GPU를 이용한 고속 영상 보간법 개발

최학남, 박은수, 김준철,정용한, 김학일
인해대학교 정보통신공학부 컴퓨터비전 연구실

Development of high-speed image interpolation method using CUDA

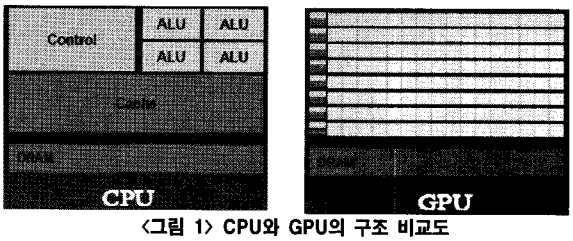
Xuenan Cui, Eunsoo Park, Junchul Kim, Younghan Jung, Hakil Kim
Computer Vision Lab., Inha University

Abstract - 본 논문에서는 GPU를 이용한 고속 보간법 개발방법에 대해 제안한다. GPU는 흔히 그래픽 연산에 사용되지만 최근에는 GPGPU가 각광을 받고 있다. 특히 NVIDIA에서 발표한 CUDA를 이용하면 GPU를 쉽게 접근하여 프로세싱 할 수 있어 많은 분야에서 GPU를 활용하고 있다. 본 논문에서는 실제 CUDA를 이용하여 여러 가지 보간법에 대한 알고리즘을 구현하여 CUDA의 성능을 확인하였다. CPU에서 구현한 알고리즘과 CUDA를 이용한 알고리즘을 비교했을 때 메모리 할당 및 전송부분을 제외한 수순 프로세싱 시간을 보면 GPU에서 훨씬 좋은 성능을 나타내었고, 메모리 할당 및 전송을 고려했을 때 작은 사이즈 영상에서는 오히려 역효과가 나타났고, 대용량 영상에서는 좋은 성능을 나타냄을 확인하였다.

1. 서 론

GPU(Graphics Processing Unit)는 1999년에 NVIDIA에서 처음 발표하였으며 주로 그래픽 처리 작업을 수행하였다. 최근 들어 GPU 장치의 비약적인 발전으로 인하여 일반 응용 프로그램에서도 GPU를 사용할 수 있게 되었다. GPU는 병렬처리와 수학연산에 특화 되어 있으며 트랜지스터가 흐름 제어나 데이터 캐싱보다 데이터 처리에 집중되어 있다. 따라서 데이터 처리가 많은 응용분야에서는 기존의 CPU를 이용하여 처리 하는것 보다는 GPU를 이용하면 훨씬 좋은 성능을 보장할 수 있다. 아래의 [그림 1]에서도 알 수 있듯이 CPU와 GPU의 구조특성상 GPU가 훨씬 많은 ALU를 가지고 있음을 알 수 있다. 이러한 구조적 특성 때문에 GPU에서의 처리속도가 훨씬 빠르게 된다.

일반 응용프로그램에서의 GPU의 사용을 편리하게 할 수 있도록 하기 위하여 2007년 2월에 NIDIA에서는 CUDA를 출시하였는데 이는 G80 시리즈 이상에서 사용할 수 있다. 따라서 최근 들어 많은 연구 분야들에서 GPU를 이용하여 고속 처리를 하고 있다. Y.Luo[1] 등은 CUDA를 이요한 Canny edge detection을 구현하였고, 염용진[2] 등은 블록암호 알고리즘을 CUDA를 이용하여 구현으며, L.Pan[3] 등은 의료영상의 segmentation을 CUDA로 구현하여 좋은 성능을 확보하였다.



<그림 1> CPU와 GPU의 구조 비교도

영상처리 분야에서 interpolation 방법은 더 정밀하고 정확하게 영상처리 결과를 얻고자 하거나, 저해상도 영상을 이용하여 고해상도 영상과 비슷한 성능을 확보하고자 할 때 사용되는 알고리즘이다. 예를 들면 컴퓨터 비전 분야에서의 웹 카메라를 이용한 물체인식을 시도하면 저화질 영상을 이용하기 때문에 알고리즘 구현 후 좋은 성능을 보지 못하는 경우가 종종 발생한다. 이러한 경우 보간법을 이용하여 화질 개선을 진행하면 보다 우수한 성능을 기대할 수 있다. 하지만 동영상에서 보간법을 사용하면 처리시간이 문제가 되기 때문에 이를 고속화할 필요가 있다. 또한 PCB 검사에서 사용하는 영상과 같은 대용량 영상에 대한 보간법은 많은 수행시간을 필요로 한다. 따라서 본 논문에서는 CUDA를 이용한 고속 영상 보간법에 대해 개발하고 성능을 확인하고자 한다.

본 논문은 서론, 본론, 결론으로 구성되었으며, 서론에서는 연구배경에 대해 간략하게 소개하고, 본론에서는 CUDA를 이용한 보간법 알고리즘 구현과 실험결과에 대해 분석하였으며, 마지막으로 결론부분에서 간략한 결론과 향후 수행해야할 과제에 대하여 서술하였다.

2. 본 론

2.1 영상 보간 알고리즘

영상 보간법은 차수에 의하여 Zero Order, First Order, High Order 보간법으로 분류할 수 있다. 흔히 Zero order 보간법은 Nearest Neighbor 보간법을 나타내고, First order 보간법은 선형보간법을 나타내며, Bicubic, Lagrange 등 보간법은 High oder 보간법이라고 볼 수 있다.

보간법은 종류는 방법에 따라 수십 가지가 존재한다. 본 논문에서는 High-order interpolation 인 Bicubic, B-spline, Lagrange에 대해서만 언급하며 관련 수식은 아래의 식 1, 식2, 식 3과 같다[4].

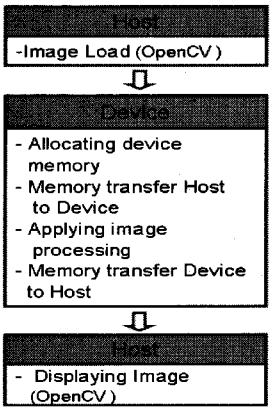
$$w_{cubic\_4} = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1 & 0 \le |x| < 1 \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a & 1 \le |x| < 2 \\ 0 & elsewhere \end{cases} \quad (1)$$

$$w_{b\_spline} = \begin{cases} (\frac{1}{2})|x|^3 - |x|^2 + \frac{2}{3} & 0 \le |x| < 1 \\ (-\frac{1}{6})|x|^3 + |x|^2 - 2|x| + \frac{4}{3} & 1 \le |x| < 2 \\ 0 & elsewhere \end{cases} \quad (2)$$

$$w_{lagra\_4} = \begin{cases} (\frac{1}{2})|x|^3 - |x|^2 - (\frac{1}{2})|x| + 1 & 0 \le |x| < 1 \\ -(\frac{1}{6})|x|^3 + |x|^2 - (\frac{11}{6})|x| + 1 & 1 \le |x| < 2 \\ 0 & elsewhere \end{cases} \quad (3)$$

2.2 CUDA를 이용한 영상 보간 알고리즘 구현[5]

그림 2는 전체 프로그램 구성도를 나타낸다. OpenCV는 영상처리 인터페이스를 지원해주는 Intel에서 제공하는 무료 라이브러리이다. 본 논문에서는 OpenCV를 이용하여 영상을 로드하고 출력하였고, 영상 보간 알고리즘은 GPU의 Global 메모리를 이용하여 개발하였다.



<그림 2> CUDA 프로그램 흐름도

그림 3은 GPU에 메모리를 할당하는 부분과 입력영상을 업로드하는 부분을 포함하고 있다. 그림 3에서 cudaMalloc함수는 GPU에 메모리를 할당하는 함수이고, cudaMemcpy는 host에서 device로 데이터를 올리거나, device에서 host로 데이터를 다운받는 함수이다. 보간법을 구현은 x방향으로 nx개의 픽셀을 삽입하는 부분과 y방향으로 ny개의 픽셀을 삽입하는 부분으로 나누어진다. 따라서 메모리 할당 시 두 가지 결과를 저장할 수 있도록 설정해야 한다.

그림 4는 실제 GPU에서 보간법 알고리즘을 실행하는 과정을 나타낸다. 그림에서 block은 매개 block의 thread 개수를 설정하는 부분으로써 본 논문에서는 블록 당 256개의 스레드를 할당하였으며, Grid는 Grid내에 포함되는 block의 개수를 설정하는데 본 논문에서는 영상의 사이즈에 종속되게 설정하였다. 그림에서 d_input은 입력영상이고, d_output_x는 x방향으로 보간된 영상이고, d_output은 양방향으로 보간 된 영상이고, height와 width는

각각 입력영상의 길이와 너비를 나타내고, nx, ny는 각각 x방향으로 보간할 픽셀의 개수와 y방향으로 보간 할 픽셀의 개수를 나타내고, D_wx1, D_wx2, D_wx3, D_wx4는 x방향으로 보간 할 때 사용되는 계수들을 나타내며, D_wy1, D_wy2, D_wy3, D_wy4는 y방향으로 보간 할 때 사용되는 계수들을 나타낸다.

```
cudaMalloc((void**)&d_input, width*height);

cudaMalloc((void**)&d_output_x, (width)*(nx+1)*height);

cudaMalloc((void**)&d_output, (width)*(nx+1)*(height)*(ny+1));

cudaMemcpy(d_input, input, imageSize, cudaMemcpyHostToDevice);
```

〈그림 3〉 메모리 할당 및 데이터 전송

```
dim3 block = dim3(16,16,1);
dim3 grid = dim3(width/block.x,height/block.y);

Interpocal_x<<<grid, block>>>(d_input,d_output_x, width,
   height,nx,D_wx1,D_wx2,D_wx3,D_wx4);

dim3 blocky = dim3(16,16,1);
dim3 gridy = dim3((width*(nx+1))/block.x,height/block.y);
Interpocal<<<gridy, blocky>>>(d_output_x,d_output,
   width*(nx+1),height,ny,D_wy1,D_wy2,D_wy3,D_wy4);
```

〈그림 4〉 Kernel 실행

```
__global__ void Interpocal_x (input arguments)
{
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;

  d_output_x[y*w*(nx+1)+x*(nx+1)] = d_input[y*w+x] ;
  ...
__syncthreads();
}
```

〈그림 5〉 보간법에 대한 Global 함수

그림 5는 실제 GPU상의 global 메모리에서 실행하게 될 보간 알고리즘을 나타낸다. 그림 5에서 보시다시피 영상의 인덱스는 block ID, Block의 차원, 그리고 Thread ID를 이용하여 접근할 수 있다. 보간 알고리즘은 기존의 알고리즘과 동일하게 구성하여 CUDA를 이용하여 실행하면 GPU상의 thread에 분산되어 실행하게 되는데 특히 단순 반복계산이 많거나 대용량 영상에서 많은 성능향상을 가져오게 된다.

2.3 실험 환경 및 결과

본 논문에서는 CUDA를 이용한 보간법의 성능을 확인하기 위하여 표 1과 같은 실험환경에서 실험을 진행하였다.

〈표 1〉 실험환경

CPU	Intel® CoreTM2 CPU 6600 @ 2.4GHz, 2401 MHz
메모리	2047MB
캐시	L1: 32KB , L2: 4MB
SSEn 지원 여부	SSE, SSE2, SSE3, SSSE3 명령어 지원
GPU	Nvidia Gforce 8800GT
운영체제	Windows XP
개발툴	Microsoft Visual Studio 2005

표 2 는 Bicubic 알고리즘에 대한 CPU상에서의 수행시간에 대한 결과이다. 수행시간은 10번 반복하여 실행시킨 결과에서 최대값, 최소값, 평균값을 취하였으며 단위는 msec이다. 보간할 픽셀의 개수는 nx, ny로 나타냈으며 각각 1, 2, 3으로 설정하였다. 여기서 nx, ny를 각각 1로 설정하면 실제 영상은 4배 커지고, 2로 설정하면, 9배 커지며, 3으로 설정하면 16배 커진다.

표 3은 GPU상에서 수행한 수행시간에 대한 결과이다. GPU상에서 실행하려면 CPU에서 GPU로 데이터를 전송하고 다시 GPU에서 CPU로 데이터를 받는 과정이 필요하다. 표에서 Memory upload는 CPU에서 GPU로 데이터를 전송하는데 필요한 시간이고 Memory download는 GPU에서 CPU로 데이터를 전송하는데 필요한 시간이며, Total time은 메모리 전송시간을 포함한 전체 시간을 나타낸다.

〈표 2〉 CPU에서의 수행시간

Image	Size	nx / ny	Processing Time (msec)	Mean (msec)	Max (msec)
Airpalne	256×256	1/1	11.45	**14.66**	40.35
		2/2	22.93	**26.33**	55.34
		3/3	38.41	**43.92**	69.80
Pepper	512×512	1/1	43.75	**47.66**	73.81
		2/2	94.57	**98.28**	119.72
		3/3	165.65	**171.22**	198.29
CAM_image	4048×4120	1/1	4129.09	**4403.84**	4678.59
		2/2	8018.47	**8022.07**	8025.66
		3/3	11227.00	**11608.21**	11989.41

〈표 3〉 GPU에서의 수행시간

Image	Size	nx / ny	Processing Time (msec)	Memory Upload (msec)	Memory Download (msec)	Total time (msec)
Airpalne	256×256	1/1	2.31	34.28	0.26	**36.85**
		2/2	4.98	31.65	0.57	**37.20**
		3/3	13.22	32.71	0.99	**46.92**
Pepper	512×512	1/1	12.08	30.40	1.00	**43.48**
		2/2	19.96	31.33	1.77	**53.06**
		3/3	52.50	30.50	2.97	**85.97**
CAM_image	4048×4120	1/1	514.34	41.12	40.22	**595.68**
		2/2	1242.03	42.69	89.53	**1374.25**
		3/3	2253.32	49.21	183.16	**2485.69**

〈표 4〉 CPU와 GPU에서의 수행시간 비교

Image	Size	nx / ny	CPU	GPU1	GPU2	Speed-up (GPU1)	Speed-up (GPU2)
Arpalne	256×256	1/1	14.66	2.31	36.85	**6.35**	**0.40**
		2/2	26.33	4.98	37.20	**5.29**	**0.71**
		3/3	43.92	13.22	46.92	**3.32**	**0.94**
Pepper	512×512	1/1	47.66	12.08	43.48	**3.95**	**1.10**
		2/2	98.28	19.96	53.06	**4.92**	**1.85**
		3/3	171.22	52.50	85.97	**3.26**	**1.99**
CAM_ image	4048×4120	1/1	4403.84	514.34	595.68	**8.56**	**7.39**
		2/2	8022.07	1242.03	1374.25	**6.46**	**5.83**
		3/3	11608.21	2253.32	2485.69	**5.15**	**4.67**

표 4는 CPU와 GPU에서의 수행시간을 비교한 도표이다. 표에서 GPU1은 순수 실행시간은 나타내고, GPU2는 메모리 전송시간을 포함한 시간을 나타내고, Speed-up(GPU1)은 CPU와 GPU의 순순 실행시간을 비교한 결과이고, Speed-up(GPU2)는 CPU와 GPU의 메모리전송시간을 포함한 실행시간을 비교한 결과이다. 실제 대용량 영상에서 메모리 전송부분을 포함하더라도 4~7배의 성능향상을 가져옴을 확인할 수 있다.

3. 결 론

본 논문에서는 CUDA를 이용한 영상 보간법에 대한 개발을 통하여 GPU상에서의 영상처리 성능을 확인하였다. 실험결과 GPU상에서의 순수 수행시간은 CPU보다 훨씬 좋은 성능을 나타내는 것을 확인할 수 있었고, CPU와 GPU사이의 데이터 전송에 필요한 시간을 고려하면 작은 사이즈의 영상에서는 오히려 역효과를 나타내는 사실도 확인하였다. 하지만 대용량 영상에서는 이러한 메모리 전송시간을 고려하더라도 성능향상을 가져오는 것을 확인하였다. 본 논문에서 개발한 방법은 global 메모리만 이용하였기 때문에 추후에 더 좋은 성능의 알고리즘을 개발할 여지가 있으며 CUDA에서 제공하는 비동기 메모리 전송방식을 이용하여 메모리 전송시간을 최대한 줄일 수 있는 방법에 대한 연구가 더 필요하다.

[참 고 문 헌]

[1] Y.Luo, R.Duraiswami, "Canny edge detection on NVIDIA CUA," 978-1-4244-2340-8/08 2008 IEEE

[2] 임용진, 조용국, "GPU 연산라이브러리 CUDA를 이용한 붉록암호 고속구현," 정보보호학회논문지, pp.23-32, 2008.06

[3] L.Pan, L.Gu, J.Xu, "implementation of medical image segmentation in CUDA," Proceeding of the 5th international conference on imformation techinology and application in Biomedicine, pp.82-85, 2008

[4] T.M. Lehmann, "Survey: Interpolation Methods in Medical Image Processing", IEEE Transactions on Medical Image, vol.18 NO. 11,pp.1049-1075, 1999

[5] NVIDIA_CUDA_Programming_Guide_2.0, 2008