한국군사과학기술학회지 제14권 제4호, pp. 663~668, 2011년 8월

학술논문 센서 · 신호처리 부문

CUDA를 이용한 초해상도 기법의 영상처리 속도개선 방법

An Image Processing Speed Enhancement in a Multi-Frame Super Resolution Algorithm by a CUDA Method

김 미 정*
Mi-Jeong Kim

Abstract

Although multi-frame super resolution algorithm has many merits but it demands too much calculation time. Researches have shown that image processing time can be reduced using a CUDA(Compute unified device architecture) which is one of GPGPU(General purpose computing on graphics processing unit) models. In this paper, we show that the processing time of multi-frame super resolution algorithm can be reduced by employing the CUDA. It was applied not to the whole parts but to the largest time consuming parts of the program. The simulation result shows that using a CUDA can reduce an operation time dramatically. Therefore it can be possible that multi-frame super resolution algorithm is implemented in real time by using libraries of image processing algorithms which are made by a CUDA.

Keywords : Super Resolution(초해상도), CUDA(쿠다), GPGPU

1. 서 론

정찰용 항공영상의 경우 물리적인 제약으로 광학장치 및 이미지 센서로부터 획득할 수 있는 영상의 해상도에 제약이 따른다. 뿐만 아니라 대기조건 및 촬영거리의 제약에 의해 저해상도 영상을 획득하게 되어 정확한 지형정보 및 목표물을 인식하는 것은 매우 어렵다. 이를 해결하기 위해 획득된 저해상도 영상들의 해상도를 높이기 위한 많은 연구가 이루어져 왔다. 저해상도 영상들을 이용한 초해상도 기법은 해상도 향상기법 가운데 하나로 군사 및 민간제품에서 많이 이용되고 있다. 그러나 알고리즘의 복잡함에 따른 긴 수행시간으로 실시간 영상처리가 불가능하여 사용에 어려움이 있다.

최근 카메라 및 검출기의 성능이 향상됨에 따라 영상처리의 성능에 대한 요구도도 증가되고 있다. 즉 영상의 크기와 frame rate가 증가하면서 초당 처리해야 될 영상의 양이 급속히 증가하고 있다. 그러나 현재의 CPU로는 이를 처리할 능력이 부족하여 병렬처리가 가능한 GPU에 대한 관심이 나타나기 시작했다. GPGPU (General Purpose computing on Graphics Processing Unit)는 기존 그래픽 카드의 코어인 GPU를 비그래픽 애플리케이션에 응용하여 CPU의 처리를 대신하는 시도를 통칭해 부르는 용어이다. 그러나 GPU의 장점을 이용

† 2011년 4월 27일 접수~2011년 7월 8일 게재승인
* 국방과학연구소(ADD)
책임저자 : 김미정(katz51@add.re.kr)

하여 비그래픽 애플리케이션에 응용하려는 노력은 쉽지 않았다. GPU는 CPU와 달리 아키텍쳐나 인스트럭션에 대한 정보가 드물고 API를 통하지 않고 직접 하드웨어를 제어하기가 무척 어렵기 때문이다. 이러한 상황을 개선하기 위해 NVIDIA에서는 2007년 C언어와 유사한 CUDA(Compute unified device architecture)를 발표하였다. CUDA는 OpenGL이나 DirectX의 그래픽스 API를 변용해 사용하는 고전적인 GPGPU 방식을 탈피해 programmability를 대폭 늘려 개발자가 C언어로 애플리케이션을 개발하면서 일부코드는 GPU에서 실행하도록 지정할 수 있도록 해준다. 이러한 CUDA는 실시간 처리를 요하는 영상처리 알고리즘의 처리속도 한계를 극복할 수 있도록 해주었다.

본 논문에서는 초해상도 알고리즘의 처리시간을 향상시키기 위하여 CUDA를 이용하는 방법에 대해 다루었다. 제 2장에서는 초해상도 알고리즘에 CUDA를 적용하기 위해 알고리즘의 처리속도를 분석하고 긴 시간이 소요되는 부분을 병렬처리가 가능한 부분과 그렇지 않은 부분으로 나누는 내용에 대해 단계별로 설명하고, 제 3장에서는 CUDA를 이용한 영상처리결과 속도 향상에 대한 정량적인 지표를 나타내었다. 제 4장에서는 제 3장에서의 결과를 이용하여 결론과 향후 연구되어야 할 내용에 대해 다루었다.

2. 본 론

가. 초해상도 알고리즘 개요

초해상도 기법은 사용하는 한 장의 영상만을 이용하는 learning-based 기법과 같은 장면에 대해 획득한 여러 장의 영상을 이용하는 reconstruction-based 기법으로 구분된다. 본 논문은 정찰용 항공기 영상에서 동일한 지역을 여러 번 촬영한 영상을 입력으로 이용한 체계에 기초를 두고 있으므로 reconstruction-based 기법을 기준으로 고안되었다.

Reconstruction-based 기법은 여러 장의 해상도가 낮은 영상을 이용하여 한 장의 높은 해상도의 영상을 복원하는 것으로 이를 위해 일반적인 영상 획득 시스템을 분석하여 해상도가 낮은 영상과 높은 영상 간의 관계를 수학적으로 모델링하고 이의 해를 구한다. 수학적 모델링을 이용하여 낮은 해상도의 영상들로부터 해상도가 높은 영상을 복원하기 위해서는 영상획득 과정에서의 왜곡을 보상하기 위한 카메라 캘리브레이션 과정, 낮은 해상도 영상사이의 정렬과정, 영상정렬 파라미터를 이용하여 해상도를 향상시키는 과정이 필수적이다. Fig. 1은 reconstruction-based 기법의 일반적인 수행순서를 나타낸다.

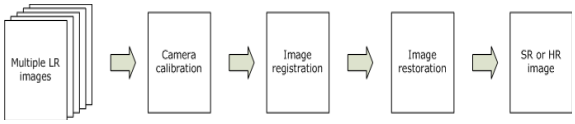

Fig. 1. reconstruction-based에 기반한 초해상도 기법

카메라 캘리브레이션 과정은 불완전한 렌즈에 의한 수차를 보정하기 위한 단계로 본 논문에서는 비교적 구현이 쉽고 신뢰성 있는 Zhengyou Zhang의 기법을 이용하였다.

영상 정렬 과정은 동일한 장면에 대해 획득된 여러 장의 영상 정보를 이용하기 위해 영상을 정렬시키는 방법으로 크게 feature 기반의 방법과 intensity 기반의 방법이 있다. 본 논문에서는 영상획득 조건과 영상처리 속도를 고려하여 feature 기반의 projective 변환 모델을 이용하였다. 세부적으로는 Harris corner detection, optical flow와 RANSAC이 이용되었다.

영상복원 과정은 동일한 장면을 촬영한 여러 장의 저해상도 영상의 정보를 종합하여 한 장의 고해상도 영상을 복원하는 과정을 일컫는다. 각 저해상도 영상들 사이에는 카메라나 피사체의 움직임에 의한 미세하게 다른 정보가 포함되어 있을 것이라는 가정을 기본전제로 하여 각기 다른 정보를 가진 저해상도 영상을 이용하여 고해상도 영상을 얻을 수 있게 되는 것이다. Fig. 2는 영상복원을 위해 고해상도 영상과 획득된 저해상도 영상사이의 관계를 모델링한 것이다.

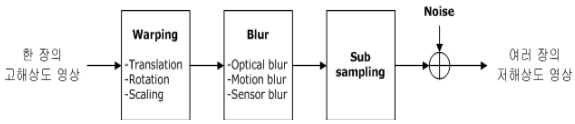

Fig. 2. 고해상도 영상과 저해상도 영상사이의 모델링

이를 기반으로 최근까지 제안된 여러 가지 영상복원 기법 가운데 비교적 안정된 해를 얻을 수 있도록 Regularization을 이용한 CLS(Constratined Least Squares) 기법을 이용하여 초해상도 기법을 완성하였고 이때 파라미터를 최적화하기 위해 gradient descent기법을 사용하였다. CLS기법의 에너지 함수는 아래와 같다.

$$\hat{x} = \arg min\{ \| y - Wx \|^2 + \lambda \| Cx \|^2 \} \quad (1)$$

식 (1)의 첫 번째 항인 $\| y - Wx \|^2$은 복원한 고해상도 영상 x를 영상획득모델을 이용하여 생성한 저해상도 영상 Wx와 실제 획득된 저해상도 영상 y가 비슷하게 되도록 하는 에너지 항이다. 그러나 첫 번째 항만으로는 복원된 고해상도 영상이 유일해라는 것을 보장할 수 없으므로 두 번째 항인 $\| Cx \|^2$ regularization 항을 사용한다. Regularization은 영상에서 인접한 픽셀들 사이의 밝기 값이 비슷하다는 가정을 이용하여 복원된 고해상도 영상에서 노이즈와 같은 에너지를 제한하는 항이다. λ는 첫 항과 두 번째 항 사이의 상대적인 비중을 조절하기 위해 사용되는 파라미터이다. 식 (1)을 최소로 하는 x를 구하기 위해 식을 미분하여 값이 0이 되는 x를 gradient descent 기법을 적용하여 구하면 아래와 같은 식을 얻을 수 있다.

$$\widehat{x_{n+1}} = \widehat{x_n} - \alpha_n \left(W^T (W\widehat{x_n} - y) + \lambda_n C^T C \widehat{x_n} \right) \quad (2)$$

$$\alpha_n = \frac{\| dx \|^2}{\| W \bullet dx \|^2 + \lambda_n \| C \bullet dx \|^2} \quad (3)$$

$$dx = W^T (W\widehat{x_n} - y) + \lambda_n C^T C \widehat{x_n} \quad (4)$$

여기에서 step size α_n은 수렴성 및 iteration 횟수와 관련된 파라미터로 iteration마다 계산된다.

나. 초해상도 알고리즘의 소요시간 분석

초해상도 알고리즘을 구현하여 각 단계에서 소요되는 시간은 Fig. 3과 같다. Fig. 3에서 총 소요 시간 중 각 과정에서 소요되는 시간의 비율을 %로 확인할 수 있다. 입력영상으로는 720*480 7장의 EO영상이 사용되었고 영상복원 과정 중 iteration을 5번 수행하여 고해상도 영상 1장을 얻어내었다. 예상한 바와 같이 영상정합과 영상복원에서 많은 시간이 소요되는 것을 볼 수 있고 반복 계산이 필요한 영상복원에서 총 소요시간의 약 70%가 소요되고 있음을 확인 할 수 있다.

위의 식 (2)를 얻기 위한 영상복원 알고리즘은 Fig. 4와 같다. Fig. 4에서는 각 단계마다 소요되는 시간이 나타나 있는데 주로 map generation, similarity cost calculation, update HR에서 많은 시간이 소모되는 것을 확인할 수 있다.

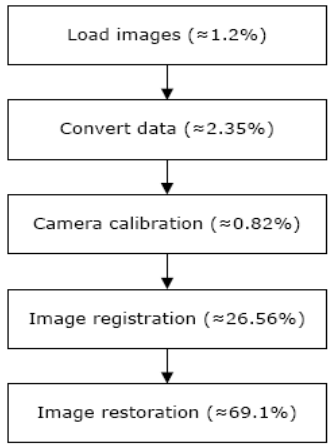

Fig. 3. 초해상도 알고리즘의 순서도와 소요시간

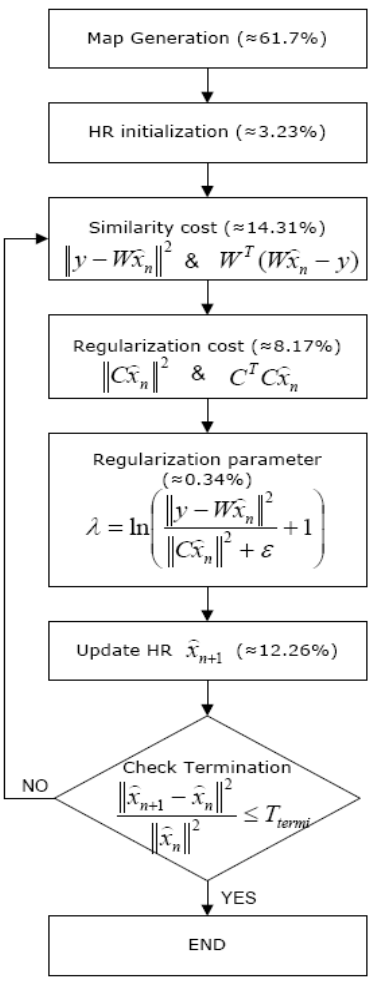

Fig. 4. 영상복원 알고리즘의 순서도와 소요시간

Update HR 과정은 식 (3)에 나타난 α_n을 계산하는 과정을 말하며 이는 병렬처리가 되지 않기 때문에 CUDA로 구현될 수 없다. 따라서 본 논문에서는 map generation과 similarity cost 과정 중 병렬처리가 가능한 함수만을 CUDA로 구현하였다.

3. 시험결과

시뮬레이션에서 사용할 HW 및 SW 제원은 아래 Table 1과 같다.

Table 1. 개발환경

HW	Workstation	Intel core2 Duo CPU T9400 @ 2.53GHz 2.53GHz, 2.93GB RAM
	GPGPU	NVIDIA Quadro FX 2700M
SW	O/S	Windows XP
	Language	Generic C + CUDA
	library	Math, CUDA

가. Map Generation

Map generation 함수는 registration parameter 정보와 카메라의 intrinsic parameter 정보를 반영하여 입력 저해상도 영상의 픽셀들과 기준 고해상도 영상의 픽셀들 사이의 위치관계를 계산하는 함수이다. 이 함수는 Fig. 4에서 확인할 수 있는 바와 같이 영상복원시간의 60%의 시간을 소요한다. 또한 각 픽셀값의 연산에 다른 픽셀 값이 이용되지 않기 때문에 CUDA를 이용하여 병렬처리하기 용이하다. 이러한 Map generation함수를 CUDA로 구현하기 위해서 먼저 함수의 연산과정에 대해 간단히 살펴보면 Fig. 5와 같은 연산이 각각의 픽셀에 대하여 수행된다.

병렬처리를 위해 소스코드를 수정하기 전 그래픽카드의 사양 및 입력영상의 크기에 맞도록 thread와 block 개수를 지정해야한다. 본 연구에서 사용된 그래픽 카드의 사양은 Quadro FX 2700M으로 block당 최대 thread의 개수는 512개이다. 본 논문에서는 720*480의 영상을 이용하므로 아래와 같이 thread 및 block의 개수를 지정하였다.

```
dim3 dimBlock(360, 1, 1)
dim3 dimGrid(iDivUp(width, dimBlock.x), iDivUp(height, dimBlock.y))
```

이 후 cudaMalloc와 cudaMemcpy를 이용하여 host 메모리의 데이터와 그래픽 카드의 메모리 데이터를 교환하고 각 픽셀단위로 연산이 이루어지고 CUDA 문법에 맞도록 소스코드를 수정하였다.

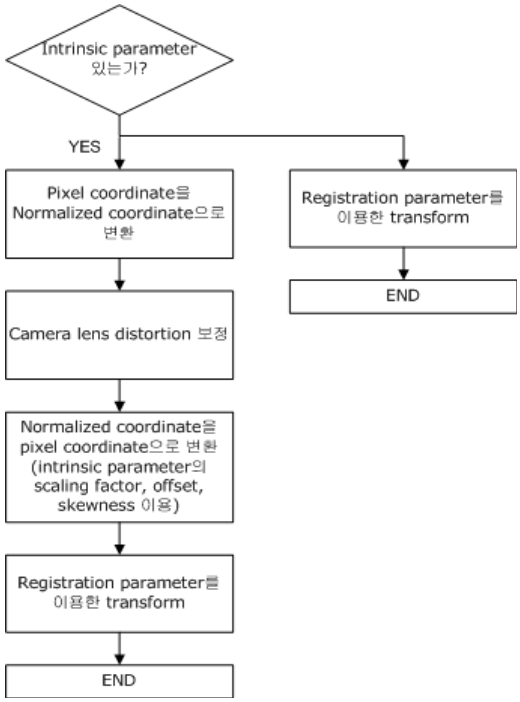

Fig. 5. Map generation 함수의 block diagram

Table 2는 Map generation 함수를 C++과 CUDA를 이용하여 구현한 결과 시간을 비교한 것이다.

Table 2. C++과 CUDA에 의한 Map generation 함수의 소모시간 비교

	C++을 이용한 Map generation 함수의 processing time	CUDA를 이용한 Map generation 함수의 processing time
#1	1399.83ms	108.67ms
#2	1396.07ms	108.20ms
#3	1392.47ms	108.53ms
#4	1390.52ms	108.11ms
#5	1395.37ms	108.03ms
평균 processing time	1394.85ms	108.31ms

Table 2에서 확인할 수 있는바와 같이 CUDA를 이용한 경우의 processing time은 C++만을 이용한 경우의 processing time에 비해 약 13배 정도 빨라짐을 확인할 수 있다. 이는 전체 초해상도 알고리즘의 processing time을 약 18% 단축시키는 결과를 나타내었다. 함수의 속도개선에 비해 전체 함수의 속도개선이 크지 않은 이유는 iteration에 의한 다른 함수의 반복된 호출에 의한 것으로 iteration 횟수에 따라 전체 함수의 processing time의 단축 효과가 결정되기 때문이다.

나. Similarity Cost

Similarity cost 함수는 map geneartaion 함수 다음으로 많은 시간이 소모되는 함수로 forward_HR2LR, backward_LR2HR, filter의 3개의 함수로 이루어져 있다. Forward_HR2LR함수는 초기 고해상도 영상 한 장으로부터 registration parameter를 이용하여 저해상도 시뮬레이션 영상들을 생성하고 실제 저해상도 영상들과 시뮬레이션 영상간의 차이를 구하는 함수이다. backward_LR2HR 함수는 이를 이용하여 고해상도 영상을 업데이트 하는 함수이다. 세 함수에서 소요되는 시간은 Table 3에서 확인할 수 있다.

Table 3. Similarity cost 함수의 내부함수 소요시간

함수 이름	소모시간
First Filter2D	50ms
Forward_HR2LR	82ms
Backward_LR2HR	54ms
Second Filter2D	57ms

Table 3에 따르면 Filter 함수에서 소모되는 시간이 크지만 함수 안의 조건문을 병렬처리가 가능하도록 변경하는 것이 불가능하여 CUDA로 구현하지 않았다. CUDA에서는 조건문을 만나면 조건문의 결과가 나타날 때까지 많은 시간을 소모하므로 다른 작업을 할 수 없어 효과가 크지 않다. 나머지 함수에도 조건문이 있으나 코드의 변형을 통하여 Forward_HR2LR 함수만이 CUDA로 구현되었다.

Forward_HR2LR 함수는 Fig. 6과 같이 Generate map 함수에서 생성된 저해상도 영상과 고해상도 영상의 위치관계(MapX_LR, MapY_LR)를 반영하여 고해상도 영상을 저해상도로 영상으로 만들어 실제 저해상도 영상과의 차이를 계산하는 함수이다. 그러나 MapX_LR과 MapY_LR의 데이터가 고해상도 영상의 크기(src_width, src_height)를 벗어나는 경우나 영상의 경계를 가리키는 경우 저해상도 영상의 픽셀 값은 아래와 같이 index를 이용하여 처리하였다.

```
float temp1 = MapX_LR[idx]/(src_width-1)
float temp2 = MapY_LR[idx]/(src_height-1)
unsigned int index1 = abs(floor(temp1));
unsigned int index2 = abs(floor(temp2));
unsinged int index = !(index1 || index2);
```

위에서 계산된 index는 if 문을 대신하는 지표로 data 값에 index를 적용하여 고해상도 영상의 크기를 벗어나는 경우 저해상도 영상의 픽셀 값을 0으로 처리하도록 하였다.

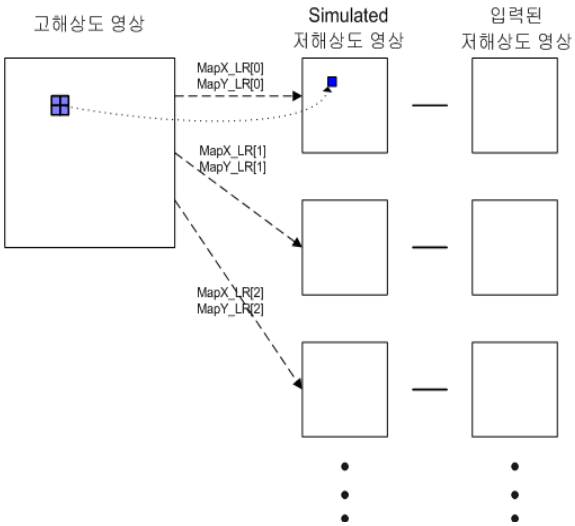

Fig. 6. Forward_HR2LR 함수의 개념도

Table 4는 Forward_HR2LR 함수를 C++과 CUDA를 이용하여 구현한 결과 소모시간을 비교한 것이다.

Table 4에서 확인할 수 있는 바와 같이 함수에서 소요되는 시간이 약 20% 향상되었으나 그 효과가 크지 않았다.

CUDA가 적용된 소스코드를 이용하여 초해상도 알고리즘을 시뮬레이션 한 결과 아래 Fig. 7과 같은 결과를 얻었다. 이는 C++로 구현한 것과 동일한 것으로 같은 결과를 더 빠른 시간에 얻어낼 수 있었다.

Table 4. C++과 CUDA에 의한 Forward_HR2LR 함수와 Backward_LR2HR의 소모시간 비교

	C++을 이용한 Forward_HR2LR 함수의 processing time	CUDA를 이용한 Forward_HR2LR 함수의 processing time
#1	81.8ms	65.4ms
#2	82.2ms	65.6ms
#3	84.6ms	65.4ms
#4	81.5ms	65.4ms
#5	82.2ms	65.5ms
평균 processing time	82.46ms	65.46ms

(a) bilinear interpolation을 적용한 결과

(b) 초해상도 알고리즘을 적용한 결과

Fig. 7. C++과 CUDA를 이용하여 시뮬레이션한 결과

4. 결 론

본 논문에서는 초해상도 알고리즘의 처리 속도를 향상시키기 위해 알고리즘의 일부인 map generation 함수와 similarity cost 함수를 CUDA로 구현한 결과를 나타내었다. Map generation 함수를 CUDA로 구현한 결과 C++로 구현했을 때 보다 약 13배 정도의 속도향상을 보여주었으나 similarity cost 함수는 20%정도의 속도향상만을 나타내 주었다. 알고리즘의 복잡성과 그래픽 카드의 성능에 따라 속도향상의 정도는 달라질 수 있으나 전반적으로 영상처리 속도시간의 감소효과를 보여주고 있으므로 초해상도 알고리즘을 CUDA로 구현하여 library화 한다면 실시간 처리도 가능해질 것이다.

References

[1] http://www.nvidia.com/object/cuda_education.html

[2] http://kr.nvidia.com/object/cuda_develop_kr.html

[3] http://www2.uic.edu/~agurmu2/CS525/final/

[4] Zhengyou Zhang, “A Flexible New Technique for Camera Calibration”, Technical Report MSR-TR-98-71.

[5] Barbara Zitová and Jan Flusser, “Image Registration Methods : A Survey”, Image and Vision Computing, Jun. 2003.

[6] S. C. Park, M. K. park and M. G. Kang, “Super-Resolution Image Reconstruction : A Technical Overview”, IEEE Signal Processing Magazine, May. 2003.

[7] R. Y. Tsai and T. S. Huang, “Multipleframe Image Restoration and Registration”, Advances in Computer Vision and Image Processing. Greenwich, CT : JAI Press Inc., pp. 317～339, 1984.

[8] Deshpande, S. D., “Max-Mean and MaxMedian Filters for Detection of Small-Targets”, Conference on Signal And Data Processing of Small Targets, SPIE, Vol. 3809, pp. 74～83, 1999.