공통 변 정보를 재 사용하는 3차원 그래픽 가속기의 삼각형 셋업 부의 설계

최문희, 박우찬, 김신덕
연세대학교 컴퓨터과학과
E-mail: mhchoi@kurene.yonsei.ac.kr

Design of the Triangle Setup Stage Reusing the Values of Shared Edge in 3D Graphics Accelerator

Moon-Hee Choi, Woo-Chan Park, and Shin-Dug Kim
Dept. of Computer Science, Yonsei University

요 약

최근 3 차원 그래픽스 분야에서 실감 영상 지원 요구에 따라 객체를 이루는 데이터의 수가 기하급수적으로 증가하게 되었다. 이에 고성능의 3 차원 그래픽 가속기에 대한 도입 뿐만 아니라 가속기에서 처리될 데이터의 표현 및 여러 처리 방법들에 대한 연구도 요구 되어지고 있다. 본 논문에서는 삼각형 스트립 기법을 이용하여 3 차원 그래픽 데이터를 효과적으로 표현할 수 있고, 이 기법의 특징을 이용하여 전체 시스템의 계산량을 줄일 수 있는 구조를 제안하였다. 즉 제안하는 구조는 3 차원 그래픽 가속기의 뒷 단인 래스터라이저의 삼각형 셋업 부에 공통 변 버퍼를 두어 인접한 삼각형 들 간에 공유되는 변들의 정보를 재 사용하도록 하였다. 이 구조는 공통 변 버퍼를 사용하지 않는 기존의 구조와 비교 했을 경우 최대 31.8%의 수행 성능 향상을 보여준다.

1. 서론

최근 3차원 그래픽스 분야의 실시간 시뮬레이션(Real-time Simulation), 애니메이션(Animation), 그리고 가상 현실(Virtual Reality) 등의 멀티미디어 환경 구축에서 실감 영상(Realistic Image)의 제공은 필수 사항이 되고 있다. 정교하고 현실감 있는 3차원 영상을 표현하기 위해 한 객체를 이루는 3차원 데이터, 즉 점, 선, 삼각형 등의 수는 기하 급수적으로 증가하고 있고, 이를 처리 하기 위한 고성능의 3차원 그래픽 가속기(3D Graphics Accelerator)에 대한 요망은 더욱 증대되고 있다.

일반적인 3차원 데이터의 표현 방식은 개별 다각형(individual triangle)표현 방식이다. 이 표현 방식은 구현은 쉽지만, 한 객체를 이루는 여러 삼각형들 간의 데이터가 중복되는 문제점이 있다. 이는 3차원 그래픽 파이프라인(3D Graphisc Pipeline)의 각 단계에서 중복 연산과 이중으로 메모리 공간의 차지하는 등의 결과를 가져온다. 이런 낭비를 막기 위해 보다 적은 양의 데이터로도 개별 다각형 표현 방식과 같은 결과를 보이는 삼각형 표현 방식이 필요하다.

삼각형 스트립(Triangle strip)은 일련의 인접한 삼각형들의 집합으로 현재의 삼각형의 꼭지점 값과 바로 다음에 오는 삼각형의 것이 중복되는 경우 그 값을 공유하는 기법이다. 따라서 이 기법은 기존 의 방식 보다 적은 양으로도 많은 삼각형들을 표현할 수 있다[1][3].

삼각형의 수의 증가에 따라 래스터라이제이션(Rasterization)의 수행 과정 중 최 상단의 삼각형 셋업 부(Triangle Setup Stage) 연산 또한 증가하게 된다. 이는 래스터라이제이션의 전체 연산량에 영향을 끼치게 되므로 이를 줄이는 연구가 요구된다[1].

따라서 본 논문에서는 셋업 부의 연산의 양을 줄이는 구조를 제안하려 한다. 즉 삼각형 스트립 내 인접한 삼각형들간의 공통 변(Shared Edge)의 정보를 이용하는 구조로서 현 삼각형에서 구해진 공통 변의 기울기(Slop value)와 각 매개 변수들의 변화량들(Gradients)을 공통 변 버퍼(Common Edge Buffer)에 저장한 후, 바로 다음의 삼각형 연산 과정에서 이 값

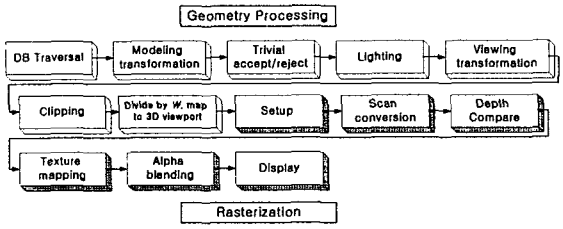
* 본 논문은 한국기술평가원(KISTEP)의 국가지정연구실 사업에 위해 수행된 연구 결과의 일부임.

들을 재 사용하도록 하려는 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 3차원 그래픽 렌더링 파이프라인(3D Graphics Rendering Pipeline)에 대해 전반적인 설명을 하고, 3장에서는 삼각형 스트립(Triangle strip) 형성 기법에 대해 살펴보겠다. 그리고 4장에서는 래스터라이저(Rasterizer)의 삼각형 셋업 부의 구조에 대해 제안하고, 마지막으로 5장에서는 셋업 부의 연산 분석 결과를 논한다.

2. 3차원 그래픽 렌더링 파이프라인

3차원 그래픽 가속기는 [그림 1]과 같이 그래픽 렌더링 파이프라인(Rendering Pipeline)으로 구성되어진다. 이 렌더링 파이프 라인은 크게 지오메트리 연산(Geometry Processing)와 래스터라이제이션(Rasterization) 두 부분은 나눌 수 있다[5].



[그림 1] 3차원 그래픽 렌더링 파이프라인

지오메트리 연산은 주로 3차원 좌표계의 물체를 시점의 따라 2차원 스크린 좌표계의 이미지로 변화시키는 기하학 연산들의 처리 과정이다. 즉 좌표와 법선벡터(Normal vector)를 시각 좌표계로 변환, 범프 매핑(Bump mapping)을 위한 물체 표면(object surface)의 굴곡화(Perturbation), 광원 처리(Lighting), 클립 좌표계로의 변환, 클리핑(Clipping), 윈도우 좌표계로의 투영(Projection)을 처리하는 연산을 수행한다. 또한 관련된 텍스쳐 좌표(Texture coordinate)는 삼차원 행렬에 의해 변환·윈도우 좌표계, 컬러와 연결되고 R,G,B, α,Z, 텍스쳐 좌표와 관련된 윈도우 좌표계 경사가 계산된다. 세 개의 꼭지점의 값 들, 즉 좌표 값(x, y, z)과 컬러 값(RGBα)으로 표현되는 삼각형은 지오메트리 연산의 출력으로 버스를 통해 래스터라이제이션으로 보내진다[4][5].

래스터라이제이션은 2차원 좌표계의 이미지에 실제 색깔 값 등을 구해 프레임 버퍼(Frame buffer)에 갱신하는 처리를 하는 과정이다. 이 부분에서는 디스플레이 할 최종 이미지를 프레임 버퍼에 갱신하기 전에 컬러와 텍스쳐, 그리고 투명도(transparency)에 대한 보간 연산(Interpolation processing)이 행해지므로 방대한 양의 연산을 필요로 한다. 래스터라이제이션은 셋업 단계와 주사 변환 단계(Scan Conversion Stage), 그리고 픽셀 연산 단계(Pixel Processing Stage)로 구성된다[2][5].

셋 업 단계는 주사 변환 단계에서 수행되는 삼각형의 변 보간 연산(Edge Processing)과 스팬 보간 연산(Span Processing)에서 사용되는 증가율(Increment)들을 계산하는 과정이다. 셋업 단계의 증가율을 구하는 과정에는 삼각형의 기울기 계산과 각 매개 변수들의 변화율 계산, 그리고 원근 텍스쳐를 위한 좌표 제법 등 여러 가지 변수들에 대한 나눗셈이 수행된다[2]. 깊이 증가율을 구하는 수식 (1)과 같이 셋 업 단계에서 구해지는 삼각형의 컬러, 깊이, 텍스쳐 좌표 증가율은 같은 역수 값에 의해 계산된다.

$$\frac{\partial z}{\partial x} = \frac{(z_2 - z_1)\cdot(y_3 - y_1) - (z_3 - z_1)\cdot(y_2 - y_1)}{(x_2 - x_1)\cdot(y_3 - y_1) - (x_3 - x_1)\cdot(y_2 - y_1)} \quad (1)$$

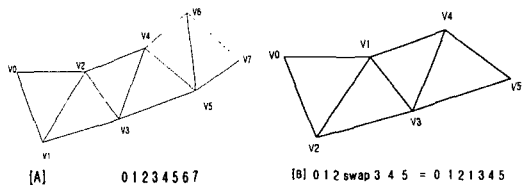
$$\frac{\partial z}{\partial y} = \frac{(z_3 - z_1)\cdot(x_2 - x_1) - (z_2 - z_1)\cdot(x_3 - x_1)}{(x_2 - x_1)\cdot(y_3 - y_1) - (x_3 - x_1)\cdot(y_2 - y_1)}$$

주사변환 단계의 변 보간 연산에서는 셋업 단계의 결과 값을 연산 수행 과정에 반복적으로 계속 적용시켜 준다. 그리고 세 꼭지점 중 한 꼭지점을 기준으로 하여 y축 방향으로 일정한 값만큼 증가시켜 주면서 삼각형 내의 모든 스팬들을 구해준다. 스팬 보간 연산에서는 변 보간 연산에서 형성된 모든 스팬들의 첫 번째 픽셀 값을 기준으로 x축 방향으로 일정한 값만큼 증가시켜 주면서 각 스팬 내의 모든 픽셀들의 값을 구하게 된다. 이 값들은 최종 디스플레이 되는 픽셀 값이 아닌 중간 값(Fragment value)으로 픽셀 연산 단계에서의 텍스쳐 매핑, 은면 처리를 위한 깊이 테스팅(Depth testing), 알파 블랜딩 (Alpha Blending) 등을 통해 화면에 최종적으로 디스플레이 하는 픽셀 값으로 된다[2][5].

3. 삼각형 스트립 형성 기법

삼각형 스트립은 3차원 데이터의 한 표현 방법 중 한 방법으로 [그림 2]과 같이 여러 다각형들을 일렬로 연결한 것이다.

삼각형 스트립에서의 기본 다각형이 사각형 이상의 다각형으로 된다면 꼬인 평면과 같은 경우가 발생할 수 있다. 따라서 기본 다각형으로 삼각형으로 제한하는 것이고, 이런 이유로 3차원 그래픽 분야에서도 일반적으로 기본 다각형을 삼각형으로 한다.



[그림 2] 삼각형 스트립

고성능의 3차원 그래픽 렌더링 파이프라인에서 그래픽 처리 수행 성능을 결정하는 가장 큰 요소는 삼각

형으로 분화된 데이터를 병목 현상 없이 각 단계별로 전송하는 비율이라고 할 수 있다.

3개의 꼭지점으로 표현되는 삼각형의 데이터를 간단하게 표현할 수 있다면, 데이터 전체량을 크게 줄일 수 있다. 이는 결국 데이터 전송 시간을 크게 줄일 수 있게 되고 데이터 저장 시 많은 이익을 얻을 수 있게 된다. 뿐만 아니라, 렌더링 내부에서의 연산에 소요되는 시간도 줄일 수 있게 된다. 이러한 점들이 가능할 수 있도록 하기 위해서 삼각형 스트립이 사용한다.

[그림 2]의 [A]에서 0~7까지의 꼭지점 정보 및 6개의 삼각형 연결 정보를 표현하기 위해서는, 원래 24개의 꼭지점 표현이 요구된다. 그런데, 트라이 앵글 스트립에서는 바로 뒤의 삼각형의 꼭지점이 직전의 삼각형을 표현하는데 그대로 사용되므로 새로이 추가되는 꼭지점만을 표현하면 된다. 즉, [A]의 경우 (0,1,2,3,4,5,6,7)과 같이 표현해 주게 된다. 따라서 n개의 심각형의 표현의 경우, 3n개의 꼭지점 정보를 표현 대신, n+2 (여기서는 2는 처음 삼각형 표현에 사용되는 초기 꼭지점의 개수이다.)만으로 표현 가능하게 된다

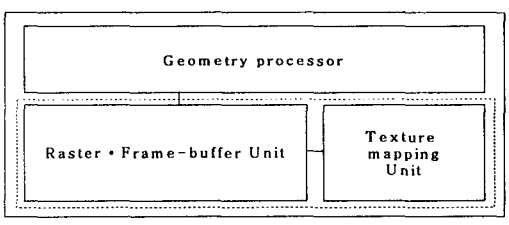
그렇지만 삼각형 스트립 생성 방법에 있어서 [A]와 같이 이상적인 상황보다는 [B]와 같은 여러 어려운 상황들이 발생한다. 재사용 되는 꼭지점은 직전의 데이터가 아닌 몇 단계 전의 것인 경우인 [B]와 같은 상황에서 0,1,2,3 다음에 4를 그대로 갖다 붙일 수 없다. 이를 해결하기 위해 [3]에서는 Swap이라는 명령어를 사용하였다. 최근 사용된 2개의 꼭지점 정보를 버퍼에 넣어 다음 꼭지점이 추가 될 때 마다 버퍼의 값과 합쳐서 삼각형을 형성하는 일반적인 삼각형 스트립 방식과 달리 Swap이라는 명령은 그 버퍼 내부에 있는 값 자체를 바로 직전에 버퍼에서 제거된 값으로 바꾸어 주는 작업을 의미 한다.

이와 같은 기법으로 삼각형 스트립의 모든 경우를 표현할 수 있게 된다.

4. 제안하는 3 차원 그래픽 가속기의 셋 업 부

4.1. 렌더링 시스템의 Top-View

본 논문에서의 제안하는 렌더링 시스템 모델은 [4]에서 제안하고 있는 시스템 모델과 같다.

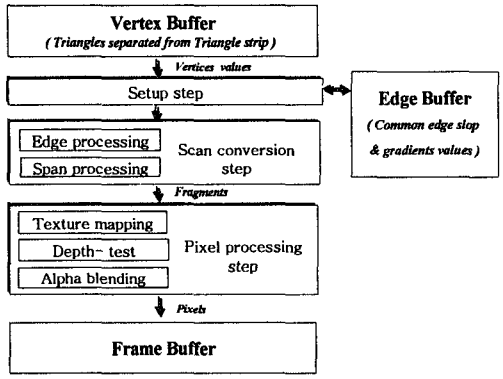


[그림 3] 시스템 모델

[그림 3]과 같이 기존의 3차원 그래픽 가속기에서와 달리 기하학 연산과정을 완전히 CPU와 분리하여 별도의 그래픽 칩 내에 삽입함으로써 CPU의 기하학 연산에 따른 부담을 감소 시킬 수 있다. 또한 래스터라이제이션 처리 유닛과 프레임버퍼 유닛을 단일 칩으로 구성하였다. 그리고 방대한 텍스쳐 데이터의 처리를 위해 텍스쳐 매핑을 위해 별도로 텍스쳐 매핑 유닛을 두었다.

4.2. 래스터라이저의 셋업 부 구성

본 논문에서 제안하는 래스터라이저(Rasterizer) 구조는 [그림 4]와 같다. 기존의 래스터라이저와 거의 같은 구조이지만, 공통 변 버퍼(Common Edge Buffer)를 첨가하여 공통 변 정보를 재 사용하도록 설계하였다.



[그림 4] 래스터라이저 블록 다이어 그램

4.2.1. 꼭지점 버퍼

일반적으로 주사변환 단계에서 처리 단위는 삼각형 스트립이 아닌 독립된 삼각형이다. 그래서 꼭지점 버퍼에서는 삼각형 스트립을 여러 개의 독립된 삼각형들로 분화시킨 후 그 스트립의 연결 순서에 따라 래스터라이저로 보낸다. 예를 들어 [그림 2]의 [A]는 인접한 6개의 삼각형들로, 즉 T0~ T5로 분화되어 순서대로 보내지게 되는 것이다.

4.2.2 셋업 부와 공통 변 버퍼

1장에서 언급했듯이 셋 업 부의 연산 양은 한 객체를 이루는 삼각형의 수에 비례하여 계속 증가하게 된다. 따라서 본 시스템에서는 삼각형 스트립 내 인접한 삼각형들간의 공통 변 정보를 최대한 재사용하기 위해 공통 변 버퍼를 추가하였다. 꼭지점 버퍼를 통해 보내어지는 3개의 꼭지점들은 모델링 단계에서부터 색인(indexing)되어 있다. 그리고 이 색인 값들을 가지고 공통 변들도 구분 짓는 것이다. 버퍼에 저장된 공통 변들의 값들은 한 번 사용되면 바로 다음에 계산되어 들어오는 공통 변 값으로 대치된다.

4.3. 병렬 래스터라이저 구조

본 절에서는 본 논문에서 제안한 단일 래스터라이저를 확장하여 병렬 래스터라이저 구조를 구성한다.

우선 꼭지점 버퍼에서 삼각형 스트립에서 분화된 여러 삼각형들과 처리 유닛인 래스터라이저의 개수를 고려하여 공통 변을 가지지 않는 삼각형들의 군으로 분류한다. 예를 들어 [그림 2]의 [A]에서 분화된 T0에서 T5까지의 6개 삼각형들을 세 개의 래스터라이저로 구성된 시스템에서 처리할 때, {T0,T2,T4}를 한 삼각형 군으로, {T1,T3,T5}를 다른 삼각형 군으로 분류하여 처리 한다. 이렇게 분류된 삼각형 군들은 순서대로 셋 업 부에서 처리되고, 공통 변 정보들을 버퍼에 저장된다. 병렬 래스터라이저 구조에서는 한 삼각형 연산 시 최대 두 개의 공통 변을 갖는 경우가 발생된다. 재 사용하는 정보는 단일 래스터라이저 보다 증가하게 되므로 래스터라이저의 수와 스트립을 이루는 삼각형의 수가 증가할수록 전체 수행 성능은 선형적으로 향상 됨을 알 수 있다.

5. 셋 업 부의 연산 분석 결과

본 장에서는 본 논문에서는 제안한 공통 변 버퍼를 사용한 구조와 기존의 래스터라이저 구조, 즉 공통 변 버퍼가 없는 구조간의 성능 평가 결과를 보여준다. 성능 평가를 위해 두 구조의 셋 업 부에서 연속으로 처리되는 두 개의 삼각형을 수행시킨 사이클을 비교하였다. 아래의 표 1은 그 수행 결과를 보여준다.

	mul	Add	div	total	수행시간 %
Unit latency	5	1	15		
No shared Edge	52	66	30		
사이클의 수	260	66	450	**776**	**100 %**
Shared Edge.	39	49	19		
사이클의 수	195	49	285	**529**	**68.2 %**

표 1 두 구조의 셋 업 부에서의 수행 사이클 비교

표 1에 의해서 공통 변 버퍼를 사용하는 래스터라이저에서의 수행 성능은 공통 변 버퍼를 사용하지 않는 기존의 래스터라이저의 성능에 비해 최대 31.8%의 향상되었다.

본 논문에서 제안한 구조는 삼각형 스트립 기법을 채택함으로써 기하 급수적으로 증가되고 있는 3차원 그래픽 데이터를 적은 양으로도 표현하였다. 그리고 공통 변 버퍼를 두어 삼각형 스트립 내의 인접한 삼각형들 간의 공통 변 정보들을 재 사용하게 함으로써 수행 성능의 향상을 꾀한다.

6. 참고 문헌

[1] Tzi-cker Chiueh, Tulika Mitra "TR-62: Mesh-oriented 3D Graphics Architecture," http://www.ecsl.cs.sunysb.edu/tech_reports.html, January 1999.

[2] Anders Kugler, "The Setup for Triangle Rasterization," 11th Eurographics Workshop on Computer Graphics Hardware, pp. 49-58, 1996.

[3] Francine Evans, Steven Skiena and Amitabh Varshney, "Optimizing Triangle Strips for Fast Rendering," The conference on Visualization '96, pp. 319-326, 1996.

[4] Chun-Ja Choi, Woo-Chan Park, Tack-Don Han, "High Performance Rendering System using a Rasterizer Merged Frame Buffer," 1999년도 한국정보과학회 가을 학술발표논문집(III) Vol. 26, No. 2, pp. 9-11. 1999.

[5] Foley, Van Dam, Feiner and Hughes, Computer Graphics principles and practice *2nd Ed*, Addson-Wesley, 1997.

[6] Tulika Mitra and Tzi-cker Chiueh,"Dynamic 3D graphics workload characterization and the architectural implications," The 32nd Annual ACM/IEEE international symposium on microarchitecture on MICRO-32, pp62-71, 1999.

[7] John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher j. Migdal, "InfiniteReality: A Real-Time Graphics System," Proceedings of the 24th annual conference on Computer graphics & interactive techniques, pp. 293-302, 1997.

[8] J. Eyles, S. Molnar, J. Poulton, T. Greer, A Lastra, N. England, and L. Westover, "PixelFlow: The Realization," Proceedings of SIGGRAPH '97, pp. 57-68, 1997.