이동통신 단말기를 위한 재구성 가능한 구조의 H.264 인코더의 움직임 추정기와 3차원 그래픽 렌더링 가속기 설계

(Reconfigurable Architecture Design for H.264 Motion Estimation and 3D Graphics Rendering of Mobile Applications)

박 정 애 † 윤 미 선 † 신 현 철 ††
(Jungae Park) (Misun Yoon) (Hyunchul Shin)

요 약 휴대용 단말기에서의 동영상 및 3차원 영상을 처리하는 것이 일반화 되면서, H.264 및 3차원 그래픽 가속기 데이타를 처리하기 위한 연산량이 크게 증가하고 있다. 본 연구에서는 H.264 인코더의 움직임 추정기 및 디코더의 움직임 보상기와 3차원 그래픽 렌더링 가속기를 재구성 가능하도록 설계 하였다. 움직임 추정기는 효율적인 데이타 스캐닝 방법과 DAU, FDVS 알고리즘을 사용하여, JM8.2에 제시된 다중 프레임 움직임 추정보다 연산량을 평균적으로 70%이상 감소시키면서 화질 열화가 없도록 하였다. 3차원 그래픽 렌더링 가속기는 중심선 트래버셜 알고리즘을 사용하여 병렬 처리 하도록 함으로써 처리량을 증가시켰다. 움직임 추정기와 3차원 렌더링 가속기의 메모리를 재구성 가능한 구조로 설계하여, 2.4Mbits (47%)의 메모리를 공유하였으며, 메모리를 8개의 블록으로 분산시켜 사용되지 않는 부분의 전력 소모를 최소화 할 수 있도록 하였다. 또한, 움직임 보상기와 3차원 렌더링 가속기의 픽셀 프로세서를 공유하여 약 7%의 하드웨어면적을 감소 시켰다.

키워드 : 움직임 추정기, 움직임 보상기, 렌더링 가속기, H.264, 3차원 그래픽 가속기, 재구성 구조

Abstract Mobile communication devices such as PDAs, cellular phones, etc., need to perform several kinds of computation-intensive functions including H.264 encoding/decoding and 3D graphics processing. In this paper, new reconfigurable architecture is described, which can perform either motion estimation for H.264 or rendering for 3D graphics. The proposed motion estimation techniques use new efficient SAD computation ordering, DAU, and FDVS algorithms. The new approach can reduce the computation by 70% on the average than that of JM 8.2, without affecting the quality. In 3D rendering, midline traversal algorithm is used for parallel processing to increase throughput. Memories are partitioned into 8 blocks so that 2.4Mbits (47%) of memory is shared and selective power shutdown is possible during motion estimation and 3D graphics rendering. Processing elements are also shared to further reduce the chip area by 7%.

Key words : Motion Estimation, Motion Compensation, 3D Rendering Accelerator, Reconfigurable Architecture

1. 서 론

디지털 멀티미디어 기술의 발달로 방송, 통신, 컴퓨터, 가전 등 다양한 분야에서 발전이 이루어지고 있다. 멀티미디어 데이타 실현을 위해서는 실시간 처리가 필수적이다. 이에 따라 ITU-T에서는 H.264 등의 국제 규격들을 제정하였다[1]. 그중 H.264는 효율성과 신뢰성을 강조하고 있으며, MPEG-2, MPEG-4 (Part 2) 등 기존 동영상 압축 표준에 비해 1.5배 이상의 압축 성능을 갖는다[2].

이동통신 단말기에는 멀티미디어와 엔터테인먼트 분야의 대중화에 따른 사용자들의 요구를 충족시켜 주는

· 본 연구는 한국과학재단 특정기초연구 (R012004000102680) 지원으로 수행되었음.
† 비 회 원 : 한양대학교 전자전기제어계측공학과
japark@digital.hanyang.ac.kr
msyoon@digital.hanyang.ac.kr
†† 종신회원 : 한양대학교 전자컴퓨터공학부 교수
shin@hanyang.ac.kr
논문접수 : 2005년 10월 4일
심사완료 : 2006년 10월 31일

다양한 기능들이 필요하게 되었다. 게임의 현실감 있는 캐릭터 움직임, 3차원 그래픽을 사용한 자연스러운 영상으로 구성된 영화, 그 외에 의료, 군사, 가상현실까지 3차원 영상은 그 범위를 확대해가고 있다.

본 연구에서는 H.264에서 많은 연산량을 차지하는 인코더의 움직임 추정기(Motion Estimation, ME) 및 디코더의 움직임 보상기(Motion Compensation, MC)와 3차원 그래픽 렌더링 가속기(3D Rendering Accelerator, 3D RA)를 이동통신 단말기 등에서 사용할 수 있도록 재구성 가능한 구조로 설계 하였다. 움직임 추정기는 효율적인 새로운 데이타 스캐닝 방법을 개발하여 기존의 Disable Approximation Unit(DAU)[3] 보다 연산량을 30% 감소시켰으며, Forward Dominant Vector Selection Method(FDVS)[4]를 통합 이용하여 JM8.2에 제시된 다중 프레임 움직임 추정 알고리즘 보다 연산량이 70% 이상 감소하도록 하였다.

3차원 그래픽 렌더링 가속기는 중심선 트래버셜 알고리즘을 사용하고, 픽셀 프로세서(Pixel Processor) 2개를 병렬 처리 하도록 함으로써 처리량을 증가시켰다. 또한, H.264에서 가장 연산량이 많은 움직임 추정기 및 움직임 보상기와 병목 현상이 가장 많이 발생하는 3차원 그래픽 렌더링 가속기 처리기까지 공유하도록 설계하였다. H.264 움직임 추정기와 3차원 렌더링 가속기의 메모리를 재구성 가능한 구조로 설계하여 2.4Mbits (47%)의 메모리를 공유하도록 하였다. 또한, 움직임 보상기의 픽셀 프로세서와 3차원 렌더링 가속기에서 픽셀 값을 계산하는 픽셀 프로세서를 공유하여 약 7%의 하드웨어 면적을 감소 시켰다.

2절에서는 H.264 움직임 추정 및 움직임 보상기와 3차원 렌더링 가속기에 대한 기존 연구에 대해 기술한다. 3절에서는 제안한 H.264 인코더의 움직임 추정기 알고리즘과 연산 감소를 위한 효율적인 데이타 스캔 방법을 기술하고, 4절에서는 제안한 H.264 디코더의 움직임 보상기에 대해 설명한다. 5절에서는 3차원 렌더링 가속기의 데이타 처리량을 증가시키기 위해 제안한 방법들에 대해 설명한다. 6절에서는 제안한 재구성 가능한 구조를 기술하고, 7절에서 결론을 맺는다.

2. 기존 연구

2.1 움직임 추정기 및 움직임 보상기

움직임 추정은 H.264 인코더에서 동영상의 시간적인 중복성을 제거하기 위한 효율적인 방법이다. 현재 프레임 블록과 참조 프레임의 블록에서 Sum of Absolute Difference(SAD) 연산을 통해 가장 비슷한 블록을 찾는 방법을 움직임 추정이라고 하며, 움직인 변위를 움직임 벡터라고 한다.

움직임 보상은 H.264 디코더에서 움직임 벡터로 예측 블록의 값을 읽어 오고, 이를 예측 오차 값과 더해서 화면을 복원하는 과정이다.

식 (1)에서 움직임 추정과 보상의 블록 사이즈는 $N \times N$ 샘플이고, c_{ij}와 r_{ij}는 현재 프레임과 참조 프레임의 (i,j) 위치의 휘도 값을 의미한다.

$$SAD = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} |c_{ij} - r_{ij}| \quad (1)$$

움직임 추정 및 움직임 보상 방법은 데이타 흐름의 규칙성, 계산의 복잡도, 하드웨어의 구현 등에서 용이한 블록 정합 알고리즘(Block Matching Algorithm)이 많이 사용되고 있다.

2.1.1 가변블록 움직임 추정

H.264는 움직임 벡터를 찾기 위해 다양한 크기의 블록을 허용한다. 이는 이동뿐만 아니라 회전 및 다양한 국부적 움직임 추정에 적합하며, 압축 효율을 높일 수 있다는 장점이 있다. 표준안에 정의된 가변블록 크기는 4×4, 4×8, 8×4, 8×8, 8×16, 16×8, 16×16 일곱 가지로, 움직임 벡터를 41개의 경우로 나타낼 수 있으며, Macroblock mode인 16×16, 16×8, 8×16, 8×8과 Sub-Macroblock mode인 8×8, 8×4, 4×8, 4×4블록 등으로 구성된다.

2.1.2 다중 프레임 움직임 추정

H.264에서는 다른 코덱보다 영상 압축을 강화하였으며, 5 프레임까지의 참조 프레임을 지원하여 보다 효율적인 움직임 추정을 할 수 있도록 하였다. 예를 들어, 그림 1에서는 현재 프레임 $F_{(n)}$이 직전프레임인 $F_{(n-1)}$보다 $F_{(n-2)}$프레임과 더욱 유사한 것을 볼 수 있다. 따라서 반복되는 영상의 압축 효율을 높이기위해 다중 프레임을 사용하면, 비트율을 낮출 수 있다.

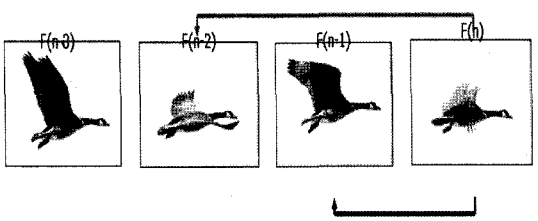

그림 1 다중 프레임 움직임 추정

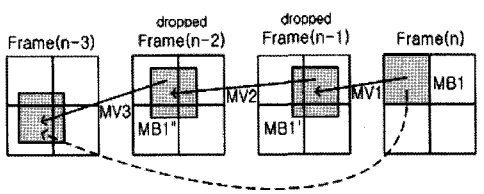

그림 2 FDVS 알고리즘

FDVS 방법[4]은 움직임 벡터를 구할 때 여러 개의 참조 프레임 중에서 현재 매크로 블록과 가장 유사한 매크로 블록의 움직임 벡터를 선택하는 알고리즘이다. 그림 2에서 보이는 것과 같이 (n-1)번째 프레임과 (n-2)번째 프레임, (n-3)번째 프레임의 움직임 벡터를 각각 MV1, MV2, MV3이라 하면 (n)번째 프레임과 (n-3)번째 프레임에서 움직임 벡터는 MV1+MV2+MV3으로 구해질 수 있다. 이 방법을 사용하여 (n-3)번째 프레임의 움직임 벡터는 다시 추정하는 과정을 거치지 않고 효율적으로 구할 수 있다.

2.2 3차원 렌더링 가속기

3차원 그래픽 가속기는 영상 데이타의 크기 및 위치 변화와 빛에 대한 변화를 처리해 주는 기하학 처리기(Geometry Processor)와 영상 데이타에 색상 및 텍스쳐를 입히는 렌더링 처리기(Rendering Processor)로 나누어진다. 최근 고성능을 요구하는 3차원 그래픽 가속기에서는 Graphics Processing Unit(GPU)과 같은 기하학 연산 처리기를 두어 Central Processing Unit (CPU)에 대한 부담을 덜고 빠르게 영상 데이타를 처리한다[5].

2.2.1 렌더링 처리 과정

렌더링 처리기 과정은 우선 세 정점을 Y축을 기준으로 가장 큰 점부터 Vmax, Vmid, Vmin을 설정한다. 먼저 각 변의 기울기 및 색상 정보의 증가분을 구하는 폴리곤 처리(Polygon Processing)를 한다.

폴리곤 처리 과정 후 각 변을 이루는 변 처리(Edge Processing)를 한다. 변 처리는 주사선마다 한 번씩 수행되며, 폴리곤 처리에서 구한 데이타를 가지고 각 주사선과 양 변이 교차하는 픽셀에 대한 위치와 색상 정보를 구한다.

마지막으로 각 주사선에 대한 픽셀들의 색상 정보를 구하는 주사 변환 처리(Scan Processing)를 함으로써 삼각형 내부의 모든 픽셀에 대한 색상 정보를 구하게 된다. 주사 변환 처리는 하나의 주사선에 대해 X축으로 이동하면서, 색상 정보의 증가분을 더하여, 각 픽셀의 색상 정보를 구한다.

2.2.2 중심선 트래버셜 알고리즘

주사선 알고리즘은 가장 위에 있는 주사선부터 시작해서 주사선상의 모든 픽셀에 대한 정보를 구하면서 마지막 주사선까지 이 과정을 반복한다. 나눗셈 연산이 필요하며, 여러 개의 픽셀들의 정보를 병렬로 구하기 어렵다는 단점이 있다.

그림 3은 중심선 트래버셜 방법을 보여준다. 중심선 트래버셜 방법은 주사 변환 처리의 병렬 처리를 가능하게 한다. Vmax를 시작으로 수직 방향으로 내려가면서 모든 주사선을 차례대로 트래버셜하는 방법이다. 현재의

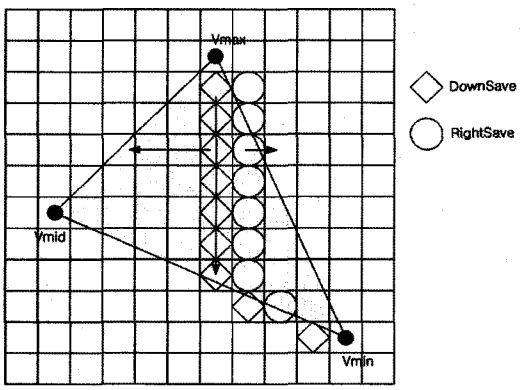

그림 3 중심선 트래버셜 알고리즘

정점의 주사선을 트래버셜 할 때, 오른쪽 픽셀이 유효한지 조사하여 유효하면 정보를 RightSave에 저장하고, 주사선의 왼쪽으로 트래버셜을 진행한다. 왼쪽의 유효 픽셀에 대한 트래버셜 진행이 끝나면, RightSave에 저장된 정보를 불러와 오른쪽으로 트래버셜을 진행한다. 수평방향과 같은 방법으로 수직방향에 대해서도 DownSave에 저장한 후 진행한다.

2.3. 재구성 가능한 구조

[6]에서 제시한 구조는 이동통신 단말기에 사용할 수 있도록 MPEG-4 디코더 중 움직임 보상기와 3차원 그래픽 렌더링 가속기를 하나의 칩으로 설계 하였다. 움직임 보상기는 8개의 픽셀 ALU를 병렬 처리하여 Simple Profile을 만족하며 QCIF 크기의 영상을 초당 15프레임 처리를 할 수 있도록 하였다. 3차원 그래픽 렌더링 가속기는 8개의 변 처리기와 64개의 주사 변환처리기를 사용하여 고라우드 세이딩을 적용하였으며 Z-buffering을 가능하도록 설계 되었다. 그림 4의 공유 부분은 전 처리 프로세서인 ARM9과 처리된 데이타를 처리하기 전에 저장하는 버퍼만이다. 움직임 보상기, 3차원 그래픽 렌더링 가속기와 메모리는 각각 독립적인 하드웨어로 구현하였다. 그러나 이동통신 단말기에서 H.264와 3차원 그래픽 렌더링 가속기를 각각 구현하면면적을 많이 차지한다는 단점이 있다. H.264 움직임 추정기에서 다중 프레임을 지원하도록 하거나, 3차원 그래픽 렌더링 가속기를 구현하기 위해서는 많은 양의 메모리를 필요로 한다. 따라서 본 연구에서는 면적을 줄이고자, H.264 움직임 추정기와 3차원 렌더링 가속기의 메모리와 움직임 보상기와 3차원 렌더링 가속기의 픽셀 프로세서를 재구성 가능한 구조로 설계하여 하드웨어 면적 감소 효과를 얻을 수 있도록 하였다.

3. 움직임 추정기

제안한 H.264 인코더의 움직임 추정기 구조는 가변

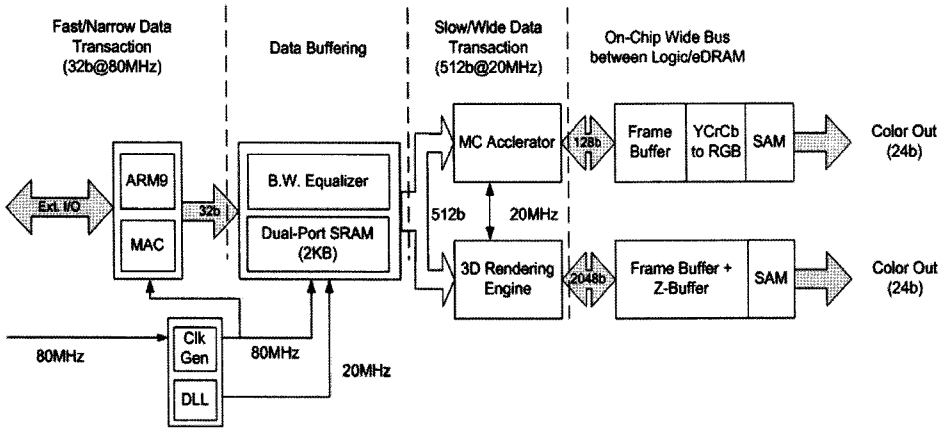

그림 4 [6]의 재구성 가능한 구조

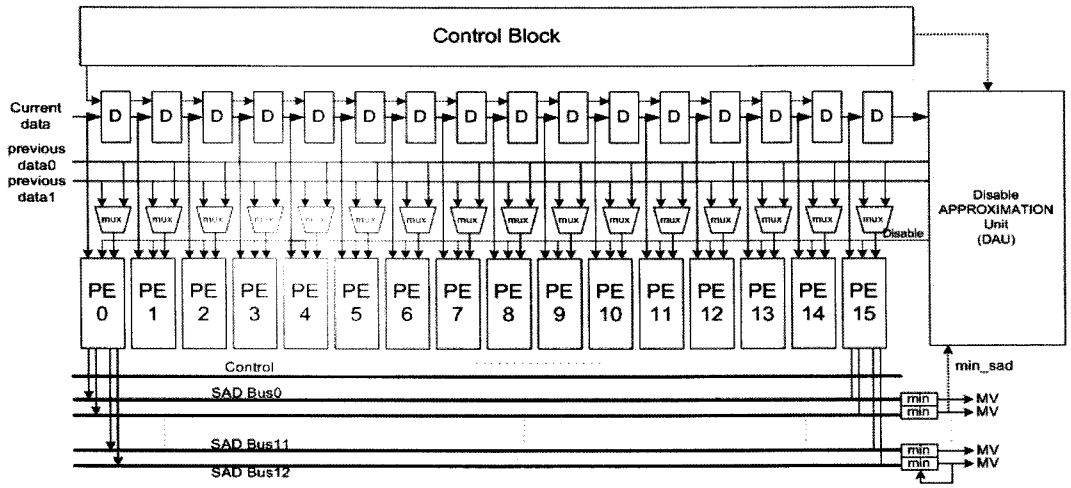

그림 5 제안한 움직임 추정기의 구조

블록을 지원하는 1D Systolic Array 구조를 사용하였다[7]. 1D Systolic Array 구조는 2D Systolic Array 구조에 비하여 유연한 구조를 가지고 있으며, 작은 면적으로 구현이 가능하다. 또한, 가변블록을 지원하기 위해 움직임 벡터를 찾기 위한 SAD를 구하는 계산 과정은 최소 단위인 4×4 크기의 SAD를 누적시킴으로써 각각의 크기에 맞는 SAD 값을 구하게 된다. 제안한 전체적인 구조는 그림 5에서 보이는 것과 같다. 제안된 구조는 크게 제어 블록과 16개 PE 블록과 DAU로 나누어진다.

본 연구에서는 일반적인 SAD 확률분포에 기초하여 효율적인 데이타 스캐닝 방법을 제시하여, 연산량을 평균 30% 감소시켰고, FDVS 알고리즘을 사용하여 다중 프레임 처리 시 연산량을 약 43% 감소시켰다. 또한 H.264 움직임 추정 시 연산량을 JM8.2에 비하여 70% 이상 감소하도록 하였다.

3.1 Disable Approximation Unit(DAU)

DAU 블록은 연산량을 줄여 전력 소모 감소시키는 것을 목적으로 설계하였다[3]. 대략적인 SAD인 $\hat{D}(u,v)$를 계산하여 현재까지 탐색한 최소 SAD $D_{\min}(V_{ssf})$와 비교한 후 현재 탐색점의 정합오차 계산 필요 여부를 결정함으로써 전력 소모를 줄일 수 있다.

식 (2)는 일반적인 SAD 연산식을 나타내며 식 (3)은 DAU에서 연산되는 대략적인 SAD 연산식을 나타낸다. (u,v)는 현재 프레임과 참조 프레임 사이의 상대적인 좌표로, 움직임 벡터를 나타낸다.

$$D(u,v)=\sum_{j=1}^{N}\sum_{i=1}^{N}|s(i+u,j+v)-r(i,j)| \qquad (2)$$

위 식에서 $r(i,j)$는 현재 프레임에서 좌표 (i,j)의 휘도성분을 나타내고, $s(i+u,j+v)$는 참조 프레임의 탐색

영역의 좌표 $(i+u,j+v)$의 휘도 성분을 나타낸다.

$$\hat{D}(u,v)=\sum_{j=1}^{N-1}|\sum_{i=1}^{N}s(i+u,j+v)-\sum_{i=1}^{N}r(i,j)| \quad (3)$$

대략적인 SAD는 일반적인 SAD를 구하는 것 보다 적은 연산량으로 구해진다.

식 (2)와 (3)의 관계는 식 (4)으로 표현할 수 있다.

$$\hat{D}(u,v) \le D(u,v) \quad (4)$$

탐색 영역 전체 V에 대한 최소정합오차를 $D_{\min}(V)$라 하고 현재까지 탐색한 부분을 V_{ssf} 라 하면 식 (5)가 성립됨을 알 수 있다.

$$D_{\min}(V) \le D_{\min}(V_{ssf}) \le \hat{D}(u,v) \le D(u,v) \quad (5)$$

따라서 $D_{\min}(V_{ssf}) < \hat{D}(u,v)$ 인 경우에는 SAD 계산을 할 필요가 없으므로, PE에 Disable 신호를 보내고 그 이후의 계산은 하지 않게 된다. 이를 이용하여 DAU 블록 1개를 사용함으로써 기존의 Full Search 방법 보다 57%이상의 연산량 감소 효과를 보였다(표 1 참조).

3.2 효율적인 데이타 스캐닝

비디오 영상에서 최소의 SAD값은 대부분 탐색 영역 원점 가까이에 분포한다[1]. 따라서 그림 6과 같이 31×31 탐색 영역에서 8번째 라인부터 시작해서 16×16 블록의 SAD값을 연산하는 효율적인 데이타 스캐닝 방법을 제안한다. 8번째 라인을 연산한 뒤에 위쪽 방향으로 진행하며 1번째 라인까지 16×16 블록의 SAD값을 연산하고, 9번째 라인부터 마지막 라인까지 16×16 블록의 SAD값을 연산하도록 메모리 컨트롤러를 설계하였다. 블록의 8번째 줄부터 16×16 블록의 데이타를 읽어와 계산함으로써 최소의 정합 오차를 빨리 찾아내 더 많은 부분을 Disable 하면 연산량을 더욱 줄일 수 있다.

표 1은 Full Search방법에 비해 DAU 사용 시의 감소 연산량을 나타낸다. 연산량은 알고리즘을 C로 구현하여 구한 것이다. 덧셈과 뺄셈을 1로 보고, 절대값을 2로 가중치를 주어 연산량을 계산하였다. %로 표현한 것

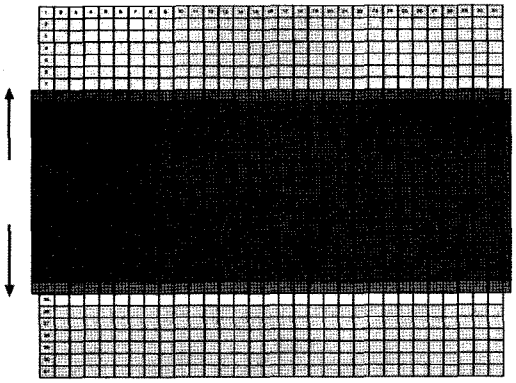

그림 6 제안한 효율적인 데이타 스캔 방법

은 Full Search 또는 DAU만 사용한 방법에 비해 줄어든 연산량의 비율을 보여준다. DAU 블록을 적용하여 구한 연산량이 JM8.2의 움직임 추정 연산량보다 57%정도 감소하였다. 본 논문에서 제안한 효율적인 데이타 스캔 방법은 JM8.2에 비해 연산량이 평균적으로 70% 감소하였다. 기존의 DAU 방법[3]에 비해 우리의 효율적인 데이타 스캔 방법은 30%정도의 연산량을 추가로 감소시킨다. 표 2는 Full Search와 DAU 사용 시 PSNR을 비교함으로 DAU를 사용하여도 영상의 화질에는 영향을 주지 않으면서 연산량을 감소시킬 수 있음을 보여준다.

3.3 FDVS를 이용한 다중 프레임 처리

다중 프레임을 사용하여 움직임을 예측하는 것은 압축 효율을 높이는 장점이 있다. 하지만 이전프레임인 F_{n-1}에서 움직임 벡터를 찾는 것과 같은 방법으로 F_{n-2}에서도 움직임 벡터를 찾으면 연산량이 참조 프레임 수에 비례해서 늘어나는 단점이 있다. 또한 필요한 메모리의 용량이 참조 프레임의 수와 비례해서 증가하게 된다. 그러나 제안한 움직임 추정기에서는 FDVS 알고리즘을 사용하여 연산량을 감소시킬 수 있다. FDVS를 이용하여 움직임 추정기에서 이미 구한 움직임 벡터를 저장하여 재사용한다. 제안한 구조에서는 FDVS 알고리즘을

표 1 DAU 사용에 따른 감소 연산량

	Full Search	DAU [3]		제안한 새로운 효율적인 데이타 스캔 방법		
	연산량	연산량	연산량 감소[1](%)	연산량	연산량 감소[1](%)	연산량 감소[2](%)
akiyo	103,809,024	40,414,940	61.07	29,784,996	71.31	26.30
forman	103,809,024	42,865,648	59.71	31,205,424	69.94	27.20
container	103,809,024	49,341,788	52.47	26,828,296	74.16	45.63
mother-daughter	103,809,024	46,157,916	55.54	35,774,400	65.54	22.45
news	103,809,024	43,933,712	57.68	29,334,904	71.74	33.23
average	103,809,024	44,542,801	57.09	30,585,604	70.54	30.97

1 : Full Search와 비교했을 때의 감소량

2 : DAU [3]와 비교했을 때의 감소량

표 2 DAU 사용에 따른 PSNR

[단위 : dB]

	Full Search	DAU[3]	제안한 효율적인 데이타 스캔 방법
akiyo	38.10	38.10	38.10
forman	31.79	31.79	31.79
container	35.58	35.58	35.58
mother-daughter	40.27	40.27	40.27
news	35.48	35.48	35.48
average	36.24	36.24	36.24

표 3 FDVS 사용에 따른 감소 연산량

		연산량	%1	%2
Single Search		103,809,024	0	100
GeneralMulti		207,618,048	100	0
FDVS Multi	16×16	104,214,528	0.39	49.80
	19×19	110,297,088	6.25	46.88
	21×21	118,407,168	14.06	42.97
	23×23	129,761,280	25.00	37.50
	25×25	144,359,424	39.06	30.47
	27×27	162,201,600	56.25	21.88
	29×29	183,287,808	76.56	11.72
	31×31	207,618,048	100.00	0.00

1 : Single Search에 비해 증가한 연산량
2 : GeneralMulti에 비해 감소한 연산량

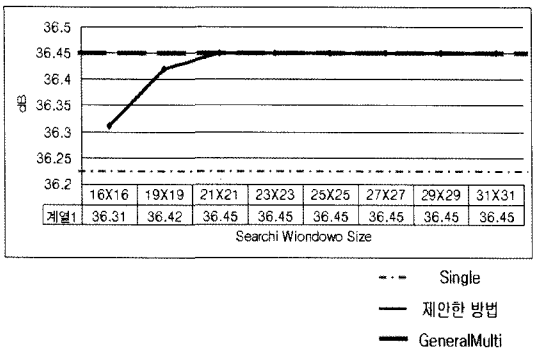

그림 7 FDVS 사용 시 Search Window 크기에 따른 PSNR

이용하여 2개의 참조 프레임을 사용하며 21×21 Search Window을 이용하여 여러 참조 프레임을 사용하지 않고도 일반적인 다중 프레임을 사용한 효과를 얻을 수 있도록 하였다.

표 3에서 Single Search는 다중 프레임 처리를 하지 않았을 경우 연산량을 의미하여, GeneralMulti는 기존 다중 프레임 처리 시 연산량을 의미한다.

그림 7은 FDVS 알고리즘을 C언어로 구현하여 탐색 영역 크기에 따른 PSNR을 그린 것이다. 기존 연구[4]에서는 FDVS를 이용하여 탐색 영역을 16×16으로 사용하였으나, 본 연구에서는 그림 7에 보이는 바와 탐색 영역 크기에 따른 PSNR을 도식화 하여, PSNR이 포화(수렴)되어 화질 저하가 없는 21×21의 탐색 영역을 사용하였다.

4. 움직임 보상기

영상 압축 코덱에서는 압축효율을 높이기 위해 현재 프레임과 참조 프레임의 오차값과 움직임 벡터만을 인코딩하여 전송한다. 이 정보는 디코더의 입력으로 사용되며 H.264 디코더에서는 움직임 벡터로 예측값을 읽어 오고, 이를 예측 오차 값과 더해서 화면을 복원한다.

그림 8은 제안한 H.264 디코더의 움직임 보상기 구조를 보여준다. Data Alignment 블록에서는 움직임 벡터를 입력값으로 받아 메모리의 주소값을 연산한다. 픽셀 프로세서는 인코더에서 전송된 블록의 오차값과 참조 프레임의 블록을 더해 화면을 복원하고 MC 프레임 버퍼에 저장한다. MC 프레임 버퍼는 각각 44Kb(16Line×352×8bits)로 설계하였으며, 데이타 출력의 지연이 발생하지 않도록 더블버퍼링 하였다.

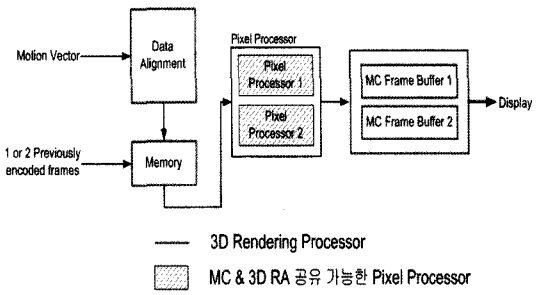

그림 8 제안한 움직임 보상기

5. 3차원 렌더링 가속기

3차원 그래픽 주사 변환 처리기 구조에서는 고라우드 셰이딩을 적용하여 색상 정보를 연산하도록 설계 하였다. 고라우드 셰이딩의 처리 시간은 퐁 셰이딩의 1/3정도로 효율적이며, 플랫 셰이딩의 단점인 마하밴드 효과를 줄일 수 있다[5].

그림 9는 제안한 3차원 렌더링 가속기의 구조를 보여준다. 폴리곤 프로세서에서는 폴리곤의 세 정점의 기울기를 구하는 폴리곤 처리를 하게 되고, 에지 프로세서에서 폴리곤의 경계값을 구하는 변처리와 픽셀 프로세서에서 폴리곤 내부의 색상정보를 구하는 주사 변환 처리의 역할을 하게 된다.

픽셀 프로세서에서 주사 변환 처리를 할 때 중심선 트래버셜 알고리즘을 적용하여 연산함으로써 병렬처리

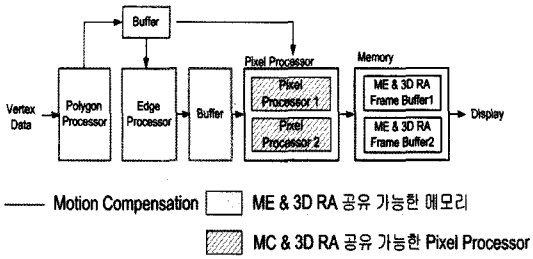

그림 9 제안한 3차원 렌더링 가속기

를 가능하게 하여 처리량을 높였다. 기존의 중심선 트래버셜 알고리즘에서는 오른쪽 픽셀의 유효값을 저장하였다가 왼쪽 픽셀의 연산을 마친 후 다시 돌아가서 연산하도록 하였다. 제안한 구조에서는 유효값을 저장하지 않고 2개의 프로세서를 동시에 연산하도록 하였다.

화면에 디스플레이 되기 전에 데이타를 저장하는 프레임 버퍼는 더블 버퍼링이 가능한 구조로 설계하여 처리 시간 지연에 따른 플리커(flicker) 현상이 발생하지 않도록 하였다. ME & 3D RA 프레임 버퍼는 R, G, B, Z 값을 각각 저장할 수 있도록 메모리를 분산시켜 사용하였다. 또한 처리된 픽셀값에서 깊이값인 Z 값을 비교하여 Z값이 작은 것을 저장하고 디스플레이 하도록 하였다.

6. 재구성 가능한 구조

본 연구에서는 H.264 인코더의 움직임 추정기와 3차원 그래픽 렌더링 가속기의 메모리와 디코더의 움직임 보상기와 3차원 그래픽 렌더링 가속기의 픽셀 프로세서를 공유하여 이동통신 기기에 적합하도록 재구성 가능한 구조로 설계하였다.

재구성 가능한 구조는 그림 10에 보였다. 2개의 픽셀 프로세서와 메모리 1~6 블록이 H.264와 3D 렌더링 가속기의 공유되는 부분이다. H.264 움직임 추정기 처리 속도는 CIF 2.3 Frame/sec를 처리할 수 있다. PSNR은 DAU나 효율적인 데이타 스캔 방법을 적용하여 화질 열화가 발생하지 않도록 하였다. 또한, 움직임 보상기 처리속도는 CIF 3.8 Frame/sec 처리가능하다.

제안한 3차원 그래픽 렌더링 가속기에서는 고라우드 셰이딩을 적용하여 색상 표현을 부드럽게 되도록 하였고, 더블 버퍼링을 사용하여 계산이 많을 경우에도 디스플레이에 문제가 없도록 하였다. 또한 픽셀 프로세서에서 깊이 값에 따른 색상의 변화를 계산하도록 구현하였다. 제안한 구조의 동작은 C언어로 모델링 하여 Simulation 결과로 정상 동작함을 확인하였다. 또한 하드웨어의 면적은 움직임 추정기와 3차원 그래픽 가속기를 각각 Verilog HDL로 기술하여 합성한 결과의 Gate Count를 기준으로 추정하였다.

6.1 픽셀 프로세서의 공유

그림 10에서 보이는 구조로 H.264 디코더의 움직임 보상기와 렌더링 가속기의 픽셀 프로세서를 공유하도록 설계하였다. 픽셀 프로세서에 움직임 보상기의 모든 블록을 포함하도록 설계하여 각각 설계하였을 경우보다 약 7%의 하드웨어면적을 감소하도록 하였다.

그림 10에서 보이는 것과 같이 2개의 픽셀 프로세서는 움직임 보상기와 3차원 렌더링 처리기에서 공유하도록 설계하였으며, 병렬처리 가능한 구조를 사용하여 처리량을 늘렸다.

그림 10의 픽셀 프로세서 중 ALU는 H.264와 3D 렌더링 가속기와 공유되는 부분이고 레지스터(REG)는 3D 렌더링 가속기로만 동작하는 부분이다. H.264 움직임 보

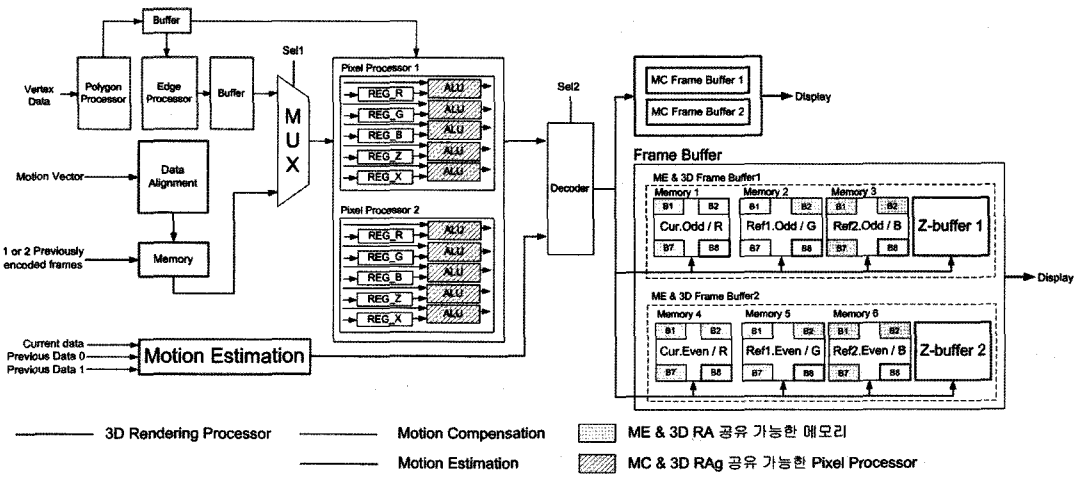

그림 10 제안한 재구성 가능한 구조

상기로 동작할 경우 레지스터의 전원을 차단하여 전력 소모의 낭비를 줄였다. 픽셀 프로세서의 동작 여부 선택은 컨트롤러에서 발생되는 Sel1 신호에 의해 결정된다.

6.2 메모리 공유

동영상 및 3차원 그래픽 영상은 많은 양의 데이타를 사용하므로, 소요되는 메모리의 양도 많아지게 된다. 본 연구에서는 H.264 인코더의 움직임 추정기의 메모리와 3차원 그래픽 렌더링 가속기의 메모리를 공유하도록 설계하여 필요한 메모리의 양을 줄였다.

H.264의 움직임 추정기에서는 현재 프레임과 2개의 참조 프레임을 저장하여야 함으로 792 Kbits (352×288×8bits) 크기의 메모리 3개가 필요하며, 총 약 2.4 Mbits의 메모리를 사용하게 된다. 3차원 렌더링 가속기에서는 256×256 크기의 영상 내의 폴리곤을 처리하며 약 5.1 Mbits (256×256×(8×3+16)×2 bits)의 메모리를 필요로 하게 된다.

그림 10에서 메모리 1~6은 각각 512 Kbits의 크기를 가지며, 8개의 블록(B)으로 나누어 하나의 블록(B)은 64 Kbits의 크기로 설계하였다. Z-Buffer 1, 2는 각각 1 Mbits로 설계하였다.

메모리 1~6은 움직임 추정기로 사용할 경우 현재 프레임과 2장의 참조 프레임을 저장한다. 현재 프레임은 메모리 1, 4에 메모리 Address에 따라 Odd, Even 라인으로 분할하여 저장하며, 2장의 참조 프레임은 2, 5와 3, 6에 현재 프레임과 같은 방식으로 저장한다. 3차원 렌더링 가속기로 사용할 경우 색상 버퍼로 R, G, B 값을 저장한다. 움직임 추정기로 동작할 경우 메모리 1~6은 각각은 396 Kbits의 메모리만 필요하게 되므로 메모리 1~6의 B8 블록의 전원을 차단한다. Z-buffer 1, 2는 3차원 렌더링 가속기의 깊이(Z-buffering) 데이타를 저장하며 움직임 추정기로 동작할 경우 전원을 차단한다. 메모리 1~6은 움직임 추정기로 동작할 경우 메모리 1~6의 B8 블록과 Z-buffer 1, 2의 메모리 블록의 전원을 차단함으로 해서 전력이 낭비되는 것을 최소화 하였다. 3차원 그래픽 가속기와 움직임 추정기에서 재구성 가능한 구조로 사용되는 Frame Buffer와 움직임 보상기의 데이타를 저장하는데 사용되는 MC Frame Buffer 1, 2의 동작 선택은 컨트롤러에서 발생되는 Sel2에 의해 결정된다. 또한, 그래픽 가속기와 움직임 추정기에서 사용되는 Frame Buffer의 Memory Scheduling은 Decoder에서 담당한다.

메모리 부분은 각각 설계 할 경우 약 7.5 Mbits의 메모리가 필요하지만, H.264의 메모리 블록과 3차원 그래픽 메모리 블록을 공유하여 약 5.1 Mbits의 메모리만 사용하도록 설계하여 메모리 면적을 32% 감소 (46% 공유) 하도록 하였다.

7. 결 론

휴대용 단말기에 동영상을 위한 H.264 코덱과 3차원 그래픽 가속기를 독립적으로 사용할 경우 큰 면적과 많은 전력 소모로 어려운 점이 많다. 본 연구에서는 H.264와 3차원 그래픽 가속기를 재구성이 가능한 구조를 이용하여 픽셀 프로세서와 메모리를 공유하도록 설계하였다. H.264와 3차원 그래픽 가속기를 독립적으로 설계하였을 때에 비해 메모리는 7.5 Mbits에서 약 5.1 Mbits로 감소하였고, 픽셀 프로세서를 공유하여 전체 하드웨어 면적의 7%가 감소하였다.

H.264 인코더 중 움직임 추정기에서는 제안한 데이타 스캔 방법과 DAU를 사용하여 연산량을 JM8.2에 제시된 움직임 추정기보다 연산량을 70% 정도 줄였으며, 기존의 DAU 방법에 비해서도 30% 감소시켰다. 전력 소모를 감소시키기 위해 공유된 블록중 사용하지 않는 블록은 전원을 차단시켜 소모 전력을 줄이도록 설계하였다.

참 고 문 헌

[1] I. Richardson, "H.264 and MPEG-4 Video Compression Video Coding for Next-generation Multimedia," John Wiley & Sons, 2003.

[2] http://www.kumnong.co.kr/

[3] Viet L. Do, Kenneth Y. Yun, "A Low-Power VLSI Architecture for Full-Search Block-Matching Motion Estimation," IEEE Trans on Circuits and Systems for Video Technology, Vol.8, No.4, pp.393-398, Aug. 1998.

[4] M. Chen, Y. Chiang, H. Li, M. Chi, "Efficient multi-frame motion estimation algorithms for MPEG-4 AVC/JVT/H.264," Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on Vol.3, pp. 737-740, May 2004.

[5] T. Akenine-Moller, E. Haines, "Real-Time Rendering 2nd Edition," AK Peters, 2002.

[6] C. Yoon, R. Woo, J. Kook, S. Lee, K. Lee, and H. Yoo, "An 80/20-MHz 160-mW Multimedia Processor Integrated With Embedded DRAM, MPEG-4 Accelerator, and 3-D Rendering Engine for Mobile Applications," IEEE Journal of Solid-State Circuits. Vol.36. Nov. 2001.

[7] S. Yeow Yap, J. V.McCanny, "A VLSI Architecture for Variable Bloack Size Video Motion Estiamtion," IEEE Trans on Ciruits and Systems, Vol.51, No.7, pp.384-389, Jul 2004.

박 정 애

2003년 숭실대학교 정보통신전자공학부 학사. 2006년 한양대학교 전자전기제어계측공학과 석사. 2006년~현재 코아로직 3D Graphics Development 팀 연구원 관심분야는 저전력 설계, 3D Graphics, H.264

윤 미 선

2005년 대진대학교 전자공학과 학사. 2006년 한양대학교 전자전기제어계측공학과 석사과정. 관심분야는 저전력 설계, H.264, 3D Graphics, 반도체 설계

신 현 철

1978년 서울대학교 전자공학과 학사 1980년 한국과학기술원 전기 및 전자공학 석사. 1983년~1987년 U.C. Berkeley Ph.D. 1983년~1987년 Fulbright scholarship. 1987년~1989년 MTS, AT&T Bell Lab's, Murray Hill N.J., USA. 1989년~현재 한양대학교 전자컴퓨터공학부 교수. 1997년~현재 IDEC 한양대학교 지역센터 센터장. 관심분야는 CAD&VLSI, 통신용 반도체 설계, 저전력 설계