논문 2007-44SP-6-2

GPU를 이용한 DWT 및 JPEG2000의 고속 연산

(Fast Computation of DWT and JPEG2000 using GPU)

이 만 희*, 박 인 규**, 원 석 진***, 조 성 대***

(Man Hee Lee, In Kyu Park, Seok Jin Won, and Sungdae Cho)

요 약

본 논문에서는 GPU (Graphics Processing Unit)를 이용하여 JPEG2000 정지영상 압축 알고리즘의 DWT (Discrete Wavelet Transform) 연산을 고속으로 수행하기 위한 효율적인 구조와 방법을 제안한다. DWT 연산은 JPEG2000에서 EBCOT (embedded block coding with optimized truncation)과 더불어 많은 계산 량을 소모하는 부분이기 때문에, 본 논문에서는 DWT 알고리즘을 GPU의 화소 쉐이더에서 고속으로 수행하기 위하여 Render-To-Texture (RTT)를 활용한 구조를 설계하였다. 실제 구현을 통해 비슷한 등급의 CPU에서의 처리에 비해 DWT 자체는 10배 이상의 수행 속도의 향상을, 기존의 JPEG2000 참조 소프트웨어인 JasPer의 DWT를 대치하였을 때 2~16배의 수행 속도의 향상을 보였으며 해상도가 증가할수록 향상 폭이 크다. 본 논문에서 제시된 프레임 버퍼 객체(Frame Buffer Object)를 이용한 render-to-texture 수행 구조는 GPU 기반 영상처리의 기본 틀을 제공하며, 이를 응용하여 일반적인 영상처리와 컴퓨터 비전 처리를 GPU 상에서 고속 수행할 수 있다.

Abstract

In this paper, we propose an efficient method for processing DWT (Discrete Wavelet Transform) on GPU (Graphics Processing Unit). Since the DWT and EBCOT (embedded block coding with optimized truncation) are the most complicated submodules in JPEG2000, we design a high-performance processing framework for performing DWT using the fragment shader of GPU based on the render-to-texture (RTT) architecture. Experimental results show that the performance increases significantly, in which DWT running on modern GPU is more than 10 times faster than on modern CPU. Furthermore, by replacing the DWT part of Jasper which is the JPEG2000 reference software, the overall processing is 2~16 times faster than the original JasPer. The GPU-driven render-to-texture architecture proposed in this paper can be used in the general image and computer vision processing for high-speed processing.

Keywords : GPU, JPEG2000, DWT, 화소 쉐이더, JasPer, 프레임 버퍼 객체

I. 서 론

최근 그래픽 가속기 (Graphics Processing Unit: GPU)의 성능이 급격히 발전하고 GPU 자체의 프로그래밍 가능한 특성이 점차 확대됨에 따라 3차원 그래픽스 이외의 범용 목적으로 GPU를 활용할 수 있는 가능성이 대두되었다. 최신 GPU의 연산처리 속도는 대응되는 최신 CPU의 성능을 수 배 능가한다. 예를 들어, 최신 GPU 코어인 NVIDIA G80 (GeForce 8800GTX)의 경우 7억 개의 트랜지스터 집적도와 350 GFLOPS에 근접하는 초고속 연산이 가능한 것에 비하여, 최신 CPU 코어인 Intel Core2 Duo 3.0GHz는 약 50 GFLOPS 가량의 연산을 처리할 수 있다[1~2].

또한, 최근 GPU의 발전 추세의 다른 하나는 GPU 내부 파이프라인의 기능을 사용자가 프로그래밍 할 수 있도록 허용한다는 것이다. 이것은 GPU를 범용으로 사용할 수 있는 가능성을 제공하며, 현재는 제한적이지만 궁극적으로는 GPU를 CPU처럼 사용할 수 있게 한다[3~4]. 이러한 방법은 정점 쉐이더 (vertex shader)와 화소

* 학생회원, ** 평생회원, 인하대학교 정보통신공학부 (Inha University)
*** 정회원, (주)삼성전자 정보통신총괄 정보통신연구소 (Samsung Electronics)
※ 본 논문의 초기 결과는 제19회 영상처리 및 이해에 관한 워크샵[15]에서 발표되었음.
※ 본 연구는 (주)삼성전자의 지원에 의해 수행되었음.
접수일자: 2007년2월15일, 수정완료일: 2007년10월31일

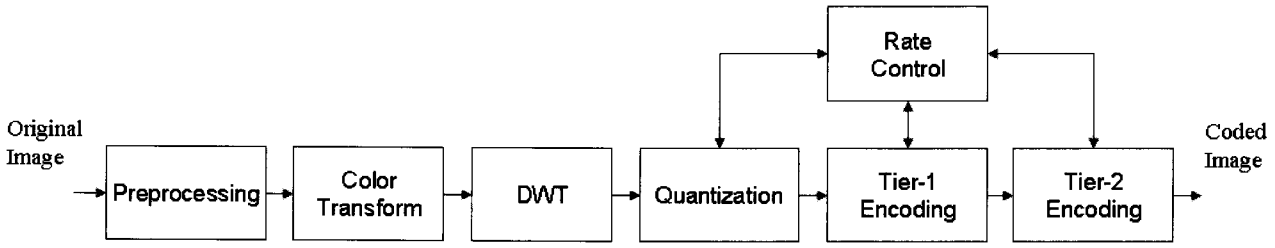


그림 1. JPEG2000의 구조[7, 10]
Fig. 1. The structure of JPEG2000.[7, 10]

쉐이더 (pixel/fragment shader)등의 기법으로 알려져 있다. 최근 DirectX 9.0과 OpenGL 2.0[5]이 발표되었고 HLSL (high-level shading language)과 GLSL (GL shading language)[6]이라는 고급 shading 언어가 표준 사양에 포함됨에 따라 GPU의 고성능 연산을 이용한 범용 어플리케이션의 개발이 보다 가속화 될 전망이다.

범용 목적 (general purpose)으로의 GPU의 응용, 즉 GPGPU를 위해 컴퓨터 그래픽스 이외에서 최근 가장 각광을 받는 응용 분야가 영상처리와 컴퓨터 비전 분야이다. 대부분의 알고리즘은 동일한 명령의 많은 양의 영상 데이터로의 동시 실행, 즉 SIMD (single instruction multiple data) 방식의 접근을 필요로 하며 이들은 3차원 그래픽 알고리즘과의 공통적인 특징이기도 하다. 또한, GPU가 PC뿐만 아니라 핸드폰, PDA와 같은 모바일 기기용으로 개발되고 있고 이러한 것들이 실제 적용되는 기술 동향을 볼 때, 향후 동영상 처리등과 같은 멀티미디어 프로세서의 역할을 GPU가 담당할 수 있을 것으로 기대된다.

본 논문에서는 고급 shading 언어를 이용한 GPU의 범용 활용을 통해 JPEG2000 정지 영상의 고속 압축 알고리즘을 구현한다. 특히, JPEG2000 알고리즘의 핵심 모듈이 되는 DWT 알고리즘을 GPU에서의 fragment shader에서 수행하기 위한 Render-to-Texture를 활용한 구조를 설계하였으며, 이것이 실제 JPEG2000에 적용이 가능한지를 테스트하기 위하여 기존의 JPEG2000을 구현한 JasPer[7~8] 코드의 해당 부분을 대치하여 전체 JPEG2000 변환 과정을 테스트 하였다. 본 논문에서 제시된 프레임버퍼 객체를 이용한 Render-to-Texture 수행 구조는 GPU 기반 영상처리의 기본 틀을 제공하며, 일반적인 영상처리와 컴퓨터 비전 처리를 GPU상에서 고속 수행을 가능하게 한다.

본 논문의 구성은 다음과 같다. 제Ⅱ장에서는 JPEG2000의 구조를 간략히 설명한다. 제Ⅲ장에서는 DWT를 GPU가 수행하도록 하기 위한 Render-to-Texture 수행 구조를 제시한다. 제 Ⅳ장에서는 실험 결과를 보이고, 제Ⅴ장에서 결론을 제시한다.

Ⅱ. JPEG2000

JPEG2000[7, 9~10]은 기존의 JPEG 방식에 비해 점진적 전송과 관심영역 코딩 등의 새로운 기능을 확보하고 낮은 전송률에서 화질을 향상시킨 정지 영상 압축 알고리즘이며 ISO/IEC 및 ITU의 국제 표준으로 등록되어 있다. 최근 디지털 카메라, 모바일 기기, 의료 영상 기기, 원격 탐사 영상 기기 등 다양한 응용 분야에서 사용하기 위한 새로운 영상 포맷으로 많은 관심을 받고 있다.

그림 1에 JPEG2000의 기본적인 인코딩 과정을 제시하였다. 입력된 원본 영상에 대하여 타일링(tiling)과 같은 전처리(preprocessing)과정을 거치게 되고, RGB 영상을 YCrCb로 변환하는 컬러 변환을 수행하게 된다. 그 후 DWT 변환을 수행하고, 양자화 과정과 Tier-1, Tier-2 코딩 과정을 거치게 되면 최종적으로 JPEG2000 압축영상을 얻을 수 있다. 한편, 디코딩 과정은 인코딩 과정의 역순으로 생각할 수 있다.

Ⅲ. GPU 에서의 DWT 알고리즘의 구현

1. Discrete Wavelet Transform

JPEG2000에서는 DWT를 이용하여 intra-component transform을 수행한다. 본 논문에서는 DWT를 구현하기 위하여 일반적으로 잘 알려져 있는 두 가지 방법인 convolution 기법과 lifting[11] 기법을 구현하여 성능의 비교를 수행한다. Convolution 기반의 DWT는 일반적인 신호처리에서의 convolution 방법과 마찬가지로, 이미 정해져 있는 저주파 대역 필터와 고주파 대역 필터의 값을 입력 신호에 곱하여 저주파 성분과 고주파 성분을 구분하게 된다. 식 (1)과 (2)는 convolution 기반의 DWT의 수행 수식을 나타낸다. 입

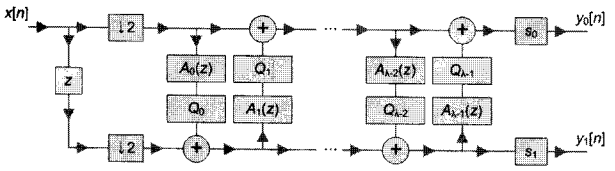


그림 2. Lifting 기반의 DWT 수행구조[10]
Fig. 2. Structure of the lifting-based DWT[10].

력신호 $x(n)$에 대하여 저주파 필터 $h(i)$와 고주파 필터 $g(i)$를 이용하여 입력 신호의 저주파 성분 $y_L(n)$과 고주파 성분 $y_H(n)$을 구할 수 있다.

$$y_L(n) = \sum_{i=0}^{x_L - 1} h(i)x(2n-i) \tag{1}$$

$$y_H(n) = \sum_{i=0}^{x_H - 1} g(i)x(2n-i) \tag{2}$$

본 논문에서는 DWT를 수행하기 위한 보다 효율적인 방법으로 lifting 기반의 DWT를 구현한다. Lifting 기반의 DWT는 convolution 기반의 DWT보다 연산의 수를 줄임으로써 좀 더 빠른 수행시간을 제공한다. 그림 2에 lifting 기반의 DWT 과정 중 분석부(analysis part)의 신호 흐름도를 도시하였다.

Irreversible transform을 의미하는 Daubechies 9/7 탭 필터의 polyphase matrix는 식 (3)과 같다[11].

$$\tilde{P}(z) = \begin{bmatrix} 1 & \alpha(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \beta(1+z) & 1 \end{bmatrix} \begin{bmatrix} 1 & \gamma(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \delta(1+z) & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \tag{3}$$

이 때, 식 (3)의 연산을 다음과 같이 6단계의 과정으로 연산을 수행할 수 있다.

(단계1) $Y_{2n+1} = X_{2n+1} + \alpha \times (X_{2n} + X_{2n+2})$

(단계2) $Y_{2n} = X_{2n} + \beta \times (Y_{2n-1} + Y_{2n+1})$

(단계3) $Y_{2n+1} = Y_{2n+1} + \gamma \times (Y_{2n} + Y_{2n+2})$

(단계4) $Y_{2n} = Y_{2n} + \delta \times (Y_{2n-1} + Y_{2n+1})$

(단계5) $Y_{2n+1} = -K \times Y_{2n+1}$

(단계6) $Y_{2n} = Y_{2n}/K$

여기서, $\alpha = -1.586134342$, $\beta = -0.052980118$,

$\gamma = 0.882911075$, $\delta = 0.443506852$,

$K = 1.230174105$

표 1에 convolution과 lifting 기반의 DWT의 연산수의 비교하였다. 표1에 제시된 바와 같이, lifting 기반의 DWT가 convolution 기반에 비하여 1/3 정도의 곱셈 연산이 감소하는 것을 확인할 수 있다.

2. GPU 기반의 영상처리 프레임워크

그림 3은 본 논문에서 제안하는 GPU와 fragment shader를 이용한 영상처리 프레임워크의 기본 구조를 나타낸다. 영상처리의 대상이 되는 입력 영상은 채널당 32bit 부동소수점 형식의 텍스처 데이터[12]로 비디오 메모리로 로딩 되며, fragment shader에서는 영상처리의 특정 알고리즘을 입력 영상에 대해 수행하여 그 결과를 Frame Buffer Object (FBO)[13]라고 불리는 또 다른 텍스처 메모리로 출력한다. 이 과정을 Off-Screen Rendering 또는 Render-To-Texture라고 하는데, fragment shader의 처리 결과를 기본 출력 대상인 frame buffer로 전달하는 것이 아니고 텍스처 메모리로 보내어 재사용이 가능하도록 하는 되먹임 (feedback) 구조이다. CPU가 수행하는 응용 프로그램에서는 화면 전체와 일치하는 사각형의 영역에 대하여 렌더링을 수행하고 FBO의 내용을 텍스처로 바인딩 하게 되면, 결국 fragment shader가 수행한 영상처리의 결과가 텍스처로 렌더링 되는 것이다. 또한, 입력 텍스처와 출력 텍스처를 부동 소수점 형식으로 지정하여 영상처리 과정에서 생성되는 중간 결과를 정확히 표현할 수 있다. 그리고 FBO의 유용한 특징들 중 하나인 입력 텍스처와

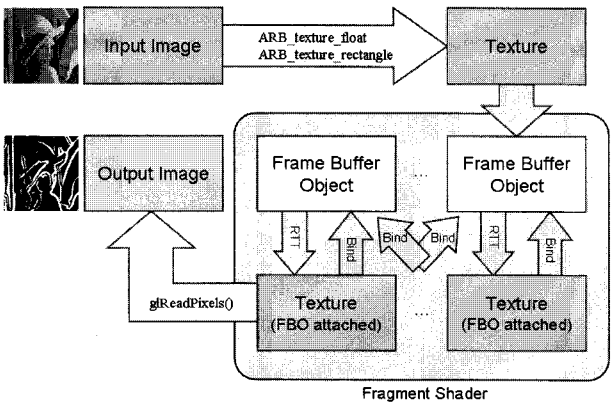


그림 3. GPU 기반 영상처리 프레임워크의 기본구조
Fig. 3. Basic framework of GPU-based image processing framework.

표 1. Convolution 기반의 DWT와 lifting 기반의 DWT의 연산수 비교

Table 1. Comparison of computations between convolution-based and lifting-based DWT.

	Convolution		Lifting Scheme	
Filter	곱셈수	덧셈수	곱셈수	덧셈수
5/3	4	6	2	4
9/7	9	14	6	8

출력 텍스처를 동일하게 지정할 수 있는 방법을 이용하여 부가적인 과정 없이 출력 결과를 그대로 입력으로 이용함으로써 수행시간을 단축할 수 있으며 비디오 메모리의 사용량 또한 감소시킬 수 있다.

3. GPU에서의 DWT 구현

GPU를 이용하여 convolution 기반의 DWT와 lifting 기반의 DWT 구현 과정을 살펴보았을 때, 두 개의 과정 모두 입력 영상은 부동소수점 형식의 텍스처로 지정된다. 우선 convolution의 경우 가로방향과 세로방향의 DWT를 위하여 총 두 번의 렌더링이 수행이 되고, 각각의 렌더링 과정 중 J. Wang[14]과 유사한 방법으로 모든 픽셀에 대해서 현재 픽셀을 기준으로 인접한 픽셀들의 값을 텍스처로부터 얻어와 곱셈과 덧셈 연산을 수행한다. 반면 lifting의 경우 식 (3)의 여섯 단계의 과정 중 마지막 세 단계를 한 번의 렌더링으로 수행할 수 있으므로 가로방향과 세로방향에 대하여 각각 네 번씩의 렌더링이 이루어지게 된다. 이때 각각의 렌더링 과정에서 출력 텍스처와 입력 텍스처를 동일하게 지정함으로써 메모리간 데이터 이동에 소요되는 수행 시간을 감소시키고, 모든 픽셀에 대하여 현재 픽셀을 기준으로 좌우의 두 픽셀들의 값을 텍스처로부터 얻어와 곱셈과 덧셈 연산을 수행한다.

IV. 실험 결과

본 논문에서는 다양한 환경에서의 성능 평가를 위하여 각각 두 가지의 GPU와 CPU상에서의 실험을 통하여 수행 속도 차이를 측정하였다. 즉, 본 실험은 (1) (AMD Athlon 64 2.0GHz CPU + NVIDIA GeForce 8800GTX (G80) GPU (756MB)) (2) (Intel Pentium D 940 3.2GHz CPU + NVIDIA GeForce 8800GTS (G80) GPU (646MB)) 조합의 컴퓨터 환경에서 수행되었다.

수행 속도 측정에 있어 GeForce 8800 GTX는 128개의 픽셀 파이프라인을 장착하고 있고, GTS는 96개의 파이프라인을 가지고 있으므로 GTX 모델에서의 수행 속도가 전반적으로 우수하게 관측될 것이다. 또한 마찬가지로 Athlon 64에 비해 듀얼 펜티엄 프로세서가 보다 좋은 성능을 보일 것이다.

1. DWT 실험 결과

그림 4에 실험의 수행 순서도를 나타내었으며, 표 2에 convolution 기반의 DWT의 수행 속도를 제시하였다. DWT의 레벨은 1, 3, 5의 3단계에 대해 실험하였으

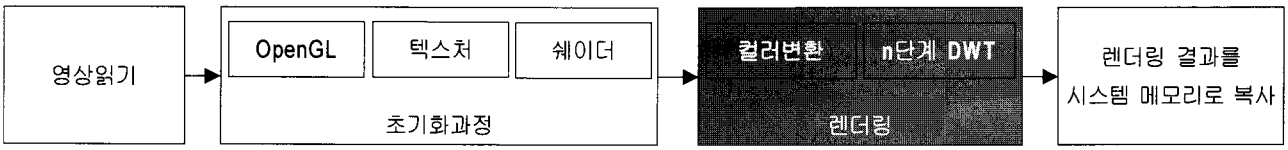


그림 4. 실험에 이용된 DWT의 수행 순서도. GPU가 수행하는 블록은 음영으로 표시.

Fig. 4. Block diagram of experimental procedure of DWT. Shaded blocks are performed on GPU.

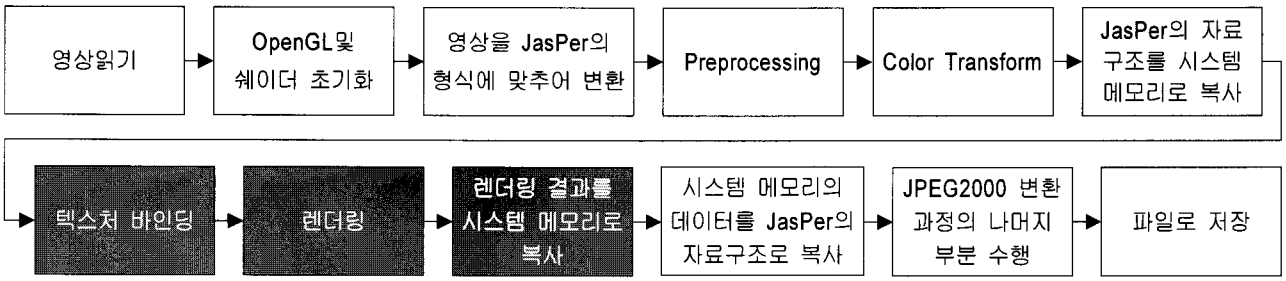


그림 5. 실험에 이용된 Jasper의 JPEG2000 인코딩 순서도. GPU가 수행하는 블록은 음영으로 표시.

Fig. 5. Block diagram of Jasper's JPEG2000 encoding procedure in the experiment. Shaded blocks are performed on GPU.

표 2. Convolution 기반 DWT의 수행 속도 비교
Table 2. Comparison of DWT processing time using convolution.

해상도 및 결과분류 \ 프로세서		GPU (Level=1)		GPU (Level=3)		GPU (Level=5)		CPU(Level=1)	
		8800GTS	8800GTX	8800GTS	8800GTX	8800GTS	8800GTX	AMD Athlon64	INTEL Pentium D
512×512	수행 시간	0.0019	0.0013	0.0027	0.0018	0.0028	0.0019	0.0715	0.0584
	CPU 수행 시간 대비 비율	3%	2%	5%	3%	5%	3%	-	-
1024×1024	수행 시간	0.0075	0.0050	0.0101	0.0068	0.0105	0.0071	0.2861	0.2330
	CPU 수행 시간 대비 비율	3%	2%	4%	2%	4%	2%	-	-
2048×2048	수행 시간	0.0294	0.0200	0.0397	0.0268	0.0408	0.0276	1.1753	0.9299
	CPU 수행 시간 대비 비율	3%	2%	4%	2%	4%	2%	-	-

표 3. Lifting 기반 DWT의 수행 속도 비교
Table 3. Comparison of DWT processing time using lifting.

해상도 및 결과분류 \ 프로세서		GPU (Level=1)		GPU (Level=3)		GPU (Level=5)		CPU(Level=1)	
		8800GTS	8800GTX	8800GTS	8800GTX	8800GTS	8800GTX	AMD Athlon64	INTEL Pentium D
512×512	수행 시간	0.0017	0.0012	0.0023	0.0017	0.0024	0.0017	0.0400	0.0267
	CPU 수행 시간 대비 비율	7%	3%	9%	4%	9%	4%	-	-
	Convolution 수행 시간 대비 비율	89%	95%	85%	91%	84%	89%	56%	46%
1024×1024	수행 시간	0.0069	0.0051	0.0091	0.0067	0.0094	0.0069	0.1662	0.1059
	CPU 수행 시간 대비 비율	6%	3%	9	4%	9%	4%	-	-
	Convolution 수행 시간 대비 비율	92%	101%	90	99%	89%	98%	58%	45%
2048×2048	수행 시간	0.0278	0.0206	0.0368	0.0272	0.0376	0.0279	0.6377	0.4177
	CPU 수행 시간 대비 비율	7%	3%	9	4%	9%	4%	-	-
	Convolution 수행 시간 대비 비율	94%	103%	93	101%	92%	101%	54%	45%

표 4. JPEG2000의 인코딩 수행 시간 비교. 퍼센트 수치는 동일한 CPU로 전체 과정을 수행하는 경우에 대한 수행 시간의 비율을 의미.
Table 4. Comparison of JPEG2000 encoding time. Percent value denotes the ratio to the running time when the whole encoding is perform on the same CPU.

Resolution \ Device	GPU				CPU	
	8800GTS (with INTEL Pentium D)		8800GTX (with AMD Athlon 64)		AMD Athlon 64	INTEL Pentium D
512×512	0.0159	52%	0.0167	14%	0.1232	0.0305
1024×1024	0.0583	10%	0.0649	6%	1.0440	0.5553
2048×2048	0.2286	8%	0.2345	5%	4.5996	2.7816

며 제시된 바와 같이 고사양의 GPU 일수록, 그리고 고사양의 CPU 일수록 수행시간이 단축되었음을 알 수 있다. CPU와의 수행 시간 비교는 DWT의 분할 레벨이 1회인 경우와 수행하였고, CPU 수행시간 대비 5%이하로 대폭 감소하였음을 알 수 있다.

다음으로 lifting 기반의 DWT에 대한 수행 속도 측정 및 비교 분석을 수행하였으며 그 결과는 표 3에 도시한 바와 같다. CPU에서 lifting을 수행했을 때와 비교하였을 때 90%이상의 수행 시간이 감소함을 알 수 있다. Convolution 기반의 DWT 수행 시간과 비교해 보았을 때, GPU상에서의 수행은 5~10%내외로 다소 감소하였으나 전반적으로 유사한 수행 시간을 보인다고 할 수 있다. 이는 lifting을 GPU상에서 shader로 구현하였을 때 발생하는 렌더링 회수가 convolution에 비해 6회 많기 때문인데, 현재 FBO를 이용한 렌더링에서는 fragment shader의 병렬 처리에서 공유 메모리가 존재하지 않는 근본적인 문제에 기인한다. 다만, 추후 새로운 GPU와 shader 모델에서는 이와 같은 제한 조건을 완화시키려는 요구를 반영할 예정이므로, lifting 기반의 DWT 수행 속도는 향후 개선될 수 있다고 할 수 있다.

2. JasPer와의 통합

본 논문에서 구현된 DWT가 실제 JPEG2000에 적용이 가능한지의 여부와 성능 평가를 위하여 본 논문에서 구현된 영상처리 프레임워크에 기존의 JPEG2000 공개 소프트웨어를 통합하였다. 이 때 이용된 공개 소프트웨어는 JasPer로 불리는 JPEG 표준화 단체의 표준 소프트웨어이다[7~8]. 또한, 기존의 DWT 부분을 본 연구에서 개발한 lifting 기반의 DWT 구현으로 변경하여 전체 JPEG2000의 변환 과정을 테스트 하였다. 그림 5에 실험의 수행 순서도를 나타내었다.

표 4에 입력 영상에 대해 JPEG2000의 비트열을 생성하는 인코딩 과정에 대한 실험 결과를 제시하였다. 제안하는 GPU 기반의 DWT 수행이 이용된 경우, CPU로 전체 과정을 수행하는 것에 비해 2048×2048의 고해상도 영상 기준 5~8%로 계산 량이 대폭 감소하였다. 이 때, 상대적인 성능 비교는 동일한 CPU가 사용된 경우에 대해 측정되었다. 또한, 8800GTX의 수행 시간이 GTS의 수행시간보다 다소 크게 측정된 것은 DWT를 제외한 부분을 수행한 CPU의 성능 차이에서 기인한다.

표 4에 제시된 결과에 의하면, 해상도가 증가할수록 전체 인코딩 과정에서 DWT가 차지하는 비중이 증가하기 때문에 계산량 감소폭이 크다는 것을 알 수 있으며, 이는 매우 유용한 결과라고 할 수 있다. 즉, JPEG2000이 고해상도 영상의 압축에 주로 이용된다는 점을 생각하면, DWT 부분의 GPU 구현을 통해 얻어진 상당한 수준의 계산량 감소가 큰 의미를 지닌다.

V. 결 론

본 논문에서는 GPU를 사용하여 JPEG2000 정지영상 압축 알고리즘의 DWT 연산을 고속으로 수행하기 위한 효율적인 구조와 방법을 제안하였다. 본 논문에서는 DWT 알고리즘을 GPU에서의 fragment shader에서 수행하기 위한 Render-to-Texture 구조를 설계하였으며, 실험 결과 CPU에서의 처리에 비해 GPU에서 구현된 DWT의 경우 10배 이상의 수행 속도의 향상을 보였다.

또한 기존의 JasPer와 성능을 비교하였을 때, 제안하는 기법으로 DWT 부분을 대치하는 경우 전체 과정을 CPU로 처리하는 것에 비하여 수행 시간이 52~5%로 감소함을 보였다.

본 연구에서 이용된 GPU는 NVIDIA사의 최신 GPU를 이용하였는데, GPU의 성능은 또한 개발사가 지원하는 드라이버의 성능에 좌우된다고 할 수 있다. 지속적으로 NVIDIA사는 개선된 드라이버를 제공하므로 본 논문에서 제시된 결과는 이에 따라 보다 더 개선될 수 있을 것이다.

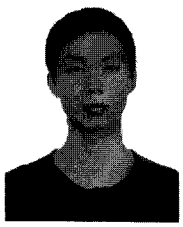
향후 연구 방향으로 본 논문에서 개발된 GPU 기반의 영상처리 프레임워크를 동영상으로 확장하여 실시간 동영상 및 비전 알고리즘 처리를 수행하는 것을 고려하고 있다. 또한, NVIDIA의 차세대 GPU 구조인 CUDA (compute unified device architecture)를 이용하여, 보다 자유로운 병렬 처리와 다양한 형태의 메모리 사용을 통해 JPEG2000에서 DWT의 다음 단계인 BPC (bit-plane coding)와 BAC (Binary arithmetic coding) 부분을 GPU로 가속함으로써 보다 확장된 JPEG2000의 가속을 위한 효율적인 GPU 기반의 영상처리 프레임워크의 구축이 필요하다고 할 수 있다.

참 고 문 헌

[1] http://developer.nvidia.com/page/documentation.html
[2] http://ati.amd.com/developer/index.html
[3] R. Fernando (editor), *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Addison-Wesley, 2004.

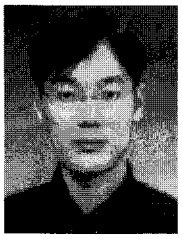
[4] M. Pharr (editor), *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Addison-Wesley, 2005.
[5] http://www.opengl.org
[6] R. Rost, *OpenGL Shading Language Second Edition*, Addison-Wesley, 2006.
[7] Information Technology - JPEG2000 Image Coding System, ISO/IEC International Standard 15444-1, ITU Recommendation T.800, 2000.
[8] M. D. Adams and F. Kossentini, "JasPer: A software-based JPEG-2000 codec implementation," *Proc. IEEE International Conference on Image Processing*, September 2000.
[9] M. Rabbani and R. Joshi, "An overview of the JPEG 2000 still image compression standard," *Signal Processing: Image Communication*, vol. 17, no. 1, pp. 3-48, January 2002.
[10] T. Acharya and P. Tsai, *JPEG2000 Standard for Image Compression*, Wiley-Interscience, 2005.
[11] W. Sweldens, "The Lifting scheme: a new philosophy in biorthogonal wavelet constructions," *Proc. SPIE*, vol. 2569: Wavelet Applications in Signal and Image Processing III, pp. 68-79, September 1995.
[12] http://www.nvidia.com/dev_content/nvopenglspecs/GL_ARB_texture_float.txt
[13] http://www.nvidia.com/dev_content/nvopenglspecs/GL_EXT_framebuffer_object.txt
[14] J. Wang, T. T. Wang, P. A. Heng and J. Wang, "Discrete Wavelet Transform on GPU," *Proc. ACM Workshop on General Purpose Computing on Graphics Processors*, pp. C-41, August 2004.
[15] 이만희, 박인규, 원석진, 조성대, "JPEG2000에서 GPU를 이용한 DWT의 가속," *제19회 영상처리 및 이해에 관한 워크샵*, pp. 415-418, 2007년 2월.

저 자 소 개



이 만 희(학생회원)
2006년 2월 인하대학교 컴퓨터공학과 공학사.
2006년 3월~현재 인하대학교 정보통신공학부 석사과정.
2007년 4월~현재 한국전자통신연구원(ETRI) 위촉연구원.
<주관심분야 : 영상기반 모델링 및 렌더링, sketch-based interface, GPGPU>

박 인 규(정회원)
1995년 2월 서울대학교 제어계측공학과 공학사.
1997년 2월 서울대학교 제어계측공학과 공학석사.
2001년 8월 서울대학교 전기컴퓨터공학부 공학박사.
2001년 9월~2004년 3월 삼성종합기술원(SAIT) 멀티미디어랩 전문연구원.
2004년 3월~현재 인하대학교 정보통신공학부 조교수.
2007년 1월~현재 Mitsubishi Electric Research Laboratories (MERL) 방문연구원.
<주관심분야 : 컴퓨터 그래픽스 및 비전, 영상기반 모델링 및 렌더링, 3D 얼굴 모델링, computational photography, GPGPU>



원 석 진(정회원)
1995년 2월 서울대학교 제어계측공학과 공학사.
1997년 2월 서울대학교 제어계측공학과 공학석사.
1997년~2002년 대우전자 디지털연구소.
2002년~2003년 액토즈소프트 게임개발실.
2005년~현재 (주)삼성전자 통신연구소.
<주관심분야 : 멀티미디어, 그래픽스, 디지털 콘텐츠>

조 성 대(정회원)
1996년 숭실대학교 전자계산학과 공학사.
2000년 Rensselaer Polytechnic Institute 전자컴퓨터공학 공학석사.
2002년 Rensselaer Polytechnic Institute 전자컴퓨터공학 공학박사.
2004년 RPI 영상처리센터 박사후 연구원.
2004년 9월~현재 (주)삼성전자 정보통신연구소.
<주관심분야 : 멀티미디어 영상처리, 컬러처리, 압축, 통신>