결합형 양방향 필터를 이용한 실시간 깊이 영상 보정 방법

*신동원 **이상범 ***호요성
광주과학기술원 실감방송연구센터
{dongwonshin, sblee, hoyo}@gist.ac.kr

Real-time Depth Image Refinement using Joint Bilateral Filter

*Dong-Won Shin **Sang-Beom Lee ***Yo-Sung Ho
Gwangju Institute of Science and Technology (GIST)

요약

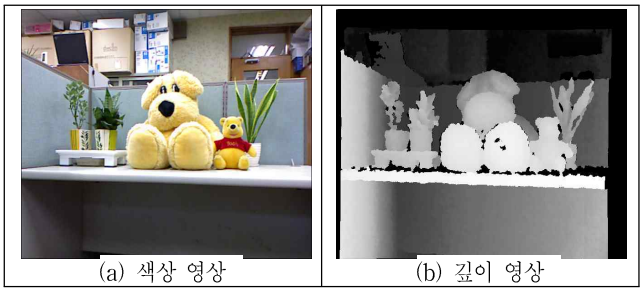
본 논문에서는 결합형 양방향 필터를 이용하여 실시간으로 깊이 영상을 구하는 방법을 제안한다. 제안한 방법에서는 Kinect 깊이 카메라로부터 얻은 깊이 영상의 화질을 실시간으로 향상시키기 위해 GPU 내의 상수 메모리와 2차원 영상 처리에 적합한 텍스쳐 메모리를 사용했다. 또한, 단일 화소에 대한 결합형 양방향 필터 연산을 각 GPU 쓰레드(thread)에 할당한 다음 병렬로 처리하여 계산량을 현저히 감소시킨다. 실험 결과를 통해, 제안한 실시간 깊이 영상 보정 방법이 깊이 영상의 화질을 향상시켰고, 초당 260화면의 속도로 동작하는 것을 확인했다.

1. 서론

최근 3차원 비디오 시스템은 초고해상도를 특징으로 하는 UHDTV, 초고속 인터넷 망을 이용하여 제공되는 양방향 서비스인 IPTV 시스템 등과 더불어 중요한 차세대 방송 시스템 중의 하나로 각광 받고 있다. 3차원 비디오 시스템은 3차원 비디오 장면 정합, 압축, 렌더링 등의 다양한 주요 요소로 구성되어 있으며, 3차원 비디오 콘텐츠는 스테레오 카메라, 다시점 카메라, 깊이 카메라와 같은 다양한 영상 제작 도구들을 이용하여 만들어 질 수 있다. 그리고 이 콘텐츠들은 3차원 비디오 압축 방법을 통해 부호화되어 영상출력장치에 전달되고, 이러한 장치들은 전달된 영상 정보를 DIBR (depth image-based rendering) 기술을 통해 재현하여 영상을 화면에 출력한다 [1].

DIBR 기술을 이용한 방송 서비스는 기존에 사용되던 스테레오 영상을 이용한 3차원 입체 영상 서비스에 비해 단일 시점 색상 영상과 색상 영상에 비해서 상대적으로 압축 효율이 좋은 깊이 영상을 사용하기 때문에 매우 적은 전송량을 사용한다. 또한, 기존에 방송되고 있는 영상에 추가적인 깊이 영상의 전송만으로 3차원 영상을 제공할 수 있기 때문에 현재 서비스 되고 있는 시스템과 역호환성을 유지할 수 있으며, 이는 사용자의 선택에 의한 2차원과 3차원 서비스간의 전환을 가능하게 한다 [2].

현재에 이르러 Microsoft 사의 Kinect 깊이 카메라가 파격적인 가격으로 출시됨에 따라 일반 가정에서도 3차원 비디오를 쉽게 접할 수 있게 되었다. 앞서 소개한 DIBR 기술은 Kinect 깊이 카메라로부터 획득한 색상 영상과 깊이 영상을 사용하여 3차원 장면을 쉽게 복원할 수 있다. 하지만 그림 1(b)에서 볼 수 있듯이, Kinect 깊이 카메라로부터 획득한 깊이 영상은 깊이값 추정에 실패한 영역들을 많이 포함하고 있다. 이러한 영역은 Kinect 깊이 카메라에 장착된 적외선 송신기와 수신기의 물리적인 거리 차이로 인한 폐색 영역 발생, 객체의 경계 불일치 등 다양한 센서 잡음 때문에 발생한다. 그림 1(a)와 그림 1(b)에서 볼 수 있듯이, 원본 깊이 영상에서 책상 위와 물체 주변에 검은색으로 표현되어 있는 센서 잡음을 확인할 수 있다. 이러한 깊이 영상의 오차는 3차원 비디오 시스템의 부호화, 복호화 등의 계산 과정에서 오차가 그대로 전파되기 때문에 재현 단에 이르렀을 때 양질의 합성영상을 시청자에게 제공할 수 없게 만든다.



(a) 색상 영상 (b) 깊이 영상

그림 1. Kinect 깊이 카메라로부터 획득한 영상.

깊이 영상 오차를 제거하기 위해 최근 다양한 방법이 연구되고 있다. 이 가운데 대표적인 방법으로 결합형 양방향 필터(joint bilateral filter, JBF)를 들 수 있다 [3]. 이 방법은 표적 화소와 주변 화소 간의 거리 차이를 반영하는 공간 (spatial) 필터와 화소 값 차이를 반영하는 범위 (range) 필터를 사용하는데, 범위 필터를 보조 영상, 즉, 색상 영상의 것으로 대체함으로써 색상 영상의 경계 정보를 깊이 영상에 그대로 반영한다. 따라서, 두 영상 간의 객체 경계 불일치 문제를 효과적으로 제거함과 동시에 깊이 값 오차를 보정할 수 있다.

본 논문에서는 결합형 양방향 필터를 기반으로 하여 실시간으로 깊이 오차를 제거하는 방법을 제안한다. 제안한 방법은 Kinect 깊이

카메라로부터 획득한 색상 영상을 이용하여 깊이 영상에 대해 결합형 양방향 필터를 적용한다. 이 필터는 성능을 결정하는 다양한 인자와 Gaussian 함수와 같은 복잡한 수식이 포함되는 필터이기 때문에, 이를 실시간으로 처리하도록 구현하기 위해 CUDA(compute unified device architecture) 병렬 프로그래밍을 통해 각 화소에 대한 계산을 동시에 처리함으로써 수행시간을 줄인다.

2. 제안하는 방법

기존에는 중앙 처리 장치(central processing unit, CPU)를 이용한 순차적인 프로그래밍이 널리 사용되었으나, 최근에는 그래픽 처리 장치(graphic processing unit, GPU)를 이용한 병렬 프로그래밍이 많은 관심을 받고 있다. 특히, NVIDIA 사에서 CUDA라는 C언어 기반의 병렬처리 라이브러리를 배포하면서 연산 시간 단축에 크게 기여하고 있다 [4].

제안하는 방법에서는 Kinect로부터 얻은 동일한 장면에 대한 깊이 영상과 색상 영상을 앞서 설명한 CUDA를 이용하여 구현된 결합형 양방향 필터를 통해 실시간으로 보정하는 방법을 다룬다. 그림 2는 제안하는 방법의 순서도를 나타낸다.

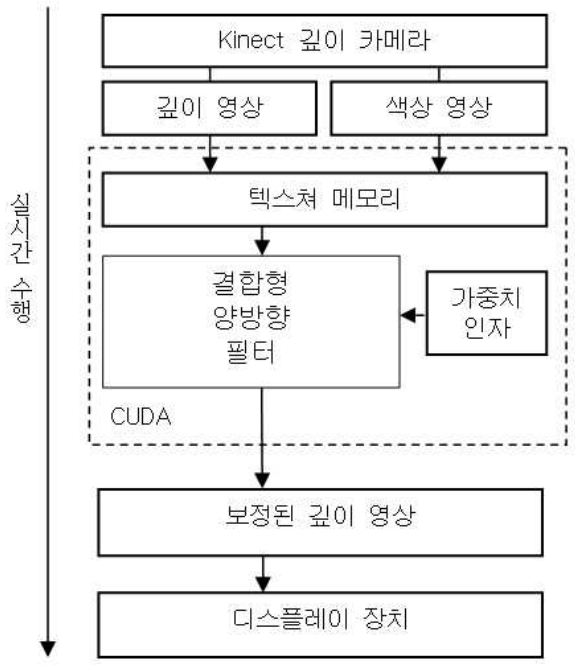


그림 2. 제안하는 방법의 순서도

먼저 Kinect로부터 얻은 깊이 영상과 색상 영상을 2차원 영상 처리에 적합하게 설계된 CUDA의 텍스쳐 메모리에 적재한 다음 이 영상에 대한 결합형 양방향 필터 계산을 병렬적으로 수행한다. 단일 화소에 대한 결합형 양방향 필터 계산을 GPU의 쓰레드 함수 하나에 할당하고 영상에 존재하는 화소의 갯수만큼 쓰레드를 동시에 수행함으로써 속도를 개선시킨다.

또한, 계산 시간을 줄이기 위해서 결합형 양방향 필터 내부에서 계산할 가중치 인자를 쓰레드 함수 실행 이전에 미리 계산해두고, 그 값이 필요할 때에만 가져다 쓰는 방법을 사용한다. 그리고 이 가중치 인자는 GPU 내부의 다른 메모리들보다 상대적으로 읽기 접근 속도가 빠른 상수 메모리에 저장하여 수행 속도를 더욱 향상시킨다.

가중치 인자를 계산할 때에는 커널의 크기가 커짐에 따라 연산량이 급격하게 증가한다. 이러한 2차원 Gaussian 함수는 1차원 함수의 곱으로 분할할 수가 있기 때문에, 이를 이용하여 보다 빠른 처리 속도를 보여주는 필터를 구현한다 [5].

$$D_o(x,y) = \frac{\sum_{u\in u_p}\sum_{v\in v_p} W(u,v)D_i(u,v)}{\sum_{u\in u_p}\sum_{v\in v_p} W(u,v)} \quad (1)$$

여기서 D_i는 입력 깊이 영상을 나타내고 W는 가중치 인자이다. 이것을 (u,v)에 위치한 화소에 대해서 각각을 계산하고 가중치 인자의 총합으로 나누는 정규화를 거쳐서 D_o를 계산한다. u와 v는 필터 커널 내부의 특정 위치를 나타내는 벡터 $\vec{u}_p$와 $\vec{v}_p$의 원소이다. 따라서 다음과 같이 나타낼 수 있다.

$$\vec{u}_p = \{x-r,\cdots,x+r\} \quad (2)$$

$$\vec{v}_p = \{y-r,\cdots,y+r\} \quad (3)$$

수식 (2)와 수식 (3)에서 각각의 기호를 그림으로 설명하면 그림 3과 같이 나타낼 수 있다. 검은색으로 표시된 격자는 영상의 화소들을 나타내고 바깥쪽의 두꺼운 선으로 표시된 사각형은 필터 커널을 나타낸다. 필터 커널에서 (u,v)는 인접 화소의 위치를 나타내고 (x,y)는 중심 화소의 위치를 나타낸다. 또한 r은 필터 커널의 반지름을 나타낸다. 앞에서 설명한 벡터 $\vec{u}_p$와 $\vec{v}_p$는 수평과 수직의 위치에 나타내었다.

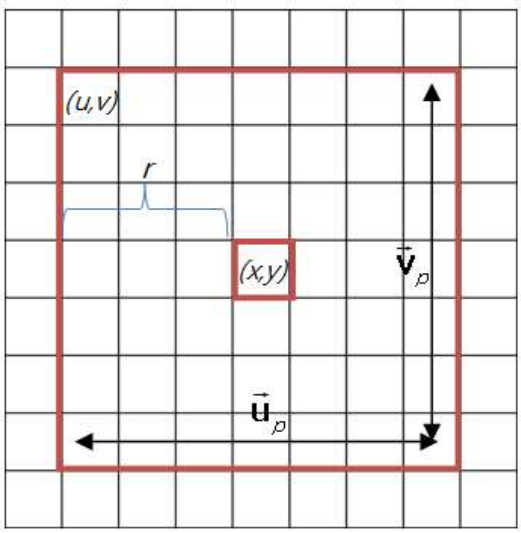


그림 3. 기호에 대한 설명

또한 가중치 인자 W는 다음과 같이 정의된다.

$$W(u,v)=\begin{cases} 0 & , \quad if \quad D_i(u,v)=0 \\ g(u,v)\cdot f(u,v) & , \quad otherwise \end{cases} \quad (4)$$

*f(u,v)*는 거리 차이에 따라 결정되는 공간 필터를 나타내고, *g(u,v)*는 필터 커널에서 표적 화소와 주변 화소와의 색상 차이에 따라 결정되는 범위 필터를 나타낸다. 이 두 함수는 모두 Gaussian 함수를 이용하여 다음과 같이 정의된다.

$$g(u,v)=\exp\{-\frac{|I(x,y)-I(u,v)|^2}{2\sigma_R{}^2}\} \quad (5)$$

$$f(u,v)=\exp\{-\frac{(x-u)^2+(y-v)^2}{2\sigma_D{}^2}\} \quad (6)$$

여기서 σ_R 과 σ_D는 각각의 Gaussian 함수에 대한 표준 편차를 의미한다. 이 값에 따라 Gaussian 분포의 폭이 결정된다.

2. CUDA

CUDA는 GPU를 이용한 대용량의 데이터에 대한 병렬 계산을 하는데 기반이 되는 통합 개발 환경이다. 본 논문에서는 CUDA를 이용하여 앞서 소개한 결합형 양방향 필터의 연산을 병렬적으로 수행함으로써 계산 시간을 줄인다.

2-1. 상수 메모리

제안한 방법은 연산 속도 향상을 위해 결합형 양방향 필터에서 사용되는 가중치 인자들을 미리 계산한 뒤 그 값을 CUDA에서 제공하는 상수 메모리에 저장하고 실제 계산 수행 시에는 이를 참조하여 쓰는 방법을 사용하였다. 상수 메모리는 DRAM에 있는 데이터를 읽기 전용으로 사용하며 캐시를 지원하는데 최초로 읽어오는 데이터는 DRAM에서 가져오기 때문에 GPU클럭으로 최소 400, 최대 500 사이클이 소요되지만 한번 캐시에 올라온 값을 반복하여 재사용할 때는 레지스터와 동일한 속도로 사용할 수 있게 되어 데이터에 빠르게 접근할 수 있도록 도와준다 [6]. 이러한 방법을 사용하여 본 논문에서는 상수 메모리와 미리 계산된 가중치 인자를 이용해서 속도를 향상시킨다.

2-2. 텍스쳐 메모리

CUDA에서 2차원 영상 처리와 관련하여 유용하게 쓰일 수 있는 것이 바로 텍스쳐 메모리이다. 텍스쳐 메모리는 오프칩인 DRAM에 대한 메모리 요청을 줄여줌으로써 매우 효과적인 대역폭을 제공할 수 있다. 다시 말해, 텍스쳐 캐시는 공간 구역성을 자주 드러내는 메모리 접근 패턴을 가진 그래픽스 어플리케이션을 위해 자주 사용된다 [7]. 여기서 공간 구역성이란 일단 하나의 기억 장소가 참조되면 그 근처의 기억 장소가 계속 참조되는 경향이 있는 특성을 의미한다 [8]. 2차원 영상 처리 어플리케이션에서는 이러한 공간 구역성이 잘 드러나는 경우를 많이 찾아볼 수 있기 때문에 CUDA의 텍스쳐 메모리는 2차원 영상 처리에 있어서 매우 우수한 성능을 보인다. 따라서, 제안하는 방법에서는 텍스쳐 메모리에 색상 영상과 깊이 영상을 미리 적재하고 결합형 양방향 필터의 계산 시에 빠르게 접근 할 수 있도록 함으로써 실시간의 성능을 낼 수 있도록 설계를 했다.

3. 실험 결과

제안한 방법의 성능을 평가하기 위해서 Geforce GTX 680 그래픽 카드를 이용하여 GPU 프로그래밍을 수행했다. 영상의 해상도는 색상 영상과 깊이 영상 모두 640x480이다. Kinect에서 기본적으로 얻어지는 영상의 해상도는 색상 영상이 640x480이고 깊이 영상이320x240 이지만 OpenNI의 depth generator를 사용하여 깊이 영상에 대해서 640x480으로 업샘플링한 영상을 사용하였다 [9]. 그리고, 결합형 양방향 필터에서 의 값은 0.1로 의 값은 5로 설정했다.

그림 4와 그림 5는 원본 깊이 영상과 결합형 양방향 필터를 이용하여 보정한 깊이 영상의 차이를 보여준다. 그림에서 보이는 것처럼 원본 깊이 영상과 보정된 깊이 영상의 차이가 육안으로도 구별할 수 있을 정도로 뚜렷한 것을 확인할 수 있다. 원본 깊이 영상에서는 깊이 값이 존재하지 않는 부분이 객체 경계 주변으로 나타나 있지만 보정된 깊이 영상에서는 이런 영역을 채워서 깔끔한 깊이 영상을 획득할 수 있었다.

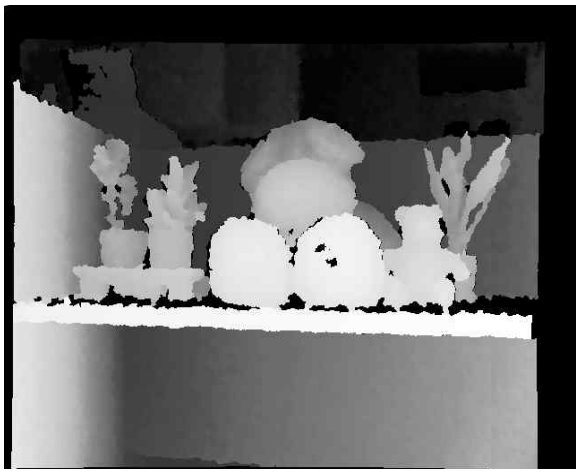


그림 4. 원본 깊이 영상

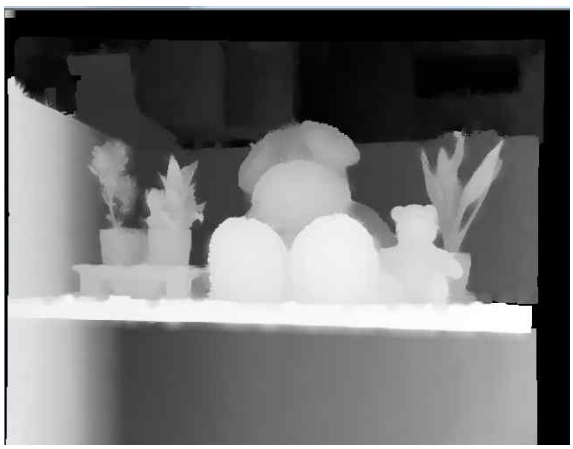


그림 5. 보정된 깊이 영상

두번째로, 제안한 깊이 영상 보정 방법을 Intel Xeon CPU로 수행된 영상과 Geforce GTX 680 GPU로 수행된 영상을 그림 6과 그림 7에 나타내었다.

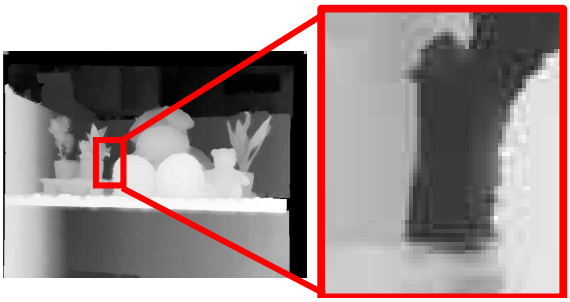


그림 6. CPU를 이용하여 계산된 영상

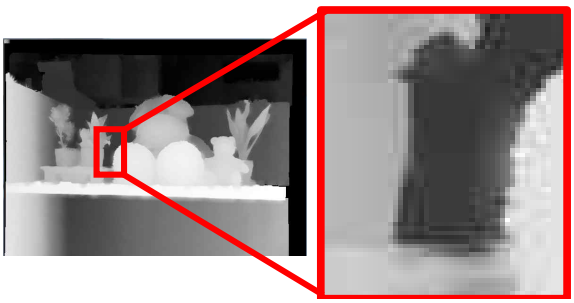


그림 7. GPU를 이용하여 계산된 영상

표 1. CPU와 GPU기반의 알고리즘 수행시간 비교

테스트영상 (640x480)	CPU(s)	GPU(s)	증감율(%)
Bowling	3070.0	3.73	-99.87%
	3009.0	3.72	-99.87%
	3058.0	3.71	-99.87%
	3197.0	3.73	-99.88%
	3020.0	3.72	-99.87%
Cloth	3166.0	3.73	-99.88%
	3108.0	3.74	-99.88%
	3111.0	3.74	-99.88%
	3133.0	3.73	-99.88%
	3105.0	3.74	-99.88%
Cones	3058.0	3.73	-99.87%
	3151.0	3.73	-99.88%
	3249.0	3.74	-99.88%
	3050.0	3.73	-99.87%
	3163.0	3.74	-99.88%
Lampshade	3091.0	3.72	-99.88%
	3073.0	3.73	-99.87%
	3091.0	3.73	-99.87%
	3184.0	3.73	-99.88%
	3078.0	3.72	-99.87%
Teddy	3201.0	3.74	-99.88%
	3073.0	3.74	-99.87%
	3090.0	3.74	-99.87%
	3072.0	3.74	-99.87%
	3223.0	3.74	-99.88%

각 영상의 오른쪽에 영상의 일부를 확대하여 나타내었다. 확대된 두 영상을 비교하여 보았을 때 전혀 화질의 차이가 존재하지 않는 것을 확인할 수 있다. 하지만 수행 시간의 관점에서 비교를 해보았을 때 GPU 프로그래밍을 이용하여 필터를 수행한 것이 CPU 프로그래밍을 이용했을 때보다 현저하게 빠른 것을 확인할 수 있었다. 표 1은 다양한 영상 샘플에 대해서 CPU와 GPU를 이용하여 계산한 수행 시간을 나타낸 것이다. GPU를 사용한 경우 99% 이상의 수행 시간 단축 효과가 있음을 확인할 수 있었다.

4. 결론

각 영상의 오른쪽에 영상의 일부를 확대하여 나타내었다. 확대된 두 영상을 비교하여 보았을 때 전혀 화질의 차이가 존재하지 않는 것을 확인할 수 있다. 하지만 수행 시간의 관점에서 비교를 해보았을 때 GPU 프로그래밍을 이용하여 필터를 수행한 것이 CPU 프로그래밍을 이용했을 때보다 현저하게 빠른 것을 확인할 수 있었다. 표 1은 다양한 영상 샘플에 대해서 CPU와 GPU를 이용하여 계산한 수행 시간을 나타낸 것이다. GPU를 사용한 경우 99% 이상의 수행 시간 단축 효과가 있음을 확인할 수 있었다.

감사의 글

본 연구는 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (NO. 2012-0009228).

참고 문헌

[1] C. Fehn, "Depth-image-based rendering (DIBR), compression and transmission for a new approach on 3-D TV," Proc. of SPIE Conference Stereoscopic Displays and Virtual Reality Systems, vol. 5291, pp. 93-104, Jan. 2004.

[2] K. Jung, Y. Park, J. Kim, H. Lee, K. Yoo, N. Hur and J. Kim, "2D/3D Mixed Service in T-DMB System Using Depth Image Based Rendering," International Conference on Advanced Communication Technology, pp. 1-2, Feb. 2008.

[3] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," ACM Transactions on Graphics, vol. 26, no. 3, pp. 1-5, July 2007.

[4] CUDA Reference Manual, Nvidia, 2011.

[5] 황선규, *영상 처리 프로그래밍 by Visual C++*, 한빛미디어, 2007, pp. 366-370.

[6] 정명훈, *CUDA 병렬 프로그래밍*, 프리렉, 2011, p. 121.

[7] J. Sanders and E. Kandrot, *CUDA by example*, 2010, p. 116.

[8] 전산용어사전편찬위원회, *컴퓨터 인터넷 IT용어 대사전*, 일진사, 2005.

[9] OpenNI User Guide, OpenNI, 2011.