ISSN: 1226-7244 (Print)
ISSN: 2288-243X (Online)
논문번호 14-03-16
106

j.inst.Korean.electr.electron.eng.Vol.18,No.3,403~408,September 2014
http://dx.doi.org/10.7471/ikeee.2014.18.3.403

Pixel Block 단위 Varying Interpolator를 적용한 타일기반 Rasterizer 설계

A Design of a Tile-Based Rasterizer Using Varying Interpolator by Pixel Block Unit

김 치 용*

Chi-Yong Kim*

Abstract

In this paper, we propose a rasterizer architecture using varying interpolator which process several pixels at a time. Proposed rasterizer is able to handle 16 pixel at a time and output the color of up to 64. It can reduce the redundancy of calculation by configuring a matrix transformation and matrix calculation for rasterization, and it can enhance the speed of rasterizer by increasing the reusability. As a result, proposed rasterizer has improve 11% in color interpolation, 17% in the processing speed of the rasterizer by comparing with conventional research.

요 약

본 논문은 Varying Interpolator를 개선하여 다수의 Pixel을 한 번에 처리할 수 있는 Rasterizer 구조를 제안한다. 설계한 Rasterizer의 Varying Interpolator는 한 번에 16 Pixel을 처리 할 수 있으며 최대 64개의 색상을 출력으로 가진다. 또한 Rasterizer의 연산을 행렬연산 및 행렬변환으로 구성하여 연산의 중복성을 줄이고 재사용성을 높여 Rasterizer의 처리 속도를 높였다. 제안하는 구조의 Rasterizer 는 기존의 연구와 비교하여 색상 보간은 11%, Rasterizer 전체 처리 속도는 17% 향상된 성능을 보였다.

Key words : rasterizer, tile-based randering, matrix transformation, varying interpolator, pixel block

Ⅰ. 서론

3D 그래픽이 점점 더 빠르게 산업 전 분야로 다양하게 적용되어지고 있고 수많은 제품들이 개발되고 있다. 다양한 분야로의 적용으로 인해 사용자의 3D 그래픽의 수요가 많아지고 그에 따라 고사양의 3D 그래픽에 대한 요구사항 또한 늘어나고 있다.

이러한 기술의 수요에 따라 방대한 연산량을 가지는 3D 그래픽을 더욱 빨리 처리할 수 있는 프로세서, 모바일기기와 같은 제한된 자원을 사용하여 동작하는 3D 그래픽 파이프라인 등의 연구가 활발하게 진행되고 있다.

Rasterizer는 화면을 구성하는 Pixel단위의 색상을 처리하는 프로세서로 3D 그래픽 파이프라인에서 중요한 역할을 한다. 타일 기반 Rasterizer는 수많은 Rasterizer 중 병렬처리에 적합하여 멀티코어 GPU에

* Dept. of Computer Science, Seokyeong University
kcy@skuniv.ac.kr 02-940-7759

※ Acknowledgment
This Research was supported by Seokyeong University in 2013.
Manuscript received Sep. 3, 2014; revised Sep. 22, 2014 ; accepted Sep. 23. 2014

적용이 용이하기 때문에 최근 많은 연구가 진행되어 지고 있다.

타일 기반 Rasterizer 설계의 주안점은 크게 두가지로 나뉘는데 하나는 하위계층 호출 빈도와 하위계층 처리 속도를 포함하는 타일링, 다른 하나는 한 번에 색상을 출력 할 수 있는 Pixel의 수와 Pixel당 색상 처리 속도를 포함하는 색상 보간이다.

본 논문에서는 Rasterizer 연산속도의 향상을 위해 행렬연산과 행렬변환의 사용으로 Data의 재사용성을 높여 중복 연산을 줄이고 다수의 Pixel을 한 번에 처리하여 색상 보간 시간을 줄일 수 있는 Varying Interpolator를 적용한 타일 기반 Rasterizer에 대해 연구하였다.

Ⅱ. 본론

1. 전처리

가. 행렬 표현

수많은 정점의 연산 및 색상값의 연산은 방대한 데이터와 복잡한 연산을 사용하기 때문에 Rasterizer 수행시간의 대부분을 차지한다. 이러한 연산들은 수학적 방법인 행렬의 표현을 이용하여 처리 속도를 늘릴 수 있다.

Rasterizer에서 모델은 Polygon 단위로 처리되고 하나의 Polygon은 그림 1과 같이 3개의 정점으로 표현하거나 3개의 직선의 방정식으로 표현한다.

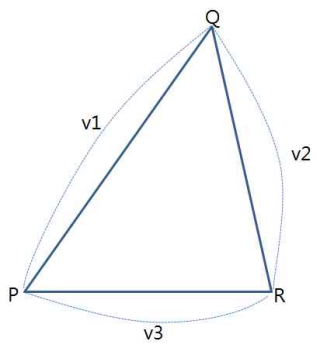

Fig. 1. Polygon Model
그림 1. 폴리곤 모델

$$\begin{aligned} v_1 &: A_1x + B_1x + C_1 = 0 \\ v_2 &: A_2x + B_2x + C_2 = 0 \\ v_3 &: A_3x + B_3x + C_3 = 0 \end{aligned} \quad (1)$$

수식 (1)은 Polygon을 3개의 직선의 방정식으로 표현한 결과이다. 선행 연구[1]에서는 Pixel의 색상을 계산하는 과정에서 현재 위치를 3개의 정점으로부터의 거리를 계산해야 한다. 모든 Pixel에서 위의 방정식을 호출하고 3개의 정점으로부터의 Data가 필요하기 때문에 동일한 연산을 중복해서 사용한다. 이러한 연산의 중복성으로 인해 Rasterizer의 성능이 떨어지게되는 단점이 있다.

$$Polygon = \begin{pmatrix} A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \\ A_3 & B_3 & C_3 \end{pmatrix} \quad (2)$$

본 연구에서는 수식 (2)와 같이 Polygon의 표현을 3*3행렬로 변환하여 전체 모델을 이루는 Polygon들의 표현을 쉽게 하고 미리 변환해둔 3*3행렬을 사용하여 연산을 수행한다. 모든 Pixel은 한번의 호출로 색상을 계산할 수 있기 때문에 선행 연구[1]에서 발생한 연산의 중복성을 막고 Rasterizer의 성능을 향상 시켰다.

나. 행렬 변환

행렬의 표현은 연산을 빠르게 하는 것 외에도 여러 가지의 형태로 변환이 쉽다는 장점을 지닌다.

본 논문에서는 수식 (2)와 같이 표현한 Polygon을 다양한 형태로 변환하여 Pixel Block의 내외부 판정 및 타일링에서의 Pixel Masking, Varying Interpolator에서의 색상 보간 연산에 사용한다.

(1) 행렬식(Determinant)

행렬의 다양한 형태로의 변환을 위해서는 먼저 행렬식이라 부르는 행렬의 대표 값을 구해야 한다. 수식 (2)에서 사용할 3*3 행렬의 행렬식 값은 다음 수식 (3)과 같이 구할 수 있다.

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix} \quad (3)$$

$$(\text{단}, \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc\ )$$

(2) 내외부 판정 행렬

타일 기반 렌더링에서 호출된 하위계층에 렌더링되는 픽셀이 존재하지 않을 경우 전체 3D그래픽파이프라인의 성능의 저하를 가져올 수 있다. 본 논문에서는 이로 인한 성능의 저하를 막기 위한 내외부 판정 알고리즘을 사용한다. 내외부 판정 알고리즘에 적용하기 위한 3*3행렬은 수식 (2)를 변환하여 구한다. 변환 과정은 아래와 같다.

먼저 아래의 수식 (4)와 같이 수식 (2)의 전치행렬을 구한다.

$$\begin{pmatrix} A_1 B_1 C_1 \\ A_2 B_2 C_2 \\ A_3 B_3 C_3 \end{pmatrix} => \begin{pmatrix} A_1 A_2 A_3 \\ B_1 B_2 B_3 \\ C_1 C_2 C_3 \end{pmatrix} \quad (4)$$

이후 수식 (3)을 통해 구한 행렬식의 부호를 수식 (4)에 곱하여 변환을 완료한다.

$$\begin{pmatrix} A_1 A_2 A_3 \\ B_1 B_2 B_3 \\ C_1 C_2 C_3 \end{pmatrix} * Determinant's\ Sign \quad (5)$$

변환된 수식 (5)의 3*3행렬을 이용하여 Pixel Block의 내외부를 판정할 수 있다.

(3) 상대 좌표 행렬

전처리 이후, 색상 보간 단계에서 처리할 해당 Pixel의 위치를 세 정점의 위치로부터 상대 좌표로 연산해야 한다. 각 정점으로부터의 상대 좌표를 연산하기 위한 3*3행렬을 수식 (2)의 행렬을 변환하여 사용한다.

먼저 행렬식의 값을 변환하고 그 값을 세 정점의 w 값에 곱한다.

$$\begin{aligned} rcpdet &= 1.0 / \det erminant \\ rcpw_1 &= w_{1*} rcpdet \\ rcpw_2 &= w_{2*} rcpdet \\ rcpw_3 &= w_{3*} rcpdet \end{aligned} \quad (6)$$

이후 수식 (6)에서 얻어진 값을 수식 (2)의 3*3행렬에 곱하여 행렬을 변환한다.

$$\begin{pmatrix} RC_1 RC_2 RC_3 \\ RC_4 RC_5 RC_6 \\ RC_7 RC_8 RC_9 \end{pmatrix} = \begin{pmatrix} A_1 B_1 C_1 \\ A_2 B_2 C_2 \\ A_3 B_3 C_3 \end{pmatrix} * \begin{pmatrix} rcpw_1 rcpw_1 rcpw_1 \\ rcpw_2 rcpw_2 rcpw_2 \\ rcpw_3 rcpw_3 rcpw_3 \end{pmatrix} \quad (7)$$

(4) 깊이 행렬

Varying Interpolator에서 처리되는 각 Pixel들은 색상 보간 수행 전 깊이 테스트를 수행하기 때문에 전처리과정에서 Pixel위치에서의 깊이 연산을 위한 1*3행렬을 수식 (7)의 3*3행렬을 변환하여 구한다.

수식 (7)의 행렬에 각 정점의 z값을 곱하여 3*3행렬을 1*3행렬로 변환한다.

$$\begin{pmatrix} D_1 \\ D_2 \\ D_3 \end{pmatrix} = \begin{pmatrix} RC_1 RC_2 RC_3 \\ RC_4 RC_5 RC_6 \\ RC_7 RC_8 RC_9 \end{pmatrix} * \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} \quad (8)$$

2. 색상보간

본 논문에서는 화면의 Polygon 들을 병렬로 처리하기 위하여 선행 연구[8]에서 사용된 병렬처리 보간 기법을 적용한다. 선행 연구[8]에서는 병렬 처리 보간 기법의 보간 알고리즘에 무게중심 좌표를 이용한 보간법을 사용하였으나 본 논문에서는 Block, Pixel 번호를 이용한 전역 좌표를 이용한 보간법을 사용한다.[9]

가. 내외부 판정

(1) Block 내외부 판정

수식 (5)의 3*3행렬을 이용하여 색상 보간을 수행할 Polygon이 Pixel Block 내부에 있는 경우에만 Varying Interpolator가 동작을 수행할 수 있도록 하여 불필요한 색상 보간 연산을 막는다.

(2) 마스크 테스트

내외부 판정을 통과한 Block에 대해서는 타일링 연산(해당 Polygon을 Pixel mask에 Mapping)을 통하여 얻어진 Pixel Mask를 이용한 마스크 테스트를 거치게 되고 이를 통과한 Pixel에 대해서만 색상 보간 연산을 수행하여 연산 속도 저하를 방지한다.

(3) 깊이 테스트

Block 내외부 판정 및 마스크 테스트를 통과한 Pixel에 한하여 동일한 위치의 먼저 수행되어진 Pixel의 깊이 값과 비교한다. 화면에 가까운 Pixel 즉, 깊이 값이 더 큰 Pixel의 색상을 해당 위치의 출력으로 사용한다.

나. 색상보간

(1) 전역좌표 계산

현재 Pixel의 색상을 각 정점의 비율에 맞게 구하기 위해서는 수식 (7)의 3*3행렬과 전역좌표를 이용해 정확한 Pixel의 상대 좌표를 계산해야 한다.

수식 (7)의 3*3행렬만을 사용하여 색상 보간을 진행할 경우 모든 Block이 같은 패턴을 가진 색상으로 칠해지는 오류가 발생한다.

Varying Interpolator는 호출된 Block의 번호와 Block 내부에서 수행중인 Pixel의 번호를 조합하여 전역좌표를 계산할 수 있다.

(2) 상대 좌표 연산

상대좌표를 연산하기 위해 수식 (7) 의 3*3행렬을 전치행렬로 변환하고 그 결과인 수식 (9)에 전역좌표를 곱한다.

$$\begin{pmatrix} RC_1 & RC_2 & RC_3 \\ RC_4 & RC_5 & RC_6 \\ RC_7 & RC_6 & RC_9 \end{pmatrix} => \begin{pmatrix} RC_1 & RC_4 & RC_7 \\ RC_2 & RC_5 & RC_8 \\ RC_3 & RC_6 & RC_9 \end{pmatrix} \quad (9)$$

$$\begin{matrix} GA \\ GB \\ GC \end{matrix} = \begin{pmatrix} RC_1 & RC_4 & RC_7 \\ RC_2 & RC_5 & RC_8 \\ RC_3 & RC_6 & RC_9 \end{pmatrix} * \begin{pmatrix} global_x \\ global_y \\ 1 \end{pmatrix} \quad (10)$$

(3) 보간 연산

Varying Interpolator의 최종 출력결과는 32bit 색상 값이다.

Rasterizer의 입력은 Varying(RGBA)으로 색상의 보간과 정규화가 필요하다.

먼저 수식 (10)의 상대좌표와 Varying(RGBA)을 곱하여 보간된 Varying(RGBA)을 구한다.

$$(IR \;\; IG \;\; IB \;\; IA) =$$

$$(GA \;\; GB \;\; GC) * \begin{pmatrix} R_1 & G_1 & B_1 & A_1 \\ R_2 & G_2 & B_2 & A_2 \\ R_3 & G_3 & B_3 & A_3 \end{pmatrix} \quad (11)$$

이후 Varying Interpolator는 수식 (12)과 같이 0~255 범위로 정규화된 Varying(RGBA)을 연산한다.

$$\begin{matrix} R \\ G \\ B \\ A \end{matrix} = \begin{matrix} IR*255 \\ IG*255 \\ IB*255 \\ IA*255 \end{matrix} \quad (12)$$

정규화된 각 8bit의 Varying(RGBA)을 연결하여 수식 (13)와 같이 Varying Interpolator의 Output을 생성한다.

$$Output = A \ll 24 | R \ll 16 | G \ll 8 | B \quad (13)$$

(4) Block 단위 보간

한번의 Varying Interpolator 호출로 처리할 수 있는 Pixel의 수는 Rasterizer의 성능에 크게 작용하는 요소이다.

선행 연구[1]에서 색상의 보간은 최하위계층에서 이루어지고 이때 Pixel의 유효값을 판정 후 해당 Pixel이 유효할 경우 즉시 색상 보간을 수행한다. 즉 선행 연구[1]의 Varying Interpolator는 1회 호출로 하나의 Pixel을 처리하는 하는 구조로 설계되어있다.

본 논문의 Varying Interpolator는 16 Pixel 로 이루어진 Pixel Block 단위로 동작한다. 여기서 각 Pixel은 Anti-aliasing을 위한 4개의 SubPixel로 구성되어 있기 때문에 Varying Interpolator는 한 번에 최대 64개의 Color 값을 연산할 수 있다.

선행 연구[1]에서 최하위계층인 8*8 Tile에서 비교하였을 때 본 논문의 Varying Interpolator는 최대 4번의 호출로, 선행 연구[1]의 Varying Interpolator는 최대 64번의 호출로 해당 Tile의 Pixel들을 처리할 수 있다.

또한 논 논문에서는 타일링 연산을 통해 얻어진 64bit Pixel Mask를 통해 마스크 테스트를 진행하고 모든 Mask가 Set이 되었을 경우에 64개의 Color를 출력으로 내보내고 Mask에 Reset으로 Mapping되어진 Pixel은 색상 보간을 수행하지 않아 불필요한 자원의 낭비와 처리 속도 저하를 막는다.

3. 실험 및 결과

전체 3D 그래픽 파이프라인의 과정은 각 프로세서가 출력을 메모리에 저장하고 또 다른 프로세서가 필요한 Data를 메모리로부터 읽어 사용한다. Rasterizer는 Varying Data, Vertex Data, Frame Size를 Memory로부터 읽어 내부의 동작으로 각 Pixel의 색상을 계산하고 해당 출력을 Memory에 저장한다. 본 연구에서 설계한 Rasterizer는 TSMC 180nm MPW 공정에서 63만 게이트로 설계되었으며 Xilinx사의

FPGA 플랫폼인 Virtex 6 ML605에 구현하여 실험하였다. 설계한 Rasterizer는 FPGA에서 100MHz로 동작하며 구조는 아래 그림 2. 와 같다.

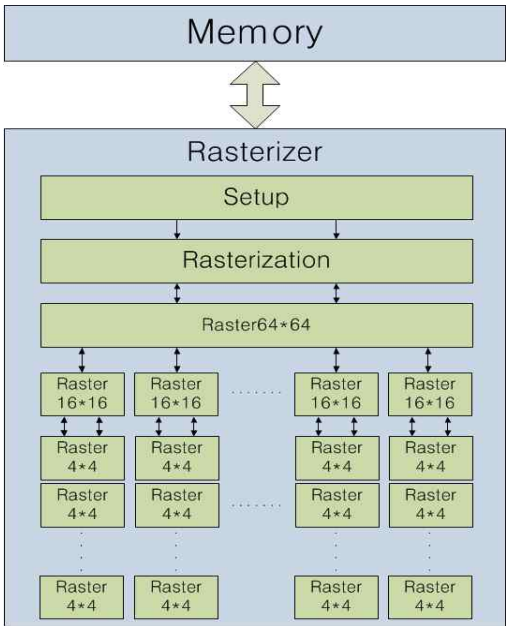

Fig. 2. Rasterizer Architecture
그림 2. Rasterizer 구조

그림 2.에서 Raster n*n 모듈은 각각 16개의 하위 모듈로 구성되어진다.

선행 연구[1]와 같이 VGA 사이즈 화면 내부의 2256개의 폴리곤을 가지는 Teapot Model의 후면 제거 후 남은 975개의 폴리곤 데이터를 이용해 실험 결과를 측정하였다.

각 알고리즘의 전체 수행시간을 측정한 결과 제안하는 알고리즘을 적용한 Rasterizer에서는 15.943ms로 선행연구[1]의 계층구조 Rasterizer의 19.208ms보다 17%, MS Tiling의 29.614ms 보다 47% 향상된 성능을 보였다.

또한 타일링 연산을 제외하고 색상 보간 처리 시간을 비교했을 경우 제안하는 알고리즘은 6.839ms로 선행 연구[1]의 7,683ms 보다 11% 향상된 성능을 보였다.

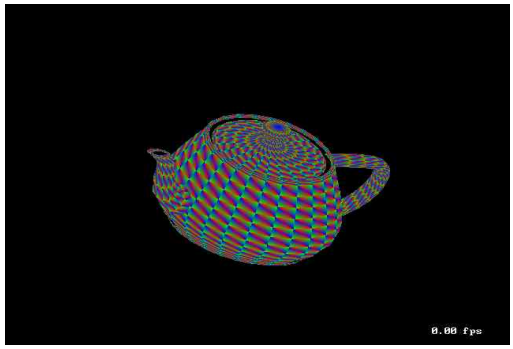

Fig. 3. Teapot Model
그림 3. Teapot Model

Table 1. Comparison operation speed(Teapot Model)
표 1. 연산속도 비교(Teapot Model)

Rasterizer 구조	연산속도
Proposed Rasterizer	15.943ms
Hierarchical Tiling[1]	19.208ms
MS Tiling[1]	29.614ms

Table 2. Comparison Varying Interporlation operation speed(Teapot Model)
표 2. 색상 보간 연산속도 비교(Teapot Model)

Rasterizer 구조	연산속도
Proposed Rasterizer	6.839ms
Hierarchical Tiling[1]	7.683ms

III 결론

본 논문은 Varying Interpolator에서 Pixel Block 단위를 한 번에 처리하여 고속의 색상 보간이 가능한 Rasterizer 구조를 제안하였다.

설계한 Rasterizer는 한 번에 16 Pixel에 대한 색상 보간 연산이 가능하고 최대 64개의 Color값을 출력할 수 있다.

또한 Rasterizer의 연산을 행렬연산으로 구현하여 전처리과정에서 색상 보간 및 타일링을 위해 구성한 여러 Data의 재사용성을 높이고 Rasterizer의 처리 속도를 향상 시켰다.

본 논문에서 제안한 Rasterizer 구조는 선행 연구[1]와의 비교에서 전체 처리 속도는 17%, 색상 보간 처리 속도는 11% 향상된 것으로 나타났다.

더 크게, 더욱 빠르게 성장하고 있는 3D 그래픽 시

장의 규모를 따라가기 위해서는 본 논문에서 연구한 Rasterizer뿐 아니라 3D 그래픽 파이프라인 전 과정에 대하여 공학, 이학의 분야를 넘어 다양한 시각에서 시장을 분석하고 그에 맞춰 다양한 연구가 진행될 필요가 있다.

References

[1] Junseo Kim, "A Design of Tile based rendering for a Multi-Core GPU", Master's thesis, SeoKyeong University. Seoul, February 2012.

[2] Dong-Young Yeo, "A Design of a 3D Graphics pipeline based on Multi-core Processor", Master's thesis, SeoKyeong University. Seoul, February 2011.

[3] Imagination Technologies. "3D Graphical Processing(Tile Based Rendering the Future of 3D)," white paper. imagination Tech. Corp, 2000

[4] Macr Olano. Trey Greem. "Triangle scan conversion using 2D Homogeneous coordinates," Proceedings of the ACM SIGGRAPH/EUROGRAPHI-CS workshop on Graphics hardware. pp89-95. Los Angeles. California. United States, August 1997.

[5] Chinh-Chieh Hsiao and Chung-ping Chung and Hui-Chin Yang, "A Herarchical Primitive Lists Structure for Tile-Based Rendering" Computational Science and Engineering, Vol2. pp. 408 Vancouver, BC. Aug 2009.

[6] James Blinn, "Jim Blinn's Corner: Calculating Screen Coverage", IEEE Computer Graphics & Applications, IEEE Computer Society, Los Alamitos, CA, May 1996

[7] Imagination Technologies. "3D Graphical Processing(Tile Based Rendering the Future of 3D)," white paper. imagination Tech. Corp., 2000.

[8] Jangseo Ku, "Design of a Rasterizer based on Parallel Processing Interpolation Algorithm for a Mobile GPU", Master's thesis, SeoKyeong University. Seoul, February 2013.

[9] Kwang-Yeob Lee, Chi-Yong Kim, "Implementation of Parallel Processing Interpolation Algorithm for Multicore GPU", Journal of IKEEE. Vol.16, No.4, 304~309, December 2012.

BIOGRAPHY

Kim Chi Yong (Member)

1981. 2. Kyungpook National University, Dept. of Statics(BS)
1986. 2. Seoul National University, Dept. of Computing Science & Statistics(MS)
1993. 2. Seoul National University, Dept. of Computing Science & Statistics(Ph. D)
1995.3~ Seokyeong Univeristy, Dept. of Computer Science, Professor
<Research interests> Stochastic process, Mathematical analysis, Computer arithematic, Computer Graphics