논문 2010-5-14

Phong Shading 알고리즘을 적용한 3차원 영상을 위한 고속 그래픽스 가속기 연구

A Study on the 3 Dimension Graphics Accelerator for Phong Shading Algorithm

박윤옥*, 박종원**

Youn Ok Park, Jong-Won Park

요 약 2차원 그래픽을 3차원 그래픽으로 변환하기 위한 삼차원 그래픽 알고리즘들은 복잡하고 다양한 기법의 사용으로 인하여 대규모의 반복 연산이 요구되고, 이로 인하여 실시간 삼차원 그래픽의 처리가 어려운 경우가 많다. 본 논문은 삼차원 그래픽 처리와 관련된 여러 가지 알고리즘 중에서 Phong Shading 알고리즘의 병렬처리 방법과 고속 하드웨어 처리를 위한 삼차원 그래픽 가속기에 관한 것으로, Park's 다중접근 기억장치와 다수의 연산기로 구성된 SIMD처리기를 사용한 삼차원 그래픽 가속기 구조를 제안하고 있으며, 제안된 가속기 구조를 HDL을 사용한 시뮬레이션을 통해 본 논문에서 제안된 삼차원 그래픽 가속기에 의해 복잡한 알고리즘을 갖은 어떠한 삼차원 그래픽 알고리즘도 병렬 처리 알고리즘을 적용하여 SIMD 가속기에 의한 실시간 처리가 가능함을 보였다.

Abstract There are many algorithms for 2D to 3D graphic conversion technology which have the high complexity and large scale of iterative computation. So in this paper propose parallel algorithm and high speed graphics accelerator architecture using Park's MAMS(Multiple Access Memory System) for Phong Shading, one of many 3D algorithms. The Proposed SIMD processor architecture is simulated by HDL and simulated and got 30 times faster result. It means any kinds of 3D algorithm can make parallel algorithm and accelerated by SIMD processor with Park's MAMS for real time processing.

Key Words : Phong Shading, Graphics, Accelerator, Park's Memory, SIMD, MAMS

Ⅰ. 서 론

일반적으로 삼차원 그래픽을 처리하는 프로그램이나 삼차원 그래픽 가속기의 성능은 삼차원의 그래픽의 기본 요소인 Polygon의 초당 처리 개수로 표시한다[1]. 삼차원 그래픽 알고리즘을 고속화하기 위한 방법으로 Sun Microsystems 사는 이러한 문제를 해결하기 위한 방법으로 4개의 부동 소수점 연산자를 4개를 둔 그래픽 가속기 구조를 제안하고 있으며, ATI사는 그래픽의 연산 부분을 여러 개의 엔진으로 나누고, 각 엔진에서 알고리즘의 각 간계를 순차적으로 처리하는 구조 등을 제시하고 있다. 3D Labs 사는 64 Bit Hyper-Pipelined 구조와 2개의 연산기를 두어 그래픽 데이터를 처리하는 구조 등을 제시하고 있다. 이러한 기존의 가속기들의 공통된 특징은 순차적인 프로그램 수행을 기본으로 하고, 프로그램의 일부만 알고리즘 처리를 복잡한 연산 처리기에 분배하여 처리하는 구조를 하고 있다. 추가적으로 사용되는 방법은 연산의 처리를 pipeline 기법[2]을 사용하여 처리 속도를 증대 시키는 방법, 가속기 내의 버스 폭을 늘려 1

*정회원, 충남대학교 정보통신공학과
**정회원, 충남대학교 정보통신공학과
접수일자 2010.9.8 수정일자 2010.10.8
게재확정일자 2010.10.15

회의 입출력 데이터양을 늘리는 방법[11]등이 사용되고 있다. 그러나 이러한 구조들은 병렬화의 단위가 크고, 구조상으로 확장성이 없으며, 영상의 기본 구조인 pixel을 직접 사용하지 않고 직렬 방법에 의한 좌표 계산을 매 프로그램 마다 수행해야 하는 관계로 각 픽셀의 데이터를 pixel 별로 병렬 처리하지 못함으로 인한 변환에 따른 시간 손실이 대단히 크므로 인하여 처리 속도 증가에 한계가 있다. 이러한 한계성을 극복하기위해서는 고도의 병렬성과 데이터와 pixel 간의 일치성을 보장해주는 경우 연산 데이터와 영상 데이터 사이의 mapping 시간을 줄임으로 인하여 보다 고속의 그래픽 데이터 처리기를 구성할 수 있다. 따라서 본 논문에서는 병렬 기법을 사용한 삼차원 그래픽 가속기에서 파이프라인 기법[2], 병렬화 기법[5], 연산데이터와 영상데이터를 동일하게 할 수 있는 기술의 혼합된 형태[1] 제시하고 있다.

II. Phong Shading 알고리즘

그래픽에서 동일한 색의 물체라도 부분에 따라 밝기가 달라지기 마련이며 이러한 빛의 방향과 세기에 따라 달라지는 밝기는 빛의 반사를 체계적으로 고찰해 보면 물체의 표면에서의 반사현상은 다음 세 가지로 설명된다.

1. 빛의 분산(Diffuse Reflection)

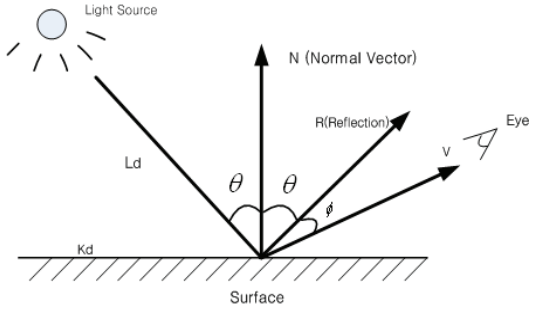

그림 1. 광원과 법선을 나타내는 벡터
Fig 1. Light Source and Normal Vector

물체는 빛을 완전히 분산시키기만 하는 물체의 표면은 관찰자의 위치에 상관없이 같은 밝기로 보이게 되며, 그 밝기는 물체 표면과 광원과의 위치, 그리고 각도에 따라 달라진다[7]. 이것을 수식으로 나타내면 다음과 같다.

식(1)은 식(2)를 벡터로 표현한 것으로, L은 표면에서 광원으로 향하는 방향의 벡터이고, N은 표면의 법선 벡터로서 그 길이는 모두 1이다.

$$I_d = I_0 * \cos\theta \quad (1)$$
$$= I_0 * K_d * (L * N) \quad (2)$$

I_d : 빛의 분산에 의한 물체 표면의 밝기
I_0 : 광원의 밝기
θ : 물체 표면의 법선 벡터와 광원의 빛이 이루는 각도
K_d : 물체의 색과 특성에 따라 다르게 주는 비례 상수

광원이 여러 개인 경우 다음과 같이 여러 광원에 대한 밝기를 합해주면 된다.

$$I_d = I_1 * K_d * \cos(1 + I_2 * K_d * \cos(2 + I_3 * K_d * \cos(3 + .. \quad (3)$$
$$= I_1 * K_d * (L_1 * N) + I_2 * K_d (L_2 * N) * I_3 * K_d * (L_3 * N) + \ldots \quad (4)$$

실제의 물체를 표현하려면 분산 현상의 효과뿐 아니라 반사 현상도 고려하여야 한다[11].

2. 배경 광원(Ambient Light)

실생활에서 빛을 직접 받지 못한 부분이라도 검게 보이지는 않는 것은 광원의 빛을 직접 받지 못한 물체라 하더라도 광원의 빛을 받은 여러 물체가 다시 퍼뜨린 약간의 빛을 받게 되기 때문이다. 이런 현상을 배경 광원(Ambient Light)이라 하고, 물체 표면의 밝기 계산에서 배경 광원의 밝기는 간단한 상수광 I_a로 나타낼 수 있다. 빛의 분산과 배경광원의 효과를 합하면 다음 식을 얻게 된다.

$$I = I_d + I_a * K_a \quad (5)$$
$$= I_o * K_d * \cos\theta + I_a * K_a \quad (6)$$

여기서 K_a는 물체에 따라 다르게 주는 비례상수이다. 광원뿐만 아니라 주변에 있는 빛의 효과를 완전히 표현하려면 모든 물체 표면에서의 빛의 흡수와 분산을 계산해 주어야 하는데 이러한 방법을 "Radiosity Method"라 한다[6]. 이 방법으로 얻은 영상은 현실감이 가장 높은 것으로 알려져 있다. 그러나 현실감이 높은 만큼 이론과 계산 과정이 복잡하고 계산량도 Ray Tracing 이상으로 많다.

3. 직접적인 반사(Specular Reflection)

물체의 색과는 상관없이 광원의 색이 나타나는 현상이 발생하는 하이라이트 부분은 광원의 빛을 직접 반사시킨 부분이므로 물체의 색보다는 광원의 색을 반사한다. 이러한 현상을 Specular reflection 이라 한다[6].

4. Phong Shading 알고리즘

물체의 한 면을 일정한 밝기로 그려내는 데는 (Constant Shading) 대략 2단계의 작업을 필요로 한다. 1단계는 시각 변환과 투영에 의해 물체의 면을 화면상의 다각형으로 변화시키는 것이고, 2단계는 화면상에서 다각형의 내부를 같은 색으로 칠하여 한 면을 완성시키는 것이다. 다각형의 내부를 칠하는 작업은 보통 화면상의 가로줄(Scan Line)단위로 이루어진다. 색의 변화를 계산하는 방업으로 대표적인 것으로 Gouraud Shading과 Phong Shading법이 있다.

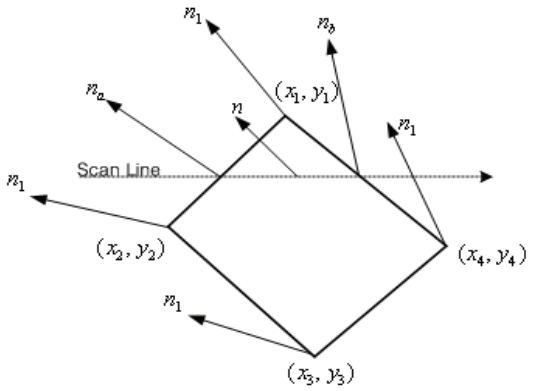

그림 2. Gouraud Shading 및 Phong Shading 에서 꼭짓점과 가중평균의 관계
Fig 2. Relation of weighted average in Gouraud Shading and Phong Shading

가. Gouraud Shading

Phong Shading 알고리즘의 병열화의 필요성을 설명하기 위해서는 Phong Shading에 비하여 사실감이나 정밀도가 떨어지는 Shading 방법 중의 하나인 Gouraud Shading 에 대한 설명이 필요하다. Scan Line 단위로 시작점과 끝점 사이를 칠해나가는 방법 중에서 화소의 색을 부드럽게 변화 시키는 방법으로 제일 간단한 것은 시작점과 끝점의 밝기를 미리 계산해 두고 그 사이의 밝기를 두 밝기의 가중 평균으로 구하는 것이다. 이 방법을 Gouraud Shading이라 한다[3].

$$I_1 = I_a + (I_b - I_a) * (y - y_a)/(y_b - y_a) \quad (7)$$
$$I_2 = I_c + (I_b - I_a) * (y - y_c)/(y_d - y_c) \quad (8)$$
$$x_1 = x_a + (x_b - x_a) * (y - y_a)/(y_b - y_a) \quad (9)$$
$$x_2 = x_c + (x_d - x_c) * (y - y_c)/(y_d - y_c) \quad (10)$$

여기서 I_a, I_b, I_c, I_d 는 법선 벡터에 의해 구한 밝기이다.

나. Phong Shading

Gouraud Shading 알고리즘[3]은 밝기를 가중 평균으로 구하므로 비교적 간단하다는 장점이 있는 반면 물체의 하이라이트가 제대로 나타나지 않는 다는 단점이 있다. Phong Shading 이란 밝기의 가중 평균 대신 법선 벡터를 가중 평균하여 화소 하나하나마다 Shading 모델을 적용시키는 방법이다. 이 방법은 영상의 현실감을 늘릴 수 있는 반면 밝기 대신 3차원 벡터를 다루므로 가중평균 계산이 Gouraud Shading의 약 3배의 연산양이 필요하고, 화소마다 Shading 모델에 의한 음영계산을 해주어야 하므로 계산양이 대폭 늘어나게 된다.

$$n_{x_1} = n_{x_A} + (n_{x_B} - n_{x_A}) * (y - y_A)/(y_B - y_A) \quad (11)$$
$$n_{y_1} = n_{y_A} + (n_{y_B} - n_{y_A}) * (y - y_A)/(y_B - y_A) \quad (12)$$
$$n_{z_1} = n_{z_A} + (n_{z_B} - n_{z_A}) * (y - y_A)/(y_B - y_A) \quad (13)$$
$$n_{x_2} = n_{x_C} + (n_{x_B} - n_{x_A}) * (y - y_C)/(y_D - y_C) \quad (14)$$

$$n_{y_2} = n_{y_C} + (n_{y_D} - n_{y_C}) * (y - y_C)/(y_D - y_C) \quad (15)$$
$$n_{z_2} = n_{z_C} + (n_{z_D} - n_{z_C}) * (y - y_C)/(y_D - y_C) \quad (16)$$

$$x_1 = x_A + (x_B - x_A) * (y - y_A)/(y_B - y_A) \quad (17)$$
$$x_2 = x_C + (x_D - x_C) * (y - y_C)/(y_D - y_C) \quad (18)$$

$$n_x = n_{x_1} + (n_{x_2} - n_{x_1}) * (y - y_1)/(y_2 - y_1) \quad (19)$$
$$n_y = n_{y_1} + (n_{y_2} - n_{y_1}) * (y - y_1)/(y_2 - y_1) \quad (20)$$
$$n_z = n_{z_1} + (n_{z_2} - n_{z_1}) * (y - y_1)/(y_2 - y_1) \quad (21)$$

이 식들을 종합하면 Phong Shading[10] 알고리즘을 수행하기 위해서는 다음과 같은 연산이 필요하다.

3개의 부동 소수점 연산 + 3개의 평방근 + 법선 벡터의 Normal 값 계산 + Intensity의 재계산 + 3 * 10개의 부동 소수점 곱셈의 계산이 필요하므로 특별히 다각형 내부에 존재하는 고화질의 하이라이트를 표현하고자 하는 경우 사용된다.

Ⅲ. 제안하는 FFT 프로세서의 알고리즘 및 하드웨어 구조

1. 고속화를 위한 Phong Shading 병렬화 방법

Phong Shading에서는 각각의 Polygon 단위로

Shading한다. Polygon의 크기와 모양은 일정하지 않고, 대부분의 모델은 삼각형 또는 사각형 모양의 다면체로 구성된다. 그림 3은 일반적인 형태의 Polygon과 Phong Shading이 적용되는 형태를 도시하고 있다.

그림 3과 같은 형태의 Polygon에 대한 Phong Shading 알고리즘을 적용할 경우 Polygon내의 Pixel 수만큼 계산을 반복적으로 해주어야 한다. 연산은 Scan Line 상의 각 Pixel 의 값을 계산해야 한다. 각 Polygon은 형태가 일정하지 않은 다각형으로써 좌측과 우측의 데이터가 수직의 Pixel 의 Shading 시 서로 다른 데이터와 Pixel의 수를 갖고 있기 때문에 병렬화 하기에 부적합한 구조를 각조 있다. 따라서 일반적인 가속기에서는 Polygon을 형성하는 각 vertex 점들의 값을 가속기에 넘겨주고 Polygon 을 처리하도록 한다. 처리된 데이터는 다시 각 Pixel 의 값을 저장함으로써 계산이 완료된다.

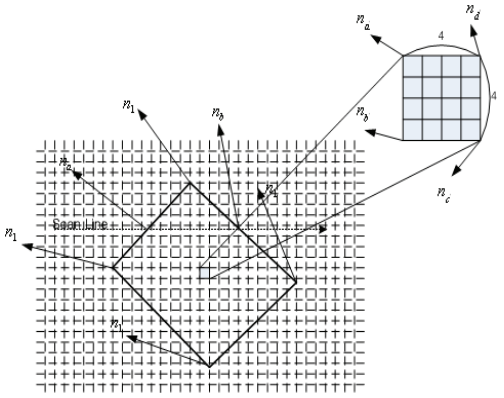

그림 3. 4x4의 크기를 갖는 정방형 Polygon으로의 분해 방법

Fig 3. Decomposition method of rectangular polygon with 4x4 size

본 논문에서는 이러한 일반적인 Polygon을 병렬처리화에 적합한 방법을 제시한다. 병렬화를 위해서는 Polygon이 일정한 크기와 규칙적인 배열성을 갖아야 한다[4]. 이러한 규칙성, 배열성, 일정한 크기를 부여하기 위하여 본 논문에서는 주어진 Polygon을 다시 4 x 4 개의 Pixel을 갖는 정방형의 Polygon으로 분해하는 방법을 사용한다. 4x4개의 pixel로 분해된 새로운 polygon은 병렬처리기로 보내져 각 PE(Processing Element)로 하여금 1개의 Pixel을 맡아 처리하도록 구성하였다. 그림 4는 일반적인 형태의 Polygon을 일정한 크기의 정방형 Polygon으로 나누는 방법을 도시하고 있다. 정방형의 크기로 Polygon을 분해할 경우 발생할 수 있는 문제는 X, Y 좌표축 상에서 사선에 위치한 정방형의 Polygon의 처리에 관한 문제가 발생한다.

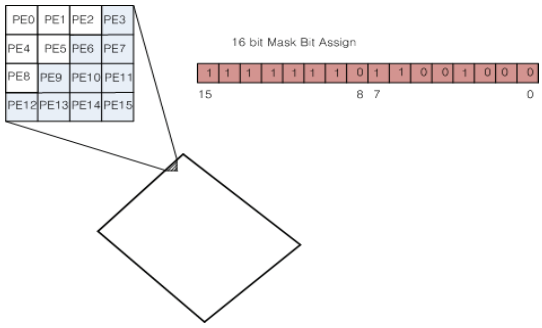

그림 4. 사선에 위치한 정방형 Polygon의 마스크 비트 할당

Fig 4. Assignment of mask bit for polygon located in diagonal line

이러한 문제는 그림 4. 에 도시한 방법에 의하여 재구성한다. 구성된 Polygon에서 사선 밖의 pixel에 대해서는 PE로 하여금 연산하지 않도록 하는 마스크 비트를 두어 마스크 비트의 값이 '0'의 값을 갖는 경우에는 연산하지 않도록 한다.

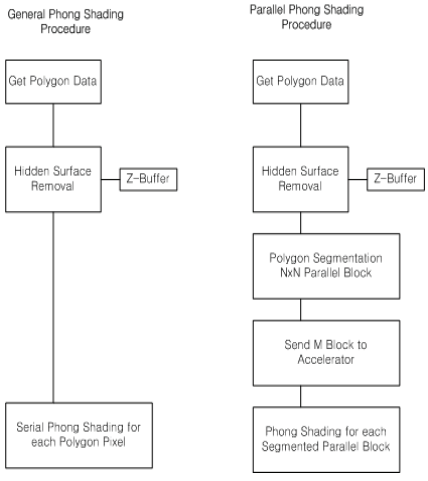

그림 5. Phong Shading 알고리즘의 일반적인 처리 및 병렬화 알고리즘 적용 처리 방법

Fig 5. Procedure of general and parallel processing for Phong Shading Algorithm

본 논문에서 제안하는 병렬화 알고리즘은 그림 5의 과정으로 수행된다.

2. SIMD 처리기의 구조

SIMD 처리기는 호스트와 데이터 및 명령 전송을 위한 PCI 버스와 PE, 병렬 메모리로 구성되어 있다. PCI 버스인터페이스는 128Mbytes/s의 전송 속도로 데이터 전송을 수행하며, PE는 알고리즘의 최종 연산 부분을 수행하도록 구성하였으며, 병렬 메모리는 16비트의 칼라를 표현할 수 있도록 하였으며, 최대 1024 x 764 크기의 프레임을 지원하도록 구성되어 있다. SIMD 처리기를 중심으로 한 삼차원 그래픽 가속기는 그림 6과 같다.

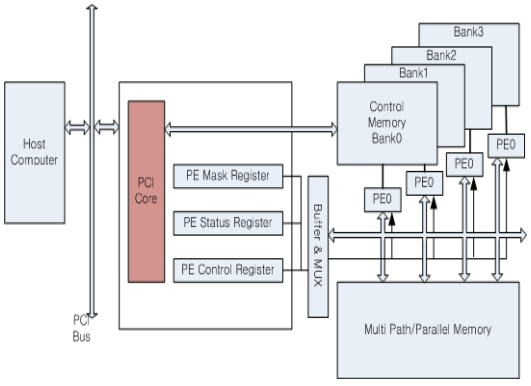

그림 6. 삼차원 그래픽 가속기 구조도
Fig 6. Architecture of 3-Dimension graphics accelerator

PCI 인터페이스에서 생성되는 어드레스와 데이터 신호는 PCI 인터페이스 내의 제어 및 상태 레지스터와 병렬 처리기가 공유하고 있으며, PCI 인터페이스 내의 레지스터는 PE 제어 레지스터와 PE 상태 레지스터, 그리고 Base Address 레지스터로 구성된다. 본 논문에서 설계한 PCI 버스 인터페이스는 Memory Mapped I/O 모드로 엑세스하는 것을 기본으로 하였다. PE는 4개의 PE를 1개의 그룹으로 한 4개의 PE그룹을 사용하도록 구성하였다.

각각의 PE 그룹은 4x4 정방형 Polygon의 4개 스캔 라인상의 4 Pixel을 한 개씩 맡아 처리하도록 하였다. Park's 병렬 메모리[4][5]는 16개의 PE에서 생성된 Shading 값을 저장하되 Block Mode로 저장하게 된다.

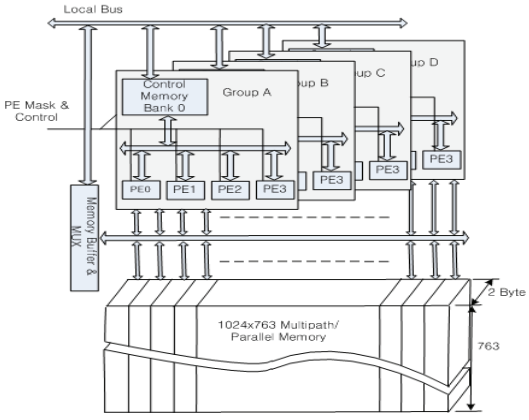

그림 7. SIMD 처리기와 다중 접근 메모리의 구성도
Fig 7. SIMD Processor and Multiple Access Memory system architecture

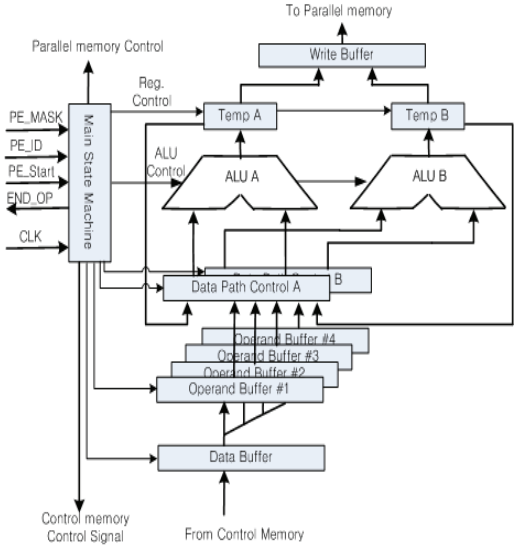

그림 8. Phong Shading 알고리즘 처리를 위한 PE의 구조
Fig 8. PE Architecture for Phong Shading Algorithm

PCI 인터페이스의 로컬 데이터 라인과 병력 메모리의 연결은 16 to 2 MUX 기능을 갖는 버퍼로 연결된다.

Park's 다중 접근 메모리[5]에 저장된 Shading 된 Pixel의 값을 읽기 위해서는 다중 접근 기억 장치의 액세스 모드로 수평 모드로 하여 읽게 되며, 한 번에 출력되는 데이터는 2x16 Bytes가 된다. 그러나 PCI 버스는 32비트버스이므로 한 번에 읽을 수 없고 8번에 걸쳐 나누어 가져가야 하므로 MUX기능을 필요로 하게 된다. PE는 복수

의 곱셈기, 덧셈기의 조합으로 구성되어 있으며 계산의 단계별 데이터는 별도의 레지스터를 사용하여 저장하고 제어 메모리는 최초 연산기에 필요한 입력 데이터를 읽어오는 경우만 액세스하고, 연산에 필요한 초기 값을 로드한 이후에는 액세스하지 않는다. PE의 구성은 상태제어기 형태로 구현되며, 연산 속도의 증가를 위해 2개의 ALU를 두어 연산 속도를 2배로 늘리도록 구성하였다. 상태제어기는 컨트롤 메모리로부터 입력되는 데이터를 Operand Register 에 저장 , 데이터의 흐름, ALU의 동작 모드를 제어하며, 최종 연산 결과를 병렬 처리 메모리로 출력하도록 구성하였다.

3. Park's 다중 접근 기억 장치

본 논문에서 제안하는 알고리즘을 적용하기 위해서는 Park's 다중 접근 기억 장치[4][5]를 사용하는 경우 가장 효율적인 액세스 모드들 제공할 수 있다. PE에서의 출력 데이터를 화면의 1024x764 Pixel과 1대1 대응시킴으로 인하여 별도의 계산 없이 frame memory와 결과를 일치시킬 수 있기 때문이다.

V. 결 론

본 논문에서는 SIMD처리기에 적합하고 병렬화가 가능한 알고리즘을 개발하였고, 반복적이면서도 연산량이 많은 부분을 SIMD 처리기에서 병렬 처리함으로써 속도면에서 상당한 성능이 향상되었음을 확인할 수 있었다.

표 1. Phong Shading에서 Shading처리 부분의 직렬 연산과 병렬 연산에 의한 수행 시간 비교

Table 1. Operating time comparison between serial computation and parallel operation

	직렬연산	직렬 수행시 연산 회수	병렬 연산 수행시 연산 회수
64 Pixel의 계산 값	For(8) For(8)	덧셈 12회x64 곱셈 20회x64 나눗셈 5회x64	덧셈 12회x4 곱셈 20회 x 4 나눗셈 5회 x 4
시간 지연요소		덧셈 10ns 곱셈 30ns +기억장치 액세스 지연시간(70ns)	덧셈 2ns 곱셈 12ns 나눗셈 32ns PCI버스 액세스 지연(80ns)
수행시간 (16PE사용시)		52us	1.7us

또한 SIMD처리기를 Park's 병렬 처리 메모리와 결합하여 그래픽 특성에 맞는 임의 방향으로의 병렬 액세스가 가능한 구조를 제안하여 고속의 Polygon 처리가 가능하도록 하였다.

향후 처리 속도 증대를 위한 PE와 Park's 병렬 처리 메모리를 대규모로 적용하여 실시간 처리가 가능한 SIMD구조가 제안될 경우 2D의 영상자원을 3D로의 변환이 필요한 Video 및 의학 영상처리 등에 적용할 수 있으므로 추가적인 연구가 필요하다.

참 고 문 헌

[1] H. K. Reghbati and A.Y.C. Lee, "Computer Graphics hardware : Image Generation and Display," Computer Society Press, 1988

[2] A. Fujimoto, G. P. Christopher, and K. Iwata, "A 3-D Graphics display system with depth buffer and pipeline processor," IEEE Computer Graphics and Applications, p.p.11-23, June 1984

[3] J. D. Foley, A. Van Dam, S. K. Feiner, and J. F. Huges, "Computer Graphics : Principles and Practice," 2nd edition, Addison-Wesley Publishing Company, 1996

[4] 박종원, "영상처리를 위한 다중 접근 기억 구조," 한국과학기술원 박사학위논문, 1991년.

[5] J. W. Park, "An Efficient Memory system for image processing," IEEE Trans. Computers, Vol. C-35, No.7, pp. 169-174, July 1986

[6] B. MacIntyre, "PC 3D Graphics accelerator FAQ," http://www.cs.colombia.edu/~bm/3dcards/dd-cards1.html.

[7] J. Clark, " The Geometry engine : a VLSI geometry system for graphics," Proc. NATO Advanced Study Institute on Microarchitecture of VLSI Computers, p.p.189-205, 1985

[8] W. Myers, "Staking out the graphics display pipeline," IEEE Computer Graphics and Applications, p.p.60-65, July 1984

[9] N. England,"A graphics system architecture for interactive application-specific display functions,

"IEEE Computer Graphics and Applications, p.p.60-70, Jan. 1986

[10] Gary Bishop and David M. Weiner,"Fast Phong Shading," SIGGRAPH '86 Proceedings, Vol 20, p.p. 103-106, Aug. 1986

[11] Dipak Ghosal and L. M. Patnaik,"Parallel Polygon Scan Conversion Algorithms : Performance Evaluation On A Shared Bus Architecture," Computers and Graphics, Vol 10, No. 1, p.p. 7-25, 1986

[12] Arie Kaufman,"Memory Organization for a Cubic Frame Buffer," Eurographics '86, p.p.93-100, Aug. 1986

[13] Frederik W. Jansen,"CSG Hidden Surface Algorithm for VLSI Hardware Systems," Advances in Computer Graphics Hardware I, Record of First Eurographics Workshop on Graphics Hardware, p.p. 75-82, 1986

[14] William H. Press and Brain P. Flannery, "Numerical Recipes in C," Cambridge University Press, 1988

저자 소개

박 윤 옥(정회원)

• 학위(석사)
• 경력
- 1986년 한양대 전자공학과 학사
- 1997년 충남대 컴퓨터공학과 석사
- 2001년~현재 충남대 정보통신공학과 박사과정
- 1987년 ~ 현재 한국전자통신연구원 책임연구원

<주관심분야 : 영상처리 기술, 이동통신 시스템, 안테나 및 전파전파, 통방융합기술>

박 종 원(정회원)

• 학위(박사)
• 경력
- 1997년 충남대 전자공학과 학사
- 1981년 KAIST 전산학과 석사
- 1991년 KAIST 전산학과 박사
- 1981년~1993년 충남대 전산학과 부교수
- 1994년현재 충남대학교 공과대학 정보통신공학과 교수

<주관심분야 : 컴퓨터구조, 영상처리, 컴퓨터비전>