논문 2009-46SD-10-7

# Multi-Thread 쉐이더 구조에 적합한 Vector 기반의 Rasterization 알고리즘의 구현

## ( Implementation of a 'Rasterization based on Vector Algorithm' suited for a Multi-thread Shader architecture )

이 주 석*, 김 우 영**, 이 보 행**, 이 광 엽***

( Ju-Suk Lee, Woo-Young Kim, Bo-Haeng Lee, and Kwang-Yeob Lee )

요 약

현재 개발되고 있는 Shader 프로세서는 처리 성능을 높이기 위하여 Multi-Core, Multi-Thread를 채택하고 있다.[1] 또한 Shader 프로세서에서 각 수행 단계별 마다 IP를 따로 구현하지 않고 하나의 Core IP를 다양한 목적으로 사용할 수 있도록 설계하고 있다.[2] 본 논문에서는 이러한 목적에 맞게 Shader-Core를 이용하여 연산이 가능하고, Multi-Core, Multi-Thread 기반에서 픽셀의 병렬처리가 가능하도록 고안된 Vector 기반의 Rasterization알고리즘을 제안한다. 이를 통하여 동일 조건의 기존 알고리즘에 비하여 약 2%의 연산량을 가지면서 각 픽셀이 독립적으로 연산이 가능하도록 하였다.

Abstract

A Multi-Core/Multi-Thread architecture is adopted for the Shader processor to enhance the processing performance.[1] The Shader processor is designed to utilize its processing core IP for multiple purposes, such as Vertex-Shading, Rasterization, Pixel-Shading, etc.[2] In this paper, we propose a 'Rasterization based on Vector Algorithm' that makes parallel pixels processing possible with Multi-Core and Multi-Thread architecture on the Shader Core. The proposed algorithm takes only 2% operation counts of th Scan-Line Algorithm and processes pixels independently

**Keywords :** Graphics, Shader, Rasterization, Vector, Multi-thread

## Ⅰ. 서 론

그래픽의 활용 분야가 광범위해짐에 따라 처리 하드웨어도 계속해서 발전해 왔다. 또한 더욱 빠른 처리를 위하여 Multi-Thread, Multi-Core로 발전하여 많은 양의 데이터를 병렬로 동시에 처리할 수 있도록 하고 있다. 최근에는 GP-GPU[3]개념으로 그래픽 처리 프로세서를 그래픽 처리뿐 아니라 많은 양의 연산이 필요한 다른 용도로 활용까지 가능해졌다. 이를 바탕으로, 그래픽 파이프라인을 구성하는 대부분의 기능을 명령어로 처리하는 구조로 발전하고 있다. 특히 Rasterizer를 전용하드웨어를 사용하지 않고 프로세서 명령어로 대체하고 있는데, Rasterizer는 많은 양의 연산을 필요로 하기 때문에 최적의 알고리즘 개발을 통하여 명령어의 실행 사이클 수를 줄일 수 있다. 본 논문에서 제안하는 Vector기반의 Rasterization 알고리즘은 위와 같이 Multi-Thread, Multi-Core를 채택한 Shader 프로세서에서 Rasterization을 수행할 경우 여러 개의 Core, 여러 개의 Thread에서 각각의 픽셀들을 병렬로 처리할 수 있도록 픽셀들 간의 연계성을 최소한으로 줄였다.

* 정회원, 충북테크노파크
(CHUNGBUK TECHNOPARK)
** 학생회원, *** 정회원-교신저자, 서경대학교 컴퓨터공학과
(Computer Engineering, Seokyeong University)
※ 본 논문은 지식경제부 시스템 2010 사업과 ETRI 시스템반도체 진흥 센터의 지원으로 작성하였으며 IDEC 지원 장비를 활용하였습니다.
접수일자: 2009년5월14일, 수정완료일: 2009년10월1일

이를 통하여 하나의 폴리곤 내의 모든 픽셀이 모두 독립적으로 처리가 가능하다.

## Ⅱ. 그래픽 파이프라인에서 Rasterization

그래픽 파이프라인 중 가장 핵심이 되는 세 개의 단계가 있다. 첫 번째는 Vertex-Shading, 두 번째는 Rasterization, 세 번째는 Pixel-Shading단계이다.

Vertex-Shading 단계에서는 모델의 정점좌표(Vertex)정보를 입력으로 받아 User가 정의한 정점처리 프로그램을 수행한다.

Rasterization 단계에서는 Index를 참조하여 Vertex-Shading 단계에서 처리가 완료된 정점좌표들 중 세 개의 정점좌표를 조합하여 하나의 폴리곤을 구성한다. 그 후 폴리곤 내의 모든 픽셀들에 대해서 Color나 Texture-Coordinate, Normal등의 속성 정보들을 Interpolation하여 모델 내의 모든 픽셀들이 고유의 속성정보를 가질 수 있도록 한다.

이렇게 Rasterization 단계가 완료되어 모든 픽셀들이 고유의 속성 정보를 가지게 되면 Pixel-Shading 단계에서 각 픽셀들에 대한 속성 정보들을 이용하여 User가 정의한 픽셀처리 프로그램을 통해 각 픽셀들에 다양한 효과를 부여하게 된다.

이와 같이 Rasterization은 그래픽 처리단계에서 중요한 부분을 담당하고 있으며 하나의 모델을 구성하는 모든 폴리곤에 대하여 구성하는 모든 픽셀의 속성 정보들을 Interpolation해야 하므로 많은 양의 연산을 필요로 한다. 이와 같은 이유로 Rasterization 단계의 가속화를 위한 다양한 알고리즘이 연구되고 있다.

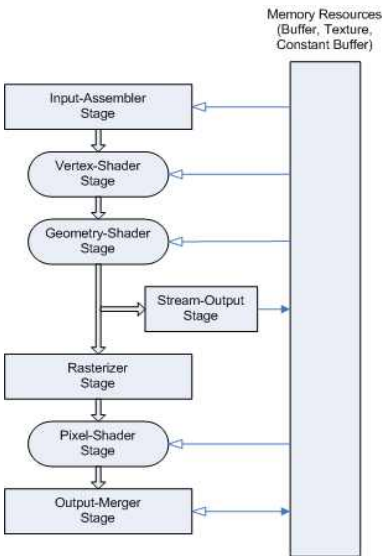



그림 1. Shader Graphics Pipeline[4]
Fig. 1. Shader Graphics Pipeline[4].

## Ⅲ. 기존 Rasterization 알고리즘

Rasterization은 앞에서 본 바와 같이 폴리곤 내의 모든 픽셀에 대해서 각 픽셀의 속성 정보들을 구하는 과정으로 그래픽 처리과정에서 꼭 필요한 단계이다. 기존에 가장 많이 이용된 알고리즘으로 Scan-Line 알고리즘이 있다. 이는 과거 모니터의 주사 방식에서 착안한 것으로써 주사선이 뿌려지는 방식과 같이 첫 번째 줄 가장 왼쪽에서부터 순차적으로 한 줄씩 Interpolation하는 방법이다.

Scan-Line 알고리즘의 기본은 폴리곤의 Edge와 Scan-Line과의 교점을 바탕으로 두 교점 사이를 Interpolation하는 방법이다. 그림 2는 이러한 Scan-Line의 기본 Interpolation을 나타내고 있다.

Scan-Line의 Interpolation은 폴리곤 Edge와 Scan-Line의 교점 중 시작 픽셀인 $V_{scan0}$의 속성 정보와 마지막 점인 $V_{scan1}$의 속성 정보의 변화량을 x의 변화량으로 나눔으로써 x값이 1 증가할 때마다 각 속성 정보의 변화량을 계산하여 x값을 1씩 증가시켜가며 각 속성 정보의 단위변화량을 더해주어 해당 픽셀의 속성 정보를 계산한다.

이러한 Scan-Line의 Interpolation을 폴리곤의 가장 작은 y값을 가지는 정점으로부터 시작하여 가장 큰 y값을 가지는 정점까지 반복하여 줌으로써 폴리곤 내의 모든 픽셀을 Interpolation하여 각 픽셀의 속성 정보를 계산한다.

이러한 Scan-Line 알고리즘은 이전 픽셀의 속성 정

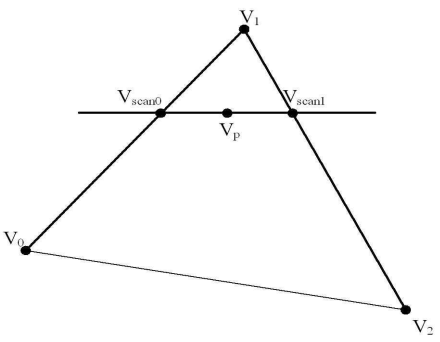



그림 2. Scan-Line 알고리즘의 Interpolation
Fig. 2. Interpolation with Scan-Line Algorithm.

보를 이용하여 현재 픽셀의 속성 정보을 계산하기 때문에 반복적으로 필요한 많은 계산과정이 생략되어 연산량이 줄어들어 비교적 간단한 계산과정을 통하여 구할 수 있다. 또한 복잡한 곱셈이나 나눗셈 연산이 적고 대부분이 덧셈 연산만으로 이루어져 있어서 연산에 대한 부담도 적어진다는 장점이 있다.

그러나 이러한 Scan-Line 알고리즘의 가장 치명적인 단점은 폴리곤 내의 모든 픽셀들이 매우 긴밀한 연계성을 가지고 있다는 것이다. 즉, 모든 픽셀이 순차적으로 처리되면서 이전 값에 변화량을 더해주어 현재 값을 구하는 방식이므로, 이전 픽셀이 계산되기 전까지는 현재 픽셀을 계산할 수 없다.

하나의 폴리곤을 구성하는 많은 픽셀들을 Interpolation하는 과정을 병렬적으로 처리하기 위해서는 각 픽셀들이 독립적으로 연산이 가능해야 하지만 Scan-Line알고리즘의 경우에는 앞에서 설명한 바와 같이 이전 픽셀이 처리된 이 후에 현재 픽셀의 처리가 가능하기 때문에 병렬처리가 불가능하다. 현재 개발되고 있는 고성능 그래픽 프로세서는 Multi-Thread, Multi-Core 구조를 채택하여 이를 바탕으로 많은 양의 그래픽 처리 데이터를 병렬적으로 처리하도록 발전하고 있는데, 기존 Scan-Line 알고리즘은 앞에서 설명한 픽셀간의 연계성으로 인하여 Multi-Thread, Multi-Core 구조에는 매우 부적합한 알고리즘이다. 따라서 Rasterization 단계의 처리를 위해서는 이를 위한 추가적인 H/W가 필요하다.

본 논문에서는 Scan-Line알고리즘의 단점으로 지적된 픽셀간의 연계성을 최소한으로 줄이고 하나의 폴리곤 내에서 모든 픽셀이 독립적으로 병렬처리가 가능하도록 하여 Multi-Thread, Multi-Core를 바탕으로 하는 고성능 그래픽 프로세서에서 추가적인 H/W 없이 Rasterization 단계를 수행할 수 있도록 하는 Vector기반의 Rasterization알고리즘을 제안한다.

## Ⅳ. Vector기반 Rasterization 알고리즘

제안하는 Vector기반 Rasterization 알고리즘은 Multi-Thread 및 Multi-Core기반에서의 Rasterization의 효율성을 높이기 위하여 하나의 폴리곤 내에서 어떠한 픽셀이라도 동시에 처리가 가능하도록 고안되었다.

### 가. 폴리곤 내 픽셀의 Vector 표현

폴리곤은 정점 3개로 구성된 하나의 삼각형으로써 Rasterization 단계를 통하여 내부에 각 픽셀들의 정보들을 구하게 된다. 제안하는 Vector방식의 Rasterization 알고리즘은 폴리곤 내의 한 픽셀을 세 개의 정점좌표들을 이용한 두 개의 Vector를 통하여 표현이 가능한 점을 기반으로 한다.

그림 3은 세 개의 정점 V0(x0, y0), V1(x1, y1), V2(x2, y2)로 구성된 하나의 폴리곤을 나타내며, Vp(xp, yp)는 이 폴리곤 내의 임의의 한 픽셀을 나타내고 있다.

이 폴리곤에서 V0와 V1의 두 정점을 이용하여 하나의 Vector $\overrightarrow{Vt0}$를 구성하고 V0와 V2를 이용하여 또 다른 하나의 Vector $\overrightarrow{Vt1}$를 구성 한다면, 폴리곤 내의 한 픽셀 Vp는 $\overrightarrow{Vt0}$와 $\overrightarrow{Vt1}$ 두 Vector를 통하여 표현이 가능하다.

그림 4와 같이 폴리곤 내의 한 픽셀 Vp는 2개의 Vector의 합으로 표현이 가능하다. 또한 이 두 개의 벡터는 앞에서 정의한 $\overrightarrow{Vt0}$, $\overrightarrow{Vt1}$를 각각 a배, b배로 축소하여 표현할 수 있다. 즉 $\overrightarrow{Vt0}$을 a배로 축소한 벡터와 $\overrightarrow{Vt1}$을 b배로 축소한 두 벡터의 합은 픽셀 Vp를 나타내는 것이다. 이를 식으로 표현하면 수식 1과 같다.

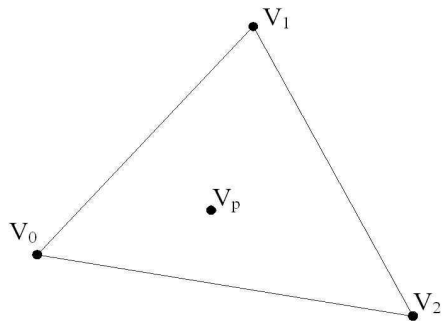



그림 3. 세 정점으로 구성된 폴리곤
Fig. 3. A Polygon of three Vertices composition.

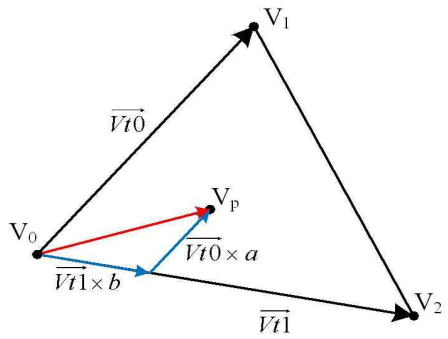



그림 4. Vector를 이용한 폴리곤 내부 픽셀의 표현
Fig. 4. Expression of pixel in polygon using the Vector.

$$V_p = V_0 + (V_1 - V_0) \times a + (V_2 - V_0) \times b \quad (1)$$

수식 1에서의 a와 b는 폴리곤 내의 모든 픽셀이 다른 값을 가지며, 이 a, b값을 통하여 해당 픽셀의 Color나 Texture u, v좌표, Normal Vector등 다른 속성 정보들의 Interpolation이 가능하다.

### 나. Vector를 이용한 Interpolation

앞에서 본 바와 같이 Vector를 이용한 픽셀 표현을 통하여 해당 픽셀을 표현하는 a, b값을 구하게 되면 Color나 Normal과 같은 속성 정보의 Interpolation을 수행할 수 있다. 이러한 Interpolation을 위해서 우선 a와 b를 구하는 식을 살펴보자.

앞에 수식 1에서 Vp를 표현하는 식을 각 x와 y로 나누어 살펴보면 수식 2와 같이 표현 할 수 있다.

$$\begin{aligned} x_p &= x_0 + (x_1 - x_0) \times a + (x_2 - x_0) \times b \\ y_p &= y_0 + (y_1 - y_0) \times a + (y_2 - y_0) \times b \end{aligned} \quad (2)$$

위의 두 식을 연립하여 정리하면 a와 b는 수식 3과 같이 구할 수 있다.

$$\begin{aligned} a &= \frac{(x_2 - x_0)(y_p - y_0) - (y_2 - y_0)(x_p - x_0)}{(x_2 - x_0)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_0)} \\ b &= \frac{(x_1 - x_0)(y_0 - y_p) - (y_1 - y_0)(x_0 - x_p)}{(x_2 - x_0)(y_1 - y_0) - (x_1 - x_0)(y_2 - y_0)} \end{aligned} \quad (3)$$

이렇게 a와 b를 구하면 이를 이용한 해당 픽셀의 속성 정보들의 Interpolation은 수식 1을 이용하여 구하면 된다. 수식 4는 픽셀의 속성 정보들 중 Color값의 Red를 구하는 식을 나타낸 것이다.

$$Rp = R0 + (R1 - R0) \times a + (R2 - R0) \times b \quad (4)$$

이 외의 다른 속성 정보들 역시 같은 방법으로 Interpolation이 가능하다. 일반적으로 Graphic Processor Unit의 경우 4D-Vector의 형태로 연산이 가능하도록 설계가 되기 때문에 Color속성의 경우 Red, Green, Blue, Alpha를 각각 따로 연산 할 필요 없이 Color 속성 정보를 한 번에 연산 할 수 있다.

### 다. Perspective Correction

Rasterization의 과정 중 Interpolation 단계는 3차원 좌표계의 모델을 구성하는 폴리곤을 2차원 좌표계의 View-Plane에 투영한 이 후에 수행되는 과정이다. 이 때 'Devide by W'의 과정이 포함된다. 그러나 앞에서 본 Interpolation의 결과는 Devide by W의 과정이 생략된 결과이다.

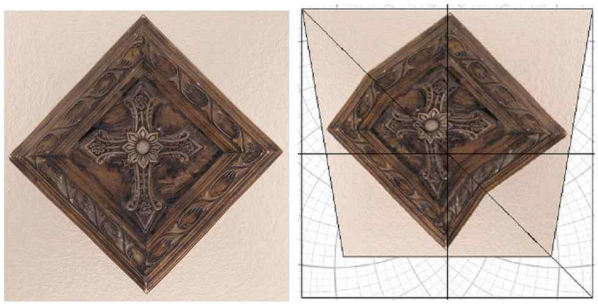


그림 5. Perspective-Correction을 고려하지 않은 Interpolation 오류

Fig. 5. Error of Interpolation without the Perspective-Correction.

그림 5는 폴리곤 2개로 이루어진 3차원 공간내의 사각형에 왼쪽 Texture-Image를 Mapping한 화면이다. 이 사각형에서 위의 두 정점에 비해 아래 두 정점이 더 큰 $w_p$값을 가질 경우 View-Plane에 투영 시 원근법에 의해 아래 두 정점으로 갈수록 작아지는 형태를 가지고 있다. 이 때 앞의 Interpolation을 통해 Mapping 할 경우 $w_p$값이 고려되지 않아 오른쪽 그림과 같은 오류를 보이고 있다. 이러한 오류를 수정하기 위해 w값의 정보를 고려하여 Interpolation을 진행할 필요가 있다.

Perspective-Correction을 위해서는 앞에서 설명한 Interpolation과정에 $w_p$값을 고려하여 Interpolation을 수행할 필요가 있다.

그림 6은 이러한 과정을 위한 폴리곤의 모습이다. 앞의 그림 3과 다른 점은 폴리곤 내의 픽셀을 $V_p$가 아닌 $V_pW_p$로 정의한 것이다. 즉, Interpolation한 값을

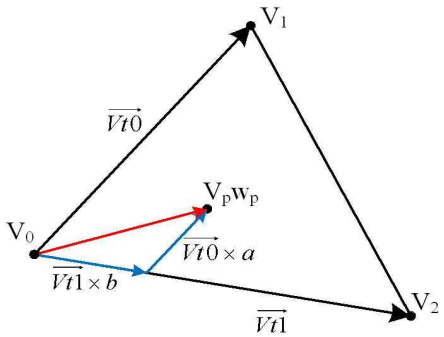



그림 6. Perspective-Correction을 위한 픽셀의 Vector 표현

Fig. 6. Expression of pixel for Perspective-Correction.

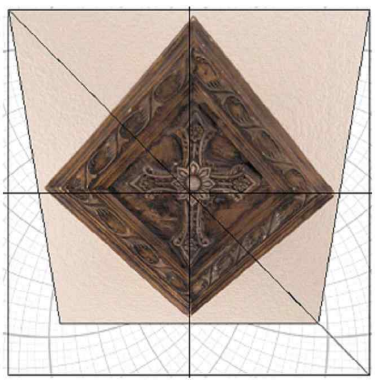


그림 7. Perspective-Correction에 기반한 interpolation 결과
Fig. 7. The result of Interpolation based on a Perspective-Correction.

'Devide by W' 과정 이전 값으로 정의하여 최종적으로 $V_p$를 구하게 되면 $W_p$로 나눠준 올바른 Interpolation 결과를 얻게 되는 것이다.

이를 위해서는 Interpolation된 $W_p$의 값을 구하는 과정도 필요하게 된다. 이렇게 $W_p$을 고려한 수식이 수식 5에 나타나 있다.

$$V_p w_p = V_0 + (V_1 - V_0)a + (V_2 - V_0)b$$

$$V_p = \frac{V_0 + (V_1 - V_0)a + (V_2 - V_0)b}{w_p} \quad (5)$$

$$w_p = w_0 + (w_1 - w_0)a + (w_2 - w_0)b$$

위의 식을 연립하여 정리하면 a와 b를 구하는 식은 수식 6과 같이 나타낼 수 있다.

$$a = \frac{y_0\alpha_2 - x_0\beta_2 + x_p(\beta_2 w_0 - y_0\gamma_2) + y_p(x_0\gamma_2 - \alpha_2 w_0)}{\alpha_1\beta_2 - \alpha_2\beta_1 + x_p(\beta_1\gamma_2 - \beta_2\gamma_1) + y_p(\alpha_2\gamma_1 - \alpha_1\gamma_2)}$$

$$b = \frac{y_0\alpha_1 - x_0\beta_1 + x_p(\beta_1 w_0 - y_0\gamma_1) + y_p(x_0\gamma_1 - \alpha_1 w_0)}{\alpha_1\beta_2 - \alpha_2\beta_1 + x_p(\beta_1\gamma_2 - \beta_2\gamma_1) + y_p(\alpha_2\gamma_1 - \alpha_1\gamma_2)}$$

$$\begin{array}{ll} \alpha 1 = x1 - x0; & \alpha 2 = x2 - x1; \\ \beta 1 = y1 - y0; & \beta 2 = y2 - y1; \\ \gamma 1 = w1 - w0; & \gamma 2 = w2 - w1; \end{array} \quad (6)$$

이렇게 Perspective-Correction을 통해 연산된 a와 b를 이용하여 Interpolation을 진행했을 때 그림 5과 같은 오류를 해결 할 수 있다. 이러한 결과가 그림 7에 나타나고 있다.

## V. 기능 검증 및 결과

본 논문에서 제안하는 Vector 기반의 Rasterization 알고리즘은 Multi-Thread, Multi-Core기반의 프로세서에서 여러 개의 처리 프로세스가 동시에 많은 양의 픽셀을 처리할 수 있도록 각 픽셀 처리 과정의 독립성을 최대한으로 보장한 알고리즘이며 이를 수행하기 위한 연산량 역시 최소한으로 줄인 알고리즘이다. 이를 확인하기 위하여 제안하는 Vector 기반의 Rasterization 알고리즘과 기존에 가장 일반적으로 사용되고 있는 Scan-Line 알고리즘을 C모델로 구현하여 그 수행과정에 필요한 연산량을 카운트하여 비교한 결과가 그림 8과 그림 9에 나타나 있다.

기존에 가장 일반적인 Rasterization 알고리즘인 Scan-Line 알고리즘의 경우 각 픽셀들을 순차적으로 처리하며 단위 변화 값을 더해줌으로써 Interpolation하는 방법이다. 이는 현재 픽셀을 Interpolation하는 과정에 이전 픽셀의 Interpolation한 결과 값을 이용함으로써 두 픽셀 간에 중복되는 연산을 줄일 수 있다. 그러나 이것은 각 픽셀이 긴밀한 연계성을 가지고 있다. 즉, 이를 Multi-Thread나 Multi-Core환경에서 병렬로 처리하기 위해서는 각각의 독립적인 실행을 보장하기 위하여 각 픽셀간의 연계성을 최소한으로 줄일 필요가 있다.

Scan-Line 알고리즘을 제안하는 Vector 기반 Rasterization알고리즘과 같이 하나의 폴리곤 내에서 어떤 픽셀도 독립적으로 처리되기 위해서는 픽셀간의 연계성을 없애야 한다. 이러한 연계성은 픽셀 간 Interpolation 과정에서 현재 Interpolation하는 ScanLine과 폴리곤의 두 Edge와의 교점의 속성 정보 값을 구하고 이 사이에서 각 픽셀의 증가에 따른 각 속성 정보들의 변화량을 계산하여 이전 픽셀에 더해주는 과정 중, 중복되는 연산을 줄이기 위해 하나의 ScanLine에서 두 교점의 속성 정보를 한 번만 구하여 ScanLine내의 픽셀이 공유하여 사용하는 과정과 현재 픽셀의 속성정보를 계산 할 때 이전 픽셀의 속성 정보 값에 증가량을 더해주는 과정에서 발생되게 된다. 그러므로 픽셀의 독립적인 처리를 위하여 이러한 연계성을 줄이게 되면, 매 픽셀마다 그 픽셀이 속한 ScanLine과 두 Edge와의 교점의 속성정보를 구하는 과정과 이 두 교점을 이용하여 처리하는 픽셀의 속성정보를 Interpolation 하는 과정이 매번 반복되어야 한다.

앞의 그림 2에서 $V_p$의 속성정보 값을 계산하는데

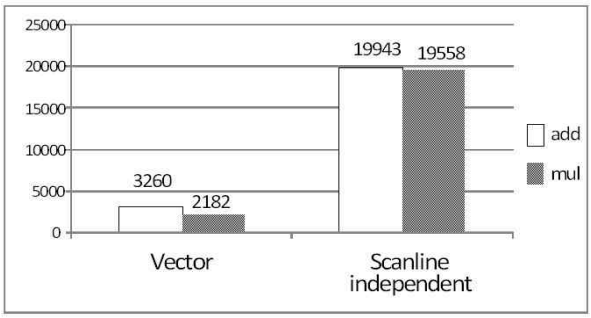
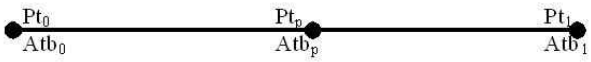



그림 8. add, mul연산량 비교
Fig. 8. Comparison of the number of add and mul operation.

V0와 V1의 속성정보 값을 Interpolation하여 Vscan0의 속성정보 값을, V1과 V2의 속성정보 값을 Interpolation하여 Vscan1의 속성정보 값을 계산하고, Vscan0와 Vscan1의 속성정보 값을 Interpolation하여 Vp의 속성정보 값을 계산하게 된다.



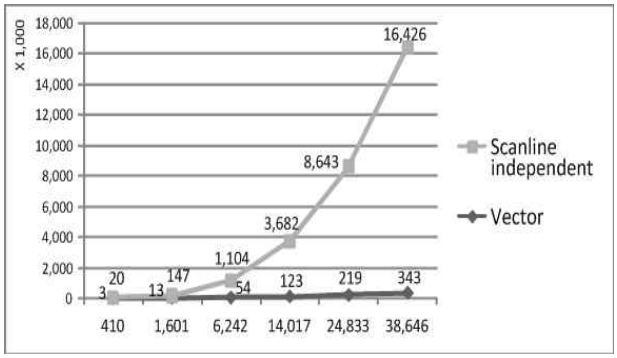

$$Atb_p = \frac{Atb_1 - Atb_0}{Pt_1 - Pt_0}(Pt_p - Pt_0) \quad (7)$$

두 점 사이의 다른 속성정보 값을 Interpolation하는 식이 식 7과 같을 때 현재 처리하는 픽셀의 하나의 속성정보 값을 Interpolation하기 위해서 식 7의 Interpolation과정을 총 3번을 실행해야만 한다. 이는 곧 많은 연산량의 증가를 초래하게 된다. 또한 Linear한 Interpolation이 아닌 Perspective Correction을 적용한 Scan-Line 알고리즘의 경우 제안하는 Vector 기반의 Rasterization알고리즘과 같이 기존 Interpolation 알고리즘에 깊이 값을 고려하여 Interpolation을 해야 하기 때문에 그 연산량이 더욱 증가하게 된다.

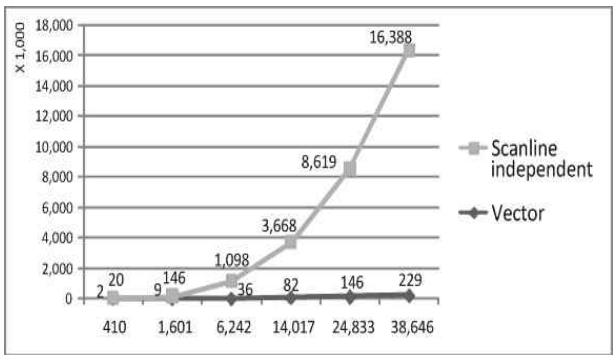
본 결과에 사용된 Scan-Line알고리즘의 연산량 수치는 이러한 알고리즘을 바탕으로 구현된 C프로그램에서 각 해당 연산의 연산량을 카운트 하여 그 결과를 비교한 수치이다.

그림 8의 그래프는 410개의 픽셀로 이루어진 폴리곤을 제안하는 Vector기반 Rasterization 알고리즘과 각 픽셀간의 연계성을 최소한으로 줄인 Scan-Line 알고리즘을 이용하여 Rasterization을 수행 할 때의 각각의 연산량을 나타내고 있다.

그림 8에서 나타난 바와 같이 Scan-Line알고리즘을 Multi-Core, Multi-Thread기반의 그래픽 프로세서에서




(a) 처리 픽셀 수에 따른 add연산량의 비교
(a) Comparison of add operation with the varying number of processing pixel




(b) 처리 픽셀 수에 따른 mul 연산량의 비교
(a) Comparison of mul operation with the varying number of processing pixel

그림 9. 처리 픽셀수에 따른 Add, Mul 연산량 비교
Fig. 9. Comparison of add and mul operation with the varying number of processing pixel.

폴리곤 내의 픽셀을 병렬로 처리하기 위하여 각 픽셀의 연계성을 최소한으로 줄일 경우 하나의 픽셀을 처리하는데 필요한 연산이 Vector 기반의 Rasterization알고리즘보다 많이 필요하여 약 6배의 연산량의 차이를 보이고 있다.

그림 9는 처리하는 픽셀의 수를 각각 410개, 1601개, 6242개, 14017개, 24833개, 38646개로 달리하며 두 알고리즘간의 연산량을 비교한 결과이다. (a)는 이 중에서 add연산의 연산량을, (b)는 mul연산의 연산량을 비교한 결과이다. 이 그림에서 보이듯이 처리하는 픽셀의 수가 증가함에 따라 두 알고리즘간의 연산량이 더욱 급격하게 차이가 나는 것을 볼 수 있다.

이 역시 위에서 설명한 바와 같이 하나의 픽셀을 처리하는데 필요한 연산량이 Vector기반의 Rasterization 알고리즘보다 Scan-Line 알고리즘이 더욱 많이 필요하기 때문이다.

두 그래프를 분석해본 결과 같은 픽셀수를 가지는 동일한 폴리곤을 처리하는 데에 본 논문에서 제안하는 Vector기반의 Rasterization알고리즘은 Scan-Line알고리즘에 비하여 add연산은 약 2%, mul연산은 약 1.5%의 연산량만을 필요로 한다.

## Ⅳ. 결 론

본 논문에서 제안하는 Vector기반의 Rasterization 알고리즘은 Multi-Core, Multi-Thread 기반에서 각 픽셀을 병렬적으로 처리하기 위하여 픽셀에 독립성을 부여한 알고리즘이다. 이를 통하여 기존 Scan-Line 알고리즘을 이용하여 수행 시 각 픽셀에 독립성을 부여하였을 경우의 연산량에 비하여 2%정도의 연산량만을 가지면서도 각 픽셀에 독립성을 부여하여 병렬처리가 가능하도록 한다. 이로써 Rasterization을 위한 별도의 하드웨어 없이 Shader Core를 Vertex Shader와 Pixel Shader의 처리뿐만 아니라 Rasterization의 처리까지도 활용함에 따라 더 적은 면적으로 빠른 그래픽 파이프라인 처리가 가능한 프로세서의 개발을 기대할 수 있다.

## 참 고 문 헌


[1] Gummaraju, J.; Erez, M.; Coburn, J.; Rosenblum, M.; Dally, W.J.; Architectural Support for the Stream Execution Model on General-Purpose Processors, PACT 2007. 16th International Conference on 15-19 Sept. 2007 Page(s):3 - 12

[2] Moya, V.; Gonzalez, C.; Roca, J.; Fernandez, A.; Espasa, R.; Shader performance analysis on a modern GPU architecture, Microarchitecture, 2005. MICRO-38. Proceedings. 38th Annual IEEE/ACM International Symposium on 16-16 Nov. 2005 Page(s):10 pp. - 364

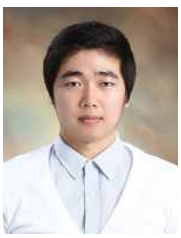
[3] Enhua Wu; Youquan Liu, Emerging technology about GPGPU, APCCAS 2008. IEEE Asia Pacific Conference on Volume , Issue , Nov. 30 2008-Dec. 3 2008 Page(s):618 - 622

[4] 하진석,정형기,김상연,이광엽,"Programmable Vertex Shader를 내장한 3차원 그래픽 지오메트리 가속기 설계," 대한전자공학회 논문지, 제 43권, SD편, 제9호, 9월 2006년


## 저 자 소 개


**이 주 석**(정회원)

1983년 서강대학교 전자공학과 졸업(공학학사)
1985년 고려대학교 대학원 전자공학과(공학석사)
1999년 고려대학교 대학원 전자공학과(공학박사)
1984년～1995년 LG전자 선임연구원
1997년～2000년 용인송담대학 전자과 조교수
2004년～2006년 엠텍비젼㈜ 연구소장
2007년～현재 충북테크노파크 차세대반도체 임베디드시스템기술개발지원센터 센터장
<주관심분야 : SoC 설계, 이미지 프로세싱, Embedded System>



**이 보 행**(학생회원)

2008년 서경대학교 컴퓨터공학과 졸업 (공학학사)
2008년～현재 서경대학교 대학원 컴퓨터공학과 공학석사과정
<주관심분야 : 임베디드 그래픽 프로세서, 3D Graphics System>

**김 우 영**(학생회원)

2008년 서경대학교 컴퓨터공학과 졸업(공학학사)
2008년～현재 서경대학교 대학원 컴퓨터공학과 공학석사과정
<주관심분야 : 임베디드 그래픽 프로세서, 3D Graphics System>

**이 광 엽**(정회원)-교신저자

1985년 서강대학교 전자공학과 졸업(공학학사)
1987년 연세대학교 대학원 전자공학과 졸업(공학석사)
1994년 연세대학교 대학원 전자공학과 졸업(공학박사)
1989년～1995년 현대전자 선임연구원
1995년～현재 서경대학교 컴퓨터공학부 부교수
<주관심분야 : 마이크로 프로세서, Embedded System, 3D Graphics System>