래스터라이저-프레임버퍼 혼합 구조에서의 원근투영 텍스쳐 매핑의 설계

이승기, 박우찬, 한탁돈
연세대학교 컴퓨터과학과
E-mail: sklee@kurene.yonsei.ac.kr

The Design of the Perspective Texture Mapping in Rasterizer Merged Frame Buffer Technology

Seung-Gi Lee, Woo-Chan Park, Tack-Don Han
Dept. of Computer Science, Yonsei University

요약

최근 3차원 그래픽스 분야는 기존의 단순 이미지의 처리가 아닌 보다 나은 화질과 보다 많은 기법의 도입이 요구되어 지고 있다. 이에 본 논문에서는 가장 기본적인 실감영상의 표현 기법인 텍스쳐 매핑 기법에 대하여 논하였고, 3차원의 객체 공간에서 2차원의 스크린 공간으로의 변환으로 인해 생길 수 있는 문제점과 렌더링 알고리즘에 대해 분석하였으며, 이에 부합하는 렌더링 시스템을 설계, 분석하였다. 또한 본 시스템은 고성능 3차원 그래픽 처리를 위하여 채택되어지고 있는 프로세서-메모리 집적 방식을 이용, 래스터라이징 유닛과 프레임버퍼를 단일 칩으로 구성하여 렌더링과 텍스쳐 매핑 과정에서 발생할 수 있는 지연현상을 제거하였다.

1. 서론

3차원 컴퓨터 그래픽스 산업에서는 지난 10여 년 동안 상당한 진보가 있었다. 보다 고성능의 그래픽스 가속기의 등장으로 이전의 영상과는 비교도 안될 정도로 고화질의 영상이 실시간에 처리할 수 있을 정도가 됐다. 이에 따라 최근에는 보다 실감나는 영상을 보여주기 위한 방법으로 텍스쳐 매핑(Texture Mapping), 범프 매핑(Bump Mapping), 환경 매핑(Environment Mapping) 등과 같은 여러 가지 매핑 테크닉들과 앨리어싱(Aliasing)을 없애주는 여러 가지 기법들에 대한 연구가 주요 이슈였으며 현재에도 계속해서 연구가 진행 중에 있다[1].

장면의 객체 위에 텍스쳐 이미지를 그려 넣는 방법에는 두 가지 방법으로 구현이 가능하다. 하나는 실세계의 각각의 객체들의 기본 정보(색, 3차원 좌표 등) 뿐만 아니라, 표면의 텍스쳐 정보 역시 모델링을 통해서 데이터베이스에 저장해 둔 후, 해당 장면의 렌더링 과정에서 로드하여 처리해주는 방법이다. 그러나 이 방법은 다량의 메모리를 필요로 하고, 매핑을 위해 많은 메모리 대역폭을 요구하므로 거의 사용되어지지 않고 있다. 또 다른 방법으로, 텍스쳐 정보와 3차원 객체의 기본 정보를 분리하여 저장해 두었다가 객체의 렌더링 과정에서 관련된 텍스쳐 정보를 해당 객체에 그려 넣는 방법이 있다. 이는 전자의 경우에 비해, 텍스쳐 정보를 위한 메모리 요구량이 많지 않으며, 메모리 대역폭 역시 크지는 않다. 그러나 복잡한 장면의 경우에는 이 역시 저장 메모리와 메모리 접근 측면에서 상당한 문제점을 갖게 된다. 후자의 경우를 보통 텍스쳐 매핑이라고 부른다.

텍스쳐 매핑의 처리는 래스터라이징 과정에서 이루어진다. 즉 3차원의 객체 공간이 아닌, 2차원의 스크린 공간에서 각 픽셀에 대한 텍스쳐 값을 보간해 낸다. 3차원 객체 공간의 속성을 갖고 있는 텍스쳐 데이터를 단순히 2차원에서 보간 처리하는 것은 영상의 텍스쳐 표현에 있어서 왜곡현상을 야기한다[2].

본 논문에서는 고성능 3차원 그래픽 처리를 위해 프로세서-메모리 집적 방식을 이용하여 래스터라이징 유닛과 프레임버퍼를 단일 칩으로 구성하였다. 그리고, 스크린 공간에서의 텍스쳐 왜곡현상과 [9]에서 제안한 시스템의 문제점을 해결할 수 있는 새로운 시스템을 제안한다.

본 논문의 구성은 다음과 같다. 2절에서는 3차원 그래픽 처리과정 중 래스터라이징 과정에 대해 살펴보고, 3절에서는 텍스쳐 매핑에 대하여 전반적으로 설명한다. 4절에서는 렌더링 알고리즘에 대해 살펴보겠다. 그리고 5절에서는 3차원 렌더링 시스템을 제안하고, 기존 방식과의 차이점에 대해 설명한다. 마지막으로 6절에서는 결론과 향후 연구계획에 대하여 논한다.

2. 래스터라이징 과정

래스터라이징(Rasterizing)은 상당한 계산을 요한다. 왜냐하면 디스플레이할 최종 이미지를 프레임버퍼(frame buffer)에 보내기 전에, 컬러와 텍스쳐, 그리고 투명도에 대한 보간이 행해져야 하기 때문이다. 처음에 하나의 삼각형은 세 개의 꼭지점의 좌표(x, y, z)와 각 꼭지점의 컬러값인 RGBα로 정의된다. 주사변환(Scan Conversion) 수행 이전에 삼각형의 변의 보간(Edge Walk)과 스팬 보간(Span Interpolation)에 사용되는 증가율(Increment)들을 계산해야 한다.

증가율을 구하는 과정에는 삼각형의 기울기(Slope) 계산, 각 매개변수들의 변화율(Gradient) 계산, 원근 텍스쳐를 위한 좌표 제법 등 여러 가지 변수들에 대한 나눗셈이 수행되어야 한다. 이 과정에서 완전한 파이프라인의 처리가 가능하도록 하기 위해서는 적어도 5개 이상의 독립적인 역수 검색 테이블이 필요하다[12]. 가드 비트를 가진 역수 테이블의 사용은 계산의 복잡도에 따른 지연의 감소와 래스터라이저 크기를 줄여주는 실용적인 방법이다[13]. 또한 삼각형의 컬러, 깊이, 텍스쳐 좌표 증가율은 모두 같은 역수 값에 의해 계산된다:

$$\frac{\partial Z}{\partial X}=\frac{(Z_2-Z_1)\cdot(y_3-y_1)-(Z_3-Z_1)\cdot(y_2-y_1)}{(X_2-X_1)\cdot(y_3-y_1)-(X_3-X_1)\cdot(y_2-y_1)}$$

$$\frac{\partial Z}{\partial y}=\frac{(Z_3-Z_1)\cdot(X_2-X_1)-(Z_2-Z_1)\cdot(X_3-X_1)}{(X_2-X_1)\cdot(y_3-y_1)-(X_3-X_1)\cdot(y_2-y_1)}$$

따라서 이미 계산된 값의 참조를 통한 증가율 계산에서 생길 수 있는 오버헤드를 줄일 수 있다.

3. 텍스쳐 매핑

이미지를 객체에 매핑할 때, 객체의 각 픽셀에서의 색은 상응하는 이미지의 색에 의해 수정되어진다. 일반적으로 이미지로부터 상응되는 색을 가져오는 것은 개념적으로 여러 과정이 필요하다[3]. 이미지는 보통 샘플된 어레이로써 저장되므로, 연속적인 이미지는 먼저 샘플로부터 재구성되어져야 한다. 그런 다음, 이미지는 디스플레이되어 질 수 있도록 사영된 객체에서 생기는 어떤 왜곡(아마도 원근감의 표현에 의해 생기는 왜곡)에 적절히 맞춰질 수 있도록 비틀리거나 휘어져야 한다. 다시 비틀어진 이미지는 고주파에 해당하는 요소를 제거하기 위해 필터링의 과정을 거치게 된다. 이와 같은 과정을 거치는 이유는 마지막 단계인 재샘플링의 과정에서 고주파 요소가 앨리어싱을 야기하기 때문이다. 재샘플링 과정은 텍스쳐 처리되어질 픽셀에 적용할 올바른 색을 얻기 위한 과정이다.

실용적으로 위의 필터링 과정은 여러 가지 방법 중 하나에 의해 근사하게 수행된다. 그 중 가장 많이 사용되는 방법은 밉-매핑(Mip-Mapping)이다[6].

위의 기본적인 텍스쳐 매핑 기법의 일반화된 기법이 많이 있다. 맵되어질 이미지가 반드시 이차원으로 구성될 필요는 없다. 샘플링과 필터링은 1차원이나 3차원의 이미지를 가지고 적용될 수도 있다[8].

기본적인 텍스쳐 매핑에서는, 텍스쳐 이미지는 폴리곤의 꼭지점에 텍스쳐 좌표를 할당함으로써 폴리곤에 적용된다. 이 텍스쳐 좌표는 텍스쳐 이미지를 인덱스하고 폴리곤의 각 픽셀에서 텍스쳐 이미지 값을 결정하기 위해 폴리곤에 걸쳐 보간되어 진다.

현재 많은 그래픽스 시스템에서 텍스쳐 매핑을 지원하는 하드웨어를 제공하고 있다. 대개의 경우 텍스쳐 매핑에 의해 장면을 만드는 데 걸리는 시간이 그렇지 않은 경우에 비해 그다지 오래 걸리지는 않는다.

텍스쳐 매핑 방식은 단순히 텍스쳐 매핑에 국한되지 않고, 에어브러시(Air-brush), 볼륨 렌더링(Volume Rendering), 퐁 쉐이딩(Phong Shading), 환경 매핑(Environment Mapping) 등과 같이 흥미로운 응용에 쓰여질 수 있는 그래픽스 드로잉 방법이다[7].

4. 원근투영 매핑을 위한 선형보간법

4.1 Affine 매핑과 투영 매핑(Projective Mapping)

3차원 그래픽스에서는 Affine 매핑과 투영 매핑, 두 가지 종류의 매핑을 사용한다. (u,v)에서 (x,y)로의 2차원의 투영 매핑의 일반적인 형태는 다음과 같다[2].

$$x=\frac{au+bv+c}{gu+hv+i},\quad y=\frac{du+ev+f}{gu+hv+i}.$$

이를 동치행렬표기방식(Homogeneous Matrix Notation)으로는 더욱 간단하게 표현된다.

$$(xw\ \ yw\ \ w)=(uq\ \ vq\ \ q)\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}.$$

Affine 매핑에는 축소확대(Scale), 회전(Rotation), 직선변환(Translation), 전단변환(Shear)이 있다. 만일 $g=h=0$ 이고 $i\neq 0$ 이기만 하면 2차원 투영 매핑은 Affine 매핑이다.

4.2 선형보간에 의한 폴리곤 렌더링

선형보간 알고리즘은 다음과 같다[4].

(1) 폴리곤의 각 꼭지점과 연관된 매개변수를 포함하는 레코드를 연계시킨다.

(2) 각 꼭지점에 대해, 4×4 객체 대 스크린 행렬을 이용하여 객체 공간좌표를 동치 스크린 공간좌표 (xw,yw,zw,w)로 변환한다.

(3) 시각 절단체(View Frustum, View Volume)의 여섯 면 각각에 대한 평면방정식으로 폴리곤을 클리핑하고, 새로 만들어진 꼭지점의 모든 매개변수를 선형보간한다.

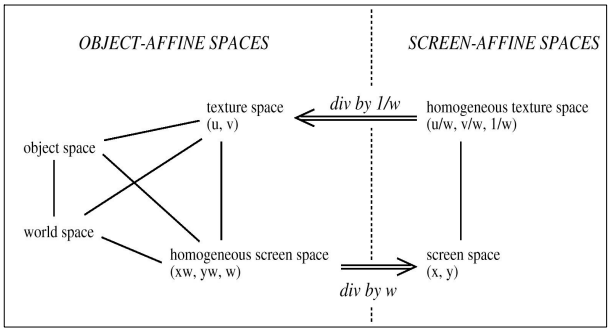
(4) $x=xw/w$, $y=yw/w$, $z=zw/w$ 를 계산하기 위해 동치제법을 수행한다.

(5) 음영을 입히기 위해 각 픽셀에서의 매개변수 값을 사용하여 모든 매개변수를 선형보간함으로써 스크린공간 내에서 주사변환(Scan Conversion)한다.

이 알고리즘의 (1)~(4) 단계는 기하학 연산과정에서 수행하고, 마지막 단계는 래스터라이징 과정에서 수행된다. 그런데 이 알고리즘에서는 객체 공간에 있는 좌표들을 스크린 좌표계로 표준화하기 위해 단순히 동치 좌표를 축척 인자(Scale Factor)로 나누어 주는 과정만을 수행해 주고 있다. 이는 텍스쳐 매핑과 같은 스크린 좌표계에서의 처리과정에 있어서 문제점을 야기한다.

4.3 선형보간의 문제점

선형보간은 텍스쳐 매핑 뿐만 아니라, 종종 고라우드 쉐이딩(Gouraud Shading), 퐁 쉐이딩(Phong Shading)에도 이용된다. 그러나 객체 공간에 연관된 매개변수를 사용하여 스크린 공간에서 선형보간을 수행하는 것은 잘못이다. 객체 공간에서 스크린 공간으로는 원근투영에 의한 매핑이 수행되기 때문이다. 따라서 위의 선형보간 알고리즘은 평행투영(Parallel Projection)이나 시점방향(Viewing Direction)과 수직인 평면의 원근투영의 경우와 같이 객체 공간과 스크린 공간이 선형변환(Linear Transformation)에 의해 매핑될 수 있는 경우에만 적용된다. 이러한 선형보간의 단점은 특히 텍스쳐 매핑의 경우에 두드러진다. 선형보간에 의한 텍스쳐 매핑의 경우 원근감을 표현해 낼 수 없다. 텍스쳐 또한 꼭지점을 가로지르는 수평선을 따라 불연속성(Discontinuity)을 띄게 된다. 그리고 여러 면을 가진 폴리곤에 대해서 Rubber Sheet 현상과 회전 변동(Rotational Variation) 현상이 발생한다. 즉 회전 불변(Rotational Invariation)한 Affine 매핑에 대해서만 선형보간이 올바르게 적용된다. 3차원 장면을 만들어내기 위해서는 기본적으로 원근투영기법을 사용하게 되므로 스크린 공간에서 객체공간 매개변수를 선형보간하는 것은 부적절하다. 이에 대한 해결책으로 종종 삼각분할(Triangulation) 방식을 사용하기도 한다. 새로운 꼭지점에서의 매개변수값을 객체 공간에서 선형보간하는 방법으로, 즉 하나의 폴리곤을 여러 개의 작은 폴리곤으로 분할하여 실제 폴리곤에 근사(Approximation)시키는 방법인 폴리곤 세분(Polygon Subdivision)도 있으나 이는 폴리곤의 수가 증가함에 따라 빠른 픽셀처리속도를 감소시키는 단점이 생길 수 있다.



[그림 1] 기본적인 텍스쳐 매핑에 필요한 좌표계들간의 상호연관성.

4.4 유리선형보간법(Rational Linear Interpolation)

앞에서 얘기한 문제점을 해결하기 위해 유리선형보간법이 제시되었다[5]. [그림 1]은 기본적인 텍스쳐 매핑에 필요한 좌표계들간의 상호연관성을 나타

낸다. 여기서 객체-Affine 공간은 객체 공간과 Affine 매핑을 통하여 변환이 가능한 공간을 의미한다. 스크린-Affine 공간은 스크린 공간과 Affine 매핑을 통하여 변환이 가능한 공간을 의미한다. 이미 말했듯이 객체 공간과 스크린 공간은 원근투영매핑을 통하여 변환된다.

만일 매개변수들이 객체 공간과 Affine하고, w=1인 조건을 가지고 객체 공간의 점을 변환함으로써 동치 스크린 좌표를 계산한다면, 동치 매개변수 벡터 $(\frac{r_1}{W},\cdots,\frac{r_n}{W},\frac{1}{W})$가 스크린 공간에 Affine한다. 따라서 이 벡터는 스크린 공간에서 선형보간법에 의해 보간될 수 있다[그림 1].

여기에서 제시한 방법은 지오메트리 단계에서와 같은 변환과정을 거치지 않고 단지 그 점에서 각 매개변수를 w로 나누고, $\frac{1}{W}$과 함께 다른 매개변수를 보간변수리스트에 부가한다. n개의 매개변수를 계산하기 위해, 꼭지점 당 n+1번의 나눗셈을 수행하고 n+1개의 변수가 보간되어져야 한다. 각 매개변수를 계산하기 위해 각 픽셀 당 보간된 $\frac{1}{W}$로 n개의 보간된 동치 매개변수를 나누어 준다.

$$r_i(x,y)=\frac{r_i(x,y)/w(x,y)}{1/w(x,y)}=\frac{a_ix+b_iy+c_i}{Ax+By+C}$$

대부분의 연산처리기에서 n번의 나눗셈을 수행하는 데 가장 빠른 방법은 제수의 역수를 계산한 다음, n번의 곱셈을 해주는 방법이다. 만일 w의 값이 모든 꼭지점에 대해 동일하다면, 매개변수들이 폴리곤에 대해 스크린-Affine하다. 따라서 각 픽셀에 대한 나눗셈을 피할 수 있다.

4.5 유리선형보간법에 의한 렌더링 알고리즘

다음은 유리선형보간 알고리즘이다[12].

(1) 폴리곤의 각 꼭지점과 연관된 n개의 매개변수 $(r_1,r_2,\cdots,r_n)$를 포함하는 레코드를 연계시킨다.

(2) 각 꼭지점에 대해, 4×4 객체-스크린 행렬을 이용하여 객체 공간좌표를 동치 스크린 공간좌표 (xw,yw,zw,w)로 변환한다.

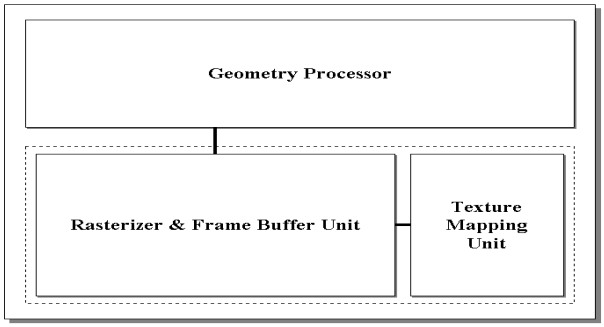
(3) 시각 절단체의 여섯 면 각각에 대한 평면방정식으로 폴리곤을 클리핑하고, 새로 만들어진 꼭지점의 모든 매개변수를 선형보간한다.

(4) 각 꼭지점에 대해, 변수리스트 $(x,y,z,r_1/w,r_2/w,\cdots,r_n/w,1/w)$를 만들기 위해 동치 스크린좌표와 매개변수 r_i , 그리고 1을 w로 나누어준다.

(5) 각 픽셀에 음영을 입히기 위해 n개의 매개변수 각각에 대해 $\frac{r_i/w}{1/w}$을 수행한 다음 그 값들을 선형보간함으로써 스크린 공간에서 주사변환한다.

이 알고리즘의 (1)~(3)의 단계는 선형보간 알고리즘과 동일하다. 그러나 나머지 두 단계에서는 원근투영 폴리곤의 처리를 위해 원근제법(Perspective Division)을 수행한다.

5. 렌더링 시스템

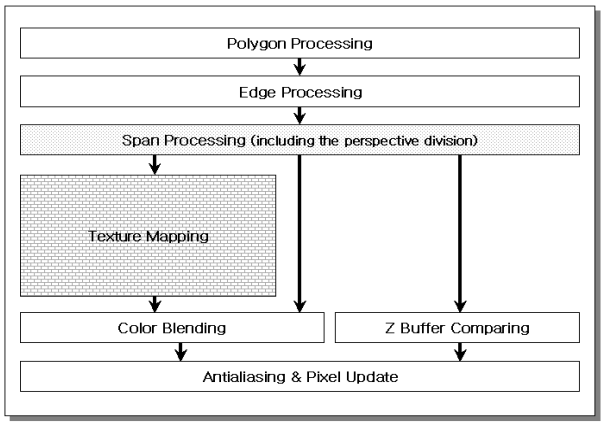


[그림 2] 시스템 모델

[그림 2]는 [9]에서 제안한 시스템 모델로써, 본 논문에서 기본적으로 가정하고 있는 시스템 모델이다. 이는 기존의 3차원 그래픽 가속기에서와 달리 기하학 연산과정을 완전히 CPU와 분리하여 별도의 그래픽 칩 내에 삽입함으로써 CPU의 기하학 연산에 따른 짐을 덜어줄 수 있다. 또한 래스터라이징 유닛과 프레임버퍼 유닛을 단일 칩으로 구성하였다. 그리고 텍스쳐 매핑을 위해 별도로 텍스쳐 매핑 유닛을 두었는데, 이는 방대한 텍스쳐 데이터의 처리를 위한 것이다.

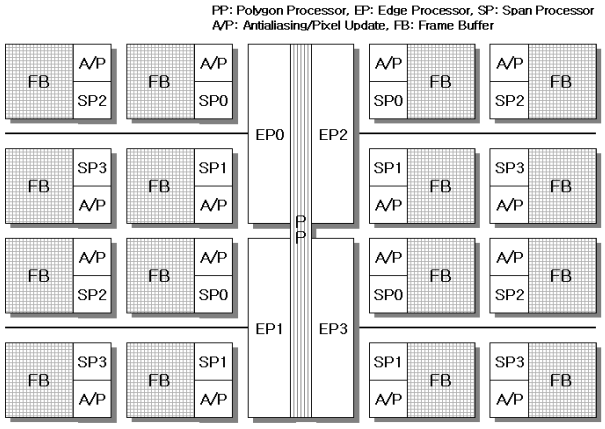
[9]에서 제안한 시스템에는 몇 가지 문제점이 있다. 폴리곤 처리과정이 기하학 처리과정에 있기 때문에 래스터라이저-프레임버퍼 유닛사이의 데이터의 전송되는 양이 폴리곤 처리과정이 래스터라이징 유닛의 상단에 놓이는 경우에 비해 상당히 크다는 것이다. 그리고 또 다른 문제점으로는 텍스쳐/범프 매핑 주소생성 로직에 필요한 하드웨어가 클 것을 예상하여 에지 프로세서 당 하나씩 구성되어있다는 것인데, 이는 텍스쳐 매핑 수행 시 스팬 처리를 4배정

도 지연시키는 결과를 초래할 수 있다. 더욱 큰 문제는 원근투영 매핑 시의 왜곡현상에 대해 고려하지 않았다는 것이다.



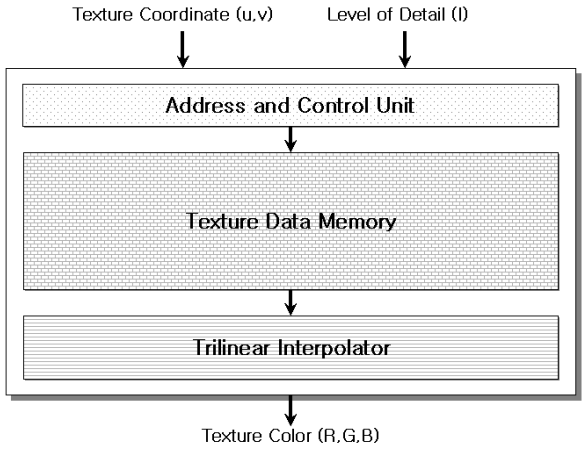
[그림 3] 래스터라이징 흐름도

본 시스템은 폴리곤 처리를 위한 셋업(Setup)의 과정을 기하학 처리 유닛에서 래스터라이징 유닛의 상단으로 옮겼다[12]. 이에 따라 기하학 처리 유닛과 래스터라이징 유닛 사이에서 발생하는 대역폭 문제를 해결하였다. 또한 원근투영 폴리곤에 따른 왜곡현상을 해소하기 위해 스팬 처리 과정에서 원근제법을 수행하였다. 원근제법은 $\frac{1}{W}$의 값을 구하기 위한 역수 검색 테이블과 이 값을 각 매개변수에 곱하기 위한 곱셈기로 구성되어 있다. 그리고 [9]에서 제안하였던 것과는 달리, 텍스쳐 데이터의 주소생성을 위한 데이터패스와 주사변환 처리과정을 위한 데이터패스를 분리하지 않고 공통된 부분을 서로 오버랩시킴으로써 효율적인 처리가 가능하도록 설계하였다[그림 3].



[그림 4] 래스터라이저와 프레임버퍼의 구조

[그림 4]는 본 논문에서 제안하는 래스터라이저-프레임버퍼의 구조를 보여준다. 제안하는 구조는 하나의 폴리곤 처리유닛과 4개의 에지 처리유닛으로 구성되어 있으며, 또한 프레임버퍼 메모리를 16개의 뱅크로 인터리빙(Interleaving)하였으며, 각 뱅크마다 하나의 스팬 처리유닛과 앤티앨리어싱/픽셀업데이트 유닛을 할당하였다. 그리고 스팬 처리 유닛에서는 원근을 고려한 나눗셈을 처리해 줌으로써 생성될 이미지의 질을 향상시켰다.



[그림 5] 텍스쳐 매핑 유닛

텍스쳐 매핑에는 많은 텍스쳐 메모리를 요구한다. 이를 적은 대역폭을 가진 오프칩 메모리로 구성한다면, 이에 대한 접근에 따른 지연은 3차원 그래픽 처리에 상당한 장애를 준다. 이와 같은 문제를 해결하기 위해 텍스쳐 매핑 처리를 위한 로직과 텍스쳐 메모리를 프로세서-메모리 집적 방식을 이용하여 하나의 칩에 구성하였다[그림 5].

이 유닛은 텍스쳐의 주소와 각 레벨을 결정하는 제어 유닛(Address and Control Unit), Trilinear 밉-매핑을 처리하기 위하여 8개의 뱅크로 구성된 텍스쳐 데이터 메모리, 텍스쳐 데이터를 보간하기 위한 Trilinear 보간 유닛으로 구성하였다[10]. 여기서 Trilinear 보간은 텍스쳐 데이터의 계산 속도를 향상시키기 위하여 두 개가 Bilinear 보간 유닛과 하나의 선형보간 유닛을 사용하였다[11].

6. 결론 및 향후 계획

본 논문에서는 주사변환 과정과 텍스쳐 매핑과정에서의 공통되는 부분인 스크린 좌표, 스크린 매개변수, 텍스쳐 좌표의 보간 과정을 오버랩시킴으로써 래스터라이징 과정을 최적화하였으며, 원근투영 폴

리곤의 처리에 따른 왜곡 현상을 해소하기 위하여 원근제법처리를 스팬 처리 유닛 내에 구성하였다. 또한 프로세서-메모리 집적 방식을 이용하여 래스터라이징 과정과 프레임버퍼 사이의 대역폭 문제를 해결하였으며, 방대한 메모리를 요구하는 텍스쳐 매핑 유닛을 하나의 칩으로 분리하여 텍스쳐 매핑 시에 발생하는 메모리 지연현상 및 대역폭 문제를 해결하였다.

현재 설계한 시스템을 검증하기 위한 시뮬레이션 환경을 구축하고 있으며, 향후에는 3차원 객체의 구체적인 표현 기법인 범프 매핑이나 환경 매핑, 더 나아가 퐁 쉐이딩을 지원하는 고성능의 렌더링 시스템을 설계할 것이다.

참고문헌

[1] Frederick M. Weinhaus and Venkat Devarajan, "Texture Mapping 3D Models of Real-World Scenes," *ACM Computing Surveys*, Vol. 29, No. 4, pp. 325-365, December 1997.

[2] Paul S. Heckbert, "Survey of Texture Mapping," *IEEE Computer Graphics and Applications*, Vol. 6, No. 11, pp. 56-67, November 1986.

[3] Paul S. Heckbert, "Fundamentals of texture mapping and image warping," M.sc.thesis, Dept. of EE and CS, University of California, Berkeley, June 1989.

[4] Paul S. Heckbert, "Generic Convex Polygon Scan Conversion and Clipping," Graphics Gems, Andrew Glassner, ed., Academic Press, Boston, 1990.

[5] P. S. Heckbert and H. P. Moreton, "Interpolation for Polygon Texture Mapping and Shading," *State of the Art in Computer Graphics: Visualization and Modeling*, Springer-Verlag, pp. 101-111, 1991.

[6] Lance Williams, "Pyramidal parametrics," *Computer Graphics (SIGGRAPH'83 Proceedings)*, Vol. 17, No. 3, pp. 1-11, July 1983.

[7] Paul Haeberli, Mark Segal, "Texture Mapping as a Fundamental Drawing Primitive," *Proceedings of the 4th Eurographics Workshop on Rendering*, pp. 259-266, June 1993, Paris, France.

[8] D. R. Peachey, "Solid texturing of complex surfaces," *Computer Graphics (SIGGRAPH'85 Proceedings)*, Vol. 19, No. 3, pp. 279-286, July 1985.

[9] 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. October 1999.

[10] Andreas Schilling, Günter Knittel, and Wolfgang Strasser, "Texram: A Smart Memory for Texturing," *IEEE Computer Graphics and Applications*, Vol. 16, No. 3, pp. 32-41, May 1996.

[11] Tzi-cker Chiueh, "Heresy: A Virtual Image-Space 3D Rasterization Architecture," *1997 SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pp. 69-77, 1997.

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

[13] D. DasSarma, D. Matula, "Measuring the Accuracy of ROM Reciprocal Tables," *IEEE Transactions on Computers*, Vol. 43, No. 8, pp. 932-940, August 1994.