RUF 버퍼를 이용한 간단하고 효율적인 안티알리아싱 기법*

(A Simple and Efficient Antialiasing Method with the RUF buffer)

김 병 욱 † 박 우 찬 †† 양 성 봉 ††† 한 탁 돈 ††††

(Byung-Uck Kim) (Woo-Chan Park) (Sung-Bong Yang) (Tack-Don Han)

요 약 본 논문은 전형적인 슈퍼샘플링과 거의 동일한 수준의 고화질 영상을 생성하는 동시에, 요구되는 메모리 크기와 메모리 대역폭을 줄일 수 있는 간단하고 효율적인 하드웨어 지원 안티알리아싱 알고리즘과 렌더링 구조를 제안한다. 본 논문에서는 가장 최근에 색상 값 결정을 위해 사용된 프레그먼트의 일부분 또는 병합된 결과를 저장하는 RUF (Recently Used Fragment) 버퍼와 RUF 버퍼의 정보를 이용하여 효과적으로 색상 값을 결정하는 알고리즘을 제안한다. 제안된 방법은 데이타 구조상 샘플링 포인트 수가 늘어날수록 슈퍼샘플링에 비해 메모리 절약 효과가 크다. 또한 본 논문의 실험결과 8산개(sparse) 샘플링 포인트를 가지는 경우, 슈퍼샘플링에 비해 제안된 안티알리아싱 기법은 약 1.3%의 색상 차이를 가지나, 렌더링 과정에서 요구되는 메모리 크기가 약 31%로 감소하였으며, 실험에 사용된 3차원 모델에 대해 평균 11%의 메모리 대역폭 감소를 보인다.

키워드 : 안티알리아싱, 슈퍼샘플링, 메모리 크기, 메모리 대역폭, RUF(Recently Used Fragment) 버퍼

Abstract In this paper, we propose a simple and efficient hardware-supported antialiasing algorithm and its rendering scheme. The proposed method can efficiently reduce the required memory bandwidth as well as memory size compared to a conventional supersampling when rendering 3D models. In addition, it can provide almost the same high quality scenes as supersampling does. In this paper, we have introduced the RUF (Recently Used Fragment) buffer that stores some or whole parts of a fragment or two more the merged results of fragments that recently used in color calculation. We have also proposed a color calculation algorithm to deteriorate the image quality as referencing the RUF buffer. Because of the efficiency presented in the proposed algorithm, the more number of sampling points increases the more memory saving ratio we can gain relative to the conventional supersampling. In our simulation, the proposed method can reduce the amount of memory size by 31% and the memory bandwidth by 11% with a moderate pixel color difference of 1.3% compared to supersampling for 8 sparse sampling points.

Key words : Antialiasing, supersampling, memory size, memory bandwidth, RUF buffer

1. 연구 소개

3차원 그래픽에서 알리아싱 문제는 대부분 부족한 샘플링에 의해 발생한다. 일반적으로 알리아싱 문제를 해결하기 위해서 보다 높은 해상도에서 3차원 물체(object)를 렌더링 한 후, 실제 화면 해상도에 맞추어 평균-하위 필터링(average-down filtering)을 수행하는 슈퍼샘플링 방식을 사용한다[1]. 그러나 이러한 방식은 커다란 메모리 크기와 메모리 대역폭을 필요로 한다. 예를 들어 n-슈퍼샘플링은 1-포인트 샘플링 방식보다 n배의 메모리 크기와 메모리 대역폭을 각각 필요로 한다. 이러한 메모리 요구량을 줄이기 위해 하나의 화면에 대하여 여러 번에 걸쳐 렌더링을 수행한 후 각각의 결과를 축적 버퍼

*이 논문은 2001년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KFR-2001-003-E00264)

† 비 회 원 : 연세대학교 대학원 컴퓨터과학과
kimbu@mythos.yonsei.ac.kr
†† 비 회 원 : 연세대학교 교수
chan@kurene.yonsei.ac.kr
††† 비 회 원 : 연세대학교 공과대학 컴퓨터과학과 교수
yang@cs.yonsei.ac.kr
†††† 종신회원 : 연세대학교 공과대학 컴퓨터과학과 교수
hantack@kurene.yonsei.ac.kr

논문접수 : 2002년 6월 19일
심사완료 : 2003년 1월 14일

(accumulation buffer)에 저장한 후, 그 결과에 적당한 가중치를 두어 프래임버퍼에 저장함으로써 최종 화면을 생성하는 기법이 제안되었다[2]. 그러나 이러한 기법은 하나의 화면에 대해 렌더링 수행을 여러 번 반복하는 다중 패스(multi-pass) 알고리즘이므로 렌더링 속도가 반복하는 회수에 비례하여 느려진다. 슈퍼샘플링과 축적 버퍼 기법은 은면제거(hidden surface removal)를 위해 대부분의 렌더링 시스템에서 사용하는 Z 버퍼 알고리즘과 잘 결합된다.

한편, 불투명 물체에 대한 안티알리아싱 뿐만 아니라 투명 물체의 블렌딩(blending)을 지원하는 A-buffer 알고리즘이 제안되었다[3]. A-buffer 알고리즘은 영역마스크(coverage mask)를 이용하여 부픽셀(subpixel)이 차지하는 영역을 나타내며, 이 영역이 가지는 평균 색상 값으로 프레그먼트를 표현한다. 이러한 방식은 부픽셀별로 색상 값이 요구되는 슈퍼샘플링 방식에 비해, 색상 값의 중복성을 줄일 수 있다. 하지만, 여러 물체의 폴리곤 프래그먼트가 중첩하거나 서로 교차(intersection)하는 관계를 나타내기 위해서는 픽셀 당 깊이 값에 의해 정렬된 프레그먼트 리스트가 필요하다. 이러한 리스트는 렌더링 파이프라인 설계를 어렵게 한다. 또한 A-buffer알고리즘이 비록 투명 물체를 잘 처리할 수 있지만, 여전히 불투명 물체의 안티알리아싱을 위해서 프레그먼트 리스트가 필요로 하며, 경우에 따라서 슈퍼샘플링 방식보다 더 많은 하드웨어 비용을 요구할 수 있다.

본 논문에서는 A-buffer 알고리즘에 기반을 둔 영역마스크를 이용하여 프래그먼트를 표현하며, 가장 최근에 사용된 프래그먼트 색상 값을 이용하여 불투명 물체에 대해서 효율적으로 안티알리아싱을 수행할 수 있는 알고리즘과 렌더링 구조를 제안한다. n-슈퍼샘플링이 픽셀 당 n 개의 부픽셀을 가지며, 각 부픽셀별로 하나의 색상 값과 깊이 값을 요구하는데 반해, 영역마스크를 통한 프레그먼트 표현은 샘플링 포인트 별로 중복되는 색상 값을 제거할 수 있으므로 메모리 크기와 메모리 대역폭을 줄일 수 있다. 하지만, 여러 개의 프레그먼트가 하나의 픽셀에 중첩되거나 서로 교차하는 경우, A-buffer 알고리즘에서처럼 가변 길이의 프레그먼트 리스트가 필요하다.

이러한 문제점을 완화하기 위해, 본 논문에서는 가장 최근에 색상 버퍼에 영향을 준 프레그먼트의 일부분 또는 병합한 결과를 저장하는 RUF(Recently Used Fragment) 버퍼를 제안하며, RUF 버퍼를 이용하여 색상 값을 계산하는 알고리즘을 제안한다. 색상 버퍼에 축적된 프레그먼트들의 일부분이 현재 프레그먼트에 의해 가려져 은면제거 되는 경우, RUF 버퍼의 값을 참조하여 발생할 수 있는 색상 값 에러를 보정한다.

본 논문에서는 제안된 방식과 다양한 샘플링 포인트를 가지는 슈퍼샘플링에 대해 필요한 메모리 크기, 대역폭 그리고 화질 차이를 비교 실험하였다. 실험결과, 제안된 방식은 샘플링 포인트가 늘어날수록 메모리 절약 효과가 슈퍼샘플링에 비해 우수하며, 메모리 대역폭도 감소한다. 또한, 다양한 3차원 모델에 대해 실험한 결과, 8개의 산개 샘플링 포인트를 가지는 경우, 슈퍼샘플링과 1.3%의 화질 차이를 가지나, 메모리 크기와 메모리 대역폭 실험에서는 각각 31%와 11%의 감소를 보인다.

본 논문은 다음과 같이 구성되어있다. 먼저, 2장에서 프레그먼트 구조, 프레임버퍼 구조 그리고 RUF 버퍼 구조를 설명하며, 제안된 알고리즘과 렌더링 구조에 대해 설명한다. 3장에서는 제안된 알고리즘과 슈퍼샘플링의 화질, 요구되어지는 메모리 크기 그리고 메모리 대역폭을 실험하며 그 결과를 제시한다. 마지막으로, 4장에서는 제안된 방식에 대해 결론을 내린다.

2. 제안된 렌더링 구조

일반적인 폴리곤기반(polygon based) 렌더링 시스템에서, 3D 모델은 지오메트리(geometry) 연산을 수행한 후, 렌더링 파이프라인으로 입력된다. 렌더링 파이프라인에서는 스캔컨버전(scan conversion)을 통해 입력 폴리곤에 대한 프레그먼트를 생성하며, 깊이 비교와 다양한 이미지 매핑을 통해 최종 색상 값을 프레임버퍼에 저장한다. 모든 프레그먼트에 대해 렌더링 과정이 끝나면, 프레임버퍼에 저장된 색상 값을 디스플레이 장치로 내보낸다. 이 장은 제안된 안티알리아싱 렌더링 구조와 알고리즘을 위한 프레그먼트, 프레임버퍼, 그리고 RUF 버퍼의 데이타 구조를 설명한다.

2.1 데이타 구조

제안된 알고리즘의 프레그먼트 구조는 기본적으로 A-buffer 알고리즘에 기반을 두고 있다. 프레그먼트는 픽셀의 어떠한 부분을 커버하고 있는지를 나타내는 영역마스크 (m), 프레그먼트의 색상 값 (C), 프레그먼트가 차지하고 있는 부픽셀별 깊이 값 ($Z_1 \cdots Z_m$), 그리고 RUF 버퍼에서 프래그먼트를 병합할 때 플래그로 사용하는 물체 태그 (ObjTag)로 표현된다. 물체 태그는 3차원 모델 물체가 가지는 유일한 값이며, 모델링 소프트웨어와의 연동을 통해 렌더링 하드웨어에서 순차적으로 생성될 수 있다[4]. 만약 두 프레그먼트가 같은 물체 태그를 가진다면, 두 프레그먼트는 같은 물체 폴리곤에서 생성되었음을 의미한다. 그림 1은 8 산개 샘플링 포인트를 가지는 경우 샘플링 포인트와 프레그먼트의 데이타 구조를 보여준다.

그림에서, 영역마스크는 m-bit으로 표현되며, 그림의 예제에서는 8개의 샘플링 포인트를 가지므로 영역마스크는 8비트이다. 색상 값 C와 깊이 값 Z는 각각 32비트로 표현된다. 본 논문에서는 물체 태그를 [4]에서와 같이 16비트로 가정한다.

그림 2는 본 논문에서 제안하는 렌더링 구조상에서 프레임버퍼와 RUF 버퍼 구조를 보여준다. 프레임버퍼는 픽셀의 색상 값을 저장하는 색상 버퍼와 깊이 값을 저장하는 깊이 버퍼로 구성된다. RUF 버퍼는 은면제거 과정을 통과한 프레그먼트 부분, 즉 가장 최근에 색상 버퍼의 색상 값 계산에 사용된 프레그먼트 부분에 대한 물체 태그, 색상 값, 그리고 영역마스크를 저장하며, 이러한 프레그먼트들이 같은 물체 태그를 가지는 경우, 프레그먼트들은 RUF 기록과정에서 병합된다.

2.2 제안된 알고리즘 및 렌더링 과정

그림 3은 본 논문에서 제안한 안티알리아싱을 위한 렌더링 파이프라인의 개략적인 모습이며, 은면제거 검사, 프레임버퍼 저장과 색상 값 결정, 그리고 RUF 버퍼 기록의 3단계 과정으로 이루어져 있다.

다음은 제안하는 구조에 대한 각 단계 별 과정을 설명한다. 제안된 알고리즘의 설명을 용이하게 하기 위해 렌더링 파이프라인에 새로 들어온 프레그먼트, 프레임에 저장된 프레그먼트, 그리고 RUF 버퍼에 기록되어 있는 프레그먼트의 속성 정보에 대해 각각 if, ff, 그리고 RUF를 아래첨자로 사용하여 표현한다. 예를 들어, M_{if}, C_{ff}, C_{RUF}는 각각 렌더링 파이프라인에 새로 들어온 프레그먼트의 영역마스크, 프레임버퍼 안에 기록된 색상 값, 그리고 RUF 버퍼에 기록된 프레그먼트의 색상 값을 의미한다.

은면제거 검사 과정: 새로 들어온 프레그먼트와 프레임버퍼에 저장된 프레그먼트의 각 부픽셀 별 Z값 비교를 수행한다. 부픽셀 별 Z값 비교는 기존의 Z buffer 기법과 같다. 은면제거 테스트를 통과한 프레그먼트 부분을 '*생존 프레그먼트*' (survived fragment, 아래첨자 sur)라 명칭하며, 이 부분이 가지는 부픽셀에 대한 영역마스크 M_{sur}는 식(1)을 통해 구할 수 있다.

$$M_{sur} = M_{if} \text{ if and only if } Z_{if(i)} < Z_{ff(i)} \quad (1)$$

식에서, 아래첨자 (i)는 샘플링 포인트의 위치, 즉 각 부픽셀의 위치를 나타낸다.

프레임버퍼 저장과 색상 값 결정 과정: 생존 프레그먼트는 렌더링 파이프라인에 새로 들어온 프레그먼트 중 사용자에게 보이는 부분이다. 따라서 이 부분의 영역마스크, 깊이 값, 그리고 색상 값은 프레임버퍼에 추가되어야 한다. 새로운 영역마스크는 프레임버퍼에 저장된 영역마스크(M_{ff})와 생존 프레그먼트의 영역마스크 (M_{sur})의 OR 비트연산자로 계산되며, 생존 프레그먼트의 부픽셀이 가지는 깊이 값으로 깊이 버퍼를 갱신한다.

색상 값 결정 과정은 생존 프레그먼트의 색상 값을 색상 버퍼에 더하며, 생존 프레그먼트에 의해 은면제거 되

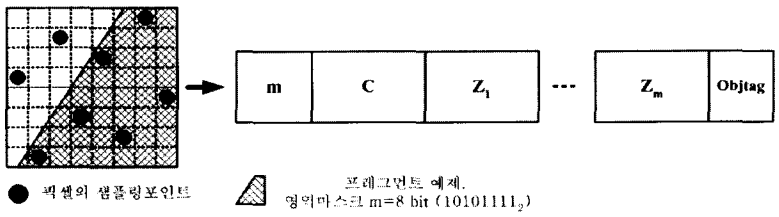


그림 1 프레그먼트 구조

Struct framebuffer[X][Y]
{
Struct color buffer {
Color C;
CoverageMask m; }
Struct depth buffer {
Depth ZI,…,Zm;}
}
	Struct RUF buffer[X][Y]
	{
	ObjectTag Objtag;
	CoverageMask m;
	Color R,G,B,A;
	}

(a) 프레임버퍼	(b) RUF 버퍼

그림 2 픽셀 좌표 [X][Y]를 위한 프레임버퍼와 RUF 버퍼의 데이타 구조

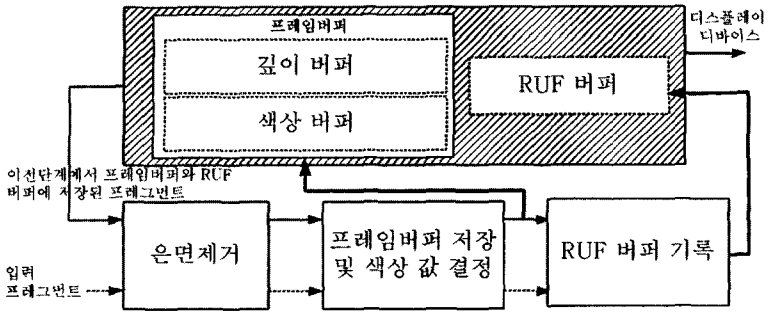


그림 3 안티알리아싱을 위해 제안된 렌더링 파이프라인 개략도

는 부분, 즉 가려지는 부분의 색상 값을 색상 버퍼에서 제거한다. 그림 4는 랜더링 과정 중에 발생할 수 있는 프레그먼트들의 상호관계와 보이는 부분에 대해 나타낸다. 그림 4 (a)와 (b)는 하나의 프레그먼트가 다른 프레그먼트의 전부 혹은 일부를 가리는 경우이며, 그림 4 (c)와 (d)는 프레그먼트들이 서로 관통하는 예이다.

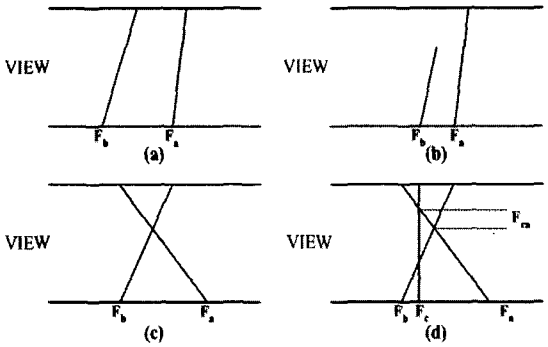


그림 4 프레그먼트들의 관계와 보이는 부분: 그림에서 랜더링 파이프라인에 입력되는 프레그먼트 순서는 F_a, F_b, F_c 이다.)

앞에서 언급한 것처럼 RUF 버퍼는 가장 최근에 프레임버퍼의 색상 값에 영향을 준, 생존 프레그먼트의 정보를 저장하며, 이 프레그먼트는 가려지는 부분의 색상 값을 구하기 위해서 재사용된다. 그림 4 (a), (b), (c)의 경우 렌더링 파이프라인에 들어오는 프레그먼트 순서를 고려하면, RUF 버퍼에 저장된 프레그먼트는 F_a이며, 이 값을 참조하여 프레그먼트 F_b에 의해 은면제거 되는 부분의 색상 값을 알 수 있다. 이러한 경우, 색상 값 은 식 (2)를 이용하여 계산할 수 있다. 즉, 현재의 색상 버퍼에 프레그먼트 F_b의 생존 부분의 영역마스크로 가중치를 계산한 색상 값을 더하고, RUF 버퍼를 참조하여 프레그먼트 F_a의 가려지는 부분이 색상 값에 기여했던 만큼을 제거한다.

$$C_{ff}^{new} = C_{ff} + C_{if} \times \frac{N(M_{sur})}{T_m} - C_{RUF} \times \frac{N(M_{sur} \cap M_{RUF})}{T_m} \quad (2)$$

식에서, 함수 N( )은 영역마스크에서 비트가 1인 개수를 리턴한다. 또한, T_m은 영역마스크의 비트 수이다. 즉, 픽셀 당 샘플링 포인트 수이다.

하지만, 그림 4 (d)의 경우, 프레그먼트 F_c에 의해 가려지는 부분 중 일부, 예를 들어F_{ca}, 는 RUF 버퍼를 참조하여 정확하게 계산할 수 없다. 왜냐하면, 현재 RUF 버퍼는 F_b의 정보만을 가지고 있기 때문이다. 이 경우, 이 부분에 대해 식(3)의 네 번째 항을 이용하여 약간의 에러를 허용하며 보정한다. 즉, F_{ca} 부분에 대해 현재의 색상 버퍼에서 그것이 차지하는 만큼의 영역 가중치를 준 색상 값을 제거하여 보정한다.

$$C_{ff}^{new} = C_{ff} + C_{if} \times \frac{N(M_{sur})}{T_m} - C_{RUF} \times \frac{N(M_{sur} \cap M_{RUF})}{T_m} - C_{ff} \times \frac{N(M_{sur} \cap M_{ff})}{T_m} \quad (3)$$

예를 들어, 그림 5의 경우를 살펴보자. 프레그먼트 A, B, C는 모두 서로 다른 물체의 일부분이며, 렌더링 파이프라인에 도착순서는 프레그먼트 A→B→C 이며, 일부분이 서로 중첩되거나 교차한다. 먼저, 프레그먼트 A와 B가 도착했을 때 색상 값, C_{AB}는 다음 식(4)와 같이 계산되어진다. 식 (4)에서 $C_\varnothing$ 는 초기상태를 의미한다.

$$C_A = C_\varnothing + \frac{2}{8} \times C_A ; C_{AB} = C_A + C_B \times \frac{4}{8} - C_A \times \frac{1}{8} \quad (4)$$

이 과정을 수행하고 나서, RUF 버퍼에는 프레그먼트 B의 영역마스크, 색상 값, 물체 태그가 저장된다. 프레그먼트 C가 도착하는 경우, 그림 3 (a)와 (b)의 두 가지 경

우를 생각할 수 있다. 그림 3 (a)의 경우, RUF 버퍼에 저장된 프레그먼트 B의 정보를 이용하여 식 (2)를 사용하여 다음과 같이 색상 값을 구할 수 있다.

$$C_{ABC} = C_{AB} + C_C \times \frac{4}{8} - C_B \times \frac{1}{8} \quad (5)$$

하지만, 그림 3 (b)와 같은 경우, 프레그먼트 A에 대한 정보를 알 수 없으므로 식(3)을 이용하여 식(5)와 같이 색상 값이 계산된다. 이 경우, $\frac{1}{8}\times(C_{AB}-C_A)$의 색상 에러가 발생한다.

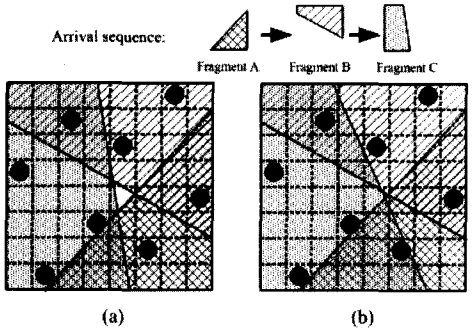


그림 5 프레그먼트의 도착순서와 상호관계에 따른 색상 값 결정 예제

RUF 버퍼 기록 과정: 일반적으로, 한 물체를 이루는 폴리곤은 같은 평면 공간(coplanar space)에 존재하며, 서로 이웃하는 폴리곤은 경계선 부근의 픽셀을 서로 공유하는 프레그먼트들을 생성한다[3,4]. 또한 이 프레그먼트들은 서로 배타적(exclusive)이므로, 병합하여 하나의 프레그먼트로 취급할 수 있다. 앞의 식(2)와 (3)에서 살펴보듯이, RUF 버퍼에 저장된 프레그먼트 부분이 픽셀에서 넓은 영역을 차지할수록 가려지는 부분에 대한 색상 값 정보를 제공할 수 있는 부분이 넓어지며, 이것은 색상 값 결정 과정에서 에러를 최소화할 수 있는 기회를 제공한다. 따라서, 다음 식(6)과 (7)을 사용하여 RUF 버퍼에서 두 프레그먼트를 병합한다.

$$M_{RUF}^{new} = M_{RUF} \cup M_{sur} \quad (6)$$

$$C_{RUF}^{new} = C_{RUF} \times \frac{N(M_{RUF})}{N(M_{RUF}^{new})} + C_{sur} \times \frac{N(M_{sur})}{N(M_{RUF}^{new})} \quad (7)$$

먼저, 생존 프레그먼트와 RUF 버퍼에 저장되어 있는 프레그먼트를 병합하기 위해서, 두 프레그먼트의 물체 태그를 비교한다. 그 값이 서로 같으면, 같은 물체의 폴리곤에서 생성된 프레그먼트이므로 식(6)과 같이 두 프레그먼트의 영역마스크(M)을 서로 OR 비트연산을 수행하고, 식(7)에서와 같이 각각이 차지하는 영역을 가중치 값을 사용하여 색상 값을 계산한 후 저장한다. 물체 태그가 다르면, RUF 버퍼를 생존 부분이 가지는 영역마스크, 물체 태그, 그리고 색상 값으로 RUF 버퍼를 갱신한다.

3. 실험 환경 및 결과

표 1은 본 논문의 실험에 사용한 3차원 모델에 대한 정점, 삼각형, 프레그먼트 그리고 물체의 수를 나타낸다. 실험에 사용한 3차원 데이타 셋은 퍼블릭 도메인에 공개되어 있는 일반적인 모델이며, OpenGL의 공개 소스 버전인 Mesa 그래픽 라이브러리에 포함되어 있다[5].

표 1 3차원 모델 데이타

모델명	버텍스 수	삼각형 수	프레그먼트 수	물체 수
AI	3618	7124	11975	35
Castle	6620	13114	17444	16
Dolphins	885	1692	4570	3
Pig	3522	7040	7499	3
Rose+vase	4028	3360	5425	5
Teapot	3644	6320	6807	1
Venus	711	1418	5464	1

각 3차원 모델 데이타에 대해 Mesa 라이브러리를 이용하여 지오메트리 연산과 스캔컨버전을 수행하였다. 스캔컨버전을 통해 얻은 부픽셀과 프레그먼트에 대해 제안된 방식과 다양한 샘플링 포인트를 가지는 슈퍼샘플링을 실험하였다. 실험에서는 1-포인트 샘플링(1p), 8 산개 슈퍼샘플링(8ss), 4×4 슈퍼샘플링(4×4s), 그리고 8×8 슈퍼샘플링(8×8s)을 적용하였다. 제안된 방식은 8 산개 슈퍼샘플링과 같은 샘플링 포인트를 가지도록 하였으며, 이하 '제안된 방식[8]' 이라 칭한다.

본 논문의 실험은 먼저 제안된 방식과 다양한 샘플링 포인트를 가지는 슈퍼샘플링을 통해 얻은 최종 영상의 화질 비교 실험을 수행 하였으며, 각 방식에 따른 요구되는 메모리 크기를 조사하였고, 마지막으로 요구되는 메모리 대역폭을 실험하였다.

3.1 평균 색상 오차

제안된 방식과 슈퍼샘플링을 통해 얻어진 최종 영상에 대해 논문 [6]에서와 같이 다음 식을 사용하여 픽셀의 오차 값을 구하였다.

$$\text{Color error} = \sum_{\forall i,j} \sum_{c=R,G,B} (r_{ijc} - t_{ijc})^2 \quad (8)$$

위 식에서 r_{ij}와t_{ij}는 각각 참조 이미지와 테스트 이미지의 같은 위치(i,j)를 나타내며, 그 위치의 픽셀 색상 값은 r_{ijc}, t_{ijc}로 표현된다. 식 (8)은 각 픽셀 별로 각 방식을 통해 얻은 테스트 이미지가 참조 이미지와 얼마나 색상 값의 차이가 나는지를 조사하며, 그 효과를 더욱 가시적으로 나타내기 위해 차이 값을 제곱하여 합산하였다.

그림 6은 8×8 슈퍼샘플링을 참조 이미지로 하여 각 모델의 색상 차를 구하여 평균한 결과이다. 실험 결과에 나타나듯이, 샘플링 포인트가 많은 경우 보다 더 좋은 화질을 나타내며, 같은 샘플링 포인트 수를 가지는 제안된 방식이 8 산개 슈퍼샘플링은 거의 비슷한 수준의 영상을 나타냄을 실험 결과 알 수 있다. 또한, 4×4 슈퍼샘플링의 샘플링 포인트 수가 16개로 8산개 슈퍼샘플링보다 2배 많은 샘플링 포인트 수를 가지지만, 산개 슈퍼샘플링 방식의 효율성[4] 때문에 색상 차이는 거의 비슷한 수준을 보인다.

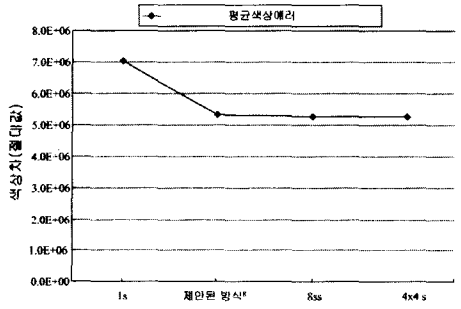


그림 6 각 샘플링 방식에 따른 평균 색상 오차

표 2는 8개의 산개 샘플링 포인트를 가지는 제안된 방식과 슈퍼샘플링의 픽셀 당 필요한 메모리 크기와 평균 색상 에러를 나타낸다.

표 2 8개의 샘플링 포인트를 가지는 경우 픽셀 당 요구되는 메모리 크기와 발생하는 평균 색상 오차

	8ss	제안된 방식	증감비율
평균색상 에러	1338250	1336154	-1.3%
픽셀 당 요구되는 메모리 크기	64Bytes	44Bytes	+31.3%

(색상 오차를 구하기 위한 참조이미지로는 8×8슈퍼샘플링으로 생성된 이미지를 사용하였다.)

결과에서 알 수 있듯이, 제안된 방식의 에러 증가율은 1.3%인데 반해 메모리 크기는 약 31% 감소하였다. 이것은 제안된 방식이 같은 샘플링 포인트를 가지는 슈퍼샘플링에 비해 색상 에러를 최소화하며, 요구되는 메모리 크기를 줄일 수 있는 좋은 방법임을 나타낸다.

3.2 픽셀 당 요구되는 메모리 크기

샘플링 포인트의 증가에 따른 슈퍼샘플링과 제안된 방식의 픽셀 당 요구되는 메모리 크기를 비교하였다. 제안된 알고리즘에서 하나의 픽셀이 차지하는 메모리 크기는 $2\times(m + C) + m\times Z + objtag$ 비트이다. 반면에 슈퍼샘플링의 경우 $m \times(C + Z)$비트가 요구된다. 표 3은 위의 공식을 사용하여 샘플링 포인트 증가에 따른 픽셀 당 필요한 메모리 크기를 나타낸다. 결과에서 알 수 있듯이, 샘플링 포인트가 늘어나면 늘어날수록 제안된 방식의 메모리 절감 효과가 커짐을 알 수 있다.

표 3 샘플링 포인트가 증가함에 따른 슈퍼샘플링 방식과 제안된 방식에서 픽셀 당 필요한 메모리 크기

샘플링 포인트 수	슈퍼샘플링	제안된 방식	감소비율
	요구되는 메모리크기(Bytes)		
4	32	27	16%
8	64	44	31%
16	128	78	39%
64	512	282	45%

표 4 총 메모리 대역폭과 감소 비율

모델명	8ss	제안된 방식[8]	감소비율	프레그먼트 당 평균 부픽셀 수
	메모리 대역폭(바이트)			
Al	518512	456654	11.9%	2.9
Castle	614394	574144	6.5%	2.8
Dolphins	229144	194176	15.3%	3.2
Pig	301489	279204	7.4%	2.7
Rose+vase	173760	163487	5.9%	2.6
Teapot	281472	260347	7.5%	2.7
Venus	338332	268950	23.5%	3.9

3.3 메모리 대역폭

그림 7은 슈퍼샘플링과 제안된 방식의 알고리즘 상에서 필요한 메모리 액세스 패턴을 나타낸다. 메모리 대역폭을 측정하기 위해, 각 메모리 액세스 패턴과 해당 메모리 액세스 패턴의 빈도수가 그림에 나타나 있다. 요구되는 총 메모리 대역폭은 식(9)와 같이 계산되어지며, 식에

서 RMB_i는 메모리 억세스 패턴 i 가 요구하는 메모리 대역폭(Required Memory Bandwidth)이다.

$$\sum_{i \in \{\text{access pattern}\}} (RMB_i \times Occurence_i) \quad (9)$$

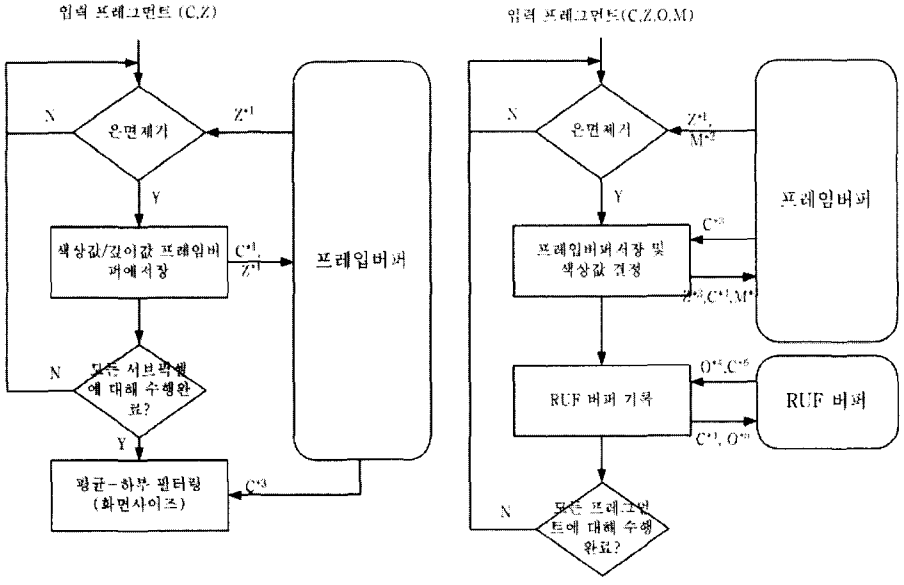
표4는 각 실험 데이타 셋에 대해 임의의 특정 영상을 생성하기 위한 식(9)를 이용하여 요구되는 메모리 대역폭을 계산한 결과이다. 결과적으로, 평균 11.0%의 메모리 대역폭이 감소되었다. 또한, 메모리 대역폭의 절약 효과는 표에서 알 수 있듯이 프레그먼트가 가지는 평균 부픽셀의 수와 관련이 있다. 즉, 프레그먼트가 가지는 부픽셀의 수가 많으면 많을수록 메모리 대역폭 절약 효과는 커진다. 그 이유는 슈퍼샘플링과 달리, 제안된 알고리즘에서 프레그먼트를 표현하기 위해 추가된 물체 태그, 영역마스크 등에 대한 요구되는 메모리 크기에 대한 부담은 프레그먼트가 가지는 부픽셀의 수가 증가할수록 줄어들기 때문이다.

그림 7에 나타난 기호와 위첨자는 다음과 같은 의미를 가진다.

기호	의미	RMB	기호	의미	RMB
C	색상 값	4 Bytes	O	물체 태그	2 Bytes
Z	깊이 값	4 Bytes	M	영역 마스크	|m| bits
위첨자(빈도수)					
*1	총 부픽셀 수		*4	총 생존 프레그먼트 수	
*2	총 프레그먼트 수		*5	RUF버퍼기록 단계에서 물체 태그가 서로 다른 경우의 총 빈도수	
*3	총 생존 부픽셀 수		*6	RUF버퍼기록 단계에서 물체 태그가 서로 같은 경우의 총 빈도수	

4. 결론

제안된 방식은 슈퍼샘플링과 비교하여 거의 비슷한 수준의 안티알리아스된 고화질 영상을 제공하면서, 메모리 크기와 메모리 대역폭을 효과적으로 절약할 수 있다. 더욱이, 샘플링 포인트가 늘어날수록 상대적으로 그 효과는 더욱 늘어난다. 제안된 렌더링 구조는 비교적 간단한 하드웨어의 추가로 구현될 수 있으며, 또한 기존의 렌더링 시스템에서 사용하는 대표적인 은면제거 기법인 Z buffer 알고리즘을 기본으로 사용하기 때문에, 기존의 시스템에 쉽게 융합할 수 있다. 결론적으로, 본 논문에서 제안한 방식은 경제적인 하드웨어 비용으로 고화질 영상을 생성할 수 있는 방법이다.



(a) 슈퍼샘플링 (b) 제안된방식

그림 7 슈퍼샘플링과 제안된 방식에서 메모리 액세스 종류와 빈도수

참 고 문 헌

[1] Alan Watt. '3D Computer Graphics', Third Edition, Addison-Wesley, 2000.

[2] Paul E. Haeberli and Kurt Akeley. "The Accumulation Buffer. Hardware Support for High-Quality Rendering", In Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 90), volume 24, pages 309-318, August 1990.

[3] Loren Carpenter, "The A-buffer, an Antialiased Hidden Surface Method", In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 84)*, volume 18, pages 103-108, July 1984.

[4] Jin-Aeon Lee and Lee-Sup Kim, "Single-Pass Full-Screen Hardware Accelerated Antialiasing", *Proceedings 2000 SIGGRAPH/EUROGRPHICS Workshop on Graphics Hardware*, pages 67-75, August 2000

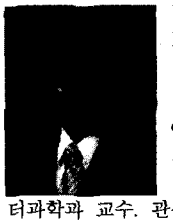
[5] The Mesa 3D Graphics Library,

[6] Norman P. Jouppi and Chung-Fa Chang, "Z3: an economical hardware technique for high-quality antialiasing and transparency", *Proceedings 1999 SIGGRAPH/EUROGRPHICS worksh op on Graphics hardware*, pages 85-93, 1998

김 병 욱
1996년 연세대학교 공과대학 컴퓨터과학과 (학사). 1998년 연세대학교 대학원 컴퓨터과학과 (공학석사). 1998년 ~ 현재 연세대학교 대학원 컴퓨터과학과 박사과정
관심분야 : 3차원 그래픽 알고리즘, 3차원 그래픽 가속기, Computational Geometry

박 우 찬
1993년 연세대학교 이과대학 전산과학과 (학사). 1995년 연세대학교 대학원 전산과학과 (이학석사). 2000년 연세대학교 공과대학 컴퓨터과학과 (공학박사). 2001년 - 현재 연세대학교 연구교수
관심분야 : 3차원 그래픽 가속기, ASIC 설계, 병렬 렌더링, 고성능 컴퓨터 구조, Computer arithmetic

양 성 봉
1984년 University of Oklahoma 컴퓨터과학(학부과정). 1986년 University of Oklahoma 컴퓨터과학(석사). 1992년 University of Oklahoma 컴퓨터과학(박사). 1992년~1992년 University of Oklahoma, Adjunct Assistant Professor. 1993년~1994년 전주대학교 전자계산학과 전임강사. 1994년~현재 연세대학교 공과대학 컴퓨터과학과 부교수. 관심분야는 병렬처리, Computational geometry, Spatial data processing, 3D graphics algorithm. GIS



한 탁 돈
1978년 연세대학교 공과대학 전자공학과 (학사). 1983년 Wayne State University 컴퓨터공학(공학석사). 1987년 University of Massachusetts 컴퓨터공학(공학박사). 1987년-1989년 Cleveland 주립대학 조교수 1989년-현재 연세대학교 공과대학 컴퓨터과학과 교수. 관심분야는 Wearable computer, 3차원 그래픽 가속기, HCI, ASIC 설계, 고성능 컴퓨터구조