CUDA를 이용한 다시점 거리영상 정합

최성인[1], 박순용[1], 김준[2], 박용운[2]
[1]경북대학교 컴퓨터공학과
ellim5th@vision.knu.ac.kr , sypark@knu.ac.kr
[2]국방과학연구소
wns3381@paran.com, woon5901@hanafos.com

Multi-view Range Image Registration using CUDA

Sung-In Choi[1], Soon-Yong Park[1], Jun Kim[2], Yong-Woon Park[2]
[1]Computer Engineering Dept. Kyungpook National University
[2]Agency for Defense Development

요 약

본 논문에서는 GPU의 성능을 이용하여 다시점 거리 영상을 실시간으로 정합하는 3차원 온라인 시스템을 제안한다. 제안한 시스템은 거리영상의 정교한 정합을 위해 IPP 알고리즘을 사용하였으며, 최신 GPU 프로그래밍 기법으로 각광받고 있는 CUDA를 이용하여 정합 알고리즘의 연산비용이 큰 부분에 해당하는 투영과 변환의 반복 부분을 수행하였다. 스테레오 기반 휴대용 거리센서에서 320x240 거리영상을 획득하여 정합 알고리즘을 수행한 결과, 초당 5장의 거리영상을 정합할 수 있었다. 제안한 온라인 시스템은 실시간 3차원 모델 복원 기술이 필요한 로봇위치 인식, 주행용 비전 기술, 문화재 원형 복원 등의 분야에서 활용될 수 있을 것이다.

1. 서 론

다시점 거리영상 정합(multi-view registration)은 서로 다른 카메라 좌표계에서 획득한 거리영상(range image)을 하나의 공통된 좌표계로 변환하는 기술이다. 이 기술은 거리센서(range sensor)를 이용하여 획득한 서로 다른 거리영상들로부터 3차원 모델을 복원해 내기 위한 중간처리 과정으로서 최종 모델의 정밀도에 영향을 주는 중요한 단계이다.

기존의 다시점 거리영상 정합에 대한 연구는 모든 거리영상을 확보한 오프라인 상태에서 정교한 정합 알고리즘 설계에 대한 내용이 주로 이루어졌다. 현재까지 알려진 방법은 크게 세 가지 분류로 나누어지며 관련 연구는 그 범위에서 크게 벗어나지 못하고 있다[1]. 하지만 최근 휴대가 가능한 레이저 센서 또는 스테레오 카메라가 일반화됨으로서 실시간으로 3차원 정합을 수행하는 온라인 시스템의 필요성이 대두됐음에도 불구하고 이에 대한 관련 연구는 거의 전무한 실정이다. 여러 가지 이유가 있겠지만 방대한 3차원 거리영상 정보를 실시간으로 처리하기 위한 시스템 자원의 부재를 주요 원인으로 생각해 볼 수 있다.

일반적으로 온라인 정합 시스템에서 실시간 정합을 수행할 때 알고리즘의 수행시간이 길어질수록 연속적으로 획득된 거리정보 사이의 움직임은 커지게 되며 결국 정합 성공률은 떨어지게 된다. 다시 말해 빠른 정합 속도가 정합 성공률에 중요한 요소가 된다. 하지만 현재까지 알려진 실시간 정합 시스템 구현 사례는 우리가 일반적으로 사용하는 PC기반의 CPU를 사용하고 있으며 이미 처리 속도의 한계에 다다른 상태이다.

최근 그래픽스 장치의 성능이 비약적으로 향상됨에 따라 이전에 범용 CPU로 처리하던 고 집적 연산 문제들을 해결하기 위한 새로운 대안으로 GPU(graphics processing unit)를 사용하는 사례가 늘고 있다. Studipta N. Sinha 등은 GPU를 이용한 KLT 특징 추적기과 SIFT[2]를 구현한 바 있으며, Alan Brunton 등은 스테레오 비전을 위한 신뢰 확산(Belief Propagation)[3] 알고리즘을 개발하였다. 이 외에 James Fung 등은 비디오 영상 기반에서 컴퓨터비전 알고리즘을 가속화 시킨 OpenVidia[4]를 소개하였다.

위와 같은 적용 사례들은 공통적으로 집약적 데이터를 빠른 시간으로 처리하기 위한 노력에서 비롯된 것임을 알 수 있다. 방대한 3차원 거리영상을 사용하는 온라인 정합 시스템 역시 GPU를 이용하면 실시간 처리에 관한 속도한계 문제를 해결할 수 있을 것이다.

본 논문에서 우리는 GPU를 이용하여 다시점 거리 영상을 획득함과 동시에 정합을 수행하는 실시간 3차원 온라인 정합 시스템을 제안한다. 특히 최신 GPU 프로그래밍 기법으로 각광받고 있는 CUDA를 사용함으로써 병렬처리에 대한 이해를 돕고 연산비용이 큰 정합 알고리즘의 수행시간을 획기적으로 개선하는 방법에 대해서 소개한다. 제안한 온라인 시스템은 실시간으로 3차원 모델 복원 기술이 필요한 로봇위치 인식, 휴대용 3차원 센서, 주행용 비전 기술, 문화재 원형 복원 등의 분야에서 활용될 수 있을 것이다.

서론에 이어 2절에서는 시스템 구현의 주요 핵심 도구인 CUDA의 아키텍처에 대해 소개하고 3절에서는 CUDA를 이용한 실시간 정합 기법에 대해 구체적으로 설명한다. 4절에서는 실험결과를 통해서 제안한 시스템의 성능을 검증하며 마지막으로 5절에서는 결론 및 향후 연구 과제를 기술한다.

2. 최신 GPGPU 기술 - CUDA

GPU는 본래 CPU의 그래픽스 작업으로 인해 생기는 병목 현상을 해결하기 위해 고안된 특수 목적 처리장치(special-purposed processor)이다. 하지만 CPU보다 높은 트랜지스터 집적도와, SIMD(single instruction multiple data) 아키텍처에 의한 탁월한 병렬처리 능력으로 인해 일찍부터 GPU를 범용 처리장치로써 사용하기 위한 연구가 활발하게 진행되어 왔다[5][6]. 일반적으로 GPGPU(general-purpose computation on GPUs)라 총칭되고 있는 이 기술은 특히 빠른 속도로 대량의 데이터를 처리해야하는 의학 영상처리나 비디오 인코딩, 컴퓨터비전 분야 등에서 알고리즘 속도 개선 효과를 위해 사용되고 있다.

CUDA(compute unified driver architecture)는 C언어를 이용하여 GPU에서 범용 컴퓨팅을 가능하게 해주는 최신 GPGPU 기술이다[7]. NVIDIA사에서 드라이버 및 소프트웨어 개발 툴킷을 직접 제작하여 배포하고 있으며 자사의 하드웨어인 Geforce 8~9 series, Quadro NVS 130M, Tesla에서 사용이 가능하다.

병렬 프로그래밍 방법에 있어서 CUDA는 전통적인 GPGPU 개념을 잘 따르고 있지만, 실제로 운용되는 시스템의 아키텍처는 그림 1에서 보이는 것과 같이 큰 차이를 보인다. 이전의 GPGPU는 Cg나 Renderman 같은 셰이딩 언어(shading language)와 확장 OpenGL의 조합으로 이루어졌으며 사용자는 그래픽스 파이프라인에 맞춰 정점 셰이더(vertex shader)와 화소 셰이더(pixel shader)를 적절하게 프로그래밍 함으로써 GPU의 성능을 간접적으로 사용하였다. 하지만 시스템에서 접근 가능한 메모리 모델이 텍스쳐(texture)로 제한되어 있고 출력 쓰기 주소가 래스터화기에 의해 고정되기 때문에 임의의 메모리 공간 영역에 대해 동시 읽기·쓰기가 불가능하다는 단점이 있다. 반면 CUDA는 그래픽스 하드웨어를 하나의 독립적인 플랫폼으로 간주하고 프로그래밍 할 수 있는 환경을 제공한다. 사용자는 더 이상 그래픽스 파이프라인에 대한 이해가 필요 없게 되었으며, PC환경과 유사한 메모리 모델로 인해 입출력에 대한 제약사항도 벗어나게 됐다. 단지 CUDA 아키텍처에 대한 이해와 C언어 코딩 능력만 있으면 누구든지 GPU를 이용한 병렬 컴퓨팅이 가능해 진 것이다.

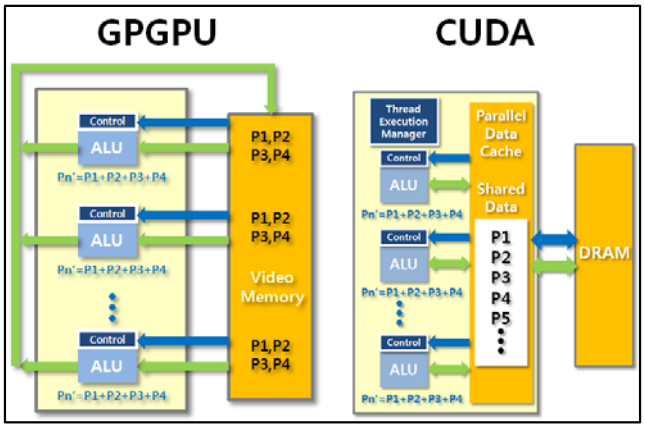

그림 1. GPGPU와 CUDA GPU 컴퓨팅 비교

3. GPU 기반 실시간 3차원 정합 시스템

본 논문에서 구현한 시스템은 투영과 변화의 반복을 이용하는 IPP(iterative projection point) 정합 기술을 사용한다. 본 절에서는 먼저 3차원 정합 기술에 대한 이해를 돕기 위해 IPP 알고리즘에 대해 설명하겠다. 이어서 우리 시스템에 적용된 CUDA 프로그래밍 기법에 대해 간략하게 설명하고 마지막으로 전체 시스템 구현 내용을 자세히 소개하도록 하겠다.

3.1 정교한 정합기술(Registration Refinement)

본 논문에서는 거리영상의 정교한 정합을 위해 점대면(point to plane) 방법과 점대투영점(point to projection) 방법의 장점을 결합한 IPP 기술을 사용한다[8][9]. 점대투영점 방식은 고속 탐색 알고리즘을 사용하는 대신 2.5차원의 거리 영상과 3차원 곡면사이의 투영관계를 이용한다. IPP 방식은 두 거리영상의 초기 위치가 큰 오차를 가지고 있을 때 점대투영점 방식보다 정확하게 거리영상을 정합한다. IPP 방법을 설명하면 그림 3과 같다. S곡면상의 점 P_0의 정합점 Q를 곡면 D에서 찾는 것이 정합의 목적이다. P_0를 D곡면의 2차원 영상으로 투영하여 좌표 p를 구한다. 좌표 p에 해당하는 D곡면상의 점 Q는 거리영상을 이용하여 구할 수 있다. 이 점을 다시 P_0의 법선에 투영하여 P_1을 구하고 위의 과정을 P_k가 수렴할 때까지 반복하여 Q를 구한다. 충분한 수의 대응점 집합 {P}와 {Q}를 구하고 두 점집합 사이의 정합 오차 ϵ를 최소화하는 변환 행렬을 수식 (1)을 SVD(singular value decomposition)를 사용해 계산한다.

$$\epsilon = \sum_i \| P_i - TQ_i \| \qquad (1)$$

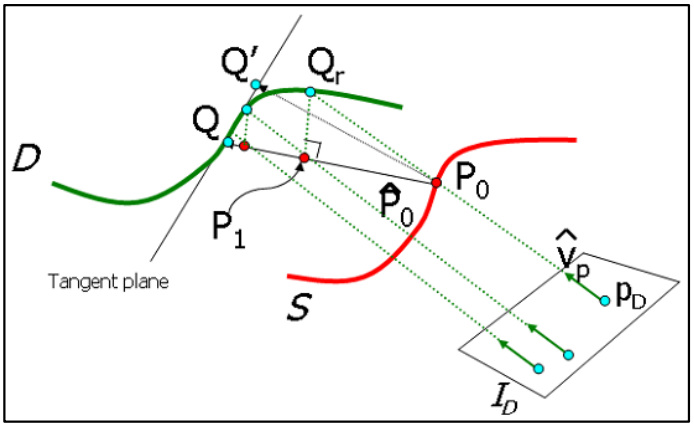

그림 2. IPP 기술을 이용한 정합점 탐색

3.2 CUDA 프로그래밍

CUDA는 cudaMalloc(), cudaMemcpy(), cudaFree()등의 함수를 통해 C언어와 유사한 방식으로 메모리 할당 및 복사, 해지 방법을 제공한다. 여기서 GPU쪽에 해당하는 메모리는 그래픽카드 상에 존재하는 DRAM을 말하며 화면출력(display)을 위해 사용하고 남은 공간을 사용하게 된다. 하지만 현재 CUDA는 메모리 페이징(memory paging) 기능이 지원되지 않기 때문에 항상 메모리를 할당 때 여유 공간을 고려하여야 한다. 효율적인 시스템 자원 사용을 위해, 우리는 전체 프레임 순서(sequence)에 해당하는 메모리를 사용하지 않고 프로그램 초기에 고정된 메모리양을 할당한 뒤 필요에 따라 데이터를 업로드하고 알고리즘을 수행하여 결과를 리드백하는 방법을 사용 한다. 메모리는 float3*320*240 크기의 입력용 배열 2개와 출력용 배열 1개를 사용하며, 추가적으로 투영 및 변환 매트릭스 입력을 위해 float*16크기의 배열 1개를 사용한다.

GPU에서 투영 및 변환 알고리즘은 그림 3과 같은 실행 모델(execution model)로 수행된다. CUDA에서 GPU는 매우 많은 수의 스레드들을 동시에 실행시킬 수 있는 병렬 처리 장치로 간주된다. 이 스레드는 우리가 일반적으로 운영체제에서 볼 수 있는 스레드와 비슷한 개념이며, GPU의 다중 처리장치(multi-processor)는 스레드별로 커널(kernel)이라고 부르는 프로그램을 수행시킨다. 그리고 스레드들은 하나의 블록(block)으로 묶여서 동시에 수행된다. 커널은 CUDA 프로그래밍의 실체적인 핵심이 되며.

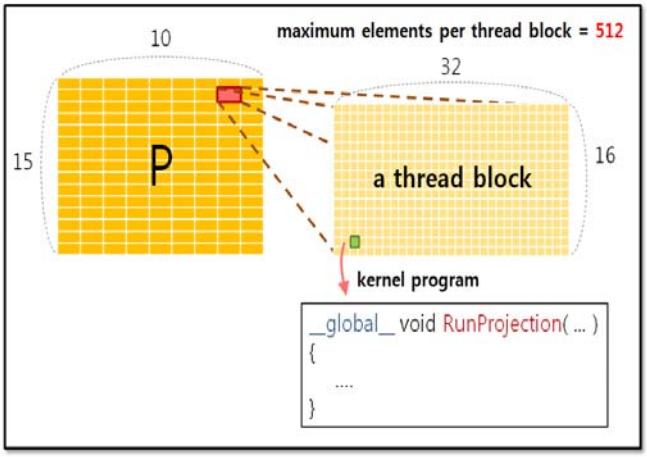

그림 3. 스레드 실행 모델

사용자에 의해 작성된다. 우리가 구현한 CUDA 프로그램 역시 투영과 변형 커널로 구성되어 있다.

CUDA 프로그램을 수행시킬 때 사용자는 한 개의 스레드 블럭 크기와 이 블럭들로 이루어진 격자(grid)를 지정하여야 한다. 격자와 스레드 블럭의 크기를 정하는 것은 사용자의 몫이다. 다만 첫째, 효과적인 병렬처리 성능 위해서 배열과 스레드가 1대 1로 사상(mapping)되어 실행되는 것을 보장해야 하며 둘째, 스레간의 통신은 스레드 블럭 내에서만 가능하다는 사실을 염두하고 커널 코딩에도 유의해야 한다. 우리는 320x240 영상에 대해 10*15 격자와 32*16 스레드 블럭을 이용하여 총 76800(10*32*15*16 = 320*240)개의 스레드를 사용하였다. 이 수치는 한 블록당

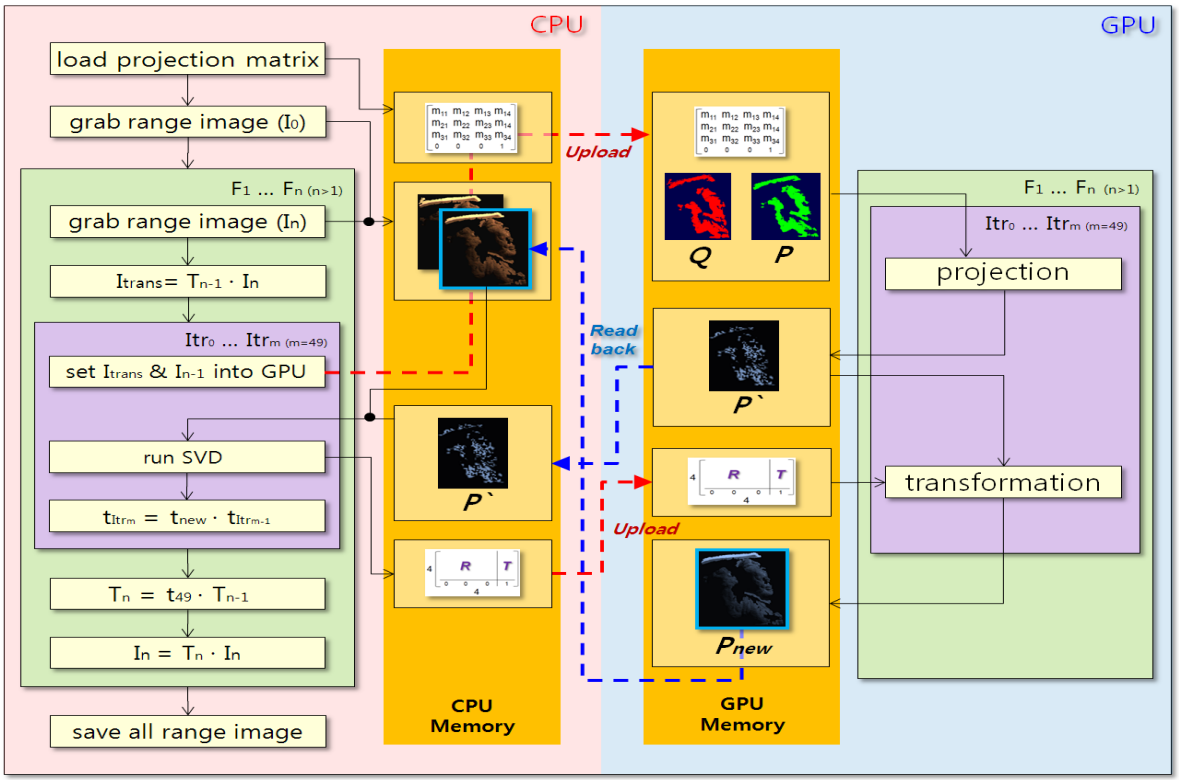

그림 4. 시스템 구성

512개의 스레드들이 묶일 수 있도록 정한 수치이며, 본 시스템에서는 한 배열 내에서는 인접 스레드끼리 참조가 필요 없기 때문에 격자 크기를 정하는데 있어서 스레드간의 통신은 고려되지 않았다. 참고로 CUDA는 G80 GPU를 기준으로 한 개의 스레드 블럭당 최대 512개의 스레드를 허용한다.

3.3 CUDA를 이용한 정합

GPU를 이용한 3차원 정합은 알고리즘의 핵심이자 연산 비용이 큰 부분에 해당하는 투영과 변환의 반복 부분을 그래픽스 장치에 올려 수행함으로써 전체적인 수행 시간을 향상시키는데 주요 목적이 있다. 그림 4는 GPU기반 실시간 3차원 정합 시스템의 전체 구성을 보여준다.

제안한 시스템은 현재 프레임을 기준으로 이전 프레임과 한 쌍이 되어 정합을 수행한다. 그래서 첫 프레임을 제외한 두 번째 프레임부터 데이터 획득 직후 정합을 시도하게 되며, 한 번의 정합 수행동안에는 Itr번의 투영과 변환이 반복된다. 여기서 Itr은 사용자에 의해 임의로 정해질 수 있는 변수이며 본 논문에서는 50으로 정해놓았다. 참고로 Itr의 수치가 클수록 좀 더 정밀한 정합을 시도하지만 전체 시스템의 수행 속도는 떨어지게 된다.

시스템의 전체 흐름은 다음과 같다. 먼저 현재 프레임의 거리영상 I_n이 획득되면 전처리 과정으로 I_n에 이전 단계의 정합 과정에서 만들어진 변환 매트릭스 T_{n-1}을 곱하여 I_{trans}를 생성한다. 이 과정은 새롭게 획득한 3차원 모델을 이전 프레임의 모델 위치에 개략적으로 이동시키는 것이며 궁극적으로 다시점에서 획득한 거리영상의 오차를 줄이는 역할을 한다. 전처리 과정이 끝나면 I_{trans}와 I_{n-1}은 각각 그림 2에서의 정합점 집합 P와 Q로 새롭게 정의되며 GPU 메모리에 업로드 된다.

P와 Q가 메모리에 업로드 되고 나면 다음 과정인 투영 단계로 넘어간다. 그림 5는 GPU에서 수행되는 투영 알고리즘의 자료구조를 보이고 있다. 투영단계는 P에서 Q로 투영되는 점의 집합을 구해내는 과정이며 다음 세 단계를 통해 이루어진다.

1) P의 [i, j] (i<320 j<240) 인덱스에 해당하는 (X_p, Y_p, Z_p)에 T_{n-1}^{-1}을 곱한 뒤 투영 매트릭스 M_{proj}를 한 번 더 곱하여 (u, v, w)를 구한다.
2) (u, v, w)를 w로 나누고, 그 결과를 반올림 하여 [u′, v′] 만든다.
3) Q의 [u′, v′] 인덱스에 해당하는 (X_q, Y_q, Z_q)를 P′의 [i, j] 인덱스에 대입한다.

투영단계를 거치고 나면 최종적으로 Q에 대한 P의 투영 결과가 저장된 P′가 생성된다. 이 P′는 CPU 메모리에 리드백 되며 변환행렬을 구하기 위한 입력 값으로 사용된다.

변환행렬은 P와 P‘를 이용하여 다음과 같은 과정으로 구해진다.

1) 거리영상 P와 P′를 같은 인덱스로 선형 샘플링 하여 약 300~500개의 3차원 점군을 추출한다. 잘못된 점에 대한 샘플링을 방지하기 위해 3차원 점의 xyz 값이 모두 0 이거나, 추출된 P와 P′ 점 사이의 벡터 크기가 10 이상인 경우는 샘플링에서 제외시킨다. P′의 xyz값은 그림 6에서 보이는 것과 같이 P′의 탄젠트 면에 P를 투영하여 찾아진 점을 사용한다.
2) 샘플링 된 두 거리영상의 중점(centroid) C_p와 $C_{p'}$를 계산한다.
3) P와 P′의 각 점군들과 중점의 차를 구하고 H를 구성한다.
4) H의 SVD를 수행한다.
5) U와 V성분을 이용해서 변환행렬을 구한다.

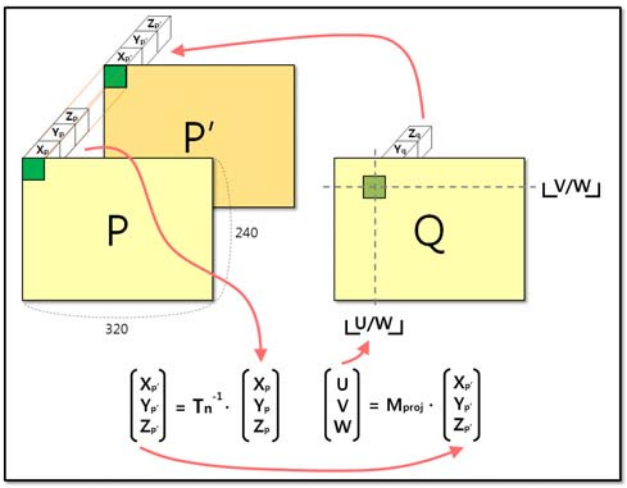

그림 5. 투영

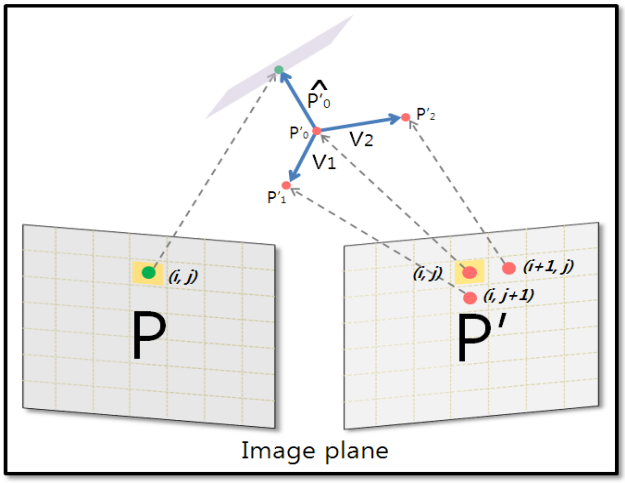

그림 6. 탄젠트 평면을 이용한 P′ 샘플링

변환행렬이 구해지면 t_n에 누적한다. 그리고 GPU 메모리에 이 변환행렬을 업로드한 뒤, 거리영상 P의 변환을

수행하여 P_{new}를 만들어 낸다. 생성된 P_{new}는 CPU 메모리에 리드백되고 다음 순서의 반복에 새로운 P의 입력 값으로 사용된다.

Itr번의 투영과 변환의 반복이 끝나면 누적된 t_n은 이전 프레임의 거리영상에 현재 프레임의 거리영상을 정합하기 위한 최종 변환행렬을 가진다. 이 최종 변환행렬을 T_n에 누적시키고 현재 프레임의 거리영상 I_n과 곱하면 한 개의 프레임에 대한 정합 과정이 끝나게 된다.

4. 실험 및 결과

실험은 사용자가 휴대 가능한 거리센서를 직접 움직여가며 문화재와 일반 환경에 대한 3차원 복원 작업을 수행한다는 시나리오 가정 하에 실시되었다. 거리센서는 캐나다 PointGrey사에서 개발한 BumbleBee 스테레오 카메라를 사용하였으며 320x240 해상도의 거리 영상을 획득하였다. 그리고 정합 시스템의 CPU와 그래픽스 장치는 각각 인텔 코어2 듀오 E6750과 Geforce 8800GTS가 사용되었다.

그림 7-(a)는 베토벤 석고상으로부터 약 0.5m 떨어진 위치에서 40장의 연속된 거리영상을 획득하여 정합을 수행한 결과이다. 석고상의 외곽에 해당하는 부분에서 조금씩 노이즈가 보이지만 전체적으로 성공적인 복원 형상을 보이고 있다.

그림 7-(b)와 7-(c)는 일반 환경에 대한 3차원 정합 실험 결과를 보여준다. 피사체와 카메라의 거리는 약 2m를 유지하면서 좌에서 우방향으로 각각 50장과 40장의 거리 영상을 획득하여 정합을 수행하였다. 최종 정합된 영상을 보면 베토벤 석고상의 정합 결과보다 좀 더 많은 노이즈가 섞여 있는 것을 확인할 수 있다. 특히 7-(b)의 왼편에 보이는 문서 수납장과 테이블, 그리고 7-(c)의 오른쪽 모니터 화면에서 이러한 현상이 두드러져 보이는데 이는 정합 오차로 인한 오류라기보다 스테레오 기반 거리 센서의 특성 및 기능상의 한계로 인한 잡음으로 판단된다. 향후 데이터 획득 직후 거리 영상의 잡음을 제거하는 정제 필터를 추가로 개발하거나 레이저 센서 혹은 프로젝터를 사용한 능동 센서를 사용하면 개선될 수 있는 문제라고 생각된다.

표 1은 본 논문에서 제안한 GPU 기반 시스템과 이전 연구에서 제안한 CPU 기반 시스템[10]과의 비교 성능을 보이고 있다. 정합에 사용된 영상의 프레임 수와 포인트의 수에 따라 조금씩 차이를 보이지만 CPU에서 수행한 결과와 GPU에서 수행한 결과가 한 프레임 평균 각각 0.8~1.1초, 0.2~0.22초가 소요됨으로서, 대략 4배 정도의 속도향상이 있는 것을 확인할 수 있었다.

(a)

(b)

(c)

그림 7. (a) 베토벤상, (b) 일반 환경1, (c) 일반 환경2

5. 결론

본 논문은 실시간으로 주변 환경 및 물체를 3차원 모델링 하기 위하여 CUDA를 이용한 다시점 거리영상 정합 방법을 제안하였다. 거리 영상의 정교한 정합을 위해 점대면 방법과 점대투영점 방법의 장점을 결합한 IPP 기법을 사용하였으며, 알고리즘의 핵심이자 연산비용이 큰 부분에 해당하는 투영과 변환의 반복 부분을 그래픽스 장치에

표 1. 정합 속도 비교

실험대상	전체 프레임	전체 포인트	평균정합속도 (sec/frame)	
			GPU	CPU
베토벤상	40	1,421,143	0.203	0.811
일반환경1	50	2,238,481	0.219	1.117
일반환경2	40	1,726,171	0.211	0.914

올려 수행함으로써 시스템의 전체 수행 속도를 획기적으로 개선시켰다.

그러나 우리가 제안한 시스템은 아직까지 완전한 실시간이라고 보기에는 다소 무리가 있는 초당 4~5프레임의 정합 수행 능력을 보이고 있다. 앞서 서론에서 설명한 것과 같이 빠른 처리 시간은 정합 성공률과도 직결되기 때문에 향후 본 시스템에서 초당 10~15 프레임 이상의 속도를 내기 위한 연구를 진행할 계획이다. 이를 위하여 현재 CPU 단에서 수행되고 있는 SVD를 GPU에서 실행되도록 개선할 예정이다. SVD가 GPU에서 실행되면 CPU 메모리와 GPU 메모리 사이에서 일어나는 업로드와 리드백 횟수를 줄일 수 있기 때문에 획기적인 속도향상 효과가 있을 것으로 기대한다.

감사의 글

이 연구는 국방과학연구소의 지원을 받아 수행되었음.

6. 참고문헌

[1] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," Proc. 3D Digital Imaging and Modeling, 145-152, 2001.

[2] Sudipta N Sinha, Jan-Michael Frahm, Marc Pollefeys and Yakup Genc, "GPU-Based Video Feature Tracking and Matching", EDGE 2006, workshop on Edge Computing Using New Commodity Architectures, Chapel Hill, May 2006.

[3] Qingxiong Yang, Liang Wang, Ruigang Yang, Shengnan Wang, Miao Liao and David Nister, "Real-time Global Stereo Matching Using Hierarchical Belief Propagation," BMVC06

[4] James Fung, Steve Mann, Chris Aimone, "OpenVIDIA: Parallel GPU Computer Vision", Proceedings of the ACM Multimedia 2005 , Singapore, Nov. 6-11, 2005, pages 849-852

[5] Physically-Based Visual Simulation on Graphics Hardware. Mark J. Harris, Greg Coombe, Thorsten Scheuermann, and Anselmo Lastra. Proc. 2002 SIGGRAPH / Eurographics Workshop on Graphics Hardware 2002.

[6] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüuger, Aaron E. Lefohn, and Timothy J. Purcell. "A Survey of General-Purpose Computation on Graphics Hardware." In Eurographics 2005, State of the Art Reports, August 2005, pp. 21-51.

[7] NVIDIA CUDA Programming Guide v1.1, 29 Nov. 2007 <http://www.nvidia.com/object/cuda_develop.html>

[8] Soon-Yong Park and Murali Subbarao, "An Accurate and Fast Point-to-Plane Registration Technique," Pattern Recognition Letter, 24 (16), pp. 2967-2976, Dec 2003

[9] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," Proc. 3D Digital Imaging and Modeling, 145-152, 2001.

[10] 백재원, 박순용, "3차원 기하정보 및 특징점 추적을 이용한 다시점 거리영상의 온라인 정합", HCI2007 학술대회, 2007