모바일 GPU에서의 병렬영상처리 라이브러리 개발

이종환 O, 강승헌, 이만희, 이성철, 김학일, 박인규
인하대학교 정보통신공학과
jhlee@vision.inha.ac.kr O, {ha00ha2,maninara}@gmail.com,
szli@vision.inha.ac.kr, {hikim,pik}@inha.ac.kr

Development of Parallel Image processing Library on Mobile GPU

Jonghwan Lee, Seunheon Kang, Manhee Lee, Shengzhe Li, Hakil Kim, Inkyu Park
School of Information & Communication Engineering, Inha University

요 약

본 논문은 스마트폰의 모바일 GPU를 이용하여 카메라로부터 실시간으로 입력되는 영상에 대하여 실시간으로 영상처리를 할 수 있는 라이브러리 개발에 대하여 다룬다. 본 라이브러리는 모바일 GPU에서 병렬 영상처리를 수행하고 매 프레임의 입력 영상을 텍스처로 저장하여 처리된 결과가 저장된 프레임버퍼를 통해 화면에 바로 출력되는 프레임워크를 기반으로 한다. 또한 국제표준으로 모바일 GPU가 지원하는 OpenGL ES 2.0 기반의 Shading Language를 이용하였다. 영상처리의 고속화 및 메인 메모리와 GPU 메모리 사이의 자료 전송의 효율화를 이뤄낼 수 있었고, 현재 CPU 기반의 영상처리 라이브러리인 OpenCV의 대표적인 40여개의 내장 함수들과의 비교를 통하여 모바일 CPU 대비 모바일 GPU를 이용한 영상처리 속도가 평균 3배 이상 가속되었다.

1. 서 론

최근 스마트폰의 대중적인 보급으로 사용자들의 모바일에 대한 지식 수준이 향상되었고 모바일 기술의 관심도가 높아졌다. 항상 휴대가 가능한 스마트폰 카메라를 이용하여 개인의 일상을 기록하고 SNS(Social Network Service)를 이용하여 그 기록을 공유하는 사용패턴이 형성되면서 모바일 카메라의 이용이 빈번해졌고 자연스럽게 고해상도의 카메라 및 디스플레이 장치에 대한 관심도가 증가하였다. 이렇게 모바일을 이용한 영상 취득과 그 사용이 증가함에 따라 기존의 단순 영상 취득하는 하여 기록하는 패턴에서 동영상이나 이 영상에 다양한 효과를 주거나 특정 목적에 의한 처리를 시도하기에 이르렀다.

이러한 관심 속에서 모바일 기술은 비약적인 발전을 이루고 있다. 그 중에서도 모바일의 두뇌라 할 수 있는 AP (Application Processor)의 발전으로 모바일의 효용성이 커졌고 많은 계산 자원이 필요한 영상처리 및 컴퓨터 비전 기술까지 사용할 수 있기에 이르렀다. 하지만 모바일에 함께 장착되는 디스플레이 장치 및 카메라, 다양한 센서들이 집약된 MEMS (Micro Electro Mechanical System) 그리고 통신모듈이 함께 발전하며 복잡도와 데이터 량이 기하급수적으로 증가하고 있다. 이에 더불어 이와 같이 디스플레이와 카메라가 고해상도의 영상을 다루기 때문에 고해상도 영상을 실시간으로 처리하기에는 모바일 CPU만으로는 부족한 상황에서 모바일 GPU를 GPGPU(General Purpose Graphics Processing Unit)로써 영상처리를 병렬화, 고속화하여 실시간 영상처리가 가능하도록 하는 연구가 이루어지고 있다.

본 논문에서는 이러한 연구의 일종으로 모바일 GPU를 이용하여 모바일에 장착된 카메라로부터 실시간으로 입력되는 영상이나 동영상 시퀀스에 대하여 실시간으로 영상처리를 할 수 있는 라이브러리 개발에 대하여 다룬다. 라이브러리 개발에는 먼저 카메라와 동영상 시퀀스를 이용한 실시간 영상처리 프레임워크를 구축에 대한 것과 복잡한 영상처리 함수들을 OpenCV의 대표 함수들을 벤치마크 하여 OpenGL ES 2.0[1] 기반의 Shading Language를 이용하여 구현하고 이를 각각 비교한 결과가 포함된다.

2. 실시간 영상처리 프레임워크

그림 1은 라이브러리를 동작하기 위한 프레임워크의 기본 구조를 도시화하고 있다. 입력 영상은 카메라로부터 들어오는 프레임과 저장된 동영상 프레임 이렇게 두 가지 종류로 구분된다. 모바일 카메라로부터의 입력영상은 모듈을 통하여 바로 텍스처로 저장되고 이를 이용하여 처리를 하게 된다. 마찬가지로 동영상의 재생 프레임의 처리도 각 모바일의 미디어플레이어를 이용하여 재생함과 동시에 매 프레임이 텍스처로 저장되고 이를 이용하게 된다. 준비된 입력 텍스처로 OpenGL ES 2.0 Shading Language를 이용하여 GPU에서 병렬영상처리를

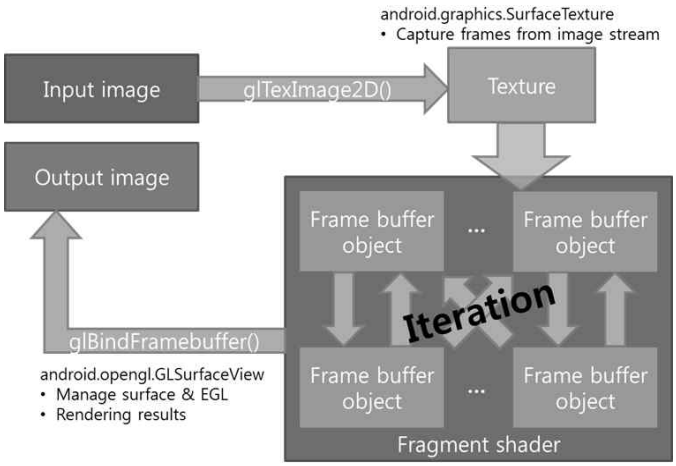

그림1. OpenGL ES 기반 영상처리 프레임워크

수행하게 되고 그 결과가 저장된 텍스처를 그대로 디스플레이 장치에 출력하는 구조를 통하여 영상처리의 고속화 및 프로세서간의 자료 이동을 최소와 할 수 있다.

3. OpenGL ES 2.0 Shading Language를 이용한 라이브러리 구현

OpenGL ES는 3D 그래픽 API를 제공하는 오픈 라이브러리인 OpenGL의 모바일 버전으로 OpenGL이 워크스테이션이나 고성능 PC와 같은 환경에서 3D 이미지를 렌더링 하는 것을 목표로 하는 것과는 달리 적은 용량의 메모리와 낮은 속도의 CPU를 갖춘 임베디드 환경에 최적화 되어있다. 이전의 그래픽 파이프라인은 프로그래머에 의해 임의로 변경할 수 없었지만 GLSL이 출시됨에 따라 프로그래밍이 가능한 GPU를 사용할 수 있게 되었다. GLSL을 이용하면 그래픽 파이프라인 중 정점 쉐이더(vertex shader)와 프레그먼트 쉐이더(fragment shader)를 임의대로 수정할 수 있는데 이를 이용하면 한번 연산된 결과가 바로 화면으로 출력되어 버린다.

보통의 영상처리 알고리즘은 여러 단계를 거쳐 알고리즘을 완성하는 데, 이를 해결하기 위하여 그림 1의 프레그먼트 쉐이더 부분과 같이 FBO(Frame Buffer Object)를 사용하게 된다. FBO란 GLSL을 통하여 계산된 결과를 프레임 버퍼에 저장하는 것이 아니라 GPU의 텍스처 메모리에 기존의 데이터 크기와 동일한 영역을 만들어 저장하고 다음 알고리즘을 적용하는 단계에서 텍스처 유닛이 직접 사용할 수 있게 하여 데이터를 복사하는 데 시간을 줄일 수 있는 방법이다.

다음의 방식으로 표 1 에서와 같이 7개의 분류를 갖는 40여개의 영상처리 함수를 구현하였고 2차적으로 OpenGL ES 2.0 GLSL 파이프라인에 맞도록 여러 최적화 기법 중 아래의 네 가지 최적화 기법을 적용함으로써 효율적인 GPU기반의 병렬영상처리 라이브러리를 완성할 수 있었다.

Function Class	Function Name	Function Class	Function Name
Basic Image Filtering	bilateralFilter Blur boxFilter buildPyramid dilate erode filter2D GaussianBlur Laplasian medianBlur pyrMeanShiftFiltering sobel	Histogram	calcHist calcBackProject compareHist equalizeHist HoGDescriptor
		Feature Detection	Canny cornerHarris cornerSubPix FindCircle FindContours goodFeaturesToTrack HoughLines HoughLinesP
Image Geometry Transform	remap resize rotate warpAffine warpPerspective	Robust Feature Detection	SIFT SURF
		Object Detection	matchTemplate CascadeClassifier
Image Pixel Trasnform	adaptiveThreshold cvtColor distanceTransform integral threshold watershed	Object Tracking	OpticalFlow

표 1. OpenGL ES 기반 라이브러리 목록

- Floating Point Precision Control
- LoopUnrolling
- Branching
- Load Sharing Between Vertex and Fragment Shaders

마지막으로 라이브러리의 결과를 모바일 CPU 기반 영상처리를 라이브러리인 OpenCV와 비교함으로써 속도 개선 및 효율성을 확인 및 검증하였다.

4. 실험 결과

개발된 프레임워크의 성능을 평가하기 위하여 논문에서는 Android 4.2.2 기반의 Nexus 10 태블릿을 이용하여 실험을 진행하였다. 이 실험기기는 1.7GHz 듀얼코어 CPU와 2GB의 메인 메모리, ARM의 Mali-T604 GPU를 탑재하고 있으며 500Mp의 카메라 및 2560x1600 해상도의 디스플레이가 내장되어 있다.

3절에서의 설명과 같이 입력되는 매 프레임은 텍스처 형태로 지정되고 OpenGL ES의 Shading Language를 이용하여 처리된다. 실험을 위한 카메라 및 동영상 재생은 Java 기반의 Android SDK를 이용하였고 OpenGL ES 2.0 설정 및 실제 영상처리는 C언어기반의 Android NDK[2]를 이용하여 개발하였다. 성능평가를 위하여 몇가지 Android용 OpenCV 라이브러리의 함수를 통하여 CPU에서의 영상처리를 살펴보고 같은 알고리즘으로 논문에서 구현된 라이브러리의 함수를 통하여 비교하였다.

Android[3]에서 미디어플레이어는 동영상을 재생하며 바로 텍스처로 저장하는 내부 파이프라인을 이용하기 때문에 이를 CPU 메모리로 다시 가져오며 자료이동 시간이 존재하게 되고 계산자원을 낭비하게 되므로

분류	알고리즘	OpenCV(ms)	OpenGL(ms)	비교
Image Filtering	bilateralFilter	102	47.5	×2.1
	boxFilter	45	17.7	×2.5
	buildPyramid	25	5.2	×4.8
	dilate	17	11.6	×1.5
	filter2D	37	17.7	×2.1
	GaussianBlur	90	27.2	×3.3
	Laplacian	68	14.2	×4.8
	medianBlur	203	16.8	×12.1
	pyrMeanShiftFiltering	7186	142.1	×50.6
	sobel	65	28.5	×2.3
Geometry Transform	remap	34	10.4	×3.3
	resize	34	9.6	×3.5
	rotate	34	9.2	×3.7
	warpAffine	34	12.5	×2.7
	warpPerspective	51	10.8	×4.7
Pixel Transform	cvtColor	17	10.9	×1.6
	threshold	8	9.3	×0.9
Histogram	calcHist	8	328.7	×0.0
	calcBackProject	10	9.9	×1.0
	equalizeHist	19	10.5	×1.8
Feature Detection	Canny	118	34.6	×3.4
	cornerHarris	127	51.1	×2.5
	FindContours	14	11.4	×1.2
	goodFeaturesToTrack	172	207.1	×0.8
	HoughLines	134	97.5	×1.4
Robust Feature Detection	SIFT	732	227.6	×3.2
	SURF	687	256	×2.7

표 2. OpenCV와 본 논문의 라이브러리의 비교

그림 2. 본 논문의 라이브러리 함수 수행 결과 영상 (위쪽부터 bilateral filtering와 SIFT 특징추출)

FFmpeg 오픈 라이브러리[4]를 이용하여 프레임을 바로 메인 메모리에서 이용할 수 있도록 해 주었다. 영상은 1280x720 해상도의 동영상을 이용하여 수행하였다.

수행 결과는 OpenCV를 이용한 영상처리 결과를 이용하여 검증하였고 그림 2 에서와 같이 라이브러리가 잘 동작함을 확인하였다. 각 알고리즘들의 복잡도 및 병렬화의 적합성에 따라 2~12배까지의 처리속도 개선을 확인할 수 있음을 표 2 에서 볼수 있다. 하지만 OpenGL ES 2.0 Shading Language의 특성으로 복잡하거나 부적합한 알고리즘에 대해서는 결과 획득을 위하여 우회하는 방법으로 구현할 수 밖에 없었다. 예를 들어 히스토그램을 계산하는 calcHist의 경우에는 전체 픽셀에 대해 각 픽셀 값의 빈도수를 누적시켜야 하므로 병렬화 기법들을 사용하고도 상당히 느린 처리결과를 보였고 이외에도 몇몇 함수들에 대하여 우회방법으로 구현된 함수에 대해서는 느린 결과를 얻을 수 밖에 없었다.

5. 결론

본 논문에서는 스마트폰에 장착된 카메라로부터 입력되는 프리뷰 영상 및 동영상에 대하여 GPU를 이용하여 실시간으로 영상처리를 수행 할 수 있는 프레임워크를 기반한 라이브러리를 소개하였고 모바일의 CPU와 GPU 각각에서의 영상처리 속도를 비교해 볼 수 있었다. 그 결과 OpenGL ES 2.0 Shading Language의 프레그먼트 쉐이더에서 출력 픽셀 정보 공간 이외의 다른 정보 저장 공간이 없다는 특성 때문에 복잡하거나 비적합한 알고리즘에 대해서는 오히려 느려질 수 밖에 없음을 확인할 수 있었지만 이는 아직은 완전하지 않고 최적화가 되지 않았지만 최근 ARM의 Mali에서의 지원이 시작된 OpenCL 1.1[5]을 이용한다면 해결할 수 있는 문제이다. 따라서 대부분의 함수에 대하여 CPU를 이용한 영상처리보다 GPU를 이용했을 때의 결과가 2~12배의 속도개선을 이룰 수 있음을 확인 할 수 있었다.

감사의 글

본 논문은 지식경제부 산업융합원천기술개발사업으로 지원된 연구결과입니다. [10041664, 멀티 Shader GPU 통합형 멀티 코어 퓨전 프로세서 원천 기술 개발]

참고문헌

[1] http://www.khronos.org/opengles/2_X/
[2] http://developer.android.com/tools/sdk/ndk/
[3] http://www.android.com
[4] http://ffmpeg.mplayerhq.hu/
[5] http://malideveloper.arm.com/develop-for-mali/sdks/mali-opencl-sdk/