



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2008-0073511
(43) 공개일자 2008년08월11일

(51) Int. Cl.

G06T 1/00 (2006.01) G06T 1/20 (2006.01)

G06F 3/00 (2006.01)

(21) 출원번호 10-2007-0012215

(22) 출원일자 2007년02월06일

심사청구일자 2007년02월06일

(71) 출원인

삼성전자주식회사

경기도 수원시 영통구 매탄동 416

인하대학교 산학협력단

인천 남구 용현동 253 인하대학교

(72) 발명자

원석진

경기도 성남시 분당구 서현동 시범단지한양아파트
311동 105호

오윤제

경기도 용인시 기흥구 마북동 삼성래미안1차아파트
107동 1801호

(뒷면에 계속)

(74) 대리인

이건주

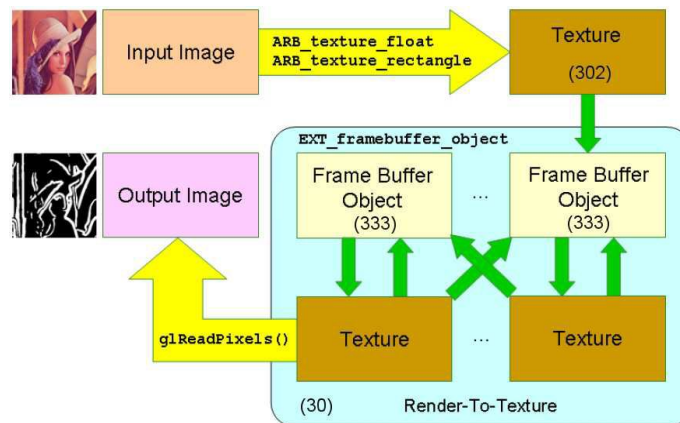
전체 청구항 수 : 총 6 항

(54) 그래픽 가속기 기반의 고속 영상 처리 방법 및 그 장치

(57) 요약

본 발명은 그래픽 가속기 기반의 고속 영상 처리 방법에 있어서, 영상 처리를 위한 입력 영상을 32비트 부동 소수점(floating point)의 텍스처의 형식으로 처리하는 과정과, 적어도 하나 이상의 프레임버퍼 객체(FBO: Framebuffer Object)를 통해 영상 처리를 위한 미리 설정된 알고리즘을 입력 영상에 대해 수행하여 그 결과를 텍스처 데이터로서 출력하는 과정을 수행한다.

대표도 - 도3



(72) 발명자

조성대

경기도 용인시 풍덕천동 664 동아아파트 112-1104

홍태화

서울특별시 구로구 신도림동 우성아파트 301동 50
6호

김수균

서울특별시 강남구 포이동 252-10 현대아트빌라
103호

이민우

경기도 수원시 장안구 천천동 560-10번지 202호

박인규

인천광역시 남구 용현동 253번지 인하대학교 산학
협력단벤처창업관 303호

이만희

인천광역시 남구 용현동 253번지 인하대학교 산학
협력단벤처창업관 303호

특허청구의 범위

청구항 1

그래픽 가속기 기반의 고속 영상 처리 방법에 있어서,

상기 영상 처리를 위한 입력 영상을 32비트 부동 소수점(floating point)의 텍스처의 형식으로 처리하는 과정과,

적어도 하나 이상의 프레임버퍼 객체(FBO: Framebuffer Object)를 통해 상기 영상 처리를 위한 미리 설정된 알고리즘을 상기 입력 영상에 대해 수행하여 그 결과를 텍스처 데이터로서 출력하는 과정을 포함함을 특징으로 하는 고속 영상 처리 방법.

청구항 2

제1항에 있어서, 상기 FBO를 통해 상기 입력 영상에 대한 알고리즘을 수행하는 것은 컨볼루션(convolution) 기반의 이산 웨이블릿 변환(Discrete Wavelet Transform: DWT)이며, 가로방향(horizontal)과 세로방향(vertical)의 DWT를 위하여 각각 하나의 FBO가 미리 마련되며, 총 두번의 렌더링을 수행하며, 상기 컨볼루션 기반의 DWT는 하기 수학식 5와 수학식 6에서 입력신호 $x(n)$ 에 대하여 로우패스 필터 $h(i)$ 와 하이패스 필터(i)를 이용하여 입력 신호의 저주파 성분 $y_L(n)$ 과 고주파 성분 $y_H(n)$ 을 구하는 방식임을 특징으로 하는 고속 영상 처리 방법.

수학식 5

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

수학식 6

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

청구항 3

제1항에 있어서, 상기 FBO를 통해 상기 입력 영상에 대한 알고리즘을 수행하는 것은 리프팅(lifting) 기반의 이산 웨이블릿 변환(Discrete Wavelet Transform: DWT)이며, 가로방향(horizontal)과 세로방향(vertical)의 DWT를 위하여 각각 하나의 FBO가 미리 마련되어 이를 이용하여 렌더링을 수행하며, 상기 리프팅(lifting) 기반의 DWT는 수학식 7에 따른 방식임을 특징으로 하는 고속 영상 처리 방법.

수학식 7

$$(S1) \ Y_{2n+1} = X_{2n+1} + \alpha \times (X_{2n} + X_{2n+2}) \quad \alpha = -1.586134342$$

$$(S2) \ Y_{2n} = X_{2n} + \beta \times (Y_{2n-1} + Y_{2n+1}) \quad \beta = -0.052980118$$

$$(S3) \ Y_{2n+1} = Y_{2n+1} + \gamma \times (Y_{2n} + Y_{2n+2}) \quad \gamma = 0.882911075$$

$$(S4) \ Y_{2n} = Y_{2n} + \delta \times (Y_{2n-1} + Y_{2n+1}) \quad \delta = 0.443506852$$

$$(S5) \ Y_{2n+1} = -K \times Y_{2n+1} \quad K = 1.230174105$$

$$(S6) \ Y_{2n} = Y_{2n} / K$$

청구항 4

그래픽 가속기 기반의 고속 영상 처리 장치에 있어서,

3차원 정점을 2차원 화면 좌표계로 변환시키는 3차원 기하 변환을 수행하며, 조명 및 반사 특성에 의해 정점의

색을 결정하는 정점 셰이더와,

상기 정점 셰이더에 출력된 데이터를 정렬하여 프래그먼트(fragment)로 변환하는 래스터라이저와,

상기 래스터라이저에서 제공받은 프래그먼트를 바탕으로 비디오 메모리에서 텍스처 데이터를 이용하여 셰이더 연산을 수행하는 화소 셰이더를 포함하며,

상기 화소 셰이더는 상기 영상 처리를 위한 입력 영상을 32비트 부동 소수점(floating point)의 텍스처의 형식으로 제공받아, 상기 영상 처리를 위한 미리 설정된 알고리즘을 입력 영상에 대해 수행하여 그 결과를 텍스처 데이터로서 출력하여 재사용 가능하도록 하는 적어도 하나 이상의 프레임버퍼 객체(FBO: Framebuffer Object)를 구비함을 특징으로 하는 고속 영상 처리 장치.

청구항 5

제4항에 있어서, 상기 FBO를 통해 상기 입력 영상에 대한 알고리즘을 수행하는 것은 컨볼루션(convolution) 기반의 이산 웨이블릿 변환(Discrete Wavelet Transform: DWT) 또는 리프팅(lifting) 기반의 이산 웨이블릿 변환이며, 가로방향(horizontal)과 세로방향(vertical)의 DWT를 위하여 각각 하나의 FBO가 미리 마련됨을 특징으로 하는 고속 영상 처리 장치.

청구항 6

제5항에 있어서, 상기 컨볼루션 기반의 DWT는 하기 수학식 8과 수학식 9에서 입력신호 $x(n)$ 에 대하여 로우패스 필터 $h(i)$ 와 하이패스 필터 $g(i)$ 를 이용하여 입력 신호의 저주파 성분 $y_L(n)$ 과 고주파 성분 $y_H(n)$ 을 구하는 방식이며, 상기 리프팅(lifting) 기반의 DWT는 수학식 10에 따른 방식임을 특징으로 하는 고속 영상 처리 장치.

수학식 8

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

수학식 9

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

수학식 10

$$(S1) \quad Y_{2n+1} = X_{2n+1} + \alpha \times (X_{2n} + X_{2n+2}) \quad \alpha = -1.586134342$$

$$(S2) \quad Y_{2n} = X_{2n} + \beta \times (Y_{2n-1} + Y_{2n+1}) \quad \beta = -0.052980118$$

$$(S3) \quad Y_{2n+1} = Y_{2n+1} + \gamma \times (Y_{2n} + Y_{2n+2}) \quad \gamma = 0.882911075$$

$$(S4) \quad Y_{2n} = Y_{2n} + \delta \times (Y_{2n-1} + Y_{2n+1}) \quad \delta = 0.443506852$$

$$(S5) \quad Y_{2n+1} = -K \times Y_{2n+1} \quad K = 1.230174105$$

$$(S6) \quad Y_{2n} = Y_{2n} / K$$

명세서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

- <6> 본 발명은 그래픽 가속기 기반의 고속 영상 처리에 관한 기술로서, 특히 그래픽 가속기의 범용 활용을 통해 JPEG2000 정지 영상의 고속 압축 알고리즘의 구현을 가능하도록 하기 위한 방법에 관한 것이다.
- <7> 최근 그래픽 가속기(Graphics Processing Unit: GPU)의 성능이 급격히 발전하고 GPU 자체의 프로그래밍 가능한 특성이 점차 확대됨에 따라 3차원 그래픽스의 기본적인 기하 변환과 래스터화(rasterization) 이외의 범용 목적으로 GPU를 활용할 수 있는 가능성이 대두되었다(하기 참조문헌[7], [8], [9], [10] 참조). 최근 GPU의 연산처리 속도는 대응되는 최신 CPU의 성능을 10개 가까이 능가한다. 예를 들어 최신 GPU 코어인 NVIDIA G80(GeForce 8800GTX)의 경우 7억 개의 트랜지스터 집적도와 350 GFLOPS에 근접하는 초고속 연산이 가능한데, 기존 코어들의 성능과 비교하였을 때 이 성능 지표의 증가도는 괄목할 만 하다.
- <8> GPU의 최근 발전 추세의 다른 하나는 GPU 내부 파이프라인의 기능을 사용자가 프로그래밍 할 수 있도록 허용한다는 것이다. 이러한 방법은 정점 셰이더(vertex shader)와 화소 셰이더(pixel/fragment shader) 등의 기법으로 알려져 있다. 예를 들어, 투영변환이나 조명모델 또는 텍스처 매핑 방법의 종류를 변화시키거나 기존의 고정된 파이프라인에서는 다룰 수 없었던 좀더 사실적인 그래픽 효과가 셰이더(shader)를 이용하여 GPU에서 그래픽스 파이프라인의 기능을 재정의 함으로써 가능하게 되었다. 근래에 DirectX 9.0과 OpenGL 2.0이 발표되고 HLSL(High Level Shading Language)과 GLSL(OpenGL Shading Language)이라는 고급 셰이딩(shading) 언어를 이용한 셰이더 프로그래밍이 DirectX와 OpenGL의 표준 사양에 포함됨에 따라 GPU의 고성능 연산을 이용한 범용 어플리케이션의 개발이 보다 가속화 될 전망이다.
- <9> 이와 같은 특징을 이용하여 컴퓨터 그래픽스 분야에서는 범프 매핑(bumpmapping), 환경 매핑(environment mapping), 파티클 효과(particle effect), 충돌 처리(collision detection)등과 같이 OpenGL이나 DirectX의 API가 직접 지원하지 않는 특수 효과를 정점 셰이더와 화소 셰이더를 이용하여 구현할 수 있게 되었다(하기 참조문헌[7], [8] 참조).
- <10> 범용 목적(general purpose)으로의 GPU의 응용, 즉 GPGPU(General Purpose GPU)를 위해 컴퓨터 그래픽스 이외에서 최근 가장 각광을 받는 응용 분야가 영상처리와 컴퓨터 비전 분야이다. 영상 처리 및 컴퓨터 비전 처리 알고리즘은 빈번한 영상 데이터의 전송과 화소, 화소 블록, 또는 특징(feature)에 대한 실시간 데이터 처리가 필수적인 분야이며 이는 3차원 그래픽 알고리즘과의 공통적인 특징이다. 또한 GPU가 PC뿐 아니라 핸드폰, PDA와 같은 모바일 기기용으로 개발되고 탑재되는 기술 동향을 볼 때, 향후 동영상 처리 등과 같은 멀티미디어 프로세서의 역할을 GPU가 담당할 수 있을 것으로 기대된다.

발명이 이루고자 하는 기술적 과제

- <11> 본 발명은 고급 GPU의 범용 활용을 통해 JPEG2000 정지 영상의 고속 압축 알고리즘, 특히 JPEG2000 알고리즘의 핵심 모듈이 되는 DWT(Discrete Wavelet Transform) 알고리즘의 구현을 가능하도록 한다.
- <12> 이에 본 발명의 일 견지에 따르면, 본 발명은 그래픽 가속기 기반의 고속 영상 처리 방법에 있어서, 상기 영상 처리를 위한 입력 영상을 32비트 부동 소수점(floating point)의 텍스처의 형식으로 처리하는 과정과, 적어도 하나 이상의 프레임버퍼 객체(FBO: Framebuffer Object)를 통해 상기 영상 처리를 위한 미리 설정된 알고리즘을 상기 입력 영상에 대해 수행하여 그 결과를 텍스처 데이터로서 출력하는 과정을 수행함을 특징으로 한다.
- <13> 본 발명의 다른 견지에 따르면, 본 발명은 그래픽 가속기 기반의 고속 영상 처리 장치에 있어서; 3차원 정점을 2차원 화면 좌표계로 변환시키는 3차원 기하 변환을 수행하며, 조명 및 반사 특성에 의해 정점의 색을 결정하는 정점 셰이더와; 상기 정점 셰이더에 출력된 데이터를 정렬하여 프래그먼트(fragment)로 변환하는 래스터라이저와; 상기 래스터라이저에서 제공받은 프래그먼트를 바탕으로 비디오 메모리에서 텍스처 데이터를 이용하여 셰이더 연산을 수행하는 화소 셰이더를 포함하며; 상기 화소 셰이더는 상기 영상 처리를 위한 입력 영상을 32비트 부동 소수점(floating point)의 텍스처의 형식으로 제공받아, 상기 영상 처리를 위한 미리 설정된 알고리즘을 입력 영상에 대해 수행하여 그 결과를 텍스처 데이터로서 출력하여 재사용 가능하도록 하는 적어도 하나 이상의 프레임버퍼 객체(FBO: Framebuffer Object)를 구비함을 특징으로 한다.

발명의 구성 및 작용

- <14> 이하 본 발명에 따른 바람직한 실시예를 첨부한 도면을 참조하여 상세히 설명한다. 하기 설명에서는 구체적인 구성 소자 등과 같은 특정 사항들이 나타나고 있는데 이는 본 발명의 보다 전반적인 이해를 돕기 위해서 제공된 것일 뿐 이러한 특정 사항들이 본 발명의 범위 내에서 소정의 변형이나 혹은 변경이 이루어질 수 있음은 이 기

술분야에서 통상의 지식을 가진 자에게는 자명하다 할 것이다.

- <15> 최근의 GPU는 프로그래머블(programmable)한 특성이 점점 증가하는 추세이며 일반 사용자가 고수준 언어로서 GPU의 파이프라인 기능을 변경할 수 있도록 한다. 도 1에는 종래의 고정 파이프라인의 GPU 구조에서 개량된 본 발명이 적용되는 프로그래머블(programmable) 파이프라인의 GPU 구조도를 도시하였다. 도 1을 참조하면, GPU(12)는 CPU(10)의 어플리케이션(100)으로 제공된 정점(Vertex) 데이터(및 원시 데이터)를 제공받아 3 차원 정점을 2 차원 화면 좌표계로 변환시키는 3 차원 기하 변환을 수행하고, 또한 조명 및 반사 특성에 의해 정점의 색을 결정하는 정점 셰이더(122)와, 정점 셰이더(122)에 출력된 데이터를 다시 정렬하여 후단의 화소 셰이더(126)에서 처리할 수 있도록 프래그먼트(fragment)로 변환하는 래스터라이저(124)와, 래스터라이저(124)에서 제공받은 프래그먼트를 바탕으로 비디오 메모리(128)에서 텍스처 데이터를 이용하여 셰이더 연산을 수행하는 화소 셰이더(126)를 구비한다. 이때 화소 셰이더(126)는 이러한 셰이더 연산을 수행하여 2차원 화면 좌표계에서, 다각형(polygon) 내부 화소의 색상을 결정하게 된다. 이러한 화소 셰이더(126)는 예를 들어 OpenGL 셰이딩 언어로 작성된 프로그램들로 구현될 수 있으며, 프래그먼트 셰이더, 또는 프래그먼트 프로세서라고도 한다. 이러한 구성을 가지므로, GPU(12)의 상기 정점 셰이더(122)는 3차원 T&L(transformation and lighting) 구조를 바꿀 수 있으며, 화소 셰이더(126)는 화소 단위의 셰이딩 구조를 바꿀 수 있다.
- <16> 상기에서 고수준 셰이딩 언어로는 2004년 3월 출시된 DirectX9.0에서 지원되는 HLSL(high-level shading language)과 2005년 10월 OpenGL 2.0 스펙에 포함된 OGLSL(OpenGL shading language, 참조문헌[13])이 대표적이다. 최근에는 OpenGL의 모바일용 표준인 OpenGL ES(embedded system) 2.0에 역시 프로그래머블 셰이더가 포함되어 PC와 모바일 기기의 모든 멀티미디어 기기에서 셰이더를 이용한 그래픽, 멀티미디어 처리 어플리케이션의 활용이 가능해졌다. 참조문헌[11]에서는 OpenGL ES 2.0의 프로그래머블 파이프라인 구조가 개시되고 있다.
- <17> 최근 기술 동향을 보면, 개발되는 알고리즘은 예외없이 셰이더를 이용하여 구현하고 고속 알고리즘화를 하고 있음을 알 수 있다. 이미 고정 파이프라인에서는 하드웨어의 가속을 받을 수 없는 알고리즘들을 셰이더를 이용하여 이미 가속하고 있으며 이러한 방향은 그래픽스 이외의 분야로 확장되고 있으며 특히 인접 분야인 영상처리 분야로의 활용이 기대되고 있다(참조문헌[14]).
- <18> 한편 GPU 기반의 영상처리 라이브러리의 구축은 현재까지는 알려진 바 없으며, 특히 웨이블릿 코덱(wavelet codec)과 같은 계산량이 많은 변환을 필요로 하는 JPEG2000 영상 압축 기법에 GPU 기반의 효율적인 알고리즘이 개발되지 못하였다.
- <19> 이에, 본 발명에서는 고수준 셰이딩 언어를 이용한 GPU의 범용 활용을 통해 JPEG2000 정지 영상의 고속 압축 알고리즘을 구현하고자 한다. JPEG2000은 웨이블릿(Wavelet) 변환 기반의 정지 영상 압축을 위한 새로운 국제 표준으로서 기존의 JPEG 방식의 단점을 개선한 정지 영상 압축 알고리즘이며, 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(참조문헌[2], [3], [4]). 또한 JPEG2000은 ISO/IEC 및 ITU의 국제 표준으로 등록되어 있다(참조문헌[1]).
- <20> 이와 같이, 본 발명에서는 JPEG2000 코덱을 GPU 및 셰이딩 언어를 이용하여 구현하기 위한 기본 영상처리 환경을 구축하고, 이를 기반으로 JPEG2000의 주요 모듈, 특히 DWT(Discrete Wavelet Transform) 모듈을 화소 셰이더를 이용하여 구현하고자 한다.
- <21> [JPEG2000의 기본 구조]
- <22> 도 2는 일반적인 JPEG2000 코어 인코더의 블록 구성도로서, 도 2를 참조하여 먼저 본 발명에서 구현하고자하는 JPEG2000에 대해 설명하기로 한다. 도 2를 참조하여 JPEG2000의 인코딩 과정을 살펴보면, 입력된 원본 영상에 대하여 전처리부(202)에서 타일링(tiling)과 같은 전처리 과정을 거치게 되고, 컬러변환부(204)에서 RGB 영상을 YCbCr로 변환하며, 그후 DWT(206)에서 입력 신호에 대한 DWT 변환이 수행되고, 이후 양자화부(207)에서의 양자화(quantization) 과정을 거치고 'Tier-1'(208), 'Tier-2'(209)에서 엔트로피 코딩 과정을 거치게 되면 최종적으로 JPEG2000의 코드 영상(code image)을 얻을 수 있다. 이때 레이트 제어부(210)를 통해 전송 레이트 조절이 이루어진다. 디코딩 과정은 단순히 도 2에 도시된 인코딩 과정의 역순으로 진행하여 원래의 원본 영상을 얻을 수 있다.
- <23> [화소 셰이더(Fragment shader, pixel shader)의 활용]
- <24> 상기한 JPEG2000의 DWT 모듈을 GPU에서 구현함에 있어서, 본 발명은 특히 화소 셰이더를 활용하여 이를 구현한다. 이러한 화소 셰이더에 대해 보다 상세히 설명하면, 예를 들어 OpenGL에서 렌더링의 결과를 화면에 출력하는

과정 중, 화소 셰이더는 프레임 버퍼의 각 화소의 최종 색을 결정하는 일을 수행한다. 이와 같은 과정은 프레임 버퍼의 각각의 픽셀 단위로 이루어지고, 이러한 구조는 화소 셰이더를 이용하여 영상 처리를 수행하는 데에 구조적인 장점을 제공한다. 영상의 화질 개선이나 경계 검출 등과 같은 일반적인 영상 처리의 경우, 각각의 화소에 대하여 독립적으로 연산이 이루어지고 결과의 출력 또한 각각의 화소로 저장되기 때문에 화소 단위로 연산이 이루어지는 화소 셰이더의 구조를 이용하여 영상 처리를 수행하는 것이 매우 적합하다. 이때, 처리할 영상을 텍스처화하여 GPU의 텍스처 메모리로 읽어들이고 화소 셰이더는 이를 입력으로 하여 출력 화소를 생성하게 되며, 이 과정에서 사용자가 원하는 영상 처리 알고리즘이 셰이딩 언어로 구현되어 수행된다.

<25> 이러한 화소 셰이더를 이용하여 영상 처리를 수행하기 위한 방법들은 크게 온스크린 렌더링(on-screen rendering)과 오프스크린 렌더링(off-screen rendering)의 두 가지 방법으로 나눌 수 있다. 온스크린 렌더링은 렌더링의 결과가 프레임 버퍼에 출력되어 실제 화면에 표현되는 방법이고, 오프스크린 렌더링은 렌더링의 결과를 내부적인 버퍼에 저장함으로써 렌더링 결과가 화면에 표현되지 않고 내부적으로 렌더링이 이루어지는 것을 의미한다.

<26> 온스크린 렌더링을 이용하여 영상 처리를 수행할 경우, 결과 영상이 프레임 버퍼에 저장되기 때문에 프레임 버퍼의 내용을 다시 시스템 메모리로 복사하는 과정에서 많은 시간이 소요되고, 여러 단계의 영상 처리가 이루어질 경우 이러한 과정들이 반복적으로 일어나기 때문에 그래픽 하드웨어를 이용한 영상처리의 장점을 잃어버리게 된다.

<27> 이러한 문제점을 보완하기 위하여 오프스크린 렌더링을 이용한 영상 처리가 대세를 이루게 되었고, 이러한 오프스크린 렌더링은 렌더링의 결과를 텍스처로 직접 저장하기 때문에 렌더투텍스처(render-to-texture: RTT)로도 알려져 있다. GPU의 프로그래밍 가능한 아키텍처와 이를 지원하는 셰이딩 언어가 점차 발전함에 따라, 다양한 종류의 렌더투텍스처 방법들이 소개되어왔다. 이에 관해 설명하면 다음과 같다.

<28> 먼저, 렌더링의 결과를 텍스처로 저장하기 위한 방법으로 'CopyTexSubImage' 함수를 이용하여 프레임 버퍼의 내용을 텍스처로 복사하는 방법이 있다. 그러나 이러한 방법의 경우 단순히 프레임 버퍼의 내용을 텍스처로 복사하는 과정이기 때문에 온스크린 렌더링에서 발생하는 문제점을 그대로 갖게 된다.

<29> 텍스처로 복사하는 기법의 단점을 극복하면서 오프스크린 렌더링을 수행하기 위하여 'pbuffer'라는 기법이 등장하였다. 'pbuffer'의 경우 현재의 프레임 버퍼와 문맥(context)이 동일한 버퍼를 내부적으로 생성하고 렌더링의 결과를 pbuffer에 출력하는 방식이다. 그러나 pbuffer와 프레임 버퍼 사이에 내부적으로 텍스처나 디스플레이 리스트(display list)를 공유할 수 없고, 렌더링의 타겟을 바꾸는 과정에서 문맥 스위칭(context switching)에 많은 시간이 소요되는 문제점을 갖고 있다.

<30> 이러한 문제점을 해결하기 위하여 새로이 프레임버퍼 객체(framebuffer object: FBO)라는 방법이 등장하게 된다. 프레임버퍼 객체의 경우 하나의 문맥(context)만이 존재하기 때문에 pbuffer에 비하여 문맥 스위치에 소요되는 시간이 더 짧고, 각각의 프레임버퍼에서 텍스처가 공유되기 때문에 메모리의 사용이 줄어들 뿐만 아니라, 렌더링의 결과를 다음 렌더링의 입력으로 그대로 사용할 수 있기 때문에 영상 처리를 하는 데에 많은 장점을 제공한다.

<31> 이와 같은 화소 셰이더를 활용하여, 본 발명에서는 JPEG2000에서 영상 압축을 위한 DWT를 구현하게 되는데, JPEG2000에서는 영상을 압축하기 위하여 DWT를 수행할 경우에, 한 단계의 DWT가 수행이 될 때마다 데이터의 범위는 두 배로 증가하게 되고 이러한 단계를 여러 번 반복한 후에 최종적으로 양자화 과정을 거치게 된다. GPU를 이용하여 DWT를 구현할 경우 한번의 일련적인 렌더링 과정에 의하여 한 단계의 DWT가 수행이 되고 출력 프레임 버퍼의 자료 정밀도에 의해 결국 매 단계마다 양자화 과정이 수행되어 오차의 크기가 증가된 결과를 얻게 될 수 있다. 이러한 중간 단계에서의 양자화 과정을 제거하기 위하여 본 발명에서는 입력과 출력 영상을 32비트 부동 소수점(floating point)의 텍스처로 지정하여 DWT를 수행한다. 이와 같이 텍스처를 32비트 부동 소수점 형식으로 지정함으로써 출력의 자료 정밀도가 증가하게 되고, 텍스처를 일반적인 영상이 아닌 데이터의 형태로 이용하여 GPU에서 처리가 가능해 진다. 32비트 부동 소수점 텍스처를 사용하기 위해서 'ARB_texture_rectangle'과 'ARB_texture_float' 확장을 사용한다.

<32> 도 3은 본 발명에 따른 GPU와 화소 셰이더를 이용한 영상처리 프레임워크의 기본 구조도이다. 도 3을 참조하면, 본 발명에 따른 GPU에서 화소 셰이더(30)는 먼저 입력으로 32비트 부동 소수점 데이터인 텍스처 데이터(302)를 제공받아 이를 처리하게 되며, 이때 화소 셰이더 내부에는 각각의 해당 설정된 기능을 수행하여 그 출력으로 텍스처 데이터를 출력하는 적어도 하나 이상의 FBO(333)를 구비한다.

<33> 즉, 영상처리의 대상이 되는 입력 영상은 텍스처 데이터로 비디오 메모리로 로딩되며, 화소 셰이더에서는 영상 처리의 특정 알고리즘이 입력 영상에 대해 수행되어 그 결과를 텍스처 데이터로서 프레임버퍼 객체(FBO)로 다시 출력한다. 상기에서와 같이 이 과정은 오프스크린 렌더링 또는 렌더투텍스처라 하며, 화소 셰이더의 처리 결과를 기본 출력 대상인 프레임 버퍼로 전달하는 것이 아니고 텍스처 메모리로 보내어 재사용이 가능하도록 하는 피드백(feedback) 구조이다. 어플리케이션에서는 화면 전체와 일치하는 사각형의 렌더링을 수행하고 FBO의 내용을 텍스처로 바인딩하게 되면, 결국 화소 셰이더가 수행한 영상처리의 결과가 뷰포트(viewport)로 렌더링 되는 것이다. 또한, 입력과 출력 텍스처의 형식을 32비트 실수 형태로 지정하여 영상처리 과정에서 생성되는 부동 소수점 데이터를 표현할 수 있다. 그리고, FBO의 유용한 특징 중 하나로서 입력 텍스처와 출력 텍스처를 동일하게 지정할 수 있으므로, 부가적인 과정 없이 출력 결과를 그대로 입력으로 이용함으로써 수행시간이 단축될 수 있으며 메모리 사용량은 감소될 수 있다.

<34> [이산 웨이블릿 변환(Discrete Wavelet Transform: DWT)]

<35> 이와 같이, 구현된 화소 셰이더를 이용하여 본 발명에 따른 DWT를 구현하기 위하여는 DWT 구현을 위해 일반적으로 잘 알려져 있는 두 가지 방법인 컨볼루션(convolution) 기반의 DWT와 리프팅(lifting) 기반의 DWT를 구현하여 성능의 비교를 수행한다. 컨볼루션 기반의 DWT는 일반적인 신호처리에서의 컨볼루션 방법과 마찬가지로, 이미 정해져 있는 로우 패스 필터(low pass filter)와 하이패스 필터(high pass filter)의 값을 입력 신호에 곱하여 입력 영상에 대한 저주파 성분과 고주파 성분을 구분하게 된다. 하기 수학적 식 1과 2는 컨볼루션 기반의 DWT의 수행 방식을 나타낸다. 하기 수학적 식 1과 수학적 식 2에서 입력신호 $x(n)$ 에 대하여 로우패스 필터 $h(i)$ 와 하이패스 필터 $g(i)$ 를 이용하여 입력 신호의 저주파 성분 $y_L(n)$ 과 고주파 성분 $y_H(n)$ 을 구하는 과정을 보여준다.

수학적 식 1

<36>
$$y_L(n) = \sum_{i=0}^{r_L-1} h(i)x(2n-i)$$

수학적 식 2

<37>
$$y_H(n) = \sum_{i=0}^{r_H-1} g(i)x(2n-i)$$

<38> 본 발명에서는 DWT를 수행하기 위한 또 다른 방법으로 리프팅 기반의 DWT를 구현한다(참조문헌[5]). 리프팅 기반의 DWT는 컨볼루션 기반의 DWT보다 연산의 수를 줄임으로써 좀더 빠른 수행시간을 제공한다. 도 4는 간략화된 리프팅 기반의 DWT 과정을 보여준다.

<39> 비가역 변환(Irreversible transform)을 의미하는 9/7 탭(tab) DWT의 다위상 매트릭스(polyphase matrix)는 하기 수학적 식 3과 같다.

수학적 식 3

<40>
$$\tilde{P}(z) = \begin{bmatrix} 1 & \alpha(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \beta(1+z) & 1 \end{bmatrix} \begin{bmatrix} 1 & \gamma(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \delta(1+z) & 1 \end{bmatrix} \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix}$$

<41> 상기 수학적 식 3의 연산을 하기 수학적 식 4와 같은 6단계의 과정으로 연산을 수행할 수 있다.

수학식 4

$$(S1) \ Y_{2n+1} = X_{2n+1} + \alpha \times (X_{2n} + X_{2n+2}) \quad \alpha = -1.586134342$$

$$(S2) \ Y_{2n} = X_{2n} + \beta \times (Y_{2n-1} + Y_{2n+1}) \quad \beta = -0.052980118$$

$$(S3) \ Y_{2n+1} = Y_{2n+1} + \gamma \times (Y_{2n} + Y_{2n+2}) \quad \gamma = 0.882911075$$

$$(S4) \ Y_{2n} = Y_{2n} + \delta \times (Y_{2n-1} + Y_{2n+1}) \quad \delta = 0.443506852$$

$$(S5) \ Y_{2n+1} = -K \times Y_{2n+1} \quad K = 1.230174105$$

$$(S6) \ Y_{2n} = Y_{2n} / K$$

<42>

<43>

하기 표 1에서는 컨볼루션 기반의 DWT와 리프팅 기반의 DWT 사이의 연산수에 대한 관계를 보여주고 있다. 표 1에서 보는 바와 같이, 리프팅 기반의 DWT가 컨볼루션 기반의 DWT에 비하여 1/3 정도의 연산수가 감소하는 것을 확인할 수 있다.

표 1

Filter	Convolution		Lifting Scheme	
	Multiplications	Additions	Multiplications	Additions
5/3	4	6	2	4
9/7	9	14	6	8

<44>

<45>

[GPU에서의 DWT 구현]

<46>

본 발명에 따른 GPU를 이용하여 컨볼루션 기반의 DWT와 리프팅 기반의 DWT 구현 과정을 살펴보았을 때 입력 영상은 모두 본 발명의 특징에 따라 32비트 부동 소수점 형식의 텍스처로 지정이 된다.

<47>

먼저 컨볼루션 기반의 DWT의 경우, 가로방향(horizontal)과 세로방향(vertical)의 DWT를 위하여 총 두번의 렌더링이 수행이 된다. 이때 가로방향 및 세로방향의 DWT를 위해 각각 하나의 FBO가 미리 마련되며, 이때 입력 텍스처를 이전 결과를 텍스처로 지정한 후 렌더링이 수행된다. 이러한 렌더링 과정 중 각각의 픽셀에 대해서 현재 픽셀을 기준으로 인접한 픽셀로의 접근을 통한 연산이 이루어진다.

<48>

리프팅 기반의 DWT의 경우에도 가로방향 및 세로방향의 DWT를 위해 각각 하나의 FBO가 미리 마련되어 두 개의 FBO를 이용하여 렌더링을 수행한다. 이때 리프팅 과정에서 하나의 FBO를 이용하여 자신의 입력 텍스처를 자신의 출력 텍스처로 지정하여 FBO의 장점을 활용한다. 이러한 리프팅 기반의 DWT의 경우에는 상기 수학식 4의 여섯 단계의 과정중 마지막 세 단계를 한번의 렌더링으로 수행함으로써 가로방향과 세로방향에 대하여 각각 네번씩의 렌더링이 이루어지게 된다. 이때 네번의 렌더링 과정에서 출력 텍스처와 입력 텍스처를 동일하게 지정함으로써 메모리 이동에 소요되는 시간을 감소시키도록 한다.

<49>

[실험 결과]

<50>

본 발명에서 제안한 GPU를 이용한 영상처리 프레임워크를 이용하여 JPEG2000의 주요 모듈을 구현하였으며, GPU와 CPU 상에서의 수행을 통하여 수행 속도 차이를 측정하였다. 본 측정 실험은 256MB의 비디오 메모리를 갖고 있는 NVIDIA GeForce 7800GTX(G70) GPU를 장착한 그래픽카드와 AMD Athlon64 2.0GHz의 CPU에 2GB의 메인 메모리를 장착한 PC에서 수행되었다.

<51>

도 5a, 5b와 하기 표 2에서 512x512 해상도의 영상에 대하여 렌더링텍스처가 아닌 'CopyTexSubImage'를 이용하여 DWT를 수행한 결과(도 5a, 5b에서는 'GPU Conv'와 'GPU Lift'로 표시함)와, 본 발명에 따른 FBO(도 5a, 5b에서는 'GPU Conv FBO'와 'GPU Lift FBO'로 표시함)를 이용하여 DWT를 수행한 결과, 그리고 CPU를 이용하여 DWT를 수행한 결과(도 5a에는 'CPU Conv'와 'CPU Lift'로 표시함)를 그래프로 보여주고 있다. 도 5a, 5b 및 표 2에서 보여주는 실험 결과는 OpenGL의 초기화 및 셰이더의 컴파일 시간을 제외하고 순수 연산에 소요되는 시간을 비교하였다.

표 2

512x512	CopyTexSubImage		FBO		CPU	
	Convolution	Lifting	Convolution	Lifting	Convolution	Lifting
ENCODE	0.003238	0.007891	0.0985	0.000519	0.37419	0.3646
Ratio (/FBO)	0.0329	15.207	1	1	3.8	702.548

<52>

<53>

먼저, 도 5a에서 나타나듯이, CopyTexSubImage가 CPU와 비교해서 1/40 ~ 1/100 정도의 수행시간을 나타내는 것을 알 수 있다. 또한 도 5b에서 나타나듯이, CopyTexSubImage와 본 발명에 따른 FBO를 비교하였을 때, 컨볼루션 기반의 DWT의 경우 시간이 증가하지만 리프팅 기반의 DWT의 경우 1/15 정도 시간이 감소하는 것을 알 수 있다.

<54>

도 5a에서 CopyTexSubImage에 대하여 컨볼루션 기반의 DWT와 리프팅 기반의 DWT를 비교하였을 때, 리프팅 기반의 변환 과정이 컨볼루션 기반에 비하여 연산수가 줄어들어 수행 시간이 더 짧아진다고 알려져 있는 것과 반대로, 오히려 컨볼루션 기반의 DWT가 더 빨리 수행되는 것을 확인할 수 있다. 이는 GPU 기반의 변환에서의 렌더링 횟수를 들 수 있다. 컨볼루션 기반의 DWT가 2차원 영상에 대한 가로방향과 세로방향으로 두 번의 렌더링이 이루어지는 것에 비하여, 리프팅 기반의 DWT 경우 상기 수학식 4에서 보는 바와 같이 한 방향의 변환에 4번의 렌더링이 필요하기 때문에 총 8번의 렌더링이 이루어지게 된다. 프레임 버퍼의 내용을 텍스처로 다시 사용하기 위하여 매 렌더링 마다 'copy-to-texture' 연산이 이루어지기 때문에, 리프팅 기반의 DWT가 비록 연산수는 감소하더라도 렌더링 횟수의 증가로 인하여 오히려 컨볼루션 기반의 DWT보다 느려지는 것을 알 수 있다.

<55>

한편, 본 발명에 따른 FBO를 사용하였을 경우 리프팅 기반의 DWT가 컨볼루션 기반의 DWT와 비교하여 1/200 정도의 시간 감소를 보여주고 있다. 이러한 결과는 기존의 CopyTexSubImage의 경우 프레임 버퍼의 출력 결과를 다시 텍스처로 생성하는데 소요되는 시간이 FBO를 사용함으로써 제거되어 리프팅의 경우 소요 시간이 감소하고, 컨볼루션의 경우 입력과 출력 텍스처의 형식이 바뀔때 따라 비교적 텍스처의 접근이 많은 컨볼루션의 경우에 수행 시간이 대폭 증가한 것으로 해석된다.

<56>

상기와 같이 본 발명의 일 실시예에 따른 GPU 기반의 JPEG2000 DWT 모듈이 구현될 수 있으며, 한편 상기한 본 발명의 설명에서는 구체적인 실시예에 관해 설명하였으나 여러 가지 변형이 본 발명의 범위를 벗어나지 않고 실시될 수 있다. 따라서 본 발명의 범위는 설명된 실시예에 의하여 정할 것이 아니고 청구범위와 청구범위의 균등한 것에 의하여 정하여져야 할 것이다.

발명의 효과

<57>

상기한 바와 같이, 본 발명은 고급 GPU의 범용 활용을 통해 JPEG2000 정지 영상의 고속 압축 알고리즘, 특히 JPEG2000 알고리즘의 핵심 모듈이 되는 DWT(Discrete Wavelet Transform) 알고리즘의 구현을 가능하도록 할 수 있다.

<58>

[참조문헌]

<59>

1. [JPEG2000 Standard] Information Technology . JPEG2000 Image Coding System, ISO/IEC International Standard 15444-1, ITU Recommendation T.800, 2000.

<60>

2. [JPEG2000 Overview] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 Still Image Compression Standard," IEEE Signal Processing Magazine, vol. 18, no. 5, pp. 36-58, September 2001.

<61>

3. [JPEG2000 Overview] M. Rabbani and R. Joshi, "An overview of the JPEG2000 still image compression standard," Signal Processing: Image Communication, vol. 17, no. 1, pp. 3-48, January 2002.

<62>

4. [JPEG2000 Textbook] T. Acharya and P. Tsai, JPEG2000 Standard for Image Compression, Wiley-Interscience, 2005.

<63>

5. [Lifting-based DWT] 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.

<64>

6. [EBCOT] D. Taubman, "High performance scalable image compression," IEEE Transaction on Image

Processing, vol. 9, no. 7, pp. 1158-1170, July 2000.

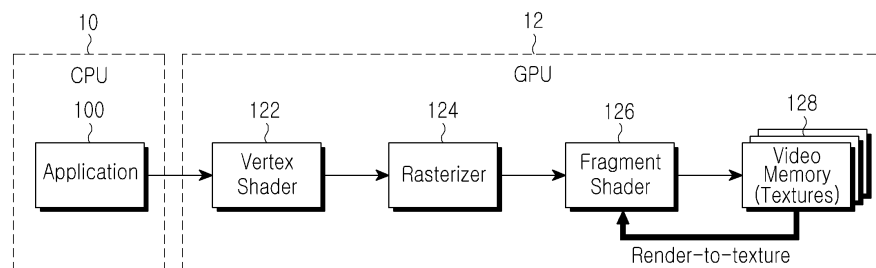
- <65> 7. [GPGPU] R. Fernando (editor), GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics, Addison-Wesley, 2004.
- <66> 8. [GPGPU] M. Pharr (editor), GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation, Addison-Wesley, 2005.
- <67> 9. [NVIDIA] <http://developer.nvidia.com/page/documentation.html>
- <68> 10. [ATI] <http://ati.amd.com/developer/index.html>
- <69> 11. [OpenGL] <http://www.opengl.org>
- <70> 12. [OpenGL Textbook] D. Shreiner, M. Woo, J. Neider, and T. Davis, OpenGL Programming Guide, Fifth Edition, Addison-Wesley, 2006.
- <71> 13. [GLSL Textbook] R. Rost, OpenGL Shading Language, Second Edition, Addison-Wesley, 2006.
- <72> 14. [CVPR Tutorial] E. Larsen, Using the Graphics Processing Unit for Computer Vision, Tutorial in IEEE Conference on Computer Vision and Pattern Recognition, June 2006.
- <73> 15. [JasPer] M. D. Adams and F. Kossentini, "JasPer: A software-based JPEG-2000 codec implementation," Proc. IEEE International Conference on Image Processing, September 2000. (Also, refer to the JasPer homepage at <http://www.ece.uvic.ca/~mdadams/jasper/>)

도면의 간단한 설명

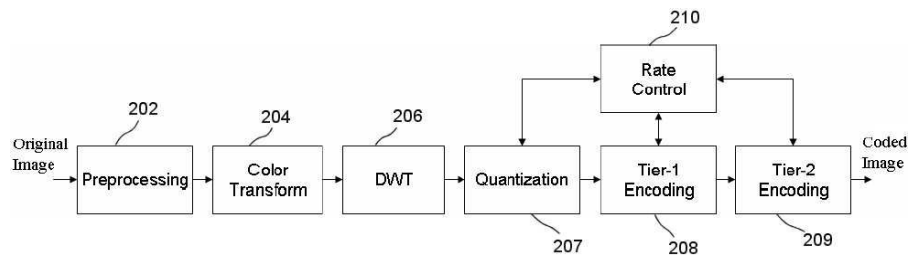
- <1> 도 1은 본 발명이 적용되는 프로그래머블(programmable) 파이프라인의 GPU 구조도
- <2> 도 2는 일반적인 JPEG2000 코어 인코더의 블록 구성도
- <3> 도 3은 본 발명에 따른 GPU와 화소 셰이더를 이용한 영상처리 프레임워크의 기본 구조도
- <4> 도 4는 일반적인 리프팅 기반의 DWT 방식을 간략히 나타낸 도면
- <5> 도 5a, 5b는 본 발명에 따른 GPU 기반의 DWT의 성능 시험 결과 그래프

도면

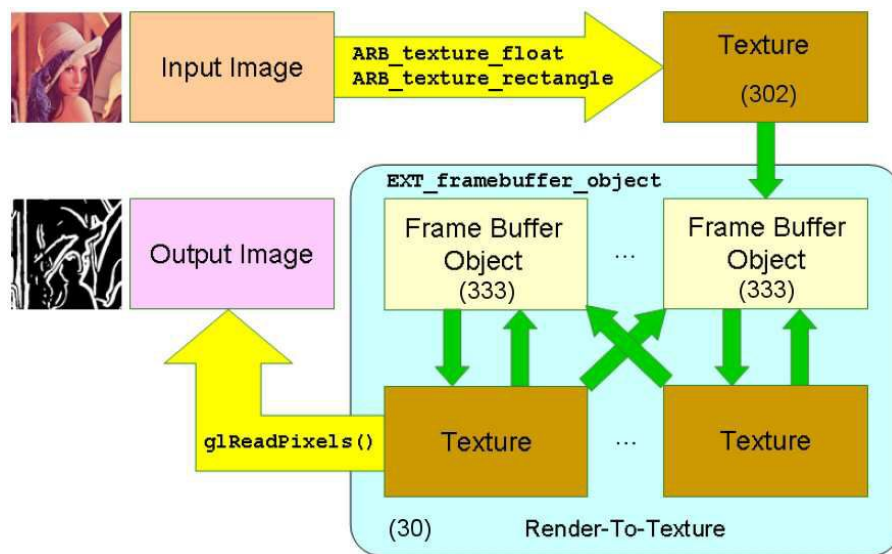
도면1



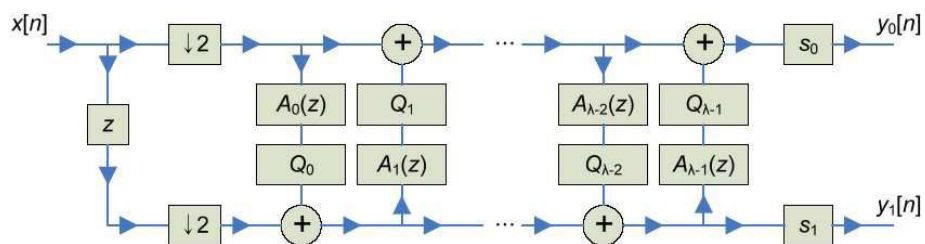
도면2



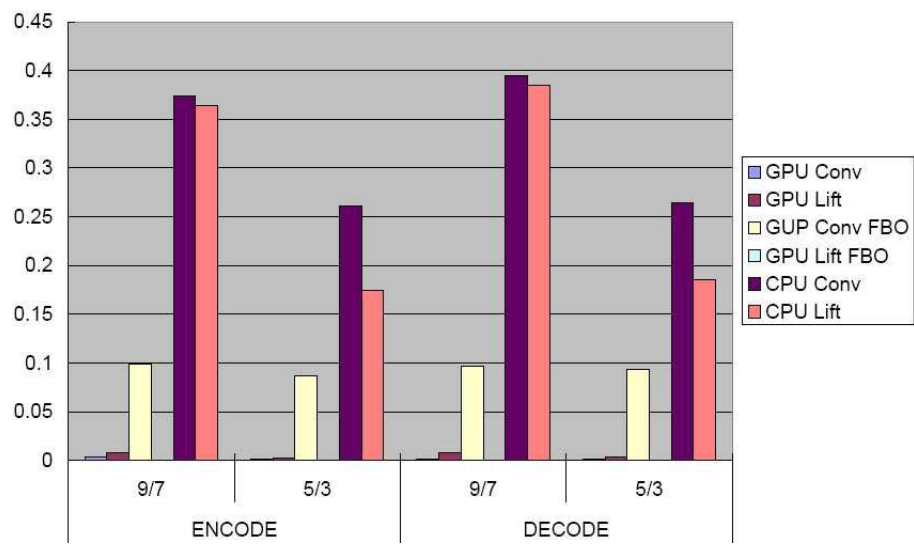
도면3



도면4



도면5a



도면5b

