다중 GPU 기반의 고속 다시점 깊이맵 생성 방법

고은상 호요성
광주과학기술원
{esko, hoyo}@gist.ac.kr

Multi-GPU based Fast Multi-view Depth Map Generation Method

Eunsang Ko Yo-Sung Ho
Gwangju Institute of Science and Technology (GIST)

요 약

3차원 영상을 제작하기 위해서는 여러 시점의 색상 영상과 함께 깊이 정보를 필요로 한다. 하지만 깊이 정보를 얻을 때 사용하는 ToF 카메라는 해상도가 낮으며 적외선 신호의 주파수 문제 때문에 최대 3대까지 사용할 수 있다. 따라서 깊이 정보를 색상 영상과 함께 사용하기 위해서 깊이 정보의 업샘플링이 필수적이다. 업샘플링은 깊이 정보를 색상 카메라 위치로 3차원 워핑하고 결합형 양방향 필터(joint bilateral filter, JBF)를 사용하여 빈 영역을 채우는 방법으로 진행된다. 업샘플링은 오랜 시간이 소요되지만 그래픽스 프로세싱 유닛(graphics processing units, GPU)를 이용하여 빠르게 수행될 수 있다. 본 논문에서는 다중 GPU의 병렬 수행을 통하여 빠르게 다시점 깊이맵을 생성할 수 있는 방법을 제안한다. 다중 GPU 병렬 수행은 범용 목적 GPU(general purpose computing on GPU, GPGPU) 중의 하나인 CUDA를 이용하였으며, 본 논문에서 제안된 방법을 이용하여 3개의 GPU 사용한 실험 결과 초당 35 프레임의 다시점 깊이맵을 생성했다.

1. 서론

최근 방송 및 디스플레이 기술이 발전함에 따라, 3차원 영상이 큰 화제를 모으며 차세대 영상 서비스로 각광받고 있다. 3차원 영상은 시청자들에게 입체감과 몰입감을 제공하여 사실감을 느낄 수 있다. 보통 이러한 3차원 영상이 제작될 때 여러 시점의 깊이 정보가 필요하다. 깊이 정보는 카메라에서 대상까지의 거리 값을 의미하며 여러 시점의 깊이 정보를 획득하기 위해서는 다시점 카메라 시스템을 이용해야 한다. 다시점 카메라 시스템에서 각 깊이 정보는 각각 다른 깊이 카메라에서 time-of-flight(ToF) 방법인 적외선 신호의 송수신 시간차를 측정하여 거리 값을 측정한다 [1, 2].

그러나 깊이 카메라의 특성상, 깊이 영상의 해상도는 매우 낮아 3차원 영상의 제작에 활용하기 위해서는 더 높은 해상도로 업샘플링 하는 과정이 필수적이다. 뿐만 아니라 적외선 신호의 주파수 문제 때문에 최대로 사용할 수 있는 개수가 정해져 있고, 깊이 카메라 하나마다 컴퓨터 한 대가 필요하다. 따라서 다시점의 깊이맵을 획득하기 위해서는 기존의 깊이 정보를 색상 카메라의 위치로 3차원 워핑을 하여 색상 카메라 개수만큼 깊이 정보를 획득한 다음 결합형 양방향 필터(joint bilateral filter, JBF)를 이용해 업샘플링을 수행하여 다시점 깊이맵을 생성하는 방법이 있다 [1, 2].

이 업샘플링 작업은 시간 복잡도가 크지만 그래픽스 프로세싱 유닛(graphics processing units, GPU)을 이용하여 빠르게 수행할 수 있다. GPU는 많은 작업을 동시에 실행시킬 수 있는 코어를 많이 가지고 있기 때문이다. GPU에서 작업을 수행하기 위해서는 범용 목적 GPU(general purpose computing on GPU, GPGPU) 기술이 필요하다. 그 중의 하나인 CUDA는 C언어 기반으로 작성되었으며, 이식성과 확장성이 커서 CUDA를 사용한 프로그램은 고성능 GPU로 대체하거나 여러 개의 GPU를 사용하여 프로그램의 속도를 이전보다 높일 수 있다 [1, 3]. 본 논문에서는 CUDA 다중 GPU를 사용한 다시점 깊이맵을 빠르게 생성하는 방법을 제안하고자 하며, [1]에서 제안된 방법보다 좀 더 효율적으로 다시점 깊이맵을 생성한다.

2. 다시점 깊이맵 생성

2.1 시스템 구성

다시점 깊이맵을 생성하기 위해서는 우선 다시점 카메라 시스템이 필요하다. 그림 1 은 본 실험을 위해 설치한 8개의 색상 카메라(Basler Pilot piA1900-32gc GigE)와 3개의 깊이 카메라(Mesa Imaging SR4000)로 구성된 다시점 카메라 시스템의 도식이다. 색상 카메라 간의 간격은 사람의 미간거리이며, 깊이 카메라는 색상 카메라 밑에 균일한 간격을 두고 설치했다. 객체는 카메라와 2~3m 거리를 두고 배경과 쉽게 구분하기 위해 블루스크린 스튜디오에서 촬영을 진행했다.

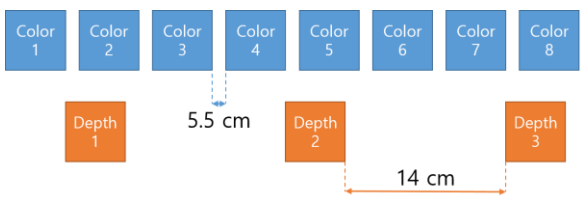

그림 1. 다시점 카메라 시스템의 도식

2.2 전처리 과정

다시점 깊이맵 시스템을 설치한 후에는 모든 카메라의 파라미터를 구하는 과정인 캘리브레이션을 진행해야 한다. 카메라 파라미터는 내부 파라미터(intrinsic parameter)와 외부 파라미터(extrinsic parameter)로 나눌 수 있다. 내부 파라미터는 초점거리(focal length)와 주점(principal point)으로 구성된 행렬하나, 외부 파라미터는 3차원 공간에서의 회전(rotation)값으로 구성된 행렬과 변형(translation)값으로 구성된 행렬로 이루어져있다. 카메라 파라미터는 체스 판 같은 특징점을 찾기 쉬운 패턴을 여러 방향과 각도에서 촬영하여 구하는 방법이 있다. 예측한 파라미터를 이용하여 다시점 카메라 시스템에서 각 카메라 간의 상관관계를 파악할 수 있으며, 다시점 깊이맵을 생성할 때에는 각 깊이 카메라에서 색상 카메라 위치로 3차원 워핑을 할 때와 색상 카메라 간 영상 정렬(image rectification)을 할 때 사용된다. 영상 정렬은 모든 영상을 y축의 공통되는 평면으로 맞추어 다시점 카메라 시스템에서 색상 카메라를 설치할 때 발생할 수 있는 약간의 틀어짐을 줄일 수 있다.

2.3 다시점 깊이맵 생성 과정

다시점 깊이맵 생성의 첫 과정은 깊이 정보 업샘플링 단계의 일부인 3차원 워핑이다. 전처리 과정에서 예측한 카메라 파라미터와 각 깊이 정보를 이용하여 색상 카메라 위치로 3차원 워핑을 하면 색상 카메라 개수만큼 깊이 정보를 얻을 수 있다. 식 (1)과 식 (2)는 3차원 워핑 식을 나타낸다. 식에서 I과 i은 각각 깊이 카메라와 색상 카메라의 영상을 나타내며, A, R, t는 각각 카메라의 내부 파라미터, 외부 파라미터의 회전 행렬과 변형 행렬, M_w과 m은 각각 3차원 영상좌표와 2차원 영상좌표를 나타낸다. 그림 2는 3차원 워핑 결과이다. 깊이 영상의 낮은 해상도 때문에 깊이 정보가 충분하지 않아 빈 공간이 생기는 것을 볼 수 있다.

다음 과정은 워핑 결과에서 빈 공간을 채워 올바른 깊이 정보를 만들기 위해 JBF를 수행하여 깊이맵을 완성한다. 식 (3)-(8)는 JBF 식을 나타낸다. $D(x,y)$와 $D_i(x,y)$는 이미지 좌표 내에서 각각 JBF 수행 결과와 결과 이전의 깊이 값을 나타내고, W는 JBF의 가중치 값이다. 여기서 JBF는 빈 공간을 채우는 것이 목적이기 때문에 깊이 값이 없는 픽셀은 가중치 값이 0이 되며, 그렇지 않은 경우에는 JBF 필터 크기 r값과 표준편차 σ값에 따라 색상 영상의 차이 값을 이용한 가중치와 가우시안 테이블의 가중치를 이용하여 최종 JBF의 가중치를 정하게 된다. 그림 3은 그림 2의 워핑결과에서 JBF를 수행한 결과다.

$$M_w = R_l^{-1} \cdot A_l^{-1} \cdot m_l - R_l^{-1} \cdot t_l \quad (1)$$

$$m_r = A_r \cdot R_r \cdot M_W + A_r \cdot t_r \quad (2)$$

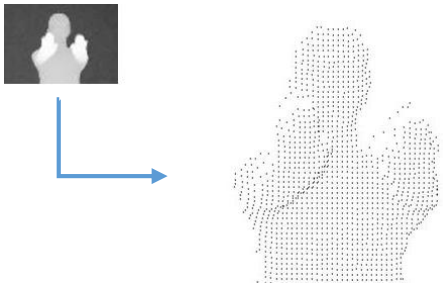

그림 2. 깊이 카메라 원본 이미지의 3차원 워핑 결과

$$D(x,y) = \frac{\sum_u \sum_v W(u,v) D_i(x,y)}{\sum_u \sum_v W(u,v)} \quad (3)$$

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

$$f(u,v) = \exp\left\{-\frac{|I(x,y)-I(u,v)|^2}{2\sigma_f^2}\right\} \quad (5)$$

$$g(u,v) = \exp\left\{-\frac{(x-u)^2+(y-v)^2}{2\sigma_g^2}\right\} \quad (6)$$

$$u = \{x-r, \cdots, x+r\} \quad (7)$$

$$v = \{y-r, \cdots, y+r\} \quad (8)$$

그림 3. JBF 수행 결과

3. 제안된 다시점 깊이맵 생성 방법

3.1 전처리 과정

제안된 방법은 여러 개의 GPU를 사용하여 다시점 깊이맵을 보다 빠르게 생성할 수 있지만 제약사항이 있다. 첫 번째는, 그래픽카드에는 고유의 메모리가 장착되어 있고 다른 GPU 간에는 그 메모리를 공유할 수 없다는 점이다. 따라서 다시점 카메라의 색상 카메라와 깊이 카메라 개수에 따라 다시점 깊이맵 생성에 사용할 GPU 개수를 설정하고 필요한 영상 데이터를 처리할 GPU마다 전송해야 한다. 두 번째는, 처리해야 할 데이터를 GPU 개수만큼 균등하게 분배하여 처리할 수 없다는 점이다. JBF에서 이웃 픽셀을 참조하게 하는데 다른 GPU의 메모리는 참조할 수 없기 때문이다. 그러므로 GPU는 최대 깊이 카메라 개수만큼 사용할 수 있고, 그 개수에 따라 GPU와 깊이 카메라가 어느 색상 카메라에 정합이 되는지 확인한 다음 분배 방법을 정해야 한다.

다음 전처리 과정은 깊이 영상의 업샘플링을 빠르게 하기 위해 3차원 워핑을 할 때 필요한 행렬과 JBF에서 사용하는 가우시안 테이블을 미리 계산하고 GPU 메모리에 전송해놓는다. 실제로 GPU에서 작업을 수행할 때 소요되는 시간의 대부분이 GPU 메모리 간 데이터를 전송하는 시간인데, GPU에 데이터를 미리 전송해놓을 수 있다면 그만큼 처리 시간에서 큰 효율을 볼 수 있다. 제안된 방법에서는 3차원 워핑에서 깊이 카메라의 파라미터 $R_l^{-1} \cdot A_l^{-1}$, $R_l^{-1} \cdot t_l$ 행렬, 색상 카메라의 $A_r \cdot R_r$, $A_r \cdot t_r$ 행렬과 JBF에서 필요한 가우시안 테이블을 나타내는 식 (6)-(8)을 미리 계산하고 전송해놓는다.

3.2 영상 불러오기

GPU의 메모리 접근 방식은 메모리 단위가 4바이트일 때 더 빠르다. 따라서 제안된 방법에서는 다시점 깊이맵 생성에 필요한 모든 영상을 불러온 후, 영상 픽셀의

자료형을 4바이트 float로 변환한다. 픽셀 자료형 변환 작업은 긴 시간을 필요로 하므로 다시점 깊이맵 생성을 시작하기 전에 변환 작업을 수행하고, 각 처리할 GPU 마다 구분하여 배열에 모아놓는다.

3.3 원형 큐

본 논문에서 제안된 방법은 최대한 빠르게 다시점 깊이맵을 생성하기 위해 GPU 메모리로 할당된 원형 큐를 사용한다. C++의 표준 템플릿 라이브러리(standard template library, STL)를 이용하여 구현한 원형 큐는 여러 버퍼를 두어 GPU 메모리 간 데이터를 전송할 때 생기는 대기시간을 없애고 다음 프레임의 다시점 깊이맵을 생성할 수 있게 한다. [1]에서 제안된 방법에서는 원형 큐의 크기가 고정되었지만, 본 논문에서 제안된 방법은 다시점 깊이맵 생성 성능을 최대로 올리기 위해 GPU 메모리 사용량(occupancy)을 최대한 활용하도록 결정된다 [1, 3].

원형 큐에서 현재 생성할 다시점 깊이맵 데이터는 현재 프레임 번호에 원형 큐 크기로 나머지 연산(modulo operation)한 인덱스에서 찾을 수 있고 다시점 깊이맵 생성 결과도 그 공간에 저장된다. 그림 4는 원형 큐의 예시를 보여준다. 그림의 원형 큐는 크기가 10이며, 현재 메모리 상태를 볼 때, 8번 프레임의 다시점 깊이맵을 생성하고 있고, 이전 7개 프레임의 다시점 깊이맵은 생성되었지만, 아직 5개 프레임의 결과는 GPU 메모리로부터 받지 못했거나 이미지 파일로 저장하고 있는 상태임을 알 수 있다. 또한 그림의 원형 큐 0번과 1번 인덱스와 같이 생성이 완료된 다시점 깊이맵을 이미지 파일로 저장하였다면 해당 원형 큐 공간은 다음에 재활용될 수 있다.

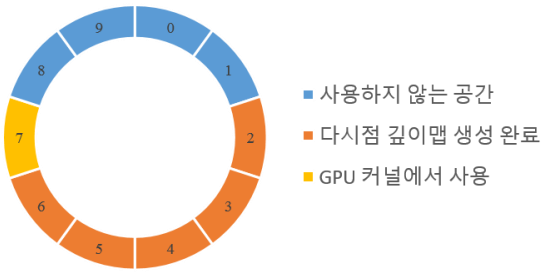

그림 4. 크기가 10인 원형 큐 메모리 상태

3.4 다시점 깊이맵 생성

다시점 깊이맵 생성은 3.2절 영상 불러오기 단계에서 영상 픽셀의 자료형이 4바이트 float로 변환된 영상을 모아놓은 배열을 GPU 메모리가 할당된 원형 큐로 전송하는 것으로 시작된다. 각 처리할 GPU 메모리로 한 프레임의 영상 데이터를 전송했으면 C++ 병렬 패턴 라이브러리(parallel pattern library, PPL)를 이용하여 여러 GPU에서 다시점 깊이맵을 생성하도록 동시에 GPU 커널 함수를 호출한다. 이 과정에서 PPL은 최적의 방법으로 병렬 알고리즘을 통한 동시성을 제공하며 하나의 GPU가 작업하는 동안 다른 GPU가 대기하는 현상을 줄일 수 있다 [4]. 한 프레임의 다시점 깊이맵 생성이 완료되면 원형 큐에 결과를 저장하고 즉시 다음 프레임의 다시점 깊이맵 생성을 시작한다.

3.5 결과 저장 및 메모리 해제

한 프레임의 다시점 깊이맵 생성이 끝나면 GPU 메모리로부터 메인 메모리로 결과를 받고 이미지 파일로 결과를 저장하게 하는데, 이 과정은 다음 프레임의 다시점 깊이맵 생성을 바로 시작하기 위해 CPU 스레드를 통해 진행된다. 이 스레드는 원형 큐를 만든 직후에 호출되며 다시점 깊이맵이 생성되는 동안 원형 큐를 계속 검사하며 다시점 깊이맵 생성이 끝난 결과를 메인 메모리로 받아 이미지 파일로 저장하여 원형 큐의 공간이 재활용될 수 있도록 한다. 모든 프레임의 다시점 깊이맵 생성이 끝났다면 이 스레드는 원형 큐 공간의 사용했던 모든 메모리를 해제하고 종료된다.

4. 실험결과

제안된 방법을 이용하여 다시점 깊이맵을 생성할 때 그림 1에서와 같이 8개의 색상 카메라와 3개의 깊이 카메라로 구성된 다시점 카메라 시스템을 사용했다. 3.1 절 첫 문단에서 설명했듯이, 다시점 깊이맵 생성에 최대로 사용할 수 있는 GPU 개수는 다시점 카메라 시스템에서의 깊이 카메라 개수만큼인데, 본 실험에서는 최대 3개의 그래픽카드를 이용하며, 모두 GeForce GTX TITAN 제품을 사용하였다. 실험에 따라 1개 또는 2개의 GPU를 사용해야 할 경우에는 CUDA에서 지원하는 함수를 이용하여 다른 GPU의 사용을 제한했다. 그림 5는 본 논문에서 제안된 다시점 깊이맵 생성 방법을 이용하여 생성한 8시점의 깊이맵 결과다.

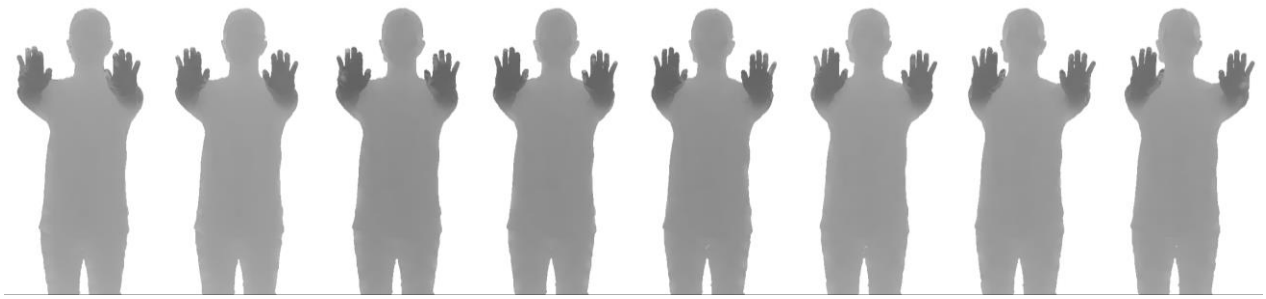

그림 5. 제안된 방법을 이용하여 생성한 8시점의 깊이맵 결과

표 1과 표 2는 각각 [1]에서 제안된 다시점 깊이맵 생성 방법과 본 논문에서 제안된 다시점 깊이맵 생성 방법을 이용하여 GPU 개수와 생성한 프레임 수에 따른 다시점 깊이맵 생성 속도를 나타낸다. 생성 속도를 측정할 때, 3.4 절에서 설명한 다시점 깊이맵 생성 과정만 포함되며, 색상 영상과 깊이 영상의 해상도는 각각 1280×720과 176×144이며, JBF 필터 크기는 15로 설정했다. 2개의 GPU를 사용할 때는 GPU 0번에서 3개의 시점, GPU 1번에서 5개의 시점의 깊이맵을 생성했고, 3개의 GPU를 사용할 때는 GPU 0번과 1번에서 3개의 시점, GPU 2번에서 2개의 시점의 깊이맵을 생성했다.

표 1. [1]에서 제안된 다시점 깊이맵 생성 속도

실험	*GPU 사용 개수*	*생성한 프레임*	*원형 큐 크기*	*Fps*
1-1	1	90	15	17.14
1-2	2	90	15	21.19
1-3	3	90	15	34.08

표 2. 본 논문에서 제안된 다시점 깊이맵 생성 속도

실험	*GPU 사용 개수*	*생성한 프레임*	*원형 큐 크기*	*Fps*
2-1	1	90	35	16.54
2-2	2	90	61	22.67
2-3	3	90	90	35.12
2-4	1	120	35	14.71
2-5	2	120	61	22.71
2-6	3	120	93	33.68

표 2의 실험결과를 보면, 본 논문에서 제안된 다시점 깊이맵 방법에서는 GPU 메모리를 최대한으로 활용하여 원형 큐 크기를 크게 잡을 수 있었고 그만큼 대기시간이 줄어드는 결과, [1]의 방법보다 초당 약 1 프레임의 다시점 깊이맵을 더 생성했다. 그러나 GPU를 1개만 사용할 경우에는 GPU 메모리 간 전송하는 데이터가 더 많아지므로 오버헤드가 많이 발생하게 되며 오히려 성능이 떨어진 것을 볼 수 있다. 그 외에는 사용하는 GPU 개수가 많을수록 입력 영상과 생성되는 결과가 각 GPU 메모리에 분배되므로 원형 큐 크기는 보다 크게 설정되며 다시점 깊이맵 생성 속도는 크게 증가하는 것을 볼 수 있다.

반면에, 실험 2-3 방법으로 다시점 깊이맵을 생성할 때, 각 GPU 메모리가 충분하여 원형 큐 크기가 크게 설정 되므로 원형 큐 공간이 재활용될 필요가 없었다. 따라서 동일한 조건으로 120 프레임의 다시점 깊이맵을 생성한 실험 2-6과 비교해볼 때, GPU 메모리 간 데이터를 전송할 때 발생하는 대기시간이 다시점 깊이맵 생성 속도에 큰 영향을 미치는 것을 알 수 있다.

5. 결론

본 논문에서는 다중 GPU를 사용하여 빠르게 다시점 깊이맵을 생성할 수 있는 방법을 제안했다. 다시점 깊이맵은 해상도가 낮은 깊이 영상을 색상 카메라 위치로 3차원 워핑을 하고 JBF를 수행하여 색상 영상의 해상도로 얻을 수 있었다. 실험결과로는 원형 큐와 PPL을 사용하여 다중 GPU의 제약사항을 해결하여 단일 GPU 환경보다 압도적으로 빠르게 다시점 깊이맵을 생성했다. 또한 원형 큐의 크기를 고정한 [1]의 방법과 달리 GPU 메모리를 최대한 활용하도록 원형 큐의 크기를 설정하여 [1]의 방법보다 1 프레임의 다시점 깊이맵을 생성하여, 3개의 GPU를 사용했을 때 초당 35 프레임의 8시점 깊이맵을 생성했다.

감사의 글

이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초사업임(No. 2011-0030079)

참고문헌

[1] E. Ko, Y. Song, and Y.S. Ho, “Real-time Multi-view Depth Generation Using CUDA Multi-GPU, ” International Conference on Embedded Systems and Intelligent Technology (ICESIT), pp. 114-116, Sep. 2014.

[2] Y. Song, D.W. Shin, E. Ko, and Y.S. Ho, “Real-time Depth Map Generation Using Hybrid Multi-view Cameras,” submitted to Asia-Pacific Signal and Information Processing Association (APSIPA), Dec. 2014.

[3] S Cook, CUDA programming: a developer’ s guide to parallel computing with GPUs. Morgan Kaufmann, Nov. 2012.

[4] Parallel Algorithms, http://msdn.microsoft.com/en-us/library/dd470426.aspx