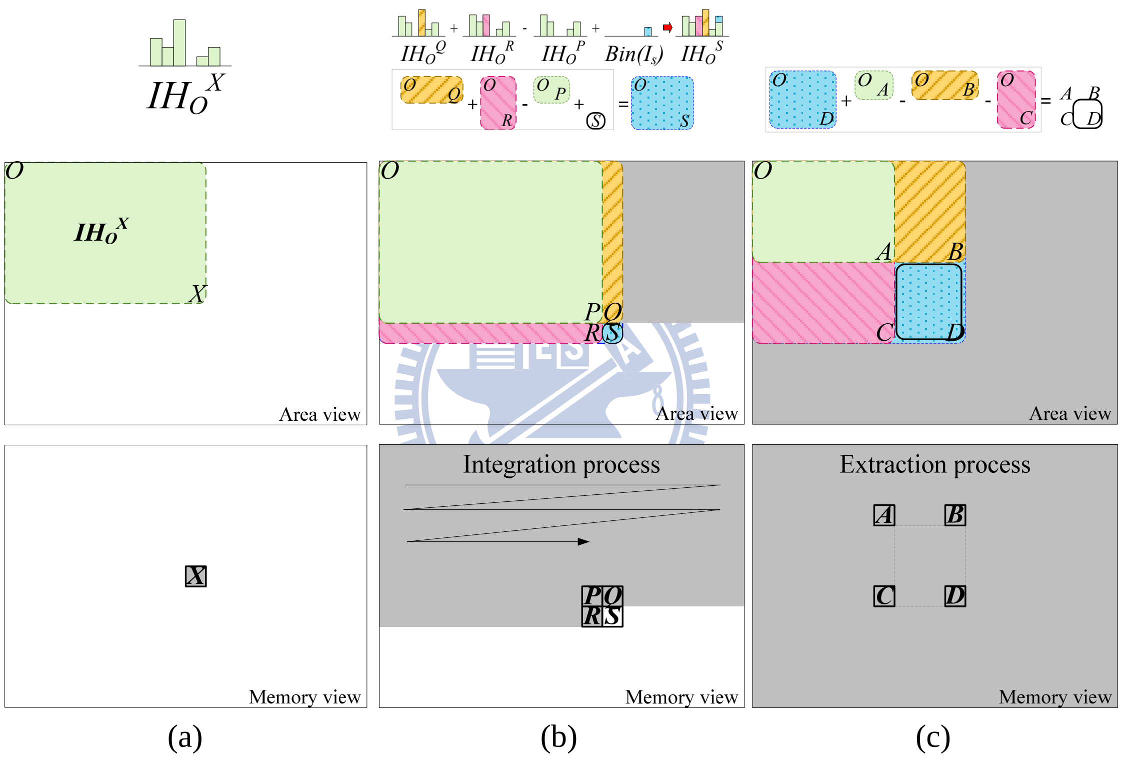


Fig. 4.1. Concept of integral histogram approach
(a) representation of an integral histogram, (b) *integration process*,
(c) *extraction process*.

First, the *integration process* progressively calculates the IH of each pixel using the equation,

$$IH_O^S = IH_O^Q + IH_O^R - IH_O^P + Bin(I_S) \tag{4.2}$$

For the pixel count histogram h_c and the pixel intensity histogram h'_c, their IHs are computed separately as shown in TABLE. 4-1. The histogram IH_O^S is computed from linearly combination of three exist integral histograms and a histogram of the target pixel I_S. We show the target pixel histogram with the notation $Bin(I_S)$ because the histogram must be a one-hot histogram. For h_c, $Bin(I_S)$ is 1 for the corresponding bin and 0 for others; on the other hand, for h'_c, this term is J_s for the corresponding bin, and also 0 for others. Adding the one-hot histogram updates only the bin corresponding to I_S so that, as shown in TABLE. 4-1, it is perform outside the loop. After this process, the IH of each pixel is produced and stored into memory.

Second, given the IHs, the *extraction process* can extract h_c or h'_c, the histograms of the window *ABCD,* which is centered by the target pixel c, is defined by equation,

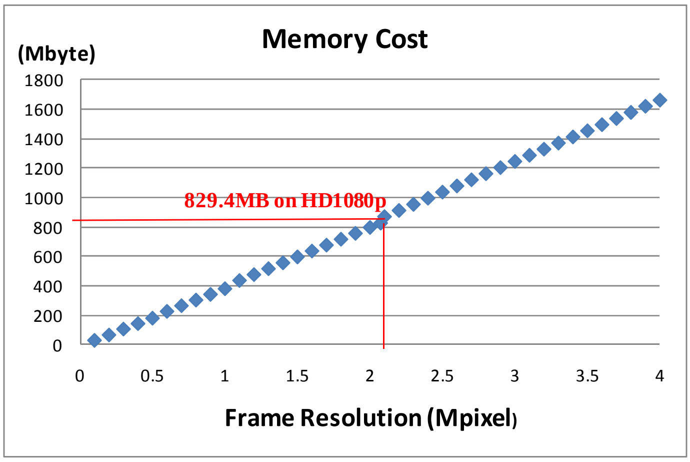
$$h_c \text{ or } h'_c = H_{ABCD} = IH_O^D + IH_O^A - IH_O^B - IH_O^C \tag{4.3}$$

As shown in Fig. 4.1 (c), a histogram with arbitrary window size can be obtained by using the IHs of four corners. With this property, the integral histogram approach can reduce computational complexity to O(|R|/s_r) which is independent of window size.

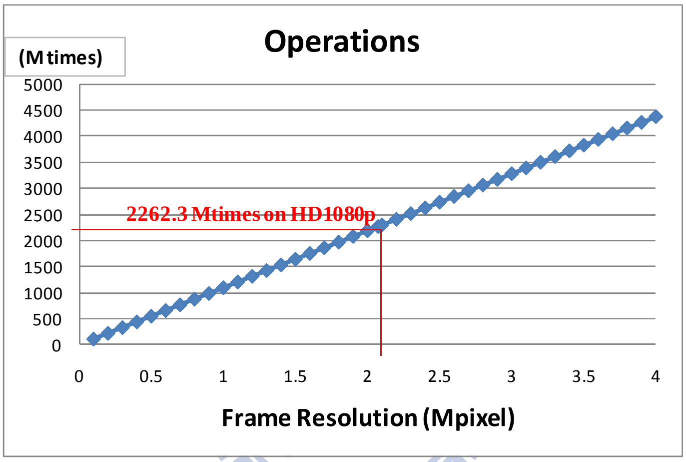
Third, the kernel calculation process computes the range kernel by a range table, which includes 256 items for the 256 possible values of $|I_c\text{-}b|$. Finally, given the range kernel g and the histograms h_c and h'_c, the convolution process calculates the result of target pixel c by (4.1).

4.2. Design Challenge

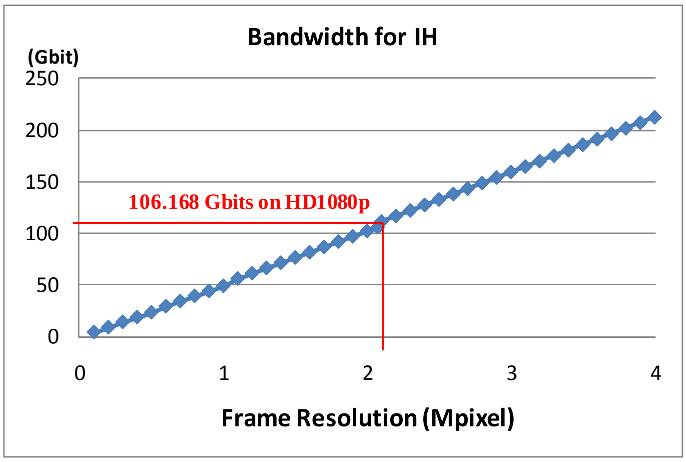
Since the complexities listed in TABLE. 4-1 are pixel wise as well as bin number dependent, they will grow quickly, as shown in Fig. 4.2, as resolution and bin number grow. The detailed design challenges are described below.



(a)



(b)



(c)

Fig. 4.2. Analysis of Design Challenges over frame resolutions

With N_b=64; (a) Memory cost, (b) Operations, (c) Bandwidth for IH

4.2.1. High Memory Cost for integral histograms

During the *integration process*, all the IHs of whole image are stored in memory. BF needs a frame-scale-magnitude memory for h_c, and JBF additionally needs another one for h'_c. Therefore, the total memory cost of JBF is

$$MN \cdot N_b w_b + MN \cdot N_b (w_b + 8) \tag{4.4}$$

where the former term is for h_c, and the later term is for h'_c. M and N is the frame height and width, N_b is the number of bin, and w_b is the bit width of a bin. Note that w_b is related to the maximal integral area, and its value equals $\log_2(MN)$. In addition, the bit width of h'_c is more than h_c by 8 bits because pixel intensity is 8 bits.

Above memory cost would be 829.4 Mbytes for the HD1080p resolution as listed in Fig. 4.2 (a) and TABLE. 6-4. For a VLSI design, these massive data could be configured into off-chip memory (i.e. DRAM) or on-chip memory (i.e. SRAM). However, compared to the on-chip memory, the off-chip memory suffers from longer access latency due to its complicated controlling mechanism [39], and limited bandwidth usage due to bus sharing by multiple masters. Hence, our strategy for the high memory cost is to reduce the memory requirement and enable data to be stored in on-chip memory for fast implementation.

4.2.2. High Computational Complexity in All Processes

According to the complexity in TABLE. 4-1, generating 1-pixel result needs $15N_b$+2 additions, $2N_b$ multiplications, and 1 division. If N_b is 64, the total complexity will be 2,262.3 million operations for an HD1080p image as shown in Fig. 4.2 (b). To meet above demands, a VLSI design with sufficient parallel operators is necessary.

4.2.3. High Bandwidth in Integration and Extraction

In TABLE. 4-1, the bandwidth for IH requires $16N_b$ for 1-pixel result, and that will reach 106.168 Gbits for an HD1080p image as shown in Fig. 4.2 (c) and TABLE. 6-4. That is because the IHs are accessed frequently. With the strategy for the memory cost problem, the IHs are stored in on-chip memory, and its data bus should be increased to address the high bandwidth problem. However, it results in over-partitioned memory and increased area. Thus, a method which can reduce the bandwidth is needed.

4.2.4. Large Range Table in Kernel Calculation

In the *kernel calculation* process, a range table with 256 items is needed. However, with the parallel operations for the computational complexity problem, this table should be duplicated. By straightforward implementation, 256 range tables, each of which corresponds to 256 possible values of $(I_c - I_q)$, must be available for parallel operations. Both the size (number of items) and the number of the range table result in large area; therefore, a table-reduction method and a table-reuse method are needed

4.3. Summary

In conclusion, for example of the HD1080p image, the integral histogram approach needs the memory cost of 829 Mbytes and the bandwidth of 106 Gbits per frame. In addition, the Porikli's approach still suffers from high computational complexity of 2,262 million operations even though it has been accelerated by integral histogram approach. Moreover, the 1-D convolution needs a large range table with 256 items for the range kernel. Due to above problems, it is hard to achieve a real time performance and thus demands VLSI hardware acceleration. In the next chapter, we will introduce our proposed memory reduction methods. And then in Chapter 6, a VLSI implementation with problem solving architecture will be addressed.

5. Proposed Memory Reduction Methods

5.1. Overview

To solve the high memory cost problem mentioned in last chapter, we propose three memory reduction methods. First, the runtime updating method (RUM) takes advantage of progressive raster-scan process to discard unnecessary data. Second, the stripe based method (SBM) avoids frame wide memory cost by dividing each frame into vertical stripes and processing them one by one. Finally, the sliding origin method (SOM) lessens the storage data dependency of the original histogram integration process to reduce the memory requirement from frame-scale-magnitude to line-scale-magnitude. With these memory methods, the memory cost can be reduced to 0.003%-0.020%. The details of the proposed methods are described below.

5.2. Runtime Updating Method (RUM)

The concept of the RUM is to perform the *integration process* and the *extraction process* at the same time, instead of two separate iterations in the original flow. Fig. 5.1 illustrates its memory configuration in the *memory view*. In Fig. 5.1 (a), the *integration process* is performed from the integral origin O to D. In the meanwhile, the *extraction process* can extract the histogram H_{ABCD} as shown by Fig. 5.1 (b). From the data lifetime analysis for raster-scan, this is the last time taking IH_O^A into extraction process. And all the IHs before the pixel A will not be used for *extraction process* anymore. Hence, only the IHs from the pixel from A to D require memory space. Thus, the memory cost is

$$|S|N \cdot N_b w_b + |S|N \cdot N_b (w_b + 8) \quad (5.1)$$

where M in (4.4) is replaced by the window width $|S|$.

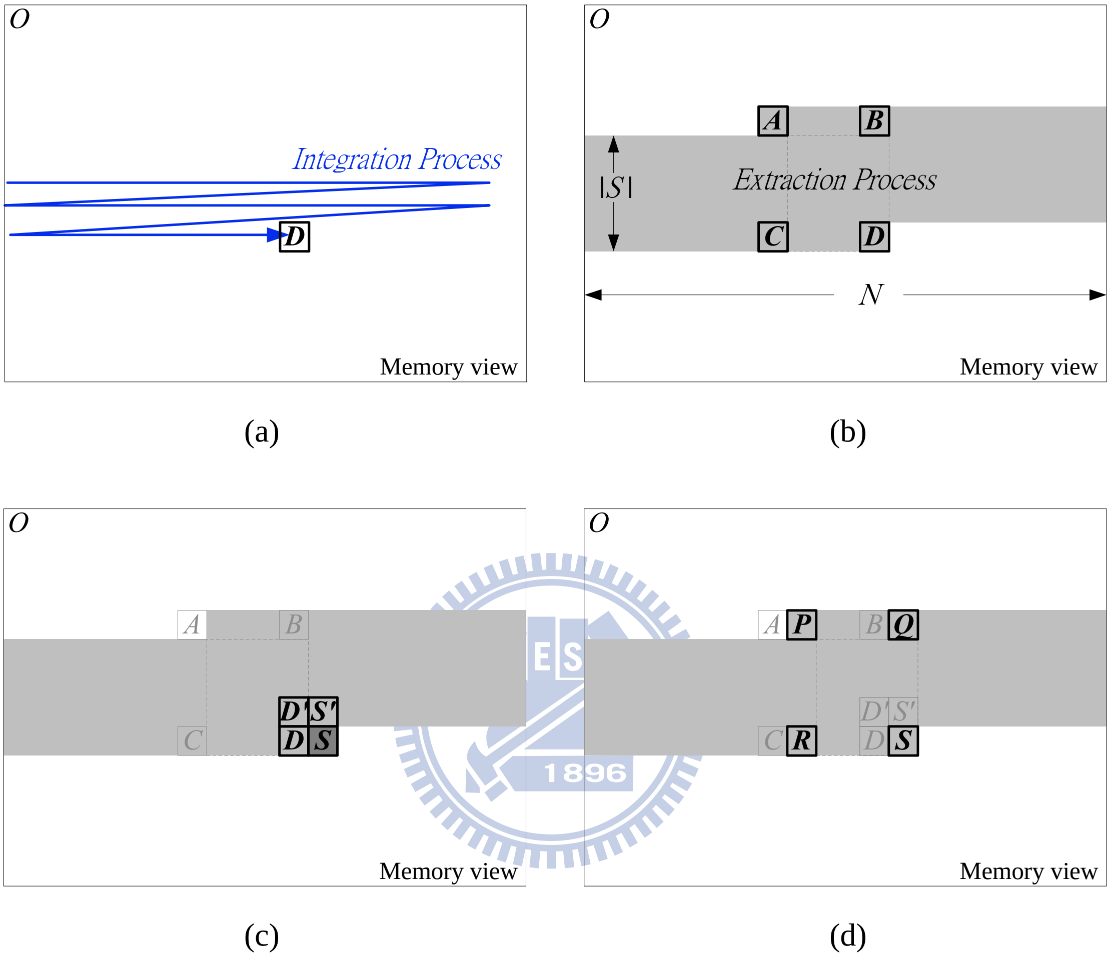


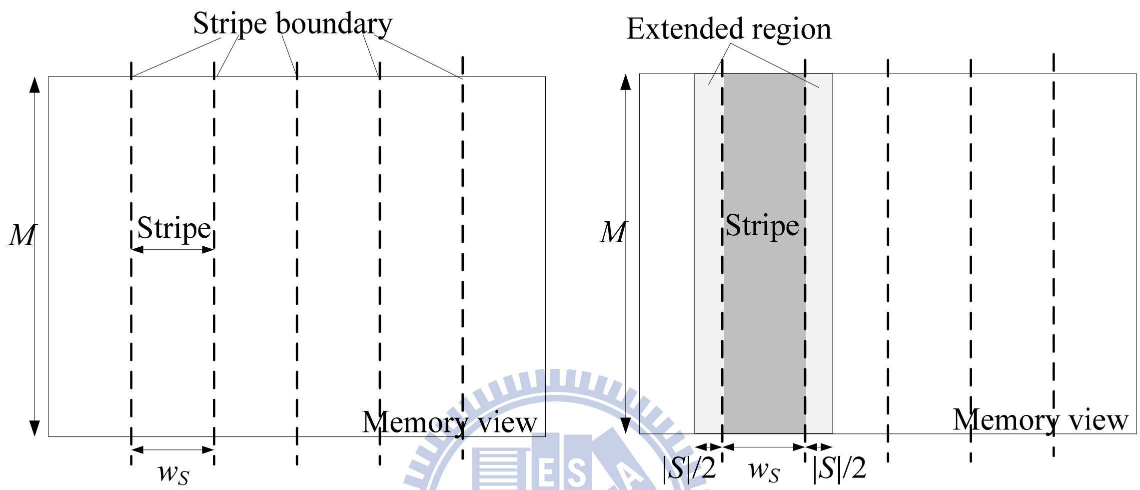
Fig. 5.1. Runtime updating method (RUM)

(a) *integration process*, (b) *extraction process* for H_{ABCD}, (c) *integration process* for S, (d) *extraction process* for H_{PQRS}

Fig. 5.1 (c) and (d) illustrate the memory updating process when the two processes moves right to the next pixel S. In Fig. 5.1 (c), the *integration process* calculates the new IH_O^S using IH_O^D, $IH_O^{D'}$, $IH_O^{S'}$, and then the new IH_O^S can overwrite the memory position of the discarded IH_O^A. In Fig. 5.1 (d), the *extraction process* extracts H_{PQRS}. On the whole, in raster scan from integral origin O to the end of region, the proposed RUM alternates between these two processes repeatedly.

With the proposed RUM, the memory cost could be reduced from a full frame to a partial frame. This method can gain considerable reduction since $|S|$ is usually much smaller than M.

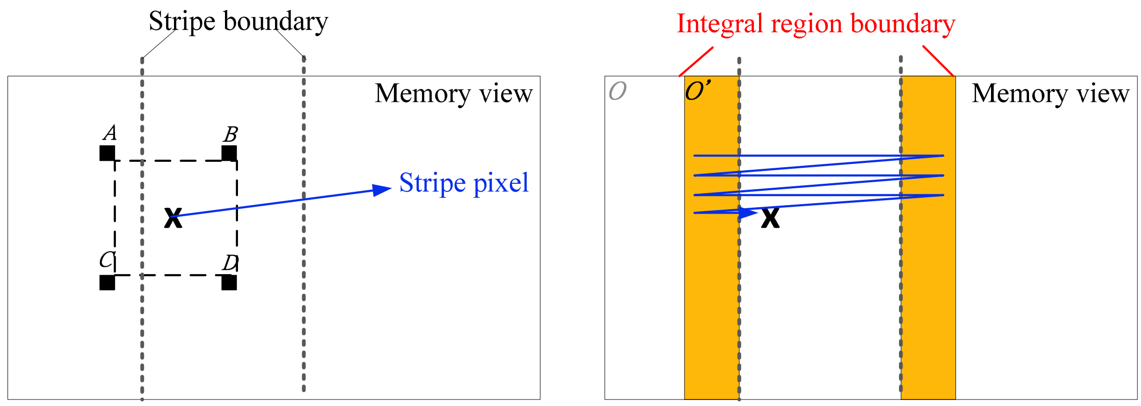
5.3. Stripe Based Method (SBM)



(a) (b)

Fig. 5.2. Stripe based method (SBM)

(a) partitioned-frame, (b) extended integral region for each stripe



(a) (b)

Fig. 5.3. Integral region of SBM is an extended stripe.

(a) four corner IHs for *extraction process*, (b) *integration process*.

The main idea of the SBM is to slice the whole frame into many vertical stripes, and the *integration* and *extraction processes* are performed stripe by stripe. Fig. 5.2 (a) illustrates a frame partitioned into stripes, and Fig. 5.2 (b) illustrates the extended region for a stripe.

Fig. 5.3 (a) shows that, in the *extraction process*, some corner IHs, such as *A* and *B*, are outside the stripe. Therefore, as shown in Fig. 5.3 (b), the *integration process* should be carried out at extended region to make sure these outside-stripe IHs are defined. On both stripe boundaries, the integral region (IR) is extended by half the window width, $|S|/2$, to include the regions that can be traversed by window corners. Note that these IHs are associated with new origin *O'* instead of *O*.

As shown by Fig. 5.2 (b), the IR of each stripe is ($|S| + w_s - 1$) pixels wide. Compare with the original, the IR is reduced from a frame to an extended stripe; as a result, the bit width w_b can be smaller. The total memory cost of the SBM is

$$M(|S|+w_s-1)\cdot N_b w_b + M(|S|+w_s-1)\cdot N_b(w_b+8)\,, \tag{5.2}$$

where w_s is the stripe width, and w_b equals $\log_2[M(|S|+w_s-1)]$. Compared to the original cost of (4.4), the SBM could significantly reduce memory if the stripe width, ($|S|+w_s-1$), is much smaller than *N*.

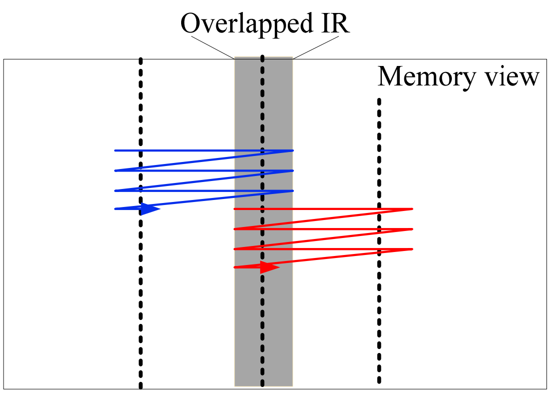


Fig. 5.4. Overlapped integration region between two adjacent stripes

The overhead of the SBM is that the extended regions result in extra computation and bandwidth due to repeatedly performed *integration processes* on these regions as shown by Fig. 5.4. Thinner stripes can reduce memory cost more, but that leads to more overheads. Thus, the selection of w_s is a tradeoff between memory reduction and overheads. That will be discussed in Chapter 6.6.

5.4. Sliding Origin Method (SOM)

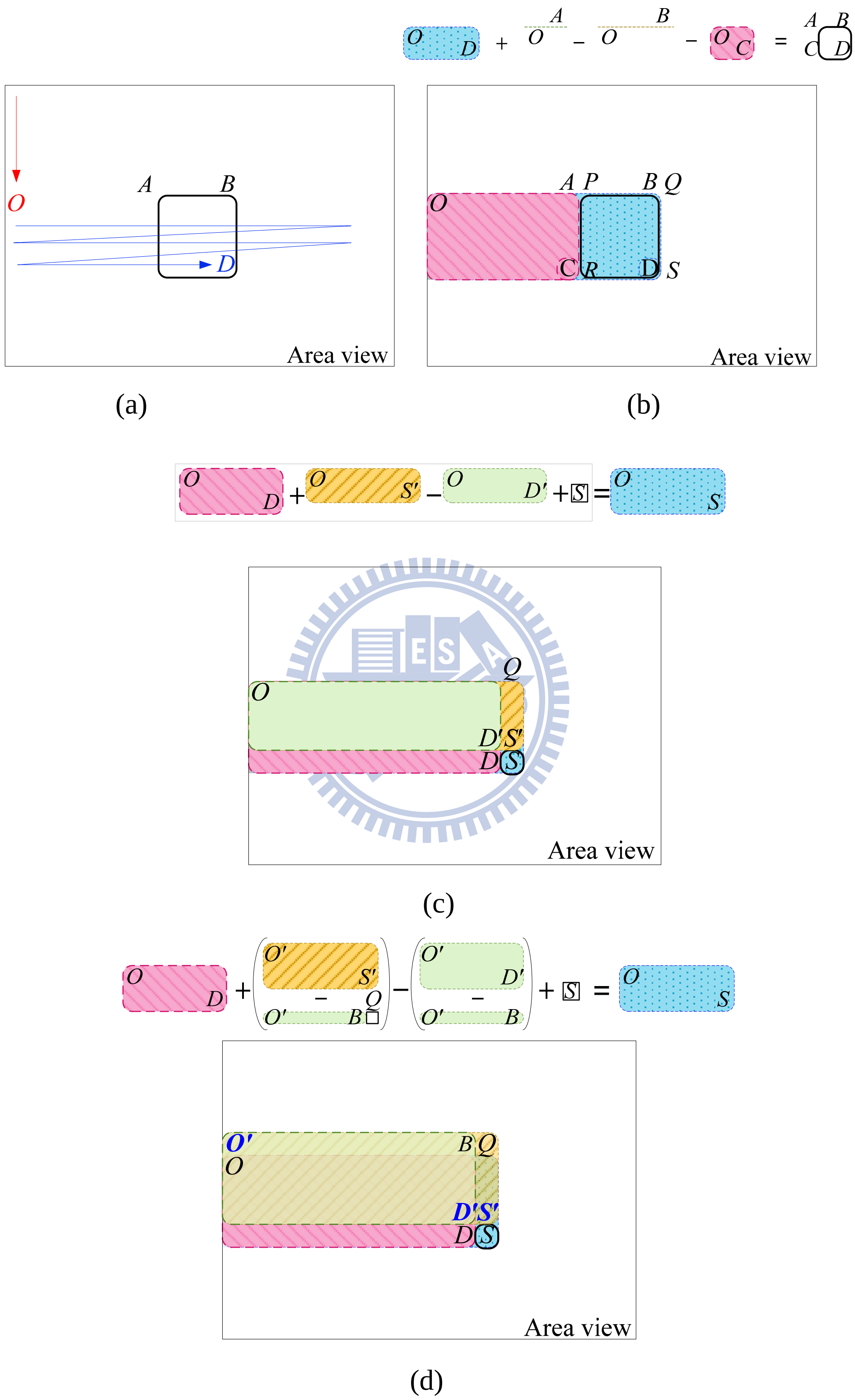


Fig. 5.5. Sliding Origin Method (SOM)

(a) Sliding origin *O*, (b) *extraction process* with sliding origin *O*,

(c) *integration process* for next pixel *S*, (d) modified *integration process*

The concept of the SOM is to vertically slide the origin pixel *O* with the *integration* and *extraction processes* to reduce memory cost from a plane to a single line. As shown in Fig. 5.5 (a), the origin pixel *O* slides downward to keep pace with the top row of the window *ABCD*. With the SOM, the *integration* and *extraction processes* can be simplified as described below.

Fig. 5.5 (b) shows that, for the *extraction process*, the original $IH_O{}^A$ and $IH_O{}^B$ cannot form meaningful histogram rectangles in area view because the position of *O* is under *A* and *B*. Hence, these two histograms are zero and (4.3) can be simplified as the following equation,

$$\begin{aligned} H_{ABCD} &= IH_O^D + IH_O^A - IH_O^B - IH_O^C \\ &= IH_O^D - IH_O^C \end{aligned} \tag{5.3}$$

Fig. 5.5 (c) shows that, for the *integration process*, the new $IH_O{}^S$ is computed by

$$IH_O^S = IH_O^D + IH_O^{S'} - IH_O^{D'} + Bin(I_S) \tag{5.4}$$

However, the *S'* and *D'* are on the previous row, and by SOM they should have been defined by the previous origin *O'* as shown in Fig. 5.5 (d), instead of *O*. Therefore, for real process, the $IH_O{}^{S'}$ and $IH_O{}^{D'}$ in (5.4) should be changed to $IH_{O'}{}^{S'}$ and $IH_{O'}{}^{D'}$ by the following derivation, which is corresponding to the *area view* of Fig. 5.5 (d).

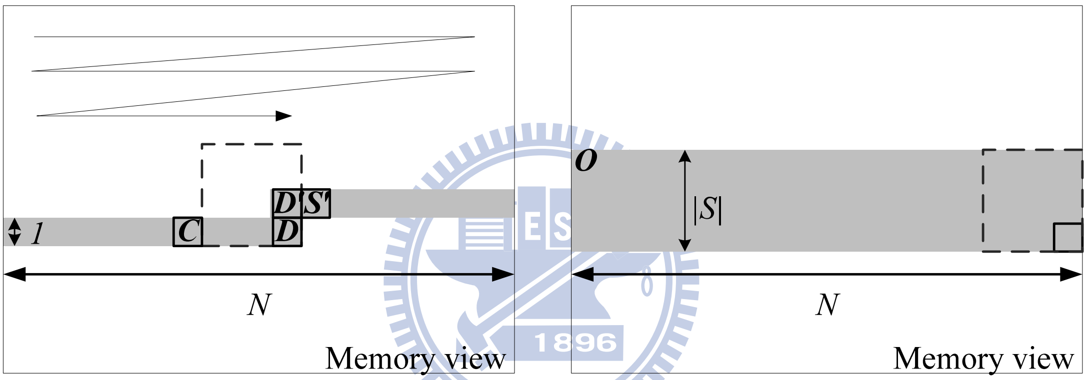
$$\begin{aligned} IH_O^S &= IH_O^D + IH_O^{S'} - IH_O^{D'} + Bin(I_S) \\ &= IH_O^D + \left(IH_{O'}^{S'} - IH_{O'}^{Q}\right) - \left(IH_{O'}^{D'} - IH_{O'}^{B}\right) + Bin(I_S) \\ &= IH_O^D + \left(IH_{O'}^{S'} - \left(IH_{O'}^{B} + Bin(I_Q)\right)\right) - \left(IH_{O'}^{D'} - IH_{O'}^{B}\right) + Bin(I_S) \\ &= IH_O^D + IH_{O'}^{S'} - Bin(I_Q) - IH_{O'}^{D'} + Bin(I_S) \end{aligned} \tag{5.5}$$

Compare with (5.4), final line of (5.5) shows that only a slight change is required (it adds the term of subtracting $Bin(I_Q)$) for the *integration process* to let the integral origin slide from *O'* to *O*.

However, the slight change makes significant difference on the memory cost. With above simplification, only the IHs of *C*, *D*, *S'* and *D'* are associated, and by the concept of RUM, only a single row of IHs from *D'* to *D,* requires memory space as shown in Fig. 5.6 (a). Thus, the total memory cost is reduced as

$$N \cdot N_b w_b + N \cdot N_b (w_b + 8) \qquad (5.6)$$

where w_b equals $\log_2(|S|N)$ since the maximal IR is $|S|N$ as shown by Fig. 5.6(b). Compared to the original cost of (4.4), the height dimension M is replaced by $|S|$, and w_b is much smaller because $|S|$ is usually much smaller than image width M.



(a) (b)

Fig. 5.6. Sliding Origin Method
(a) Memory Cost, (b) Maxima integral region.

5.5. Combination

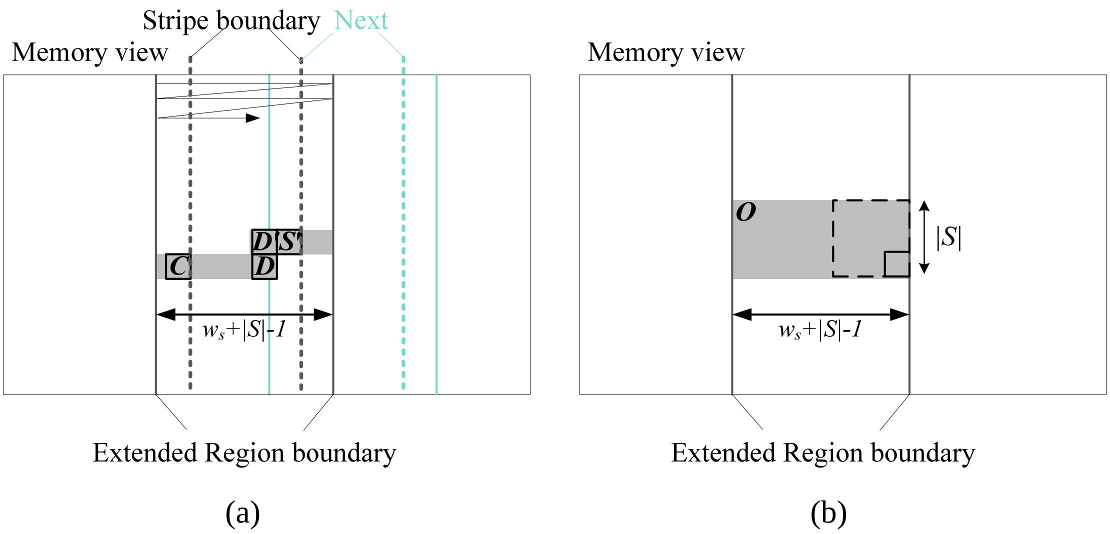


Fig. 5.7. Combination of memory reduction methods
(a) Memory Cost, (b) Maximum integral region

The proposed memory reduction methods could be easily combined as shown by Fig. 5.7 (a). First, the SBM partitions a whole frame into stripes. Then, by stripes, the RUM and SOM are performed row by row. This combination can reduce the memory cost to

$$(|S|+w_s-1)\cdot N_b w_b + (|S|+w_s-1)\cdot N_b(w_b+8) \quad (5.7)$$

where w_b equals $\log_2[|S|(|S|+w_s-1)]$, $|S|(|S|+w_s-1)$ is the area of maximum integral region as shown by Fig. 5.7 (b). Compared to the original cost of (4.4), M is decreased to 1 since the SOM reduces data dependency of the *extraction process* and the RUM discards unnecessary data. Besides, N is decreased to $(|S|+w_s-1)$ since SBM cuts image into narrow stripes. Note that in this memory cost formulation, N_b and $|S|$ are related to the application quality, and w_s is related to hardware performance. The analysis of parameter selection will be further presented in Chapter 6.6.

5.6. Comparisons

Refer to the analysis in [34], we use the 31-pixel-wide window (i.e. $|S|$ is 31) and 64-bin histogram (i.e. N_b is 64). In addition, we choose stripe width as 60 pixel (i.e. w_s is 60) as an example and compare the original memory cost defined by equation (4.4) and the reduced memory cost computed by equation (5.7) for different frame resolutions. TABLE. 5-1 shows that the reduced memory cost is independent of the frame resolution. With above mentioned parameters, the memory cost is 23.04 Kbytes constantly. The amount is 3 to 5 decimal magnitude smaller than the original memory costs of different resolutions. For every resolution, other than the number of required integral histograms is reduced from frame-scale-magnitude to a line-scale-magnitude, its w_b is also reduced due to the IR reduction.

TABLE. 5-1 Comparisons of original and reduced memory cost

Cost Unit: Bytes
Integration region (IR) Unit: square pixel
Histogram bin bit width (w_b) Unit: bit

Resolution	CIF (352x288)	VGA (640x480)	HD720p (1280x720p)	HD1080p (1920x1080p)	4Kx2K
Original cost	34.1M	113.0M	353.9M	829.4M	3456M
Original *IR*	101.3K	307.2K	921.6K	2073.6K	8M
Original w_b	17	19	20	21	23
Reduced cost	23K(0.067%)	23K(0.02%)	23K(0.0065%)	23K (0.0028%)	23K(0.0007%)
Reduced *IR*	2.79K (2.75%)	2.79K(0.91%)	2.79K(0.30%)	2.79K(0.13%)	2.79K(0.03%)
Reduced w_b	12 (70.59%)	12 (63.16%)	12 (60%)	12 (57.14%)	12 (52.17%)

$|S|=31$; $w_s=60$; $N_b=64$

6. Architecture Design and Implementation

6.1. Overview

TABLE. 6-1 Modified computational flow and complexity analysis for each pixel in the integral histogram approach for JBF

Process	**Complexity (operation)**	**BW for IH (data)**	**BW for pixel (data)**
Integration process:			
Pixel count histogram h_c			
Loop b=0 to N_b-1			
$IH_O^S(b)=IH_O^D(b)+IH_{O'}^{S'}(b)-IH_{O'}^{D'}(b)$	ADD: 2N_b	4N_b	
$IH_O^S(I_S)$ += 1, $IH_O^S(I_Q)$ -= 1	ADD: 2		
Pixel Intensity histogram h'_c			
Loop b=0 to N_b-1			
$IH_O^S(b)=IH_O^D(b)+IH_{O'}^{S'}(b)-IH_{O'}^{D'}(b)$	ADD: 2N_b	4N_b	
$IH_O^S(I_S)$ += J_S, $IH_O^S(I_Q)$ -= J_Q	ADD: 2		4 pixels
Extraction process:			
Pixel count histogram h_c			
Loop b=0 to N_b-1			
$h_c(b) = IH_O^S(b) - IH_O^R(b)$	ADD: N_b	N_b	
Pixel Intensity histogram h'_c			
Loop b=0 to N_b-1			
$h_c(b) = IH_O^S(b) - IH_O^R(b)$	ADD: N_b	N_b	
Kernel calculation process:			
Loop b=0 to N_b-1			
$G(b) = g(\|I_c-b\|)$	ADD, LUT: N_b		1 pixel
Convolution process:			
Nu=0, De=0			
Loop b=0 to N_b-1			
De += $G(b)$ x $h_c(b)$	MUL, ADD: N_b		
Nu += $G(b)$ x $h'_c(b)$	MUL, ADD: N_b		
Result = Nu / De	DIV: 1		1 pixel
Total	**11N_b+5**	**10N_b**	**6 pixels**

With memory reduction methods introduced in last chapter, the computational flow of JBF in TABLE. 4-1 is changed to that in TABLE. 6-1, and its hardware cost is presented in TABLE. 6-3. The *integration process* has added an I_Q-relate subtraction term and the *extraction process* has simplified to be a two-term process. Therefore, the corresponding complexity and bandwidth are reduced consequently. And these reduction methods have reduced the memory cost from frame-scale-magnitude to line-scale-magnitude. On the other hand, there are still three problems left to be solved with VLSI implementation. They are high parallelism-demand problem, high bandwidth problem, and large range table problem.

To solve these problems and efficiently implement the architecture, we first propose the *R-parallelism* method to execute parallel computations in range domain to meet required throughput. Then, for on-chip bandwidth reduction, we take advantages of the timing relationship of data in the progressive computation to buffer the computed IHs, named *delay-buffer method*. The large range table size due to parallelism is further reduced by exploiting the numerical properties of Gaussian function. With memory reduction methods and these architecture design techniques, an efficient hardware design is proposed, which can be easily scalable to different performance target. For ease of explanation, we use an example for the performance target of HD1080p resolution to present the design. The details of these design techniques are presented in the rest of this chapter.

6.2. Overall architecture

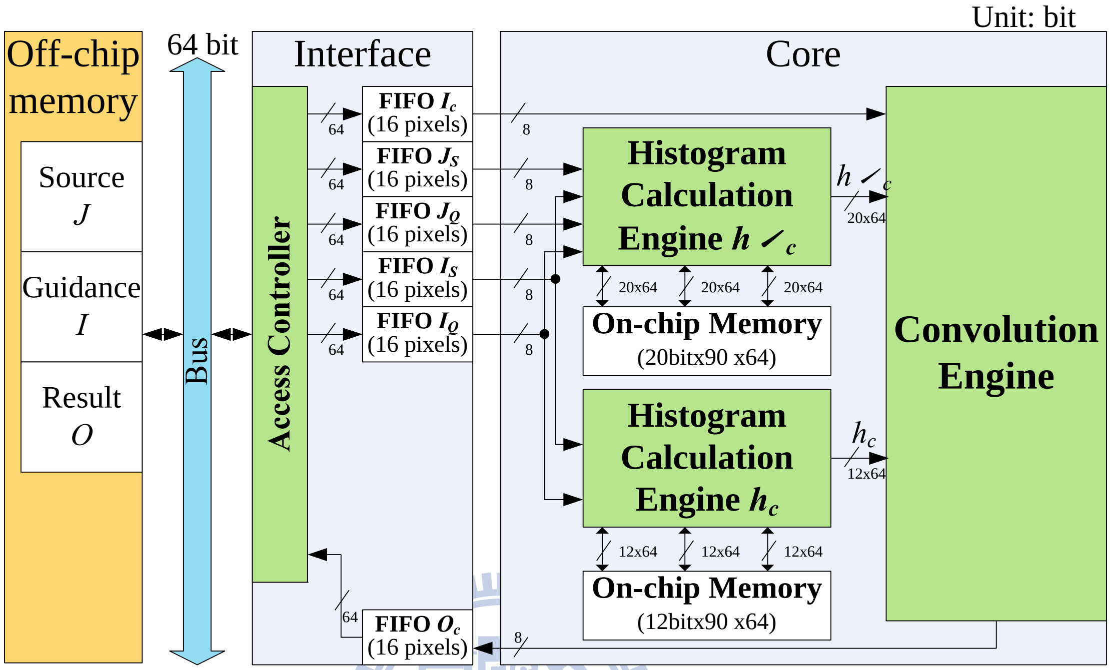


Fig. 6.1. Proposed architecture of JBF

Fig. 6.1 shows the overall architecture that contains two parts, interface and core. In this architecture, the image pixels and the IHs are stored at the off-chip and on-chip memory, respectively. The interface accesses pixels from the off-chip memory through a 64-bit bus, and the core performs the computation of JBF.

In the interface, the access controller allocates the bus priority to the input and output first-in-first-out (FIFO) buffers by round-robin policy. The size of each buffer is associated with off-chip bandwidth. Large buffers can support data reuse schemes to reduce the off-chip bandwidth. Because of sufficient off-chip bandwidth in this architecture, we do not apply any data reuse schemes here to have lower buffer cost, and set its size as 2x8-pixel, where the value of 8 is to meet the bus width, and the value of 2 is to support ping-pong mechanism for simultaneous reading and writing.

6.3. Interface

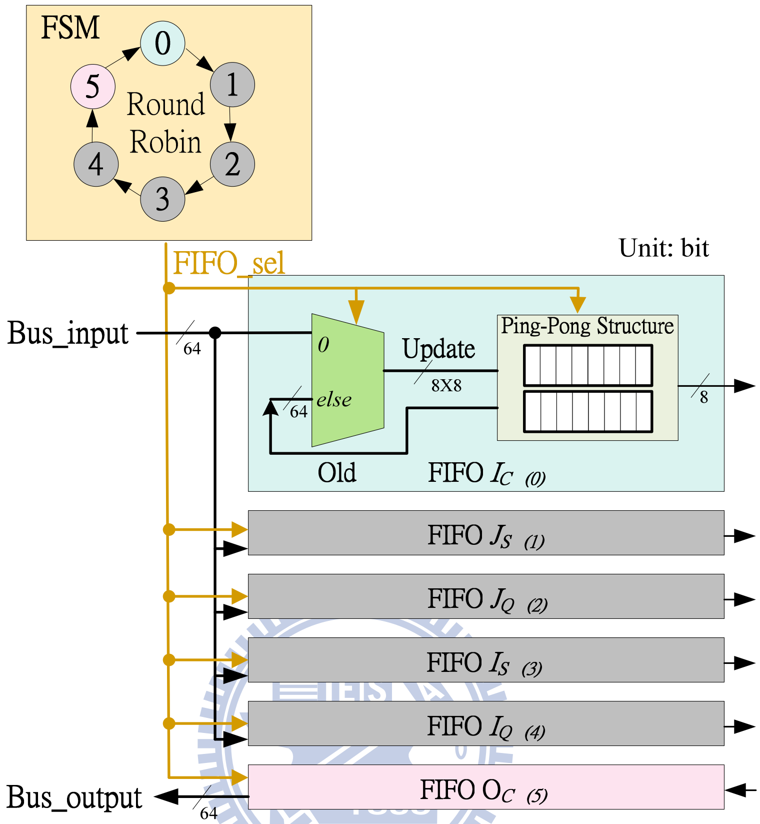


Fig. 6.2. Mechanism of input and output data control

In the interface, the round-robin finite state machine (FSM) has six states. State 0 to 4 associate to input FIFO buffers; state values determine which FIFO buffer should take the input of an 8-pixel data. For example, as shown by Fig. 6.2, the FIFO buffer of I_c takes input when state is zero; at the other time, it keeps old stored data. State 5 associates to output FIFO buffer, an 8-pixel packaged result in FIFO buffer of O_c are sent to bus when state is 5; at the other time, this FIFO is loaded with newly processed result from the core.

The FIFO buffer of any input is in 2x8-pixel ping-pong structure. For any time, one of two 8-pixel buffer is in *Update mode* and the other is in *Give mode.* The structure is used to make scheduling time easier because it enables buffer to receive data (by *Update mode* buffer) and to give data (from *Give mode* buffer) at the same cycle. By our schedule, The *Update-mode* buffer will be loaded with an 8-pixel input in a cycle; for example, Fig. 6.3 (a) shows an input is coming and then in Fig. 6.3 (b) the *Update mode* buffer is loaded with the data. At the same cycle, the *Give mode* buffer gives out a pixel into the core. The mode will exchange after *Update mode* buffer is loaded data and *Give mode* buffer gives out all data as shown by Fig. 6.3 (c). After the switching, the loaded data starts to pour out and the empty buffer waits to be loaded again as in Fig. 6.3 (d). During the process, the mode exchanges continuously.

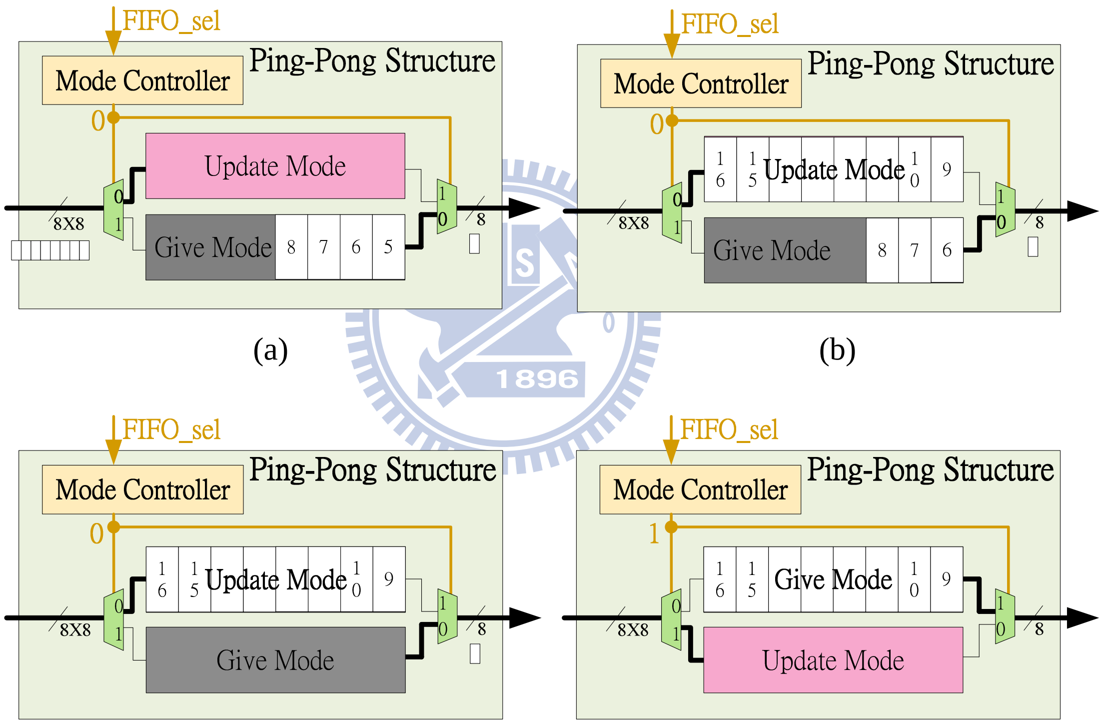


Fig. 6.3. Process of Ping-Pong Structure

(a) input is coming, (b) the next cycle, *Update mode* buffer loaded by input and *Give mode* gives out a pixel, (c) ready for mode exchange, (d) after mode exchange.

6.4. Time Schedule

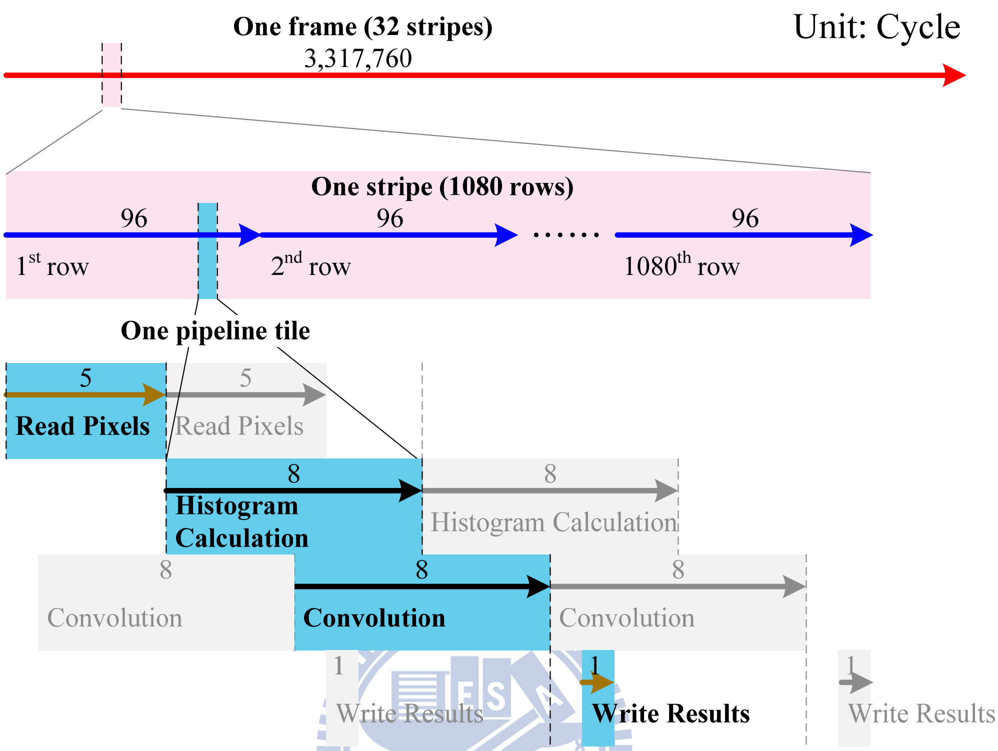


Fig. 6.4. Schedule of the proposed architecture

The operations of the architecture are described below with the schedule in Fig. 6.4, which is hierarchically sliced from a frame to pipeline tiles. The throughput of each pipeline tile is the computational result of 8 pixels. In a pipeline tile, the access controller in the interface first reads pixels from the off-chip memory, and stores them into the FIFO buffers. It takes 5 cycles to switch through 5 states (state 0 to 4) of the round robin FSM. Then the two histogram calculation engines in the core begin to compute h'_c and h_c, and the convolution engine consecutively produces 8 pixel results which are then sent to the output FIFO buffer. Finally, the interface moves 8-pixel packaged results from the buffer to the off-chip memory at the state 5 of FSM.

This schedule refers to the quality analysis in [34], it uses 31 pixels as window width and sets stripe width to be 60 pixels. Therefore, an HD1080p image is sliced into 32 stripes and the width of an integral region is 90 pixels. 12 pipeline tiles are required for each row of integral region since each tile can calculate 8-pixel-wide histogram. By fully-pipelined schedule, performing 12 pipeline tiles takes 96 cycles. To sum up over 32 stripes, for a HD1080p frame, 3,317,760 cycles are needed.

6.5. Design Components

In the core, the main components are two histogram calculation engines and one convolution engine for the TABLE. 6-1 computations, which have high computational complexity as mentioned above. Thus, the proposed *R-parallelism* method unrolls all computational loops in the range domain R. The details of this method are described in each engine as follows.

6.5.1. Histogram Calculation Engine

The histogram calculation engines perform the *integration* and *extraction processes* for h_c and h'_c as shown in TABLE. 6-1. With the *R-parallelism* method, we design their architectures as shown in Fig. 6.6, where the selected-bin adder (SBA) is depicted in Fig. 6.5. These two engines can achieve the throughput of 1 histogram per cycle. Note that the difference of the two engines is that the integral value of SBAs is the source pixel J in the engine h'_c, instead of the constant 1 in the engine h_c. In addition, all bit widths of data in the engine h'_c are more than those in h_c by 8 bits.

According to equation (4.2), the integral values, J or 1, should be added into a corresponding bin of guided pixel; at the same time, other bins should keep their origin value. In SBA, before adder, a selector is used to select the corresponding bin; and after adder, a selector array updates the result back to the corresponding bin. All the selectors are controlled according as the value of guided pixel.

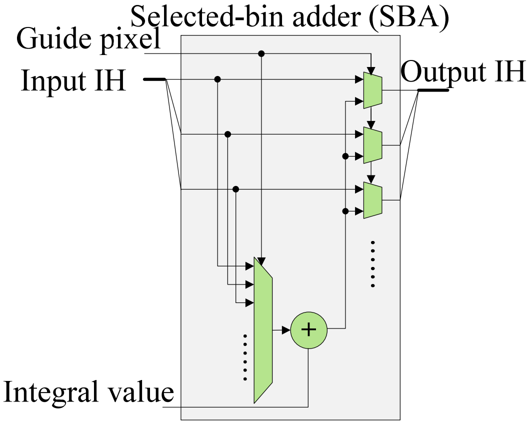


Fig. 6.5. Selected-bin adder in the histogram calculation engines

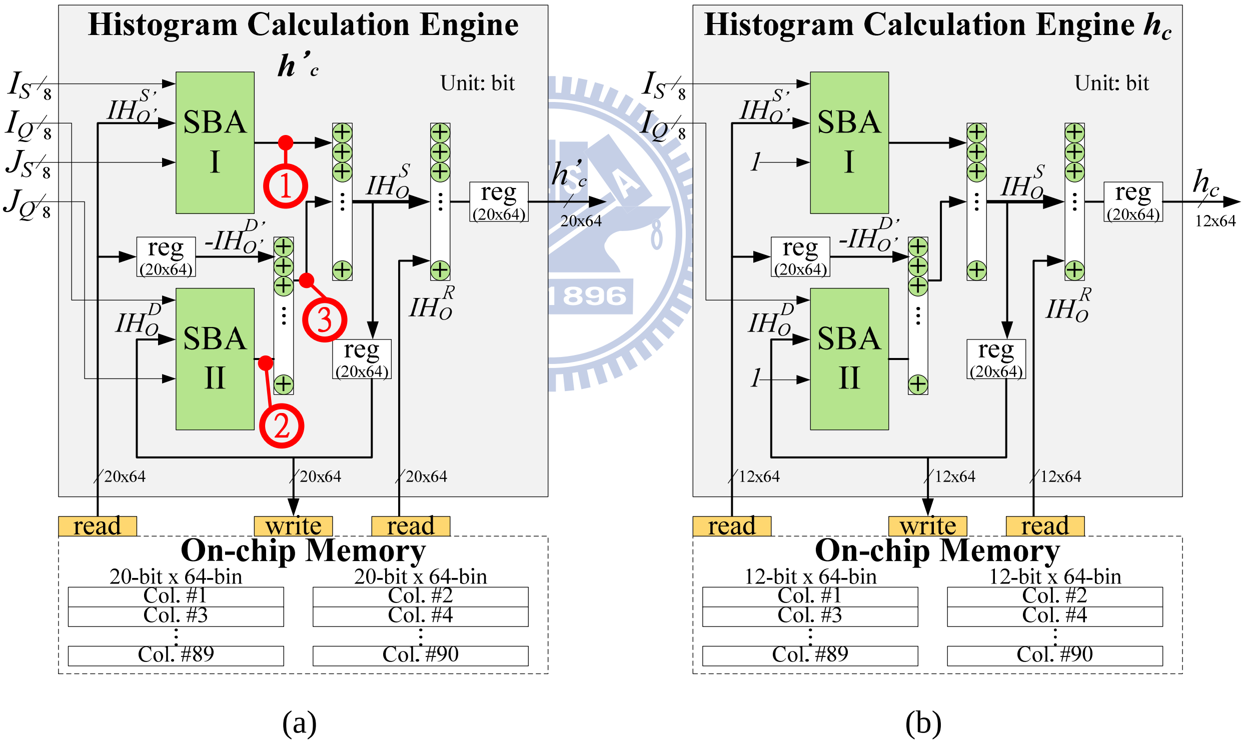
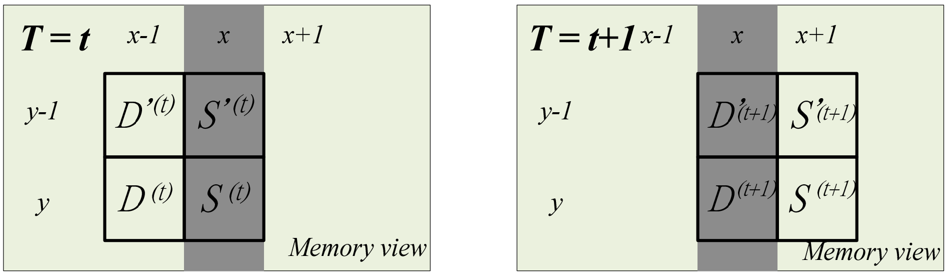


Fig. 6.6. Architectures of histogram calculation engines h'_c and h_c



(a) (b)

Fig. 6.7. The delay-buffer method

(a) $S'^{(t)}$, $S^{(t)}$ at time=t are delayed to be (b) $D'^{(t+1)}$, $D^{(t+1)}$, respectively

In above architectures, each engine needs to access the five IHs: $IH_{O'}^{S'}$, $IH_{O'}^{D'}$, IH_O^S, IH_O^D, and IH_O^R, from on-chip memory in one cycle. To reduce the bandwidth problem, we propose the *delay-buffer method*, which is presented as follows by data dependency of the associated IHs in two successive cycles. Assume that the pixels *S*, *S'*, *D*, and *D'* shown in Fig. 5.5 (d) are located (*x*,*y*), (*x*,*y-1*), (*x-1*,*y*), and (*x-1*,*y-1*) in the cycle *t*, respectively. As shown in Fig. 6.7 (a), their IHs are notated by

$$S^{(t)}: IH_O^{(x,y)}, \quad S'^{(t)}: IH_O^{(x,y-1)}, \quad D^{(t)}: IH_O^{(x-1,y)}, \quad D'^{(t)}: IH_O^{(x-1,y-1)} \tag{6.1}$$

For the next cycle *t+1*in Fig. 6.7 (b), their x-coordinates are increased by 1 as follows,

$$S^{(t+1)}: IH_O^{(x+1,y)}, \quad S'^{(t+1)}: IH_O^{(x+1,y-1)}, \quad D^{(t+1)}: IH_O^{(x,y)}, \quad D'^{(t+1)}: IH_O^{(x,y-1)} \tag{6.2}$$

From the (6.1) and (6.2), we can find that $D^{(t+1)}$ equals $S^{(t)}$, and $D'^{(t+1)}$ equals $S'^{(t)}$. That means $IH_{O'}^{D'}$ and IH_O^D can be obtained by delaying $IH_{O'}^{S'}$ and IH_O^S for one cycle, respectively. Therefore, we can use two delay-buffers to avoid accessing $IH_{O'}^{D'}$ and IH_O^D from the on-chip memory, and reduce bandwidth from five IHs to three IHs.

The on-chip memory is divided into two banks, because there are two read demands from the engine. One demand is for $IH_{O'}^{S'}$ and the other is for IH_O^R. As shown in Fig. 6.8, it marks even bank and odd bank of memory with white and dark respectively. It shows that choosing stripe width w_b as an even number can make two reading demands from different banks.

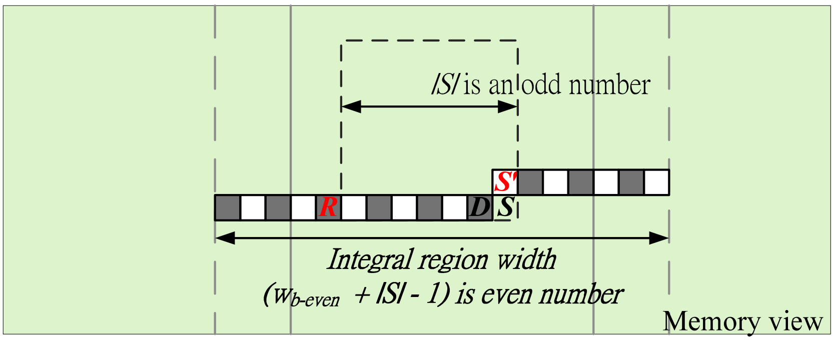


Fig. 6.8. On-chip memory with even bank and odd bank

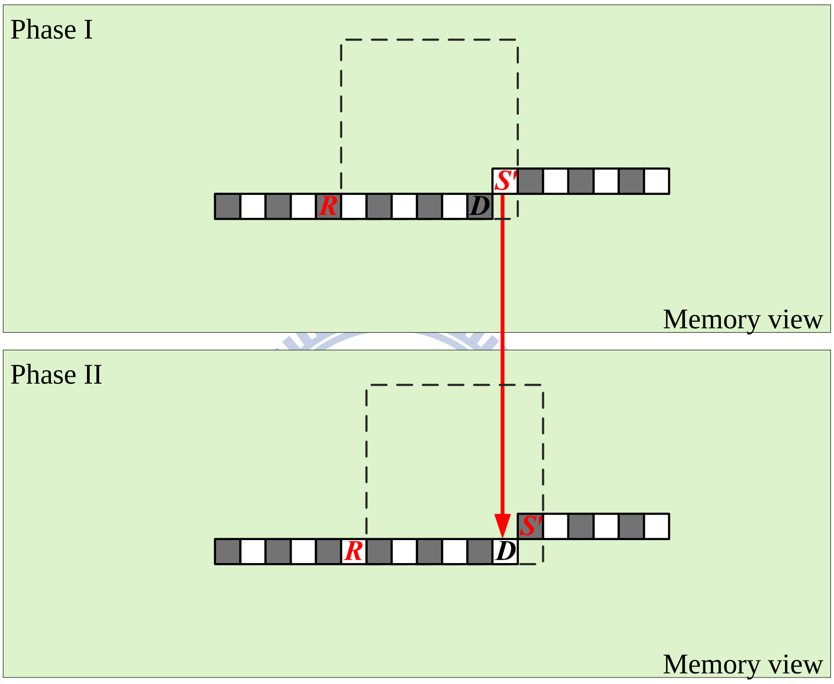


Fig. 6.9. Schedule phases of on-chip memory

The detail schedule is performed in two alternating phases. With these phases, the even bank and odd bank of on-chip memory are alternatively used for reading and writing as shown by Fig. 6.9. At the phase I, $IH_O^{S'}$ and IH_O^{R} are read from the even bank and the odd bank, respectively. In the meanwhile, IH_O^{D} is written into the odd bank. Then at the phase II, IH_O^{D} is written into the different (even) bank. As the arrow shows, the written IH_O^{D} replaces the oldest integral histogram ($IH_O^{S'}$ of the prior phase) since this data will not be used anymore. In the meanwhile, $IH_O^{S'}$ and IH_O^{R} are read from the odd bank and the even bank, respectively. On the whole, the two phases exchange iteratively for the overall engine process.

In the following paragraphs, we will explain the computation of the two histogram calculation engines. Their computation flows are almost the same; therefore, we show the detail only with engine of h'_c.

The computation of the SBA I in Fig. 6.6 (a) is defined by (the check point one)

$$IH_1 = IH_{O'}^{S'} + Bin(I_S), \tag{6.3}$$

which means one of bins of $IH_{O'}^{S'}$ is added by J_S.

The computation of the SBA II in Fig. 6.6 (a) is defined with check point one by

$$IH_2 = IH_O^D - Bin(I_Q), \tag{6.4}$$

which means one of bins of IH_O^D is subtracted with J_Q.

The *integration process* result IH_O^S is calculated by

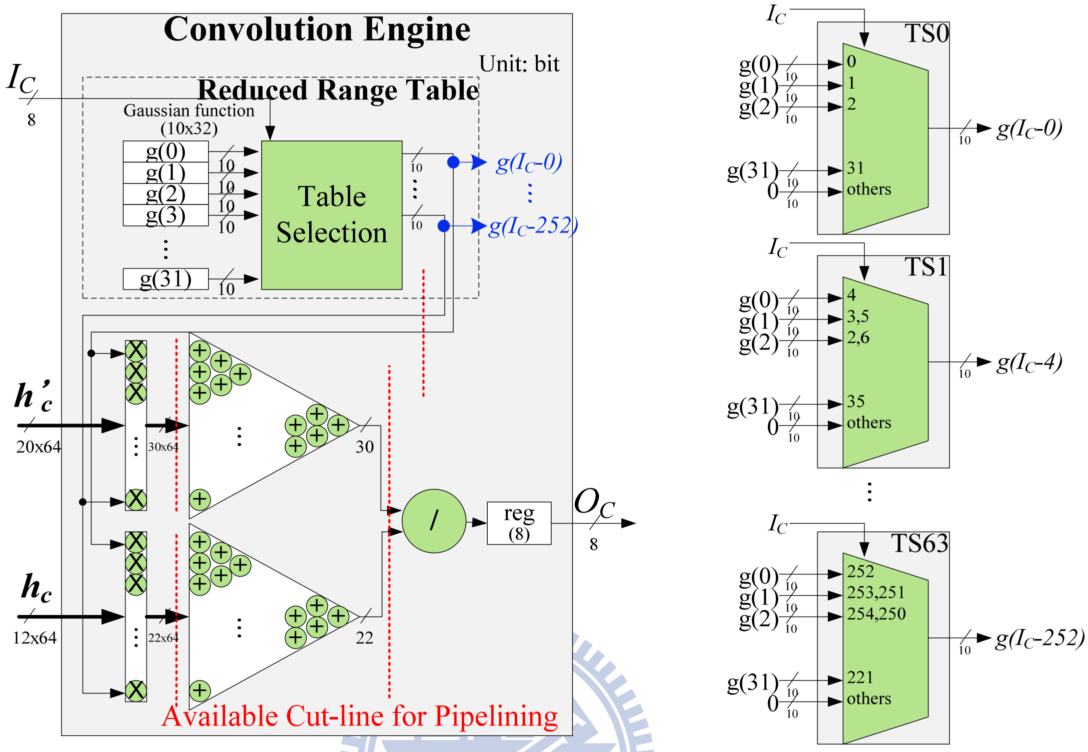
$$\begin{aligned} IH_O^S &= IH_1 + IH_3 \\ &= IH_1 + (IH_2 - IH_{O'}^{D'}) \end{aligned}, \tag{6.5}$$

which is the same as (5.5). Especially note that the addition and subtraction in (6.5) represents additions and subtractions of all bins respectively. With *R-parallelism* method, they are implemented by an array of adders. The number of adders is equal to the number of bins N_b. Finally, by using an array of adder as well, the engine performs *extraction process* defined by (as the notation in Fig. 5.5)

$$h'_c = IH_{PQRS} = IH_O^S - IH_O^R \tag{6.6}$$

to calculate the histogram of the window h'_c.

6.5.2. Convolution Engine



(a) (b)

Fig. 6.10. Proposed architecture

(a) convolution engine and (b) table selection modules

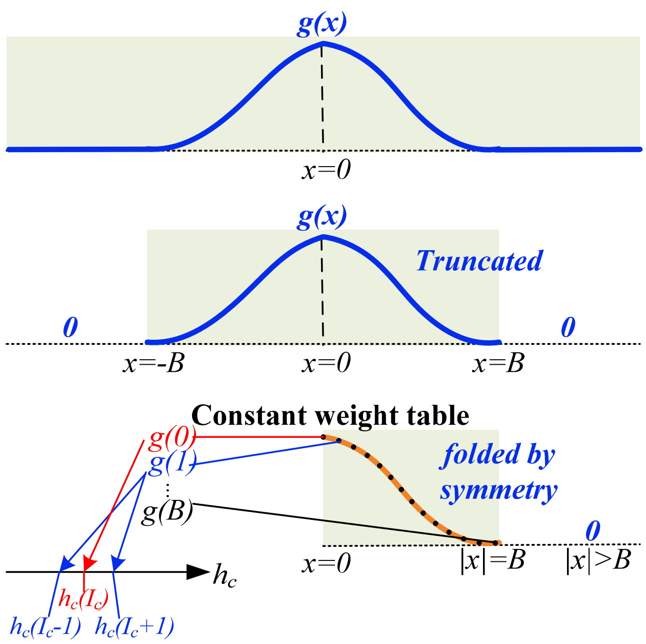


Fig. 6.11. Construction of constant weight table

The convolution engine uses the histograms h_c and h'_c to further compute the pixel result by the kernel calculation and convolution processes in TABLE. 6-1. Its architecture is shown in Fig. 6.10 (a). With the proposed *R-parallelism* method, the convolution process can achieve the throughput of 1 pixel per cycle. Higher throughput can be further attained by the available cut-lines for pipelining in the figure, which can enable working clock be higher.

The *R-parallelism* method brings high throughput but suffers from large size and large number of range table. With 256-level R, for any given target pixel intensity I_c, there should be a corresponding 256-item range table. Therefore, for 256 intensity levels, the amount of all table items should be 256x256. To reduce the range table, we take advantages of the symmetry and truncation property of Gaussian function to decrease its size from 256 to 32. Fig. 6.11 shows a curve shape of Gaussian functions can be truncated by considering required digit. For example, we can truncate values smaller than 2^{-8} for keeping 8-bit decimal digits. Furthermore, by taking advantage of symmetry property of Gaussian function, the negative side and positive side are folded together. Finally, a constant weight table is sampled from the folded curve. Nevertheless, the table size determines the quality so that it should be adjusted to meet the quality demand. In the proposed architecture, we use 32 for example because table of this size is enough to provide sufficient digit precision for usual BF processing ($\sigma_r < 32$).

In addition, to avoid the large number of range table, we share one table by the table selection module as shown in Fig. 6.10 (b), which reduces the number of table to one. Each table selector chooses a weight from the table for its corresponding bin. For example, if I_c is 2, the selector TS0 selects *g(2)* for the first bin (represents for intensity 0) and selector TS1 also selects *g(2)* for the second bin (represents for intensity 4), etc.. Any bin represents for intensity more than 34 is given 0. Then, 64 selected weights and h_c and h'_c are sent into multiplier array and adder trees for computation of the equation of (4.1).

6.5.3. Parameters versus hardware cost

TABLE. 6-2 Parameters and their associated engine components

Parameter	Histogram Calculation Engine	Convolution Engine	Selected Value
Window width $\|S\|$	On-chip memory size Signal bit width	Signal bit width	31
Range kernel σ_r		Constant weight table size	<32
Stripe width w_s	On-chip memory size		60
Bin number N_b	On-chip memory size Operator array length	Operator array length (adder/ multiplier array)	64 (s_r = 4)

There are four main parameters: window width $|S|$, range kernel parameter σ_r, stripe width w_s, and bin number N_b, influencing hardware cost of the proposed histogram calculation engine and convolution engine. The associated engine components of these parameters are shown in TABLE. 6-2. For example, $|S|$, w_s, and N_b are associated to the on-chip memory size of the calculation engine. This can be easily explained with the equation (5.7): the memory cost for integral histogram is determined by these three parameters.

According to TABLE. 6-2, the function block layout of the core architecture doesn't have to be redesigned for different parameter selections because these parameters do not affect its operation flow. (Especially note that the operation flow is invariant even to window size since the processes of integral histogram algorithm are independent of window selection.) Instead, these parameters affect the size or the operator number of their corresponding engine components. Therefore, if an application has variant parameter selection demands, the size and the operator number of equipped engine components in its hardware design must be fulfill the most critical demand. For example, I select 31 as the window size for the proposed architecture since it is larger than the selections of most acceleration algorithms and applications. This makes sure that my architecture is suitable for most applications.

6.5.4. Summary to design components

Overall speaking, the histogram calculation engines and the convolution engine can be serially connected to achieve the throughput of 1 pixel per cycle. Their function block layouts and operation flows are invariant to parameter selection (even to the window size selection). For further high speed demand, more engines can be used to process multiple cascaded pixels simultaneously for higher throughput. The proposed memory reduction methods could be directly extended to support the processing of multiple pixels. In addition, note that for simpler BF, the histogram calculation engine h'_c and its on-chip memory in the core module, and the two input FIFOs in the interface module could be reduced.

6.6. Memory Cost Analysis

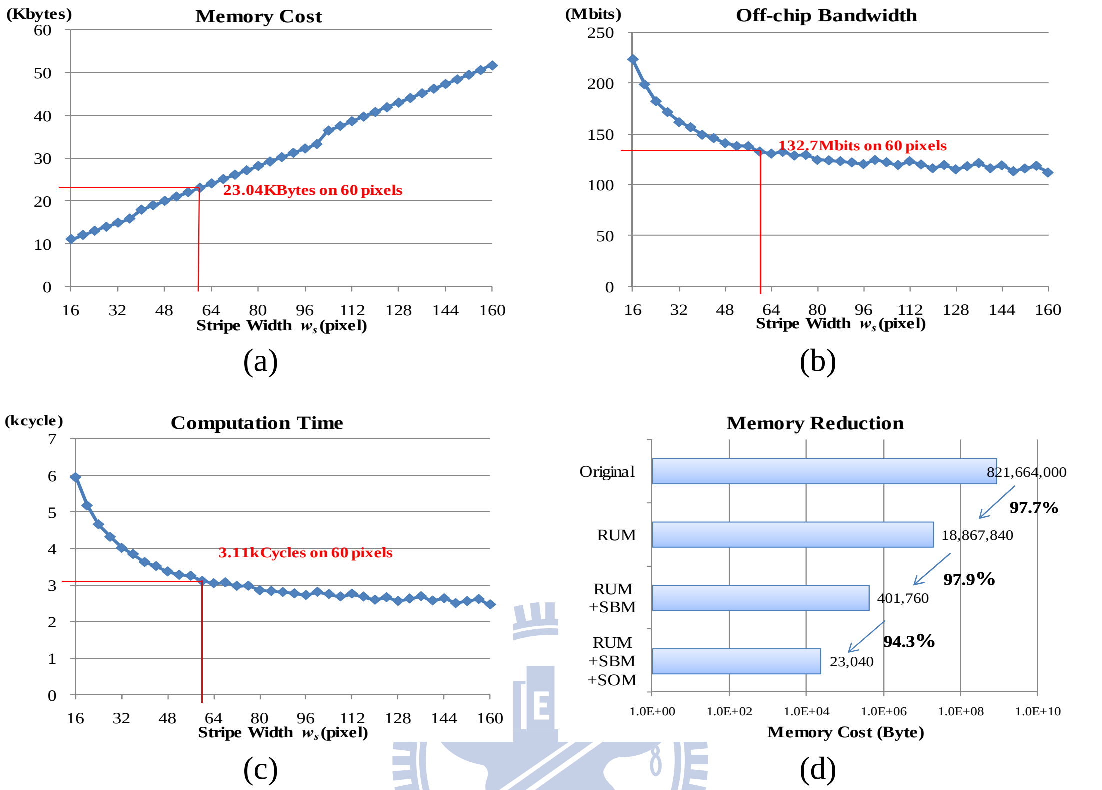


Fig. 6.12. Analysis of Hardware performance and memory reduction (a)-(c) Hardware performance per frame with different w_s; (d) memory reduction with the proposed methods for w_s of 60 (M=1080, N=1920, N_b=64, $|S|$=31).

In this chapter, we analyze the parameter selection in the proposed memory reduction methods. Show the overall memory reduction by three methods combined.

As the combined memory cost in (5.7), there are three parameters, the window size of space kernel width $|S|$, the number of bin N_b, and the stripe width w_s, where the former two are related to application quality, and the last one is related to target performance. Referring to the quality analysis in [34], we select 31 for $|S|$ and 64 for N_b as an example to illustrate how to determine w_s by considering hardware performance.

Fig. 6.12 (a)-(c) estimates the hardware performance of JBF with different w_s for the resolution HD1080p. The memory cost is computed with (5.7) and plotted in Fig. 6.12

(a). The off-chip bandwidth and computation time are calculated by the following equations and plotted in Fig. 6.12 (b) and (c), respectively,

$$M\left(N/w_s\right)\left(|S|+w_s-1\right)\cdot 4pix + M\left(N/w_s\right)w_s\cdot 2pix \tag{6.7}$$

and

$$M\left(N/w_s\right)\left(|S|+w_s-1\right)\cdot 1cycles \tag{6.8}$$

where $M(w_s+|S|-1)$ is the stripe area with extended regions, and N/w_s is the number of stripe in a frame. For the bandwidth, the term with 4 pixels is required by the *integration process*, and the other term with 2 pixels is required by other processes. Since the *integration process* should additionally perform on the extended integral regions as in Fig. 5.2, its bandwidth is more than the other processes'. For the computation time, the proposed architecture takes 1 cycle to produce 1-pixel integral process result.

The selection of w_s is mainly related to the target frame rate. If our target is 30 frames per sec, the constraint of computation cycles is 3.3k; therefore, we could select 60 for w_s, as the example used by this chapter (as shown in TABLE. 6-2), when the working clock is 100 MHz. With the choice, the off-chip bandwidth will be 62.2%, and the memory cost can be reduced to 23 Kbytes, which is 0.003% of the original cost as shown in Fig. 6.12 (d).

6.7. Implementation Result

With above selected parameters, the proposed architecture of JBF has been implemented by Verilog and synthesized under the 90-nm CMOS technology process. TABLE. 6-3 lists the implementation result of the proposed architecture. The hardware design spends less than 300K equivalent gate counts and 23 Kbytes on-chip memory to achieve the throughput of HD1080p 30 frames/sec at the clock rate of 100MHz. Moreover, it can process at 200 MHz by pipelining on the available cut-lines in the convolution engine, and further achieve the throughput of 124 Mpixels per sec for HD1080p at the frame rate of 60 frames per sec.

TABLE. 6-3 Example implementation result of the proposed architecture

Technology		UMC 90nm	
Image Size MxN		1920x1080	
Number of Bin N_b		64	
Window Size $\|S\|$x$\|S\|$		31x31	
Stripe Width w_s		60	
Clock Rate (Hz)		100M	200M
Frame Rate (Frame/Sec.)		30	60
Logic Cost Excluding Memories (Equivalent Gate-Count)	Interface	9,578	9,917
	Histogram Cal.	97,766	148,649
	Convolution	168,333	197,351
	Total	276,178	355,917
On-chip Memory (Byte)		23K	23K

TABLE. 6-4 compares the complexity, memory requirement, and bandwidths between the proposed methods and the original integral histogram in different resolutions. With the proposed memory reduction and architecture design techniques, the complexity can be reduced to 0.15%, and the memory requirement can be reduced to 0.003%-0.02%. In addition, the bandwidth for IH (i.e. on-chip bandwidth) can be reduced to 32%-36%, but the bandwidth for pixels (i.e. off-chip bandwidth) is increased to 20.3-132.7 Mbits. (That is, bandwidth per second is about 1200-8000 Mbit for speed of 60-frame-per-second) Nevertheless, the off-chip bandwidth is affordable by the 64-bit bus processing at 200 MHz. (The maximum affordable bandwidth is 12800 Mbit per second.) Note that the stripe width w_s is specifically selected for the resolution HD1080p. Thus, it can be re-selected by means of the mentioned analysis in Chapter 6.6 to acquire better performance for another resolution.

TABLE. 6-5 compares our proposed hardware design with the previous implementations. Note that this paper is the first VLSI implementation to the best of author's knowledge, and thus only other GPU and CPU approaches are listed for reference comparison. Although the throughput is less than that of Bilateral Grid, the proposed design still achieves best performance because of its significantly reduced memory cost. Comparing to other design, the proposed architecture could efficiently utilize the hardware cost to achieve real-time speed and low memory cost.

TABLE. 6-4 Comparison of hardware cost per frame

	Resol.	Complexity (million operation)	Memory Requirement (Kbyte)	Bandwidth for IH (Mbit)	Bandwidth for pixels (Mbit)
Original	VGA	335.1 (100%)	113,050 (100%)	14,470 (100%)	9.8 (100%)
	HD720p	1,005.5 (100%)	353,894 (100%)	45,299 (100%)	29.5 (100%)
	HD1080p	2,262.3 (100%)	829,440 (100%)	106,108 (100%)	66.4 (100%)
Mem. Reduction	VGA	197.0 (59%)	23 (0.020%)	9,083 (63%)	20.3 (206%)
	HD720p	591.1 (59%)	23 (0.007%)	27,250 (60%)	60.8 (206%)
	HD1080p	1,289.7 (57%)	23 (0.003%)	59,454 (56%)	132.7 (200%)
Mem. Reduction + Archi. Design Tech.	VGA	5.1 (0.15%)	23 (0.020%)	5,191 (36%)	20.3 (206%)
	HD720p	1.5 (0.15%)	23 (0.007%)	15,571 (34%)	60.8 (206%)
	HD1080p	3.3 (0.15%)	23 (0.003%)	33,974 (32%)	132.7 (200%)

Number of bin N_b=64, Window width $|S|$=31, Stripe width w_s=60

VGA=640x480, HD720p=1280x720, HD1080p=1920x1080

TABLE. 6-5 Comparison of different implementations

	Support-Pixel-First				Target-Pixel-First	
	Durand and Dorsey [13]	Chen et al. [36]	Yang et al. [29]	Adams et al. [37]	Porikli [34]	Proposed
Approach	Piecewise-linear Subsampling (s_s=24, s_r=19)	Bilateral Grid (s_s=16, s_r=10)	Piecewise-linear (s_r=32)	Gaussian KD-tree	Integral Histogram (s_r=4)	Integral Histogram (s_r=4)
Implementation	CPU P4 2GHz	GPU Geforce 8800GTX	GPU Geforce 8800GTX	GPU GeForce GTX260	CPU P4 3.2GHz	ASIC
Transistor count (Tech. Process)	55M (130nm) [42]	681M (90nm) [40]	681M (90nm) [40]	1,400M (TSMC 65nm) [41]	55M (130nm) [42]	2.5M (UMC 90nm)
Image Size (Pixel)	10.4M	1.0M	1.0M	10M	1.0M	2.07M
Frame Rate (Frame/sec)	0.16 (high dynamic range)	222	66	0.01-1	3.22	60
Throughput (Pixel/sec)	1.6 M	222M	66M	0.1M-10M	3.22M	124M
Memory (Byte)	-	625K	4M	100M-1G	96M	23K

7. Conclusion

The main contribution of this thesis is to propose efficient hardware architecture with three memory reduction methods for real-time integral histogram based JBF. The three proposed memory reduction methods combined reduces the memory cost to 0.003% compare to the original integral histogram based JBF. The efficient hardware architecture can process large amount of parallel histogram bins simultaneously to achieve 1 pixel per cycle high throughput. The ASIC implementation of the architecture can achieve 124Mpixel (60 frames) per second with HD1080p resolution image under 200MHz clock rate. The chip consumes totally 355 K gate counts and 23KBytes internal memory. The off-chip bandwidth requirement is 132.7Mbits per frame, which is 60% of the total bandwidth of 200 MHz clock rate. For higher throughput, the architecture and memory reduction methods can be directly extended to support the processing of multiple cascade pixels.

Future Work

In the thesis, we have proposed efficient architecture for IH based JBF and its design concept is also suitable for any integral image based applications but limited to those use the box spatial kernel. Nevertheless, Mohamed et al. [43] has shown that a more complicated kernel can be approximated by the linear combination of many basic box kernels. This extends the integral image approach to more complex applications. For the complex application, multiple parallel hardware cores of basic box kernel must be put together and thus the overall interface of data transfer and communication, and the analysis of internal memory and bandwidth requirement must be re-estimated elaborately for the best performance.

On the other hand, the proposed architecture is suitable for gray-level image process. For extended use for multi-color channels, extra software or hardware has to be further designed for blending color channels to gray level. Nevertheless, these methods usually depend on different applications. For example, for producing human visual consistent gray level images, Faust [44] has to includes human vision knowledge and visual aspects to present an enhance conversion.

Reference

[1] C. Tomasi and R. Manduchi, “Bilateral filtering for gray and color images,” *in Proc. of IEEE Int’l Conf. on Computer Vision*, pp. 839-846, 1998.

[2] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, “Joint bilateral upsampling,” *in Proc. of ACM SIGGRAPH*, vol. 26, no. 3, p. 96, 2007.

[3] K.-J. Yoon and I. S. Kweon, “Adaptive support-weight approach for correspondence search,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 650-656, April 2006.

[4] K.-J. Yoon and I. S. Kweon, “Stereo matching with symmetric cost functions,” *in Proc. of IEEE Computer Vision and Pattern Recognition*, pp. 2371-2377, 2006.

[5] O. Stankiewicz, K. Wegner, and M. Wildeboer, “A soft-segmentation matching in depth estimation reference software (DERS) 5.0,” *ISO/IEC JTC1/SC29/WG11*, doc. M17049, Xian, China, October 2009.

[6] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, “A gentle introducing to bilateral filtering and its applications,” *in Proc. of ACM SIGGRAPH tutorial*, 2008.

[7] E.P. Bennett, J. L. Mason ,and L. McMillan, “Multispectral bilateral video fusion,” *IEEE Trans. on Image Processing*, vol. 16, no. 5, pp. 1185-1194, May 2007.

[8] T.R. Jones, F. Durand ,and M. Desbrun, “Non-iterative, feature-preserving mesh smoothing,” *ACM Trans. on Graphics*, vol. 22, no. 3, pp. 943-949, July 2003

[9] S. Fleishman, I. Drori ,and D. Cohen-Or, “Bilateral mesh denoising ,” *ACM Trans.*

on Graphics, vol. 22, no. 3, pp. 950-953, July 2003.

[10] E. Eiseman and F. Durand, "Flash photography enhancement via intrinsic relighting," *ACM Trans. on Graphics*, vol. 23, no. 3, pp. 673-678, August 2004.

[11] G. Petschnigg, M. Agrawala, H. Hoppe, R. Szeliski, M. Cohen, and K. Toyama, "Digital photography with flash and no-flash image pairs," *in Proc. of ACM SIGGRAPH*, vol. 23, no. 3, pp. 664-672, 2004.

[12] B.M. Oh, M. Chen, J. Dorsey, and F. Durand, "Image-based modeling and photo editing," *in Proc. of ACM SIGGRAPH*, pp. 433-442, 2001

[13] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," *in Proc. of ACM Int'l Conf. on Computer Graphics and Interactive Techniques*, pp. 257-266, 2002.

[14] M. Elad, "On the bilateral filter and ways to improve it," *IEEE Trans. on Image Processing*, vol. 11, no. 10, pp. 1141-1151, October 2002.

[15] S.B. Bae, E. Paris, and F. Durand, "Two-scale tone management for photographic look," *ACM Trans. on Graphics*, vol. 25, no. 3, pp. 637-645, 2006.

[16] H. Yu, Y.-L. Zhao, and H. Wang, "Image denoising using trivariate shrinkage filter in the wavelet domain and joint bilateral filter in the spatial domain," *IEEE Trans. Image Process.*, vol. 18, no. 10, pp. 2364-2369, October 2009.

[17] C. Varekamp and B. Barenbrug, "Improved depth propagation for 2D to 3D video conversion using key-frames," *in Proc. of IET European Conf. on Visual Media Production*, pp. 167-173, 2007.

[18] C.-C. Cheng, C.-T. Li, P.-S. Huang, T.-K. Lin, Y.-M. Tsai, and L.-G. Chen, "A block-based 2D-to-3D conversion system with bilateral filter," *in Proc. of IEEE*

Int'l Conf. on Consumer Electronics, pp. 393-394, January 2009.

[19] Q. Yang, R. Yang, J. Davis, and D. Nister, "Spatial-depth super resolution for range images," *in Proc. of IEEE Computer Vision and Pattern Recognition*, pp. 1845-1852, 2007.

[20] D. Chan, H. Buisman, C. Theobalt, and S. Thrun, "A noise-aware filter for real-time depth upsampling," *in Proc. of Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Application – M2SFA2*, 2008.

[21] A. K. Riemens, O. P. Gangwal, B. Barenburg, and R-P. M. Berretty, "Multi-step joint bilateral depth upsampling," *in Proc. of SPIE Visual Communications and Image Processing*, p. 72570M, 2009.

[22] M.-C. Chuang, Y.-N. Liu, T.-H. Chen, and S.-Y. Chien, "Color filter array demonsaicking using joint bilateral filter," *in Proc. of IEEE Int'l Conf. on Multimedia and Expo,* pp. 125-128, 2009.

[23] C. Xiao, Y. Nie, W. Hua, and G. Feng, "Fast multi-scale joint bilateral image and video texture upsampling," *The Visual Computers*, Springer Berlin/Heidelburg, pp.154-157, December 2009.

[24] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister, "High quality real-time stereo using adaptive cost aggregation and dynamic programming," *in Proc. of Int'l Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, pp. 798-805, 2006.

[25] Z. Gu, X, Su, Y. Liu, and Q. Zhang, "Local stereo matching with adaptive support-weight, rank transform, and disparity calibration," *Pattern Recognition Letter*, vol. 29, issue 9, pp. 1230-1235, July 2008.

[26] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister, "Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion

handling," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 31, no. 3, pp.2347-2354, March 2009.

[27] J. Lu, S. Rogmans, G. Lafruit, and F. Catthoor, "Stream-centric stereo matching and view synthesis: a high-speed approach on GPUs," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 19, no. 11, pp. 1598-1611, November 2009.

[28] N. Y.-C. Chang, T.-H. Tsai, P.-H. Hsu, Y.-C. Chen, and T.-S. Chang, "Algorithm and architecture of disparity estimation with mini-census adaptive support weight," *IEEE Trans. Circuits Sys. Video Technol*, vol. 20, no. 6, pp. 792-805, June, 2010.

[29] Q. Yang, K.-H. Tan, and N. Ahuja, "Real-time O(1) bilateral filtering," *in Proc. of IEEE Computer Vision and Pattern Recognition*, pp. 557-564, 2009.

[30] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *International Journal of Computer Vision,* vol. 81, no. 1, pp.24-52, 2006.

[31] T. Q. Pham and L. J. van Vliet, "Separable bilateral filtering for fast video processing," *in Proc. of IEEE Int'l Conf. on Multimedia & Expo*, pp. 454-457, 2005.

[32] T.-S. Huang, "Two-dimensional digital signal processing II: transforms and median filters," *Spring-Verlag*, New York, pp. 209-211, 1981.

[33] B. Weiss, "Fast median and bilateral filtering," *ACM Trans. on Graphics*, vol. 25, no. 3, pp. 519-526, July 2006.

[34] F. Porikli, "Constant time O(1) bilateral filtering," *in Proc. of IEEE Computer Vision and Pattern Recognition*, pp.3895-3902, 2008.

[35] R. Deriche, "Recursively implementing the Gaussian and its derivatives", *in Proc. of International Conference on Image Processing*, pp. 263-267, 1992.

[36] J. Chen, S. Paris, and F. Durand, "Real-time edge-aware image processing with the bilateral grid," *ACM Trans. on Graph*, vol. 26, no. 3, article 103, pp. 1-9, July 2007.

[37] A. Adams, N. Gelfand, J. Dolson, and M. Levoy, "Gaussian KD-trees for fast high dimensional filtering," *ACM Trans on Graph*, vol. 28, no. 3, p. 21, 2009.

[38] M.-H. Ju, and H.-B. Kang, "Constant time stereo matching," *in Proc. of Int'l Machine Vision and Image Processing Conf.*, pp. 13-17, 2009.

[39] Micron Technology, "Synchronous DRAM MT48LC2M32B2-1 Meg x 32 x 4 banks," Available:http://download.micron.com/pdf/datasheets/dram/sdram/128MbSDRAMx32.pdf

[40] A. Wong, "NVIDIA GeForce 8800 GTX/GTS Tech Report," Available: http://www.techarp.com/showarticle.aspx?artno=358&pgno=0

[41] A. L. Shimpi and D. Wilson, "Nvidia's 1.4 billion transistor GPU: GT200 arrives as the GeForce GTX 280 & 260," Available: http://www.anandtech.com/show/2549

[42] "CPU World," Available: http://www.cpu-world.com/index.html

[43] M. Hussein, F. Porikli, and L. Davis, "Kernel integral images: a framework for fast non-uniform filtering," *in Proc. of IEEE Computer Vision and Pattern Recognition*, pp. 1-8, 2008.

[44] L. Neumann, M. Cadik, and A. Memcsics, "An Efficient Perception-based Adaptive Color to Gray Transformation", *in int'l Conf. of on Wireless Communications and Signal Processing,* pp. 1-4, 2009.

Biographical Notes

姓名：許博雄

學歷：

國立交通大學電子所系統組	碩士	(民國 97 年 09 月 ~ 民國 99 年 08 月)
國立交通大學電資學院學士班	學士	(民國 93 年 09 月 ~ 民國 97 年 06 月)
國立台南第一高級中學		(民國 90 年 09 月 ~ 民國 93 年 06 月)

著作：

[1] **Po-Hsiung Hsu**, Yu-Chen Tseng and Tian-Sheuan Chang, "Low Memory Cost Bilateral Filtering Using Stripe-based Sliding Integral Histogram," *in proceeding of IEEE International Symposium on Circuit and System*, pp. 3120-3123, 2010.

得獎事蹟：

- ✓ 98 學年度大專院校積體電路設計競賽標準元件數位電路設計組「佳作」
- ✓ 97 學年度大專院校積體電路設計競賽標準元件數位電路設計組「設計完成獎」