한국게임학회 논문지
제7권 제3호 2007년 9월

향상된 가시성 검사를 수행하는 3차원 그래픽 가속기의 픽셀 파이프라인 구조

김일산[0], 박우찬*, 박진홍**, 한탁돈***
연세대학교 컴퓨터과학과
sany@yonsei.ac.kr[0], pwchan@sejong.ac.kr*, {jhpark, hantack}@kurene.yonsei.ac.kr**,***

A Pixel Pipeline Architecture with Effective Visibility Test for 3D Graphics Accelerators

Il-san Kim[0], Woo-Chan Park*, Jin-Hong Park**, Tack-Don Han***
Dept. of Computer Science, Yonsei University

요 약

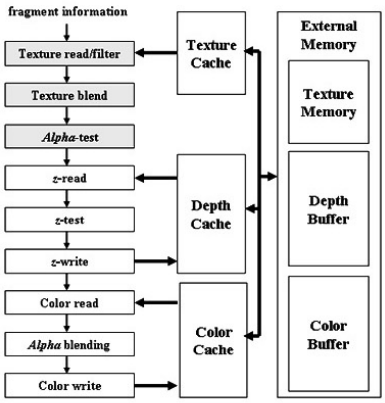
본 논문에서는 향상된 가시성 검사를 수행하여 기존의 중-텍스처링 구조에 비하여 데이터 전송량 및 깊이 캐쉬의 셀 면적을 감소시킨 픽셀 파이프라인 구조를 제시하였다. 제안하는 구조는 인접한 픽셀들 간의 가시성이 동일할 확률이 높다는 점을 이용하여 한 번의 가시성 검사만 수행하면서도 중-텍스처링 구조와 대등한 성능을 보이는 픽셀 파이프라인 구조이다. 실험결과, 제안하는 구조는 중-텍스처링 구조에 근접하는 성능을 보이면서도 깊이 캐쉬의 전송량은 평균 25%, 깊이 캐쉬의 면적은 약 40%가 감소하였다.

Abstract

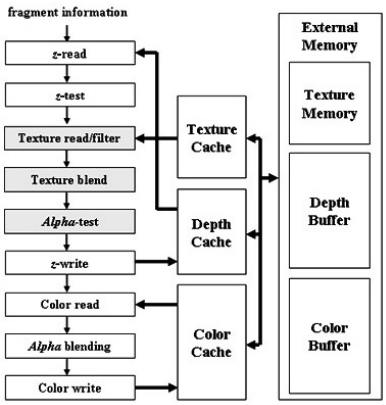
In this paper, we proposed an effective visibility test architecture with improving the mid-texturing architecture. The proposed architecture uses the property of fragments that the visibility of adjacent fragments is identical, and performs only a single visibility test per fragment. To compare with the mid-texturing architecture, simulation results show that the bandwidth requirements and the cell area of the depth cache in the proposed architecture are reduce by 25% and 34%, respectively, in exchange for less than 5% performance decline.

Keyword : Visibility, Depth-Test, Bandwidth Requirements, and Cell Area

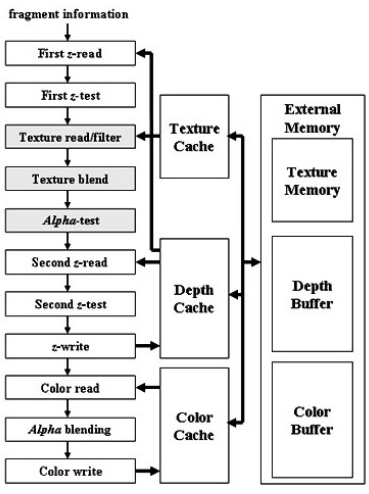
1. 서 론



(a) 선-텍스처링 구조



(b) 후-텍스처링 구조



(c) 중-텍스처링 구조

[그림1] 깊이 검사부의 위치에 따른 픽셀 파이프라인 구조

반도체 기술의 급속한 발전에 따라 휴대용 기기에서 실시간 3차원 그래픽 영상을 표현하기 위한 노력이 증대되고 있다[1][2]. 이러한 3차원 그래픽 영상은 많은 양의 폴리곤 및 텍스처 데이터를 요구하기 때문에 빠른 연산능력과 높은 메모리 대역폭[3]을 처리할 수 있는 전용 하드웨어가 필수적이다. 특히 저전력 구조가 필수적인 휴대용 기기에서 3차원 그래픽 전용 하드웨어를 사용하기 위해서는 낮은 전력소비를 갖는 특화된 구조의 연구가 필수적이다.

깊이 검사부는 픽셀 프래그먼트의 가시성을 결정하는 대표적인 처리장치로써, 기존에 발표된 대표적인 3차원 그래픽 가속기의 픽셀 파이프라인 구조는 깊이 검사부의 위치에 따라 선-텍스처링, 중-텍스처링, 그리고 후-텍스처링 구조로 나눌 수 있다[4]. 선-텍스처링 구조는 텍스처 작업을 수행한 후 깊이 검사를 수행하는 구조이고, 후-텍스처링 구조는 깊이 검사 이후에 텍스처 작업을 수행하는 구조이다.

그림 1-a)의 선-텍스처링 구조는 구현이 비교적 간단한 반면 보이지 않는 픽셀들도 텍스처 작업이 처리된다는 단점이 있고, 그림 1-b)의 후-텍스처링 구조는 보이지 않는 픽셀의 텍스처 작업은 수행되지 않지만, 파이프라인이 길어질수록 픽셀의 일관성(consistency) 유지를 위한 하드웨어의 부담이 커지는 단점이 있다.

그림 1-c)의 중-텍스처링 구조는 텍스처 작업 이전과 이후에 깊이 검사를 수행하는데, 첫 번째 깊이 검사는 보이지 않는 픽셀을 제거하여 불필요한 텍스처 작업을 줄이고, 두 번째 깊이 검사를 통해 픽셀의 일관성을 유지시켜 하드웨어의 부담을 감소시켰다. 또한, 다른 구조에서는 깊이 읽기가 실패이면 파이프라인 멈춤(pipeline stalls)이 발생하지만, 중-텍스처링 구조에서는 중-텍스처링 구조에서는 첫 번째 깊이 읽기가 실패하면 선인출(Prefetch)을 수행하면서 파이프라인이 계속 진행되기 때문에 깊이 캐쉬의 접근 실패에 따른 페널티가 크게 감소한다. 하지만 두 번의 깊이 검사의 수행으로 인하여 중-텍스처링 구조는 깊이 캐쉬의 요구량이 크게 증가하였고, 깊이 캐쉬에서 사용하는 포트의 수가 증가하기 때문에 깊이 캐쉬의 셀 크기 역시 증가하였다.

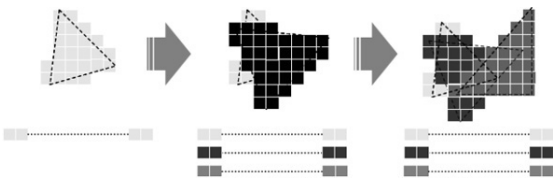
본 논문에서 제안하는 향상된 가시성 검사 구조는 인접한 프래그먼트들간의 가시성이 연속성을 보일 확률이 높은 점을 이용하여 깊이 읽기의 요구량 및 깊이 캐쉬의 포트 수를 감소시키면서 중-텍스처링 구조와 대등한 성능을 보이는

픽셀 파이프라인 구조이다. 이를 위해서 제안하는 구조에서는 이전 프래그먼트의 깊이 검사 결과를 z-tag 레지스터에 저장하고, 이를 참조하여 보여지지 않는 프래그먼트가 들어올 경우 첫 번째 깊이 검사를 수행하여 제거하고, 보여지는 프래그먼트들이 들어올 경우에는 두 번째 깊이 검사를 수행한다. 실험을 통해 제안하는 구조와 중-텍스처링 구조를 비교한 결과, *Quake3*[5]의 결과에서 제안하는 구조는 중-텍스처링 구조에 비하여 성능은 약 4%정도만 감소한 반면, 깊이 캐쉬의 요구량은 약 25%, 셀 면적은 약 34%가 감소하였다.

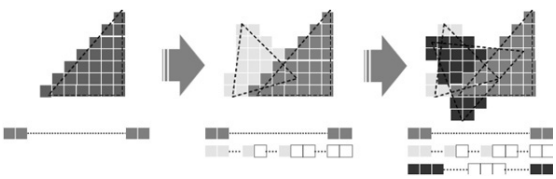
2. 제안하는 구조 및 처리과정

2.1 가시성의 연속성

폴리곤 기반의 3차원 그래픽 연산에서는 입력으로 들어온 세 정점의 정보를 격자화(raster)[6]하여 스팬을 생성하고, 각 스팬은 스크린 위치에 해당하는 프래그먼트로 처리되어 텍스처 작업 및 가시성 검사를 통해 프레임버퍼에 픽셀로 저장된다. 이때 가시성 검사는 프래그먼트 단위로 수행되며, 스팬 내부의 프래그먼트들간에는 가시성이 동일할 확률이 높다.



[그림 2] 정렬된 경우의 연속적 가시성

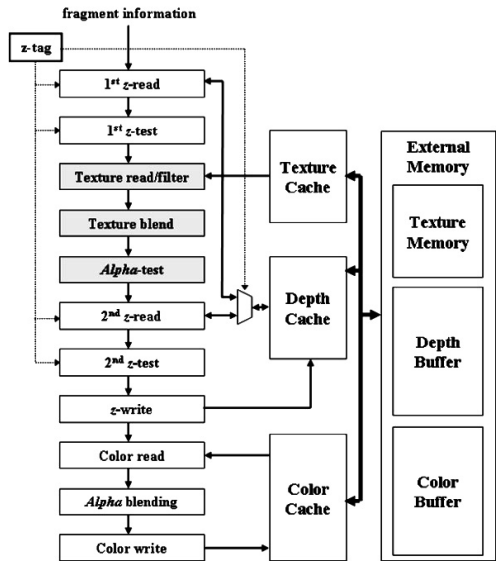


[그림 3] 정렬되지 않은 경우의 연속적 가시성

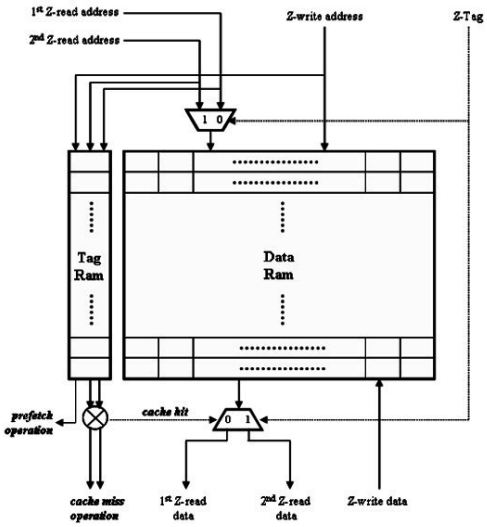
그림 2)와 그림 3)은 세 개의 삼각형을 그리는 예로써, 그려지는 폴리곤의 순서는 다르지만 최종적으로 프레임 버퍼에 그려지는 이미지는 같다. 그림 2)처럼 깊이 정보에 따라 먼 순서로 정렬된 폴리곤들을 그릴 경우에는, 모든 프래그먼트들은 가시성 검사에서 보여지는 것으로 판단되며, 가시성의 결과는 연속적이다. 그림 3)에서는 임의의 순서로 삼각형을 그리기 때문에 먼저 그려진 프래그먼트에 의해 나중에 그려지는 프래그먼트가 가려지는 경우가 발생한다. 이런 경우에는 가시성 검사의 결과가 변하지만, 가시성 검사의 결과가 변한 이후에는 다시 연속적으로 동일한 결과를 보인다. 실험결과, *Quake3*에서는 99.997%, *Viewperf*의 *Lightscape*에서는 99.974%의 연속성이 존재하였다.

2.2 제안하는 구조

그림 4)는 본 논문에서 제안하는 향상된 가시성 검사 구조의 픽셀 파이프라인과 그에 따른 깊이 캐쉬의 구조를 나타낸 그림이다. 제안하는 구조는 중-텍스처링 구조와 비슷하게 두 개의 깊이 검사부를 가지고 있지만, 중-텍스처링 구조와는 달리 한 번에 한 개의 깊이 검사부만 동작한다. 이는 2.1절의 가시성의 연속성을 이용한 것으로써, 보여지지 않는 프래그먼트들이 연속적으로 입력되는 경우에는 첫 번째 깊이 검사부를 동작시켜 파이프라인에서 제거하고, 보여지는 프래그먼트들이 연속적으로 입력되는 경우에는 선-텍스처링 구조와 같이 동작하면서, 첫 번째 깊이 읽기를 통해 처리중인 프래그먼트의 깊이 정보를 선인출 요청하여 두 번째 깊이 검사에서 발생하는 캐쉬 접근 실패를 감소시킨다. 이를 위해서 깊이 검사 결과를 z-tag 레지스터에 저장하고, 각 깊이 검사부는 이를 참조하여 동작여부를 판단한다.



(a) 제안하는 픽셀 파이프라인 구조



(b) 제안하는 깊이 캐쉬의 구조

[그림 4] 제안하는 픽셀 파이프라인 및 깊이 캐쉬의 구조

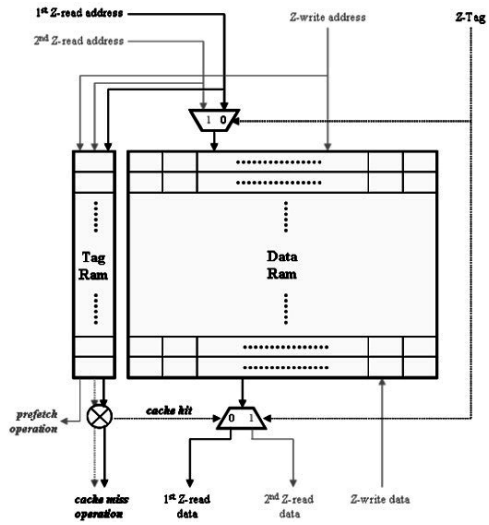
또한, 텍스처 캐쉬도 z-tag를 참조하여 현재 깊이 정보를 요청하는 깊이 검사부를 판단하여 그에 따른 캐쉬의 접근 성공/실패 처리 및 선인출 작업을 수행한다.

2.3 제안하는 구조의 처리과정

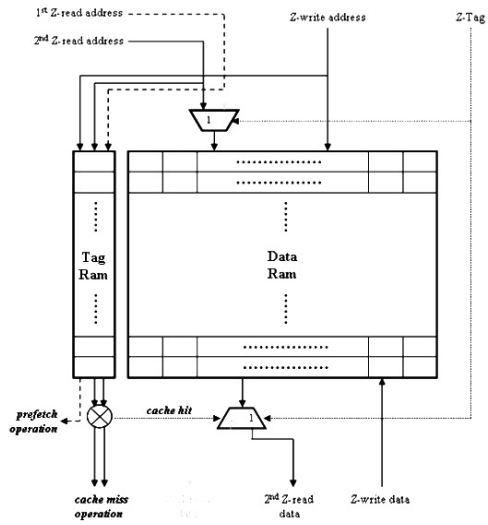
제안하는 구조의 동작은 크게 3가지로 나눌 수 있다. 첫 번째는 연속적으로 보여지지 않는 프래그먼트들이 입력되는 경우로써, z-tag에는 0이 저장된다. 두 번째는 연속적으로 보여지는 프래그먼트들이 입력되는 경우로써, z-tag에는 1이 저장된다. 마지막으로, 입력되는 프래그먼트의 가시성이 이전 프래그먼트와 다른 경우로써, z-tag의 값이 0에서 1로, 또는 1에서 0으로 변경되는 경우이다.

2.3.1 z-tag가 0일 때

그림 5)는 z-tag의 값이 0인 경우로서, 이전 프래그먼트가 보여지지 않았기 때문에 현재 프래그먼트 역시 보여지지 않을 확률이 높은 경우이다. 따라서 제안하는 구조는 첫 번째 깊이 검사부를 동작시켜 프래그먼트의 가시성을 검사한다. 만약 이전 깊이 검사 결과와 마찬가지로, 깊이 검사의 결과가 실패이면 해당 프래그먼트는 보여지지 않기 때문에 파이프라인에서 제거되어 불필요한 텍스처 작업이 제거되며, z-tag는 계속 0을 유지하면서 다음 프래그먼트를 처리한다. 만약 깊이 검사의 결과가 성공이면, 해당 프래그먼트는 보여지기 때문에 z-tag의 값이 0에서 1로 변경되는데, 이는 2.3.3절에서 설명하였다.



[그림 5] z-tag가 0인 경우 깊이 캐쉬의 동작



[그림 6] z-tag가 1인 경우 깊이 캐쉬의 동작

2.3.2 z-tag가 1일 때

그림 6)은 z-tag의 값이 1로써, 이전 프래그먼트가 보여졌기 때문에 현재 프래그먼트 역시 보여질 확률이 높은 경우이다. 따라서 이 경우에는 제안하는 구조는 텍스처 작업이 수행된 후 두 번째 깊이 검사부에서 프래그먼트의 가시성을 검사한다. 이때 첫 번째 깊이 검사부의 깊이 읽기는 깊이

캐쉬로 선인출을 요청하여 두 번째 깊이 검사부의 깊이 읽기에 따른 캐쉬 접근 실패를 감소시킨다. 이를 위해 깊이 캐쉬의 tag-ram은 선인출, 깊이 읽기, 그리고 깊이 쓰기를 동시에 수행하기 위하여 3-port SRAM을 사용하는데, tag-ram은 주소비교를 위한 tag 정보만 저장하기 때문에 전체 깊이 캐쉬의 셀 크기에 큰 영향이 없다.

만약 깊이 검사 결과가 실패여서 현재 프래그먼트가 보여지지 않는다면, z-tag의 값을 1에서 0으로 변경한 후 파이프라인을 진행하는데, 이는 2.3.3절에서 설명하였다.

2.3.3 z-tag의 값이 변경될 때

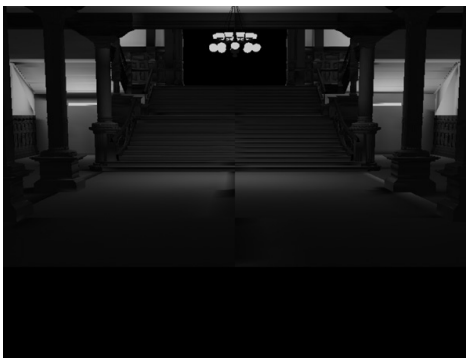
z-tag의 값이 변경되는 경우는 두 가지가 있는데, 첫 번째는 0에서 1로 변경되는 경우이다. 이 경우에는 멈춤 없이 파이프라인을 진행하여도 프래그먼트의 가시성이 두 번째 깊이 검사부에서 판단되기 때문에 일체성의 문제가 발생하지 않는다.

두 번째는 z-tag의 값이 1에서 0으로 변경되는 경우이다. 이 경우에는, 위의 경우와는 달리 파이프라인 멈춤이 발생하는데, 이는 두 깊이 검사부 사이에 진행중인 프래그먼트들이 존재하기 때문이다. 만약 파이프라인 멈춤 없이 z-tag를 1에서 0으로 변경시켜 첫 번째 깊이 검사부에서 가시성을 검사하도록 제어를 변경하면, 두 깊이 검사부 사이에 진행중인 프래그먼트들은 가시성이 판단되지 않기 때문에 일체성에 문제가 발생하게 된다. 따라서, z-tag의 값이 1에서 0으로 바뀌는 경우에는, 첫 번째 깊이 검사부 이 전의 프래그먼트들은 멈추게 되고, 두 깊이 검사부 사이에 진행중인 모든 프래그먼트들이 두 번째 깊이 검사부에서 가시성이 판단되면 z-tag를 1에서 0으로 변경한 후 멈추어진 프래그먼트들의 처리를 진행시킨다. 이때 발생하는 파이프라인 페널티는 두 깊이 검사간의 파이프라인 수 및 z-tag가 1에서 0으로 변경되는 횟수에 비례하는데, 이에 따른 성능은 3.3절에서 분석하였다.

3. 실험 환경 및 성능 분석

중-텍스처링 구조와 제안하는 구조의 성능을 측정하기 위하여, 우리는 3차원 연산을 통해 화면을 구성하는 벤치마크들을 실행하여 얻어진 트레이스를 통해 캐쉬 시뮬레이션 및 성능분석을 수행하였다. 깊이 정보의 트레이스를 얻기 위해서, 우리는 리눅스 RedHat 7.1[7] 운영체제하에서 널리 알려진 OpenGL 호환 API인 Mesa3D[8]를 수정하여 사용하였고, Dinero IV[9]를 사용하여 캐쉬 시뮬레이션을 수행하였다. 사용된 벤치마크들로는, 널리 알려진 3차원 게임인 *Quake3*와 OpenGL의 성능을 측정하기 위한 벤치마크인 *SPECviewperf*[10]중에서 *Lightscape*를 사용하였다.

(a) *Quake3*

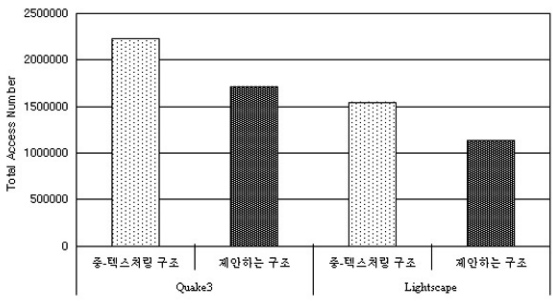


(b) *Lightscape* of *SPECviewperf*

[그림 7] 실험에 사용된 벤치마크들

3.1 깊이 캐쉬의 접근 회수

현재 프래그먼트의 가시성을 판단하기 위한 깊이 검사를 수행하기 위해서는 깊이 캐쉬로 읽기 및 쓰기가 수행된다. 보여지지 않는 프래그먼트의 경우에는 제안하는 구조와 중-텍스처링 구조 모두에서 깊이 읽기만 수행된다. 하지만 보여지는 프래그먼트의 경우에는, 중-텍스처링 구조에서는 두 번의 깊이 읽기와 한 번의 깊이 쓰기가 수행되는 반면, 제안하는 구조에서는 한 번의 깊이 읽기와 한 번의 깊이 쓰기만 수행된다. 이를 반영하여 두 구조에서 요구되는 깊이 캐쉬의 접근 횟수를 측정한 결과는 그림 8)과 같다.



[그림 8] *Quake3*와 *Lightscape*에서 깊이 캐쉬의 접근 횟수

그림 8)은 *Quake3*와 *Lightscape*에서 프레임당 평균 접근 횟수를 측정한 결과이다. 실험결과에서, 제안하는 구조의 캐쉬 접근 횟수가 중-텍스처링 구조에 비해 적음을 알 수 있는데, 이는 앞에서도 언급한 바와 같이, 보여지는 프래그먼트에 대한 제안하는 구조의 깊이 읽기 횟수가 중-텍스처링 구조와 비교하여 절반으로 감소했기 때문이다. 전체 접근 횟수에서, 제안하는 구조는 *Quake3*에서는 약 23%, 그리고 *Lightscape*에서는 약 27% 정도가 중-텍스처링 구조에 비하여 감소하였다.

3.2 포트 수에 따른 깊이 캐쉬의 셀 면적

깊이 캐쉬를 구성하는데 사용되는 메모리는 크게 tag-ram과 data-ram으로 나눌 수 있는데, tag-ram은 주소 비교를 위한 tag 정보만 저장하기 때문에 깊이 정보가 저장된 data-ram에 비하여 크기가 아주 작다. 따라서 캐쉬의 셀 면적은 대부분 data-ram의 크기에 영향을 받는다. 또한 캐쉬의 메모리에 사용되는 SRAM의 셀 크기는 데이터 입출력을 위한 포트의 수에 영향을 받는다. 기존의 연구[11]에 따르면, SRAM 포트의 수가 1개 증가하면, 데이터 셀의 면적은 약 45%, 그리고 디코딩과 라우딩을 위한 면적이 약 15%정도 증가한다고 알려져 있다. 따라서 포트 수의 증가에 따른 셀 면적의 증가비율은 다음과 같이 구할 수 있다[11].

$$\text{Ratio of Cell Area} = 1+(0.6\ (N-1)),$$

이때 N은 포트의 수이며, 따라서 1개의 포트를 사용하는 SRAM의 면적은 1이 된다. 이를 2-port를 사용하는 일반적인 깊이 캐쉬와 3개의 포트를 사용하는 중-텍스처링 캐쉬, 그리고 3-port tag-ram과 2-port data-ram을 사용하는 제안하는 캐쉬의 면적 비율을 계산하였는데, 그 결과는 표 1)과 같다.

	2-port 캐쉬	중-텍스처링 캐쉬	제안하는 캐쉬
면적 비율	1.6	2.2	1.643

[표 1] 포트 수에 따른 깊이 캐쉬의 면적 비율

표 1)의 결과에서, 2-port를 사용하는 일반적인 깊이 캐쉬의 면적비율이 가장 낮았으며, 중-텍스처링 캐쉬는 3포트를 사용하기 때문에 면적비율이 가장 높았다. 제안하는 구조는 캐쉬 면적의 대부분을 차지하는 data-ram에 2-port를 사용하기 때문에 2-port 캐쉬와 비슷한 면적을 보였다

3.3 성능분석

선-텍스처링 구조, 제안하는 구조, 그리고 중-텍스처링 구조의 성능을 측정하기 위하여, 우리는 프래그먼트당 소모되는 평균 사이클 수 (*ACPF, Average Cycles Per Fragment*)를 계산하였다. 3차원 그래픽 픽셀 파이프라인에서 발생하는 성능저하는 텍스처, 깊이, 그리고 컬러 캐쉬의 접근 실패에 의해 발생하는 페널티가 가장 큰 요인이다. 이중 컬러 캐쉬의 접근실패율은 세 구조가 동일하여 성능차이를 반영하지 못하기 때문에 *ACPF*에서 제외되었다. 먼저 선-텍스처링 구조에 대한 *ACPF* 계산식은 다음과 같다[4].

$$ACPF_{pre}=H_{pix}H_{tex}+(1-H_{pix})\,T_{pix}+(1-H_{tex})\,T_{tex},$$

여기서 H_{pix}와 H_{tex}는 깊이 캐쉬와 텍스처 캐쉬의 접근성공률이고 T_{pix}와 T_{tex}는 픽셀캐쉬와 텍스처 캐쉬의 접근 실패에 따른 페널티이다. 픽셀캐쉬와 텍스처 캐쉬가 모두 접근 성공이면 $ACPF_{pre}$=1이 되고, 깊이 캐쉬가 접근 실패이면 $ACPF_{pre}=T_{pix}$, 텍스처 캐쉬가 접근 실패이면 $ACPF_{pre}=T_{tex}$가 된다.

다음으로 중-텍스처링의 *ACPF* 계산식은 다음과 같다[4].

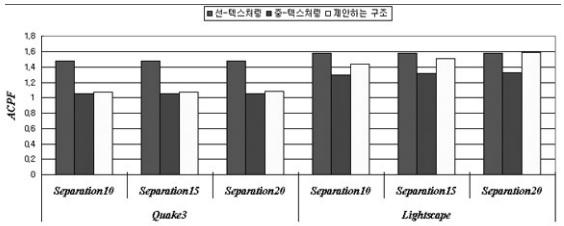
$$\begin{aligned}ACPF_{mid}= {} & H_{pix}\times\gamma+H_{pix}\times(1-\gamma)\times \\ & (H1_{pix}\times H_{tex}+(1-H1_{pix})\times T_{pix}+ \\ & (1-H_{tex})\times T_{tex})+ \\ & (1-H_{pix})\times((1-H_{tex})\times T_{tex}+ \\ & T_{pix}\times(1-reduction)),\end{aligned}$$

이때 γ은 첫 번째 깊이 검사에 의해 제거되는 비율이고, $H1_{pix}$=1-$M1_{pix}$인데 $M1_{pix}$는 두 깊이 검사간의 파이프라인 깊이에 따른 깊이 검사 상이에 따른 비율, 그리고 reduction은 두 깊이 검사간의 파이프라인 깊이에 따른 깊이 캐쉬의 감소율을 의미한다.

마지막으로 제안하는 구조의 *ACPF*는 다음과 같다.

$$\begin{aligned} ACPF_{proposed} = & H_{pix} \times \gamma + H_{pix} \times (1-\gamma) \times \\ & (H1_{pix} \times H_{tex} + (1-H1_{pix}) \times T_{pix} + \\ & (1-H_{tex}) \times T_{tex}) + \\ & (1-H_{pix}) \times ((1-H_{tex}) \times T_{tex} + \\ & T_{pix} \times (1-reduction)) + P_{sf}, \end{aligned}$$

제안하는 구조의 *ACPF*는 $ACPF_{mid}$에 P_{sf}를 더하여 구할 수 있다. P_{sf}는 2.3.3절에서 언급된 z-tag가 1에서 0으로 변경될 때 발생하는 페널티로써, Separation N_{tag}로 구해지는데, Separation은 두 깊이 검사부 사이에 존재하는 파이프라인 수이고, Ntag는 z-tag의 값이 1에서 0으로 변경되는 횟수이다.



[그림 9] 세 구조간 ACPF 결과

그림 9)은 *Quake3*와 *Lightscape*에서의 Separation에 따른 세 구조간 ACPF를 계산한 결과로써, 낮을수록 높은 성능을 의미한다. 두 깊이 검사간의 파이프라인 증가에 따른 영향을 비교하기 위하여 Separation은 10, 15, 그리고 20으로 가정하였고, 깊이 캐쉬 및 텍스처 캐쉬는 모든 구조에서 동일한 파라메터인 direct-mapped 방식의 16Kbyte의 캐쉬 크기와 64 byte의 블록 크기로 실험하였다.

실험결과, 중-텍스처링이 가장 낮은 *ACPF*를 보였으며, 다음으로 제안하는 구조의 *ACPF*가 낮았고, 선-텍스처링 구조의 *ACPF*가 가장 높았다. *Quake3*에서, 제안하는 구조의 *ACPF*는 Separation의 증가에 따른 변화가 거의 없는 반면, *Lightscape*에서는 Separation이 증가할수록 제안하는 구조의 *ACPF*가 크게 증가하였다. 이렇게 벤치마크에 따라 *ACPF*의 변화량에 차이가 나는 이유는 벤치마크에서 수행하는 화면구성(scene management) [12]에 의해 2.1절에서 설명한 화면에 그려지는 사물의 순서에 따라 스팬 내부의 가시성 변화가 달라지기 때문이다. *Quake3*에서는 화면구성을 통한 최적화를 수행하여 N_{tag}가 낮았지만, *Lightscape*는 OpenGL 처리능력을 측정하기 위하여 화면구성 없이 raw 데이터를 화면에 그리기 때문에 *Quake3*에 비해 N_{tag}가 크게 증가하였다. 따라서 P_{sf} 페널티 역시 크게 증가하였고, Separation이 20인 경우에는 선-텍스처링 구조와 비슷할 정도로 ACPF가 증가하였다. 일반적으로, 대부분의 3차원 응용프로그램들은 성능을 높이기 위해 자체적으로 화면구성을 수행하여 화면에 그려지는 사물을 최적화하기 때문에, 이러한 응용프로그램들에서 제안하는 구조의 성능은 *Quake3*의 결과와 같이 중-텍스처링 구조와 비슷한 성능을 보일 것이다.

4. 결 론

본 논문에서 우리는 기존의 고성능 3차원 픽셀 파이프라인 구조인 중-텍스처링 구조를 개선한 향상된 가시성 검사를 수행하는 픽셀 파이프라인 구조를 제시하였다. 제안하는 구조는 중-텍스처링 구조에 비하여, 깊이 캐쉬의 접근 횟수는 *Quake3*에서는 약 23%, *Lightscape*에서는 27%를 감소시켰고, 깊이 캐쉬의 셀 면적은 중-텍스처링 구조의 깊이 캐쉬에 비해 약 40%가 감소하였다. 또한 상용 3차원 게임엔진인 *Quake3*에서는 중-텍스처링에 근접한 성능을 보였다.

현재 우리는 Verilog HDL로 제작한 표준 3차원 그래픽 파이프라인의 RTL을 FPGA를 통해 검증하였으며, 이를 바탕으로 본 논문에서 제안한 픽셀 파이프라인 구조의 RTL을 FGPA를 통해 검증할 예정이다.

참고문헌

[1] ATI, available at http://www.ati.com/companyinfo/press/2004/4719.html.

[2] nVidia, available at http://www.nvidia.com/page/goforce_3d_4500.html.

[3] nVidia, "Technical Brief: AGP 4X with Fast Writes," nVidia Corporation, available at http://developer.nvidia.com/object/Technical_Brief_AGP_4X.html.

[4] Woo-Chan Park, Kil-Whan Lee, Il-San Kim, Tack-Don Han, and Sung-Bong Yang, "An Effective Pixel Rasterization Pipeline Architecture for 3D Rendering Processors," IEEE Transactions on Computers, Vol. 52, No. 11, pp. 1501-1508, Nov. 2003.

[5] http://www.idsoftware.com/games/quake/quake3-arena/

[6] A. Kugler, "The setup for triangle rasterization," Proceedings of 11th Eurographics Workshop on Computer Graphics Hardware, Aug. 1996, pp. 49?58.

[7] Redhat, http://www.redhat.com/

[8] http://www.mesa3d.org/

[9] Jan Edler, Mark D. Hill, available at http://www.cs.wisc.edu/~markhill/DineroIV/

[10] Standard Performance Evaluation Corporation, available at http://www.spec.org/

[11] H. J. Mattausch, "Hierarchical n-port memory architecture based on 1-port memory cells,'' Proceedings of the 27th European Solid-State Device Research Conference, pp. 348-351, Sep. 1997.

[12] L. Bishop, D. Eberly, T. Whitted, M. Finch, and M. Shantz, "Designing a PC Game Engine," IEEE Computer Graphics and Applications, vol. 18, no. 1, pp. 46-53, Jan. 1998.

김 일 산 (Il-san Kim)

연세대학교 컴퓨터과학과에서 2000년, 2002년 각각 학사 및 석사학위를 수료. 현재 동대학원에서 박사과정을 이수중이며, 연구분야로는 3차원 그래픽 프로세서 및 메모리 구조에 대한 연구를 수행 중.

박 우 찬 (Woo-Chan Park)

연세대학교 컴퓨터과학과에서 1993년, 1995년, 2000년에 각각 학사, 석사, 박사학위 수료. 2001년부터 2003년까지 동대학원에서 연구교수로 재직하였으며, 현재 세종대학교 컴퓨터공학과에서 교수로 재직 중. 연구분야로는 그래픽 하드웨어 및 렌더링 프로세서에 대한 연구를 진행 중.

박 진 홍 (Jin-Hong Park)

연세대학교 컴퓨터과학과에서 2000년 2002년에 각각 학사 및 석사학위 수료. 현재 동대학원에서 박사과정을 이수중이며 연구분야로는 3차원 그래픽 프로세서 및 메모리 구조에 대한 연구를 수행 중.

한 탁 돈 (Tack-Don Han)

연세대학교 전자공학과에서 1978년에 학사학위 수료. Wayne 주립대학교에서 1983년 석사학위 수료. 그리고 1987년 메사추세츠 공대에서 박사학위 수료. 현재 연세대학교 컴퓨터과학과에서 정교수로 재직중이며 연구분야로는 고성능 컴퓨터구조, 미디어 시스템 구조, HCI (Human Computer Interface)에 대해 연구를 진행 중.