論文 91-28A-5-9

가려진면 제거와 색도 계산을 위한 그래픽스 가속기

(A Graphics Accelerator for Hidden Surface Removal and Color Shading)

方 卿 鎰* 裵 成 玉** 慶 宗 旻*

(Kyung Il Bang, Seong Ok Bae, and Chong Min Kyung)

要 約

본 논문에서 제안된 그래픽스 가속기는 선형 보간기, 곱셈기와 Edge painting tree로 구성된다. 결합된 이진 트리 구조를 갖고 있는 선형 보간기는 두 점의 함수값을 선형 보간한다. 두개의 곱셈기는 선형 보간기의 입력값들을 동시에 계산하며, 보간영역 바깥의 데이타를 없애기 위한 매스크 패턴은 edge painting tree에서 생성된다. 본 논문에서 제안된 하드웨어 구조는 64개의 화소들을 담당하며, 10×10의 크기를 갖는 다각형을 1초에 최대 5,900여개 처리할 수 있다.

Abstract

This paper presents a graphics accelerator for fast image generation. The accelerator has three major functional blocks: linear interpolator, multipliers and Edge Painting Tree. Linear interpolator with coupled binary tree structure interpolates functional values of two end points. Two multipliers compute input values of interpolator in parallel. Mask pattern which removes out invalid data is generated by Edge Painting Tree. The proposed architecture in this paper is responsible for 64 pixels and can process about 5,900 10×10 polygons per second.

I. 서 론

컴퓨터 그래픽스에서는 데이타 베이스에 있는 물체들의 절대 좌표계에서의 좌표값으로부터 나타낼 물체들의 화면상의 좌표값을 계산한 후, 이 좌표값들을 가지고 화면상에 물체들을 그려주는데, 물체를 그리는 방법으로는 크게 물체의 윤곽선만을 그리는 외형선 표현(wireframe drawing)방법과 물체표면의 색깔값까지 계산하여 주는 방법의 두 가지가 있다. 이때, 실제적인 영상을 얻기 위하여 물체표면의 색깔값을 결정하는 과정을 컴퓨터 그래픽스에 있어서 shading이라 하며, 광선의 물리적 성질과 물체표면의 성질에 따라 색깔값이 달라지게 된다.

여러 shading 알고리듬들 중에서 Gouraud shading[2][3]은 intensity값의 선형보간(linear interpolation)에 의해 간단히 이루어지므로 여러 하드웨어적인 접근법에서 널리 사용된다. 그리고 관측자로부터 물체까지의 거리인 z값(viewdepth)역시 선형 보간에 의해 구할 수 있으므로 선형 보간기를 가지면 z값 계산과 Gouraud shading을 할 수 있다. 선형보간에 의해 z값을 계산하고 Gouraud shading을 하는 아키텍쳐에는 Super Buffer,[4] PIXEL PLANES[5] 등이 대표적이다.

*正會員, **準會員, 韓國科學技術院
(Dept. of Electrical Eng., KAIST)

接受日字 : 1990年 6月 23日

선형 보간을 위한 또 하나의 하드웨어 아키텍쳐로서 임의의 면을 처리할 수 있는 PIPE[6]라는 아키텍쳐가 제안된 바 있는데, PIPE에서는 중간점 나눔(midpoint-subdivision)방법을 사용하여 보간을 행한다. 본 논문에서는 중간점 나눔 방법에 근거한 bit-serial 보간기와 더불어 관측자로부터 물체까지의 거리인 z값을 비교할 수 있는 z값-비교기, 출력 데이타의 매스킹을 위한 EPT(edge painting tree)[1]및 입출력 버퍼들을 가지고 가려진면 제거(hidden surface removal)와 Gouraud shading을 할 수 있는 그래픽스를 위한 하드웨어 가속기(이하 shading 엔진)를 설계하고 Daisy 워크스테이션을 이용하여 TTL 레벨로 시뮬레이션하였다.

설계된 shading 엔진은 64개의 화소들을 담당하며, bit-serial로 데이타를 처리한다. 모든 연산소자는 1-bit 전가산기와 D flip-flop 등의 간단한 소자들로 구성되므로 적은 하드웨어 비용을 갖는다. 또한 설계된 shading 엔진은 모듈화 되어 있으므로 사용자의 형편에 따라 여러가지 형태의 그래픽스 시스템을 구현할 수 있다.

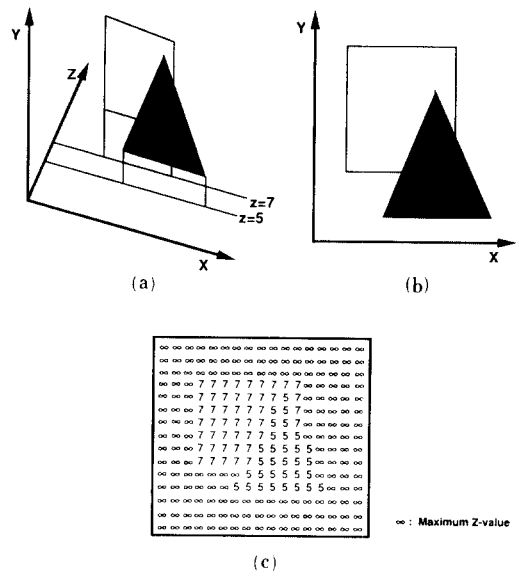


그림 1. Z-버퍼 알고리듬의 구현예
Fig. 1. An example of the z-buffer algorithm.

Ⅱ. 가려진면 제거 알고리듬과 Shading알고리듬

1. 가려진면 제거 알고리듬

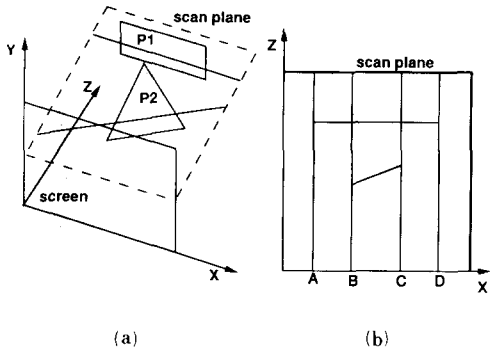
3차원 물체를 화면상에 표현할 때 다른 면에 의해 가려지는 면을 제거하기 위한 대표적인 알고리듬으로 z-버퍼 알고리듬,[3] 스캔라인 알고리듬[3] 등이 있다.

본 논문에서 사용한 가려진면 제거 알고리듬은 z-버퍼 알고리듬으로서 이 알고리듬에서는 화면상의 각 화소마다 관측자로부터의 거리인 z값(view-depth)를 저장하는 z-버퍼(메모리)를 두고, 새로운 다각형을 처리할 때마다 그 새로운 다각형을 이루는 화소들의 z값을 계산하여 z-버퍼에 있는 기존의 z값들과 각각 비교한다. 새 다각형에 속한 화소의 z값이 기존의 z값보다 크면 그 화소는 보이지 않는 화소가 되며, 작은 경우에는 보이는 화소이므로 새 다각형의 해당 화소를 화면에 그려주고, z-버퍼를 새로운 z값으로 대체한다. 다음의 그림 1은 z-버퍼 알고리듬의 보기로서 z=5인 면에 있는 삼각형과 z=7인면에 있는 사각형에 대한 z-버퍼 알고리듬의 적용결과를(c)에 보이고 있다. Z-버퍼 알고리듬은 다각형들을 임의의 순서대로 처리할 수 있고 수행이 간단하다는 이점이 있지만 메모리 비용이 많이 드는 단점이 있다.

또하나의 가려진면 제거 알고리듬인 스캔라인 알고리듬에서는 다각형들을 스캔라인 차례로 그려주게 되는데 z-버퍼 알고리듬이 다각형 단위로 수행되는데 반해 한 스캔라인 단위로 수행되는 점이 z-버퍼 알고리듬과 다른 점이다. 스캔라인 알고리듬은 그림 2(a)에 보인 것과 같이 한 스캔라인을 z-축을 따라 이동시켜 scanplane을 정의한 후, 이 scanplane과 다각형과의 교차로 인해 생성되는 선분(이것을 spen이라고 한다)을 구한다음 이 선분들을 가지고 화면상에 그림을 그리게 된다. 이 과정으로 인해 가려진면 제거 알고리듬은 가려진선 제거(hidden line removal 알고리듬으로 바뀌며, scanplane상에 있는 선분들의 양 끝점의 z-값을 선분들 상호간에 비교함으로써 보이는 부분을 결정한 후 화면상에 그려주게 된다. 그림 2(b)에서 AB, CD부분은 다각형 P1의 색으로, BC 부분은 삼각형 P2의 색으로 그려준다. 스캔라인 알고리듬은 메모리 오버헤드(overhead)는 없지만, 다각형들을 스캔라인 순서로 분류해야하는 단점이 있다.

2. Shading 알고리듬

가려진면 제거 알고리듬에 의해 보이는 면들을 결정한 후에는 이 보이는 면들을 shading하여야 한다. 이 shading을 위한 알고리듬에는 Gouraud shading,[2] Phong shading,[7] 광선 추적 알고리듬,[8] 등이 대표적이다. 이 중 Gouraud shading은 Phong shading 이나 광선 추적 알고리듬보다 실감이 떨어지지만 그 알고



(a) (b)

그림 2. 스캔라인 알고리듬의 예
(a) 스캔라인과 다각형과의 교차로 인한 span의 생성
(b) 생성된 span에 대한 가려진선 제거

Fig 2. An example of the scanline algorithm.
(a) generation of a span,
(b) hidden line removal for the span.

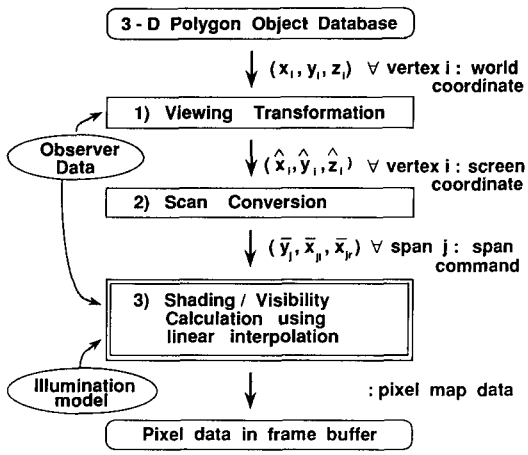


그림 3. 3차원 그래픽스에 있어서의 이미지 생성과정

Fig. 3. Image generation procedure in 3-D computer graphics.

리듬이 간단하므로 컴퓨터 그래픽스를 위한 병렬처리 방법에서 널리 사용되고있다.

그래픽스에서 곡면은 작은 다각형들로 근사화하거나 직접 곡면의 식을 이용하여 나타내게 되는데 직접 곡면의 식으로 나타내는 것은 그리기 위한 계산시간이 많이 소요되고 복잡하므로 보통은 작은 다각형들로 그 곡면을 근사화하는 방법을 사용하게 된다. 이 때 작은 다각형들을 일정한 색도값으로 shading을 하게 되면 색도의 불연속에 의해 원래의 곡면의 감을 얻기가 힘들다. Gouraud shading은 이러한 경우에 있어 색도의 불연속을 줄이기 위한 알고리듬으로서 다각형의 각 꼭지점에서 색도를 구한 후 선형보간을 이용하여 각 화소의 색도를 계산함으로써 급작스러운 색도의 변화를 줄인 것이다. Gouraud shading에 대해서는 Ⅲ장의 개요에서 좀 더 자세히 언급될 것이다.

Ⅲ. 가려진면 제거와 Shading을 위한 그래픽스 가속기의 구조

1. 개요

3차원 그래픽스에 있어서의 이미지 생성과정은 그림 3과 같이 요약될 수 있다.

3차원 데이타 베이스에 있는 다각형으로 이루어진 물체(object)에 대해 절대 좌표계에서의 물체의 좌표를 관찰자가 기준이 되는 화면 좌표계에 대한 좌표로 바꾸어 주는 viewing transformation을 하게 되면 절대 좌표계에서의 다각형의 꼭지점들은 화면상의 점들로 맵핑되며, 화면상에 다각형을 생성하게 된다. 그림 4에는 화면좌표상에 삼각형이 생성된 경우를 예로 들었다. 그림 4의 경우와 같이 삼각형의 각 꼭지점, A, B, C에 대한 화면좌표(X_s, Y_s, Z_s)가 구해지면 각 꼭지점에 대하여 주위면들, 즉 3차원 물체를 이루는 면들 중에서 한 꼭지점을 공통으로 포함하는 면들(화면상에는 보이지 않을수도 있음)의 normal vector의 평균을 취하는 방법으로 각 꼭지점의 normal vector을 구한다. 다음, 꼭지점의 normal vector와 광원과의 관계를 이용하여 각 꼭지점에서의 색도를 구한다. 각 꼭지점의 z값(Zs)과 색도가 결정되면 각 스캔라인에 대한 span 명령어(다각형과 현재의 스캔라인 Yc가 만나는 부분에 대한 정보)[Xl, Xr, f(Xl), f(Xr)]를 얻게 된다.

이때, f(x)는 z값 또는 색도값을 나타내며 두 꼭지점의 f(x)값을 선형보간하여 구한다. 예를 들면, 그림 4에서 점 P_1의 z값인 z(Xl)은 꼭지점 A의 z값과 꼭지점 B의 z값을 선형보간하여 구하게 된다.

이 span 명령어로부터 span P_1P_2에 대한 직선의 식,

$$f(X) = A \cdot X + B \quad (A: \text{기울기}, \ B: \text{절편}(= f(0)))$$

을 얻게되며 이 직선의 식이 구해지면 Xl부터 Xr까지의 화소들에 대한 f(x), (z값 또는 색도)를 선형보간에 의해 구하게 된다. 선형보간을 하는데 있어서는 z값을 먼저 계산하여 각 화소의 가시성 여부를 먼저 판별한 후에 각 화소의 색도계산을 하고 관찰자에게 보이게 되는 화소에 대한 색도값을 프레임

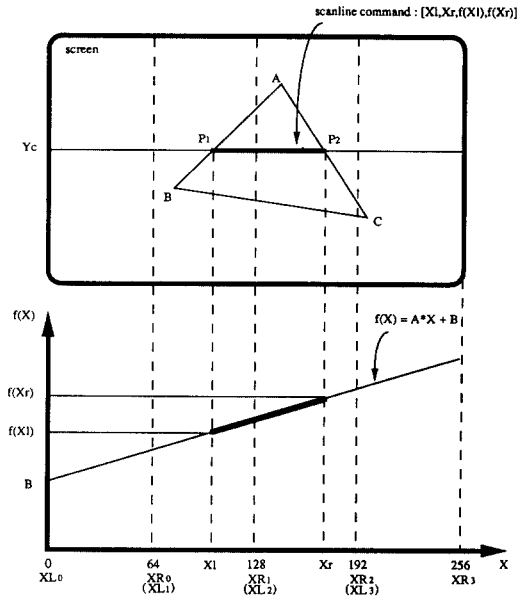


그림 4. Span 명령어
Fig. 4. Span command.

버퍼에 써주게 된다. 다음 비데오 콘트롤 시스템에서는 프레임 버퍼에 있는 값을 화면에 뿌려줌으로써 우리가 원하는 화면을 얻게 된다.

본 논문에서 제안된 shading 엔진이 그림3의 이미지 생성 단계중 담당하는 부분은 단계 3 으로서, 일반적으로 소프트웨어로 수행할 경우에는 루프를 돌면서 덧셈을 하여 각 화소의 함수값을 계산하던 것을, (이 경우 span의 길이만큼 루프를 돌게 된다.), 하드웨어적인 접근법을 사용하여 한 span을 이루는 여러개의 화소에 대한 선형보간을 동시에 함으로써 계산시간을 줄이도록 한 것이다. 이러한 병렬처리 하드웨어를 설계하는데 있어, 수백~수천개의 화소들에 대한 병렬처리 엔진을 만드는 것은 일이 크므로 화면을 분할하여 한개의 shading 엔진이 한 스캔라인상의 64개의 화소들을 담당하도록 하였고 그래픽스 시스템은 똑같은 구조를 갖는 여러개의 shading 엔진을 사용하여 구성될 수 있도록 하였다.

화면상에서 shading 엔진이 담당하는 영역은 XL과 XR값으로 결정되는데 i번째 shading 엔진이 담당하는 영역의 XL_i과 XR_i은,

$$XL_i = 64 \times i,\ i:0, 1, 2, \cdots, \#\ \text{of SEs} - 1$$
$$XR_i = XL_i + 64$$

의 식으로 나타내어 진다. 다음의 그림5는 여러개의 shading 엔진(SE)을 사용한 그래픽스 시스템의 구성예이다.

본 논문에서 설계된 한개의 shading 엔진에 대한 블럭 다이아그램을 그림 6에 보였다(굵은 네모 안 부분). 각 shading 엔진에는 호스트로부터의 span 데이타를 저장하고 내부자원의 콘트롤을 위한 콘트롤 블럭, 보간기의 입력을 만들어 주는 두개의 serial곱셈기(L-multiplier와 R-multiplier)보간기, 출력 데이타의 매스킹을 위한 EPT(edge painting tree), z값 비교와 그 결과의 저장을 위한 z값 비교 블럭과 WE signal storage, z값 비교를 위해 기존의 z값을 shading 엔진 외부의 z-버퍼로부터 읽어 shading엔진 내부에 저장하기 위해 사용되는 old- z-block, 보간결과를 저장하고 메모리와의 데이타 교환을 위한 입출력 버퍼(new z-block, RGB-block)들로 이루어진다. 보관결과는 보간기로부터 bit-serial로 생성되므로 입출력 버퍼로서 shift registor를 사용하였고, 메모리 참조시간을 줄이기 위하여 4-way 인터리빙(interleaving)방법을 사용하므로 4 개의 입출력 포트를 갖는다.

Shading 엔진의 동작은 다음과 같다. 64화소 단위로 나누어진 화면의 i번째 부화면(sub screen)을 담당하는 i번째 shading 엔진은 시스템 초기화시에 자신이 담당하는 영역에 대한 데이타 XL_i과 XR_i를 받는다. 다음 정상동작시에는 span 명령어를 만들어 주는 스캔라인 프로세서 (별도의 하드웨어 혹은 호스트)로부터 span에 대한 데이타(Xl, Xr), z값과 색도

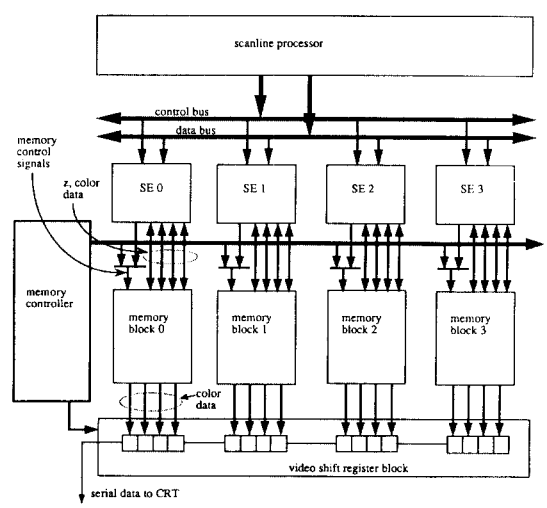


그림 5. 그래픽스 시스템의 구성예
Fig. 5. An example of graphics system construction.

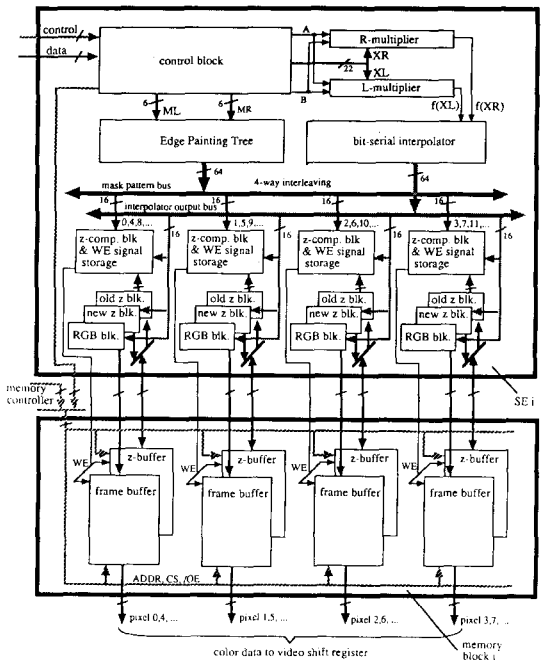


그림 6. Shading 엔진의 블럭도
Fig. 6. Block diagram of shading engine.

에 대한 직선함수의 기울기 A와 절편 B값을 받아 콘트롤 내부내에 저장한다. 다음 A와 B값을 두개의 곱셈기에 serial하게 보내주면 두 곱셈기는 shading 엔진이 담당하는 영역의 양 끝점에서의 함수값 $f(XL_i)$과 $f(XR_i)$을 각각 계산한다. 다음, 이 두값은 보간기에 전해지고 보간기는 XL_i부터 XR_i까지의 64개의 화소들에 대한 함수값을 parallel하게 보간한다.

함수값의 보간은 z값을 먼저하고 색도를 나중에 하며, 보간기의 결과는 각 함수값을 위한 내부 입출력 버퍼(그림 6의 'new z blk'와 'RGB blk')에 저장된다. EPT에 의한 매스킹은 z값이 보간될 때 동시에 행해지며 span에 대한 매스킹 값, ML_i과 MR_i를 콘트롤 블럭으로부터 받아 행해진다. Z값 비교를 위해 기존의 z값을 z-버퍼로부터 shading 엔진내부의 버퍼 'old z blk'에 저장하는 동작도 z값의 보간과 동시에 이루어지며 보간기로부터 새로운 z값이 출력되면 'old z blk'내의 기존의 z값과 새로운 z값과의 비교가 이루어진다. EPT에 의한 매스킹 결과는 z값 비교에 의한 결과와 AND되어 그림 6의 'WE signal storage' 레지스터에 저장되며, z값에 이어 색도에 대한 보간이 끝나면 new z-block과 RGB-block에 저장되어 있는 현재의 span에 대한 각 화소들의 새로운 z값과 색도값이 'WE signal storage' 레지스터에 저장되어 있는 각 화소의 메모리 Write Enable 신호에 따라 z-버퍼와 프레임 버퍼에 쓰여지게 된다. Z-버퍼에 쓰여진 z값은 새로운 span에 대하여 z값을 비교할 때 쓰이게 되며, 프레임 버퍼에 쓰여진 색도값은 비데오 콘트롤 시스템에 의해 읽혀져서 화면상에 디스플레이 되게 된다.

2. L, R-곱셈기들과 보간기

보간기는 주어진 영역의 왼끝점에서의 함수값 f(XL)과 오른끝점에서의 함수값 f(XR)을 입력으로 하는데, 직선의 기울기와 교점으로부터 f(XL)과 f(XR)을 계산하기 위하여 직선의 식, $f(X)=A\cdot X+B$의 함수값을 파이프라인 방식으로 계산할 수 있는 serial 곱셈기를 사용하였다. 다음의 그림 7은 곱셈기의 기본 구조로서 그림 7에서 X의 데이타 크기는 m+1이다. Shading 엔진에서는 이 serial 곱셈기를 사용하여 보간기의 입력값을 계산하며, 화면의 크기가 1024×N인 경우까지 처리할 수 있도록 곱셈기의 stage수는 11로 하였다. 따라서 곱셈결과는 (11+n)Tc, (여기서 n은 f(X)의 데이타 크기, Tc는 클럭주기) 만에 얻을 수 있으며 z의 함수값이 24-bit, Red, Gree Blue의 함수값이 각각 16-bit의 데이타 크기를 가지는 경우에 10 MHz의 동작주파수에서 z값은 3.5μsec

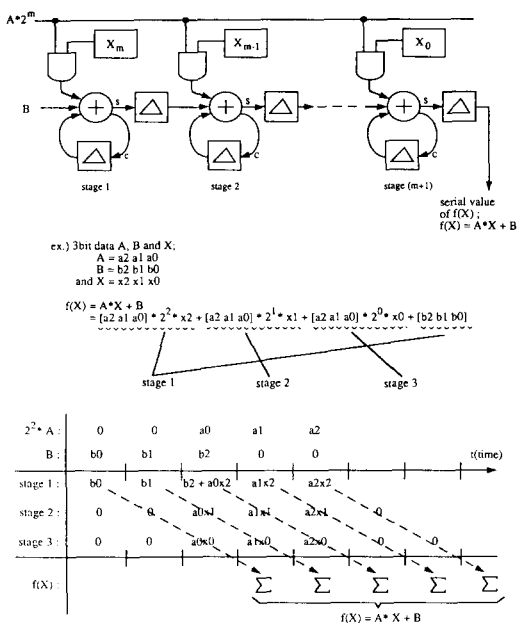


그림 7. Serial 곱셈기
Fig. 7. Serial multipulier.

만에, Red, Green, Blue값 각각은 2.7μsec만에 보간기의 입력을 계산할 수 있다.

두개의 함수값 f(XL)과 f(XR)로부터 XL부터 XR 사이의 화소들의 함수값을 선형 보간하기 위해서 PIPE에서 제안된 보간기의 기본 원리인 중간점 나눔방법을 이용하였으며 그 예는 다음과 같다. 예를 들어, 9개의 화소들, 화소 0부터 화소 8까지의 함수값들을 구하는 경우 트리의 레벨은 3레벨이 된다. 레벨 1에서는 f(4)의 값을 f(0)과 f(8)을 가지고 계산하며,

$$f(4)=(f(0)+f(8))/2$$

레벨 2에서는 f(0), f(4)과 f(8)을 가지고 아래의 식을 이용해

$$f(2)=(f(0)+f(4))/2$$
$$f(6)=(f(4)+f(8))/2$$

f(2)와 f(6)를 동시에 구한다. 다음, 레벨 3에서는 마찬가지 방법으로 f(1), f(3), f(5), f(7)을 구해 최종적으로 f(0)부터 f(8)을 얻을 수 있게 된다.

본 논문에서 제안된 shading 엔진에서는 위의 중간점 나눔방법을 가지고 serial 보간기를 설계하였으며, PIPE에서는 중간값을 add and shift right에 의해 구하고 있는데 비해 serial 보간기에 있어서는 각 데이타의 시간지연을 다르게 함으로써 구현하고 있다. 이 경우, add and shift right에 의해 보간을 행할 때 보간기의 각 레벨에 있는 연산소자에서 shift right를 할 때 발생하는 truncation error가 없게 되며, 데이타 크기(한 데이타 당 bit수)가 달라지는 경우에도 적절히 대처할 수 있다. 그림 8은 9개의 화소들을 위한 보간기의 블럭 다이아그램이다.

한 span에 대한 기울기와 교점으로부터 f(XL)과 f(XR)을 구해 보간을 하는 경우에 f(XL)과 f(XR) 값의 부호가 서로 다른 경우가 생긴다. 이 경우에도 올바른 보간 결과를 얻기 위해서는 보간기의 각 연산소자에서 sign bit를 고쳐주어야 하는데 이것은 부호 확장(sign extension)에 의해 해결할 수 있다. 그림 9는 올바른 연산을 위한 부호 확장 회로이다.

3. Edge Painting Tree와 z값의 비교

보간기에서는 shading 엔진이 담당한 영역내의 모든 화소들의 함수값을 계산해 내지만 실제적으로 우리가 필요로 하는 것은 주어진 다각형과 스캔라인이 겹치는 부분인 span에 속하는 화소에서의 함수값으로 이외의 함수값은 제거되어야 한다. 이를 위하여 EPT를 사용하였는데 EPT는 span에 속해있는 화소들에 대해서만 플래그 값을 1로 세트 해줌으로써 매

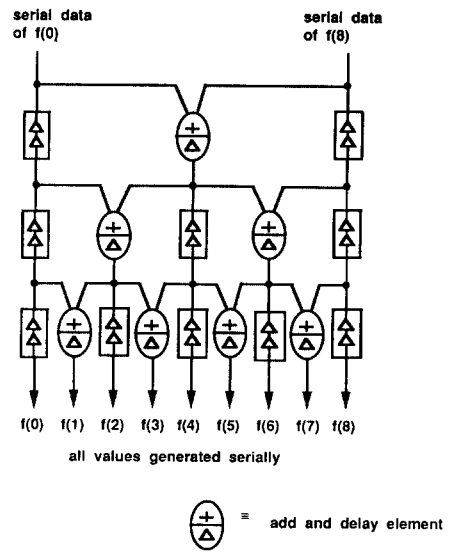


그림 8. 9개의 화소들을 위한 bit-serial 선형 보간기
Fig. 8. Bit-serial linear interpolator for 9 pixels.

스킹을 가능케 한다.

Span을 이루는 화소들에 대한 가려진면 제거는 z값의 비교를 위한 1-bit 비교기와 shift register를 가지고 행하며 다음의 그림10은 1개의 화소에 대한 z값 비교회로이다. 비교를 행하기 전에 미리 기존의

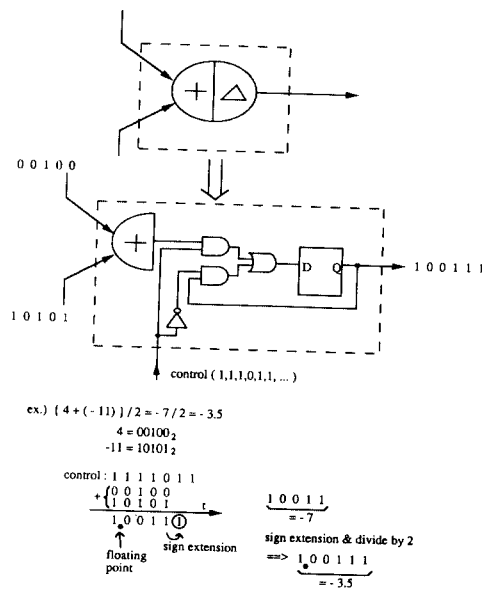


그림 9. 부호 확장 회로
Fig. 9. Sign extension logic.

z값을 메모리로부터 읽어 old z-block에 저장하며, z값의 비교는 새로운 z값의 LSB가 보간기로부터 출력될 때 시작되어 MSB가 출력되면 끝나게 된다. 두 값의 비교에 있어서 두 값이 동시에 준비되는 경우에는 MSB부터 비교를 하는 것이 빠르나 앞의 경우와 같이 serial로 데이타가 발생될 때에는 MSB부터 비교를 하기 위해서는 n-bit 데이타에 대해 n clock-cycle의 준비시간이 필요하고 총 비교시간은 (n+k, k≤n) clock cycle이 된다. 따라서 본 논문에서는 LSB부터 비교를 하여 n clock cycle만이 비교가 끝나도록 하였다.

그림10의 flag 1은 NZ_i (새로운 z값의 i번째 bit)가 OZ_i (기존 z값의 i번째 bit)보다 작을 때 1이 된다. NZ_i가 OZ_i와 같거나 클때에는 이전의 값이 유지된다. flag 1의 초기값은 0이다. Z값의 MSB 비교가 끝난 후에 flag 1의 값은 현재 처리되고 있는 다각형의 내부 화소에서의 가시성을 나타낸다. 이 z값 비교는 새로운 z값을 출력버퍼에 저장하는 것과 동시에 이루어지며, flag 1의 값은 EPT로부터의 매스크 bit와 논리적 AND 되어 각 화소의 함수값을 메모리에 쓸 때 write enable 신호로서 사용된다.

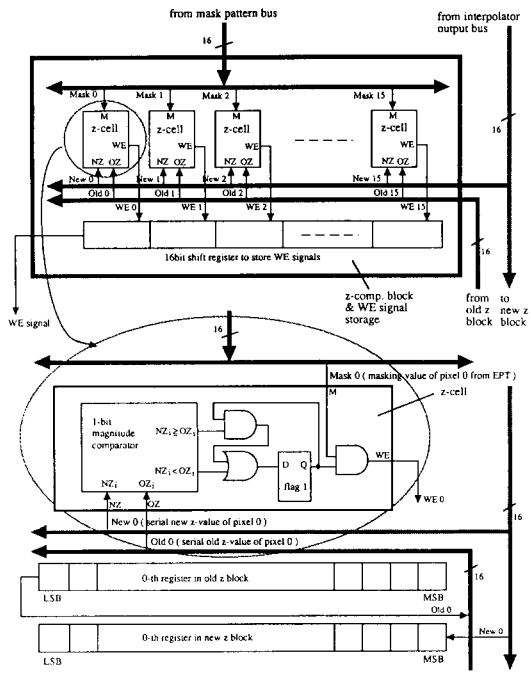


그림10. 1개의 화소에 대한 z값 비교 회로
Fig. 10. Z-depth comparison logic for one pixel.

Ⅳ. 결과 및 고찰

본 논문에서 제안된 가려진 면 제거와 색도 계산을 위한 그래픽스 가속기인 shading 엔진을 daisy 워크스테이션을 이용하여 TTL 레벨로 시뮬레이션하였다. 위의 shading 엔진은 간단한 연산소자들을 수천개씩 사용하여 구현되므로 비록 TTL 레벨로 로직 시뮬레이션을 했다고 하더라도 상용의 TTL 소자들로 구현하는 것은 매우 어려울 뿐만 아니라 구현 비용에 대한 가치를 찾기 어렵다. 따라서 복잡도와 실현성에 비추어 칩으로 만드는 것이 절대적으로 필요하며 제안된 shading 엔진은 단순한 로직소자들만으로 구성이 가능하므로 현재의 VLSI기술로 칩으로 제조하는 것은 문제가 없을 것으로 생각된다.

시뮬레이션에 있어서, Daisy 시뮬레이션 tool의 제한(시뮬레이션이 가능한 최대 gate수 등)때문에 여러개의 shading 엔진들을 사용하는 시스템에 대한 시뮬레이션은 행하지 못하였고 앞의 그림6에 해당하는 한개의 shading 엔진에 대한 시뮬레이션만 행하였다. 시뮬레이션을 위한 입력으로는 10MHz의 동작주파수, shading 엔진이 담당하는 영역, 그리고 2개이상의 연속되는 span들에 대한 각 span의 양끝점의 x좌표값들과 z값, Red값, Green값과 Blue값들에 대한 기울기와 절편값을 연속적으로 주었으며, 이 때, span의 양끝점의 x좌표를 변화시키거나 양끝점들의 함수값(z, Red, Green, Blue)을 변화시키면서 여러경우에 대하여 결과가 올바로 나오는지를 확인하였다. 시뮬레이션 결과 64개의 화소들을 위한 한개의 shading 엔진의 gate수는 약 24,000여개였으며 shading 엔진의 출력단에서 한 span에 대한 가시성 계산과 색깔값의 결과는 17 μsec마다 출력되었다. 이 17 μsec는 shading 엔진내에서 한 span에 대한 데이타를 latch하는 시간, 보간기의 입력을 만들어 주기 위해 serial 곱셈기에서 f(XL)과 f(XR)을 계산하는 시간, 보간기의 set-up time, z값에 대한 보간기의 보간시간, EPT에 의한 매스킹 시간, z값의 비교시간, 색도에 대한 보간시간 모두를 포함하는 시간으로 만약 10개의 span들로 이루어진 다각형의 데이타가 연속적으로 주어진다면 shading 엔진 자체만으로는 한 다각형을 0.17ms안에 처리할 수 있으며, 1초 동안에는 이러한 다각형을 최대 5,900여개까지 처리할 수 있을 것이다.

이러한 shading 엔진의 성능을 시스템이 구성되지 않은 상태에서 다른 시스템과 직접 비교하는 것은 무리가 있긴 하지만, Pixel-Planes 5[10]와 비교해 볼 수 있다. Pixel-Planes 5는 참고문헌[5]의 최근 버

젼으로서 Phong shading을 제공하며 1초에 약 100만개의 삼각형을 처리할 수 있는 고성능의 그래픽 시스템이다. 그러나 위와 같은 성능을 얻기위해서 Pixel-Planes 5는 128개의 QEE(quadratic expression evaluator)칩을 갖고 있는 Renderer 16개, 32개의 GP(geometry processor), 그리고 약 5Gbits/sec의 전송능력을 가진 Ring Network을 가지고 있다. 본 논문의 shading 엔진의 성능을 그대로 비교한다면 shading 엔진의 성능은 Pixel-Planes 5에 비해 크게 뒤지는 것이다. 그렇지만 본 논문에서 제안된 shadding 엔진이 구현 이전의 로직 검증단계이고 로직 간소화가 완전하게 이루어지지 않은 점, Pixel-Planes 5에서 shading을 위해 사용된 QEE칩에 비해 하드웨어 복잡도가 낮은 점, 앞으로 shading 엔진을 VLSI 칩으로 구현하여 동작주파수를 10MHz 이상에서 동작시키는 경우 등을 감안한다면 shading 엔진의 성능은 더욱 향상될 것으로 기대된다.

다음의 그림11은 시뮬레이션 결과중의 하나로서, shading 엔진이 담당하는 영역을 (XL, XR) = (0, 64)로 정하고 한 개의 span의 z값, Red, Green, Blue에 대한 intensity가 각각,

(Xl, Xr) = (2, 9)
(Z_Xl, Z_Xr) = (10H, 3ffH)
(R_Xl, R_Xr) = (0H, 7H),
(G_Xl, G_Xr) = (7H, 0H), (B_Xl, B_Xr)
= (3H, 0H)

인 경우에 대해 z값의 비교에 의해 얻은 메모리 write enable신호(그림11의 'Z_WE~'와 'RGB_WE~'신호)와 선형 보간에 의해 얻은 z값, 색도의 결과를 16진수로 보여주고 있다. 그림11을 보면, Xl('ZX_ADDR' or 'RGB_XA') = 2부터 Xr('ZX_ADDR' or 'RGB_XA') = 9까지, z값은 010, 0A0, 12F, ···, 3ff(hex) 로 색도중 Red는 0, 2, 2, ···, 7(hex)의 순으로, Grden 은 7, 6, 5, ···, 0(hex)의 순으로 Blue는 3, 3, 2, ···, 0(hex)의 순으로 선형보간되고 있음을 볼 수 있다.

V. 결　　론

컴퓨터 그래픽스에 앞어서 보이는 면들을 결정하고 보이는 면들의 색도값 계산을 위한 그래픽스 가속기인 shading 엔진을 설계하였고 Daisy 워크스테이션을 이용하여 동작을 검증하였다. 설계된 shading 엔진은 가려지는 면을 제거하기위한 알고리듬으로 z-버퍼 알고리듬을 사용하며 shading을 위한 알고리듬으로는 Gouraud shading 알고리듬을 지원한다.

설계된 shading 엔진은 64개의 화소들을 담당하여 이들에 대한 연산을 병렬로 수행하며 선형 보간기, 선형 보간기의 입력값을 만들어 주기 위한 곱셈기, 출력 데이타의 매스킹을 위한 EPT, 데이타와 메모리와의 인터페이스를 위한 입출력 버퍼등으로 이루어진다. 또한 shading 엔진이 모듈화되어 있기 때문에 사용자의 형편에 맞추어 여러가지 형태의 그래픽스 시스템을 구성할 수 있다.

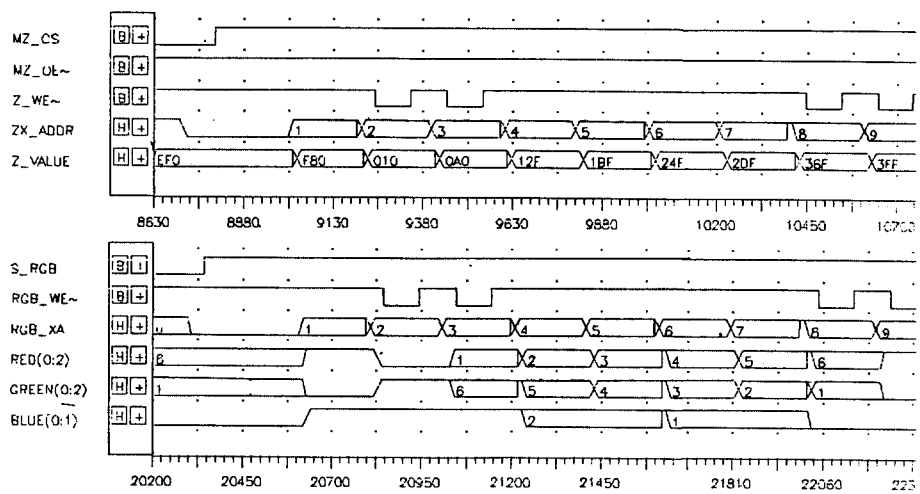


그림11. 시뮬레이션 결과
Fig. 11. Simulation result.

현재 설계된 shading 엔진은 시뮬레이션을 한 결과 10MHz의 동작주파수에서 최대 1초에 59,000여개의 span들에 대한 z값 비교와 색도계산을 할 수 있을 것으로 보이며, 앞으로 로직을 간소화하고 VLSI 칩으로 만든다면 성능이 더욱 향상될 것으로 기대된다. 최근에는 칩 제조의 전단계로서 수천 gate의 집적도를 갖는 FPGA(field programmable gate array) 칩을 10개 사용하여 shading 엔진을 구현한 후, 이를 이용한 IBM PC용 그래픽스 보드를 제작하여 실험중에 있다.

본 논문은 많은 계산량으로 인해 시간이 많이 걸리는 그래픽스에 있어서 계산시간을 줄여주기 위한 새로운 하드웨어 구조를 제안하고 실제 칩의 제작에 앞서 그 제안된 구조를 시뮬레이션 및 prototype 시스템의 제작을 통해 검증했다는 데 그 의의를 둘 수 있다.

參 考 文 獻

[1] S.S. Kim,K.S. Eo, and C.M. Kyung, "A new hardware architecture for fast image generation," *Electronics Letters,* 31st March, vol. 24, no. 7, pp. 382-383.

[2] H. Gouraud, "Continuous shading of curved surfaces," *IEEE Transactions on Computers,* vol. c-20, no. 6, pp. 623-629, June 1971.

[3] David F. Rogers, *Procedural Elements for Computer Graphics,* McGraw-Hill, 1985.

[4] N. Gharachorloo and C. Pottle, "Super buffer: A systolic VLSI graphics engine for real time raster image generation," *1985 Chapel Hill Conference on Very Large Scale Integration,* Computer Science Press, Inc., pp. 285-305.

[5] J. Poulton, H. Fuchs, J.D. Austin, J.G. Eyles, J. Heineeke, C.H. Hsieh, J. Goldfeather, J.P. Hultquist, S. Spach, "PIXEL PLANES: Building a VLSI-based graphic system," *1985 Chapel Hill Conference on Very Larse Scale Integration,* Computer Science Press, Inc., pp. 35-60.

[6] K.S. Eo, C.M. Kyung, and S.S. Kim, "PIPE: A hardware accelerator for scanline interpolation and hidden surface removal," *VLSI 89 IFIP Working Conference on Very Large Scale Integration,* Munich, FRG, August 16-18, 1989.

[7] Bui Tuong Phong, "Illumination for computer generated pictures," *Communications of the ACM, June 1975,* vol. 18, no. 6, pp. 311-317.

[8] Robert A. Goldstein and Roger Nagel, "3-D visual simulation," *Simulation, Jan.* 1971, pp. 25-31.

[9] R.F. Lyon, "Two's complement pipeline multipliers," *IEEE Transactions on Communications,* pp. 489-425, April 1976.

[10] Henry Fuchs, John Poulton, John Eyles, "Trey Greer, Jack Goldfeather, David Ellsworth, Steve Monlnar, Greg Turk, Brice Tebbs, Laura Israel, "Pixel-Planes 5:A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," *Computer Graphics,* vol. 23, no. 3, July 1989.

著 者 紹 介

方 卿 鎰(正會員)

1965年 10月 27日生. 1988年 서강대학교 전자공학과 공학사 학위 취득. 1990年 한국과학기술원 전기 및 전자공학과 석사학위 취득. 현재 삼성전자 중형 컴퓨터 개발실 근무. 주관심분야는 Computer Graghics. Computer System Architecture 등임.

裵 成 玉 (準會員) 第25卷 第10號 參照
현재 한국과학기술원 전기 및 전자공학과 박사과정.

慶 宗 昱(正會員)

1953年 6月 21日生. 1975年 서울대학교 전자공학과 공학사학위 취득. 1977年 한국과학기술원 전기 및 전자공학과 석사학위 취득. 1981年 한국과학기술원 전자공학과 박사학위 취득. 현재 한국과학기술원 전기 및 전자공학과 정교수. 주관심분야는 CAD, Computer Graphics, Neural Net. 등임.