# 고성능 텍스쳐 매핑을 위한 압축된 텍스쳐의 캐쉬 시스템 설계

양 진 기○ 박 우 찬 한 탁 돈
연세대학교 병렬처리시스템 연구실

# Cache System Design of Compressed Texture for High Performance Texture Mapping

Jin-Ki Yang, Woo-Chan Park, Tack-Don Han
Dept. of Computer Science, Yonsei University

## 요 약

보다 현실적인 3차원 영상을 얻기 위한 텍스쳐 매핑은 대부분의 그래픽 시스템에서 사용하고 있다. 3차원 그래픽 시스템이 생성한 객체의 표면 위에 2차원 이미지를 입힘으로써 그래픽 시스템의 성능저하를 가져오지 않으면서 영상의 현실성을 높이는 텍스쳐 매핑은 텍스쳐 이미지를 저장하기 위해 많은 메모리가 요구되며 고성능 텍스쳐 시스템을 위해 빠른 메모리 접근과 광대한 대역폭이 요구된다. 본 논문에서는 벡터 양자화(Vector quantization) 압축기법을 이용하여 텍스쳐 이미지에 대한 효율적인 압축을 통해 많은 메모리 요구를 해결하며 압축된 텍스쳐 이미지의 효율적인 캐싱을 통해 빠른 메모리 접근과 광대한 대역폭 문제를 해결할 수 있는 구조를 제시한다. 본 논문에서 제안된 구조는 버퍼링을 통해 메모리 접근 시간을 숨김으로써 고성능 텍스쳐 시스템을 지원할 수 있다.

## 1. 서론

그래픽 시스템에 기하학적 연산 량을 증가시키지 않고서도 보다 현실적인 3차원 영상을 얻기 위한 방법인 텍스쳐 매핑은 현존하는 모든 3차원 그래픽 시스템에서 채택하는 방법이다. 텍스쳐 매핑은 화면 공간에서 텍스쳐 공간으로 의 매핑을 통하여 화면 화소에 해당하는 텍스쳐 이미지의 영역을 결정하고 그 영역 위에서 필터링을 통해 하나의 텍스쳐 화소값을 생성하여 화면 화소의 색깔 값을 대치한다 텍스쳐 매핑은 객체의 표면 위에 텍스쳐 이미지를 입힘으로써 적은 폴리곤으로 추상화된 객체가 보다 세밀하고 현실적인 표현을 갖도록 한다 [1] 텍스쳐 매핑 기법은 화면 화소의 텍스쳐 이미지 상에서의 영역을 결정하는 매핑에 필요한 많은 연산 량과 텍스쳐 이미지를 저장하기 위한 많은 메모리를 요구한다. 끊임없이 발전해온 반도체 기술은 텍스쳐 매핑이 요구하는 연산량를 어느 정도 해결하고 있지만 많은 텍스쳐 메모리 요구와 데이터 전송 량은 고성능 텍스쳐 매핑시스템 설계의 병목점이다. 실시간 고성능 영상을 얻기 위한 그래픽 시스템의 텍스쳐 매핑은 초당 10~100 백만의 텍스쳐 메모리 접근과 적은 회전 지연 시간을 요구하고 2~3 MB에서 수십MB의 메모리를 요구한다 [3,4,5]

텍스쳐 매핑을 지원하는 그래픽 시스템은 두 가지로 나눌 수 있다 첫째, 기속기에 텍스쳐 이미지를 저장할 메모리를 할당하는 시스템은 빠른 메모리 접근을 얻는 장점은 있지만 메모리 양에 대한 한계, 낮은 메모리 활용도, 그리고 개발자가 메모리를 관리해야 하는 단점이 있다. 둘째, 시스템 메모리에 텍스쳐 이미지를 저장하는 그래픽 시스템은 메모리의 한계는 없지만 가속기가 요구하는 데이터 전송 량을 지원할 수 없다는 단점이 있다 [6].

본 논문에서는 텍스쳐 이미지를 위한 할당된 메모리를 갖는 시스템의 빠른 메모리 접근의 장점과 시스템 메모리에 텍스쳐 이미지를 저장하는 시스템의 메모리 양의 장점을 고루 수용하며 고성능 텍스쳐 매핑 시스템에 요구되는 많은 메모리 요구 량과 데이터 전송 량을 해결하기 위해 텍스쳐 이미지를 압축하고 캐싱하는 텍스쳐 매핑 시스템을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 텍스쳐 매핑에서 많이 사용되는 밉매핑(Mip mapping)에 대해서 알아본다 3장에서는 제안하는 텍스쳐 매핑 구조와 관련된 연구에 대해서 알아본다. 4장에서는 제안하는 텍스쳐 매핑 구조를 설명한다.5장에서는 성능비교를 자세히 설명한다. 끝으로 6장에서는 본 논문의 결론을 제시한다.

## 2.밉매핑(Mip mapping)

그래픽 시스템이 생성한 화소의 색깔 값을 텍스쳐 이미지의 색깔 값으로 대치하는 텍스쳐 매핑은 화면 위의 삼각형 내의 각 화소에 대해 2차원 텍스쳐 이미지내의 해당 영역을 결정하는 매핑과정과 매핑된 영역에서 화면 화소에 필요한 색깔 값을 결정하는 필터링 과정으로 구분된다. 텍스쳐 공간에서 하나의 화면 화소에 해당하는 텍스쳐 화소는 정확히 하나의 화소에 해당하지 않는다. 여러 개의 텍스쳐 화소가 하나의 화면 화소에 해당하는 경우(축소)와 하나의 텍스쳐 화소가 여러 개의 화면 화소에 해당하는 경우(확대)가 있다. 이런 경우에 생기는 왜곡현상(Ahasing)을 줄이기 위해 필터링이 행해진다. 밉매핑은 그래픽 시스템에서 가장 많이 사용되고 있는 필터링이다. <그림 1>에서 보는 것과 같이 밉매핑은 원래의 텍스쳐 이미지를 기본이미지로 1/4 씩 해상도를 줄여가며 이미지 피라미드를 생성한다. 각 화면 화소에 대해 텍


< 그림 1 밉매핑 >

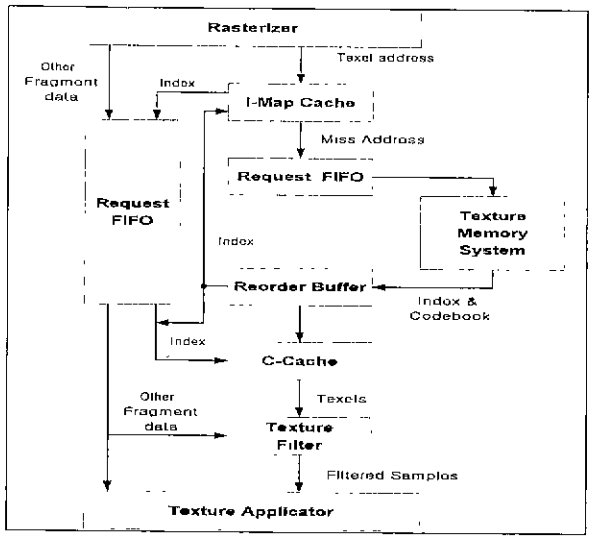
스쳐 공간에서의 좌표 값과 레벨 값을 계산한다. 화면 화소와 텍스쳐 화소의 비율을 레벨 값으로 사용하며 이미지 피라미드에서 이 레벨 값에 가장 근접한 두 밉맵 레벨을 선정한다 각 레벨 이미지에서 텍스쳐 좌표와 가까운 네 개의 화소 값을 구하고 이 여덟 개의 텍스쳐 화소 값을 평균하여 하나의 텍스쳐 색깔 값을 결정한다. 이를 삼각선형 보간법(Trilinear Interpolation)이라 한다. 밉매핑은 이미지 왜곡현상(Aliasing)을 줄이기 쉽고 매핑 연산이 효율적이면서도 메모리 요구 량이 원이미지의 4/3배만 필요하기 때문에 가장 많이 사용되고 있는 필터링방법이다. 그리고 여러 해상도의 이미지에서 레벨 값에 따른 매핑은 화면 화소와 텍스쳐 화소의 비율을 거의 1:1 이 되도록 만들기 때문에 메모리 접근이 밀접한 특성을 가지고 있다 [2].

## 3 관련 연구

밉맵을 이용한 텍스쳐 매핑 그레픽 시스템은 하나의 화면 화소를 텍스쳐 매핑하기 위해 8개의 텍스쳐 화소를 필요로 한다[2]. 실시간 영상을 지원하는 고성능 그레픽 시스템은 초당 10~100 백만 개의 화면 화소를 생성한다. 이 화소에 대한 텍스쳐 매핑은 한 화소에 대해 8번의 텍스쳐 메모리 접근이 요구되기 때문에 초당 2~3 GB정도의 메모리 대역폭과 빠른 메모리 접근이 필요하다. 지금까지 고성능 텍스쳐 시스템에서는 텍스쳐 매핑의 많은 메모리 요구, 대역폭, 빠른 메모리 접근에 대한 문제를 해결하기 위한 노력이 진행되어 왔다. 실리콘 그래픽스의 Reality Engine은 다중 프래그먼트 생성기(Multiple fragment generator)를 사용하는데 각 프래그먼트 생성기마다 16MB 정도의 메모리를 할당해서 빠른 메모리 접근을 제공한다. 그러나 리얼리티 엔진은 빠른 메모리 접근을 위해 모든 지역 메모리에 텍스쳐 이미지가 복사되어야 하기 때문에 실질적으로 이용할 수 있는 메모리 용량은 단지 16MB뿐이고 많은 텍스쳐 이미지를 요구하는 영상에서는 메모리 용량 한계의 문제를 가지고 있다 [7]. 스탠포드의 Hakura는 가속기와 시스템 메모리 사이에 SRAM을 두는 캐싱기법을 통해 빠른 메모리 접근과 광대역폭 문제를 해결하고 있다 [3]. 텍스쳐 시스템의 메모리 접근은 연속적인 메모리 접근보다 인접한 메모리 접근 특성을 지니고 있다. Hakura는 텍스쳐 시스템의 메모리 접근 특성을 이용하여 캐쉬의 구조와 텍스쳐 이미지의 메모리 저장방법을 제시하고 있다. 그의 연구 결과는 보통 32KB 크기와 2차웨이 연관캐쉬(2-way set associativity)와 블록 표현의 메모리 저장이 빠른 메모리 접근과 광대역폭 문제를 해결할 수 있다고 제시하고 있다. 또한 [4]에서는 캐쉬구조에 선 인출(Prefetching)기법을 적용하여 더 빠른 메모리 접근을 제시하고 있는데 이는 캐쉬에서 태그부분을 분리하여 실제 캐쉬내의 데이터를 접근하기 전에 미리 필요한 데이터의 캐쉬내 존재 여부를 판단하는 방법을 이용하여 캐쉬의 접근 실패율을 줄이고 있다 위 두 방법은 텍스쳐 시스템의 빠른 메모리 접근과 광 대역폭 문제의 해결을 제시하고는 있지만 많은 텍스쳐 메모리 요구는 해결하지 못하고 있다. 텍스쳐 메모리를 효과적으로 줄이기 위한 방법으로 [5]에서는 벡터 양자화(Vector quantization) 방법을 이용하여 텍스쳐 데이터를 압축하는 방법을 제시하고 있다. 벡터 양자화(Vector quantization) 압축은 이미지에 대해 4x4 블록단위로 256개의 대표블록을 가진 부호책(codebook)을 생성하고 이 부호책(codebook)에대한 색인지도(Index map)을 만들어서 압축하는 방법이다. 벡터 양자화 압축방법은 압축을 푸는데 걸리는 시간의 매우 적기 때문에 텍스쳐 매핑에서 필요한 빠른 메모리 접근의 요구를 충분히 수용할 수 있다. 이런 벡터 양자화 방법을 통해 텍스쳐 이미지의 크기를 효율적으로 줄이고 있다. 또한 밉맵에서 벡터 양자화 압축방법도 제시하고 있다. 여러 레벨의 이미지를 3개의 레벨씩 하나의 그룹으로 묶고 각 그룹에 대해 하나의 부호책(codebook)과 색인지도(Index map)를 생성한다. 이 연구에서는 95%정도의 압축 결과를 제시하고 있다

## 4. 제안하는 구조

밉맵에 적용된 벡터 양자화(Vector quatization) 압축기법은 3개 레벨의 이미지에 대해 첫째 레벨에서 4x4 블록 단위로 부호단어(codeword)를 생성하고 두 번째 레벨에서 2x2블록, 세 번째 레벨에선 1x1 블록에 대해 압축을 실행한다. 이 방법은 3개 레벨에 대해 하나의 부호책(codebook)과 색인지도(Index Map)를 생성하여 압축의 효율을 높일 수 있다. 밉맵 텍스쳐 매핑에서 이웃하는 화면 화소는 텍스쳐 이미지 내에서 인접한 메모리를 접근하는 특성이 있다. 텍스쳐 데이터를 블록단위로 저장하는 방법은 텍스쳐 메모리 접근의 높은 지역성을 이용하고는 있지만 밉맵의 이미지 피라미드는 하위레벨과 상위레벨의 지역성이 같지 않다 상위레벨로 갈수록 그




< 그림 2 압축된 텍스쳐의 캐싱 구조 >

지역성은 더욱 높다 벡터 양자화 압축기법을 통한 밉맵의

표현은 블록단위의 압축을 통해 텍스쳐 메모리 접근의 지역성을 이용할 뿐만 아니라 3개의 레벨사이의 다른 지역성을 반영하고 있다. 본 논문에서는 이런 압축기법을 통해 압축된 텍스쳐 이미지에 대한 캐쉬구조를 제시한다. <그림 2>에서 그 구조를 보여주고 있다. 색인지도를 위한 캐쉬(I-Map Cache)와 부호책을 위한 캐쉬(C-Cache)를 두어 색인지도와 부호책 데이터의 캐싱을 수행하는데 시간상으로 색인지도 캐싱을 앞서 수행함으로써 부호책 캐쉬에대한 접근 실패율을 낮추는 구조를 가지고 있다. 구조의 제어 흐름을 살펴보면 다음과 같다. 먼저 레스터기가 생성한 텍스쳐 좌표에서 해당 그룹과 블록값, 부호단어(codeword)내의 오프셋(offset)을 생성한다 그룹값과 블록 값을 이용하여 I-Map Cache내에서 해당 색인(index)값이 있는지를 결정한다. 해당 색인(index)값이 없을 경우 메모리 요구 신호를 보내고 이 신호는 비퍼링을 통해 파이프리이닝(pipelining)효과를 얻는다. 이때 만약 그룹 값이 바뀌었을 경우는 새로운 그룹의 부호책이 필요한 경우이기 때문에 부호책에 대한 메모리 요구를 수행한다. 하나의 부호책에 3개의 레벨에 대한 데이터가 저장되어 있기 때문에 부호책에 대한 메모리 요구는 자주 발생하지 않는다. 부호책의 크기는 256개의 부호단어(codeword)와 각 부호단어에 63 바이트의 정보가 저장되어있기 때문에 C-cache의 크기는 16KB정도이다. 제안한 구조는 레스터기에서 생성된 화면 화소에 대해 I-Map cache에 대한 접근을 함과 동시에 Fragment FIFO를 통해 그 값을 전달하고 메모리 요구시 Request Buffer를 통해 파이프 라이닝(Pipelining)을 시킴으로써 메모리 접근 시간을 효과적으로 숨길 수 있다.

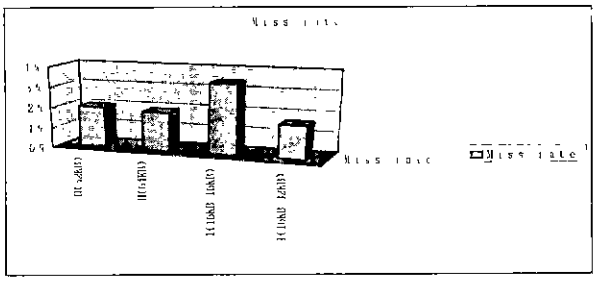
## 5. 성능분석

캐싱을 사용하지 않은 경우와 제안한 구조와의 성능을 분석해 보면 다음과 같다. 캐쉬 접근 시간을 t1,메모리 접근시간을 t2라 하자. I-Map 캐쉬 접근 성공률을 h1, 집근 실패시 그룹이 바뀌는 경우를 h2라 하자. 캐싱을 하지 않는 경우의 평균접근 시간은 2*t2이다. 제안한 구조의 평균 접근 시간은 식(1)과 같다

$$(1+h1)*t1+(1-h1)*(1+h2)*t2 \qquad (1)$$

Hakura의 연구에서 32KB 크기의 캐쉬에서 평균 접근 실패율은 2%미만이다. 대개 캐쉬의 접근 시간은 25ns이며 Rambus DRAM의 경우 메모리 접근 시간은 90ns이다 하나의 그룹에 대해서는 부호책이 바뀌지 않기 때문에 h2의 값은 0이고 메모리 접근 시간은 버퍼링 시간을 포함한 100ns를 가정한다. 하나의 텍스쳐 화소값을 얻기 위해 압축된 이미지가 시스템 메모리에만 존재할 경우 180ns 정도의 평균 접근 시간이 걸리며 본 구조에서는 약 50 ns정도의 평균 접근 시간이 걸린다. 본 구조의 경우 캐싱을 고려하지 않은 경우보다 약 70% 정도의 평균 접근 시간의 향상을 기대할 수 있다.

<그림3>은 Hakura의 구조와 제안한 구조와의 캐쉬 접근 실패율을 비교한 것이다 64B 블록 크기와 2차웨이 연관 캐쉬를 기본으로 Hakura의 경우 32KB와 제안한 구조의 16KB I-Map cache크기, 16KB C-cache 크기를 비교했을 경우 제안한 구조의 접근 실패율이 2배정도 높음을 볼수 있다. 그러나 제안 구조의 접근 실패율의 약 1/2은 삼각 선형 보간법(Trilinear Interpolation)을 실행할 때 두 밉맵 레벨이 서로 다른 그룹에 속해있을 경우 반복해서



<그림 3 캐쉬 접근 실패율 비교>

C-Cache에 대한 메모리 요구를 하기 때문이다. C-cache를 2개 두어 각각 서로 다른 그룹의 부호책을 가질 경우 위와 같은 실패율은 현저하게 줄일 수 있다. 위의 경우 총 48KB크기의 캐쉬를 요구하게 된다. Hakura의 경우 <그림 3>의 2번째 그래프(64KB 캐쉬크기)에서 보다시피 32KB이상의 캐쉬를 가지더라도 캐쉬 접근 실패율의 감소는 그리 크지 않다. 그러므로 본 구조는 Hakura가 제안한 구조보다 약간의 캐쉬크기의 증가로 같은 성능을 보임과 동시에 메모리 요구량에서도 95% 이상의 감소량을 제공한다.

## 6. 결론 및 향후 연구

본 논문에서는 벡터 양자화(Vector quantization) 압축기법을 통해 텍스쳐를 압축함으로써 텍스쳐 매핑의 메모리 요구를 효과적으로 줄이고 압축된 데이터를 효과적으로 캐싱하는 캐쉬 구조를 제시함으로써 텍스쳐 매핑의 많은 메모리 요구와 빠른 메모리 접근과 광대한 대역폭의 요구를 해결한다. 이 구조는 메모리 요구 량에서 21.2 : 1의 효율을 보임과 동시에 약 70 %의 평균 접근 시간 향상을 기대할 수 있다. 향후 밉맵 텍스쳐 매핑에 좀더 효율적인 압축방법의 개발을 통해 본 구조의 캐싱 효과를 높이도록 할 것이다

## 7. 참고 문헌


[1] Paul S. Heckbert. "Survey of Texture Mapping," *IEEE Computer Graphics and Applications*, p.p. 56-67, Nov. 1986.

[2] L. Williams, "Pyramidal Parametrics," *In Proceedings of SIGGRAPH '83*, vol. 17, p.p 1-11, 1983.

[3] Ziyad S. Hakura, Anoop Gupta, "The Design and Analysis of a Cache Architecture for Texture Mapping," *Proceedings of the 24th International Symposium on Computer Architecture*, 1997

[4] Homan Igehy, Matthew Eldridge, Kekoa Proudfoot, "Prefetching in a Texture Cache Architecture," *Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '98*

[5] Andrew C. Beers, Maneesh Agrawala, Navin Chaddha, "Rendering from Compressed Textures." *In Proceedings of SIGGRAPH '96*. p.p. 373 - 378. August 1996.

[6] Michael Cox, Narendra Bhandari, Michael Shantz, "Multi-Level Texture Caching for 3D Graphics Hardware," *In Proceedings of the 25th Annual International Symposium on Computer Architecture(ISCA '25)*. June 1998.

[7] Kurt Akeley, "Reality Engine Graphics," *In Proceedings of SIGGRAPH '93*, p.p. 109-116, August 1993.