제 26 회 한국정보처리학회 추계학술발표대회 논문집 제 13 권 제 2 호 (2006. 11)

각 연산을 이용한 픽셀 당 변위 매핑

이승기*, 이원종**, 한탁돈***
*삼성전자 정보통신총괄 통신연구소
**삼성종합기술원 컴퓨팅 테크놀로지 Lab.
***연세대학교 컴퓨터과학과
e-mail : sklee@yonsei.ac.kr

Per-Pixel Displacement Mapping Using Angular Operations

Seung-Gi Lee*, Won-Jong Lee**, Tack-Don Han***
*Samsung Electronics
**Samsung Advanced Institute of Technology
***Dept. of Computer Science, Yonsei University

요 약

본 논문에서는 극 좌표계에서의 벡터 표현 방식을 이용한 per-pixel displacement mapping 방법을 제시한다. Per-pixel displacement mapping 은 triangle mesh 의 처리 방식에 상관없이 변위매핑을 수행할 수 있도록 한 것으로, 2 차원 screen space 로 projection 된 triangle 의 각 pixel 의 위치를 객체 표면 정보에 따라 displacement 해주는 방법이다. 이는 기 검증된 범프매핑 하드웨어에 약간의 하드웨어를 추가함으로써 변위매핑을 수행할 수 있도록 한 효과적인 구조이다. 제안 방식에 의해 생성된 영상과 기존 방식에 의해 생성된 영상을 비교해본 결과, 시각적으로 거의 차이가 없음을 알 수 있다.

Keywords: displacement mapping, surface shading, polar coordinate system, graphics hardware

1. Introduction

Blinn[1]이 최초로 제안한 범프매핑(bump mapping) 기법을 이용하면 적은 양의 연산을 통해 상당한 정도의 실감을 느낄 수 있는 영상을 생성할 수 있다. 그러나, 범프매핑은 객체의 기하를 실제로 변형하지 않고 단지 normal vector 만을 변형하여 객체 표면의 질감을 표현하기 때문에 변형된 객체의 모서리 부분에 대한 silhouette 을 정확하게 처리할 수 없다.

Silhouette 을 정확하게 표현하기 위해서는 표면 기하의 원래 좌표가 새로운 좌표로 실제로 이동한 상태에서 쉐이딩되어야 한다. Cook[2]이 최초로 제시한, 변위매핑(displacement mapping)은 실제 기하의 변경을 통해 범프매핑이 갖고 있는 silhouette 문제를 해결하는 기법이다. 변위매핑을 이용하여 표면 기하를 변경하는 가장 명료한 방법은 표면을 표현하는 triangle mesh 의 정점을 변경하여 매 프레임마다 triangle mesh 를 다시 렌더링하는 방법이다. 이 방법은 증가된 triangle 의 정점을 projection 해야 하는 프로세서로써는 매우 많은 비용이 들 것이며, 단지 triangle mesh 의 정점에 대해서만 정확하게 처리될 것이다. 따라서, 객체 전 부분에 걸쳐 정확한 변위매핑을 처리하기 위해서는 객체의 보다 세밀한 부분의 기하를 변환하는 방식이 필요하다. 이의 대표적인 예로 Doggett 이 제안한 변위매핑 방식[3]을 들 수 있는데, 이는 pixel-level 에서 triangle mesh 의 기하(좌표)를 변형하기 위해 scan conversion hardware 를 이용하고 있다.

본 논문에서는 displacement vector map 을 이용한 per-pixel displacement mapping 방법을 제시한다. Per-pixel displacement mapping 은, 객체 표면의 세부 정보에 따라 triangle mesh 를 tessellation 해주어야 하는 per-vertex displacement mapping 과 달리, triangle mesh 의 처리 방식에 상관없이 수행될 수 있도록 한 것으로, 2 차원 screen space 로 projection 된 triangle 의 각 pixel 의 위치를 객체 표면 정보에 따라 displacement 해주는 방법이다. 이 방법은 Lee et al.[4]에서 제시하였던 범프매핑 방법의 vector rotation 알고리즘과 illumination calculation 알고리즘을 그대로 사용하였으며, 기존의 bump vector map 에 normal vector 의 크기를 가미한

displacement vector map 을 사용하여 해당 pixel 의 좌표를 변화시키는 데 이용하였다.

본 논문은 다음과 같이 구성된다. 2 절에서는 변위매핑과 관련된 연구에 대해 살펴본다. 3 절에서는 제안하는 per-pixel displacement mapping 방식에 대해 상세히 서술한다. 다음으로, 4 절에서는 제안 방식의 실험 환경 및 결과에 대해 논한다. 끝으로, 5 절에서는 결론을 맺는다.

2. Related Work

변위매핑은 software-based rendering 방식을 제시한 Cook 의 논문[2]에서 처음으로 언급되었다. 그 이후, ray tracing 을 이용하여 변위매핑을 수행하는 기법이 Pharr et al.[5]에 의해 제시되었다. 그러나 이들 방식은 최근에 나온 프로그래머블 그래픽스 하드웨어에서 구현된 방식에 비해 훨씬 복잡하다.

최근 몇 년 동안 변위매핑 처리를 위한 전용 하드웨어 구조에 대한 몇 가지 제안들이 있었다. Doggett et al [3]은 LOD(level of detail)-driven rasterization 방식을 제안하였는데, 이는 base mesh 에 새로운 정점을 삽입하여 객체 표면의 프리머티브(primitive)의 기하 구조를 변형한다. Gumhold et al[6]도 이와 유사한 방식을 제시하였다. Doggett and Hirche[7]는 변위 맵의 평균 변위에 따라 새로운 정점을 삽입하는 적응적 tessellation 기법을 제안하였다. Hirche and Ehlert[8]은 tessellation 의 결정에 필요한 계산을 줄여주기 위해 기 계산된 (pre-computed) 판단 기법을 사용하였다. 위에서 언급한 접근 방법들은 현존하는 그래픽스 하드웨어의 버텍스 쉐이더(vertex shader)에서 새로운 정점들의 생성이 필요하다. 또한 이들 방법들에서는 새로 생성된 정점들을 저장하기 위한 버텍스 어레이의 크기가 정확히 계산될 필요가 있다.

Matrox 에서는 버텍스 쉐이더 내의 변위 맵으로부터 변위를 가져오는 것을 허용한 Parhelia[9]를 소개하였는데, 이는 타사의 하드웨어에서는 이용할 수 없는 특성이다. 그리고, DirectX 9 에서와 같이, 변위 매핑을 위한 tessellation 은 버텍스 쉐이더 내에서 제어할 수 없는 pre set tessellation 레벨에서만 수행될 수 있다.

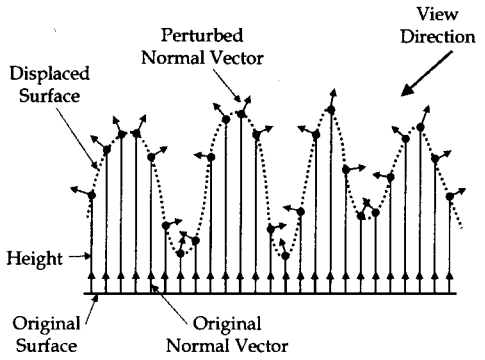
Kautz et al.[10]은 base mesh 를 돌출시키기 위해 볼륨렌더링(volume rendering)과 유사한 기법을 사용하였다. 이 방식은 볼륨의 모든 slice 가 가시성에 관계없이 렌더링되어야 하며, 높은 fill rate 와 높은 텍스처 대역폭을 요구한다는 문제점이 있다. Wang et al [11]은 미리 계산하여 테이블에 저장해놓은 가시 정보를 이용하여 변위 맵을 렌더링하는 기법을 제시하였다.

3. Proposed Per-Pixel Displacement Mapping

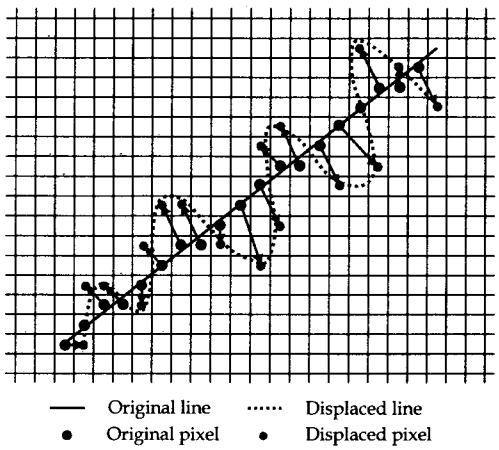
3.1 Concept of per-pixel displacement mapping

일반적으로 변위매핑은 geometry processing 에서 tessellation 과 displacement 의 과정을 거쳐 수행된다. 이와 같은 변위매핑 방식은 surface 의 triangle mesh 가 어떻게 tessellation 되느냐가 상당히 중요한 부분으로 인식되고 있다. 따라서 연구 초점이 tessellation 방법에 많이 치우쳐 있었다. 그러나 triangle mesh 가 다수의 triangle 로 tessellation 됨에 따라 geometry processing 에서 처리해야 할 triangle 의 수가 크게 증가되어 결과적으로 성능에 상당히 나쁜 영향을 미치게 되는 문제가 발생한다.

Per-pixel displacement mapping 은 pixel pipeline 에서 screen 표현의 최소 단위인 pixel 의 위치와 normal vector 를 변형하기 때문에 pixel 의 수가 증가하지 않는다. 이 방식에서는 triangle 을 어떻게 tessellation 하느냐 보다는 어떤 displacement map 을 사용하느냐에 따라 변위매핑의 성패가 좌우된다.



(그림 1) 2 차원 객체공간에서의 Displacement mapping.



(그림 2) 2 차원 스크린공간에서의 position displacement.

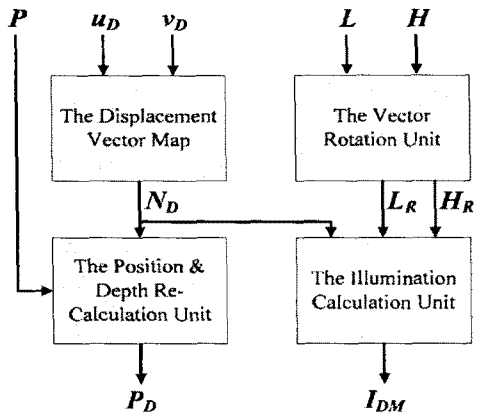
Surface displacement 는 original surface 에 displacement 를 적용하여 새로운 surface 를 생성하는 것을 의미하며, 이 과정에서 새로 생성된 surface 위의 point 들 간에는 object space 상에서 view source 및 light source 와의 visibility 문제가 발생한다. 그림 1 은 object space 에서의 surface displacement 를 보여준다. 이를 view direction 에 의해 screen space 에 projection 하면 그림 2 와 같이 pixel 이 위치하게 된다. Per-pixel displacement mapping 에서는 screen space 상의 pixel 의 위치와 normal 을 동시에 바꿔줌으로써 screen 좌표의 새로운 pixel 을 생성하게 된다. 이는 displacement map

을 direction (angles)과 height (magnitude)를 갖는 displaced (perturbed) normal vector 로 구성함으로써 쉽게 처리될 수 있다.

3.2 Processing flow of per-pixel displacement mapping

Per-pixel displacement mapping 은 object surface 의 displacement information 을 이용하여 2 차원 screen space 로 projection 된 triangle 의 각 pixel 의 position 과 depth 를 재계산하여 옮겨주고, 그 위치에서의 color 값을 계산해주는 방법이다.

그림 3 은 제안하는 per-pixel displacement mapping 수행 구조를 보여주는데, The Displacement Vector Map (DVM), The Vector Rotation Unit (VRU), The Position & Depth Re-Calculation Unit (PDU), 그리고 The Illumination Calculation Unit (ICU)로 구성된다. DVM 은 Chapter 4 에서 사용했던 bump vector map 에 displaced normal vector 의 크기 d 를 가미한 data structure 를 갖는다. VRU 와 ICU 에서는 Lee et al.[4]에서 제시한 방법을 변형없이 그대로 사용한다.



(그림 3) 제안하는 per-pixel displacement mapping.

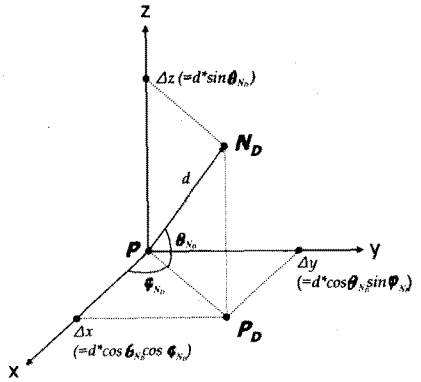
Per-pixel displacement mapping 는 pixel 의 position $P(x, y, z)$, light vector L, halfway vector H, 그리고 displacement coordinate (u_D, v_D)를 입력으로 받는다. Displacement coordinate 에 의해 DVM 으로부터 가져온 displaced normal vector N_D 를 PDU 에 입력하여 screen space 의 다른 좌표로 이동될 pixel P_D 의 position 과 depth 를 계산하여 Z-buffer 에 저장하고, VRU 에 의해 transformed vector L_R, H_R 과 displaced normal vector N_D 를 ICU 에 입력하여 pixel 의 color 값을 계산하여 Frame Buffer 에 저장한다. Frame Buffer 에 저장된 color 값은 이후 shadow calculation 에 이용된다. 이 과정은 모든 pixel 이 처리될 때까지 반복하여 수행된다.

3.3 Geometry of position/depth re-calculation

그림 4 는 DVM 으로부터 가져온 N_D 의 geometry 를 Cartesian coordinate system 에서 표현한 것이다. Displaced normal vector N_D 의 x, y, z 좌표는 displacement factor 로 사용되는데, 이의 계산은 다음과 같다.

$$\Delta x = d^*\cos\theta_{N_D}\ \cos\varphi_{N_D} \quad \text{(식 1)}$$
$$\Delta y = d^*\cos\theta_{N_D}\ \sin\varphi_{N_D} \quad \text{(식 2)}$$
$$\Delta z = d^*\sin\theta_{N_D} \quad \text{(식 3)}$$

여기서 d, φ_{N_D}, θ_{N_D} 는 N_D 의 DVM 에서의 polar coordinate system 의 표현 방식이다.

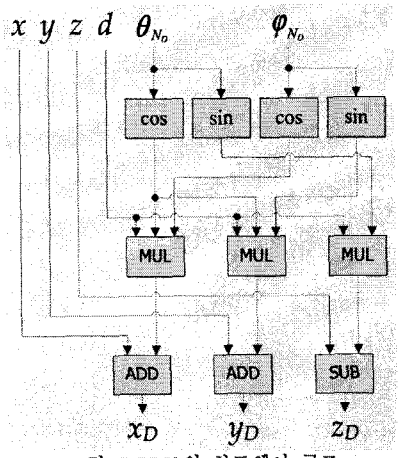


(그림 4) Position/depth re-calculation 을 위한 기하정보

Screen space 상의 pixel P_D 의 position (x_D, y_D)와 depth z_D 는 displaced normal vector N_D 의 x, y, z 좌표를 이용하여 다음과 같이 계산된다.

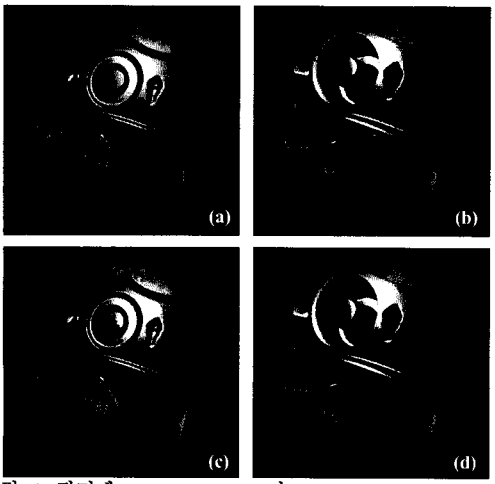
$$x_D = x + \Delta x = x + d^*\cos\theta_{N_D}\ \cos\varphi_{N_D} \quad \text{(식 4)}$$
$$y_D = y + \Delta y = y + d^*\cos\theta_{N_D}\ \sin\varphi_{N_D} \quad \text{(식 5)}$$
$$z_D = z - \Delta z = z - d^*\sin\theta_{N_D} \quad \text{(식 6)}$$

3.4 Position and depth re-calculation unit

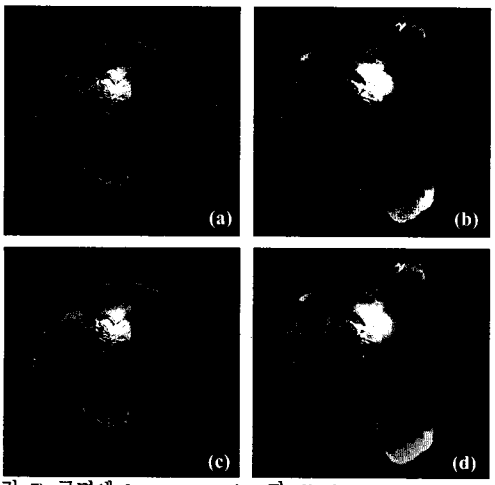


(그림 5) PDU 의 하드웨어 구조.

그림 5 는 Position & Depth Re-Calculation Unit (PDU)의 하드웨어 구조를 보여준다. 여기서 계산된 depth z_D 는 Z-buffer 의 position (x_D, y_D)에 저장된다.



(그림 6) 평면에 bump mapping 과 displacement mapping 을 적용한 장면. (a) 기존의 bump mapping 방식 (b) 기존의 displacement mapping 방식 (c) Lee 의 제안 bump mapping 방식 (d) 제안 displacement mapping 방식



(그림 7) 구면에 bump mapping 과 displacement mapping 을 적용한 장면. (a) 기존의 bump mapping 방식 (b) 기존의 displacement mapping 방식 (c) Lee 의 제안 bump mapping 방식 (d) 제안 displacement mapping 방식

4. Experimental Environment and Results

본 논문에서 제시한 알고리즘과 하드웨어 아키텍처는 C 언어에 의한 시뮬레이션을 통해 검증되었다. 이는 OpenGL 과 유사한 3 차원 그래픽스 라이브러리인 Mesa 6.x[12]를 기본하고 있다. 본 시뮬레이션의 수행을 위해, Mesa 6.x 을 수정하여 기존 변위매핑과 제안 변위매핑을 구현하였다. 그림 6 과 그림 7 은 시뮬레이터를 통해 생성된 영상으로, 각각 평면과 구면에 범프 및 변위 매핑한 것이다. 여기서 (c)는 Lee et al.에서 제안하였던 방식에 의해 생성된 범프매핑된 영상이다. 그림 6(b)와 그림 6(d)를 비교해보면, 두 영상의 시각적 차이가 거의 없음을 알 수 있다. 그림 7(b)와 그림 7(d)의 비교 결과 또한 마찬가지다. 결론적으로 시뮬레이션 결과를 통해서 제안 변위매핑 방법이 제대로 수행되고 있음을 알 수 있다.

5. Conclusions

본 논문에서는 극 좌표계의 벡터표현 방식을 이용한 per-pixel displacement mapping 방법을 제시하였다. 제안 per-pixel displacement mapping 은 triangle mesh 의 처리 방식에 상관없이 수행될 수 있도록 한 것으로, Lee et al.[3]에서 제시하였던 기 검증된 범프매핑 하드웨어에 약간의 로직을 추가함으로써 변위매핑을 수행할 수 있도록 한 효과적인 구조이다. 이는 기존의 bump vector map 에 normal vector 의 크기를 가미한 displacement vector map 을 사용하여 해당 pixel 의 좌표를 변환하는데 이용하였다. 본 제안 방식에 의해 생성된 영상을 기존 방식에 의해 생성된 영상과 비교해 본 결과, 시각적으로 거의 차이가 없음을 알 수 있다.

참고문헌

[1] James F. Blinn, "Simulation of Wrinkled Surfaces," Computer Graphics (Proceedings of SIGGRAPH 78), 12(3), pp.286-292, August 1978.

[2] Robert L. Cook, "Shade Trees," *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3), pp.223-231, July 1984, Minneapolis, Minnesota.

[3] Michael Doggett, Anders Kugler, and Wolfgang Straßer, "Displacement Mapping using Scan Conversion Hardware Architectures," *Computer Graphics Forum*, 20(1) pp.13-26, March 2001.

[4] S.G.Lee, W.C.Park, W.J.Lee,S.B.Yang, and T.D.Han, "An Effective Hardware Architecture for Bump Mapping using Angular Operations," In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pp.68-75, August 2003.

[5] Matt Pharr and Pat Hanrahan, "Geometry caching for raytracing displacement maps," In *Eurographics Workshop on Rendering*, June 1996.

[6] Stefan Gumhold and Tobias Hüttner, "Multiresolution rendering with displacement mapping," In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pp.55-66, August 1999.

[7] Michael Doggett and Johannes Hirche, "Adaptive view dependent tessellation of displacement maps," In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pp.59-66, August 2000.

[8] Johannes Hirche and Alexander Ehlert, "Curvature-driven sampling of displacement maps," presented at ACM,2002) SIGGRAPH 2002 as a Technical Sketch, July 2002.

[9] Matrox, http://www.matrox.com/mga/products/parhelia 512/technology/disp_map.cfm.

[10] Jan Kautz and Hans-Peter Seidel, "Hardware accelerated displacement mapping for image based rendering," In *Graphics Interface*, pp.61–70, June 2001.

[11] Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heun-Yeung Shum, "View-dependent displacement mapping," In *ACM Transactions on Graphics*, volume 22, pp.334–339. ACM, 2003.

[12] The Mesa 3D Graphics Library, http://www.mesa3d.org.