공학박사 학위 논문

메모리 대역폭이 감소된 다중 프레임 레이트 옵티칼 플로우

Multi-Frame Rate Optical Flow with Reduced External Memory Bandwidth

2015년 2월

서울대학교 대학원

전기 · 정보공학부

성 한 수

Multi-Frame Rate Optical Flow with Reduced External Memory Bandwidth

메모리 대역폭이 감소된
다중 프레임 레이트 옵티칼 플로우

지도교수 김수환
이 논문을 공학박사 학위논문으로 제출함
2015년 2월

서울대학교 대학원
전기 · 정보공학부
성 한 수

의 공학박사 학위 논문을 인준함
2015년 2월

위 원 장: 채 수 익 (인)
부위원장: 김 수 환 (인)
위 원: 최 진 영 (인)
위 원: 이 혁 재 (인)
위 원: 이 찬 호 (인)

초 록

최근 high frame rate camera의 비약적인 발전으로 이미 4K 1000FPS camera가 출시되었고 휴대폰에서도 1080P 240FPS를 지원하고 있다. Camera의 Frame rate 증가는 optical flow의 구현에 시사하는 바가 큰데, 그 이유는 frame rate이 올라갈수록 frame 간의 움직임 크기가 줄어들기 때문이다. 그동안 큰 움직임에 대한 부정확한 optical flow 문제를 해결하기 위해서 다양한 알고리즘이 사용되어 왔지만, 이로 인한 computation의 증가 또는 알고리즘 dependency로 인해 늘어난 연산 시간은 real-time operation에 제약으로 작용한다. 하지만 camera의 frame rate이 올라가면 모든 움직임들은 이에 반비례해서 작아지므로, 결국 high frame rate camera는 간단한 알고리즘으로 정확한 optical flow를 얻을 수 있는 길을 열고 있다.

본 논문은 accurate real-time optical flow의 구현을 위해서 multi-frame rate and multi-scale optical flow 알고리즘을 제안한다. High frame rate camera를 이용한 multi-frame rate and multi-scale optical flow 알고리즘은 real-time optical flow의 hardware 구현에 적합하도록 iterative calculation없는 알고리즘이다. Multi-frame rate 알고리즘은 다양한 frame rate의 optical flow를 연산하고 서로간의 연관관계를 이용하여 slow motion 뿐만 아니라 high motion에 관해서도 optical flow 결과를 얻게 함으로써 측정 가능한 움직임을 확장시킨 알고리즘이다. 이 알고리즘은 frame rate 증가에 따른 시스템 연산량 증가를 기존 연구의 O(n)에서 O(log n) 수준으로 감소시킴으로써 system

performance에 의한 제약을 크게 줄인다. Multi scale 알고리즘은 high frame rate system을 위한 full density 지원 알고리즘이다.

또한 본 논문에서는 frame rate의 증가에 따른 external memory access bandwidth 증가 문제를 풀기 위해서 spatial & temporal bandwidth reduction 알고리즘을 제안한다. 이 방법은 기존 LK optical flow알고리즘의 연산 순서를 바꾸고, iterative sub-sampling scheme, temporal Gaussian tail cut 그리고 frame reuse 등 다양한 방식의 알고리즘들을 제안함으로써 high frame rate system의 external memory access bandwidth를 감소시킨다.

마지막으로 Multi-Frame rate and multi-scale optical flow 알고리즘의 Multi-scale 구조의 hardware 의 구현 시 multiplier의 개수를 mxm크기의 윈도우 처리를 위해 m개의 multiplier를 이용해서 convolution방식으로 구현하던 기존의 방법을 윈도우의 크기에 상관없이 2개의 multiplier로 mxm multiplication을 구현하는 방식을 제안한다. 이 방식을 기반으로 multi frame rate과 multi-scale의 hardware architecture를 제안하고 single level LK optical flow의 fpga구현을 통해서 제안한 architecture의 hardware 동작을 검증한다. 이상의 과정들을 통해서 accurate real-time optical flow system을 위한 multi-frame rate and multi-scale optical flow의 알고리즘 제안부터 architecture 검증까지의 연구를 진행한다.

주요어: Optical Flow, Lucas-Kanade, High Frame Rate, Pyramidal LK, Multi-Frame Rate, Multi-Scale

학번: 2010-30222

차 례

그림 목차

표 목 차

제1장 서 론

1.1 연구 배경

Optical Flow는 연속된 이미지에서 나타나는 물체나 무늬패턴의 움직임 변화를 나타내는 벡터를 말한다. 1950년 처음 소개된 이후로 [1] Optical flow 는 Robot Navigation[2][3], Moving Object Detection[4][5], Action Recognition[6][7] 그리고 3D Shape Reconstruction[8] 등, 많은 Computer Vision응용 분야에 적용되고 있다. Optical Flow 를 구하는 방법으로 Gradient Based 알고리즘이 널리 사용되는데, global regularized 방법인 Horn and Schunck(HS)[9] 와 local window based 방법인 Lucas Kanade(LK)[10] 등이 대표적이다. HS 알고리즘은 Global Optimized Solution을 구할 수 있지만, Iterative calculation으로 많은 연산량을 요구하므로 처리 속도가 느린 단점이 있다[11][12]. 반면 LK based 알고리즘은 Local optimized solution을 구하지만 Closed Form Solution을 구하기 때문에 빠른 연산이 가능하다. 특히, real-time operation을 위해 처리 속도 혹은 HW 구현의 적합도를 고려한 optical flow 연산 알고리즘들을 비교한 논문들의 연구 결과 LK 기반의 least square fitting 접근방식이 정확한 결과를 얻으면서도 Cost Efficient한 알고리즘이라고 판단하였다 [13]~[18].

LK Optical Flow알고리즘에서 Fast Movement에 의해서 만들어지는 Large Displacement의 정확도를 높이기 위해서 사용되는 방법으로는 Pyramidal

LK optical flow방법[20]과 High Frame Rate을 이용하는 방법이 있다[22]~[24]. Pyramidal LK 방법은 Downscale된 이미지로부터 optical flow결과를 구하고 그 결과를 next scale optical flow 연산의 initial position에 적용하여 iterative calculation 함으로써 accuracy 와 velocity range를 향상시키는 방법이다. 하지만, Warping 연산을 위해서 Frame buffer 를 embedded DRAM으로 chip 내부에 구현하거나[34] external SRAM을 사용하여 구현한[35] 것과 같이 많은 HW Resource를 요구하는 것과 iterative calculation의 dependency에 의해 늘어나는 연산시간은 Real-Time Operation에 제약사항이 된다.

High frame rate camera를 이용한 기존의 optical flow 알고리즘들은 high frame rate의 optical flow결과를 누적하거나[22], frame 간의 밝기 변화를 누적하는 방법[23]을 이용하여 normal frame rate의 optical flow를 구한다. 이전 연구들에서 매 Frame마다 누적하는 알고리즘을 사용하는 이유는 frame rate의 증가로 움직임의 크기가 너무 작아질 경우 시스템의 연산 가능한 범위를 넘어갈 수 있기 때문이다. 이러한 누적 기반의 알고리즘들은 인접한 frame간의 optical flow 처리를 해야 하므로 frame rate의 증가에 따라서 처리해야 하는 연산량도 비례해서 증가하는 구조를 갖고 있다. 즉, normal frame rate대비 증가하는 frame rate의 비율을 n이라고 할 때, 요구되는 시스템 연산량은 n에 비례하는 O(n) 이다. 이로 인해 high frame rate camera를 사용하더라도 optical flow의 측정 가능한 움직임 범위는 system performance 에 의해서 크게 제약 받는다.

이와 같은 문제들을 정리하면, real-time optical flow를 위해서 global optimum 기반의 알고리즘의 경우 많은 연산량과 dependency에 의한 연산

시간을 요구하는 문제가 있고 Local optimum 기반의 pyramid LK optical flow 방법은 image warping 연산과 iterative한 알고리즘으로 인해 많은 HW Resource와 연산시간을 요구하는 문제가 있다. 또한 high frame rate camera를 이용한 기존의 optical flow 알고리즘도 누적기반의 알고리즘을 사용하기 때문에 frame rate이 증가할수록 O(n)의 연산량 증가를 요구하므로 system performance가 측정 가능한 움직임 범위를 크게 제한하는 문제가 있다. 이와 같이 accurate real-time optical flow의 구현에 대한 기존의 문제들을 해결하기 위해서 본 논문에서는 새로운 Multi-Frame Rate and Multi-Scale Optical Flow 알고리즘을 제안한다. 제안하는 알고리즘의 세부내용은 다음 장에서 설명한다.

1.2 연구 내용

Accurate real-time optical flow의 구현을 주제로 크게 세 가지의 연구내용이 있다. 첫 번째로는 real-time optical flow의 accuracy를 향상시키기 위해서 high frame rate camera를 사용하여 multi-frame rate and multi-scale optical flow를 구하는 알고리즘을 제안하였고, 두 번째로는 frame rate 이 증가할 때 발생하는 frame memory access bandwidth 증가 문제를 풀기 위해 다양한 접근방법의 알고리즘들을 제안하였다. 세 번째로는 high frame rate camera를 이용한 multi frame 및 multi window optical flow 의 real-time hardware 구현을 위한 architecture를 설계한 것이다.

High frame rate camera를 이용한 multi-frame rate and multi-scale optical flow 알고리즘은 real-time optical flow의 hardware 구현에 적합하도록 고정된 연산량을 갖는 iterative calculation없는 알고리즘이다. Multi-frame rate 알고리즘은 다양한 frame rate의 optical flow를 연산하고 서로간의 연관관계를 이용하여 slow motion 뿐만 아니라 high motion에 관해서도 optical flow 결과를 얻게 되어 기존의 Pyramidal LK optical flow의 한계인 측정 가능한 움직임 범위를 확대시킨 알고리즘이다. 또한 Frame rate의 증가에 따른 연산량의 증가를 기존 high frame rate 기반 알고리즘의 O(n)에서 O(log n)수준으로 감소시키고 iterative calculation을 사용하지 않아서 dependency에 의한 연산시간 증가를 제거한다. Multi-scale optical flow는 pyramidal image를 기반으로 각 image level의 muti-rate optical flow결과들을 merge하는 알고리즘이다. 이를 통해서 기존의 high frame rate기반의 알고리즘들이 구현하지 못했던 full density를 지원한다. Multi-scale 알고리즘 역시 iterative calculation을 사용하

지 않는다.

High frame rate sequence를 사용하므로 증가하는 연산량을 O(log n)수준으로 낮추었지만, 그럼에도 높아진 frame rate에 비례한 external memory access bandwidth의 증가는 전체 system의 real-time operation에 제약으로 작용한다. 이 bandwidth 문제를 해결하기 위해서 다양한 spatial and temporal bandwidth reduction 알고리즘들을 제안한다. Spatial 알고리즘은 기존 LK optical flow알고리즘의 연산 순서를 바꾸고, iterative sub-sampling scheme을 사용하는 방법이고, temporal 알고리즘은 temporal Gaussian tail cut 그리고 frame reuse를 사용하는 방법이다.

마지막으로 Multi-Frame rate and multi-scale optical flow의 hardware implementation을 위한 architecture를 제안하였다. LK optical flow 알고리즘은 각각의 픽셀위치에서 x축, y축 그리고 t축 방향의 gradient정보간의 mxm matrix multiplication을 5번해야 한다. 뿐만 아니라 multi-scale 처리로 인해서 늘어나는 operation time을 없애고자 parallel처리를 할 경우는 multiplier는 더 추가되어야 한다. 일반적으로 하나의 mxm matrix multiplication의 구현을 위해서는 m개의 multiplier가 사용되는데, 본 논문에서는 윈도우 크기 m에 독립적인 2개의 multiplier로 matrix multiplication방법을 제안한다. 이 방식을 기반으로 multi frame rate과 multi-scale을 hardware architecture를 제안하고 single level LK optical flow의 fpga구현을 통해서 제안한 architecture의 hardware 동작을 검증한다.

1.3 논문 구성

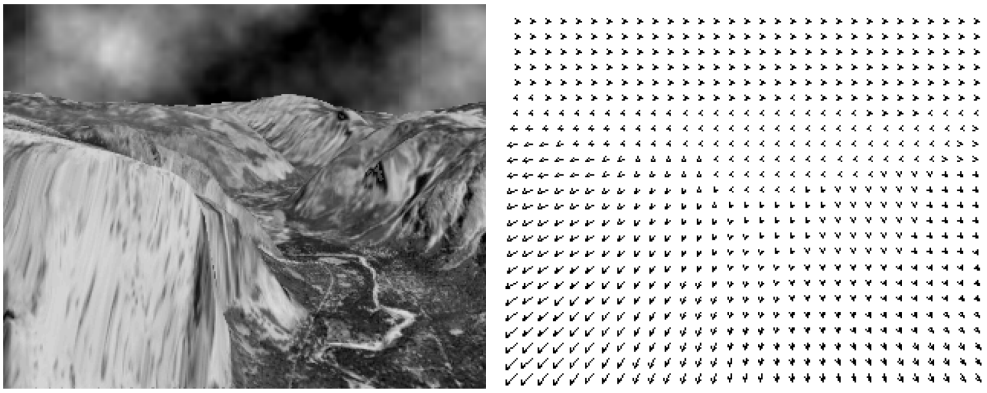
본 연구의 나머지 구성은 다음과 같이 구성된다. 2장에서는 optical flow 기본 알고리즘인 LK optical flow에 대해서 설명하고 Large displacement 문제를 해결하기 위한 기존의 연구를 소개한다. 3장에서는 Multi-Frame Rate Optical Flow Algorithm을 제안하고 high frame rate을 위한 test sequence를 만들어서 그 성능을 검증한다. 4장에서는 처음으로 high frame rate을 이용한 full density를 만족하는 Multi-Frame Rate and Multi-Scale Optical Flow Algorithm을 제안한다. 5장에서는 real time 구현을 위해 Accuracy 감소는 최소로 하면서 external memory access bandwidth를 80%이상 감소시키는 spatial and temporal bandwidth reduction 방법을 제안한다. 마지막으로 6장에서는 본 연구에 대한 결론을 맺는다.

제2장 이전연구

2.1 LK Optical Flow

LK Optical Flow는 매우 정확한 결과를 얻으면서 cost efficient한 알고리즘이기 때문에 많은 연구가 진행되어 왔다. Spatial and Temporal Gaussian smoothing 필터를 사용하면 정확도를 높이면서 Robust한 결과를 얻을 수 있다. [27] 아래 표 2-1를 보면 ground truth 값이 알려져 있는 그림 2-1의 Yosemite test sequence에 대한 LK optical flow의 결과를 보여준다. 그림 2-1의 (a)는 Yosemite test image를 나타내고 (b)는 ground truth optical flow를 나타낸다. Gaussian smoothing 필터의 sigma값의 변화에 따라서 accuracy가 변하는 것을 확인할 수 있는데, Smoothing filter를 적용하지 않을 경우의 accuracy는 가장 나쁘고 sigma 값이 증가할 수록 accuracy가 향상되는 것을 볼 수 있다. 하지만, Gaussian sigma 값이 너무 커지게 되면 이미지가 blur되어서 많은 정보를 잃게 되므로 작은 물체나 거친 texture와 같은 high frequency를 갖는 물체의 움직임을 측정하기 어려워지므로 density가 줄어드는 단점이 있다. 여기서 density는 이미지의 전체 픽셀 수 중 optical flow를 측정 가능한 픽셀 수의 비율을 말한다. 그러므로 Gaussian sigma값은 accuracy와 density의 trade-off 관계에서 선택하게 된다. 그림 2-2는 sigma에 따른 optical flow 결과를 화살표로 표현한 것이다. sigma값이 커질수록 화살표의 방향성은 향상되지만 결과값의 개수는 줄어드는 것을 확인할 수 있다. 일반적으로 normal LK optical flow를 위해서 0.707 ~ 1.225 사이의

Gaussian sigma값을 사용하면 robust하면서 accurate한 결과를 얻을 수 있다. Error측정 방법은 Optical flow 결과와 Ground truth값 간의 AAE (Average Angular Error)와 AEP (Average Endpoint Error)를 이용하여 나타낸다.

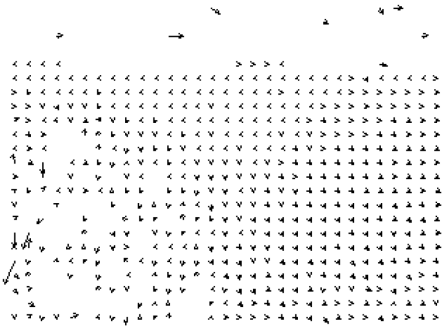


(a) (b)

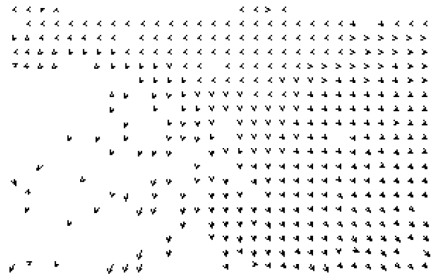
그림 2-1 Yosemite test sequence & ground truth

표 2-1 Sigma에 따른 LK optical flow의 AAE, AEP결과

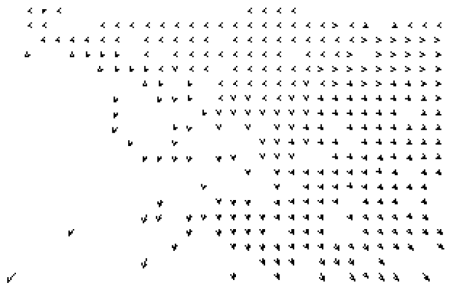
	Sigma					
	0	0.707	1	1.225	1.414	1.581
AAE	26.89	7.69	4.64	4.12	3.87	3.78
AEP	1.11	0.31	0.14	0.12	0.11	0.1
Density	71.27	54.36	43.97	35.7	28.27	22.73



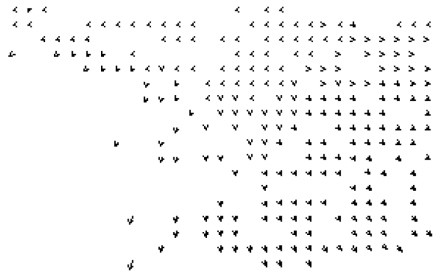
(a) sigma = 0



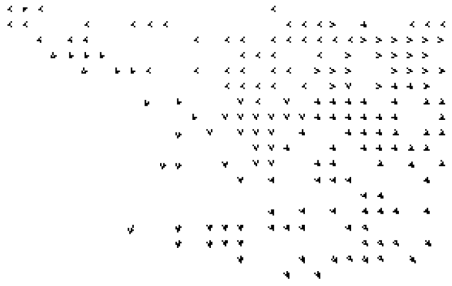
(b) sigma = 0.707



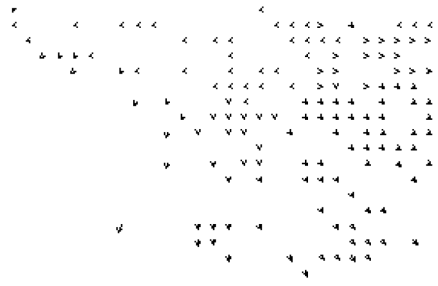
(c) sigma = 1.0



(d) sigma = 1.225



(e) sigma = 1.414



(f) sigma = 1.581

그림 2-2 sigma에 따른 LK optical flow results

2.2 Large Displacement Problem

2.2.1 Pyramidal LK Optical Flow

LK optical flow를 포함한 gradient based의 optical flow알고리즘은 수 픽셀 이내의 작은 움직임에 대해서는 정확한 결과를 얻지만, 큰 움직임에 대해서는 정확도가 매우 떨어지는 문제가 있다. 이러한 문제를 해결하기 위해 다양한 방법들이 제안되었는데, 그 중 하나는 pyramid LK optical flow이다 [36]. Original Image를 [1 4 6 4 1] Gaussian Kernel로 Smoothing한 뒤, 2:1 down scale한다. Down sample된 결과를 가지고 smoothing과 down sampling과정을 반복해서 Pyramid Image를 만든다. 이 과정을 거치면 최 상위 level의 이미지는 가장 작은 이미지가 되고, 이미지 내의 움직임 또한 작은 움직임으로 변환된다. LK optical flow 연산은 최상위 level의 작은 움직임을 측정하면서 시작한다. 움직임 측정은 맨 처음 두 이미지의 동일한 위치에서 측정하고, 다음 움직임 측정은 그 결과에 해당하는 위치에서 다시 측정한다. 이러한 반복측정은 측정결과가 수렴할 때까지 진행하지만, 일반적으로 반복시도는 최대 20회가 넘지 않도록 한다. 최상위 level의 결과가 정해지면 그 결과의 위치에 해당하는 차 상위 level의 위치에서 움직임 측정을 시작한다. 동일한 방법으로 20회가 넘지 않는 범위에서 수렴할 때까지 반복해서 측정한다. 이런 과정을 통해서 전체 level의 움직임 측정이 완료되면 Pyramid LK optical flow 연산이 완료된다. 이 방법의 문제는 pyramidal image를 만들면서 움직임의 크기가 줄어듦과 동시에 object의 크기도 줄어든다는 것이다. 결국 pyramid level을 너무 높게 가져갈 경우 object가 사라질 수도 있기 때문에 object의 크기에 따라서 측정 가능한 움직임 거리가

제한된다.

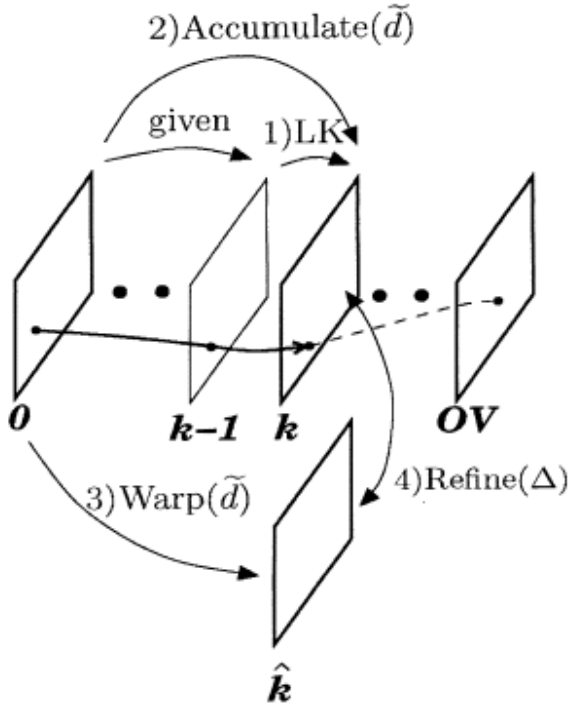
2.2.2 High Frame Rate Optical Flow

큰 움직임에 대한 오류를 해결하기 위해 제안된 방법 중 또 다른 방법은 High Frame Rate의 system을 사용하는 것이다. High frame rate system은 frame sample period가 짧아지므로 normal frame rate에서 큰 움직임이라도 high frame rate에서는 frame 간의 시간이 짧아지므로 작은 움직임으로 바뀌게 된다. 즉, high frame rate system은 역설적으로 슬로우 모션을 찍는 system이 되는 것이다. High frame rate system을 이용하는 기존의 연구 중 하나는 [22]에 의해서 제안된 temporally oversampled Optical flow 방법이고, 다른 하나는 [23]에 의해서 제안된 high frame rate optical flow system에 관한 방법이다. 두 방법 모두 High frame rate sequence를 이용해서 normal frame rate의 optical flow를 측정하는 방법이다.

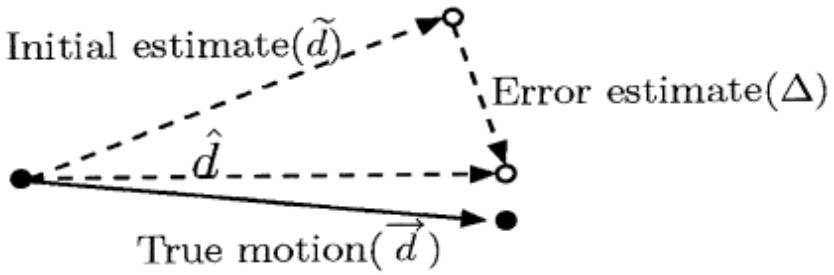
2.2.3 Oversampled Optical Flow System

[22]의 방법은 그림 2-3에 나와 있다. 처음에는 high frame rate의 optical flow를 구하고, 두 번째 frame 부터는 현재 frame의 optical flow 결과에 이전 frame까지의 누적된 optical flow값을 더한다. 이후, 합해진 optical flow결과를 가지고 처음 frame을 warping 하여 새로운 frame 을 만든다. Warping 된 frame과 현재 frame 간의 optical flow를 다시 구해서 이 오차를 지금까지의 누적 optical flow에 보상하여 refine된 optical flow를 구한다. Base 알고리즘으로는 LK optical flow를 이용하였고, 전체 과정을 최종 normal frame rate까지 매 frame마다 accumulate, warping and refinement를 반복한다. [22]의

방법은 High frame의 작은 움직임 결과를 매 frame 마다 accumulate 함으로써 큰 움직임의 측정을 가능하게 한다. 이 경우, 부정확한 값이 누적될 수 있는데, 그 때마다 그 오차의 발산을 막기 위해 누적 결과를 가지고 warping frame을 만들고 refinement의 과정을 거치게 하였다. 즉, [22]는 accumulation and refinement 과정을 통해서 매우 작은 움직임의 오류를 해결하는 시스템을 제안하였다. 하지만, 이 방법은 real-time operation에 적용하기 위해서 제안된 방법은 아니고, real-time에 적용하기 위해서는 몇 가지 문제가 있다. 첫 번째 문제는 warping 과정은 optical flow결과를 이용한 frame 재구성 과정인데, 이 과정에서 external memory access가 많이 발생하게 되며 external memory 의 길고 불규칙적인 latency로 인해 pipeline처리가 어렵게 된다. 이 warping문제는 pyramid LK optical flow의 real-time구현에 있어서도 동일하게 발생한다. Pyramid LK의 warping 문제를 해결하기 위해서 [34]에서는Frame Buffer를 embedded DRAM으로 사용하였고, [35]에서는 external memory를SRAM으로 사용하였다. [22]가 제안한 알고리즘이 Real-time operation에 적합하지 않는 또 다른 문제는 frame rate이 증가하면 처리해야 할 연산량도 비례해서 Linear하게 증가한다는 점이다. 연산량의 급격한 증가는 시스템에서 처리 가능한 최대 frame rate을 제약하게 되고 이는 결국 accuracy의 제약으로 연결된다.



(a)



- New estimate $\hat{d} = \widetilde{d} + \Delta$

(b)

그림 2-3 Lim's temporally oversampled optical flow

2.2.4 High Frame Rate Optical Flow System

[23]는 LK optical flow의 temporal gradient부분을 변형한 variable frame rate optical flow를 제안하였다. High frame rate으로 갈 수록 움직임이 작아지므로 normal frame rate의 It는 high frame rate의(작은 시간단위의) It 변화량의 합으로 표현할 수 있다는 가정을 사용하였다.

$$S_{xx} = \sum_{\Gamma} I_x I_x, \quad S_{xy} = \sum_{\Gamma} I_x I_y, \quad S_{yy} = \sum_{\Gamma} I_y I_y$$
$$S_{xt} = \sum_{\Gamma} I_x I_t, \quad S_{yt} = \sum_{\Gamma} I_y I_t.$$

$$S_{xt}^n(t) = \sum_{\Gamma} I_x \frac{I(x, y, t+n\tau) - I(x, y, t)}{n\tau}$$
$$S_{yt}^n(t) = \sum_{\Gamma} I_y \frac{I(x, y, t+n\tau) - I(x, y, t)}{n\tau}.$$

$$\begin{aligned} S_{xt}^n(t) &\approx \frac{1}{n}\left(\sum_{\Gamma} I_x \frac{I(x,y,t+n\tau) - I(x,y,t+(n-1)\tau)}{\tau}\right. \\ &\quad \left. + \cdots + \sum_{\Gamma} I_x \frac{I(x,y,t+\tau) - I(x,y,t)}{\tau}\right) \\ &= \frac{1}{n}\sum_{i=0}^{n-1} S_{xt}^1(t+i) \equiv \tilde{S}_{xt}^n(t) \end{aligned}$$

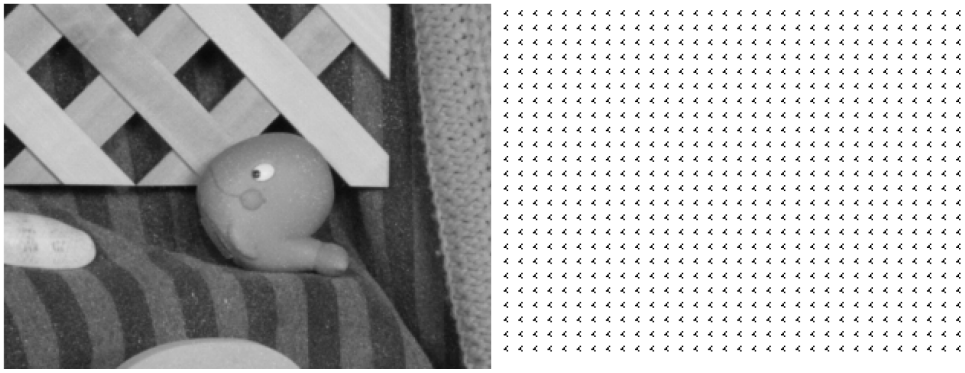
결국 It를 구하기 위해서는 pixel의 temporal 방향으로의 변화량을 누적시켜서 평균을 내고 그 누적된 변화량이 일정 크기를 넘어가는 순간에 optical flow를 구한다. [23]의 방법은 high frame rate에 의한 작은 움직임의 문제를 temporal 픽셀값의 변화가 일정한 기준 이상일 때까지 여러frame의 픽셀값을 누적시키는 방법을 사용함으로써 해결하였다. [23]의 논문에서 Temporal 픽셀값의 변화량이 정확한 optical flow연산에 중요한 역할을 하

므로 오차의 누적을 방지하기 위해서 10 bit의 pixel bitwidth를 사용하고, 32x32 픽셀의 블록단위로optical flow를 연산하였다. 픽셀 단위의32x32 product sum을 구하는 과정은 Hardware로 구현하였고, 그 결과를 이용한 block 단위의 연산은 PC에서 소프트웨어로 구현되었다. 또한 사용하는 high frame rate image는 2 frame을 이용하였다. [23]의 논문을 픽셀단위의 real-time optical flow hardware system에서 사용하기에는 몇 가지 문제가 있다. 그 중 첫 번째 문제는 temporal pixel의 변화량을 알기 위해서는 이전 frame까지 누적된 값을 저장해야 하는데 있다. 논문에서는 IxIt 와 IyIt의 product sum을 사용하였는데, 이 경우 저장하는 값의 bitwidth는 product sum과 누적하는 최대 frame 수에 의해서 증가해야 하므로 저장 공간 및 읽어와야 하는 데이터도 크게 늘어나야 한다. 두 번째 문제는 [22]의 경우와 마찬가지로 모든 frame에 대해서 optical flow를 처리하기 때문에 frame rate이 증가할 수록 연산량 및 external memory access bandwidth도 linear하게 증가한다는 점이다. 마지막으로 세 번째 문제는 variable frame rate의 결과는 temporal pixel의 변화량이 충분한 시점에 만들어 지게 되므로 모든 optical flow의 연산결과는 모두 다른 시점을 기준으로 측정될 수 있다는 점이다. 즉, It관련된 값은 누적되지만, Ix나 Iy 값은 다른 시점의 Ix와 Iy를 기준으로 연산 되므로 accuracy의 보정이 필요하다. 그러므로 최종적으로 normal frame rate의 처리를 위해서 특정시점으로 optical flow의 보정 및 통합 등의 후처리(post processing) 과정이 필요하므로 연산량이 증가하게 된다.

2.3 Problems in High Frame Rate System

2.3.1 Test Sequence for High Frame Rate System

움직임에 대한 LK optical flow의 결과를 얻고 정확도를 측정하기 위해서는 ground truth정보를 알고 있는 test sequence가 필요하다. 하지만, High frame rate을 위한 test sequence는 없기 때문에 기존의 연구들도 검증을 위해서 임의의 사진을 가지고 test sequence를 만들거나 움직인 거리를 정밀하게 조정한 물체를 High Frame Rate Camera로 찍어서 사용하였다. 이 방법들의 문제는 다음 연구를 진행하는 사람이 기존연구와 비교하기 위해 공유할 test sequence가 없다는 것이다. 이런 이유로 본 논문에서는 기존에 널리 사용되는 Middlebury test sequence를 이용해서 high frame rate을 위한 test sequence를 만든다. 다양한 속도의 움직임을 구하기 위해서 원하는 속도의 크기만큼 전체 이미지를 shift하고 이 shift값을 ground truth로 사용한다. 그림 2-4는 test sequence 이미지와 다양한 속도의 ground truth를 보여준다. (a)는 Middlebury test sequence 중 Rubberwhale이다. Shift를 하기 위해서 전체 크기 중 320x240크기의 일부를 사용한다. 그림 (b) 부터 (d)까지는 normal frame rate 기준으로 다양한 속도에 대한 ground truth값을 보여준다. (b)는 1 pixel/frame, (c)는 8 pixel/frame 그리고 (d)는 32 pixel/frame의 속도에 대한 ground truth를 보여준다.



(a) Rubberwhale (b) 1 pixel/frame

(c) 8 pixel/frame (d) 32 pixel/frame

그림 2-4 test sequence for high frame rate – rubberwhare of middlebury

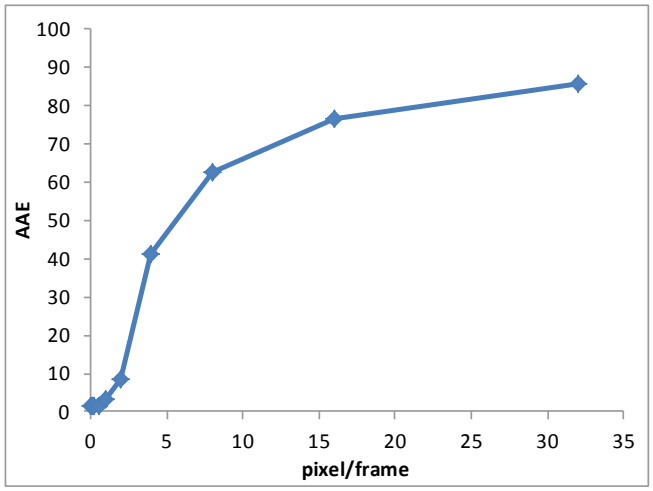
2.3.2 Saturated Accuracy for High frame rate

그림 2-5는 다양한 속도의 test sequence에 대한 LK optical flow의 측정 결과를 나타낸다. LK optical flow system에서 사용한 gradient filter는 spatial의 경우 [1 -8 0 8 -1]/12를 사용하였고 temporal의 경우 [-1 0 1]/2를 사용하였다. 이후 명시하지 않은 경우는 이 값을 사용한 결과이다. Gaussian smoothing filter의 sigma값은 0.707이고 사용하였다. (a)와 (b)는 AAE결과를 나타내고 (c)와 (d)는 AEP결과를 나타낸다. (a)와 (c)는 전체 움직임 크기에 대한 결

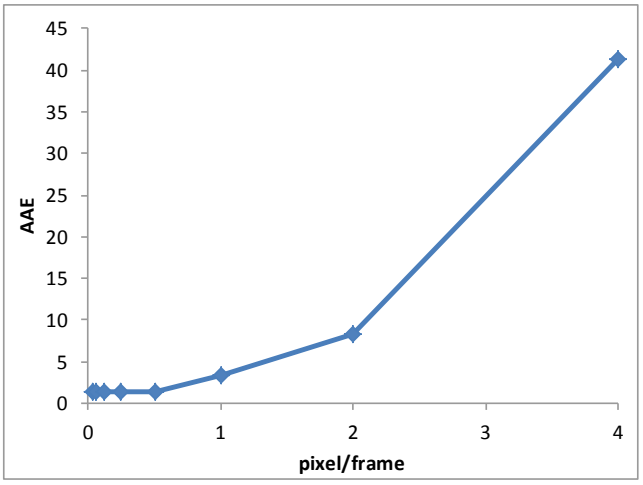
과를 보여주고 (b)와 (d)는 작은 움직임 크기에 대한 결과를 보여준다. 가로축은 이미지가 frame당 움직이는 픽셀 수를 나타낸다. (a)와 (c)의 결과를 보면 움직임 크기가 작을 때는 Error가 작은 수준을 유지하지만, 움직임 크기가 커질수록 Error 가 증가하는 것을 확인할 수 있다. 이 결과는 앞에서 설명한 대로 low frame rate 보다는 high frame rate으로 갈 수록 움직임 크기가 작아지게 되어 정확도가 향상되는 결과를 설명한다. 하지만, (b)와 (d)의 결과를 보면, pixel 의 움직임이 작아질 수록 Error는 감소하지만, 어느 순간을 지나면서 Error의 감소율이 pixel움직임이 줄어드는 비율보다 줄어든다. 즉, 움직임이 작아질 수록 LK optical flow system은 그것에 비례해서 정확한 결과를 측정하지 못하고 saturate되는 것을 확인할 수 있다.

작은 움직임에 대해서 측정 accuracy가 saturate되는 원인은 크게 환경적인 요인과 구조적인 요인으로 볼 수 있다. 환경적인 요인으로는 Gaussian Noise이다. 즉, 주변의 빛과 움직임, 카메라의 위치 등 모든 조건이 일정한 조건에서 물체를 찍는다고 하더라도 매 frame마다 Gaussian Noise가 발생하여 동일한 위치의 pixel 정보도 일정하지 않게 된다. 이 Gaussian Noise level보다 움직임에 의한 밝기의 변화량이 작다면 optical flow system에 의한 측정이 불가능 해 진다. 또 다른 요인인 구조적 요인은, fixed pixel bitwidth와 window 크기의 summation 과정에 있다. 움직임 변화는 pixel의 밝기 변화로 나타나는데, 이 정보는 8 bit인 256 level로 저장된다. 움직임으로 인한 pixel밝기 변화가 1/256 level보다 작은 경우 측정이 불가능 해 진다. 또한 LK optical flow는 gradient의 변화를 이용하여 일정한 크기의 window 크기의 gradient값을 곱하고 더한다. 이 과정에서 average하는 효과

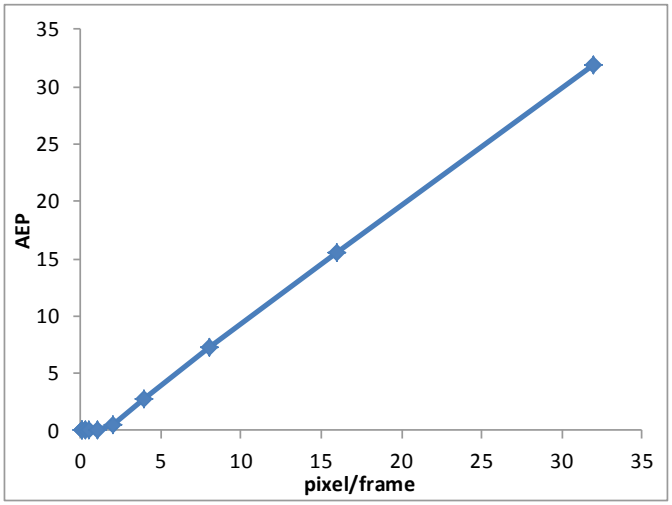
가 발생하기 때문에 작은 gradient 의 summation이후의 변화는 더욱 작아지게 된다. 이런 이유로 움직임이 작아질 수록을 정확한 값을 측정하기가 어려워진다.



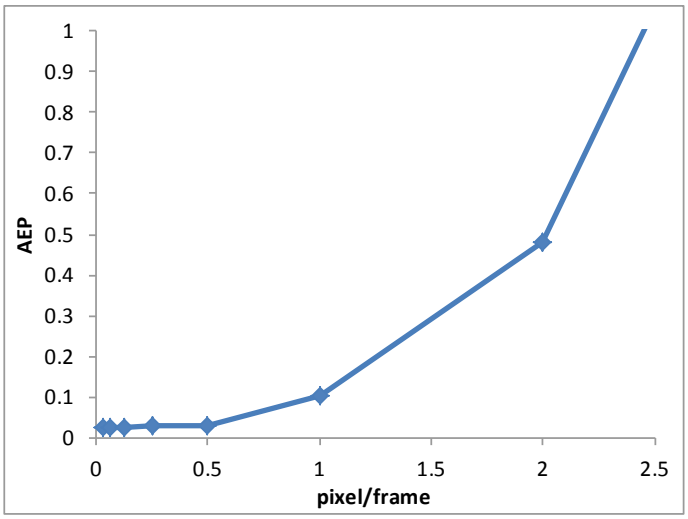
(a) AAE of LK optical flow



(b) Small displacement AAE



(c) AEP of LK optical flow

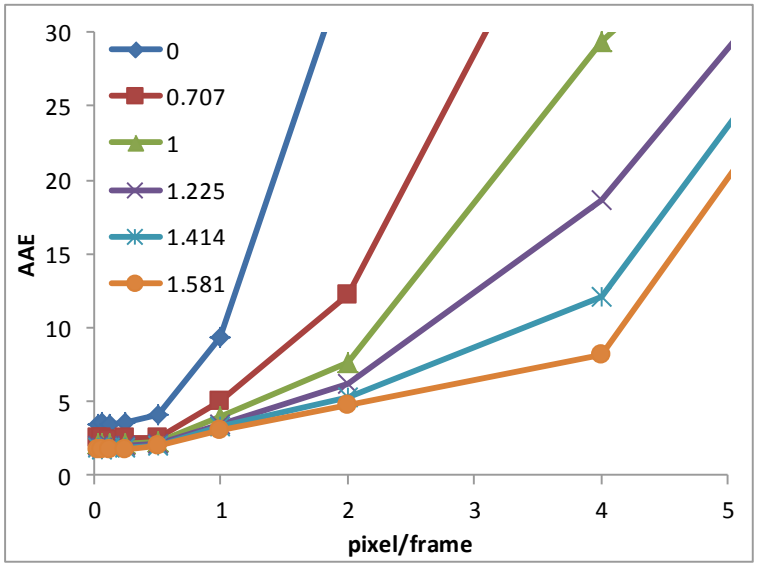


(d) Small displacement AEP

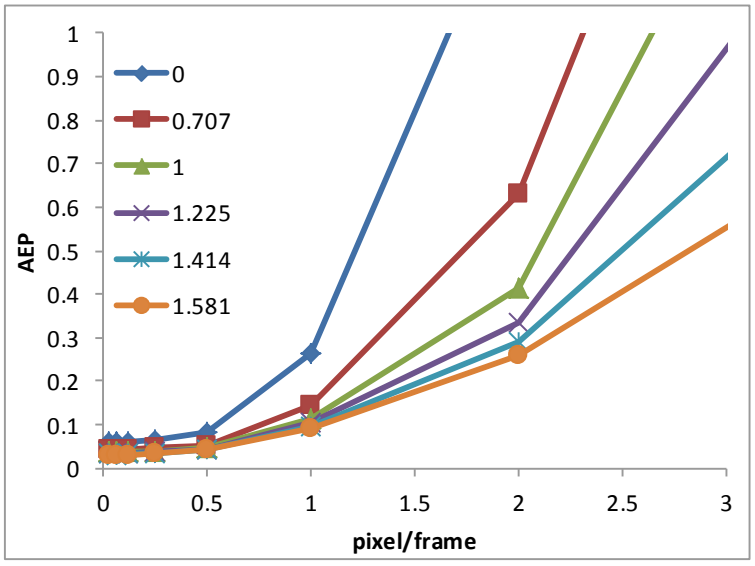
그림 2-5 다양한 속도에 대한 LK optical flow

2.3.3 Accurate displacement for LK optical flow

그림 2-6의 (a), (b)의 결과들은 LK optical flow 처리를 위해 Gaussian spatial and temporal smoothing filter의 sigma값을 각각 0.0, 0.707, 1, 1.225, 1.414, 1.581로 설정하였을 때의 AAE와 AEP 결과이다. Gaussian sigma에 따라서 density에 영향을 끼치므로 동일한 조건의 결과를 얻기 위해 Eigenvalue threshold는 모두 0으로 고정하였다. Sigma값이 커질 수록 큰 움직임의 측정 error가 줄어드는 것을 확인할 수 있는데, 이 결과는 앞 그림의 Yosemite test sequence 결과에 의해서도 확인된 내용으로 sigma값이 커질 수록 accuracy는 증가하지만 물체의 경계가 흐려지기 때문에 eigenvalue threshold를 사용할 경우 density는 감소하는 단점이 있다. Gaussian Sigma값이 0인 경우 움직임의 크기가 0.5를 넘어서면서 Error가 급격하게 증가하는 것을 확인할 수 있다. 반면, Gaussian smoothing filter를 사용하는 경우, 1 픽셀 움직임 까지는 Gaussian sigma를 키우더라도 Error의 개선 정도가 큰 차이를 보이지 않지만, 움직임의 크기가 커질 수록 큰 Gaussian sigma의 error개선 효과를 확인할 수 있다.



(a) AAE with various Gaussian Sigma



(b) AEP with various Gaussian Sigma

그림 2-6 AAE and AEP with Various Spatial & Temporal Gaussian sigma

2.3.1 Accurate frame rate of High frame rate system

일정한 속도로 움직이는 물체를 다양한 frame rate으로 처리하는 것은 다양한 속도의 물체를 일정한 frame rate에서 측정하는 것과 유사한 상황의 결과를 얻는다. 그림 2-8은 normal frame rate인 30 frame rate에서 32 pixel/frame의 속도로 움직이는 물체에 대해서 다양한 frame rate system으로 optical flow AAE와 AEP 측정 결과를 보여준다. 그림 2-8의 (a)와 (b)의 결과를 보면 high frame rate으로 갈 수록 error는 감소하는데 이 이유는 High frame rate으로 갈 수록 frame period는 짧아 지므로 측정되는 움직임의 크기가 줄어들기 때문이다. 그림 (c) 와 (d)는 (a)와 (b)에서 구한 각 frame rate의 결과를 normal frame rate으로 변환한 결과이다. Normal frame rate으로의 변환은 측정 frame rate과 normal frame rate간의 frame rate 비율만큼 optical flow 결과를 확장시키는 방법을 사용하고 normal frame rate의 ground truth를 기준으로 변환된 optical flow결과의 Error를 측정한다. (c)의 AAE의 결과는 normal frame 으로 변환하더라도 error의 변화가 없지만, (d)의 AEP 결과는 High frame rate으로 갈 수록 (b)보다 endpoint error 가 크게 증가한 것을 확인할 수 있다. 그림 2-7처럼 optical flow결과가 normal frame rate으로 변환될 때, angular error는 각도이므로 변함이 없지만, endpoint error는 확장되는 비율만큼 증가하게 된다. 그런데, normal frame rate에서 high frame rate으로 옮겨가는 중간에 AEP가 감소하는 이유는 frame rate 비율로 인한 Error증가량보다 작아진 움직임으로 인한 측정 error의 감소폭이 더 컸기 때문이다. 하지만 앞장의 다양한 속도의 움직임에 대한 optical flow결과인 그림 2-5에서 확인할 수 있었던, 작은 움직임에 대한 accuracy saturation이

발생하기 때문에 high frame rate으로 갈 수록 측정 error의 감소폭이 줄어들게 되어normal frame rate으로 변환된 AEP가 증가하게 된다.

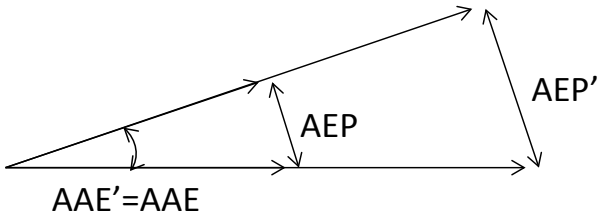
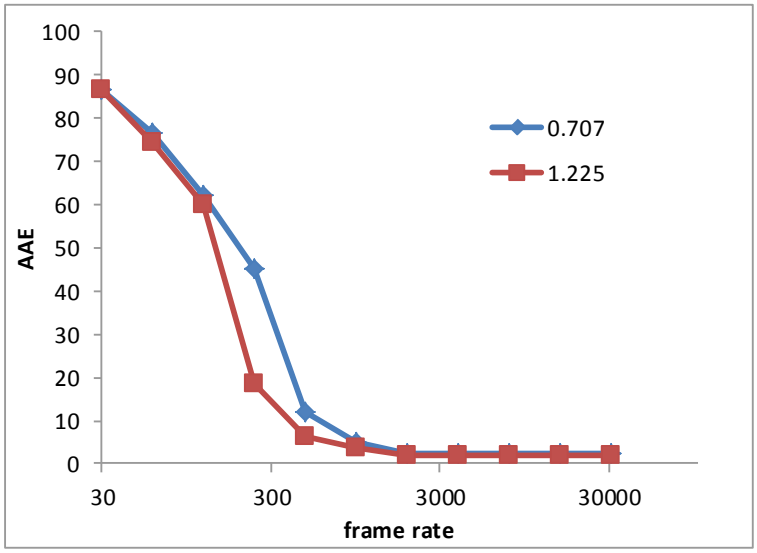
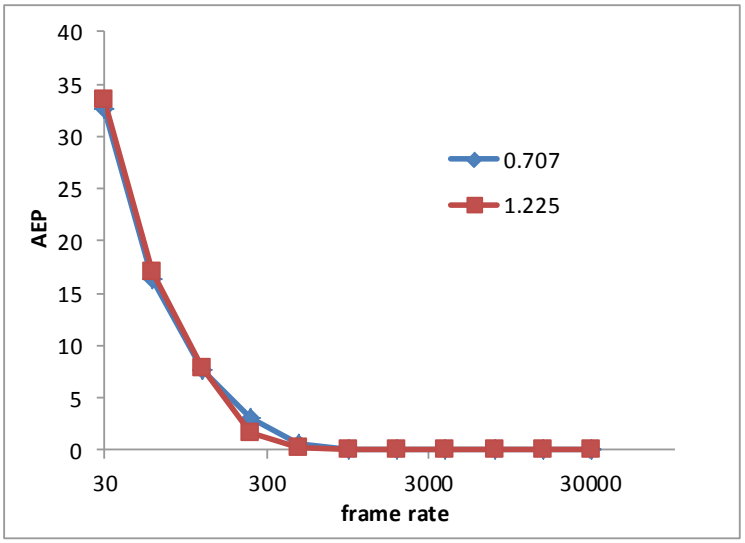


그림 2-7 AAE AEP의 확장

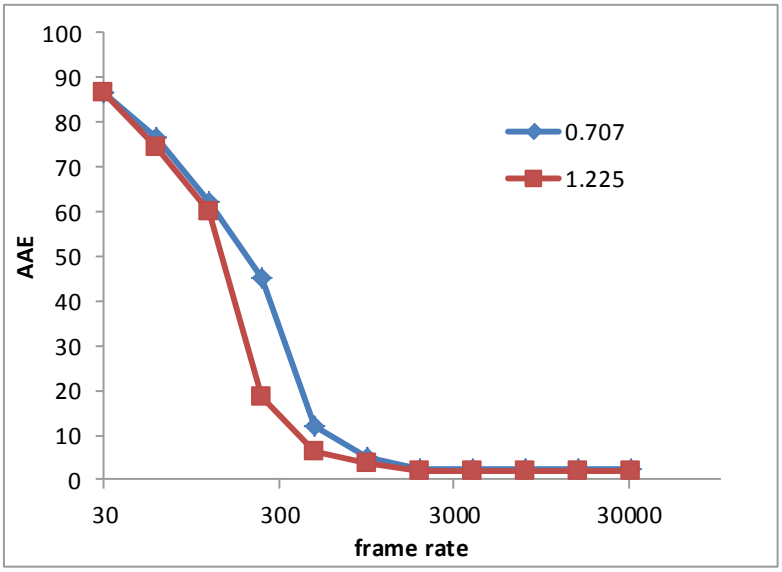
이 결과를 통해서 확인할 수 있는 것은, high frame rate으로 움직이는 object를 찍어서 움직임의 크기를 매우 작게 만든다고 해도 그 움직임이 측정가능 한 optical flow system의 범위를 벗어나게 된다면 accuracy saturation이 발생하므로 high frame rate에서 측정한 optical flow result를 normal frame rate으로 변환 했을 때, 오히려 더 큰 Error를 보일 수 있다는 것이다. 그러므로 high frame rate을 이용한 가장 정확한 결과를 얻기 위해서는 normal frame rate으로 변환한 AAE와 AEP가 동시에 가장 작은 값을 가질 수 있는 frame rate을 선택해야 한다는 것이다. 결론적으로 가장 정확한 optical flow결과는 frame rate이 클수록 정확한 값을 얻는 게 아니라 어떤 속도의 움직임을 가장 잘 측정할 수 있는 특정 frame rate에서 가장 정확한 결과를 얻는 다는 것을 알 수 있다. 자세한 측정값은 표 2-2에 나와 있다. 다음 장에서는 가장 accurate한 frame rate을 선택하고 full density를 지원하는 multi-frame rate and multi-scale optical flow 알고리즘을 제안한다.



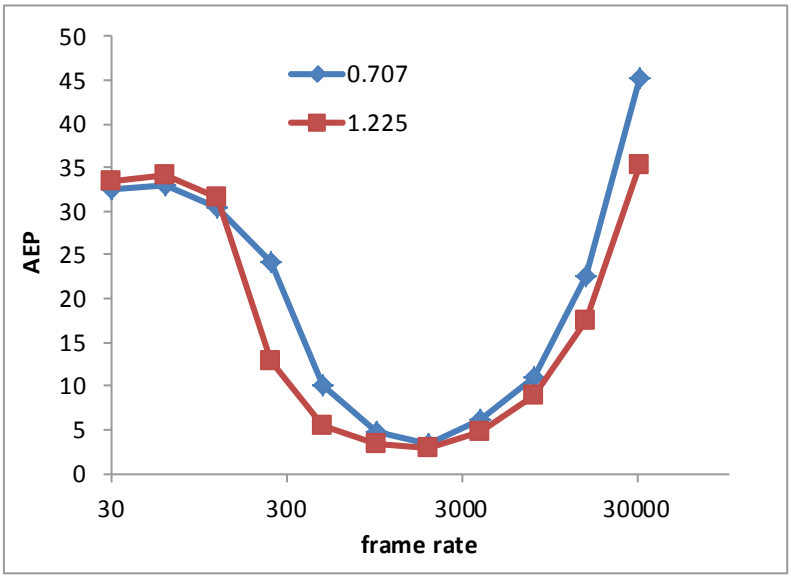
(a) AAE for various frame rate



(b) AEP for various frame rate



(c) AAE at normal frame rate



(d) AEP at normal frame rate

그림 2-8 high frame rate system에서 일정한 속도의 움직임에 대한 결과

표 2-2 AAE and AEP at normal frame rate for various Gaussian Sigma

		Gaussian Sigma					
frame rate	px/frame	0	0.707	1	1.225	1.414	1.581
30720	0.03125	3.428434	2.516991	2.170343	1.967959	1.832381	1.71882
15360	0.0625	3.538098	2.500852	2.144426	1.947148	1.813111	1.701344
7680	0.125	3.445272	2.427583	2.13411	1.985267	1.876628	1.776415
3840	0.25	3.57092	2.583637	2.207592	1.986117	1.849565	1.752039
1920	0.5	4.082487	2.573229	2.267561	2.13721	2.056715	1.990603
960	1	9.385423	5.043738	3.962091	3.525231	3.268469	3.067052
480	2	34.79738	12.18182	7.624361	6.189911	5.336799	4.768163
240	4	63.24015	45.04113	29.3541	18.62513	12.04149	8.192461
120	8	72.85784	62.12035	60.79939	59.77991	58.34522	56.14743
60	16	80.93259	76.57577	75.49217	74.32736	73.17601	72.30876
30	32	87.44534	86.80658	86.30254	86.45773	86.92052	87.31424

(a) AAE

		Gaussian Sigma					
frame rate	px/frame	0	0.707	1	1.225	1.414	1.581
30720	0.03125	61.56288	45.1625	38.93248	35.29933	32.8663	30.82957
15360	0.0625	31.79008	22.46707	19.2727	17.50528	16.30822	15.30726
7680	0.125	15.59194	10.9975	9.6768	9.003776	8.513792	8.064512
3840	0.25	8.347904	6.073344	5.190656	4.67712	4.366592	4.150784
1920	0.5	5.27008	3.380928	2.998592	2.842432	2.753536	2.685952
960	1	8.439712	4.681248	3.693504	3.299968	3.072768	2.898464
480	2	22.00736	10.07461	6.607152	5.374912	4.631456	4.140336
240	4	29.67913	24.09035	17.93384	12.76729	9.1226	6.789952
120	8	31.18974	30.41366	31.30382	31.63812	31.46549	30.7495
60	16	31.87864	32.87818	33.95052	34.18135	34.15149	34.134
30	32	32.02144	32.56673	33.09184	33.43705	33.56678	33.56487

(b) AEP

제3장 Multi-Frame Rate Optical Flow

이전의 두 연구 [22][23]는 high frame rate의 frame 간의 매우 작은 움직임 또는 밝기의 변화량을 누적하여 optical flow system이 처리 가능한 움직임 범위를 이용해서 결과를 얻는 방법을 사용하였다. 이러한 방법들은 Pyramid LK에서 반복해서 downscale한 이미지를 만들고 가장 움직임이 작은 최상위 level의 가장 작은 이미지로부터 optical flow연산을 시작하는 것과 동일한 원리이다. High frame rate system은 normal frame rate system보다 단위 시간당 입력되는 frame 수가 증가하기 때문에 frame rate이 올라갈수록 system의 처리 능력은 비례해서 증가해야 한다. 이는 System비용의 증가를 초래하게 된다. 그럼에도 불구하고 기존의 high frame rate 알고리즘들이 매 frame 반복해서 처리해야 하는 알고리즘을 사용하는 이유는 기존 알고리즘들은 frame access bandwidth를 줄이기 위해서 2장을 사용하는 [-1 1]의 gradient kernel을 사용하였는데, 이 경우 frame 을 skip하게 되면 움직임의 크기가 커지게 되어 측정 오류가 급격하게 증가하기 때문이다.

본 논문에서는 각 픽셀 별로 신뢰할 수 있으며 accurate한 결과를 얻는 최적의 frame rate을 선택하는 방법을 사용하여 high frame rate을 이용한 최적의 optical flow 결과를 얻는 방법을 제안한다. 또한 이를 위해 사용하는 frame rate도 기존의 방법처럼 high frame rate의 모든 frame에 대해서 처리하는 것이 아니고 기존 방법대비 기하급수적으로 감소하는(O(n) → O(log n)) 방법을 사용하면서도 accuracy를 잃지 않는 방법을 제안한다.

3.1 Ideal and Real Optical Flow System

그림 3-1은 다양한 Optical flow system의 특성을 보여준다. 실선 구간은 확률적으로 accurate 한 optical flow를 구할 수 있는 구간이고 점선구간은 그렇지 못한 구간을 의미한다. (a) 시스템은 작은 움직임에서부터 큰 움직임까지 모든 범위의 움직임에 대해서 accurate한 optical flow result를 얻는 시스템을 의미한다. (b) 시스템은 일정한 크기 이하의 움직임에 대해서만 정확한 결과를 얻는 시스템이다. 이런 경우, frame rate을 높일 수록 큰 움직임들을 정확하게 측정 가능한 범위에서 처리할 수 있으므로 frame rate이 높을 수록 유리하게 된다. (c) 시스템은 일정한 크기 이상의 움직임에 대해서만 정확한 결과를 얻는 시스템이다. 즉, frame rate이 낮아질 수록 움직임이 커지므로 더 정확한 결과를 얻는다. (a), (b), (c)와 같이 무한대로 작은 움직임이나 무한대로 큰 움직임에 대해서 정확한 결과를 얻는 것은 현실적으로 존재하기 어려운 ideal 한 시스템을 의미한다. 즉, (d)와 같이 일정한 범위의 움직임을 정확하게 측정하는 것이 현실적인 optical flow system의 특성이다. (e)는 optical flow system 자체의 한계로 모든 범위의 움직임에 대해서 정확한 결과를 얻지 못하는 경우를 말한다. 앞장의 실험결과를 통해서 볼 때, LK optical flow system은 (d)의 특성처럼 일정한 크기 이하 또는 크기 이상의 경우 정확도가 떨어지는 시스템인 것을 확인할 수 있다.

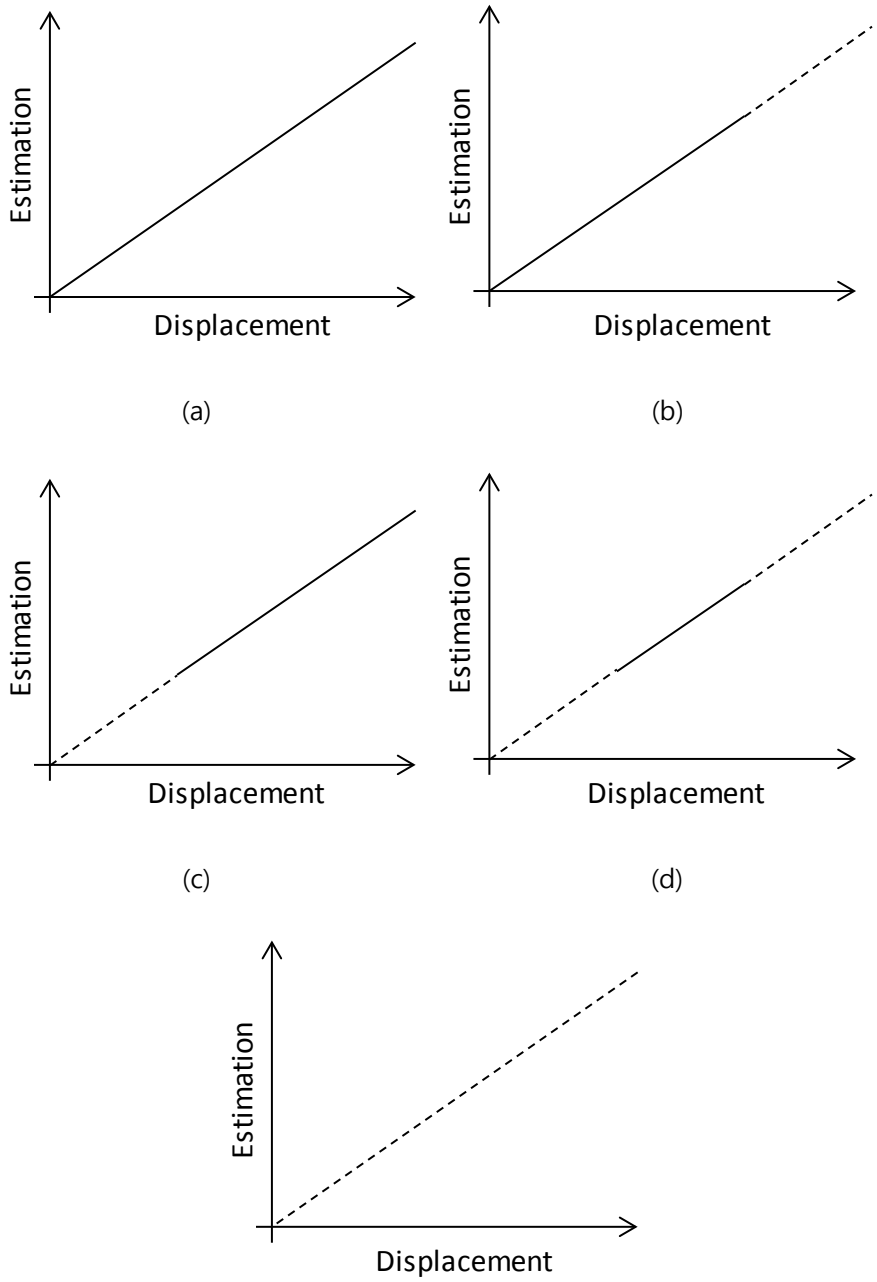


그림 3-1 target optical flow system

3.2 Multi-Frame Rate Optical Flow

LK system의 accurate한 연산의 구간은 표 2-2에서 normal frame rate으로 변환된 결과인 (b)를 보면 Gaussian Sigma값이 0인 경우 0.5 픽셀 움직임에 대해서만 가장 AEP가 작으나, sigma값이 증가할수록 AEP값이 작아짐과 동시에 움직임 구간도 증가하여 0.5 ~ 1.0 픽셀 구간을 형성한다. 그러므로 Gaussian Sigma최대한 이 구간을 이용하여 처리하는 것이 LK optical flow system을 이용할 때 오류가 적은 결과를 얻을 수 있는 확률을 높이는 것이다. 여기서 최대 움직임인 1.0 픽셀은 최소 움직임인 0.5 픽셀의 2배에 해당하는 움직임의 범위이다. 본 논문에서는 이 범위를 이용하여 frame rate을 효과적으로 이용하는 방법을 제안한다. 그림 3-2는 optical flow 의 accurate구간이 2배인 경우 이 구간에 포함되기 위해 처리하는 모든 움직임의 크기를 동일한 2배 관계를 유지하도록 frame rate을 선택한 결과를 보여준다. 예를 들어, 30 fps에 해당하는 frame period는 60 fps의 frame period 보다 2배에 해당하는 시간을 갖는다. 이와 같은 방식으로 60 fps는 120 fps 보다 2배의 frame period를 갖는다. 물체가 constant 속도로 움직인다고 가정한다면, 2배의 시간 동안 움직인 거리는 2배가 된다. 이 관계를 확장하면 30, 60, 120, 240, 480, 960, 1920, 3840 … fps 로 frame 을 2배 관계로 올릴 수 있으므로 frame rate이 2배로 올라갈 때마다 ½배로 줄어든 움직임을 갖는 frame들을 얻을 수 있다. 즉, 모든 속도의 움직임을 1/2씩 줄여나간다면 모든 속도의 움직임은 최소한 한번은 0.5 ~ 1.0 픽셀의 움직임 구간을 갖게 되고 그 때의 frame rate을 선택할 때, normal rate으로 변환된 가장 정확한 optical flow값을 얻게 된다.

연산해야 하는 frame 수는 모든 high frame마다 처리하던 기존의 알고리즘이 O(n)의 frame 처리량을 갖는 반면, 제안하는 배수관계의 frame rate을 이용하는 방법은, O(log n)의 frame 처리량을 갖기 때문에 frame 이 증가할수록 처리해야 하는 frame수가 기존 알고리즘 보다 기하급수적으로 줄어들게 된다.

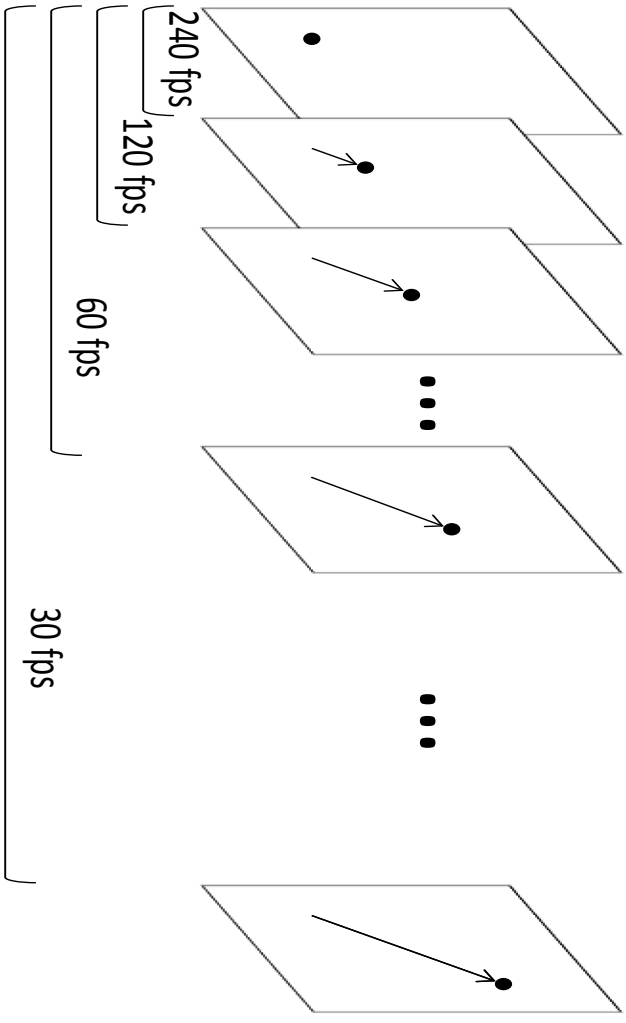


그림 3-2 frame rate과 displacement 간의 관계

3.3 Accurate Frame Rate Selection

이전 장 까지는 모든 frame을 이용하지 않고 Multi frame rate을 사용하면서도 모든 움직임을 LK optical flow system의 accurate 범위로 변환 하는 것을 다루었다. 이번 장에서는 optimum frame rate을 선택하는 방법을 제안한다.

3.3.1 Magnitude Selection Algorithm

Multi frame rate을 이용하면 움직임의 크기는 high frame rate에서 가장 작은 움직임을 갖고 normal frame rate으로 갈수록 2배씩 증가하게 된다. LK optical flow system이 그림 3-1에서 설명한 target system (d)에 해당하므로 움직임이 매우 작을 때는 그림 2-6에서와 같이 error 가 saturate되어서 normal frame rate으로 변환할 경우 그림 2-8의 (d)와 같이 error가 증폭되어 accurate한 결과를 얻지 못한다. 하지만 그림 3-3에서와 같이 error가 일정한 값으로 saturate된다는 것은 optical flow의 결과가 실제 움직임을 고정된 error만큼을 가지면서 따라간다는 것을 의미한다. 즉, 실제 움직임이 매우 작을 경우 그 움직임의 estimation 결과는 saturated error 만큼을 갖는다는 것이다. 이 사실을 이용하면, high frame rate 부터 low frame rate순서로 optical flow magnitude를 비교할 경우 accurate frame rate을 선택할 수 있다. 그림 3-4는 x축과 y축 방향의 optical flow 결과가 (u,v)로 표현될 경우 magnitude는 $\sqrt{u^2+v^2}$ 가 되며, 비교할 threshold인 uv_th를 이용해서 accurate frame rate을 선택하는 알고리즘을 보여준다. 가장 높은 frame rate의 optical flow결과부터 일정한 값의 threshold와 비교를 시작한다. 만약 처음

결과부터 threshold를 넘게 되면, 이 경우는 움직이는 물체의 속도가 너무 빨라 가장 빠른 frame rate으로도 처리가 어려운 경우를 나타낸다. 이 경우 가장 빠른 frame rate을 선택하고 invalid flag를 띄운다. 반대로 처음 결과가 threshold보다 작으면 다음 단계로 빠른 frame rate의 결과를 비교한다. Threshold 보다 큰 값이 나올 때까지 이 과정을 반복한다. Threshold보다 큰 값이 나오면 threshold 간의 차이를 구하고 바로 이전 frame rate의 결과와 threshold간의 차이와 비교하여 더 작은 값을 갖는 frame rate을 선택하고 valid flag를 띄운다. 만약 가장 느린 frame rate까지 threshold보다 큰 값이 나오지 않으면 마지막 frame을 선택하고 valid flag를 띄운다. 이 경우는 움직임이 없거나 가장 느린 frame rate으로도 측정하기 어려운 매우 느린 움직임의 경우이다. Frame rate이 선택되면 다음 pixel을 처리한다.

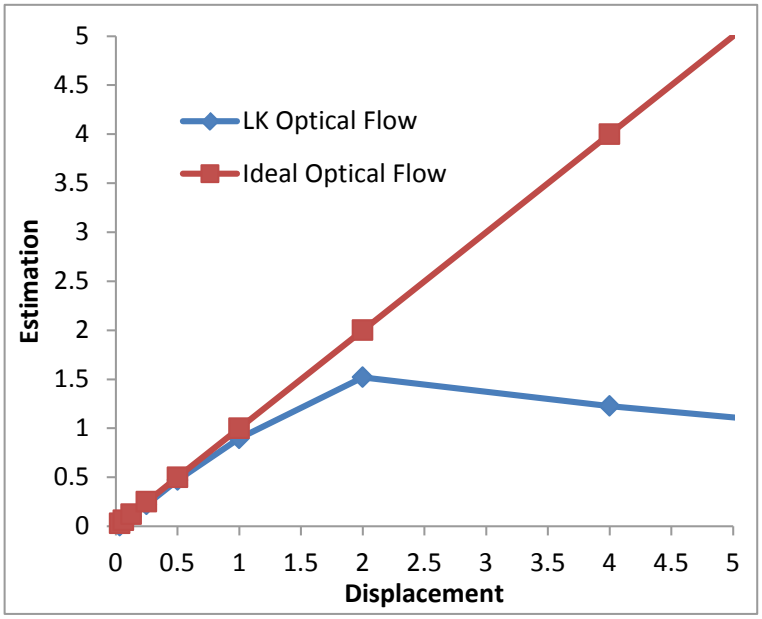


그림 3-3 Displacement와 Estimation 결과

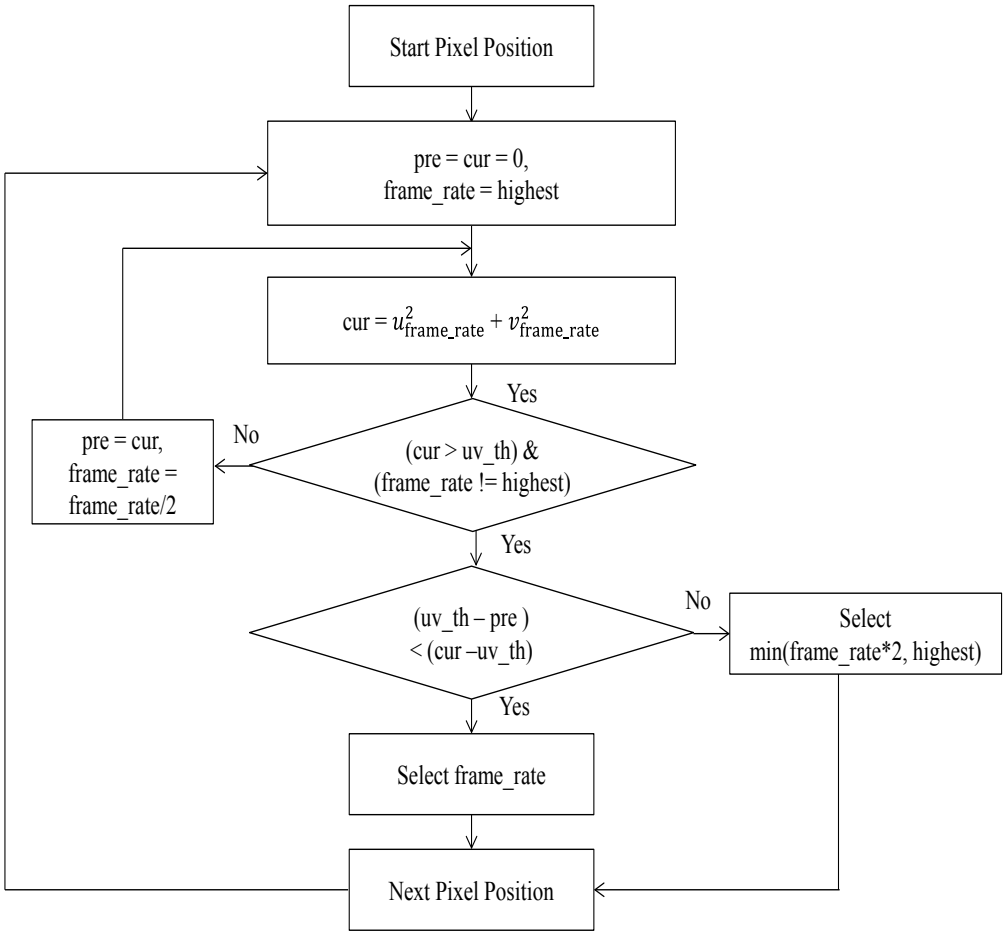


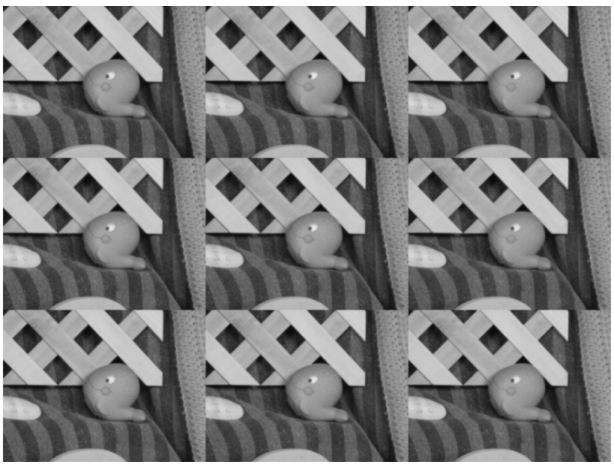
그림 3-4 optical flow result를 이용한 frame rate selection 알고리즘

3.3.2 Magnitude Algorithm Validation

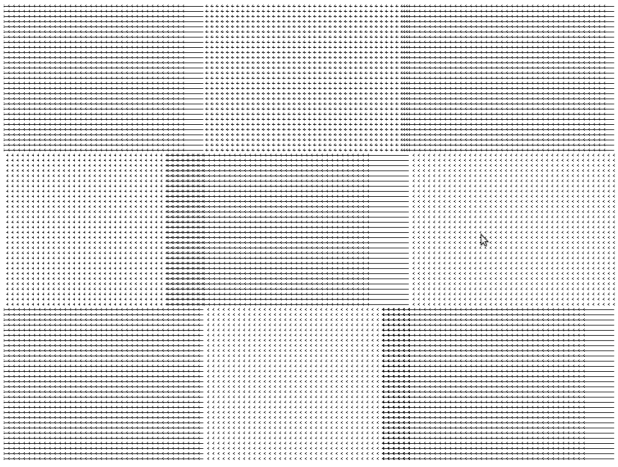
그림 3-5는 제안한 알고리즘의 동작을 확인하기 위해서 9가지의 다양한 속도로 움직이는 test sequence를 나타낸다. Test sequence는 다양한 속도로 움직이는 물체를 high frame rate의 camera로 찍은 후 multi frame rate에 해당하는 frame들을 선택하는 것과 같은 결과를 나타낸다. Multi frame rate의 test sequence에 대한 LK optical flow의 결과를 비교함으로써 다양한 frame rate에 대한 LK optical flow의 동작을 분석한다. 움직이는 물체들은 검은색 점으로 표현되며, 움직이는 속도는 9가지로 바꿔가면서 만들어진다. 점들의 움직이는 속도는 normal frame rate인 30 frame rate을 기준으로, 매 frame당 0.5, 1, 2, 4, 8, 16, 32, 48 그리고 64 픽셀의 속도로 좌측으로 움직인다. 가장 빠른 속도가 64픽셀의 속도로 움직이므로 충분히 작은 움직임으로 표현하기 위해서 최대 frame rate은 3840 frame rate으로 설정하였다. 이 경우, normal frame rate기준으로 64 pixel/frame의 속도는 3840 frame rate에서는 0.5 pixel/frame의 속도로 바뀌게 된다. 아래 그림은 9가지 속도를 갖는 점들의 test sequence 와 해당 ground truth값을 보여준다. Test sequence는 9개의 범위로 나누고 각 범위는 독립적인 움직임 속도를 갖는다. 9개의 구간 중 좌측 상단부터 우측 순서로 32, 4, 16, 좌측 중간부터 우측 순서로 2, 64, 1 그리고 좌측 하단부터 우측 순서로 8, 0.5, 48 pixel/frame의 속도를 각각의 구간이 갖는다.

그림 3-6은 optical flow result를 이용한 frame rate selection 결과를 보여준다. X축의 Frame rate은 상대적인 비율로 나타내었고 y축은 선택된 횟수를 나타낸다. 모든 속도의 구간에 대해서 87% 이상의 경우에 0.5 pixel/frame

을 선택하였다. 단 48pixel/frame의 속도로 움직이는 구간에서는 0.5 pixel/frame의 속도 구간이 3840 frame rate과 1920 frame rate중간에 형성되므로 두 frame rate로 나뉘어서 선택되었다. 두 frame rate을 합하면 92%의 선택 결과를 갖는다. 결과에서 확인할 수 있듯이, 제안한 방법은 해당 움직임 속도에 대한 가장 optimum한 frame rate을 선택하는 것을 보여준다.



(a) 9 velocity test sequence



(b) 9 velocity test sequence ground truth

그림 3-5 9가지 속도의 test sequence와 해당 ground truth

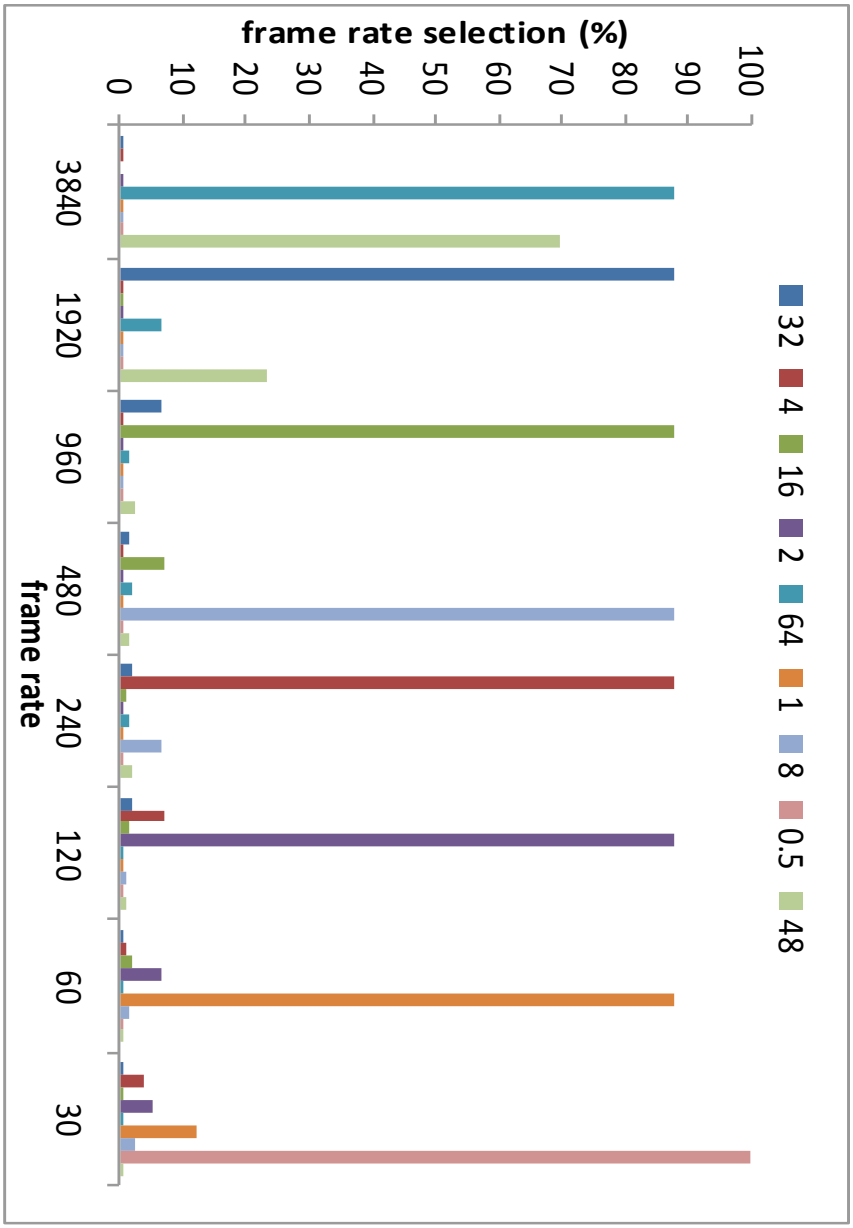
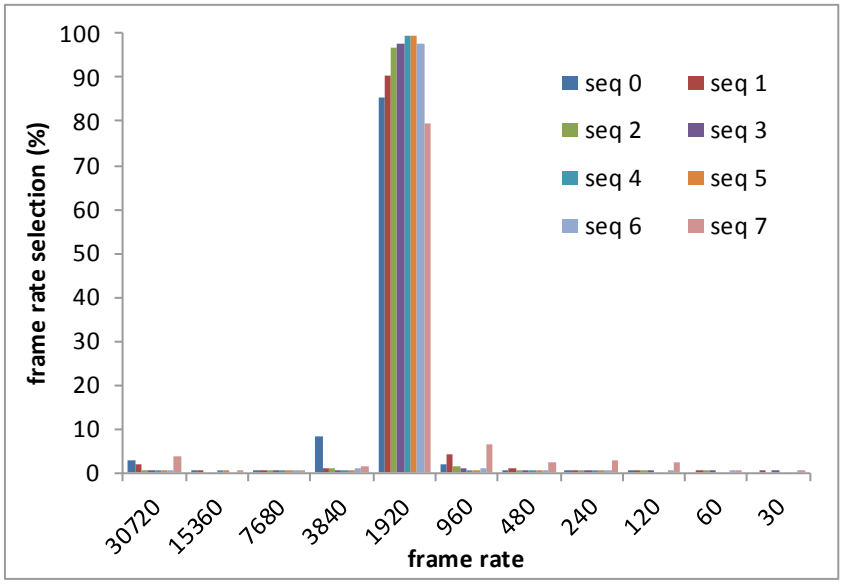
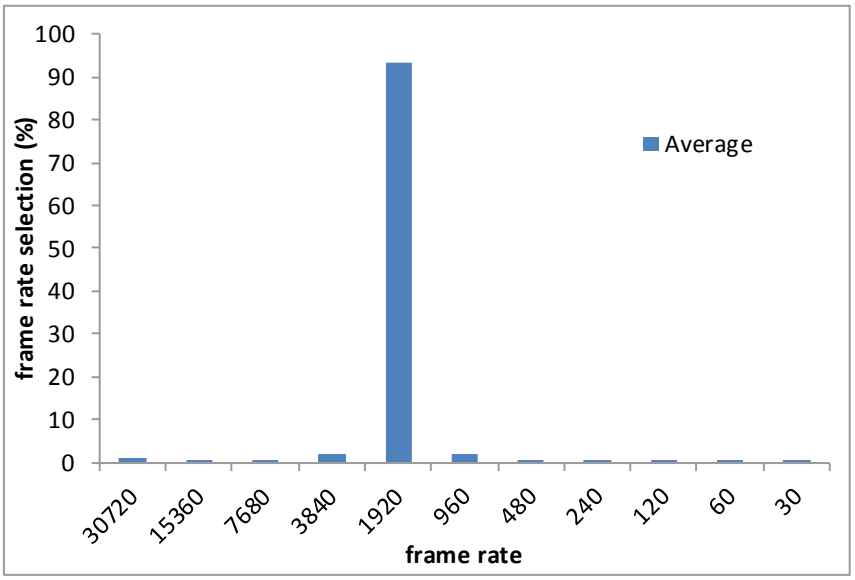


그림 3-6 9가지 속도에 대한frame rate selection 결과

이번에는 동일한 움직임 속도를 갖는 다양한 이미지에 대한 frame rate selection 알고리즘의 결과를 얻기 위해서 Middlebury의 8개를 이용하여 만든 high frame rate test sequence를 가지고 실험하였다. 8개의 test sequence들은 normal frame rate에서 32pixle/frame의 속도로 shift하도록 만들어졌으며, 그림 3-7은 8개의 test sequence에 대한 앞에서 제안한 uv알고리즘을 적용한 결과이다. Frame rate은 모든 test sequence의 모든 pixel별로 선택되고 각각의 test sequence별로선택된 frame rate횟수가 백분율로 표시된다. (a)는 각 test sequence별로의 frame selection 결과를 보여주고 (b)는 모든 test sequence에 대한 frame rate selection을 average한 결과를 보여준다. (a)와 (b)의 결과를 보면, 1920 frame rate이 가장 많이 선택되었다. 30 fps에서 32pixle/frame의 속도로 움직이므로 1920 fps에서는 0.5 pixel/frame의 속도로 움직인다. 결국 LK optical flow로 측정할 때, 가장 accuracy가 높은 움직임의 범위의 frame rate을 선택하였음을 확인할 수 있다. (b)의 결과를 보면 average 93.21 %의 픽셀들이 0.5 pixel/frame의 움직임을 갖는 frame rate을 선택하였음을 확인할 수 있다.



(a) frame selection (%)



(b) average frame selection (%)

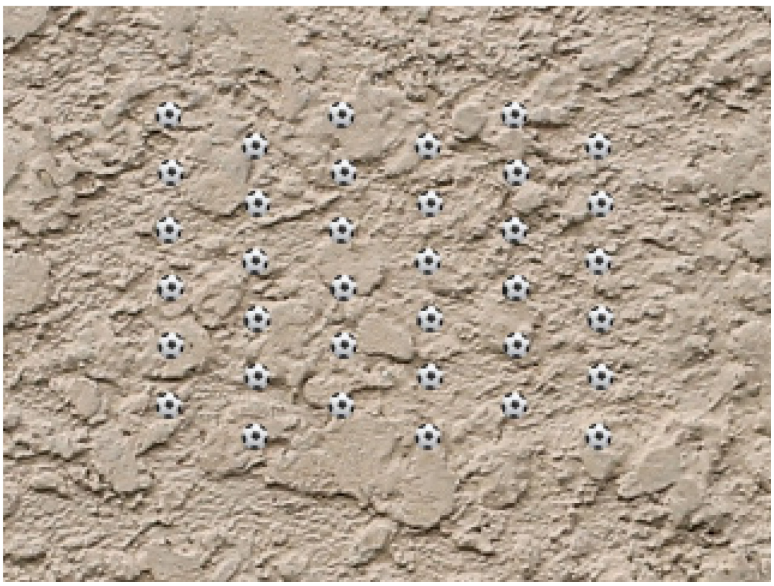
그림 3-7 optical flow result를 이용한 frame selection 실험 결과

지금까지 high frame rate을 위한 test sequence들은 이미지 전체가 shift하도록 만들어 졌다. 하지만 이런 test sequence를 가지고는 움직임의 불연속이 존재하는 물체의 경계부분에서 경우 알고리즘이 어떤 결과를 나타내는지 확인이 어렵다. 물체의 경계부분은 LK optical flow에서 사용하는 window안에 서로 다른 움직임을 갖는 픽셀들이 존재하기 때문에 근접한 픽셀들은 모두 동일한 optical flow를 갖는다는 LK system의 가정을 위배하게 된다. 이 경우는 앞에서 설명한 target system (e)에 해당하는 경우처럼 시스템의 가정이 만족되지 않기 때문에 어떤 optical flow결과를 얻게 될지 알 수 없게 된다. 이런 상황에서 제안하는 frame rate selection 알고리즘이 어떤 성능을 보이는지 확인해 본다.

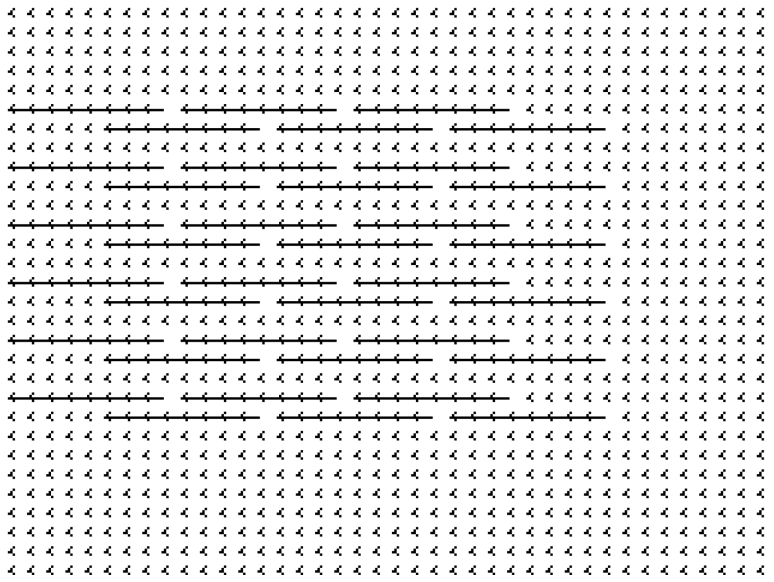
물체의 경계부분을 테스트 하기 위해 고정된 배경위에 물체가 움직이는 test sequence를 만든다. 고정된 움직임의 background 이미지는 일반 jpg영상을 이용하였고, 움직임이 있는 foreground 이미지는 Blending 정보가 있는 RGBA type의 png 영상을 이용하였다. Foreground 이미지의 작은 축구공들이 움직이는 영상을 사용한다. Foreground영역은 opaque 하게 보여지기 위해서 Blending값인 A채널의 값을 255로 설정하였고, Foreground의 이미지에서 color가 없는 영역은 transparent하게 하여 background이미지가 보이도록 blending 값을 0으로 두었다. Foreground 이미지의 움직임을 구현하기 위해 pixel 단위의 움직임은 해당 integer 만큼 foreground 이미지의 좌표를 변환하여 구하였고, sub-pixel움직임은 x축, y축 방향의 Lanczos3 6 tap 필터를 이용한 interpolation으로 구하였다. 움직임 변환된 foreground 이미지는

alpha값을 이용하여 background 이미지와 blending 되어 해당 움직임의 test sequence가 만들어진다. 그림 3-8은 고정된 배경위에서 움직이는 작은 축구공들의 test sequence와 ground truth이다. (a)는 Test sequence를 나타내며, 배경이미지 위의 축구공들이 동일한 속도로 좌측으로 이동한다. (b)는 64 pixel/frame의 속도에 대한 ground truth를 나타낸다.

그림 3-9 는 그림 3-8의 test sequence에 대한 multi-frame rate optical flow 결과를 나타낸다. (a)는 경계 부분의 움직임 정보를 자세히 확인하기 위해서 사용된 HSV(Hue-Saturation-Value) model로 optical flow 방향과 크기를 color code하는 방법이다. Test sequence에서 축구공은 좌측으로 shift하므로 cyan색으로 표현되고 배경은 움직임이 없으므로 크기가 0인 검정색으로 표현된다. (b)는 제안한 magnitude algorithm을 가지고 64 pixel/frame test sequence에 적용하였을 때, 얻은 결과이다. 눈으로 확인할 수 있듯이 물체의 경계부분이 동그랗게 원형을 그리지 못하고 있으며 색상 또한 cyan이 아닌 다른 임의의 색상이 움직임 경계 주변으로 많이 있는 것을 확인할 수 있다. 이 현상의 이유는 서로 다른 움직임의 경계부분에서는 LK optical flow의 가정이 맞지 않으므로 optical flow result는 실제 움직임을 표현하지 못하기 때문이다. 다음 장에서는 이 문제를 해결하기 위해서 개선된 알고리즘을 제안한다.

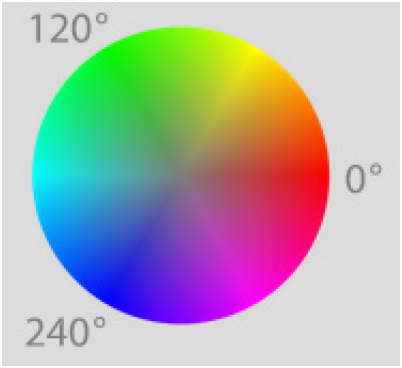


(a) small moving dots

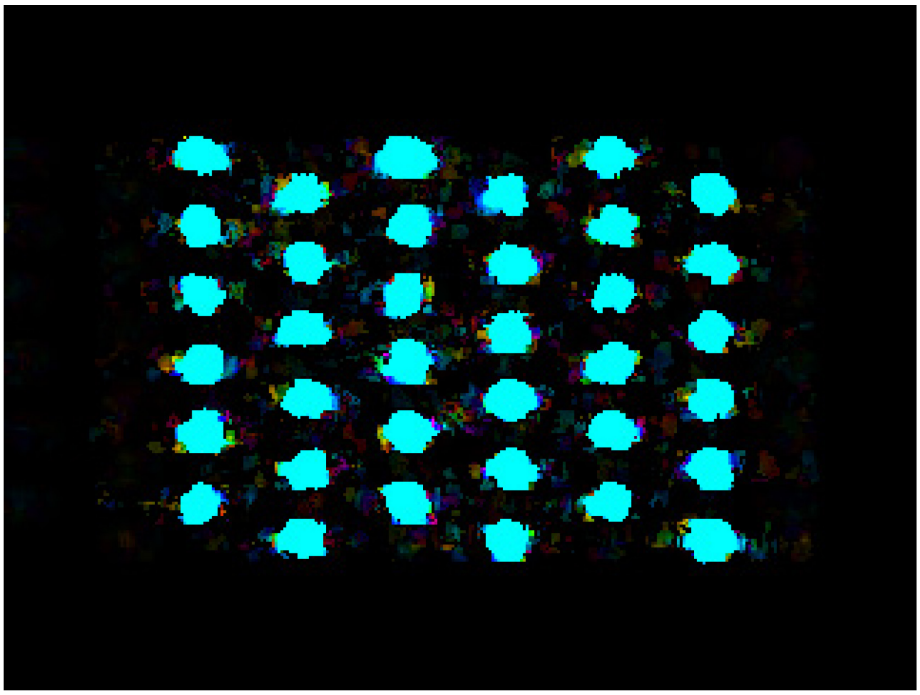


(b) 64 pixel/frame

그림 3-8 Small balls shifting test sequence and ground truth



(a) HSV color model



(b) Frame rate select algorithm

그림 3-9 magnitude algorithm을 적용한 결과

3.3.3 SSD(Sum of Squared Difference)

Optical flow결과의 신뢰도 측정은 현재 frame의 픽셀 값과 optical flow 결과에 의한 next frame의 픽셀 값간의 차이에 의해서 구한다. 이를 수치로 측정하는 방법은 SAD(Sum of Absolute Difference)나 SSD(Sum of Squared Difference)를 사용할 수 있다. SAD와 SSD는 아래 수식으로 정의된다.

$$\mathrm{SAD} = \sum_{\mathrm{x}=x_0-\mathrm{m}}^{\mathrm{x}=x_0+\mathrm{m}} \sum_{\mathrm{y}=\mathrm{y}_0-\mathrm{m}}^{\mathrm{y}=\mathrm{y}_0+\mathrm{m}} |J(x,y) - I(x+u, y+v)|$$

$$\mathrm{SSD} = \sum_{\mathrm{x}=x_0-\mathrm{m}}^{\mathrm{x}=x_0+\mathrm{m}} \sum_{\mathrm{y}=\mathrm{y}_0-\mathrm{m}}^{\mathrm{y}=\mathrm{y}_0+\mathrm{m}} (J(x,y) - I(x+u, y+v))^2$$

J와 I는 frame image를, x_0, y_0는 픽셀의 x축, y축 현재 위치를 나타내고 처리하는 윈도우 크기는 (2m+1)x(2m+1)이다. u, v 는 x축, y축 방향의 optical flow이다. SSD는 SAD보다 연산량이 많은 단점이 있으나 Error를 square하여 증폭시키기 때문에 error의 변화가 작을 때, 최소값을 찾는데 유리하다. Frame rate selection을 위해서는 temporal aliasing을 발생시키지 않는 움직임이 작은 구간의 error변화를 비교하기 때문에 SSD를 사용하기로 한다. 또한 이 값을 앞으로는 residual, $\varepsilon(u, v)$ 로 부르기로 한다. $I(x+u, y+v)$를 1차 Taylor seriese로 치환하면

$$\varepsilon(v) = \sum_{\mathrm{x}=x_0-\mathrm{m}}^{\mathrm{x}=x_0+\mathrm{m}} \sum_{\mathrm{y}=\mathrm{y}_0-\mathrm{m}}^{\mathrm{y}=\mathrm{y}_0+\mathrm{m}} (J(x,y) - I(x,y) - \frac{\partial I}{\partial x}u - \frac{\partial I}{\partial y}v)^2$$

여기서 $J(x,y) - I(x,y)$는 temporal image 미분값을 의미하므로, $-I_t$가 되고, $\frac{\partial I}{\partial x}$와 $\frac{\partial I}{\partial y}$는 각각 I_x와 I_y가 되고 부호를 바꾸면,

$$\varepsilon(v) = \sum_{x=x_0-m}^{x=x_0+m} \sum_{y=y_0-m}^{y=y_0+m} \left(I_x u + I_y v + I_t\right)^2$$

$$= \sum_{x=x_0-m}^{x=x_0+m} \sum_{y=y_0-m}^{y=y_0+m} (I_x I_x u^2 + I_y I_y v^2 + I_t I_t + 2I_x I_y uv + 2I_x I_t u + 2I_y I_t v)$$

가 된다. 이 수식에서 확인할 수 있듯이, residual을 구하기 위해서 필요한 요소들 중 $I_t I_t$ 만을 제외한 5개의 $I_x I_x$, $I_x I_y$, $I_y I_y$, $I_x I_t$, $I_y I_t$ 값들과 u, v 는 LK optical flow과정에서 구한 값들로 이미 알고 있는 값들이다. 결과적으로 Residual은 $I_t I_t$ 만 추가로 연산하면 구할 수 있다. 이렇게 Taylor series를 이용해서 연산량을 크게 줄였지만, 1차 미분값을 사용했기 때문에 움직임이 큰 경우 간략화해서 구한 residual 과 실제 SSD값간의 오차가 커지게 된다. 그러므로 residual의 사용은 움직임이 작은 구간에 대해서만 사용해야 한다.

위 식을 보면 residual의 Ix, Iy, It값들은 gradient정보인데, 이 값들은 이미지의 edge부분과 같이 밝기의 변화가 있는 경우 발생한다. LK optical flow에서 사용하는 window크기 내에 얼마나 많은 gradient정보가 있는지에 따라서 전체 residual값이 영향을 받게 된다. 이로 인해서 residual을 이용해서 신뢰도를 측정하기 위해서는 gradient에 의한 dependency를 제거한 normalized residual, ε_n을 사용한다. Normalized residual은 아래 수식과 같이 residual값을 gradient Ix Ix, Iy Iy, ItIt의 합으로 나눈 값으로 정의한다. [29]

$$\varepsilon_n(v) \triangleq \sum_{x=x_0-m}^{x=x_0+m} \sum_{y=y_0-m}^{y=y_0+m} \frac{\left(I_x u + I_y v + I_t\right)^2}{(I_x I_x + I_y I_y + I_t I_t)}$$

Normalized residual값은 error의 정도를 나타내는 수치이므로 threshold값

인 nr_th를 이용해서 optical flow result의 신뢰도를 측정할 수 있다. 하지만, Normalized Residual을 얻기 위해 필요한 나누기 연산으로 hardware 구현에 제약이 생긴다. 이번 장에서는normalized residual의 hardware구현이 용이하도록 divider없이 구현하는 방법을 제안한다. 아래 식은 normalized residual이 threshold보다 큰지 작은지를 판단하는 식이다.

$$NR < nr\_th$$

이 조건의 만족여부를 알면 optical flow result의 신뢰도 여부를 판단할 수 있으므로 frame rate selection 에서의 optical flow result 신뢰도 문제에 적용할 수 있다. 즉, 필요한 정보는 normalized residual값이 아니고, 위 수식의 만족 여부이다. 위 수식을 풀어서 쓰면,

$$\sum_{x=x_0-m}^{x=x_0+m}\sum_{y=y_0-m}^{y=y_0+m}\frac{(I_x u + I_y v + I_t)^2}{(I_x I_x + I_y I_y + I_t I_t)} < nr_{th}$$

가 된다. $(I_x I_x + I_y I_y + I_t I_t)$는 항상 양수이므로 양변에 곱하면,

$$\sum_{x=x_0-m}^{x=x_0+m}\sum_{y=y_0-m}^{y=y_0+m}(I_x u + I_y v + I_t)^2 < \sum_{x=x_0-m}^{x=x_0+m}\sum_{y=y_0-m}^{y=y_0+m}(I_x I_x + I_y I_y + I_t I_t)\, nr_{th}$$

$$\sum_{x=x_0-m}^{x=x_0+m}\sum_{y=y_0-m}^{y=y_0+m}(I_x u + I_y v + I_t)^2 - \sum_{x=x_0-m}^{x=x_0+m}\sum_{y=y_0-m}^{y=y_0+m}(I_x I_x + I_y I_y + I_t I_t)\, nr_{th} < 0$$

가 된다. 이 값들 중, $I_t I_t$를 제외한 나머지 5개의 항목들은 LK optical flow구현 과정에서 구한 값들이므로 Normalized Residual(NR)을 구하기 위해서 추가적으로 필요한 값은 $\sum_{x=x_0-m}^{x=x_0+m}\sum_{y=y_0-m}^{y=y_0+m} I_t I_t$이다. 그러므로 divider 없이 Normalized Residual을 이용한 optical flow의 신뢰도를 측정할 수 있게 된다.

3.3.4 Magnitude with NR Selection Algorithm

Magnitude selection algorithm이 움직임의 불연속 부분에서 문제를 발생시키는 이유는 LK optical flow의 주변 픽셀은 동일한 움직임을 갖는다는 가정이 성립하는 구간의 결과이기 때문이다. 신뢰할 수 없는 optical flow result에 의해서 잘못된 frame rate을 선택하고 선택된 frame rate의 error는 신뢰도를 모르는 LK optical flow system의 error이므로 결과적으로는 임의의 frame rate의 결과를 normal frame rate으로 변환하는 것과 동일한 결과가 된 것이다. 이번 장에서는 magnitude selection 알고리즘의 결과를 향상시키기 위해서 신뢰도 측정을 추가한다. 그림 3-10은 그림 3-4에 normalized residual을 이용해서 신뢰성을 판단하는 부분이 추가된 algorithm을 나타낸다. 즉, magnitude 를 비교하기 전, 우선 NR을 판단한 뒤, 신뢰할 수 있는 경우에만 다음단계로 진행한다. 만약 신뢰할 수 없는 경우, 바로 이전 frame_rate인 2배 높은 frame_rate을 선택하거나, 현재 frame_rate이 가장 높은 frame_rate인 경우 현재 frame rate을 선택한다.

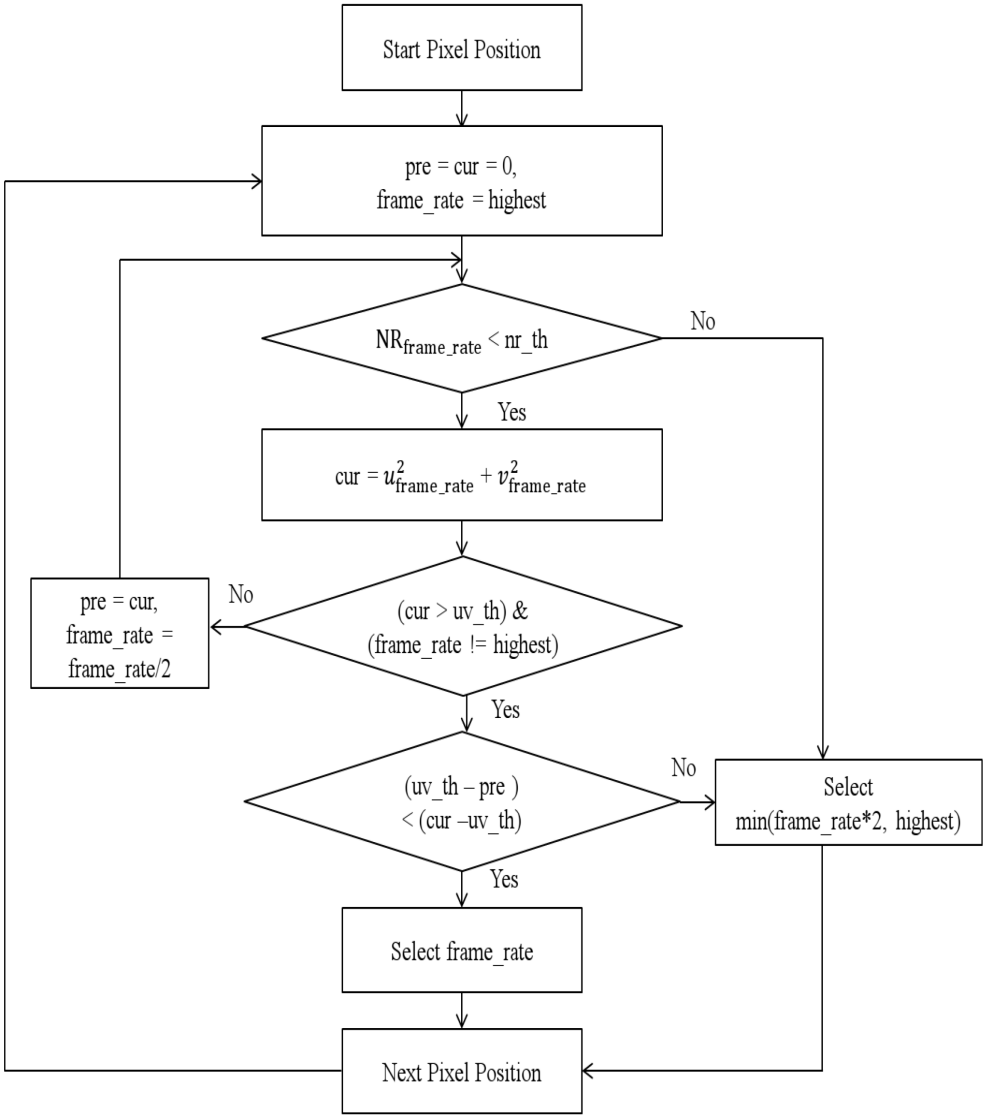


그림 3-10 optimal frame rate selection algorithm

그림 3-11은 수정된 축구공 test sequence를 가지고 magnitude with NR selection algorithm을 적용한 결과를 보여준다. 이전 결과와 비교해 볼 때, 축구공이 더욱 동그랗게 되었으며 경계부근의 울긋불긋한 색상도 많이 개선된 것을 확인할 수 있다.

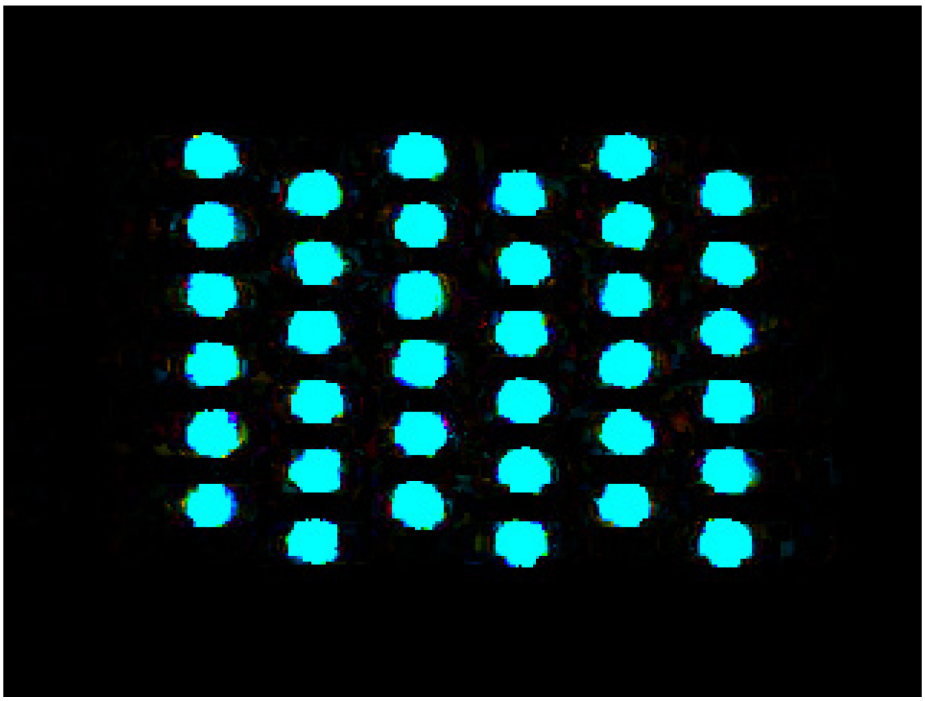


그림 3-11 magnitude with NR algorithm 결과

3.3.5 Temporal Aliasing

High frame rate sequence를 사용할 때 발생하는 문제 중 하나는 high frame rate은 temporal aliasing을 발생시키지 않더라도 normal frame rate에서는 temporal aliasing이 발생할 수 있다는 것이다. 예를 들어, 영화를 볼 때 수레바퀴가 처음 굴러가기 시작 할 때는 시계방향으로 굴러가다가 일정한 속도보다 빨라지게 되면 점차 수레바퀴의 회전 속도가 줄어들기 시작하여 정지하거나 오히려 반 시계방향으로 돌기 시작하는 것을 볼 수 있다. 이처럼 수레바퀴의 회전 속도가 카메라가 영상을 찍는 frame rate 보다 빨라지기 때문에 발생하는 왜곡이 temporal aliasing이다. [22]의 연구에서는 temporal aliasing이 발생하는 경우를 방지하기 위해서 spatial frequency를 고려한 maximum frame iterative calculation 횟수를 제한하였다. 횟수 제한 이유는 high frame rate에서부터 normal frame rate까지 정확하게 연산을 하더라도 temporal aliasing이 발생하면 optical flow result error는 크게 증가하기 때문이다. 즉, 쉽게 설명하면 횟수를 제한하지 않으면 수레바퀴가 거꾸로 회전하는 연산결과를 얻게 된다는 것이다. 제안하는 magnitude with NR selection algorithm은 LK optical flow의 accurate range 를 기준으로 frame rate을 선택하는데, 이 값이 0.5 ~ 1 부근이므로 자연스럽게 temporal aliasing 발생을 차단하게 된다. 또한 Taylor series로 간략화한 residual을 가지고도 좋은 결과를 얻을 수 있는 이유도 NR을 가지고 신뢰성을 확인하는 구간은 LK optical flow의 accurate range에 의해서 selection되기까지의 구간이므로 그 값이 충분히 작은 값이기 때문이다. 즉, 작은 움직임 구간에 대해서 frame rate selection algorithm을 사용하므로 Taylor series를 적용한 normalized

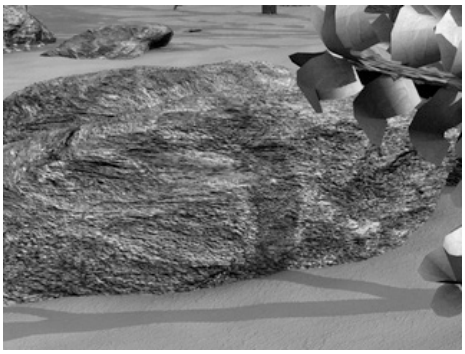
residual도 신뢰할 수 있고 temporal aliasing의 발생도 자동으로 피할 수 있다.

3.4 Multi Frame Rate Optical Flow Test Result

지금까지 제안한 Magnitude with NR selection algorithm을 이용하여 high frame rate system의 optical flow결과를 테스트한다. Middlebury 8개의 test sequence를 이용해서 shifting과 rotating 움직임의 high frame rate optical flow system을 위한 test sequence를 그림 3-12에서 나타낸다. Shifting 과 rotating 할 경우 test sequence의 margin 부분을 처리하기 위해서 전체 이미지를 사용하지 않고 320x240 크기를 사용하였다. Shifting 의 경우 normal frame rate 에서 32 pixel/frame의 움직임을 가정하였고, 최대 30720 frame rate까지의 움직임에 대한 test sequence를 제작하였다.

(a) Seq0 Urban3 (b) Seq1 Urban2

(c) Seq2 Demetrodone

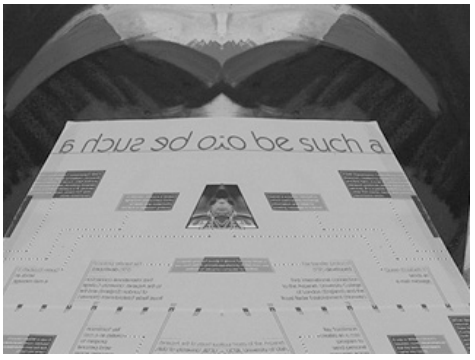


(d) Seq3 Grove2

(e) Seq4 Grove3

(f) Seq5 Hydrangea

(g) Seq6 Rubberwhale



(h) Seq7 Venus

그림 3-12 Middlebury test sequence

표 3-1는 optical flow의 AAE와 AEP 결과를 나타낸다. 실험 결과는 normal frame rate으로 변환한 결과를 나타낸다.

표 3-1 AAE와 AEP 결과

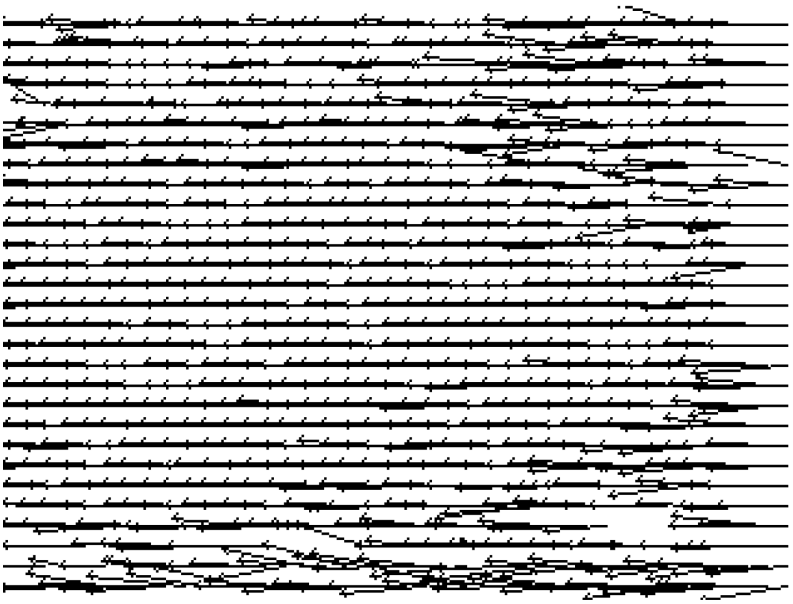
	Seq0	Seq1	Seq2	Seq3	Seq4	Seq5	Seq6	Seq7
AAE	6.27	3.29	3.79	1.5	1.44	2.18	2.84	3.47
AEP	7.46	3.18	3.36	1.9	1.44	2.24	2.55	3.63
Density	59.76	73.21	77.95	95.43	98.79	97.48	87.81	61.5

(a) Shift Test Sequence

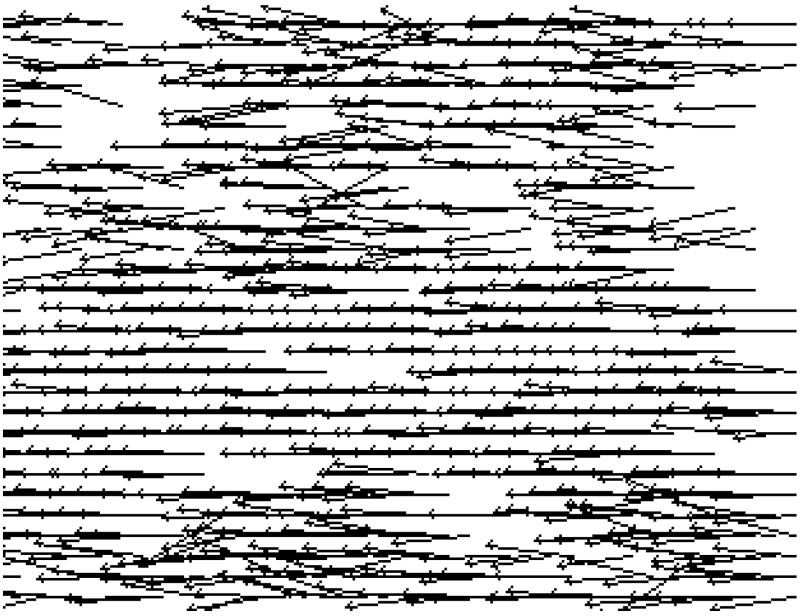
	Seq0	Seq1	Seq2	Seq3	Seq4	Seq5	Seq6	Seq7
AAE	10.01	7.19	6.81	5.93	6.22	5.84	6.9	7.47
AEP	5.89	5.02	4.69	4.46	4.3	3.78	4.37	5.35
Density	62.34	67.95	71.85	91.9	96.8	94.76	85.69	63.51

(b) Rotate Test Sequence

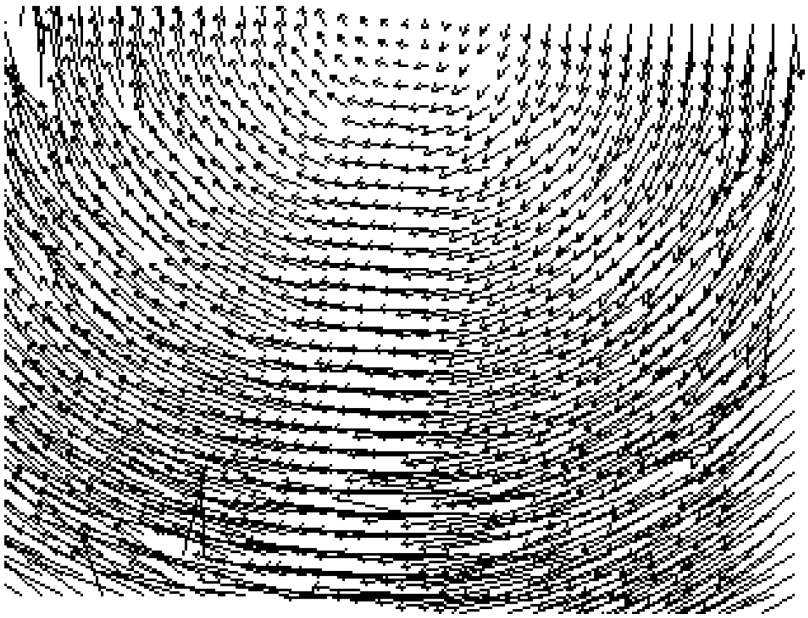
그림 3-13은 multi frame rate optical flow의 결과를 화살표로 나타낸 것이다. high frame rate sequence에서 frame rate selection이 정확히 이루어졌기 때문에 normal frame rate에서도 정확한 결과를 얻을 수 있다. 하지만 전체 이미지 중 많은 부분에 대해서는 optical flow결과를 얻지 못하였다. 이 이유는 LK optical flow에서 사용하는 window가 5x5의 고정된 크기를 사용하기 때문에 test sequence에서 corner가 window 크기보다 너무 큰 경우가 있기 때문이다. 이 현상은 Aperture problem으로 불리며 window크기 내에 측정 가능한 corner가 존재하지 않을 경우 optical flow를 구할 수 없다. 이 문제를 해결하기 위해서 다음 장에서 Multi-Scale Optical Flow를 제안한다.



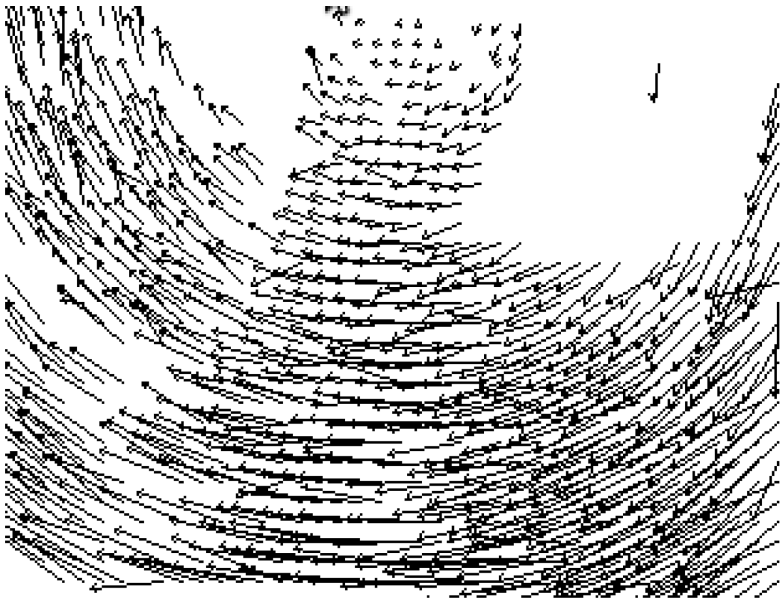
(a) seq 3



(b) seq 1



(c) seq4



(d) seq7

그림 3-13 shifting and rotating optical flow result

3.5 Comparisons with previous works

표 3-2는 제안한 multi-frame rate 알고리즘과 기존의 high frame rate 알고리즘과의 비교를 나타낸다. 두 번째 column은 각 알고리즘의 기본이 되는 optical flow알고리즘을 나타낸다. Proposed 알고리즘과 Lim의 알고리즘은 LK optical flow를 기본 알고리즘으로 사용하고 Ishii의 알고리즘은 frame 간의 픽셀 변화량을 이용해서 temporal gradient를 구하도록 수정한 modified LK 알고리즘을 사용한다. 세 번째 column은 frame 당 요구되는 연산량을 나타낸다. Proposed 알고리즘은 1회의 LK optical flow와 그 결과의 magnitude를 이용한 frame rate selection과정으로 이루어 졌다. Lim의 알고리즘은 2회의 LK optical flow와 1회의 Warping과정으로 이루어 졌다. 1차 LK optical flow 처리 후, 그 결과를 이용해서 image warping한 뒤 보정과정을 위해 1회의 LK optical flow가 추가되는 구조이다. Ishii의 알고리즘은 1회의 modified LK optical flow와 merging과정으로 이루어졌다. Ishii의 알고리즘은 Various frame rate을 만들기 때문에 각 픽셀의 연산시점이 모두 달라질 수가 있다. 그러므로 최종적으로 normal frame rate으로 optical flow결과를 merge하는 과정이 추가된다. 네 번째 column은 frame 당 요구되는 외부 memory access bandwidth를 나타낸다. 제안한 알고리즘은 temporal gradient kernel로 [-1 0 1]을 사용하기 때문에 3장의 frame이 필요하고 temporal Gaussian smoothing처리를 위해서 2장의 frame 이 추가된다. Lim의 알고리즘은 temporal gradient kernel로 [-1 1]을 사용하므로 2장의 frame이 필요하고 warping을 위해 1장이 추가로 사용될 수 있다. Ishii의 알고리즘은 [-1 1] temporal gradient kernel을 사용하므로 2장의 frame이 사용되지만 modified

LK 알고리즘은 pixel의 변화량이 정확도에 중요한 영향을 끼치므로 픽셀당 10 bit를 사용하였다. 마지막 column은 high frame rate이 normal frame rate의 n배 빠른 경우, 즉 n frame ratio인 경우 요구되는 연산량과 bandwidth를 나타낸다. Lim과 Ishii의 알고리즘의 경우 frame rate증가와 동일한 비율로 n배의 연산량과 bandwidth가 증가하는 반면 제안하는 알고리즘은 O(log n)의 비율로 증가한다. 즉, frame rate이 증가할수록 제안하는 알고리즘은 기존 알고리즘 대비 요구되는 연산량과 bandwidth는 기하급수적으로 감소한다.

표 3-2 Comparisons with previous high frame rate algorithms

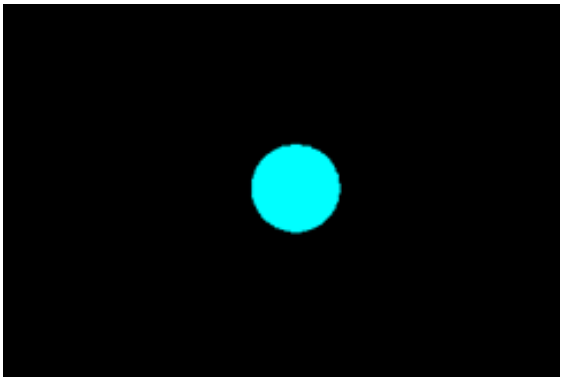
	Base Algorithm	Computation per Frame	Memory Access BW per Frame	Computation for n Frame Ratio
Proposed	LK	1 LK + Selecting	3+2 ([-1 0 1] + Gaussian)	O(log n)
[Lim]	LK	2 LK + Warping	2+1 ([-1 1] + warping)	O(n)
[Ishii]	Modified LK	1 Modified LK + Merging	2.25 ([-1 1] + 10bits)	O(n)

Lim의 알고리즘은 real-time 을 고려한 알고리즘이 아니고, Ishii의 논문에서도 frame rate의 증가에 연산량이 linear하게 증가하는 문제로 32x32 크기의 block 단위로 구현을 하였듯이 본 논문에서 목표로 하는 pixel단위의 real-time and accurate optical flow 알고리즘으로는 적합하지 않다. 그러므로 제안하는 알고리즘의 accuracy비교는 pixel레벨의 real-time optical flow가 가능한 LK optical flow기반의 알고리즘 중 가장 정확한 pyramidal LK optical flow와 비교하도록 한다. Pyramidal LK는 full density를 지원하므로 정확한 비교를 위해서 다음 장에서 제안하는 multi-scale 알고리즘을 포함한 Multi-frame rate and multi-scale optical flow와 Pyramidal LK알고리즘을 비교한다.

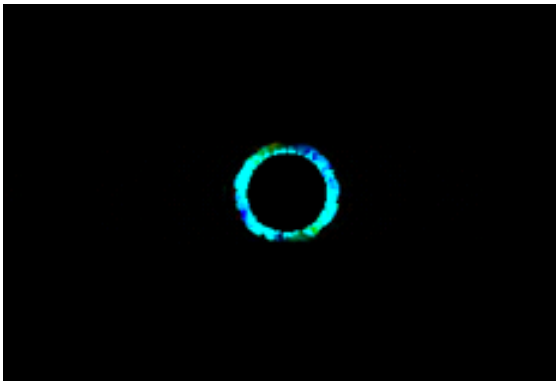
제4장 Multi-Scale Optical Flow

이번 장에서는 Multi frame rate optical flow방법의 이용해서 full density지원을 위해 Pyramidal Image를 사용하는 multi scale optical flow를 제안한다. 그림 4-1의 (a)는 검은색 dot이 64 pixel/frame의 속도로 좌측으로 움직이는 test sequence이다. (b)는 ground truth정보를 color code로 나타낸 것이고 (c)는 제안하는 multi frame rate optical flow 결과를 color code하여 나타낸 결과이다. (c)를 보면 Dot 움직임의 가운데 부분은 optical flow가 측정되지 않은 것을 확인할 수 있다. Optical flow 연산이 안 되는 이유는 window size 내부에 측정할 수 있는 corner 정보가 없기 때문이다. 이 문제를 aperture problem이라고 하며 일정한 크기의 window를 가지고 다양한 이미지의 optical flow를 측정하려 할 때 corner의 크기가 너무 클 경우 인식하기 어렵기 때문에 발생한다. 이 문제를 해결하기 위해서는 window 크기를 늘리거나 이미지의 크기를 줄임으로 써 window의 크기와 corner의 크기가 측정하기 적합한 크기가 되도록 하는 방법이 있다. 본 논문에서는 이미지의 크기를 줄임으로 써 corner 크기를 일정한 크기의 window에 맞추는 pyramidal Image 접근방법을 이용하여 multi frame rate을 이용한 optical flow의 density 향상시키는 방법을 제안한다.

(a) Moving Dot



(b) Ground Truth of Moving Dot



(c) Proposed Multi Frame Optical Flow Result

그림 4-1 Moving Dot 결과

4.1 Pyramidal Level Selection

그림 4-2는 original image를 1/2 크기로 downscale을 반복하여 만든 Pyramidal Image를 이용해서 다양한 level의 optical flow결과를 merge하는 구조를 보여준다. Pyramidal Image의 각 level들은 Single level에서 사용한 동일한 방법으로 LK optical flow결과를 구하고 optimum frame rate을 선택한다. Optimum frame rate이 선택되면 가장 신뢰할 수 있는 level의 결과를 선택한다. Level 0의 결과는 level0의 1개의 픽셀에 대한 결과이고 Level 1의 결과는 level 0의 4개 픽셀의 결과이다. 이처럼 level 이 올라갈 수록 optical flow 결과들은 4의 지수로 늘어나는 level 0의 픽셀면적의 움직임을 나타낸 결과이다. 이러한 이유로 동일한 조건이라면 낮은 level의 결과를 우선 선택하는 것이 더 정교한 결과를 얻을 수 있다. 반면에, 낮은 level의 결과에 신뢰도가 떨어질 경우 신뢰도를 만족하는 상위 level의 결과를 선택해야 density와 accuracy를 모두 향상시킬 수 있다.

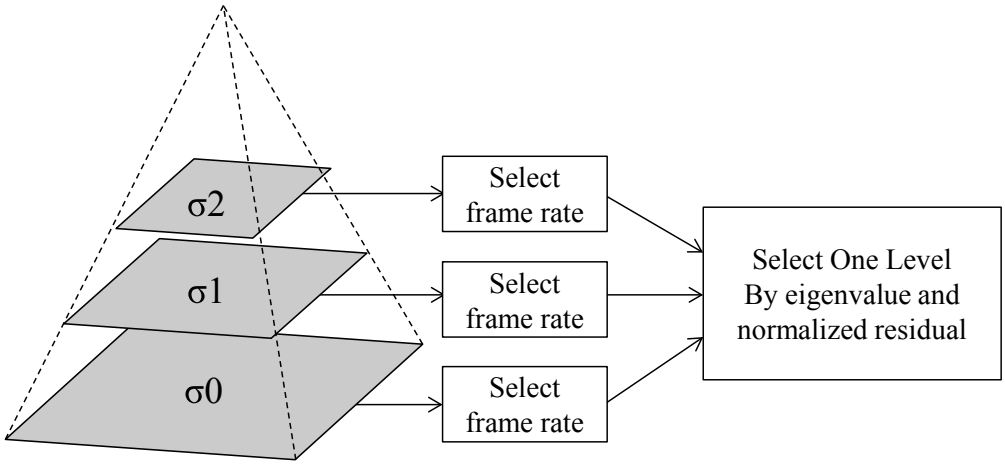


그림 4-2 Multi Scale LK Optical Flow System

4.2 Level Selection Algorithm

제안하는 알고리즘은 eigenvalue threshold인 eig_th 와 normalized residual threshold인 nr_th를 가지고 신뢰도를 측정하고 이를 만족하는 가장 낮은 level을 선택한다. 이 과정은 그림 4-3에 나와있다. Eigen은 window안에 있는 pixel들의 cornerness정도를 측정하기 위한 것이고 NR은 accuracy정도를 측정하기 위한 것이다. 만약 모든 level에 대해서 이 조건을 만족하는 결과를 얻지 못할 경우 eigenvalue 값을 1/2씩 줄여나가면서 검색을 반복한다. 최대 반복 횟수는 max_eig_count로 설정한다.

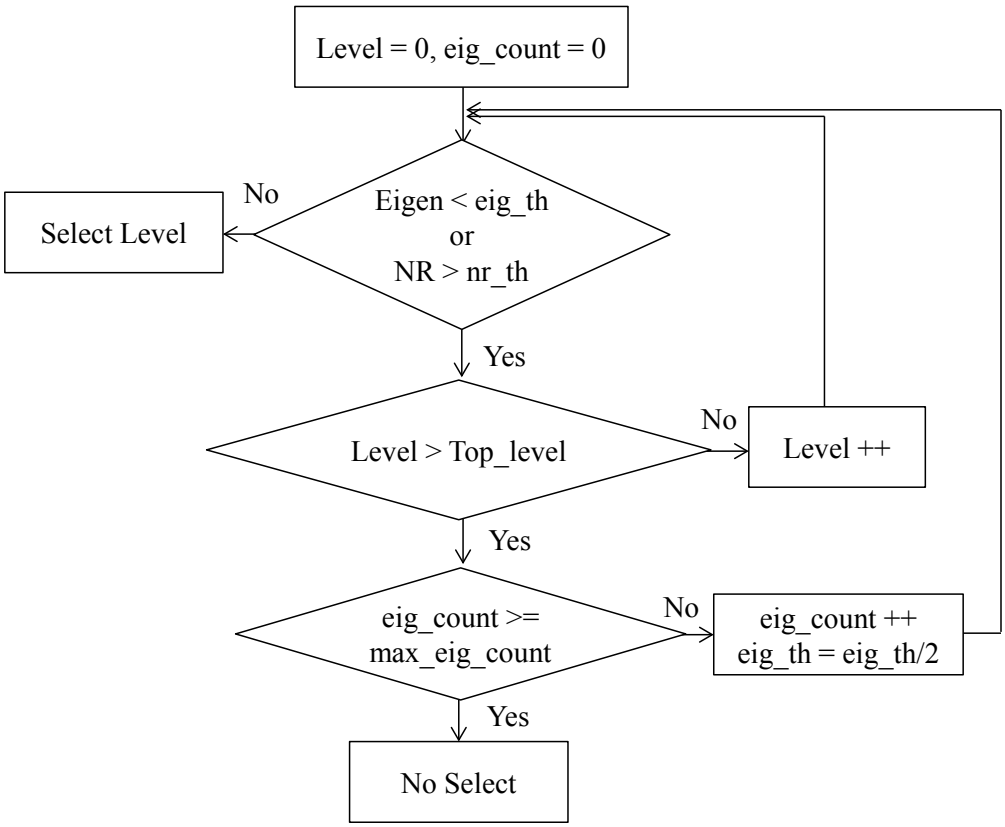


그림 4-3 Level Selection Algorithm

4.3 Pyramidal Image Generation

이미지의 크기를 줄여서 corner의 크기를 줄이는 방법은 pyramidal LK optical flow [36] 에서 제안되었다. Original 이미지를 Gaussian sigma가 1인 [1 4 6 4 1] kernel로 Smoothing 필터 처리한 뒤, 가로 세로 방향으로 2:1 downscale해서 가로와 세로 길이가 1/2로 줄어든 이미지를 만든다. downscale된 이미지를 가지고 동일한 방법으로 다시 1/2로 크기가 줄어든 이미지를 만든다. 일반적으로 2~4회 반복해서 pyramidal image 세트를 만든다. 이 방법으로 만들어진 Pyramidal Image들은 LK optical flow의 결과를 얻은 뒤, 그 결과위치에서 다시 optical flow연산을 수렴할 때까지 반복하는 Iterative LK optical flow를 적용한다. 이때, spatial and temporal Gaussian smoothing filter는 사용하지 않는다. 하지만, 본 논문에서 기본이 되는 LK optical flow algorithm은 iteration을 하지 않지만 spatial and temporal Gaussian smoothing filter를 사용하는 LK optical flow이다[27]. 그러므로 sigma가 1인 Gaussian Filter 를 반복해서 사용하면 Pyramid Level 이 올라갈수록 0.577의 sigma값으로 수렴하게 된다. 이 값은 single LK optical flow 를 위한 Gaussian Filter 의 sigma로는 너무 작은 값이 되므로 accuracy 가 떨어지게 된다. 이런 문제로 본 논문에서는 [37]에 의해 제안된 Scale invariant pyramidal image 방법을 변형하여 사용한다.

Scale invariant pyramidal Image 방법은 모든 level의 이미지가 갖는 frequency response가 동일한 수준으로 만들어서 동일한 sigma의 Gaussian smoothing filter로 처리된 것과 같은 결과를 얻는다. 그림 4-4은 이 방법을 설명한다. Level 0의 이미지가 Gaussian sigma, σ_0^0를 가지고 smoothing한 결

과일 때, downscale된 상위 Level의 이미지도 Gaussian sigma, σ_0^0 로 smoothing 한 결과와 동일한 결과를 얻는 것을 목적으로 한다. 즉, $\sigma_1^0 = \sigma_2^0 = \sigma_3^0 = \sigma_0^0$를 만족시키는 scale invariant pyramidal image generation이 목적이다. Level 1의 이미지를 σ_0^0로 smoothing 한 결과를 얻기 위해서는 Level 0의 이미지를 downscale하기 전 $2\sigma_0^0$로 smoothing 한 결과를 만들면 된다. $2\sigma_0^0$ 로 smoothing하는 것은 σ_0^0 로 smoothing하는 것보다 frequency response를 1/2로 축소시키는 것을 의미한다. σ_0^0 로 smoothing된 이미지를 가지고 $2\sigma_0^0$ smoothing된 결과를 얻기 위해서는 동일한 Gaussian σ_0^0 로 smoothing을 3회 반복한다. 이 결과를 가지고 2:1 downscale 하면 frequency response를 2배 확장하는 결과를 나타내므로 downscale된 이미지는 Gaussian sigma는 σ0로 Level 0와 동일한 sigma로 smoothing한 결과를 얻는다. 이 과정을 반복하면 아래 그림의 $\sigma_0^1 = \sqrt{2}\sigma_0^0$, $\sigma_0^2 = \sqrt{3}\sigma_0^0$ 그리고 $\sigma_0^3 = 2\sigma_0^0$가 된다.

제안하는 High frame rate system에서는 scale invariant pyramidal image generation을 변형하여 사용한다. 본 논문에서 Pyramidal image를 사용하는 이유는 optical flow density를 높이기 위해서 인데, Level 0의 Gaussian sigma 인 σ_0^0의 크기가 커지는 경우 상위 level의 이미지로 갈 수록 corner 정보가 blur 되어 density는 오히려 줄어든다. 결국 모든 level에 큰 값의 σ_0^0를 적용할 경우 pyramidal image를 사용하더라도 density의 증가가 충분하지 못해서 full density를 만족하지 못하는 경우가 발생한다. 이런 이유로 2:1 downscale을 위해 적용하는 3회의 Gaussian sigma 값을 0.707로 고정하여 사용한다. 결론적으로 변형된 방법은 각 level을 고정된 sigma의 Gaussian

Filter를 1회 처리 후 2:1 downscale을 하는 것과 동일하고 Pyramid Level이 올라갈수록 Gaussian Sigma 는 0.707로 수렴한다. 이 방법을 사용했을 때, pyramidal image의 각 level에 해당하는Gaussian sigma σ0, σ1 그리고 σ2는 표 4-1와 같다.

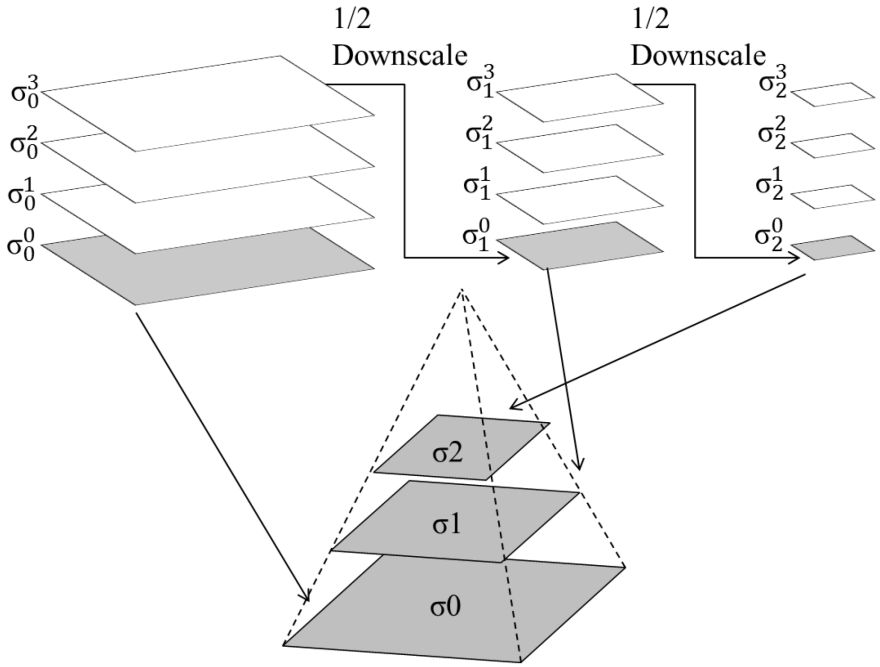


그림 4-4 scale invariant pyramidal image generation

표 4-1 Gaussian sigma of each level with 0.707 smoothing sigma

	Gaussian Sigma				
σ0	0.707	1	1.225	1.414	1.581
σ1	0.707	0.79	0.866	0.935	1
σ2	0.707	0.729	0.75	0.77	0.79
σ3	0.707	0.713	0.718	0.723	0.729
σ4	0.707	0.708	0.71	0.711	0.713
σ5	0.707	0.707	0.708	0.708	0.708

4.4 Proposed Algorithm Verification

이번 장에서는 지금까지 제안한 Multi Frame Rate and Multi Scale Optical Flow를 검증한다.

4.4.1 Accuracy comparison

표 4-2는 제안한 optical flow 방법(Gaussian Sigma=1.225) 과 Pyramidal LK optical flow결과를 비교한다. 사용한 test sequence는 앞장에서 설명한 Middlebury 이미지를 이용한 shifting 과 rotating test sequence이다. Pyramidal LK가 32 pixel/frame의 움직임에 대해서 정상적인 동작이 불가하기 때문에 유의미한 결과를 나타내는 8 pixel/frame의 sequence를 가지고 test 를 진행하였다. 결과를 보면 AAE, AEP의 대부분의 test sequence에 대해서 또한 average결과에 대해서도 pyramidal LK보다 뛰어난 결과를 나타내는 것을 확인할 수 있다.

표 4-2 AAE and AEP Comparison with Pyramidal LK

		AAE								
		Seq0	Seq1	Seq2	Seq3	Seq4	Seq5	Seq6	Seq7	Average
Shifting	Pro.	4.55	2.5	3.31	1	1.07	1.61	2.81	3.72	2.57
	Pyr. LK	9.78	4.55	4.24	3.26	1.96	3.12	3.88	8.25	4.88
Rotating	Pro.	8.82	6.56	7.15	6.45	6.04	5.9	7.6	10.88	7.43
	Pyr. LK	15.79	9.06	6.3	8.47	5.57	5.61	10.6	10.38	8.97

(a) AAE Result

		AEP								
		Seq0	Seq1	Seq2	Seq3	Seq4	Seq5	Seq6	Seq7	Average
Shifting	Pro.	1.35	0.53	0.66	0.28	0.27	0.39	0.55	0.82	0.61
	Pyr. LK	2.56	1.3	0.94	1.41	1.81	1.9	3.62	4.3	2.23
Rotating	Pro.	1.35	1.03	1.43	0.92	0.82	0.78	1	1.25	1.07
	Pyr. LK	3.1	1.91	2.54	2.89	1.32	1.12	3.32	2.07	2.28

(b) AEP Result

4.4.2 연산량 및 알고리즘 특성 comparisons

표 4-3는 제안하는 multi-frame rate and multi-scale 알고리즘과 pyramidal LK 알고리즘간의 computation 및 특성 비교를 나타낸다. 두 번째 row는 각 알고리즘의 기본이 되는 optical flow알고리즘을 나타낸다. 두 알고리즘 모두 pyramid Image를 사용하지만 Pyramidal LK의 경우 iterative LK를 사용하고 proposed 알고리즘은 본 논문에서 제안하는 multi-frame rate and multi-scale LK를 사용한다. 세 번째 row는 Level당 요구되는 연산량을 나타낸다. Pyramidal LK의 연산은 수렴할 때까지 반복하므로 이미지에 따라서 처리 횟수가 가변적이나 최대 20회로 제한한다. Pyramidal LK는 [-1 1] gradient kernel을 이용하고 Gaussian smoothing도 하지 않기 때문에 accurate구간이 매우 작기 때문에 iteration 횟수가 accuracy에 중요한 역할을 한다. 제안하는 알고리즘은 level당 1회의 LK를 처리한다. 네 번째 row는 모든 level을 포함하는 전체 pyramidal image 처리에 요구되는 연산량을 나타낸다. Pyramidal image는 level이 증가할수록 면적은 1/4 로 감소하므로 전체 면적은 1.5배를 넘지 않는다. 그러므로 요구되는 연산량은 각각 1.5배가 필요하다. 다섯 번째 row는 high frame rate이 normal frame rate보다 n배 높을

경우 요구되는 연산량을 나타낸다. Pyramidal LK는 normal frame rate에 대해서만 처리하므로 연산량은 증가하지 않는다. 제안하는 알고리즘은 frame ratio 증가에 따라서 기존 연산량에 log n 배 만큼 증가한다. 여섯 번째 row는 real-time 구현에 용이한 알고리즘 여부를 나타낸다. Pyramidal LK를 구현한 기존의 연구들을 보면 iterative warping 알고리즘으로 인해 많은 연산량이 요구된다. 또한 real-time 조건을 만족하기 위해서 frame buffer 용으로 embedded DRAM이나 external SRAM 을 사용해야 하기 때문에 frame image의 크기가 커질 수록 비용이 크게 증가하는 문제가 있다. 제안하는 알고리즘은 iterative operation을 사용하지 않고 LK operation뿐 아니라 frame rate selection 알고리즘도 pipeline구조로 가능하기 때문에 real-time 구현에 용이하다. 마지막 row는 수레바퀴의 역전 현상과 같은 temporal aliasing에 대한 각 알고리즘의 처리를 나타낸다. Pyramidal LK는 normal frame rate만을 처리하기 때문에 temporal aliasing을 인지하지 못한다. 제안하는 알고리즘은 자동으로 temporal aliasing의 발생을 방지한다.

표 4-3 Computation Comparison with Pyramidal LK

	Pyramidal LK	Proposed
Base Algorithm	Pyramidal Iterative LK	Multi-Frame rate and Multi-Scale LK
Computation per Level	5~20 (LK + warping)	1 LK
Computation for all Levels	7.5~30 (LK + warping)	1.5 LK
Computation for n Frame Ratio	7.5~30 (LK + warping) (normal frame rate)	1.5(log n) LK
Real-time Implementation	Hard (warping)	Easy
Temporal Aliasing	Weak	Strong

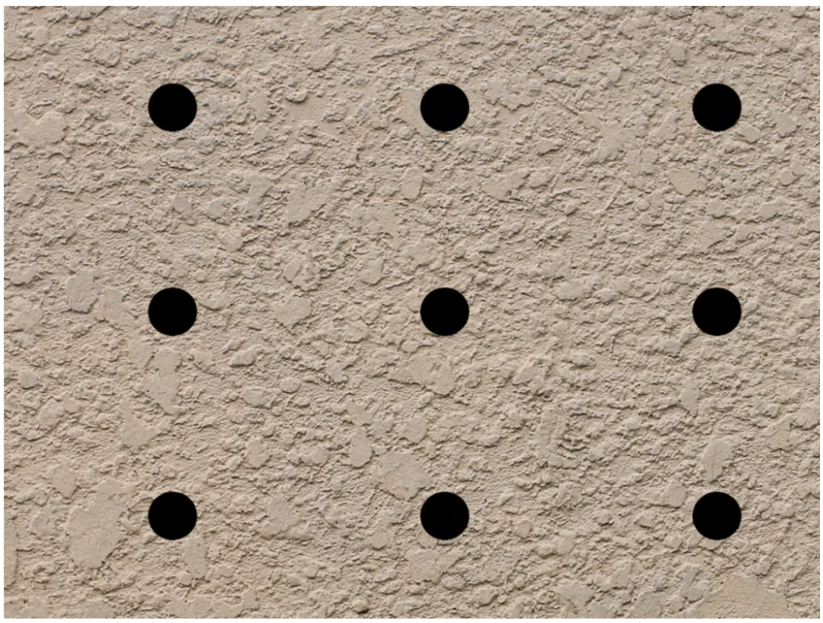
4.4.3 Graphical Result with various test sequences

이번에는 다양한 test sequence에 대해 pyramidal LK와 제안하는 optical flow의 결과를 비교하였다. 모든 pixel에 대한 optical flow결과를 보기 위해서 optical flow의 결과를 HSV color code하여 나타내었다.

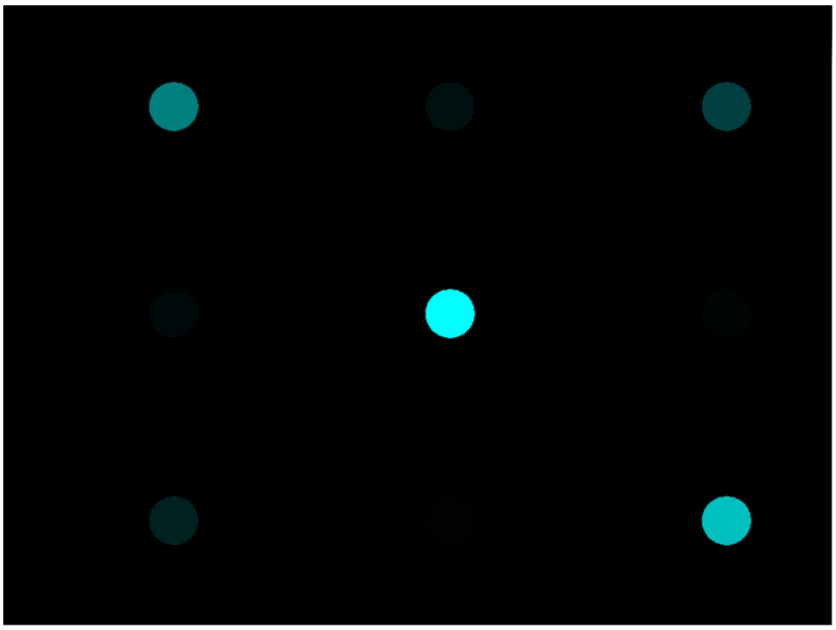
그림 4-5는 9가지의 다양한 속도로 움직이는 점들을 나타내는 test sequence이다. (a)는 각 점들은 좌측 상단부터 우측 하단 순서로 32, 4, 16, 2, 64, 1, 8, 0.5, 48 pixels/frame의 속도로 좌측으로 움직인다. 이때 frame은 normal frame rate기준이다. (b)는 해당 움직임에 대한 ground truth값을 나타낸다. 움직임이 클수록 밝기가 밝아지고, 좌측으로 움직이므로 cyan색을 갖는다. (c)는 pyramidal LK결과를 나타낸다. 빠른 움직임에 대해서 정상적인 optical flow를 구하지 못하는 것을 확인할 수 있다. 약 8 pixel/frame의 속도부터 정상적인 결과를 얻을 수 있는 것을 확인할 수 있다. 이 결과는 Pyramidal Image의 Maximum Level을 3으로 설정하였을 때 결과인데 Level을 5까지 올리더라도 개선되지 않고 오히려 더 악화되었다. (d)는 제안하는 알고리즘의 결과를 나타낸다. (b)의 ground truth결과와 비교해 볼 때, 다양한 속도로 움직이는 점들에 대해서 각각 정확한 방향과 크기를 나타내고 있다.

그림 4-6은 4K 1000FPS로 촬영한 실제 비디오 영상을 가지고 테스트한 결과이다. PHANTOM FLEX 4K 1000FPS (ULTRAHD)라는 제목으로 youtube.com에 올려져 있는 영상으로부터 test sequence를 추출하였기 때문에 ground truth값은 존재하지 않는다. 해당 영상의 27초부터 33초 사이에 오토바이가 우측으로부터 빠른 속도로 달려와서 점프하는 장면이 나오는

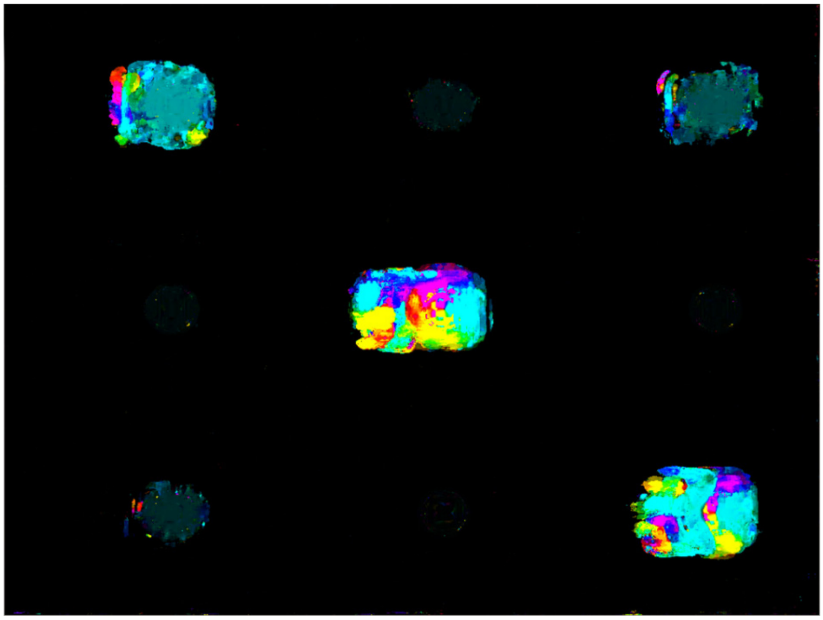
데, 카메라는 고정되어 있고 오토바이는 점프하는 순간의 영상이므로 배경의 움직임은 없고 오토바이만 좌측으로 진행하면서 약간 상승하는 방향으로 움직인다. 그림 (b)는 31.25FPS속도에 해당하는 움직임을 Pyramidal LK로 측정한 결과이다. 오토바이의 형체도 알아보기 힘들고 움직임 방향도 일관되지 않는 정상적인 결과가 아닌 것을 확인할 수 있다. (c)는 62.5FPS속도의 Pyramidal LK결과인데 이 결과 역시 형체와 방향성을 알기 어렵다. (d)는 125FPS속도의 Pyramidal LK결과인데 이 결과부터 오토바이의 형체와 움직이는 방향성을 확인할 수 있다. 하지만 많은 노이즈가 발생하는 것을 볼 수 있다. (e)는 250FPS속도의 Pyramidal LK결과이다. (d)보다 노이즈는 감소하고 형체도 좀더 개선된 것을 볼 수 있다. Frame rate이 높기 때문에 측정한 움직임의 크기가 작으므로 전체적으로 어두운 색을 갖는 것을 볼 수 있다. (f)는 31.25FPS 속도의 제안하는 Multi-Frame Rate and Multi-Scale optical flow 결과를 나타낸다. 오토바이의 형체를 명확히 확인할 수 있으며 배경과의 경계부분도 뚜렷한 것을 확인할 수 있다. 또한 움직임도 일관되게 좌측 약간 상향 방향을 가리키고 있다. 이 결과를 통해서 확인할 수 있는 것은 Pyramidal LK 를 가지고 달리는 오토바이의 움직임을 측정하기 위해서는 250 FPS의 frame rate속도로 Pyramidal LK 알고리즘을 동작시켜야 한다는 것이다. 이때 요구되는 computation은 기존 30FPS보다 8배 이상 증가하게 된다. 이 결과를 통해서 제안하는 알고리즘이 실제 비디오 영상에 대해서도 뛰어난 결과를 보여주는 것을 확인할 수 있다.



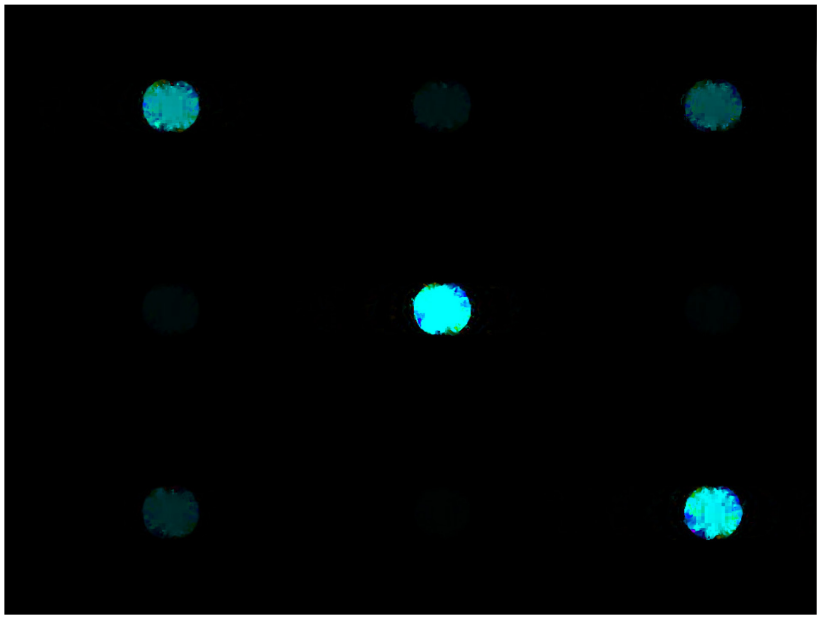
(a) Moving Dots at 9 velocities



(b) Ground Truth of Moving Dots at 9 velocities



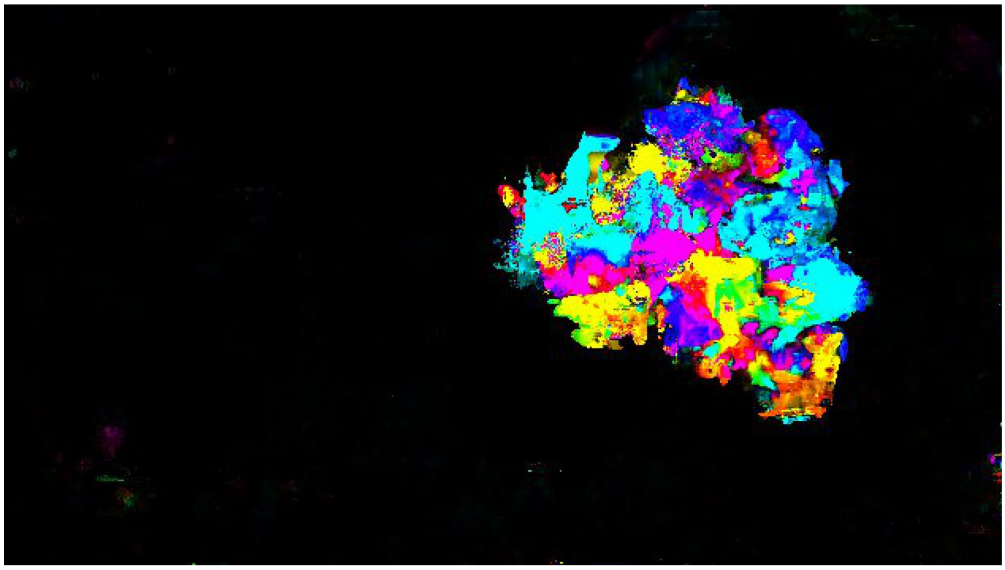
(c) Pyramid LK Result



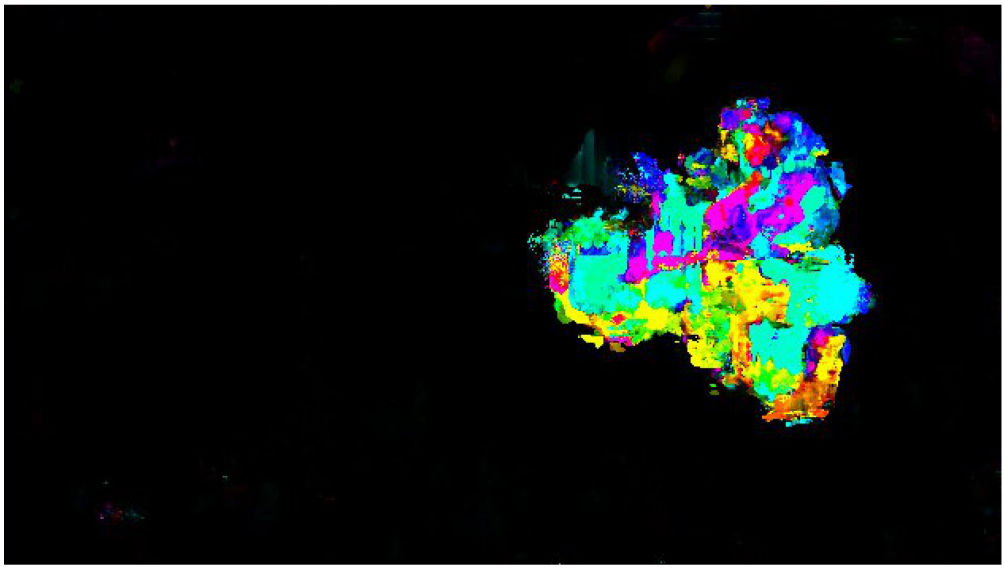
(d) Proposed Multi Frame Rate and Multi Scale Optical Flow

그림 4-5 Moving at 9 velocity

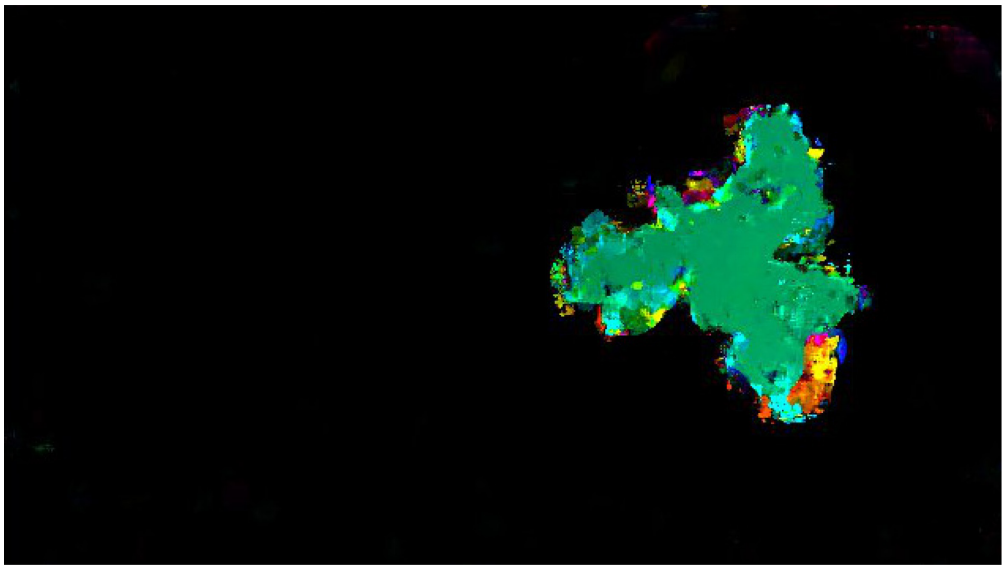
(a) 4K 1000FPS Video from youtube.com



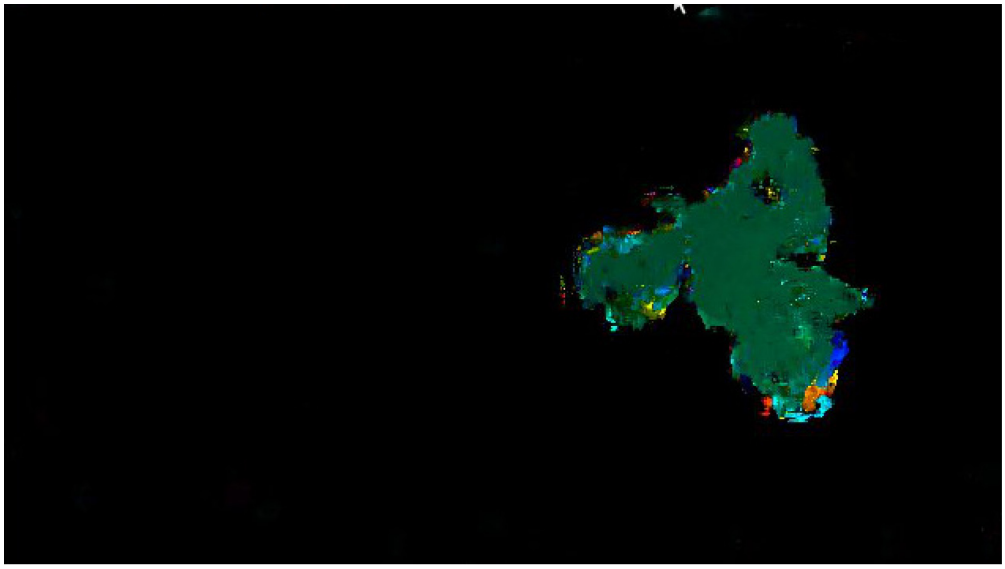
(b) Pyramidal LK at 31.25 FPS



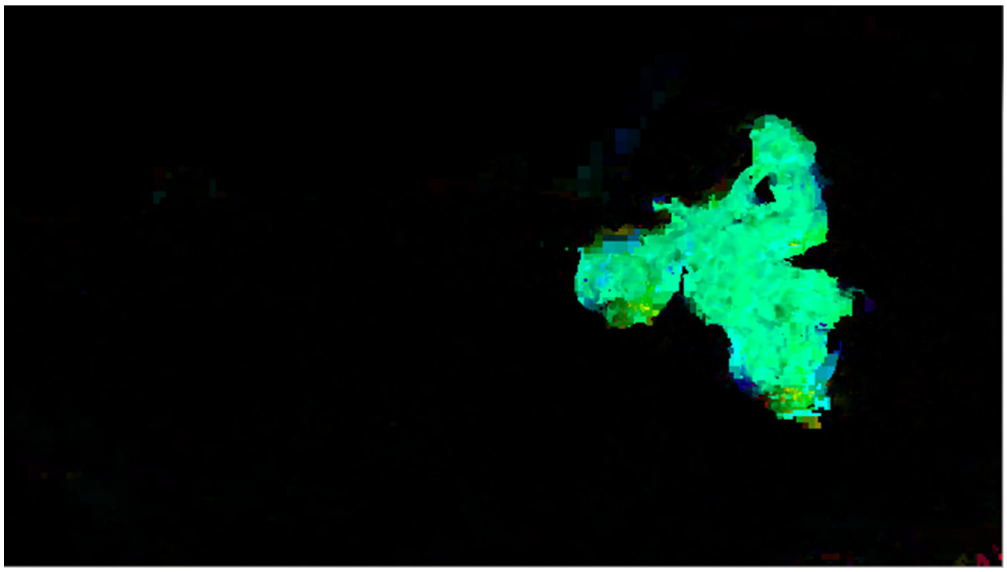
(c) Pyramidal LK at 62.5 FPS



(d) Pyramidal LK at 125 FPS



(e) Pyramidal LK at 250 FPS



(f) Proposed Multi Frame Rate and Multi Scale Optical Flow at 31.25 fps

그림 4-6 4K 1000FPS Video from youtube.com

제5장 Memory Bandwidth Reduction

5.1 Single Level LK Optical Flow System

이전 장에서는 High frame rate camera를 이용한 Multi-Frame Rate and Multi-Scale Optical Flow System을 제안하였다. Multi-frame rate and multi-scale optical flow algorithm은 multi-frame rate과 multi-level에 대한 각각의 Single level LK optical flow 결과를 가지고 최상의 결과를 얻기 위해 판단하는 selection block으로 구성된다. Selection block은 미리 정한 threshold들을 가지고 algorithm에 따라 동작하는 간단한 블록이므로 real-time implementation에 큰 문제는 없다. 그러므로 이번 장에서는 제안한 알고리즘의 Real Time Hardware Implementation을 위해 single level LK optical flow system의 문제들에 집중해서 해결하도록 한다.

5.1.1 Bandwidth Problem

Real-time system 의 구현에 관한 문제 중 하나는 external memory access bandwidth 문제이다. 예를 들어 Pyramidal LK optical flow의 real time implementation의 어려움 중 가장 큰 것은 Iterative optical flow 알고리즘의 frame memory access 문제이다. Iterative optical flow는 이전 결과에 dependent하게 동작하므로 pipeline구조를 갖기 어렵기 때문에 frame memory access latency는 real time operation에 매우 중요한 요인이 된다. 기존의 pyramidal LK hardware implementation에서는 이 문제로 인해서 embedded DRAM을 사

용하여 chip 내부에 frame buffer를 아예 포함하게 하거나 [34] external SRAM을 frame buffer를 위해 사용하여 매우 짧고 일정한 시간의 latency를 만족하게 하였다[35]. Multi frame rate and multi scale algorithm은 iterative optical flow를 사용하지 않으므로 위 문제는 발생하지 않지만, spatial and temporal Gaussian smoothing filter를 사용하므로 여러 장의 frame buffer를 사용하는 문제가 있다. 본 논문에서는 이 문제를 spatial bandwidth reduction 과 temporal bandwidth reduction 방법으로 나눠서 해결한다.

5.1.2 Matrix multiplication

LK optical flow는 least square problem 방법으로 optical flow를 구한다. 이를 위해서 window 크기의 matrix multiplication 을 수행하는데, multi-level의 optical flow를 parallel로 구현하고자 할 때는 multiplier의 개수가 증가하게 된다. 상위 레벨의 computation횟수는 픽셀 수에 비례하게 되는데, pyramid 이미지의 면적은 상위로 갈수록 1/4씩 줄어들게 되므로 상위 이미지의 면적을 모두 합치더라도 original image보다 커지지는 않는다. 그러므로 multiplier의 증가를 최소로 하면서 parallel구조를 가져갈 때는 상위 level의 연산은 time share하여 처리하고 Level0 는 독립적으로 처리하는 구조를 갖게 된다. 이런 구조의 경우 기존보다 두 배의 multiplier를 필요로 하게 된다. 다음 장에서는 약간의 buffer를 이용하여 matrix generation에 필요한 multiplier의 개수를 기존 5개에서 2개로 줄이는 구조를 제안한다.

5.1.3 FPGA Implementation

High frame rate camera가 없기 때문에 제안한 Multi-Frame Rate and multi

scale 알고리즘을 구현하기는 어렵지만 기본이 되는 Single level LK optical flow의 구현을 통하여 제안하는 Bandwidth reduction 알고리즘과 matrix multiplication 구조를 검증한다.

5.2 Spatial Bandwidth Reduction

5.2.1 LK Optical Flow System Architecture

이 subsection 에서는 Lucas-Kanade algorithm을 구현하는 HW 구조에 대해서 설명한다. 그림 5-1(a)는 기존에 사용되는 Optical Flow연산처리 구조를 보인다 [25][27]. 카메라에서 입력된 영상을 외부 메모리에 저장하고, optical flow 연산을 위해서 필요한 데이터를 읽는다. Temporal Gaussian filter 이후 spatial Gaussian filter를 적용한다. 그 결과로 smoothing 이미지가 생성된다. Smoothing을 하는 이유는 Gradient의 accuracy를 향상시킬 수 있기 때문이다[27]. Gaussian Filter Coefficient는

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

의 distribution을 갖는다. HW구현에서는 distribution equation을 대신하여 Binomial Distribution을 이용한 Simplified Gaussian Coefficient가 많이 사용된다 [31][32]. 예를 들어, [1 2 1]/4, [1 4 6 4 1]/16, [1 6 15 20 15 6 1]/64 and [1 8 28 56 70 56 28 8 1] /256와 같이 확률이 0.5일 때의 binomial coefficients를 Simplified Gaussian Kernel로 사용한다. 이때 각 Coefficient에 해당하는

Standard Deviation은 Binomial Distribution의 standard deviation으로 구할 수 있으므로 각각 √(2/4)=0.707, √(4/4)=1, √(6/4)=1.225 and √(8/4)=1.414 을 갖는다. Simplified Gaussian Coefficient는 Hardware구현 시 Shifter 와 Adder만의 조합으로 구현이 가능한 특징을 갖는다.

Gaussian Filter를 거친 결과는 X, Y, 그리고 time방향의 Gradient filter를 통해서 3가지 gradient 정보를 추출한다. 이때 사용된 Gradient Kernel은 [1 -8 0 8 -1]/12이다 [27][29]. 이 경우도 2의 지수 곱하기의 특성을 이용해서 shifter와 adder로 구현이 가능하다. Gradient 정보는 Least Square Matrix 과정에 의해서 Matrix로 연산된다. 이때 사용된 weight는 [27] 에서 사용된 separable 1D kernel인 [0.0625 0.25 0.375 0.25 0.0625] 이다. 이 weight는 Binomial Coefficient 인 [1 4 6 4 1]/16와 동일하므로 shifter와 adder를 이용해서 간단하게 HW 구현이 가능하다. 최종적으로 equation solver 단계에서는 Inverse Matrix생성 및 division을 사용한다. 마지막으로, Validation단계에서는 THeigen와 THnres를 고려하여 Optical Flow 연산 결과가 의미 있는 결과인지의 여부를 판단한다.

5.2.2 External Memory Bandwidth Requirement

Gaussian Filter의 standard deviation인 σ 값이 Optical Flow의 accuracy에 큰 영향을 끼친다고 알려져 있다[27][29]. 즉, Gaussian Filter의 σ 가 커질수록 optical flow연산 결과로 구한 motion vector의 AAE가 작아지는 경향을 보인다. 한편, Gaussian Filter의 σ 가 증가하면 Gaussian Filter의 Tap 수가 증가하게 되고, 따라서 Temporal Gaussian 처리를 위해서 외부 메모리에서 읽어와야 할 frame 의 개수도 증가한다.

표 5-1는 Gaussian Filter의 σ 값의 증가가 optical flow의 정확도 및 외부 메모리 접근량에 미치는 영향을 보여준다. Gaussian Filter은 binomial coefficients 를 이용하였고, Yosemite Test Sequence를 사용하여 AAE 를 비교하였다. 동일한 Density, 35%에서의 결과를 구하기 위해서 THeigen을 조정하고 THnres는 0.01로 고정한다[27][29]. 첫 번째 칼럼은, Gaussian Filter의 σ를 나타내고, 두 번째 칼럼은 Simplified Gaussian Kernel을 나타내고, 세 번째 칼럼은 optical flow의 AAE를 나타낸다. 네 번째는 AAE의 Standard Deviation을 나타내고 그리고 마지막으로 다섯 번째는 사용해야 하는 frame의 개수를 나타낸다. 표 5-1에서 보면 두 번째 컬럼의 Gaussian Filter σ 가 1.225가 될 때까지 Accuracy가 지속적으로 향상이 되지만, 1.225를 초과하면 어느 정도 saturation이 되는 것을 볼 수 있다. sigma를 1.225로 선택하는 경우 optical flow 결과를 얻기 위해서 11장의 frame image를 사용하여야 한다. 이렇게 많은 frame을 읽어오는 경우 외부 메모리의 bandwidth가 충분하지 않은 경우 HW 처리 속도가 dramatically 떨어질 수가 있다.

표 5-1 Average Angular Error and Frame Bandwidth with Various Simplified Gaussian Kernel

Gaussian Filter σ	Simplified Gaussian Kernel	AAE	Std. Dev.	Read Frames
0.707	[1 2 1] /4	4.11	8.35	7
1	[1 4 6 4 1] /16	3.58	8.27	9
1.225	[1 6 15 20 15 6 1] /64	3.44	8.10	11
1.414	[1 8 28 56 70 56 28 8 1] /256	3.46	8.14	13

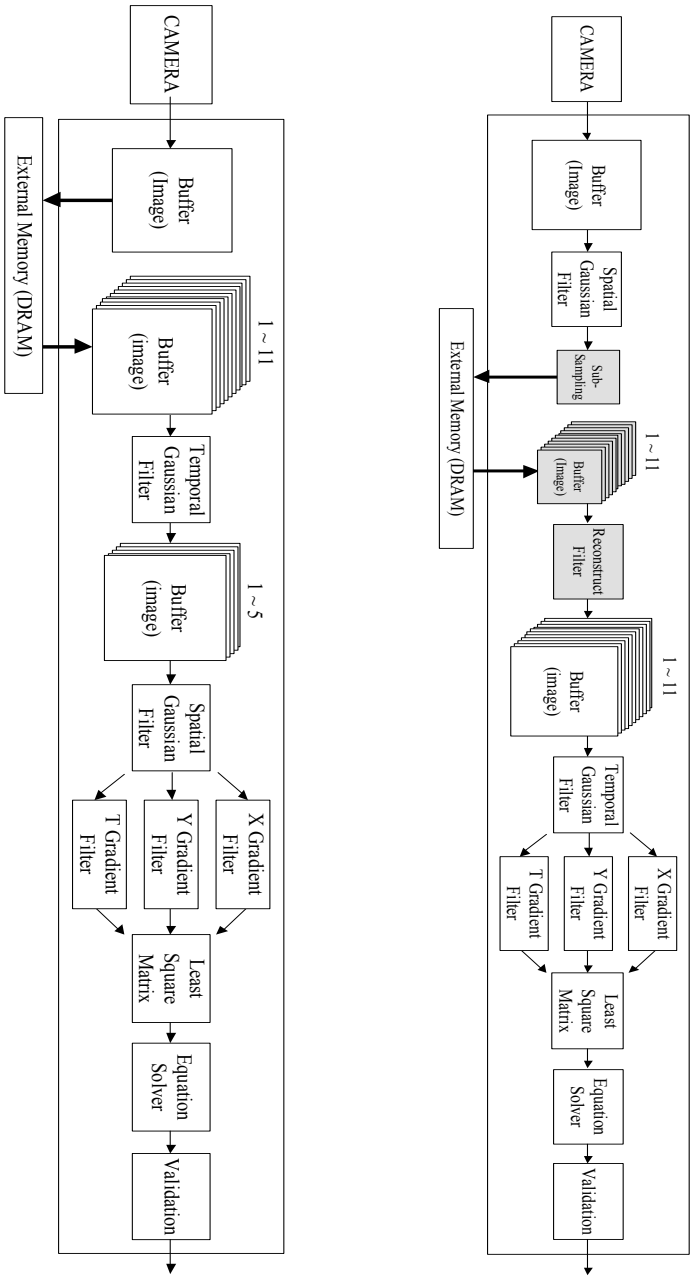
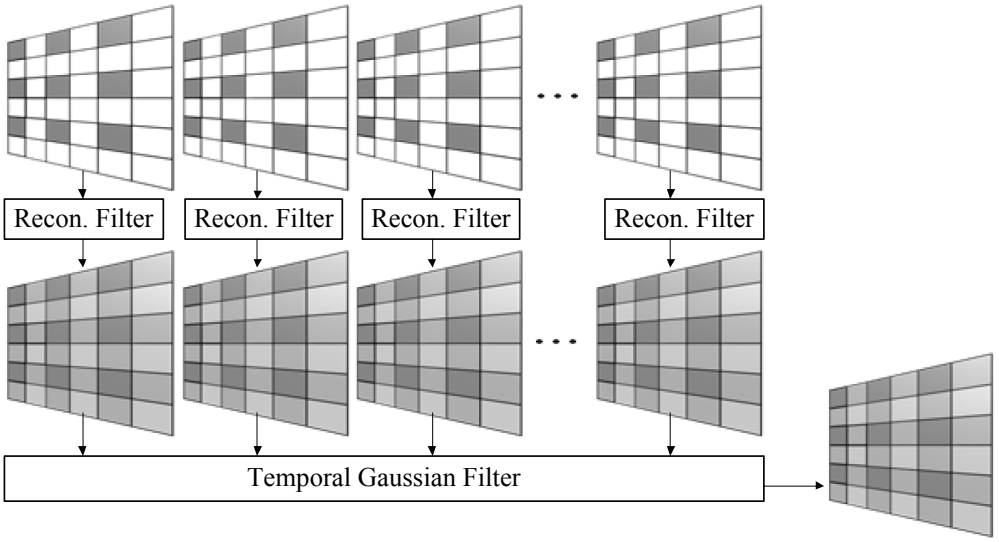


그림 5-1 Lucas-Kanade optical flow system structure

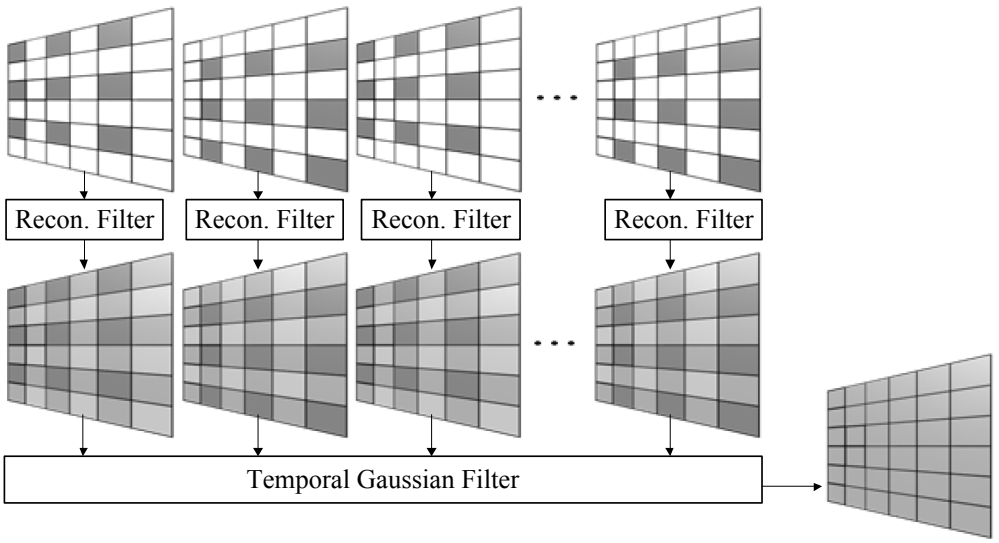
5.2.1 제안하는 알고리즘

그림 5-1 (a)와 같이 conventional optical flow 연산 처리 순서를 사용한 HW 구현할 때 발생하는 excessive 외부 메모리 접근 문제를 해결하기 위해서, 본 논문에서는 그림 5-1 (b)와 같은 새로운 연산 처리 순서와HW 구조를 제안한다. (a)와 비교하여 (b)에서 새로 추가된 sub-sampling 모듈과 reconstruction 모듈이 shaded box로 표시가 되어 있다. (a)와는 달리 (b)에서는 입력 이미지를 외부 메모리에 저장을 하는 것이 아니라, Spatial Gaussian filtering을 거친 이미지를 sub-sampling하여 외부 메모리에 저장한다. 이때 Gaussian filtering이 low pass filtering 효과를 가지므로, sub-sampling을 하여 저장을 해도 화질의 열화가 심하지 않다. 가로, 세로 방향으로 2:1 subsampling 하여 1/4로 줄어든 이미지를 저장을 함으로써 외부 메모리 접근량을 대폭 줄일 수 있게 된다. Subsampling된 데이터를 외부 메모리에서 읽어온 후에 reconstruction filter를 사용하여 원래 영상과 같은 크기의 이미지를 생성한 후 Temporal Gaussian filtering 적용함으로써 원하는 optical flow 결과를 생성할 수 있다. (a)와 비교하여 spatial, temporal Gaussian filtering의 순서가 바뀌게 된다. 그러나, Gaussian Filter는 Linear Filter이므로 처리 순서가 바뀌더라도 동일한 결과를 생성한다. 이렇게 외부 메모리에 저장할 때에 subsampling한 이미지를 저장함으로써 메모리 밴드위스를 significantly 줄일 수가 있게 된다. Spatial 방향의 Gaussian Filter 처리가 된 결과는 Low pass filter 처리된 결과가 되므로 Gaussian Filter의 σ가 커짐에 따라서 aliasing이 줄어들게 된다. 하지만 표 5-1에서와 같이 Gaussian Filter의 σ 값의 증가는 Frame Bandwidth의 증가를 유도하므로 subsampling을 고

려한 suitable Gaussian Filter σ 의 선택은 Reconstruction Filter와 Temporal Gaussian 등을 포함한 전체 optical flow system에 의한 Accuracy와 Frame Bandwidth 간의 trade-off 를 고려해서 정한다.



(a)



(b)

그림 5-2 Reconstruction Filter and Temporal Gaussian Filter Operation

그림 5-2에서는 Subsamplng되어 외부 메모리에 저장되었던 프레임들이 Reconstruction Filter 및 Temporal Gaussian Filter로 처리되는 과정을 보여주고 Frame Bandwidth의 증가를 최소로 하면서 Accuracy를 최대로 높이기 위해 제안하는 interleaved subsampling 방법을 설명한다. 그림 5-2 (a)는 Conventional Subsampling 방법을 보여준다. 모든 프레임에 대해서 x축, y축 방향으로 Even (Even, Even) 또는 Odd (Odd, Odd)의 동일한 위치에서 Sub-Sampling 한다. 그림 5-2 (b)는 제안하는 Interleaved Subsampling 방법을 보여준다. 프레임마다 sample하는 픽셀의 위치를 (Even, Even)위치와 (Odd, Odd)위치를 번갈아 가며 선택하는 방법이다. Conventional sub-sampling의 경우 모든 프레임에 대해서 동일한 위치의 픽셀이 Sampling되기 때문에 Reconstruction된 픽셀들의 위치도 모든 프레임에 대해서 동일하다. Reconstructed 프레임의 (Odd,Even) 와 (Even,Odd) 위치는 (Even,Even) 위치에서 Sample된 픽셀을 가지고 x축, y축 1-dimensional Reconstruction Filter 처리를 통해서 생성되는데, 이때 Aliasing과 Non-Ideal Filter로 인해 Data Loss가 발생한다. (Odd,Odd)위치의 픽셀은 Data Loss를 갖는 (Odd,Even) 또는 (Even,Odd) 위치의 픽셀을 이용해서 Reconstruction Filter처리를 해야 하므로 Data Loss는 더욱 커지게 된다. Temporal Gaussian filter또한 동일한 위치의 픽셀 사이에서 이루어지므로 reconstruction된 pixel들의 Data loss가 Temporal Gaussian 이후에도 전달되어 불균등은 더욱 심화된다. 그림 5-2 (b)의 proposed interleaved sub-sampling에서 역시 reconstructed filter의 결과는 conventional sub-sampling과 발생 위치만 바뀔 뿐, 각 프레임의 Data loss 불균등은 동일하게 발생한다. 하지만, Spatio-Temporal의 3D 공간에서 보면

프레임들 간에 Sample되는 픽셀의 위치와 reconstructed되었을 때 가장 Data loss가 심한 위치가 t축 방향으로 인접하게 된다. 따라서, 모든 Reconstruction된 픽셀은 Sub-Sample된 픽셀과 Spatially 또는 Temporally 인접하는 구조를 갖는다. 결과적으로 proposed interleaved sub-sampling은 Temporal Gaussian Filter에 의해서 Reconstruct된 픽셀 값의 Data loss 불균등을 줄어들게 한다.

5.2.2 Simulation Result

그림 5-3는 다양한 Gaussian Filter σ 값에 대하여 Subsampling과 Reconstruction filter를 사용하였을 때와 그렇지 않았을 때의 Gaussian filter 결과를 비교하였다. 0.707, 1, 1.225, 1.414, 그리고1.581의 σ 갖는 Gaussian Filter를 사용하고, Reconstruction Filter로 Cubic Convolution(4 tap), Lanczos2 Filter(4 tap) 그리고 Lanczos3 Filter(6 tap)를 적용하였다. Yosemite 를 Test Sequence로 사용하였다. 가로축은 Gaussian Filter σ 값을, 세로축은 두 영상 간의 PSNR(Peak Signal to Noise Ratio)을 나타낸다. Dash curves represent the conventional Subsampling, whereas solid curves represent the proposed interleaved subsampling. In both cases, Gaussian Filter σ값이 커짐에 따라서 Aliasing으로 인한 Reconstruction Error가 줄어들기 때문에 PSNR 결과가 커지는 경향을 확인할 수 있다. 3개의 reconstructed filter 중에서는 Lanczos3 Filter의 PSNR 결과가 가장 좋음을 확인할 수 있는데, Lanczos3 Filter가 가장 많은 tap을 가지고 Ideal Reconstruction Filter인 sinc function을 이용해서 설계되었기 때문이다. 그림 5-3에서 볼 수 있듯이, the proposed Interleaved Subsampling 의 PSNR결과가 Gaussian Filter σ 값이나 reconstructed filter종류에 상관 없이

conventional subsampling 보다 평균 2dB 향상되었음을 알 수 있다.

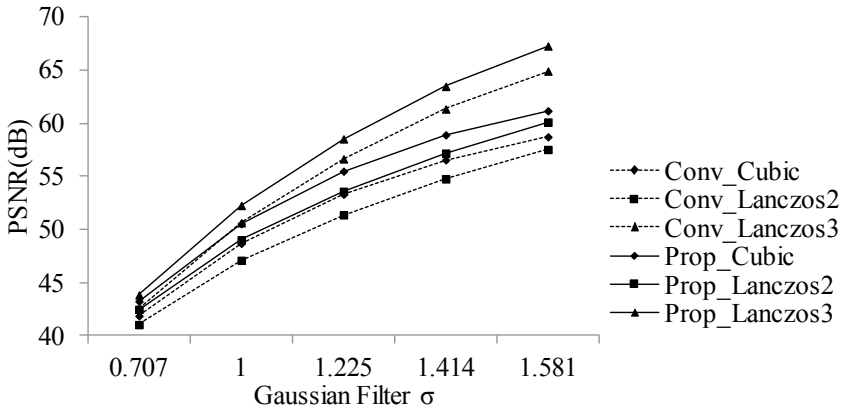


그림 5-3 PSNR result with Cubic, Lanczos2 and Lanczos3 Reconstruction Filters

표 5-2는 픽셀의 Even/Odd 위치에 따른 conventional과 interleaved subsampling의 PSNR값을 비교한 것이다. Subsampling, Reconstruction filter 그리고 Temporal Gaussian filter를 사용한 결과와 Subsampling을 하지 않은 Original Image를 Temporal Gaussian Filter한 결과를 비교하여 PSNR값을 얻는다. Reconstructed filter는 Lanczos3를 사용하였다. 첫번째 column은 subsampling 방법을 두 번째 column은 reconstructed frame의 픽셀 위치를 나타낸다. (E,E), (O,E), (E,O) 그리고 (O,O)의 E는 Even 좌표의 위치를, O는 Odd 좌표의 위치이다. From third to seventh columns, 0.707, 1, 1.225, 1.414, 그리고1.581의 σ 갖는 Gaussian Filter를 사용하였다. Conventional Subsampling의 결과에서 (E,E)위치의 값은 Sample된 original 픽셀과 동일하므로 PSNR 값은 infinite이 된다. 모든Gaussian Filter σ 값에 대해서 동일하게 (O,O)위치의 PSNR 값은 (O,E) 와 (E,O)위치의 PSNR 값보다 작음을 확인할 수 있다.

그림 5-2에서 설명했듯이, (O,E) 와 (E,O)위치의 픽셀은 Sample된 Pixel인 Original 픽셀을 이용해서 reconstruct되지만, (O,O)픽셀은 (O,E)와 (E,O)위치의 reconstruct된 픽셀을 이용해서 구하기 때문이다. 한편, 본 논문에서 제안하는 Interleaved Subsampling 결과를 보면 (E,E), (O,E), (E,O) 그리고 (O,O) 위치의 PSNR값이 균등할 뿐 아니라 conventional subsampling과 비교하여 개선된 결과를 보여준다. 이 차이는 Interleaved Subsampling에 의해서 sample된 original 픽셀들이 (E,E)와 (O,O)위치에 번갈아 가면서 위치하게 되므로 Temporal Gaussian Filter 에 의해서 reconstruction시에 발생한 data loss가 temporal 방향으로 인접한 original 픽셀에 의해서 보상되기 때문이다.

표 5-2 PSNR of Reconstructed Pixels with Lanczos3 Filter

PSNR	Position	Gaussian filter Sigma				
		0.707	1	1.225	1.414	1.581
Conventional Subsampling	(E,E)	Inf.	Inf.	Inf.	Inf.	Inf.
	(O,E)	42.80	51.09	57.29	62.09	65.67
	(E,O)	42.33	50.25	56.14	60.84	64.55
	(O,O)	39.63	47.62	53.62	58.27	61.77
	All	42.60	50.64	56.65	61.35	64.92
Interleaved Subsampling	(E,E)	45.35	52.50	58.49	63.40	67.25
	(O,E)	43.36	51.95	58.34	63.37	67.24
	(E,O)	44.01	52.44	58.69	63.58	67.33
	(O,O)	42.67	51.95	58.53	63.53	67.30
	All	43.74	52.20	58.51	63.47	67.28

그림 5-4는 제안된 interleaved subsampling 기법을 사용한 AAE결과를 보

여준다. Reconstruct filter로는 Lanczos3 filter를 사용하였다. 가로축은 Gaussian filter σ 의 값을, 세로축은 AAE를 나타낸다. 공정한 비교를 위하여 Density를 35%로 맞추기 위해서 THeigen을 조정하였고 THnres=0.01로 고정하였다. Black graph denoted by Original은 Subsampling 을 하지 않은 그림 5-1 (a) 시스템의 결과이며, light gray와 dark gray graphs represent conventional and interleaved subsampling, respectively. Subsampling된 AAE 결과는 Gaussian filter σ가 커질수록 Original AAE결과로 수렴하는 경향을 보인다. Interleaved Subsampling AAE 결과의 경우는 Gaussian filter σ가 1.225에서부터 Original AAE결과의 2.77 % 로 수렴하게 된다. 하지만 conventional Subsampling의 경우 Gaussian filter σ가 1.225일 때에는 Reference AAE결과의 9.13 %차이를 보이고 1.414가 되어서야 2.63% 로 수렴하게 된다. AAE값으로 비교하면, Interleaved Subsampling의 Gaussian filter σ가 1.225 일 때의 AAE값 3.53는 Conventional Subsampling을 사용해서 Gaussian filter σ 가 1.414일 때의 AAE 값 3.56보다도 작다. 표 5-1에서 볼 수 있듯이, Gaussian Filter σ 의 증가는 Frame Bandwidth를 증가시키므로 동일한 AAE를 나타내는 Interleaved Subsampling과 Conventional Subsampling방법간의 Gaussian filter σ 차이는 2 Frame 만큼의 bandwidth reduction 효과가 있음을 확인할 수 있다. 본 논문의 구현에서는 Gaussian filter σ 증가에 따른 AAE Saturation정도를 고려해서 σ=1.225로 정한다.

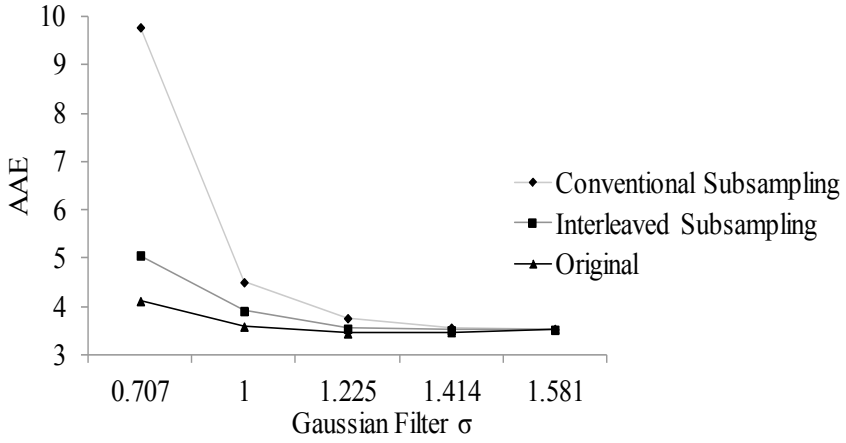


그림 5-4 AAE Comparisons at 35% Density

그림 5-5에서 가로축은 subsampling한 픽셀당 할당되는 number of bits를, 세로축은 그에 따른 AAE값을 보여준다. 32 bit floating point로 저장하는 경우 AAE는 3.53이다. Bandwidth를 줄이기 위해서fixed point로 저장을 하면서, 각 pixel당 사용하는 bit수를 줄일수록 AAE는 증가한다. Pixel당 사용하는 bit수가 9bit인 경우 AAE는 3.57, 8bit 인 경우 3.64를 갖는다. 한 pixel당 8bit로 저장을 하는 경우 floating-point operation의 결과와 어느 정도 근접할 결과를 얻을 수 있으며, 특히 memory addressing을 효과적으로 할 수 있는 장점이 있다. 따라서, 본 논문의 구현에서는 8bit로 선택하였다.

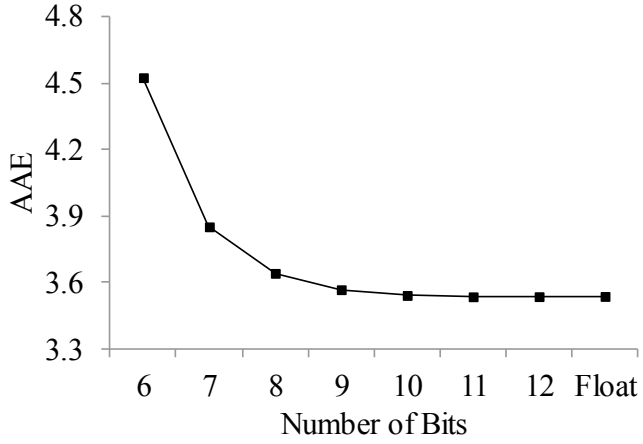


그림 5-5 Save Bits vs. AAE

(Yosemite with cloud test sequence is used and THeigen is manipulated for 35% Optical Flow Density)

표 5-3는 다양한 Input sequence를 가지고 Reference SW[27][29] 의 결과와 Interleaved Subsampling이 적용된 Proposed SW 결과를 비교하였다. σ는 1.225인 simplified Gaussian kernel을 사용하였고 THeigen=2 그리고 THnres=0.01이 사용되었다. Proposed SW에서 Subsampling후 외부 메모리에 저장할 때의 the number of bits per a pixel은 8bit 로 하였다. 첫번째 column 은 실험에 사용된 test sequence이다. From second to fourth columns, AAE, std dev, density from the reference SW are shown, whereas AAE, std dev, density from the proposed SW are shown from fifth to seventh columns. Reference SW 결과와 비교해 볼 때, 평균적으로 0.244의 AAE 증가가 발생하지만 subsampling으로 인한 75%의 Frame Bandwidth Reduction을 고려하면 충분히 무시할 수 있는 수준이다.

표 5-3 AAE with Various Test Sequences

Input Sequence	Reference SW			Proposed SW		
	AAE	Std. Dev.	Density (%)	AAE	Std. Dev.	Density (%)
Yosemite	3.44	8.10	35.35	3.65	8.21	36.33
Trans Tree	0.77	0.69	39.25	1.60	1.69	41.45
Div Tree	1.84	1.80	47.10	1.93	1.82	51.00
Sinusoid1	2.47	0.10	100	2.48	0.38	100
Square2	0.28	0.21	6.26	0.36	0.33	8.89

5.3 Temporal Bandwidth Reduction

이번 장에서는 external memory access bandwidth의 temporal bandwidth reduction을 위해서 두 가지 접근 방법을 제안한다. 첫 번째는 LK optical flow system의 연산과정을 분석하여 bandwidth를 증가시키는 요인을 찾고, optical flow system의 accuracy의 영향을 최소화하면서 bandwidth를 감소시킬 수 있는 방법을 제안하는 것이고, 두 번째 방법은 system 적으로 접근해서 hardware resource 사용은 최소로 하면서 data reuse를 통해서 bandwidth를 감소시킬 수 있는 방법을 제안한다.

5.3.1 Tail Cut

LK optical flow 처리시 읽어 들여야 하는 frame 수에 영향을 끼치는 것은 temporal filters에 의해서 이다. 즉, Temporal Gaussian filter 와 temporal gradient filter의 tap수가 증가하는 것은 읽어 들여야 하는 frame 수가 증가

하는 것을 의미한다. Temporal Gaussian filter와 temporal gradient filter의 관계를 분석하기 위해서 기존 system의 filter 처리 순서에서 spatial Gaussian filter와 temporal Gaussian filter의 순서를 바꾸었다. Gaussian Filter 는 FIR filter로 Linear system인 convolution으로 구현되어 교환법칙이 성립하므로 temporal Gaussian과 spatial Gaussian의 순서가 바뀌어도 동일한 결과를 얻는다. 아래 그림 5-6은 이렇게 변경된 구조를 보여준다.

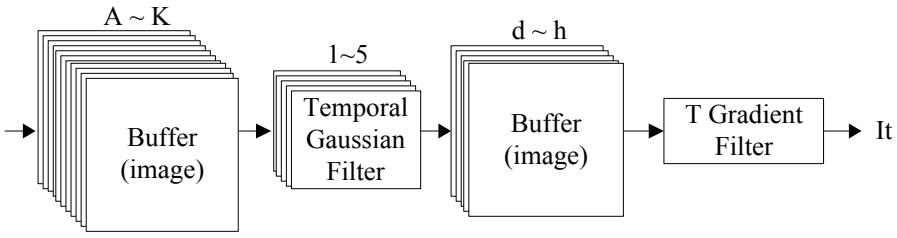


그림 5-6 Temporal Gaussian & Gradient Filter

이 시스템의 결과인 It는 Gaussian filter와 gradient filter의 convolution을 통해서 구하게 되므로,

$$[1\ 6\ 15\ 20\ 15\ 6\ 1]/64 * [1\ -8\ 0\ 8\ -1]/12$$
$$= [1\ -2\ -33\ -92\ -98\ 0\ 98\ 92\ 33\ 2\ -1]/8192$$

와 같이 하나의 filter로 표현할 수 있다. 이 결과는 각 frame에 적용되는 coefficient는 중간 값인 f frame의 coefficient 가 zero인 것을 제외하면 중간으로 갈수록 coefficient의 절대값이 커지고, 양 끝으로 갈수록 작아진다. A와 K에 해당하는 coefficient는 가장 큰 값을 갖는 E와 G와 비교할 경우 1.02 % 밖에 되지 않고, 전체 coefficient 중에서는 0.22 % 밖에 되지 않는다. 결국 A와 K frame은 다른 frame들과 비교할 때, 최종 결과에는 거의

영향을 끼치지 않으면서도 사용하는 External Memory Bandwidth는 전체 Bandwidth 중에서 18.18% 나 차지하고 있다.

표 5-1에서 볼 수 있듯이, Optical flow에서 사용되는 Frame Bandwidth는 Gaussian filter의 sigma에 의해서 영향을 많이 받은 것을 확인할 수 있다. Frame bandwidth를 줄이기 위해서 sigma를 줄이거나, 동일한 sigma에 대해서 tap 수를 일괄적으로 줄일 경우 Accuracy가 감소하게 된다. 하지만 It를 만드는 최종 coefficient 중 가장 영향력이 작은 coefficient를 없애면 [0 -2 -33 -92 -98 0 98 92 33 2 0]/8192 의 coefficient를 갖게 되어 2 tap이 줄어든 filter를 갖게 된다. 이 방법을 이용할 경우 optical flow system의 accuracy의 감소를 최소로 하면서 2 frame의 Bandwidth를 줄일 수 있게 된다.

하지만 Filter 계수를 바꾸는 것은 filter의 주파수 특성뿐 아니라 phase 특성이 바뀌게 되어 filter 처리된 결과는 이미지 왜곡이 발생할 수 있다. 이런 필터특성 변화로 인한 이미지의 왜곡을 최소로 하기 위해 단순히 최종 coefficient의 tap을 제거하는 것이 아니라, 각 filger의 단계별로 분석한다. 그림 5-6에서 보여주고 있는 Temporal Gaussian filter와 temporal gradient filter의 과정을 단계별로 살펴보면, Buffer A~K는 spatial filter를 거친 결과이고, temporal Gaussian filter 1~5에 의해서 Buffer d~h 가 만들어진다. 중간 버퍼인 d~h는

$$d = (A + 6B + 15C + 20D + 15E + 6F + G) /64$$
$$e = (B + 6C + 15D + 20E + 15F + 6G + H) /64$$
$$f = (C + 6D + 15E + 20F + 15G + 6H + I) /64$$
$$g = (D + 6E + 15F + 20G + 15H + 6I + J) /64$$

$$h = (E + 6F + 15G + 20H + 15I + 6J + K) /64$$

가 된다. It는 buffer d~h를 이용해서 t gradient filter를 통해서 구한다.

$$It = (d – 8e + 0 + 8g – h) /12$$
$$= (A – 2B – 33C – 92D – 98E + 0F + 98G + 92H + 33I + 2J – K) / 8192$$

It는 A~K frame을 가지고 temporal Gaussian 과 temporal gradient filter를 hardware 구조로 적용한 결과인데, 위에서 [1 -2 -33 -92 -98 0 98 92 33 2 -1]/8192 의 FIR filter를 적용한 것과 같은 결과를 확인할 수 있다. 이 과정을 통해서 보면, 왜 A와 K frame이 작은 coefficient를 갖는지가 설명된다. Frame A와 K는 temporal Gaussian을 거쳐서 각각 d와 h를 만드는데, 이 과정에서 frame A와 K는 Gaussian filter의 가장 작은 coefficient가 곱해진다. d부터 h는 temporal gradient filter를 거쳐서 It를 만드는데, 이 과정에서 A와 K가 적용된 d 와 h에는 0을 제외한 가장 작은 coefficient가 곱해진다. 결과적으로 A와 K frame은 Gaussian filter와 gradient filter의 가장 작은 coefficient가 연속으로 곱해지기 때문에 전체 coefficient 중에서 0.22 % 밖에 되지 않는 weight를 갖는다. 만약 가장 작은 weight를 갖는 A, K를 제거해서 요구되는 bandwidth를 줄인다면, d와 h를 구하기 위한 Gaussian coefficient는 각각 [0 6 15 20 15 1] 과 [1 6 15 20 15 0]이 된다. 이는 좌 우 비대칭의 FIR filter가 되므로 Non-linear 특성을 갖기 때문에 이미지 왜곡이 발생한다. 그러므로 d와 h를 위한 Linear phase FIR filter를 구현하기 위해서는 coefficient로 [0 6 15 20 15 0]을 사용한다. 이 결과를 적용한 결과는 아래

와 같다.

$$It = (d - 8e + 0 + 8g - h) /12$$

$$= (0A - 2B - 33C - 92D - 97E + 0F + 97G + 92H + 33I + 2J - 0K) / 8192$$

결과적으로 weight가 가장 작은 A, K는 제거되었고 Linear phase를 갖는 FIR filter를 구현하였고, Frame 수는 처음 11개에서 2개가 줄어 든 9개만 필요하게 된다. 이 결과는 표 5-1에서 볼 수 있듯이, Gaussian sigma가 1일 때 요구되는 Frame 수와 동일하며 coefficient 는 Gaussian Sigma 가 1.225 일 때와 유사한 Accuracy를 가질 것으로 기대된다. 제안된 방식에 의해서 수정된 filter의 frequency response, phase characteristic, 그리고 optical flow accuracy 분석은 section 4 experimental result 부분에서 설명한다.

5.3.2 Frame Reuse

그림 5-7의 (a)는 6번 frame의 optical flow 연산처리를 위해 1번부터 11번까지의 frames을 외부 메모리로부터 읽어오는 그림 5-1 (a)의 conventional system의 처리과정을 보여준다.

6번 frame의 optical flow 연산처리가 끝나면 다음 frame인 7번 frame의 optical flow를 구하기 위해 다시 2번부터 12번까지 11장의 frame을 읽어온다. 이러한 방식으로 8번, 9번 frame의 결과를 위해서 각각 3번부터 13번, 4번부터 14번의 frame을 읽어온다. 즉, 매 frame마다 읽어오는 11개의 frame중 10개의 frame은 이전 frame 연산과정에서 동일하게 사용된 frame 이다. 이처럼 conventional system의 구조는 frame데이터를 overlap하게 읽어오는 구조이기 때문에 external memory access bandwidth를 낭비하는 문제가

발생한다.

이 문제를 해결하기 위해서 본 논문에서는 external memory에서 읽어 온 overlap되는 frame을 이용해서 1장이 아닌 multiple frame의 optical flow 연산을 처리할 수 있는 hardware 구조를 제안한다. (b)와 (c)는 읽어 들인 frames을 이용해서 two frames의 optical flow를 처리하는 과정의 예를 보여준다. 한번에 6번과 7번에 대해 optical flow처리하기 위해 1번부터 12번 frame의 데이터를 읽어온다. 연산을 시작하는 시점은 12번 frame이 준비된 이후에야 가능하므로 (a)의 6번 frame이 처리를 시작하는 시점보다 1 frame delay되어 시작된다. 두 frame에 대한 연산은 서로 독립적으로 처리가 가능하여 parallel또는 serial하게 처리가 가능하다. Parallel하게 처리하면 두 frame의 연산을 1 frame 시간 동안 처리하고 serial 하게 처리하면 2 frame의 연산을 2 frame 시간 동안 처리한다.

Parallel처리를 보여주는 그림이 (b)이다. 이런 구조의 경우 optical flow를 처리하는 resource가 두 배로 늘어나게 되는 단점이 있지만, latency 증가는 1 frame 밖에 되지 않는다. (c)에서는 serial 처리 구조를 보여주며, 이런 구조는 hardware resource 증가를 최소로 할 수 있는 장점이 있지만, 동시처리를 위해 latency는 2 frame으로 늘어난다는 단점이 있다.

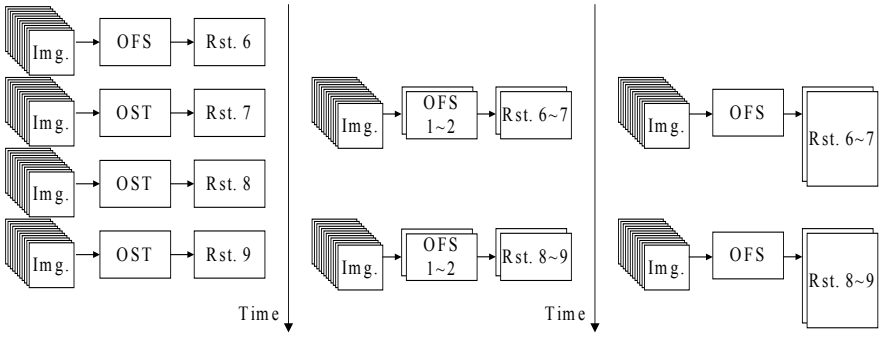


그림 5-7 Frame reuse operation

아래 표 5-4은 multi frame reuse를 사용할 때, memory access bandwidth와 latency의 관계를 보여준다. Gaussian sigma는 1.225일 때를 가정한다. 첫 번째 column은 한번에 처리하는 frame의 수를 나타내고, 두 번째 column은 하나의 frame 처리당 필요한 required frame bandwidth를 나타낸다. Gaussian sigma가 1.225이므로 1 frame처리의 경우 11장의 frame이 필요하고, 한번 읽어서 처리하는 frame수가 늘어날수록 frame당 required frame bandwidth는 줄어드는 것을 확인할 수 있다. 1개의 frame의 optical flow를 처리하기 위해서는 11개의 frame을 사용하데, 2, 3 frame 을 동시에 처리하면, 사용 frame 수는 12 frames와 13 frames가 되므로 기존의 22개에서 각각 45.5%와 60%가 감소하는 것을 확인할 수 있다. 세 번째 column은 parallel처리 구조를 가질 때의 latency를 나타내고 네 번째 column은 serial처리 구조를 가질 때의 latency를 나타낸다.

기존 구조의 경우 6 frame의 latency를 갖지만, 2 frame 처리를 하는 경우 parallel 구조는 7 frame, serial 구조는 8 frame으로 동시처리 frame의 수가 늘어날수록 latency 도 증가하는 것을 확인 할 수 있다.

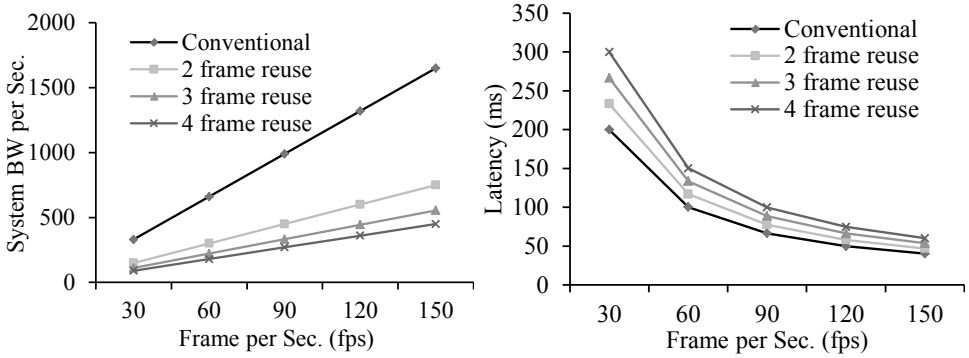
표 5-4 Bandwidth Reduction and Latency

Number of Frame calculation	Read frame BW per frame	Reduced frame bandwidth	Parallel Latency	Serial Latency
Normal	11	0	6	6
2 frames	12/2	45.5 %	7	8
3 frames	13/3	60.6 %	8	9
4 frames	14/4	68.2 %	9	10

그림 5-8은 original, 2, 3 그리고 4 frame reuse 를 사용할 때 frame rate의

변화에 따른 required bandwidth 와 Latency 의 관계를 보여준다. (a)는 frame rate이 증가에 따른 요구되는 system bandwidth per second를 frame 수로 보여준다. x축은 다양한 frame rate을 나타내고 y축은 1초당 요구되는 frame bandwidth를 나타낸다. 표 5-4 에서 알 수 있듯이, Conventional 방법의 frame당 요구되는 BW가 가장 크므로 System BW per second도 frame rate이 증가할수록 가장 크다. 하지만, frame reuse가 사용된 경우는 frame rate이 증가하더라도 reuse값이 커질수록 요구되는 system BW per second 의 값이 가파르게 감소할 뿐 아니라 frame rate 증가에 따른 system BW 의 증가 기울기도 줄어들고 있음을 확인할 수 있다. Conventional 구조의 경우 30 fps 의 frame rate일 때는 330 frame을 1초간 읽어오면 되지만, 90 fps의 frame rate일 때는 frame rate이 늘어난 비율 그대로 990 frame을 읽어와야 한다. 이런 구조를 가지고 frame rate을 증가시키는 경우, 전체 system의 cost가 상승하는 결과를 초래한다. 이에 반해서 3 frame reuse 구조를 사용하면 90 fps frame rate을 위해서는 333 frame을 1초간 읽어오면 된다. 이 값은 conventional 구조에서 30 fps의 경우와 거의 동일한 값으로, 기존 system에서 3 frame reuse 구조를 사용하면 추가적인 cost 상승 없이 3배의 frame rate을 지원할 수 있다. (b)는 frame rate의 증가에 따른 optical flow calculation latency 를 시간단위로 보여준다. x축은 frame rate의 변화를, y축은 latency를 나타낸다. 표 5-4 에서 보면 frame reuse 횟수가 증가할수록 frame 단위의 latency가 증가하지만, frame rate증가로 인해 one frame period 가 짧아지는 정도가 훨씬 크기 때문에 시간단위의 latency는 오히려 급격하게 줄어드는 것을 확인할 수 있다. 이처럼 frame reuse구조를 normal

frame rate에서 사용할 경우 latency가 증가하게 되어 system의 응답속도가 늦어지게 되지만, high frame rate을 이용할 경우 latency가 급격히 감소하게 된다. 일반적으로 가장 빠른 latency는 2 frame 시간 인데, 1 frame시간은 현재 frame 이후, 최소 1 frame의 이미지가 필요하기 때문이고 또 다른 1 frame 시간은 optical flow가 연산되는 시간이다. 30 fps 의 frame rate의 경우, 2 frame latency의 시간이 66.7 ms인 것을 감안할 때, 90 fps에서 2 frame reuse를 사용했을 때의 latency가 77.7 ms를 나타내는 것은 real-time application에 적합한 수준이라는 것을 의미한다. 본 논문에서는 이와 같은 required system bandwidth, latency, hardware cost, frame rate등의 관계를 고려하여 2 frame reuse를 사용하는 serial 구조를 선택하여 optical flow를 hardware를 구현한다.



(a) (b)

그림 5-8 System Bandwidth & Latency for various Frame Rate

5.3.3 Experimental Result

Gaussian Sigma 가 1.225 일 때, Coefficient 의 단순 양쪽 tail을 제거할 경우와 제안한 방법으로 tail을 제거하는 경우의 filter특성에 어떤 영향을 끼

치는지 확인하기 위해서 frequency and phase response를 분석한다. 그림 5-9에서는 Original, 단순 tail cut, 그리고 제안하는 방법의 tail ct에 의한 temporal Gaussian and gradient filter의 frequency and phase response를 보여준다. Original, 단순 tail cut 그리고 proposed tail cut scheme은 각각 line, dotted line 그리고 dashed line으로 보여준다. (a)와 (b)는 temporal Gaussian과 gradient filter의 과정을 frequency 와 phase response를 보여준다. Original scheme은 [1 -2 -33 -92 -98 0 98 92 33 2 -1]/8192 kernel, 단순 tail cut scheme은 [0 -2 -33 -92 -98 0 98 92 33 2 0]/8192 kernel, 제안하는 tail cut scheme은 [0 -2 -33 -92 -97 0 97 92 33 2 0]/8192 kernel이 사용된 결과이다. Gaussian filter와 gradient filter의 space domain에서의 convolution 된 결과인데, 이는 frequency domain에서는 곱의 결과이므로, (a)의 frequency response는 두 filter의 frequency response 간의 곱의 결과로 이해할 수 있다. Dashed and dotted line은 tail cut 으로 인해 frequency response가 변형되어 high frequency ripple이 발생한 것을 볼 수 있다. Proposed tail cut scheme 결과는 -50dB이하, simple tail cut scheme 은 -35dB정도의 high frequency ripple이 있음을 확인할 수 있다. (b)는 original, simple tail cut 그리고 proposed tail cut scheme의 phase response를 나타낸다. Simple tail cut의 phase response는 original 과 proposed tail cut phase response보다 shift 되었지만, 동일하게 linear phase response를 보인다. (c)와 (d)는 각각 temporal Gaussian filter의 frequency 와 phase response를 보여준다. Original scheme에서의 temporal Gaussian Kernel은 [1 6 15 20 15 6 1]/64이고 단순 tail cut scheme은 [0 6 15 20 15 6 1]/64 와 [1 6 15 20 15 6 0]/64 이며 proposed tail cut scheme은 [0 6 15 20 15 0]이다. (c)에서는 frequency response를 보여준다.

단순 tail cut scheme과 proposed tail cut scheme 모두 tail cut으로 인해 high frequency 성분의 ripple이 발생하는 것을 확인할 수 있다. (a)는 Gaussian filter의 결과인 (c)에 gradient filter를 적용한 결과이다. Gradient filter 로 사용된 [1 -8 0 8 -1]/12 kerne은 high frequency noise를 감소시키는 특성을 갖는 finite central difference 기반의 kernel이기 때문에 tail cut으로 인해 발생한 high frequency ripple은 gradient filter를 거치면서 감소되는 효과가 있다. (a)와 (c)에서 보여지는 tail cut으로 인한 high frequency ripple은 gradient filter를 거치면서 proposed scheme인 dashed line은 -30dB에서 -50dB이하로, simple scheme인 dotted line은 -35dB로 감소된 결과를 확인할 수 있다. (d)에서는 phase response를 보여준다. Dashed line인, proposed tail cut scheme의 결과는 original scheme과 거의 동일한 linear phase 특성을 갖지만, dotted line인 단순 tail cut의 phase 특성이 서로 다른 curve 모양의 non-linear phase 특성을 나타내는 것을 확인할 수 있다. [0 6 15 20 15 6 1]/64 의 phase response는 line의 위에서 curve를 그리고 있는 dotted line이고 [1 6 15 20 15 6 0]/64의 phase response 는 line 아래로 curve를 그리고 있는 dotted line이다. FIR filter가 symmetric 하지 않을 때 발생하는 non-linear phase 특성은 이미지 distortion을 발생시킨다. 그림 5-6에 있는 d부터 h의 이미지 중에, d와 h는 각각 [0 6 15 20 15 6 1]/64와 [1 6 15 20 15 6 0]/64에 의해서 서로 다른 이미지 distortion이 발생하고, e부터 g 는 original filter가 사용되어 distortion없는 이미지가 된다. 결과적으로 simple tail cut scheme을 사용하면 (a), (b)의 최종 결과만을 보면 linear phase를 만족시키는 결과라고 판단되었지만, (d)를 통해 확인할 수 있는 것은 (b)의 linear phase는 (d)의 서로 다른 non-linear

phase response가 compensation한 결과이다. 실제로는, 중간단계에서 gradient 를 구하기 위해 이용되는 tail cut 된 Gaussian filter가 symmetric 하지 않기 때문에 (d)의 dotted line처럼 non-linear phase characteristic을 갖게 되고, 이미지들에 서로 다른 distortion을 발생시킨다. 이로 인한 gradient calculation error 는 결과적으로 optical flow accuracy에 영향을 끼치게 된다. 반면 proposed tail cut scheme은 (d)의 dashed line 처럼 original Gaussian 의 phase response와 동일한 linear phase response를 갖는다.

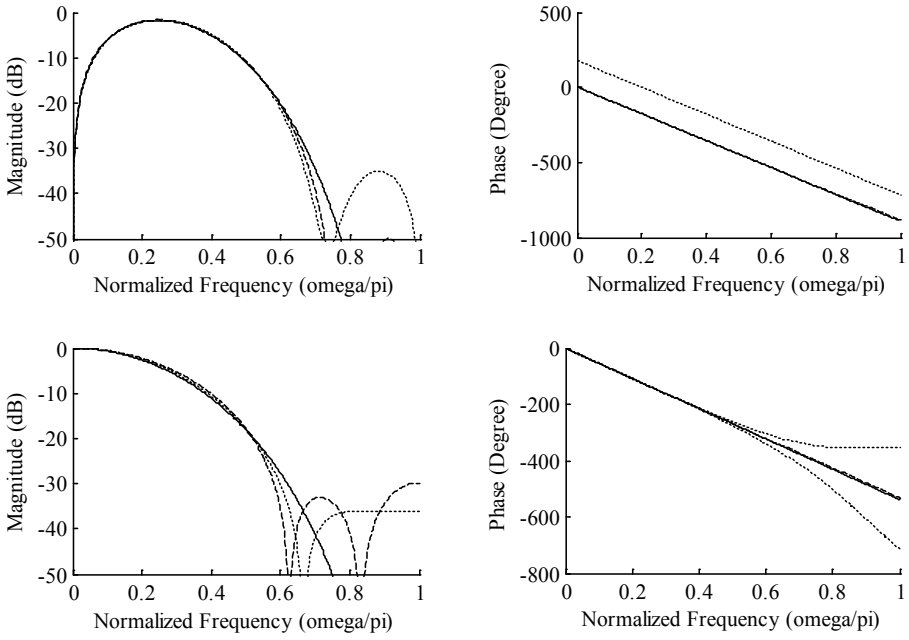


그림 5-9 Magnitude and Phase response for tail-cut filter

그림 5-10은 2 frame reuse를 사용하는 serial scheme의 hardware 구조를 나타낸다. Conventional system대비 제안하는 시스템에서 추가되거나 수정되는 모듈은 회색 블록으로 표시한다. Frame A부터 L까지의 기존시스템보다 1 frame 추가된 12장의 이미지를 외부 메모리로부터 읽어온다. 먼저 frame 6

의 처리시에는 1번부터 11번의 frame 데이터가 temporal and spatial Gaussian filter를 거쳐 처리된다. 이 결과는 x, y, t gradient filter를 거쳐 least square matrix, equation solver and validation의 과정을 통해서 optical flow 연산을 수행한다. 이 과정은 conventional 처리 과정과 동일하다. 한 line의 처리가 끝나면 2번부터 12번의 frame 데이터를 가지고 frame 7의 optical flow 연산이 진행된다.

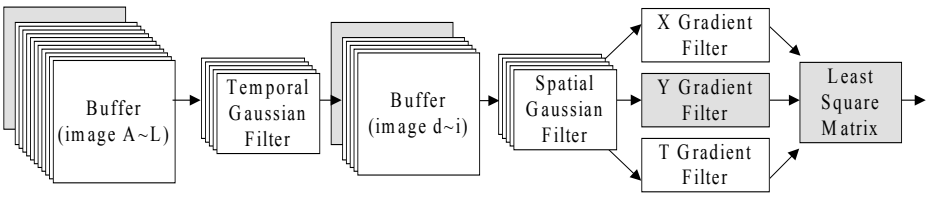


그림 5-10 Modified Blocks for frame reuse

그림 5-11은 line 단위로 번갈아 가며 처리하는 y gradient filter 블록 (a)와 least square matrix 블록 (b)의 구조를 보여준다. Serial scheme은 raster scan order로 line 단위로 번갈아 가며 처리한다. 이때, vertical 방향의 연산을 위해서는 이전 line의 처리 결과값이 있어야 하기 때문에 y gradient filter와 least square matrix 블록에는 이전 line의 결과를 임시로 저장할 buffer가 필요하다. 6번 Frame의 처리가 끝나고 7번 frame이 시작되면, 2번부터 12번 frame의 데이터는 temporal & spatial Gaussian filter를 거쳐 x, y, t gradient filter 처리된다. 이때, y gradient filter의 경우 y 축 방향의 처리를 위해 buffer가 사용되는데, 이전 frame 6에서 사용된 데이터와 분리되어 독립적으로 처리되어야 하므로 frame 6과 frame7을 위한 각각의 buffer가 사용된다. Least square matrix 모듈의 연산은 5x5 matrix연산을 위해 IxIx, IxIy, IyIy, IxIt 그리고 IyIt 연산이 필요하므로 x, y, t gradient filter의 결과인 Ix, Iy,

It를 저장하기 위한 buffer가 필요하다. 이 경우도 frame 6과 frame 7을 위한 각각의 buffer가 사용된다. 즉, 2 frame reuse를 하는 경우 2개의 buffer가 사용되고, 만약 3 frame reuse 구조를 사용한다면 3개의 buffer가 사용된다. 각각의 buffer는 한 line 연산이 끝나면 mux에 의해서 다음 순서의 buffer가 선택된다. Serial 처리 scheme을 사용할 경우 computation logic은 공유하고 Buffer 만 추가되기 때문에 frame reuse를 위해 추가되는 hardware resource 증가를 최소로 할 수 있다.

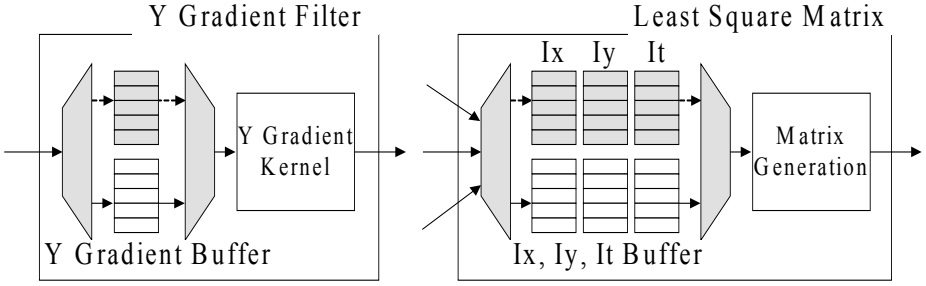


그림 5-11 Additional Buffer for frame reuse

5.4 Matrix Generation

이번 장에서는 LK optical flow의 matrix multiplication에 구조에 대해서 설명하고 사용되는 multiplier의 개수가 window의 크기에 독립적으로 2개만을 사용하는 새로운hardware architecture를 제안한다.

5.4.1 Matrix Multiplication

그림 5-12는 M과 N이 5일 때의 processing을 보여준다. Real-time processing을 위하여 위의 operation이 매 cycle마다 하나의 결과를 출력하기 위해서는 일반적으로 M x N개의 multiplier를 사용하거나 Column방향의 multiplication을 수행하기 위한 N개의 multiplier와 accumulation을 하며 중간(intermediate) 처리 data를 저장하기 위한 M×1 size의 buffer를 사용하는 구조와 resource를 필요로 한다. 이러한 구조로 구현을 하는 경우, window의 size가 증가함에 따라 필요로 하는 resource는 linear 하게 증가 함으로 윈도우 사이즈가 증가하는 경우 구현에 있어서 큰 제한을 가진다.

이러한 resource 문제를 해결하고자 기존의 연구에서는 소수의 multiplier를 이용하고 결과를 내부 또는 외부memory에 저장하는 방법을 사용하거나 sequential한 처리를 통하여 2개의 multiplier만을 가지는 구조를 사용한다[25].

또한 이와 같이 window내의 픽셀 값들을 사용하는 영상처리를 위해서는 현재의 pixel 정보뿐 아니라 horizontally, vertically neighborhood에 있는 pixel들의 정보 또한 필요하다. 이를 위해서는 이전 pixel 들의 정보를 가져와 사용할 수 있어야 하며 이를 위해 널리 사용되는 방법은 (window의

vertical 사이즈의 크기를 가지는) internal line buffer를 가지는 것이다[38]. 따라서 window 사이즈의 증가는 line buffer 사이즈의 linear한 증가 역시 필요로 한다.

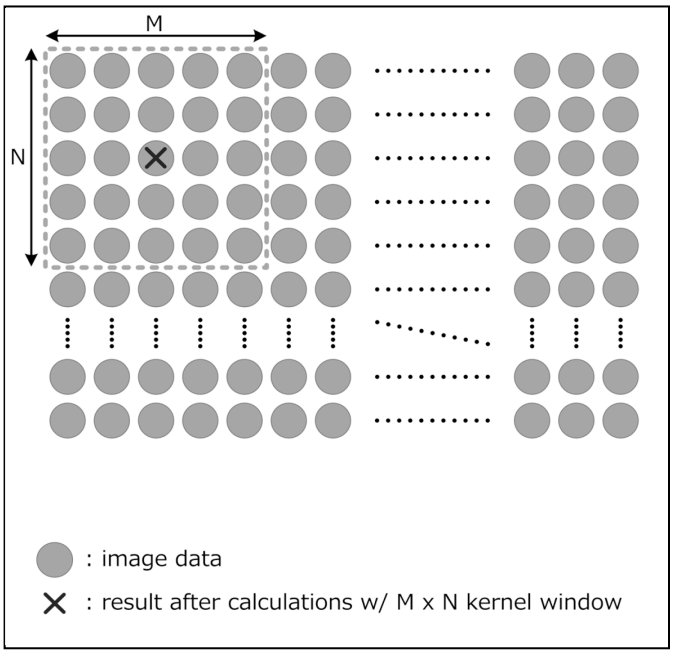


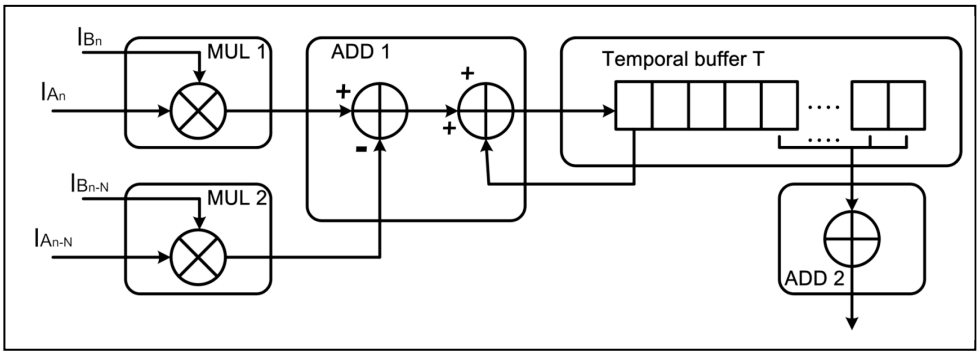
그림 5-12 Operation with kernel size of 5x5 (M, N=5)

5.4.2 Proposed Matrix Multiplication

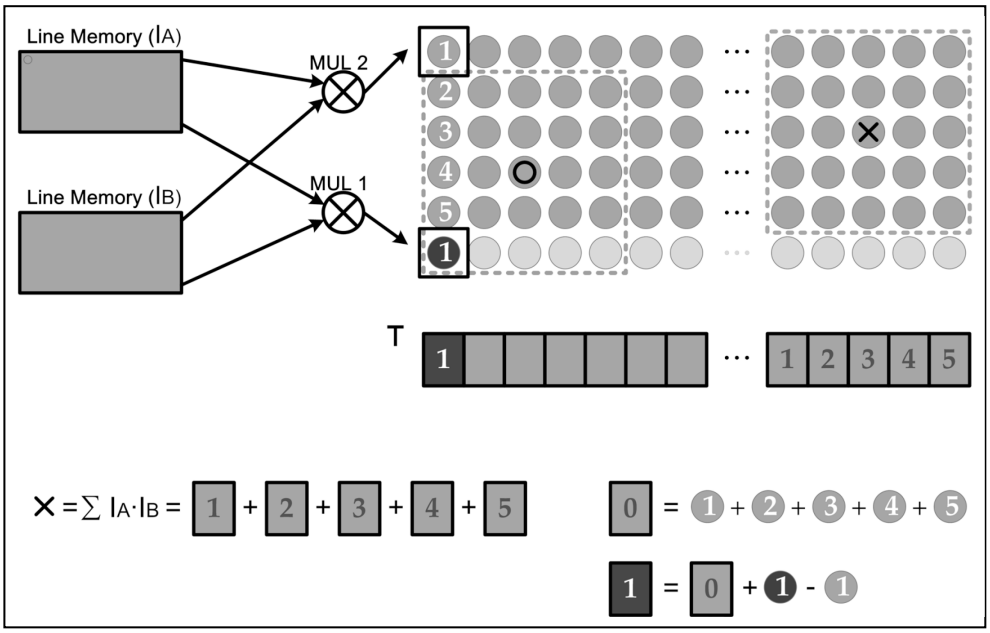
그림 5-13은 제안하는 방법의 구조와 연산과정을 보여준다. Raster 주사 방향으로 입력되는 영상의 gradient 값 I_A와 I_B는 매 cycle 차례대로 MUL1을 통해 곱해지며 그 결과는 temporal buffer T에 저장된다. 이때 temporal buffer T의 크기는 처리하는 영상의 가로 사이즈만큼 요구된다. 다음 line에 입력되는 gradient 값은 마찬가지로multiplication이 수행된 후, temporal buffer T 에 기 저장된 이전 값과 합해져 다시 temporal buffer T에 저장된다.

이러한 과정은 window의 vertical size인 N line동안 accumulation 되며 수행되어, 그 결과 temporal buffer T에는 N라인 동안 multiplication되고 accumulation 되어진 gradient결과 값이 저장되어 있다. 즉, q line에서의 Temporal buffer T T(x,q)는 N 라인 동안 accumulate되어진 gradient 값 I_A와 I_B의 곱이다. 따라서 저장되어 있는 값을 horizontally M개를 더하면 M x N 윈도우의 Frobenius inner product 값 $\sum I_A I_B$을 얻을 수 있다. 즉, Position (p,q)에서의 결과 값 O(p,q)는 q line에서의 temporal buffer T에서 p 픽셀을 기준으로 M개의 값을 더함으로 써 얻어진다.

윈도우의 vertical 사이즈인 N개 라인의 계산을 위하여, N+1 번째 라인의 경우, 이전 라인들과 같이 입력되는 gradient 값들을 multiplication하여 temporal buffer T에 accumulate하고, 동시에 N번째 이전 라인(즉, 라인 1)의 multiplied된 gradient 값을 빼 주어야 한다. 따라서 이 N번째 이전 line의 gradient값을 multiply하기 위한 MUL2가 존재한다. 즉, q+1라인에서의 temporal buffer T T(x,q+1)는 q 라인에서의 temporal buffer T(x,q)와 새로운 라인에서 입력되는 gradient 값들의 곱 $\mathrm{I_A}\left(\mathrm{x, q+1+\frac{N-1}{2}}\right)\cdot \mathrm{I_B}\left(\mathrm{x, q+1+\frac{N-1}{2}}\right)$ 그리고 N 라인 이전의 gradient 값들의 곱 $\mathrm{I_A}\left(\mathrm{x, q-\frac{N-1}{2}}\right)\cdot \mathrm{I_B}\left(\mathrm{x, q-\frac{N-1}{2}}\right)$ 들의 합과 차로 계산된다. 그림 5-13 에서 볼 수 있듯이 제안하는 방법은 window 사이즈의 증가가 추가적인 hardware resources를 필요로 하지 않는다. 윈도우 사이즈 크기의 증가와 관계없이 적은 수(2)의 multiplication만을 사용하여 구현되었으며, 동시에 매 cycle 출력을 가질 수 있는 구조를 가짐으로써 real-time processing에도 적합하다.



(a)



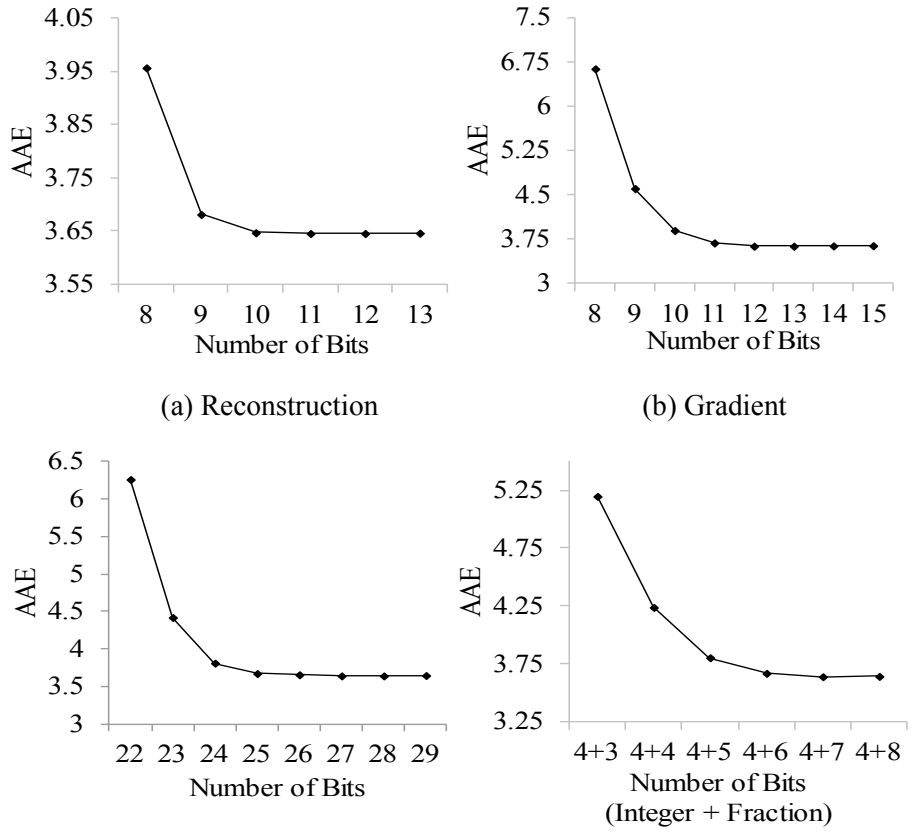
(b)

그림 5-13 Proposed Matrix Multiplication

5.5 FPGA Implementation

아래 그림 5-12에서는 제안하는 그림 5-1 (b)의 HW 구조에서 각 Block 들의 중간값 저장에 필요한 bit-width의 결정을 위해 각각의 bit-width가 AAE에 미치는 영향을 실험적으로 분석하였다. Gaussian Filter의 σ는 Simplified 1.225인 7 tap, [1 6 15 20 15 6 1]/64 Kernel을 사용하였으며 Interleaved Subsampling 및 10 bit Lanczos3 Reconstruction filter를 사용하였다. Subsampling의 pixel당 8bits 를 할당하여 외부 메모리에 저장하였다. 각각의 test 과정에서 명시하지 않은 나머지 모듈들의 bit width들은 모두 32bit float precision을 갖는다. 35%의 Density에서 AAE를 구하기 위해서 THeigen이 조절되었고 THnres는 0.01이다. 그림 5-14 (a) 는 Reconstruction filter의 output 데이터의 bit width 변화에 따른 AAE 결과를 보여준다. 그림 5-14 (b)는 Gradient operation 결과의 bit width 변화에 따른 AAE 결과이다. 그림 5-14 (c) 는 Matrix Generation 결과의 output bit width에 따른 AAE 결과이다. 그림 5-14 (d)는 32/16 bit Integer Divider 블록의 결과를 외부 메모리로 저장할 때 사용하는 Output Data 의 bit width 변화에 따른 AAE 결과이다. Integer Divider의 Accuracy를 높이기 위해서 numerator 와 denominator는 leading zero 또는 leading one 을 고려하여 shift연산을 거친 후 Integer Divider로 입력된다. 그림 5-14를 가지고 각 모듈의 bit width를 설정할 때, AAE의 improvement가 saturation되기 시작하는 bit width를 선택한다. Reconstruct Filter Output bits 는 11 bits, Gradient Output Bits 는 11bits, Matrix output bits는 26bits로 정하였다. 최종 output 모듈인 Integer Divider output의 경우 선택한 Bit width는 바로 Memory Bandwidth에 영향을 끼치므로

Accuracy 와 Memory Bandwidth간의 Trade-off를 고려해야 한다. 따라서 기본적으로 4bit integer와 6bit fraction으로 사용하고, 필요에 따라 가변 할 수 있도록 구현하였다.



(a) Reconstruction (b) Gradient

(c) Matrix Generation (d) Velocity Output

그림 5-14 Bit-width allocation for HW modules

본 논문에서 제안된 interleaved subsampling 알고리즘을 이용한 Lucas-Kanade Optical Flow HW Core를 Verilog HDL을 사용하여 그림 5-1 (b)에서 제시된 Lucas-Kanade Optical Flow Hardware 구조로 구현하였다. 표 5-5는 제안된 HW Core와 기존에 발표된 구현 연구와 비교하였다. First Row는 제

안하는 구조를 포함한 3가지 구현을 나타낸다. Second and third Rows는 AAE와 std.dev를, fourth row는 density를, fifth row는 Read Frame Bandwidth per Frame을 나타낸다. 제안하는 방법의 HW구현의 최종 AAE는 3.75로 표 5-3에서의 SW 모델의 결과대비 0.1의 오차가 증가한다. Mahalingam의 결과는 11장의 frame memory를 읽어와서, AAE가 6.37이다. 즉, 본 논문 보다 4배의 많은 bandwidth를 필요로 하면서도, AAE도 2.62크다. Diaz의 결과는 본 논문 보다 AAE는 0.24작지만, memory bandwidth는 약 2.5배 크다. Barranco는 Diaz의 LK optical flow core[24]를 사용하여 Multi-scale LK optical flow를 구현하였다[35]. Warping 구현을 위해서 gradient tap 수를 기존 5tap 에서 3tap으로 수정하였고, 이 system을 이용해서 mono scale (single level) optical flow도 구할 수 있게 하였다. 외부 frame을 5장을 이용하여 Yosemite without cloud sequence를 이용하여 구한 AAE는 5.97로 제안한 방법보다 81%의 frame을 더 사용하면서 AAE는 2.78나 크다. Sixth row는 Core Processing Throughput을 나타낸다. 제안한 방법은 Diaz처럼 Pipeline 구조로 설계되었기 때문에 1 pixel / cycle의 처리 속도를 갖는다. Seventh, eighth and ninth row에서는 Gate count, Memory Usage 그리고 Max Frequency를 나타낸다. Proposed 방식은 Dongbu 130nm 공정의 library를 가지고 Synopsys의 design compiler로 Synthesis한 결과이고, Diaz는 the DK Synthesizer by Agility(Previously named Celoxica)로 synthesis 한 결과이고 Mahalingam은 ISE의 Equivalent Gatecount Estimation 결과이다. 제안한 방식은 Diaz와 동일한 조건에서 비교하기 위해서 1024 pixel width 크기의 이미지 처리가 가능하도록 설계한 결과인데, 동일한 공정 및 합성 툴에 의한

결과가 아니어서 Gatecount 수치를 서로 비교하기에는 다소 무리가 있지만 gatecount는 감소하였고, Reconstruction Filter처리를 위한 모듈이 추가되었기 때문에 Line Memory의 사용량은 증가한 것을 확인할 수 있다. Tenth row는 각 Frame의 처리 가능한 최대 Frames Per Second를 나타낸다. Mahalingam은 640x480 영상의 처리 결과이고 Diaz와 Proposed는 800x600 영상의 처리 결과이다. Frame Per Second는 Throughput에 동작 Frequency 의 곱으로 정해지므로 이전 연구의 Maximum FPS처리 속도는 각각 640x480 32 fps 와 800x600 170 fps의 처리 속도를 갖고 Proposed 방식은 Maximum FPS는 800x600 196 fps 의 처리속도를 갖는다. 마지막으로 eleventh row는 Read Frame Bandwidth per Second를 나타낸다. Proposed HW Core는 Diaz와 동

표 5-5 Hardware IP Core Comparisons with other Implementations

Implementation Results	Mahalingam [25]	Diaz [24]	Barranco [35]	Proposed
AAE	6.37	3.51	(5.97)	3.75(3.19)
Std. Dev.	11.37	9.24	-	8.22(3.50)
Density (%)	38.3	36.47	(59.88)	35.24(61..56)
Read Frame BW per Frame	11	7	5	2.75
Core Throughput (Px/Cycle)	0.18	1	1	1
Gate Counts (Kgates)	1580	1642	-	352
Memory (Kbits)	360	720	-	1068
Max Freq (MHz)	55	82	83	94
Max FPS	32 (640x480)	170 (800x600)	172 (800x600)	196 (800x600)
Read Frame BW per Sec.	352 (32 fps)	1190 (170 fps)	850 (170 fps)	467 (170 fps)

Gaussian Sigma σ is 1.225. THeigen is 2 and THnres is 0.01. Subsampled image pixels are truncated in 8 bits. AAE, Std. Dev. and Density in parenthesis are results with Yosemite without Cloud sequence.

일한 처리속도인 800x600 170fps 일 때, 요구되는 Read Frame Bandwidth는 640x480 32fps 를 나타내는 Mahalingam의 수준을 사용하는 것을 확인 할 수 있다. 결과적으로 제안된 interleaved subsampling 방법으로 구현한 HW Core는 Accuracy의 degradation은 최소화하면서 Required Read Frame Bandwidth는 기존연구 대비 75%와 61% Reduction을 이루었음을 확인할 수 있다.

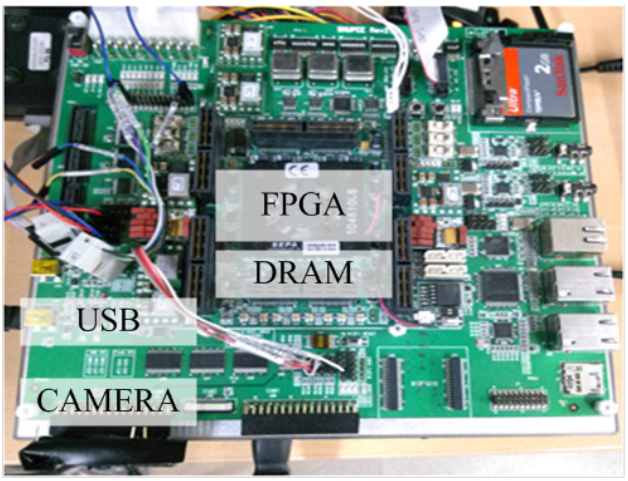
제안한 Hardware를 Xilinx Virtex-6 LX760 FPGAFPGA에 targeting하여 Functional Verification을 하였다. 전체 system의 Resource Usage는 표 5-6에 나와있다. FlipFlop, Lookup Table, Block RAM 그리고 DSP 의 사용량을 나타내고 RAMB36E1과 RAMB18E1은 각각 36Kb 와 18Kb block RAM을, DSP48E1은 25x18 bit multiplier와 48bit Accumulator로 구성된 DSP를 말한다. 구현된 HW 결과는 전체 시스템의 2~8 %의 resource를 사용하여 구현되었다.

표 5-6 FPGA System Resource Usage

Logic Block	Used	Available	Utilization
Slice Register	22389	948480	3 %
Slice LUT	38269	474240	8 %
RAMB36E1	12	720	2 %
RAMB18E1	108	1440	8 %
DSP48E1s	56	864	7 %

그림 5-15 는 Optical flow 하드웨어의 검증을 위하여 사용된 Xilinx Virtex-6 LX760 FPGA가 내장된 SNUPEE 2.0 board를 보여준다. 카메라로부터 640x480 30fps 영상을 입력 받아서, Subsampling된 결과를 DRAM에 저장한다. FPGA에서 DRAM에 저장된 subsampling된 영상을 읽어 복원하고

optical flow operation을 수행하고, 그 결과를 다시 DRAM에 저장을 한다. USB 2.0 channel을 통해서 operation 결과를 PC로 출력하여 실시간으로 display 함으로써 정확히 동작하는 것을 확인할 수 있다.



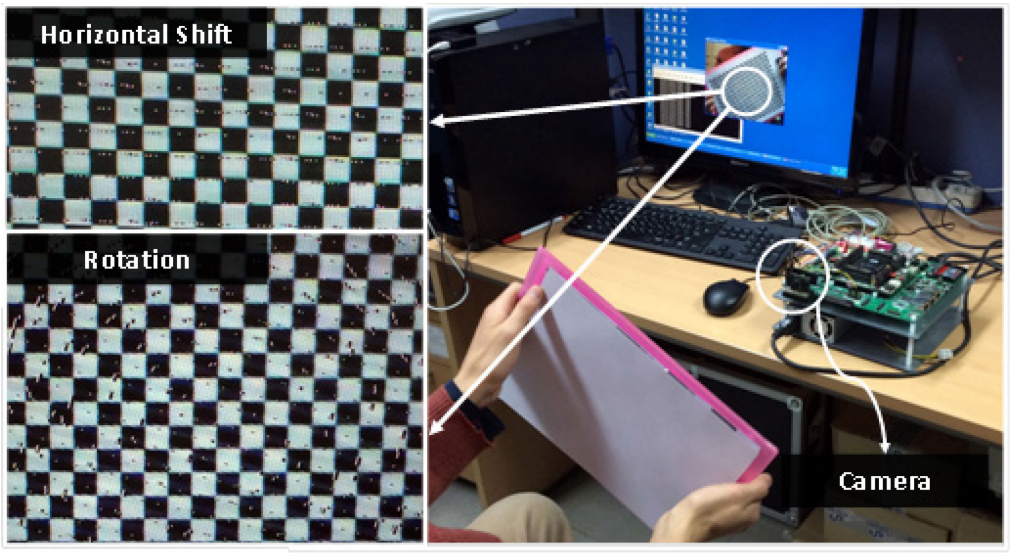


그림 5-15 SNUPEE 2.0 FPGA Board

제6장 결론

Accurate Real-time optical flow 구현을 위해 high frame rate image를 이용하여 multi-frame rate and multi-scale optical flow 알고리즘을 제안하였다. 제안한 알고리즘은 high frame rate system을 위한 최초의 full density optical flow 알고리즘이다. High frame rate을 위한 test sequence를 generation하고 제안한 알고리즘을 검증하였다. Synthetic test sequence뿐 아니라 youtube.com의 4K 1000FPS Natural 비디오 영상을 통해 테스트 하였을 때, 기존의 pyramidal LK optical flow보다 큰 움직임 뿐 아니라 작은 움직임에 대해서 정확한 optical flow를 얻는 것을 Error를 나타내는 수치 뿐 아니라 optical flow 결과로 확인할 수 있었다.

알고리즘의 real-time operation을 위해 accuracy감소는 최소화하면서 요구되는 frame memory access bandwidth를 전체 1/10로 감소시킬 수 있는 spatial and temporal bandwidth reduction 방법을 제안하였다. 또한 LK optical flow에서 least square matrix generation을 위해서 사용하는 Multiplier의 줄이는 architecture를 제안하였고, FPGA를 통해서 동작을 확인하였다.

이상의 제안한 알고리즘 및 검증을 통해서 accurate real-time optical flow system을 제안한다.

참 고 문 헌

[1] J. J. Gibson, *The perception of the visual world*. Oxford, England: Houghton Mifflin, 1950.

[2] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart, "Mav navigation through indoor corridors using optical flow," *in Proc. IEEE Conf. Robotics and Automation(ICRA),* 2010, pp. 3361-3368.

[3] G. N. Desouza and A. C. Kak, "Vision for Mobile Robot Navigation: a Survey," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 24, no. 2, pp. 237-267, Feb. 2002.

[4] R. Cucchiara, M. Piccardi, and A. Prati, "Detecting moving objects, ghosts, and shadows in video streams," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 25, no. 10, pp. 1–6, Oct. 2003.

[5] P. Sundberg, T. Brox, M. Maire, P. Arbeláez, and J. Malik, "Occlusion boundary detection and figure/ground assignment from optical flow," *in Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2011, pp. 2233–2240.

[6] S. Ali and M. Shah, "Human Action Recognition in Videos Using Kinematic Features and Multiple Instance Learning," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 32, no. 2, pp. 288-303, Feb. 2010.

[7] H. Wang, A. Klaser, C. Schmid, and C.-L. Liu, "Action Recognition by Dense Trajectories," *in Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2011, pp. 3169-3176.

[8] G. Adiv, "Determining 3D Motion and Structure from Optical Flows Generated by Several Moving Objects," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 7, no. 4, pp. 384-401, Jul. 1985.

[9] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence,* vol. 17, no. 1, pp. 185–203, Aug. 1981.

[10] B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” *in Proc. 7th Int’l Joint Conf. Artificial Intelligence,* 1981, pp. 674–679.

[11] A. Plyer, G. L. Besnerais, and F. Champagnat, “Massively parallel Lucas Kanade optical flow for real-time video processing applications,” *Journal of Real-Time Image Processing,* to be published.

[12] C. Zach, T. Pock, and H. Bischof, “A Duality Based Approach for Realtime TV-L1 Optical Flow,” *in Proc. Ann. Symp. German Assoc. Pattern Recognition,* 2007, pp. 214-223.

[13] A. Bainbridge-Smith and R. G. Lane, “Determining optical flow using a differential method,” *Image and Vision Computing,* vol. 15, no. 1, pp. 11–22, Jan. 1997.

[14] A. Bainbridge-Smith and R. Lane, “Measuring Confidence in Optical Flow Estimation,” *Electronics Letters,* vol. 32, no. 10, pp. 882-884, May 1996.

[15] S. Maya-Rueda and M. Arias-Estrada, “FPGA processor for real-time optical flow computation,” *Lecture Notes in Computer Science,* vol. 2778, pp. 1103–1016, 2003.

[16] B. McCane, K. Novins, D. Crannitch, and B. Galvin, “On benchmarking optical flow,” *Computer Vision and Image Understanding,* vol. 84, no. 1, pp. 126–143, Oct. 2001.

[17] H. C. Liu, T. S. Hong, M. Herman, T. Camus, and R. Chellappa, “Accuracy vs efficiency trade-offs in optical flow algorithms,” *Computer Vision and Image Understanding,* vol. 72, no. 3, pp. 271–286, Dec. 1998.

[18] J. Díaz, E. Ros, F. Pelayo, E. M. Ortigosa, and S. Mota, “FPGA-based real-time optical-flow system,” *IEEE Trans. Circuits and System for Video Technology,* vol. 16, no. 2, pp. 274–279, Feb. 2006.

[19] J. R. Bergen, P. Anandan, K. J. Hanna and R. Hingorani, “Hierarchial model-based motion estimation,” *in Proc. Second European Conf. Computer Vision,* 1992, pp. 237-252.

[20] J.Y. Bouguet, “Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the Algorithm,” technical report, Intel Microprocessor Research Labs, 1999.

[21] L. Alvarez, J. Weickert, and J. Sa´nchez, “Reliable Estimation of Dense Optical Flow Fields with Large Displacements,” *Int’l J. Computer Vision,* vol. 39, no. 1, pp. 41-56, Aug. 2000.

[22] S. Lim, J. Apostolopoulos, and A. Gamal, “Optical Flow Estimation Using Temporally Oversampled Video,” *IEEE Trans. Information Processing,* vol. 14, no. 8, pp. 1074-1087, Aug. 2005.

[23] I. Ishii, T. Taniguchi, K. Yamamoto, and T. Takaki, “High-frame-rate optical flow system,” *IEEE Trans. Circuits and System for Video Technology,* vol. 22, no. 1, pp. 105–112, Jan. 2012.

[24] J. Díaz, E. Ros, R. Agís and J. L. Bernier, “Superpipelined high- performance optical-flow computation architecture,” *Computer Vision and Image Understanding,* vol. 112, no. 3, pp. 262–273, Dec. 2008.

[25] V. Mahalingam, K. Bhattacharya, N. Ranganathan, H. Chakravarthula, R. Murphy, and K. Pratt, “A VLSI architecture and algorithm for lucas- kanade-based optical flow computation,” *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 18, no. 1, pp. 29–38, Jan. 2010.

[26] D.J. Fleet and K. Langley, “Recursive Filters for Optical Flow,” *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 17, no. 1, pp. 61-67, Jan. 1995.

[27] J.L. Barron, D.J. Fleet, and S.S. Beauchemin, “Systems and Experiment: Performance of Optical Flow Techniques,” *Int’l J. Computer Vision,* vol. 12, no. 1, pp. 43-77, Jan. 1994.

[28] E. P. Simoncelli, “Design of multi-dimensional derivative filters,” *in Proc. Int. Conf. Image Processing,* vol. 1, 1994, pp. 790-793.

[29] J. W. Brandt, “Improved accuracy in gradient based optical-flow estimation,” *Int’l J. Computer Vision,* vol. 25, no. 1, pp. 5-22, Oct. 1997.

[30] T. Camus, “Real-time quantized optical flow,” *J. Real-Time Imaging,* vol. 3, no. 2, pp. 71-86, Apr. 1997.

[31] Miller, Irwin, John E. Freund, and Richard Arnold Johnson. *Probability and statistics for engineers.* Vol. 4. Englewood Cliffs, NJ: Prentice-Hall, 1965, pp. 187-191.

[32] R.A. Haddad and A.N. Akansu, “A Class of Fast Gaussian Binomial Filters for Speech and Image Processing,” *IEEE Trans. Acoustics, Speech and Signal Processing,* vol. 39, pp. 723-727, Mar. 1991.

[33] M. Anguita, J. Diaz, E. Ros, and F. J. Fernandez-Baldomero, “Optimization strategies for high-performance computing of optical-flow in general-purpose processors,” *IEEE Trans. Circuits and System for Video Technology,* vol. 19, no. 10, pp. 1475–1488, Oct. 2009.

[34] Murachi, Yuichiro, et al. "A VGA 30-fps realtime optical-flow processor core for moving picture recognition." *IEICE transactions on electronics* 91.4 (2008): 457-464.

[35] Barranco, Francisco, et al. "Parallel architecture for hierarchical optical flow estimation based on FPGA." *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vo. 20, no. 6, pp. 1058-1067, Jun. 2012.

[36] Bouguet, Jean-Yves. "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm." *Intel Corporation* 5 (2001).

[37] Lowe, David G. "Distinctive image features from scale-invariant keypoints." *International journal of computer vision* 60.2 (2004): 91-110.

[38] P. -Y. Hsiao, C. -L. Lu, and L. -C. Fu. “Multilayered Image Processing for Multiscale Harris Corner Detection in Digital Realization.”, *IEEE Trans Industrial Electronics*, vol. 59, no. 5, pp. 1799-1805, May 2010.

Abstract

Optical flows are vectors that represent the motion of an object or pattern in a video stream. It has been used in various computer vision applications, such as robot navigation, moving object detection, action recognition and three-dimensional (3D) shape reconstruction. However, complex algorithms for accurate optical flow increase computation time and hardware resources. The latest Horn-Schunck (HS) based implementation with graphics processing unit (GPU) acceleration takes longer than one second to calculate the optical flow for a 640x480 image. Hardware Implementations of pyramidal Lucas-Kanade (LK) use embedded DRAM or external SRAM for frame memory. Real-time accurate optical flow is the key problem for realization of various computer vision algorithms.

Multi-frame rate and multi-scale optical flow algorithm for real-time accurate optical flow is proposed. Multi-frame rate algorithm uses high frame rate camera, merge the results of multi frame rate and increases accurate optical flow displacement. Previous algorithms for high frame rate camera require O(n) computation when n is the increased ratio from normal frame rate. Proposed algorithm reduces the computation to O(log n) by selecting frames in multiple frame rate relation. As a system's computing power limits the maximum high frame rate, with a same computing power, proposed algorithm processes much higher frame rate and this increases the maximum

object displacement for accurate optical flow calculation.

Spatial and temporal bandwidth reduction for frame memory access is proposed. In general, computing performance can be improved by parallel processing using additional hardware logics. However, the increase in the amount of frame memory access by a high-frame-rate operation is a more difficult problem to solve, even with additional hardware resources. Therefore, an algorithm that requires small amount of frame memory access bandwidth is better for real-time operation. The proposed spatial and temporal bandwidth reduction algorithm reduces frame access bandwidth by 61% and 75 % when compared to the values in previous works.

Matrix multiplication architecture that requires constant number of multipliers for various size of matrix size is proposed. Matrix multiplication is required for structure tensor in LK optical flow algorithm. In general, hardware implementation of nxn matrix multiplication requires n multiplies and this limits the window size of optical flow. Proposed architecture uses only two multipliers for various size of matrix multiplication.

High frame rate test sequences are generated with Middlebury test sequences and proposed algorithms and hardware architecture are verified by software-hardware simulation and FPGA implementation.

Key word : Multi-frame rate, High frame rate, Optical flow, Lucas-Kanade, Bandwidth reduction

Student number : 2010-30222