각 연산을 이용한 효과적인 범프 매핑 하드웨어 구조 설계

(Design of an Effective Bump Mapping Hardware Architecture Using Angular Operation)

이 승 기 † 박 우 찬 †† 김 상 덕 † 한 탁 돈 †††
(Seung-Gi Lee) (Woo-Chan Park) (Sang-Duk Kim) (Tack-Don Han)

요 약 범프 매핑은 복잡한 모델링 과정 없이 기하 매핑을 통하여 땅콩 껍질의 돌기와 같은 객체 표면의 세밀한 부분을 표현해내는 기법이다. 그러나 이 기법은 법선 벡터 쉐이딩과 같은 상당한 복잡도를 가진 연산을 픽셀 당 처리해줘야 하므로, 이의 하드웨어 구현은 상당한 비용을 필요로 한다. 본 논문에서는 극 좌표계를 이용한 새로운 범프 매핑 알고리즘 및 하드웨어 구조를 제안한다. 이는 참조 공간으로의 변환을 위한 새로운 벡터 회전 방식과 연산이 최소화된 조명 계산 방식을 갖는 구조로, 기존의 구조에 비해 범프 매핑을 효과적으로 수행한다. 결과적으로 제안하는 구조는 범프 매핑에 필요한 연산 및 하드웨어를 상당량 줄였다.

키워드 : 3차원 그래픽스 하드웨어, 범프 매핑, 극 좌표계, 각 회전

Abstract Bump mapping is a technique that represents the detailed parts of the object surface, such as a perturberance of the skin of a peanut, using the geometry mapping without complex modeling. However, the hardware implementation for bump mapping is considerable, because a large amount of per pixel computation, including the normal vector shading, is required. In this paper, we propose a new bump mapping algorithm using the polar coordinate system and its hardware architecture. Compared with other existing architectures, our approach performs bump mapping effectively by using a new vector rotation method for transformation into the reference space and minimizing illumination calculation. Consequently, our proposed architecture reduces a large amount of computation and hardware requirements.

Key words : 3D Graphics Hardware, Bump Mapping, Polar Coordinate System, Angular Rotation

1. 서 론

현재 대부분의 3차원 그래픽 가속기 시스템(3D graphics accelerator system)에서는 보다 세밀하고 현실적인 영상을 생성하기 위해 텍스쳐 매핑(texture mapping), 범프 매핑(bump mapping), 반사 매핑(reflection mapping) 등의 실감 매핑 기법이 사용되고 있다[1-5]. 텍스쳐 매핑과 반사 매핑은 폴리곤으로 표현된 객체의 표면 위에 2차원 텍스쳐 이미지를 입히는 기법이다[1,2,6-10]. 범프 매핑은 매끄러운 객체의 표면에 융기나 홈 같은 것을 표현해주는 기법으로, 2차원의 범프 맵으로부터 해당 픽셀의 변위값을 가져와 법선 벡터를 변형하여 새로이 조명(illumination)을 계산해줌으로써 수행된다[1,2,9,11].

그런데 이들 기법들 중 텍스쳐 매핑과 반사 매핑은 좌표 계산과 텍스쳐 데이터의 필터링과 같은 정적인 계산 과정만을 통해 수행이 가능하다. 그러나 범프 매핑은 공간 상의 벡터들의 재설정 및 새로운 조명 계산과 같이 픽셀에 따라 공간 구성 요소가 바뀌는 동적인 연산의 수행을 통해 가능하다. 범프 매핑을 처리하기 위해서는 픽셀 당 법선 벡터 쉐이딩(normal vector shading)을 처리해 주어야 한다[12,13]. 따라서 현재의 대부분의 범프 매핑 관련 연구는 Phong shading 방식을 기반으로 하여 진행되고 있다[9,13-18].

† 비 회 원 : 연세대학교 컴퓨터과학과
sklee@kurene.yonsei.ac.kr
askieee@cs.yonsei.ac.kr
†† 비 회 원 : 세종대학교 컴퓨터공학부 교수
pwchan@korea.com
††† 종신회원 : 연세대학교 컴퓨터과학과 교수
hantack@kurene.yonsei.ac.kr
논문접수 : 2002년 10월 17일
심사완료 : 2003년 8월 13일

범프 매핑의 처리는 법선 벡터 변형(normal vector perturbation), 벡터의 단위 벡터화(vector normalization), 그리고 조명 계산(illumination calculation)으로 구성된다. 이 중 법선 벡터 변형은 공간-의존적인(space-dependent) 연산으로, 공간상의 위치의 방향 벡터를 통해 매 픽셀마다 재계산을 해줘야하는 부담을 갖고 있다. 이 문제를 해결하기 위해 제시된 방법은 참조 프레임(reference frame) 또는 참조 공간(reference space)을 이용하는 것이다[16,17,19,20]. 이 방법은 미리 정해진 공간 - 접평면에 의해 이루어진 공간(tangent space) - 을 이용하여 법선 벡터의 공간-의존적인 성질을 제거함으로써 법선 벡터 변형 연산의 전처리를 가능케할 뿐만 아니라, 지역적으로 편평하다는 가정의 적용을 통해 이 연산에 필요한 연산량을 상당히 줄일 수 있는 장점을 가지고 있다[16]. 여기서 참조 공간은 범프 매핑을 가능한 정확하게 계산해주기 위해서 픽셀 혹은 작은 폴리곤마다 존재해야 한다.

참조 공간 위에서 범프 매핑을 수행하기 위해서는 빛의 방향 벡터 L과 중간 벡터 H를 각 픽셀 혹은 작은 폴리곤에 대해 참조 공간 위의 좌표계로의 변환(transformation) 연산을 수행하여 참조 공간에서의 조명 계산에 필요한 L', H'을 생성해 주어야 한다. 이러한 변환 연산에는 참조 공간으로의 변환 행렬(transformation matrix)의 재정의(re-definition)가 필요한데, SGI[16]의 경우에는 객체 표면 위의 점에서의 법선 벡터 n, 접선 벡터 t, 그리고 종법선 벡터 b를 사용하여 변환 행렬을 정의하였다.

그런데 이들 벡터들은 일반적으로 직교 좌표계(Cartesian coordinate system) 방식으로 표현된다. 따라서 이들 벡터들을 참조 공간으로 변환하기 위해서는 3×3 행렬 연산을 필요로 할 뿐만 아니라, 변환 후에 조명 계산을 위해 각 벡터를 단위 벡터화 해주어야 한다[16,21]. 반면 이들 벡터들을 극 좌표계(polar coordinate system)에 의한 벡터 표현 방식을 이용하여 처리하면 하드웨어적으로 효과적이다[17]. 즉, 벡터를 크기 정보를 제외한 두 개의 각(angle) 정보만으로 표현함으로써 전달될 데이터의 양을 줄일 수 있을 뿐만아니라, 벡터의 단위 벡터화 과정 없이 L과 H의 변환을 통해 빛의 세기를 계산할 수 있다. 결과적으로 참조 공간에서의 법선 벡터 변형과 각에 의한 벡터 표현을 같이 이용하면 범프 매핑에 필요한 연산 및 하드웨어를 상당히 줄일 수 있는데, Kim[17]이 한 예이다.

본 논문에서는 극 좌표계 방식으로 표현된 벡터들의 각 연산을 통한 새로운 범프 매핑 처리 알고리즘 및 하드웨어 구조를 제안한다. 기존의 변환 방식에서는 참조 공간 위의 새로운 지역 좌표계를 정의하여 사용하지만, 제안 회전 변환 방식에서는 새로운 좌표계의 정의 없이 기존의 객체 공간 좌표계를 그대로 사용한다. 이는 객체 위의 한 점에서의 법선 벡터 N의 두 각 만큼 L과 H를 회전하여 참조 공간 위의 L'과 H'으로 변환하는 효과를 줌으로써 범프 벡터 맵의 변형된 법선 벡터 N'을 객체 공간에서 직접 사용할 수 있게 해준다. 제안 방식은 참조 공간 위의 좌표계로의 변환 행렬을 재정의할 필요가 없을 뿐만 아니라, 기존의 방식들에 비해서 변환에 필요한 하드웨어가 적게 소요된다. 또한 본 논문에서 제안하는 조명 계산 방식은 각을 이용한 방식들 중 최소한의 연산 및 하드웨어를 필요로 한다. 결과적으로 제안하는 구조는 알고리즘 및 하드웨어 비용 측면에서 효율적인 범프 매핑 방식이다.

본 논문의 구성은 다음과 같다. 2장에서는 범프 매핑 하드웨어 구조와 관련된 연구에 대해 소개한다. 3장에서는 범프 매핑의 수행 흐름에 따른 처리 방법에 대해 개략적으로 설명하고, 4장에서는 벡터의 극 좌표계 표현을 이용한 회전 변환 및 조명 계산 알고리즘에 대해 상술한다. 5장에서는 제안하는 범프 매핑 하드웨어 구조에 대해 설명하고 이를 기존 구조와 비교한다. 6장에서는 제안 구조의 실험 환경 및 결과를 보여준다. 마지막으로 7장에서는 결론을 맺는다.

2. 관련 연구

범프 매핑은 J. F. Blinn[11]에 의해 처음으로 소개된 기법으로, 복잡한 객체의 모델링 과정 없이 간단한 기하 매핑을 통하여 복잡한 객체를 표현해 내는데 유용하게 사용된다. 범프 매핑은 기본적으로 두 가지 과정을 거쳐 수행된다. 첫 번째 과정에서는 객체 표면 위의 한 점에서 접평면에 수직인 법선 벡터 N을 범프 맵의 변위값에 의해 기울임으로써 변형된 법선 벡터 N'을 구한다. 이의 계산식은 식 (1)과 같다. 여기서 B_u, B_v는 범프 변위값을 각각 매개 방향(parameter direction) u, v에 대해서 편미분한 것이고, P_u, P_v는 객체 표면 위의 점의 위치벡터를 표현하는 함수 P를 각각 매개 방향 u, v에 대해서 편미분한 것이다. 두 번째 과정에서는 변형된 법선 벡터 N'에 대한 새로운 조명을 계산하는데, 일반적으로 식 (2)와 같은 Phong 조명 모델(illumination model)을 사용한다. 여기서 I_a는 주변광(ambient light)의 세기, I_i는 입사광(incident light)의 세기, 그리고 k_a, k_d, k_s는 $k_a+k_d+k_s=1$을 만족하는 주변광, 확산광(diffuse light), 경면광(specular light)의 비례 상수이다.

$$N' = N + B_u(N \times P_v) - B_v(N \times P_u) \quad (1)$$

$$I = I_a k_a + I_i(L \cdot N') + k_s(N' \cdot H)^n) \quad (2)$$

지난 수년에 걸쳐 Blinn이 제안한 범프 매핑을 하드

웨어적으로 처리해 주기 위한 연구가 Phong 쉐이딩 구조와 연관하여 진행되어 왔다. Truga001[14,15,18]은 단일 LSI 칩에 범프, 반사, 굴절(refraction), 그리고 텍스처 매핑을 지원하는 고성능의 Phong 쉐이딩 하드웨어를 구현하였다. 이는 법선 벡터를 변형하기 위해 하드웨어 부담이 큰 행렬연산 로직을 사용하고 있으며, 조명계산을 위하여 본래의 Phong 조명 모델을 그대로 구현하고 있다. 여기서 경면광의 세기 계산에 필요한 반사각(reflection angle) α를 L과 N의 선형 보간(linear interpolation)에 의해 구한다. 그러나 α의 정확한 계산을 보장할 수 없으며, 크기가 큰 폴리곤의 경우에는 상당히 큰 오차가 생긴다.

텍스처, 범프, 반사 매핑을 단일 칩에서 처리할 수 있는 구조를 가진 IMEM[22]은 산술 연산 유닛과 참조 테이블을 하나의 전용 메모리 칩에 집적하였다. 이 구조는 법선 벡터를 극 좌표계 표현으로 변환하고 세 번의 회전을 통하여 변형한다. 벡터 회전은 두 번의 덧셈과 한 번의 LUT(Look-Up Table) 참조를 통해 수행되고, 조명 계산은 확산광 맵과 경면광 맵을 이용한다. IMEM은 미리 계산된 LUT들과 맵들을 이용하여 연산량을 줄였지만, 사용한 맵들의 크기가 상당히 크다.

Visa+[13,21]에서는 변환 행렬 A를 이용하여 텍스처 맵에 저장되어 있는 변위 벡터를 지역 폴리곤 좌표계에서 텍스처 좌표계로 투영시킨 다음, 표면 범선 벡터에 더함으로써 변형된 법선 벡터를 구하는 구조를 제시하였다. 그리고 확산광 맵과 경면광 맵의 참조를 통해 픽셀의 색을 계산하였다. 이 구조는 매 픽셀마다 재정의된 변환 행렬 A에 의한 행렬곱 연산을 수행해야 하며, 맵의 주소를 계산하기 위해 나눗셈 연산과 ATAN 연산을 필요로 한다.

Texram[19]에서는 정규직교 지역 좌표계(orthonormal local coordinate system)를 통한 MML (Merged Memory Logic) 구조의 전용 범프 매핑 유닛을 제시하였다. 이는 보간된 법선 벡터 N과 고정된 참조축(constant polar axis)과 같은 중심방향 h로부터 정규직교 좌표계를 형성하여 $N\times P_v$, $N\times P_u$의 계산에 따른 연산을 제거하였으나, 새로운 지역 좌표축 n, e_1, e_2를 만들기 위해 단위 벡터화와 외적(cross product)에 따른 연산을 처리해줘야 한다.

SGI[16]에서는 Phong shading 구조에 최소한의 하드웨어를 부가함으로써 범프 매핑을 지원할 수 있는 구조를 제시하였다. 범프 매핑 수행에 있어서 상당한 연산량을 필요로 하는 법선 벡터 변형 연산을 참조 공간(탄젠트 공간)을 도입하여 래스터라이저가 아닌, 전처리기나 정점 당 프로세서(per-vertex processor)에서 처리하도록 하였다. 그 결과 L과 H의 참조 공간으로의 변환과

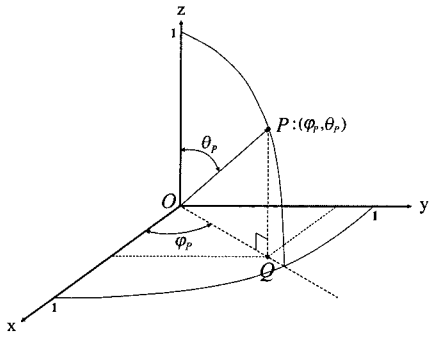

그림 1 벡터의 극 좌표계 표현

조명 계산을 통하여 범프 매핑을 처리할 수 있게 된다. Ernst[20]은 SGI의 범프 매핑 알고리즘에 의거하여 제안된 구조로, SGI의 벡터 변형식의 k를 -1로 고정하여 범프 맵을 간단하게 구성하였다. 그런데 SGI와 Ernst은 조명 계산을 위해 3×3 행렬 변환연산과 각 벡터의 단위 벡터화를 필요로 한다.

IMEM[22], Truga001[14,15], Kim[17]은 벡터의 극 좌표계 표현을 이용한 범프 매핑 처리 방법에 대해 논하고 있다. 벡터 P의 극 좌표는, 그림 1에서와 같이, 벡터 P를 xy 평면 위로 사영한 벡터 Q와 x축이 이루는 각 φ_P와 벡터 P와 z축이 이루는 각 θ_P에 의해 (φ_P, θ_P)로 표현된다. 그러나 IMEM, Truga001은 상당히 복잡하거나 비싼 하드웨어 비용을 수반한다. Kim은 범프 맵으로부터 변형각 ($\varphi_{N'}$, $\theta_{N'}$)을 입력받아 내적(inner product)을 계산해주는 식 (3)과 식 (4)를 유도하여 조명 연산 하드웨어의 복잡도를 다소 줄였다.

$$n'_L \cdot l_L = \cos\theta_l \cos\theta_{n'} + \sin\theta_l \sin\theta_{n'} \cos(\varphi_l - \varphi_{n'}) \quad (3)$$

$$n'_L \cdot h_L = \cos\theta_h \cos\theta_{n'} + \sin\theta_h \sin\theta_{n'} \cos(\varphi_l - \varphi_{n'}) \quad (4)$$

그러나 위에서 제시한 방식들은 새로운 좌표계를 사용하기 때문에 좌표계 변환 연산을 수행하기 이전에 반드시 변환을 위한 행렬을 매 픽셀 (혹은 매 폴리곤) 마다 재정의해줘야 한다는 단점이 있다.

3. 제안 범프 매핑 기법의 수행 흐름

그림 2는 본 논문에서 제안하는 범프 매핑 방법을 보여준다. 벡터 회전 단계(Vector Rotation Stage)에서는 그림 1에서와 같이 극 좌표계 형태로 표현된 벡터들을 3차원 객체 공간에서 3차원 참조 공간으로 회전 변환한다. 이 변환은 객체 공간 위의 빛의 방향 벡터 $L=(\varphi_L, \theta_L)$과 중간 벡터 $H=(\varphi_H, \theta_H)$를 회전값인 법선 벡터 $N=(\varphi_N, \theta_N)$을 이용하여 참조 공간 위로 변환해줌으로써 참조 공간에 맞는 지역 조명 환경인

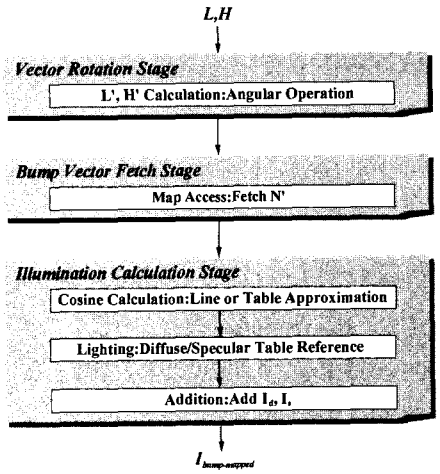

그림 2 범프 매핑 수행 흐름도

L', H'을 생성해 준다. 이는 구 상에서의 비례를 이용하여 계산되어지는데, 이에 대한 알고리즘과 하드웨어 구조는 각각 4.1장과 5.1장에서 자세히 설명한다.

범프 벡터 패치 단계(Bump Vector Fetch Stage)에서는 범프 벡터 맵으로부터 전처리 단계에서 미리 계산된 변형된 법선 벡터 N'을 가져오는 역할을 수행한다. 범프 벡터 맵은 참조(tangent) 공간 위에 위치해 있으며, SGI[16]에서 제시했던 방식에 의해 전처리 단계에서 계산되어진다. 맵의 벡터 표현은 극 좌표계 방식을 사용하여 $(\varphi_{N'}, \theta_{N'})$과 같이 표현된다.

조명 계산 단계(Illumination Calculation Stage)에서는 참조 공간 위로 옮겨진 벡터들 L', H'과 변형된 법선 벡터 N'을 입력으로 받아 내적을 계산한 다음, 계산된 값들을 이용하여 확산광 테이블과 경면광 테이블을 참조하여 새로운 조명값을 계산한다. 이 단계의 알고리즘과 하드웨어 구조에 대한 자세한 논의는 각각 4.2장과 5.2장에 있다.

4. 제안 범프 매핑 알고리즘

본 장에서는 벡터의 극 좌표계 표현 방식을 이용하여 범프 매핑에 필요한 연산 및 하드웨어를 최소화할 수 있는 처리 방식에 대해 논한다. 4.1장에서는 극 좌표계 방식의 새로운 벡터 회전 알고리즘을 제안하고, 4.2장에서는 조명 계산을 최소화하는 방식을 제시한다.

4.1 벡터 회전

4.1.1 제안하는 벡터 회전 방식

제안하는 벡터 회전 방식은 실세계 공간(world space) 상의 법선 벡터 $N=(\varphi_N, \theta_N)$을 참조 공간 상의 법선 벡터 $N=(0,0)$과 일치시킴으로써 참조 공간 위에 있는 미리 계산된 변형된 법선 벡터 $N'=(\varphi_N, \theta_N)$과 회전에 의해 생성되어질 $L'=(\varphi_{L'}, \theta_{L'})$, $H'=(\varphi_{H'}, \theta_{H'})$으로 구성된 조명 환경을 구축한다. 이 방식은 변환하고자 하는 벡터를 실세계 공간의 좌표축 z, y에 대해서 순서대로 회전함으로써 수행된다. 여기서 벡터의 회전각으로 객체 표면 위의 한 점에서의 법선 벡터 N의 φ_N과 θ_N을 이용한다. 그림 3과 그림 4는 이를 보여주는데, 이 회전 변환을 통해 벡터 A는 벡터 A_{ref}로 변환된다. 벡터 A_{ref}의 $\varphi_{A_{ref}}$와 $\theta_{A_{ref}}$는 각각 사영과 구 상에서의 비례를 이용하여 구할 수 있다. 벡터 $A=(\varphi_A, \theta_A)$는 실세계 공간 상에 있는 임의의 벡터로써, 범프 매핑 수행 시에는 빛의 방향 벡터 $L=(\varphi_L, \theta_L)$과 중간 벡터 $H=(\varphi, \theta_H)$가 이에 해당한다.

그림 3에서와 같이, 첫 번째 회전에서는 벡터 $N=(\varphi_N, \theta_N)$과 $A=(\varphi_A, \theta_A)$를 z축을 중심으로 $-\varphi_N$

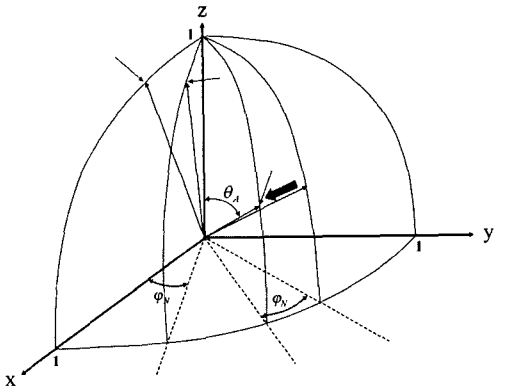

그림 3 첫 번째 회전: z축을 중심으로 $-\varphi_N$ 만큼 회전시킨다.

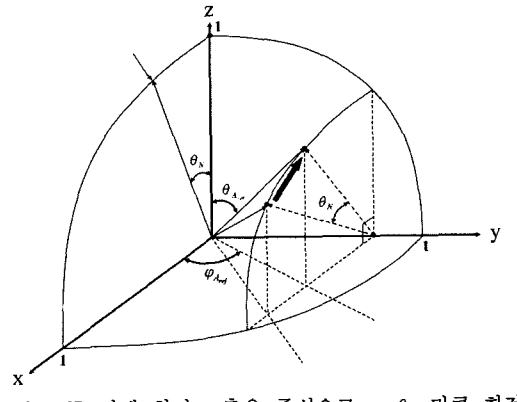

그림 4 두 번째 회전: y축을 중심으로 $-\theta_N$ 만큼 회전시킨다.

만큼 회전한다. 이 회전을 통해 법선 벡터 $N=(\varphi_N,\theta_N)$은 $N_Z=(0,\theta_N)$으로 바뀌고, $A=(\varphi_A,\theta_A)$은 $A_z=(\varphi_A-\varphi_N,\theta_A)$로 바뀐다. 이 회전은 벡터 N을 xz 평면 위의 벡터 N_z로 옮김으로써 y축에 대한 정확한 회전을 가능하게 한다.

다음으로, 그림 4와 같이, 변환된 벡터 N_z와 A_z를 y축을 중심으로 회전각 $-\theta_N$ 만큼 회전함으로써 두 번째 회전을 수행한다. 이 회전을 거치면 벡터 N_z는 실세계 공간의 z축 방향의 단위 벡터와 일치하게 되며, 참조 공간의 법선 벡터 N_{ref}와 동일한 방향을 향하게 된다. 마찬가지로 벡터 A 또한 $A_{ref}=(\varphi_{A_{ref}},\theta_{A_{ref}})$로 이동하여 참조 공간 위의 벡터 A'과 동일한 벡터가 된다.

4.1.2 벡터 회전 후의 각을 구하기 위한 정보들

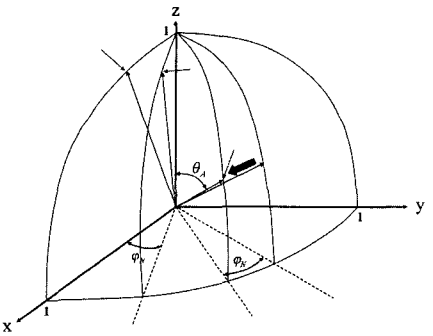

그림 5 변환된 벡터 $A_{ref}(=A')$의 구성요소들을 구하는데 필요한 벡터들의 기하 정보

그림 5는 최종 변환된 벡터 A'의 $\varphi_{A'}$과 $\theta_{A'}$을 구하는데 필요로 하는 벡터들의 정보들을 보여주고 있다. 그림 5에서 새로 정의한 벡터들에 대한 설명은 다음과 같다.

- $A_{z_{xz}}$: 원점을 시점으로 하고, A_z를 지나면서 yz 평면에 평행한 원과 xz 평면이 만나는 점을 종점으로 하는 단위 벡터. 이 벡터와 z축 사이의 각을 θ_x라 놓는다. θ_x는 $\theta_{A'}$를 구하는데 필요한 각으로, 이에 대한 자세한 설명은 아래에 있다.
- $A_{z_{xy}}$: 벡터 A_z를 xy 평면 위에 사영한 벡터. 이 벡터의 x, y 좌표는 각각 $\sin\theta_A\cos(\varphi_A-\varphi_N)$, $\sin\theta_A\sin(\varphi_A-\varphi_N)$이다.
- $A_{ref_{xz}}$: 원점을 시점으로 하고, A'을 지나면서 yz 평면에 평행한 원과 xz 평면이 만나는 점을 종점으로 하는 단위 벡터. 이 벡터와 z축 사이의 각은 $(\theta_x-\theta_N)$이고, x, z 좌표는 각각 $\sin(\theta_x-\theta_N)$과 $\cos(\theta_x-\theta_N)$이다.
- $A_{ref_{xy}}$: 벡터 A'을 xy 평면 위에 사영한 벡터. 이 벡터의 x, y 좌표는 각각 $\sin(\theta_x-\theta_N)$, $\sin(\theta_A\sin(\varphi_A-\varphi_N)$이다.

앞에서 정의한 θ_x는 식 (5)와 같이 벡터 $A_{z_{xz}}$의 x 좌표에 cosine 법칙을 적용한 다음, Sin^{-1} 함수를 이용하여 구할 수 있다.

$$\theta_x=Sin^{-1}\frac{1}{2}\left[\sin\left(\frac{\theta_A+(\varphi_A-\varphi_N)}{2}\right)+\sin\left(\frac{\theta_A-(\varphi_A-\varphi_N)}{2}\right)\right] \quad (5)$$

여기서 X, Y를 식 (6)과 같이 놓으면, θ_x는 식 (7)과 같이 쓸 수 있다.

$$X=\frac{\theta_A+(\varphi_A-\varphi_N)}{2},\quad Y=\frac{\theta_A-(\varphi_A-\varphi_N)}{2} \quad (6)$$

$$\theta_x=Sin^{-1}\left[\frac{\sin X+\sin Y}{2}\right] \quad (7)$$

4.1.3 $\varphi_{A'}$의 계산

그림 6은 벡터 A'을 xy 평면 위로 사영(projection)한 것으로, 벡터 A'의 $\varphi_{A'}$을 구하는 데 이용된다. $\varphi_{A'}$는 식 (8)과 같이 Tan^{-1} 함수를 사용하여 쉽게 구할 수 있다.

$$\varphi_{A'}=Tan^{-1}\left[\frac{\sin\theta_A\sin(\varphi_A-\varphi_N)}{\sin(\theta_x-\theta_N)}\right] \quad (8)$$

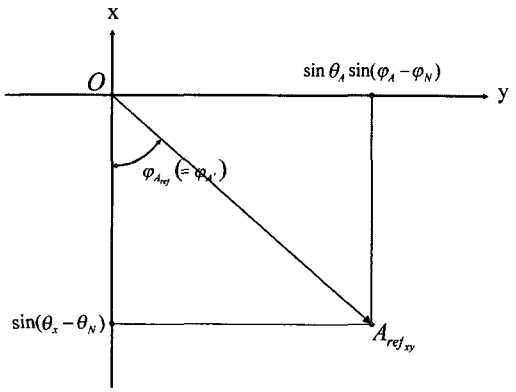

그림 6 $\varphi_{A'}$의 계산: 벡터 A'을 xy 평면 위로의 사영함으로써 사영 벡터와 x축이 이루는 각 $\varphi_{A'}$을 계산한다.

여기서 그림 5에 있는 벡터 $A_{ref_{xz}}$의 z축과 이루는 각을 식 (9)와 같이 Z라 놓은 다음, cosine 법칙을 적용하면 식 (8)은 식 (10)과 같이 다시 쓰여질 수 있다.

$$Z = \theta_x - \theta_N \tag{9}$$

$$\varphi_{A'} = Tan^{-1}\left[\frac{\cos Y - \cos X}{2\sin Z}\right] \tag{10}$$

4.1.4 벡터와 구 간의 기하학적 상관 관계 및 이를 통한 각 계산식 유도

벡터 A'이 z축과 이루는 각 $\theta_{A'}$을 구하는 방법은 $\varphi_{A'}$를 구하는 방법에 비해 상당히 복잡하다. 각 $\theta_{A'}$을 구하기 위해서는 먼저 직교 좌표계에서의 벡터와 구 간의 기하학적 상관 관계를 규정하는 것이 필요하다. 이는 규정된 기하학적 상관 관계를 통해 비로소 각 $\theta_{A'}$을 구할 수 있을 뿐만 아니라, 각 $\theta_{A'}$을 구하는 데 필요한 각 변화 정도 θ_{prop}도 구할 수 있기 때문이다. 본 장에서는 벡터와 구 간의 기하학적 상관 관계를 규정하며, 기하학적 비례를 이용한 각 $\theta_{A'}$의 계산 과정을 설명한다. 계속해서 4.1.5장에서는 주어진 기하학적 정보를 이용한 각 변화 정도 θ_{prop}의 계산 과정에 대해 서술한다.

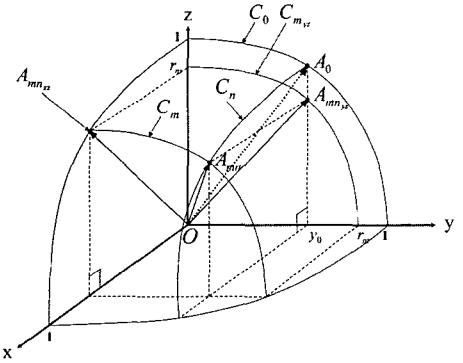

그림 7 벡터 A_{mn}의 xz 평면, yz 평면과의 기하학적 상관 관계

그림 7은 임의의 벡터 A_{mn}의 xz 평면과의 기하학적 상관 관계와 yz 평면과의 기하학적 상관 관계를 보여주는데, 이에 대한 설명은 다음과 같다.

- C_m : yz 평면과 평행하면서 점 $(\varphi_{A_{mn}}, \theta_{A_{mn}})$을 지나는 원. 이 원의 반지름을 r_m이라 놓는다.
- $C_{m_{yz}}$: 원 C_m을 yz 평면 위로 사영한 원.
- C_n : xz 평면과 평행하면서 점 $(\varphi_{A_{mn}}, \theta_{A_{mn}})$을 지나는 원.
- $A_{mn_{xz}}$: 원점을 시점으로 하고 점 $(0, \theta_{A_{mn_{xz}}})$을 종점으로 하는 벡터. 여기서 점 $(0, \theta_{A_{mn_{xz}}})$은 원 C_m과 xz 평면이 만나는 점이다.
- A_0 : 원점을 시점으로 하고 점 $(\varphi_{A_0}, \theta_{A_0})$을 종점으로 하는 벡터. 여기서 점 $(\varphi_{A_0}, \theta_{A_0})$는 원 C_n과 yz 평면이 만나는 점이고, 이 점의 y 좌표를 y_0라 놓는다.
- $A_{mn_{yz}}$: 벡터 A_{mn}을 yz 평면에 사영한 벡터. 이 벡터의 종점 $(\varphi_{A_{mn_{yz}}}, \theta_{A_{mn_{yz}}})$는 원 $C_{m_{yz}}$ 위에 있다.

그림 7에서 원점을 시점으로 하고 xz 평면과 평행한 평면 위의 점을 종점으로 하는 임의의 두 벡터가 각각 원 C_m와 원 $C_{m_{yz}}$를 따라 움직일 때, 두 벡터의 z축과 이루는 각의 변화폭은 각각 $90 - \theta_{A_{mn_{xz}}}$ 과 90이다. 그런데 이 벡터들의 임의의 위치에서의 변화각과 변화폭의 비는 원 C_m과 원 $C_{m_{yz}}$을 따라 일정하다. 따라서 다음의 비례식은 성립한다(식 (11)).

$$\frac{\theta_{A_{mn}} - \theta_{A_{mn_{xz}}}}{90 - \theta_{A_{mn_{xz}}}} = \frac{\theta_{A_{mn_{yz}}}}{90} \tag{11}$$

$$\theta_{A_{mn}} = \theta_{A_{mn_{xz}}} + \frac{\theta_{A_{mn_{yz}}}}{90}\left(90 - \theta_{A_{mn_{xz}}}\right) \tag{12}$$

여기서 θ_{prop}을 식 (13)과 같이 놓으면, 식 (12)는 식 (14)와 같이 쓸 수 있다.

$$\theta_{prop} = \frac{\theta_{A_{mn_{yz}}}}{90} \tag{13}$$

$$\theta_{A_{mn}} = \theta_{A_{mn_{xz}}} + \theta_{prop}\,(90 - \theta_{A_{mn_{xz}}}) \tag{14}$$

4.1.5 벡터의 위치에 따른 θ_{prop}의 계산

전 장에서 유도된 식 (14)의 $\theta_{A_{mn}}$은 θ_{prop}의 계산을 거친 후에 구할 수 있다. θ_{prop}은 그림 8과 같이 $\theta_{A_{mn_{yz}}}$의 계산을 이용하여 구할 수 있다. 그림 8은 그림 7에서 정의한 벡터 A_{mn}과 벡터 A_{mn}의 관련 요소들을 yz 평면 위로 사영한 것이다.

그림 8에서 벡터 A_{mn}을 yz 평면 위로 사영한 벡터 $A_{mn_{yz}}$의 연장선과 원 C_0가 만나는 점의 y좌표는 y_0 / r_m 이다. 여기서 벡터 $A_{mn_{yz}}$와 y축이 이루는 각의 cosine을 구하면 식 (15)와 같다.

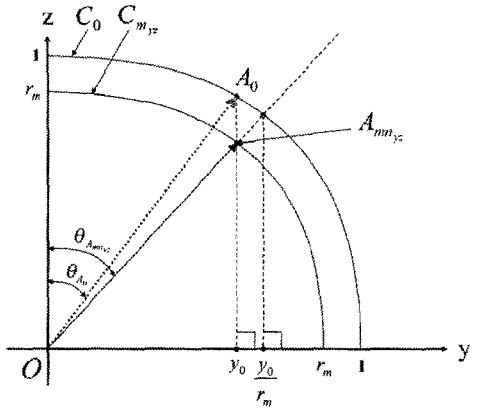

그림 8 벡터 A_{mn}과 이의 관련 요소들의 yz 평면 위로의 사영

$$\frac{y_0}{r_m} = \cos\left(90 - \theta_{A_{mn_{yz}}}\right) \tag{15}$$

식 (15)를 $\theta_{A_{mn_{yz}}}$ 을 좌변으로 하여 정리하면 식 (16)과 같다.

$$\theta_{A_{mn_{yz}}} = 90 - Cos^{-1}\frac{y_0}{r_m} \tag{16}$$

식 (16)에 식 (13)을 적용하면 θ_{prop}은 식 (17)과 같이 구할 수 있다. 여기서 r_m과 y_0는 이미 계산되어 있기 때문에 θ_{prop}은 계산될 수 있으며, 0과 1 사이의 값을 갖는다.

$$\theta_{prop} = 1 - \frac{1}{90}Cos^{-1}\frac{y_0}{r_m} \tag{17}$$

4.1.6 $\theta_{A'}$의 계산

이제 이전 장들에서 유도한 식들을 통해 변환된 벡터 A'의 $\theta_{A'}$를 구할 수 있다. 식 (14)로부터 $\theta_{A'}$는 식 (18)과 같이 구할 수 있다.

$$\theta_{A'} = \theta_{A'_{xz}} + (90 - \theta_{A'_{xz}}) \times \theta_{prop} \tag{18}$$

그런데, 그림 5에서 알 수 있듯이, 벡터 A'의 $\theta_{A'_{xz}}(=\theta_{A_{ref\,xz}})$는 $(\theta_x - \theta_N)$이므로 식 (18)은 식 (19)와 같이 쓸 수 있다.

$$\theta_{A'} = (\theta_x - \theta_N) + [90 - (\theta_x - \theta_N)] \times \theta_{prop} \tag{19}$$

식 (19)에 식 (9)를 적용하면, 변환된 벡터 A'의 $\theta_{A'}$은 식 (20)과 같이 다시 쓸 수 있다.

$$\theta_{A'} = Z + (90 - Z) \times \theta_{prop} \tag{20}$$

한편 θ_{prop}은, 4.1.4장에서 제시한 방법에 의거해 계산하면, 식 (21)과 같은 계산식으로 표현될 수 있다.

$$\theta_{prop} = 1 - \frac{1}{90}Cos^{-1}\frac{\sin\theta_A \sin(\varphi_A - \varphi_N)}{\cos(\theta_x - \theta_N)} \tag{21}$$

식 (21)은 cosine 법칙과 식 (6), 식 (9)를 적용하여 다시 쓰면 식 (22)와 같다.

$$\theta_{prop} = 1 - \frac{1}{90}Cos^{-1}\left[\frac{\cos Y - \cos X}{2\cos Z}\right] \tag{22}$$

4.2 조명 계산

조명 계산 단계에서는 4.1장에서 제시한 벡터 회전 방식에 의해 구해진 벡터 $L' = (\varphi_{L'}, \theta_{L'})$, $H' = (\varphi_{H'}, \theta_{H'})$와 범프 벡터 패치 단계에서 보내진 벡터 $N' = (\varphi_{N'}, \theta_{N'})$을 이용하여 픽셀의 색의 세기를 구하는데, 이의 계산식은 식 (2)와 같다. 식 (2)의 벡터들의 내적을 구하기 위해서는 해당 벡터들을 식 (23), 식 (24), 식 (25)와 같이 극좌표계 표현에서 직교 좌표계 표현으로 변환해야 한다.

$$N' = (\sin\theta_{N'}\cos\varphi_{N'}, \sin\theta_{N'}\sin\varphi_{N'}, \cos\theta_{N'}) \tag{23}$$

$$H' = (\sin\theta_{H'}\cos\varphi_{H'}, \sin\theta_{H'}\sin\varphi_{H'}, \cos\theta_{H'}) \tag{24}$$

$$L' = (\sin\theta_{L'}\cos\varphi_{L'}, \sin\theta_{L'}\sin\varphi_{L'}, \cos\theta_{L'}) \tag{25}$$

변환된 벡터를 이용하여 벡터의 내적 $N' \bullet L'$과 $N' \bullet H'$을 구하면 다음과 같다.

$$N' \bullet L' = \cos\theta_{L'}\cos\theta_{N'} + \sin\theta_{L'}\sin\theta_{N'}\cos(\varphi_{L'} - \varphi_{N'}) \tag{26}$$

$$N' \bullet H' = \cos\theta_{H'}\cos\theta_{N'} + \sin\theta_{H'}\sin\theta_{N'}\cos(\varphi_{H'} - \varphi_{N'}) \tag{27}$$

식 (26), 식 (27)은 각각 Kim[17]에서 유도한 식 (3), 식 (4)와 동일하다. 그런데 Kim에서 유도된 식에 cosine의 법칙을 적용하면 내적의 연산량을 줄일 수 있다. 이는 벡터들의 각의 합과 차에 대한 cosine을 계산함으로써 구할 수 있으며, 이의 계산식은 식 (28), 식 (29)와 같다. 이는 극 좌표계로 표현된 벡터들의 내적 계산에 있어 곱셈과 cosine의 계산량을 줄여주는 계산식이다.

$$\begin{aligned} N' \cdot L' &= \tfrac{1}{2}[\cos(\theta_{L'} + \theta_N) + \cos(\theta_{L'} - \theta_N)] \\ &\quad + \tfrac{1}{2}[\cos(\theta_{L'} + \theta_N) - \cos(\theta_{L'} - \theta_N)] \\ &\quad \cos(\varphi_{L'} - \varphi_N) \end{aligned} \tag{28}$$

$$\begin{aligned} N' \cdot H' &= \tfrac{1}{2}[\cos(\theta_H + \theta_N) + \cos(\theta_H - \theta_N)] \\ &\quad + \tfrac{1}{2}[\cos(\theta_H + \theta_N) - \cos(\theta_H - \theta_N)] \\ &\quad \cos(\varphi_{H'} - \varphi_N) \end{aligned} \tag{29}$$

식 (28)과 식 (29)에 의해 계산된 내적의 값들을 이용하여 확산광의 세기와 경면광의 세기를 구한 다음, 두 값을 더함으로써 최종의 범프-매핑된 픽셀의 색의 세기 값을 구해낸다.

5. 제안 범프 매핑 알고리즘에 대한 하드웨어 구조

5.1 벡터 회전 유닛

벡터 회전 유닛은 식 (20)과 식 (21)를 처리해주기 위해 6개의 테이블, 3개의 곱셈기, 8개의 덧셈/뺄셈기를 필요로 하는데, 이의 하드웨어 구조는 그림 9와 같다.

Sine과 cosine의 계산은 테이블(sin & cos table)을 이용하는데, 이 테이블들은 sine과 cosine 테이블로 구성되어 있다. Sine과 cosine의 역수(reciprocal) 계산은 sine과 cosine을 구한 후에 역수를 구하는 두 과정(2-step)이 아닌 한 과정에 처리할 수 있도록 단일화(unification)하였으며, sine & cosine table과 마찬가지로, sine의 역수 테이블과 cosine의 역수 테이블로 분리되어 구성된 1/sin & 1/cos table을 이용한다. Kim[17]과 IMEM[22]에서 사용한 ROM 테이블 구조를 본 제안 구조의 sin & cos table, 1/sin & 1/cos table에도 동일하게 적용하고 있는데, 이는 극 좌표계 방식에서의 일반적인 접근 방식이다. 비록 테이블 사용으로 인해 계산값의 오차가 발생하지만, 6장의 실험 결과에서 알 수 있듯이, 사람의 눈으로 화질의 차이를 쉽게 구별하기 힘듦을 알 수 있다.

θ_{prop}은 식 (17)을 임의의 값 y_k에 적용한 식 (30)에 의해 미리 계산되어 비례 테이블 T_{prop}에 저장되어 있다.

$$\theta_{prop} = 1 - \frac{Cos^{-1} y_k}{90} \tag{30}$$

비례 테이블 T_{prop}에 저장되어 있는 θ_{prop}은 식 (22)로부터 얻어진 식 (31)에 의해 y_k를 계산함으로써 가져올 수 있다.

$$y_k = \frac{\cos Y - \cos X}{2\cos Z} \tag{31}$$

5.2 조명 계산 유닛

조명 계산 유닛은 벡터 회전 유닛의 출력인 L', H'과 범프 벡터 맵으로부터 가져온 N'을 입력으로 받아 범프 맵된 픽셀의 값을 계산한다. 그림 10은 벡터 회전 유닛 및 조명 계산 유닛을 포함한 제안하는 전체 범프 매핑 하드웨어 구조이다. 여기서 조명 계산 유닛은 두 부분으로 나뉘어지는데, 확산광 세기를 계산하는 부분과 경면광 세기를 계산하는 부분이 이에 해당한다. 각 부분에서는 식 (28)과 식 (29)의 내적을 계산하고 테이블 참조에 의한 빛의 세기를 계산한다. 이 유닛은 6개의 cosine 테이블, 2개의 diffuse/specular light 테이블, 2개의 곱셈기, 그리고 13개의 덧셈/뺄셈기로 구현된다. 각을 이용하여 내적을 계산한 이전의 방법인 Kim[17]의 경우가 12개의 테이블, 6개의 곱셈기, 그리고 5개의 덧셈/뺄셈기를 필요로 했던 것에 비하면, 제안 유닛은 하드웨어를 상당히 줄여주는 구조이다. 이와 관련하여 제안 구조와 다른 구조들 간의 비교를 5.3장에서 논한다.

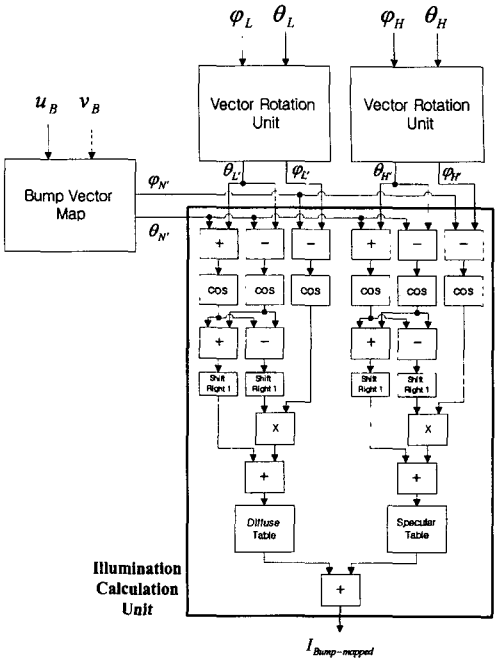

그림 10 제안 범프 매핑 하드웨어

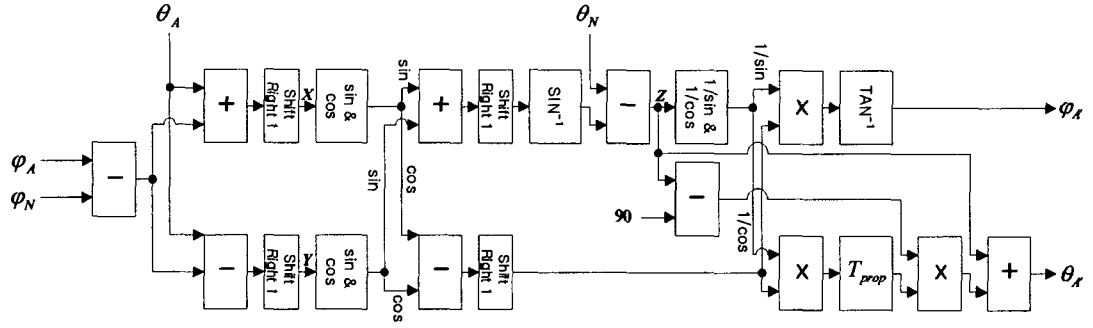

그림 9 벡터 회전 유닛(Vector Rotation Unit)

Light 테이블 방식[17]을 이용한 빛의 세기 계산 방법은 조명 매개변수에 따라 미리 계산해 놓은 diffuse light 테이블과 specular light 테이블을 참조하여 빛의 세기를 구한다. 이 방식은 계산된 내적 값, 즉 스칼라 값에 의해 테이블의 엔트리를 인덱싱하므로 테이블의 크기가 $2^8\times16\sim2^{10}\times16$ 비트로 아주 작은 반면, IMEM[22]과 같이, 2~3개의 좌표, 즉 벡터 값에 의해 인덱싱되는 맵 방식의 경우에는 $2^{20}\times16$ 비트 이상으로 그 크기가 상당히 크다.

5.3 기존의 다른 구조와의 비교

표 1은 범프 매핑 처리 과정을 크게 두 부분으로 나누어 이전에 제안되었던 구조들과 본 논문에서 제안한 구조 간의 연산 하드웨어의 복잡도를 비교한 것이다. 첫 번째 부분은 조명 계산을 준비하는 단계로, 참조 공간으로의 변환과 법선 벡터 변형, 그리고 단위 벡터화 과정을 포함한다. 두 번째 부분은 조명을 계산하는 단계이다. 여기서 덧셈기와 뺄셈기는 동일시하여 덧셈기로 표기하였다. 비교된 각 구조의 덧셈, 곱셈, 나눗셈, 그리고 SQRT 연산의 입력은 모두 32 비트를 사용하고 있다. 그리고, 극 좌표계 방식에서 사용하고 있는 LUT들의 입력은 8~10 비트를, 엔트리 당 데이터 크기는 16 비트를 사용하고 있다.

먼저 조명 계산 준비 단계의 하드웨어 복잡도에 대해 살펴보면 다음과 같다. Blinn[11]의 방법에 의하면 법선 벡터 변형을 수행하기 위해서는 20번의 곱셈과 8번의 덧셈을 필요로 한다. 이를 간단히 처리하기 위해서 IMEM[22]에서는 LUT와 Map을 이용하여 구현하였다. 이 구조는 5개의 곱셈기, 2개의 덧셈기, 3개의 LUT, 그리고 1개의 맵으로 구성되어 있는데, 맵 크기가 32KB에 이른다. 참조 공간 방식을 이용한 SGI[16], Ernst[20]의 경우에는 두 벡터 L, H의 참조 공간으로의 변환을 위해 두 번의 3×3 행렬연산을 해줘야 하므로 18개의 곱셈기와 12개의 덧셈기를 필요로 하고, 변환된 벡터를 단위 벡터화하는데 9개의 곱셈기, 3개의 SQRT, 9개의 나눗셈 유닛, 그리고 6개의 덧셈기를 필요로 한다. Kim[17]은 극 좌표계 표현 방식을 이용하여 변환 연산을 수행했는데, 이 연산 유닛은 31개의 곱셈기, 6개의 LUT, 그리고 18개의 덧셈기로 구현된다. 그런데 SGI, Ernst, Kim의 경우에는 변환을 위한 행렬의 재계산이 필요하므로 표 1에 나와 있는 것에 비해 훨씬 더 많은 연산량을 필요로 한다. 반면 제안 구조에서는 행렬 연산 없이 단지 회전과 테이블의 참조만으로 참조 공간으로의 변환을 처리해줌으로써 변환 연산에 필요한 하드웨어를 6개의 곱셈기, 12개의 LUT, 그리고 16개의 덧셈기로 줄였다.

한편 조명 계산 단계의 하드웨어 복잡도를 각 구조별로 살펴보면 다음과 같다. IMEM은 light 맵의 주소 생성 연산과 light 맵의 접근, 그리고 색 혼합(color blending)을 통해 내적 연산 없이 조명을 계산한다. 8개의 곱셈기, 4개의 덧셈기, 그리고 5개의 LUT를 주소 생성 연산을 하는데 사용하며, 빛의 세기를 구하기 위해 2개의 1MB light 맵을 사용한다. Ernst는 Phong 조명 모델의 직접적인(straightforward) 방법을 사용하여 조명을 계산하고 있으며, 7개의 곱셈기, 5개의 덧셈기, 그리고 1개의 제곱(power) 테이블을 필요로 한다. Kim은 각에 의해 벡터의 내적을 계산하고 계산된 값을 이용하여 확산광/경면광 테이블을 참조함으로써 조명 계산을 수행한다. Kim의 조명 계산 하드웨어는 6개의 곱셈기, 12개의 LUT, 그리고 5개의 덧셈기로 구성되어 있다. 제안 구조는 Kim과 동일한 방식을 이용하여 조명을 계산하며, 2개의 곱셈기, 8개의 LUT, 그리고 11개의 덧셈기를 필요로 한다.

각 구조 간의 상대적인 비교에 덧붙여, 제안 구조에서 사용하는 각각의 연산 하드웨어의 증감에 따른 특성에 대해 살펴보면 다음과 같다. 본 제안 구조에서는 다른 구조에 비해 곱셈기가 상당수 감소된 반면, 상대적으로 LUT의 수가 증가하였다. 하드웨어의 구현 측면에 대해 언급한 Piñeiro[23]에 의하면, 8~10 입력 비트와 16 비

표 1 제안 구조와 기존의 다른 구조와의 하드웨어 복잡도 비교

Item		SGI[16]	IMEM[22]	Kim[17]	Ernst[20]	Ours
Coordinate System		Cartesian	Polar	Polar	Cartesian	Polar
Processing Step	Illumination Environment Setup	27 Multipliers 9 Division Units 3 SQRTs 18 Adders	1 Map 5 Multipliers 3 LUTs 2 Adders	31 Multipliers 6 LUTs 18 Adders	27 Multipliers 9 Division Units 3 SQRTs 18 Adders	6 Multipliers 12 LUTs 16 Adders
	Illumination Calculation	Not Available	8 Multipliers 2 Maps 5 LUTs 4 Adders	6 Multipliers 12 LUTs 5 Adders	7 Multipliers 1 LUT 5 Adders	2 Multipliers 8 LUTs 11 Adders

트 엔트리를 갖는 ROM 테이블의 하드웨어 면적(area)은 32×32 곱셈기 면적의 1/4 정도이며, 딜레이 또한 절반 수준으로 상당히 빠른 처리가 가능하다고 한다. 제안 구조의 경우, Illumination Environment Setup에서 12개의 LUT가 3개의 곱셈기와, Illumination Calculation에서 8개의 LUT가 2개의 곱셈기와 동일한 면적을 차지한다.

표 1에서 알 수 있듯이, 제안 구조를 Ernst와 같이 직교 좌표계를 이용한 다른 구조와 비교해 보면, 조명 계산에 필요한 하드웨어 뿐만 아니라, 조명 계산의 준비 단계에 요구되는 하드웨어 역시 상당히 많이 줄어든다. 또한 IMEM, Kim과 같이 극 좌표계를 이용한 다른 구조에 대해서도, 제안 구조는 두 과정의 처리에 요구되는 하드웨어를 상당히 줄여줌을 알 수 있다. 결론적으로 제안 구조는 이전의 다른 구조들에 비해 연산 하드웨어의 복잡도 측면에서 상당히 우수한 구조라 할 수 있다.

6. 실험 결과

본 논문에서 제시한 범프 매핑 알고리즘과 하드웨어 구조에 대한 가능성을 C 언어를 이용하여 검증하였다. 본 실험은 OpenGL 3차원 그래픽 라이브러리와 매우 유사한 Mesa 3.0 [24]을 기반으로 이루어졌다. 또한 제안 범프 매핑과 기존의 범프 매핑을 지원할 수 있도록 약간의 코드를 수정하였고, 부가의 코드를 삽입하였다.

첫 번째 실험에서는 범프 맵의 해상도에 따른 제안 범프 매핑의 수행을 보여준다. 그림 11과 그림 12는 격자 형태의 범프 맵을 각각 평면과 구에 적용하여 생성한 영상들이다. 그림 11(a)는 해상도 256×256인 범프 맵을 원근 투영된 평면 위에 매핑한 것이고, 그림 11(b)는 해상도 512×512인 범프 맵을 좀더 비스듬한 위치에 놓여있는 평면 위에 매핑한 것이다. 그림 12(a)는 해상도가 256×256인 범프 맵을 이용하여 범프 매핑을 수행한 것이고, 그림 12(b)는 해상도가 512×512인 범프 맵을 구의 표면 위에 매핑하여 생성한 영상이다. 실험 결과에서 알 수 있듯이, 평면 및 구 위에 범프 맵이 적절히 매핑되어지는 것을 알 수 있다.

두 번째 실험에서는 다양한 객체와 다양한 텍스쳐 및 범프 맵을 사용하여 극 좌표계를 이용한 제안 방식과 직교 좌표계를 이용한 기존 방식 간의 화질 차이를 비교하였다. 그림 13은 나무 울타리 텍스쳐와 범프 맵을 평면에 매핑함으로써 생성된 결과이다. 그림 13(a)는 객체 공간에서의 직접적인 매핑을 사용하였고, 그림 13(b)는 참조 공간으로의 변환 후 매핑하는 방식을 사용하였다. 이들 영상을 비교해 보면, 육안으로 구별할 수 없을 정도로 영상의 질 차이가 없다. 그림 14는 벽돌 모양의 텍스쳐와 범프 맵을 정육면체 위에 매핑한 결과이다. 이

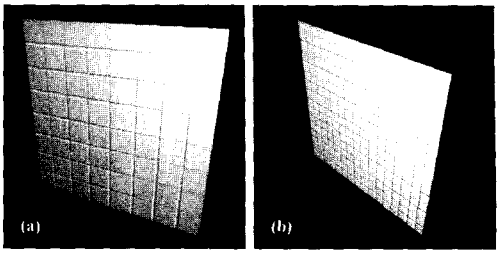

그림 11 격자 형태의 범프를 평면 위에 매핑하여 생성한 영상. (a) 해상도 256×256인 범프 맵을 사용한 경우 (b) 해상도 512×512인 범프 맵을 사용한 경우

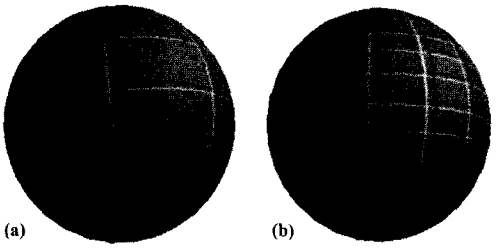

그림 12 격자 형태의 범프를 구의 표면 위에 매핑하여 생성한 영상. (a) 해상도 256×256인 범프 맵을 사용한 경우 (b) 해상도 512×512인 범프 맵을 사용한 경우

결과 역시, 평면의 경우 그림 13과 마찬가지로, 거의 영상의 질 차이가 없음을 알 수 있다. 그림 15는 세계 지도를 구에 매핑하는 실험을 통해 생성되었다. 이 실험에서는 해상도가 512×256인 세계 지도 텍스쳐/범프 맵을 해상도가 512×512인 구 객체에 매핑하였다. 이로 인해 전체적인 영상이 무딘 느낌을 주고 있으나, 두 방식 간의 영상의 질 차이를 거의 구별할 수 없다. 결과적으로, 위의 실험들을 통해, 제안 범프 매핑 방식이 적절한 영상 생성 과정을 수행함을 알 수 있다.

7. 결 론

본 논문에서는 하드웨어적으로 효과적인 범프 매핑에 대한 새로운 처리방식을 제안하였다. 제안 구조는 극 좌표계 표현 방식을 이용하여 참조 공간에서의 범프 매핑를 효과적으로 처리할 수 있도록 하였다. 벡터 회전을 이용한 제안 벡터 변환 방식은 벡터와 구 간의 기하학적 상관 관계를 통하여 참조 공간 위의 벡터를 구하는데, 이는 기존의 변환 행렬을 이용한 방법에 비해 상당량의 연산 하드웨어를 줄여준다. 또한 본 논문에서는 기존의 방식에서 사용했던 벡터 내적 계산에 필요한 연산을 줄임으로써 조명 계산을 효과적으로 처리할 수 있는

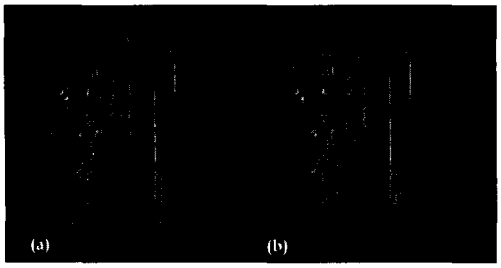

그림 13 나무 울타리 텍스쳐 및 범프 맵을 평면 위에 매핑하여 생성한 영상들. (a) 기존의 방식 (b) 제안 방식

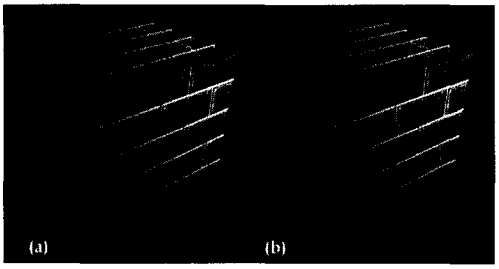

그림 14 벽돌 텍스쳐 및 범프 맵을 정육면체 위에 매핑하여 생성한 영상들. (a) 기존의 방식 (b) 제안 방식

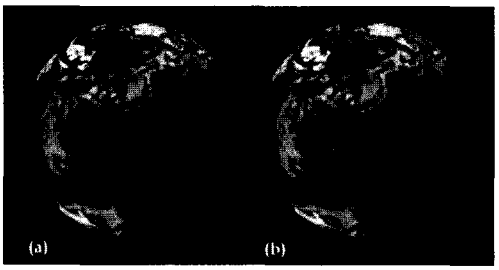

그림 15 세계 지도 텍스쳐 및 범프 맵을 구 위에 매핑하여 생성한 영상들. (a) 기존의 방식 (b) 제안 방식

방식을 제시하였다. 이는 cosine 법칙을 적용하여 벡터의 내적 계산에 필요한 곱셈 및 cosine의 계산량을 효과적으로 줄여준다. 결론적으로 제안 구조는 범프 매핑에서의 변환 및 조명 계산에 필요한 연산 및 하드웨어를 상당량 줄여준다.

본 연구 결과는 반사 매핑(reflection mapping)과 같이 각을 이용한 다른 렌더링 기법에도 적용이 가능할 것으로 생각된다. 본 논문에서 제안한 방법과 유사한 각 연산과 ROM 테이블을 사용할 경우, 기존의 직교 좌표계에 의한 계산 방법에 비해 하드웨어 비용을 상당량 줄여줄 수 있을 뿐만아니라, 제안 범프 매핑 방법과의 연동을 통해 보다 실감나는 영상을 생성할 수 있을 것이다. 향후, 제안 구조의 타 구조에의 적용 및 확장에 대한 연구를 진행할 예정이다.

참 고 문 헌

[1] A. Watt, *3D Computer Graphics*, 3rd Ed., Addison-Wesley Publishing Company, 2000.

[2] T. Möller and E. Haines, *Real-Time Rendering*, A K Peters, Ltd., 1999.

[3] NVIDIA Corp., *http://www.nvidia.com/view.asp?PAGE=products.*

[4] T. McReynolds, D. Blythe, B. Grantham, and S. Nelson, "Advanced Graphics Programming Techniques Using OpenGL," In Course Notes of *SIGGRAPH '98*, July 1998.

[5] M. Kilgard, "A Practical and Robust Bump-mapping Technique for Today's GPUs," NVIDIA Corporation, Apr. 2000.

[6] F. M. Weinhaus and V. Devarajan, "Texture Mapping 3D Models of Real-World Scenes," *ACM Computing Surveys*, Vol. 29, No. 4, pp. 325-365, Dec. 1997.

[7] P. Haeberli and M. Segal, "Texture Mapping as a Fundamental Drawing Primitive," In *Proceedings of 4th Eurographics Workshop on Rendering*, pp. 259-266, June 1993.

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

[9] G. Miller, M. Halstead, and M. Clifton, "On-the-Fly Texture Computation for Real-Time Surface Shading," *IEEE Computer Graphics and Applications*, Vol. 18, No. 2, pp. 44-58, Mar.-Apr. 1998.

[10] W. Heidrich and H. Seidel, "Realistic, Hardware-accelerated Shading and Lighting," In *Proceedings of SIGGRAPH '99*, pp. 171-178, Aug. 1999.

[11] J. F. Blinn, "Simulation of Wrinkled Surfaces," In *Proceedings of SIGGRAPH '78*, pp. 286-292, 1978.

[12] D. Jackèl and H. Rüsseler, "A Real Time Rendering System with Normal Vector Shading," In *Proceedings of the 9th Eurographics Workshop on Graphics Hardware*, pp. 48-57, 1994.

[13] K. Bennebroek, I. Ernst, H. Rüsseler, and O. Wittig, "Design Principles of Hardware-based Phong Shading and Bump Mapping," *Computers and Graphics*, Vol. 21, No. 2, pp. 143-149, 1997.

[14] T. Ikedo and J. Ma, "An Advanced Graphics Chip with Bump-mapped Phong Shading," In *Proceedings of IEEE Computer Graphics International '97*, pp. 156-165, 1997.

[15] T. Ikedo and J. Ma, "The Truga 001: A Scalable Rendering Processor," *IEEE Computer Graphics and Applications*, Vol. 18, No. 2, pp. 59-79, Mar. 1998.

[16] M. Peercy, J. Airey, and B. Cabral, "Efficient Bump Mapping Hardware," *Computer Graphics*,

Vol. 31, No. 4, pp. 303-306, 1997.
[17] J. S. Kim, J. H. Lee, and K. H. Park, "A fast and efficient bump mapping algorithm by angular perturbation," *Computers and Graphics*, Vol. 25, No. 5, pp. 401-407, 2001.
[18] T. Ikedo and E. Obuchi, "A Realtime Rough Surface Renderer," In *Proceedings of IEEE Computer Graphics International 2001*, pp. 355-358, 2001.
[19] A. Schilling, G. Knittel, and W. Straβer, "Texram: A Smart Memory for Texturing," *IEEE Computer Graphics and Applications*, Vol. 16, No. 3, pp. 32-41, May 1996.
[20] I. Ernst, H. Rüsseler, H. Schulz, and O. Wittig, "Gouraud Bump Mapping," In *Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pp. 47-53, Aug. 1998.
[21] I. Ernst, D. Jackèl, H. Rüsseler, and O. Wittig, "Hardware Supported Bump Mapping: A Step towards Higher Quality Real-Time Rendering," In *Proceedings of 10th Eurographics Workshop on Graphics Hardware*, pp. 63-70, Aug. 1995.
[22] A. Kugler, "IMEM: an intelligent memory for bump- and reflection-mapping," In *Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pp. 113-122, Aug. 1998.
[23] J. A. Piñeiro, J. D. Bruguera, "High-Speed Double-Precision Computation of Reciprocal, Division, Square Root and Inverse Square Root," IEEE Transactions on Computers, Vol. 51, No. 12, pp. 1377-1388, 2002.
[24] The Mesa 3D Graphics Library, http://www.mesa3d.org.

이 승 기
1997년 연세대학교 이과대학 수학과 졸업(학사). 2000년 연세대학교 대학원 컴퓨터과학과(공학석사). 2000년~현재 연세대학교 대학원 컴퓨터과학과 박사과정 관심분야는 3차원 그래픽 가속기, ASIC 설계, 고성능 컴퓨터 구조, 이미지 프로세싱, 내장형 시스템

박 우 찬
1993년 연세대학교 이과대학 전산과학과 졸업(학사). 1995년 연세대학교 대학원 전산과학과(이학석사). 2000년 연세대학교 공과대학 컴퓨터과학과(공학박사) 2001년~2003년 연세대학교 연구교수 2003년~현재 세종대학교 컴퓨터공학부 조교수. 관심분야는 3차원 그래픽 가속기, ASIC 설계, 병렬 렌더링, 고성능 컴퓨터 구조, Computer arithmetic

김 상 덕
2001년 연세대학교 이과대학 수학과 졸업(학사). 2003년 연세대학교 대학원 컴퓨터과학과(공학석사). 2003년~현재 연세대학교 대학원 컴퓨터과학과 박사과정 관심분야는 3차원 그래픽 가속기, ASIC 설계, Ray tracing

한 탁 돈
1978년 연세대학교 공과대학 전자공학과 졸업(학사). 1983년 Wayne State University 컴퓨터공학(공학석사). 1987년 University of Massachusetts 컴퓨터공학(공학박사). 1987년~1989년 Cleveland 주립대학 조교수. 1989년~현재 연세대학교 공과대학 컴퓨터과학과 교수. 관심분야는 Wearable computer, 3차원 그래픽 가속기, HCI, ASIC 설계, 고성능 컴퓨터 구조