이미지 피라미드를 이용한 변위 맵의 실시간 렌더링

(Real-Time Rendering of a Displacement Map using an Image Pyramid)

오 경 수 † 기 현 우 ††
(Kyoungsu Oh) (Hyunwoo Ki)

요 약 역변위 매핑은 기하정보를 증가시키지 않고, 모델에 상세함을 더하는데 사용된다. 우리는 부드럽고 정확한 굴곡을 표현할 수 있는 GPU 기반의 실시간 역변위 매핑 기법을 제안한다. 이를 위하여, 렌더될 각 픽셀에서 광선을 만들고 이를 전진시켜 나가며 변위 맵과의 교차점을 찾는다. 광선 추적을 안전하고 효율적으로 수행하기 위하여, 변위 맵을 쿼드트리 형태의 이미지 피라미드로 만들고, 이 트리를 하향식으로 탐색하며 전진해 나간다. 나아가, 변위 맵이 화면에서 확대되었을 때 선형 보간을, 그리고 화면에서 멀어져 작게 보일때는 때는 밉맵 필터링을 통해 화질을 향상시키고 렌더링을 가속화한다.

실험을 통해, 기존의 GPU 기반의 기법들과는 달리 날카로운 변위 맵에 대해 예각에서도 깨끗한 이미지를 생성하는 것을 확인하였다. 초당 수 백 프레임의 빠른 속도로 렌더링할 수 있었으며, 변위 맵의 해상도가 커져도 렌더링 속도의 저하가 적었다. 우리의 기법은 구현이 간단하고 수행속도가 빠르기 때문에 현존하는 게임이나 가상 현실 시스템 등에 쉽게 적용할 수 있다.

키워드 : 역변위매핑, 실시간렌더링, 영상기반렌더링, GPU, 쿼드트리

Abstract Inverse displacement mapping enables us to add realistic details to polygonal meshes without changing geometry. We present a real-time artifacts-free inverse displacement mapping method. In each pixel, we construct a ray and trace the ray through the displacement map to find an intersection. To skip empty regions safely, we traverse the image pyramid of displacement map in top-down order. Furthermore, when the displacement map is enlarged, intersection with bilinear interpolated displacement map can be found. When the displacement map is at distance, our method supports mipmap-like prefiltering to enhance image quality and speed.

Experimental results show that our method can produce correct images even at grazing view angles. Rendering speed of a test scene is over hundreds of frames per second and the influence of resolution of displacement map to rendering speed is little. Our method is simple enough to be added to existing virtual reality systems easily.

Key words : Inverse Displacement Mapping, Real-time Rendering, Image-based Rendering, GPU, Quad-tree

1. 서 론

텍스처 매핑은 가상현실이나 게임에서 다각형 모델에 이미지를 입혀서 기하 정보를 증가시키지 않고도 복잡한 외양을 표현할 수 있는 중요한 기법이다. 텍스처 매핑 기법 중 하나인 범프 매핑(bump mapping)[1]은 높이 정보를 다각형에 입혀 조명값에 변화를 줌으로써 표면의 상세도를 높인다(그림 3). 변위 매핑(displacement mapping)[2]은 다각형에 입혀진 높이값을 가지고 실제로 기하를 이동시키는 기법이다. 여기서 변위(displacement)는 물체 표면의 고도에 따른 높이차이를 의미한다(그림 2). 반면에, 역변위 매핑(inverse displacement mapping)[3]은 기하 정보의 변경없이 변위 맵(displace-

· 그림 1에서 사용된 얼굴 데이타를 제공해주신 [18]의 저자와 논문의 완성도 증대에 도움을 주신 익명의 심사위원분들께 감사드립니다. 본 연구는 학술진흥재단 중점연구소(KRF-2004-005-D00198) 지원으로 수행되었습니다.

† 종신회원 : 숭실대학교 미디어학과 교수
oks@ssu.ac.kr
†† 학생회원 : 숭실대학교 미디어학과
kih@ssu.ac.kr

논문접수 : 2006년 8월 22일
심사완료 : 2007년 4월 11일

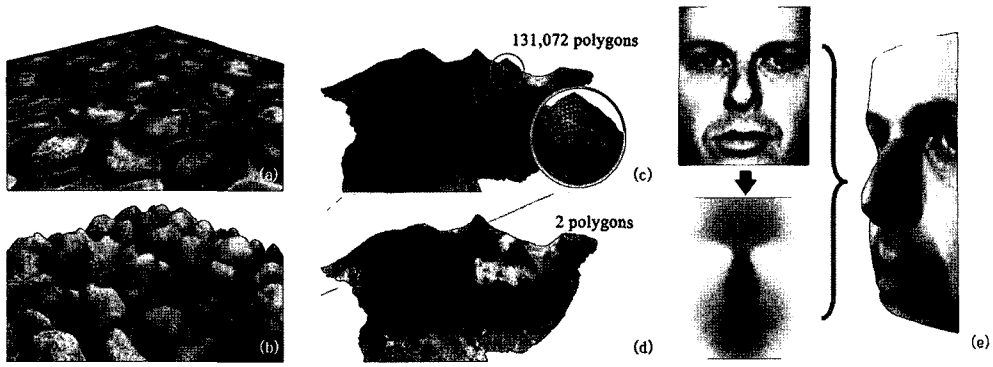

그림 1 제안된 기법(b, d, e)은 범프 매핑 (a)처럼 모델에 상세도를 더하거나, 높이 매핑(height mapping; c)과 같이 지형 렌더링에 활용할 수 있다. 또한, 영상 기반 렌더링 관점에서, 촬영하여 얻은 2D 영상으로부터 다른 시점에서 바라본 입체 영상을 생성할 수 있다(e).

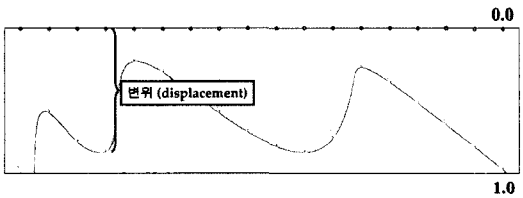

그림 2 변위 맵의 각 픽셀에는 물체 표면의 고도에 따른 높이차이(변위, displacement)를 저장한다.

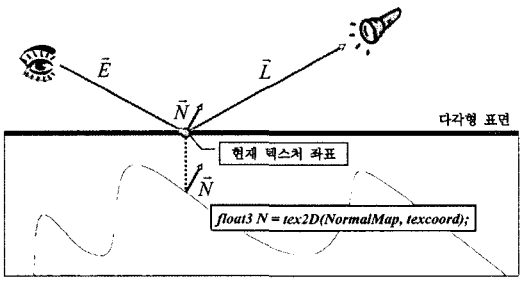

그림 3 범프 매핑은 현재 텍스처 좌표에서의 변위에 따른 법선(normal) 값을 사용하여 조명을 계산한다.

ment map)에 저장된 높이값들과 각 텍셀에서 출발한 광선과의 교차점을 구해서 텍셀의 위치를 이동함으로써 표면 상세도를 변화시키는 방법이다(그림 4).

최근에 등장한 GPU 기반의 역변위 매핑 알고리즘으로는 시차 폐색 매핑(parallax occlusion mapping)[10], 릴리프 매핑(relief mapping)[11] 등이 있다. 이들은 교차 검사를 위해 선형 탐색(linear search) 또는 이진 탐색(binary search)을 수행하는데, 이는 심각한 시각적 오류를 유발하였다(그림 6,19,22).

이 문제를 해결하기 위하여, 곡률에 따라 샘플링율을 변화시키는 기법이 있지만[12], 매우 날카로운 변위 맵에 대해서는 완전한 해결책이 되지 못한다. 광선이 교차

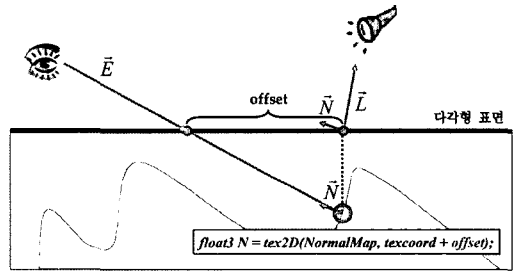

그림 4 역변위 매핑은 눈에서 출발한 광선과 변위 맵과의 교차점을 찾고, 이 교차점에서의 법선값을 사용하여 조명을 계산한다.

점을 놓치지 않고 이동할 수 있는 안전 거리(safety distance)를 전처리 과정에서 구하는 기법[13,14]도 있었다. 하지만 이러한 전처리는 매우 긴 시간을 필요로 한다.

역변위 매핑은 면에 상세도를 더하는 것 뿐만 아니라 변위 맵으로 표현된 지형을 렌더링하는 데에도 사용될 수 있다. 변위 맵을 렌더링하기 위한 광선 추적법은 주로 CPU기반이었다[4-7]. CPU 기반의 기법들은 렌더링에 많은 시간을 소요한다. 변위 맵에 대한 이미지 피라미드를 사용하는 [4]는 변위 맵을 쿼드트리 형태로 계층화하고, 점진적인 알고리즘(incremental algorithm)[8,9] 형태의 광선 추적을 사용하여 속도를 가속화한다. 하지만, 다른 CPU 기반의 기법들과 마찬가지로 실시간 처리가 어려우며, 변위 맵이 확대되는 경우 다중 샘플링(super sampling)을 사용하는 한계를 보였다.

우리는 [4]와 유사하게, 피라미드 변위 맵을 사용한 광선 추적법을 통해 실시간에 변위 맵을 렌더링하는 역변위 매핑 기법을 제안한다. 하지만 우리는 실시간 렌더링을 위하여 알고리즘을 GPU에 친화적인 상태로 구현하고, 맵이 확대(magnification)되는 경우와 축소(mini-

fication)되는 경우에 대해서도 정확하고 효율적으로 렌더링하도록 개선한다. 피라미드 변위 맵은 가장 상세한 변위 맵을 최하위 레벨로 시작해서 각 레벨별 이미지의 픽셀값으로, 하위 이미지의 4개 픽셀(쿼드트리의 자식 노드)의 최소값(가장 작은 변위)을 저장한 이미지 피라미드이다. 렌더링할 때, 각 픽셀에서의 텍스처 좌표를 시작점으로 하고 시선 방향과 일치하는 광선을 생성한다. 시작점에서 출발하여 광선의 깊이값이 저장된 변위값보다 작아지는 지점까지 추적한다. 피라미드 변위 맵을 GPU만을 사용하여 하향식으로 계층적으로 탐색해 나가며 점진적인 알고리즘으로 광선과 변위 맵의 정확한 교차점을 빠르게 찾는다. 그림 1, 5, 19와 22는 렌더링 결과를 기존의 기법들과 비교한 것으로, 우리의 기법이 보다 정확하고 인상적인 이미지를 생성할 수 있음을 보여준다.

우리의 기법은 모든 형태의 변위 맵에 대하여, 모든 시점 각도에서 교차점을 정확하게 찾을 수 있다. 또한, 화질과 속도가 변위 맵의 해상도에 큰 영향을 받지 않는다. 또한, 변위 맵이 화면에서 확대되었을 때 선형 보간을, 그리고 화면에서 작게 보일 때는 LOD(상세 레벨, level of detail) 효과를 주어 화질을 향상시키고 렌더링을 가속화한다. 영상 기반 렌더링 관점에서, 촬영한 2차원 영상으로부터 다른 시점에서 바라본 입체 영상을 생성하여 게임, 가상현실 등의 실시간 시스템에 적용할 수 있다. 구현이 간단하여, 많은 개발자들은 이 기법을 쉽게 습득하고 그들의 시스템에 적용하여 적은 자원으로 보다 사실적이고 정확하게 가상 세계를 표현할 수 있을 것이다.

2. 관련 연구

Cohen은 CPU 상에서 피라미드 자료 구조와, 딧셈과

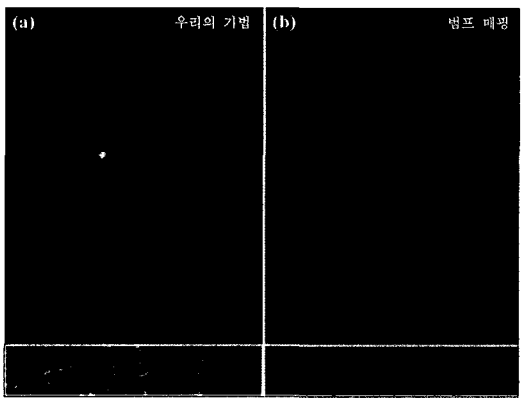

그림 5 우리의 기법은 표면의 상세도를 높여서 가상 세계의 사실감을 높일 수 있다.

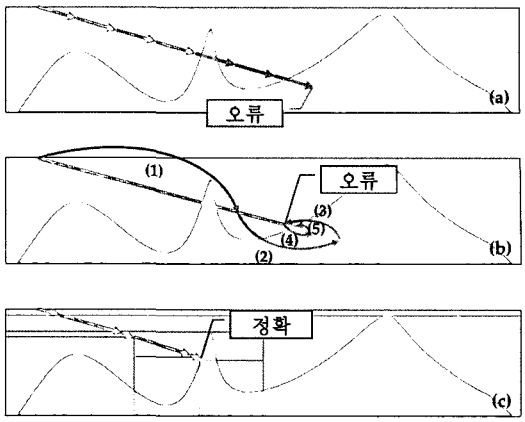

그림 6 선형 탐색은 광선을 일정 간격으로 전진시키며 교차점을 찾는다(a). 이진 탐색은 현재 위치와 끝 점의 중간 위치로 광선을 이동시켜 나간다(b). 이 기법들은 종종 잘못된 교차점을 찾는다. 계층적 탐색은 교차점을 지나치지 않는 범위 내에서 높은 곳에서부터 낮은 곳으로 내려가며 광선을 이동시킨다(c).

시프트 연산만을 사용하는 midpoint 래스터라이즈 기법을 사용하여 지형 렌더링의 속도를 가속화하였다[4]. 피라미드의 각 픽셀에는 가장 높은 높이값을 저장하고, 광선을 하향식으로 탐색해 나간다. 이는 다른 CPU 기반의 기법들[5-7] 보다 향상된 성능을 보이지만 여전히 실시간 동작이 어렵다. 뿐만 아니라 보간된 부드러운 굴곡의 표현이 어렵다.

반면에 GPU를 사용하는 변위 맵 렌더링 기법들은 매우 높은 초당 프레임율로 렌더링이 가능하다. 시차 매핑(parallax mapping)[15]은 범프 매핑과는 달리 *시차(parallax) 효과*를 통해 보다 나은 굴곡을 표현한다. 이 기법은 적은 비용으로 렌더링할 수 있지만, 불규칙한 변위 맵에 대해서는 부정확한 결과를 낳는다.

시차 폐색 매핑[10]은 같은 간격으로 광선을 전진하며 광선과 변위 맵의 교차점을 근사한다. 하지만 이러한 선형 탐색은 날카로운 물체와 예각에서 매우 잘못된 결과를 발생시킨다(그림 6(a)와 19). 릴리프 매핑[11]은 선형 탐색 이후에 이진 탐색을 추가로 수행하여 하여 더 정확하게 교차점을 근사한다. 하지만 이진 탐색은 텍스처 정밀도에 제한이 있으며, 시점이 예각이거나 날카로운 물체에서 심각한 오류를 발생시킨다(그림 6(b)). 동적인 시차 폐색 매핑[12]은 시점 각도나 지형의 주파수(날카로운 정도)에 따라 샘플링율을 변화시켜 이러한 오류를 완화시킨다. 하지만 매우 날카로운 변위 맵이나 예각에서는 여전히 해결책이 되지 못하며, 더 많은 샘플을 사용할 경우 속도가 현저하게 감소되는 단점을 보인다.

Donnelly[13]는 교차점을 지나치지 않는 안전 거리를 미리 계산하여 저장한 3차원 텍스처를 사용하여 정확한 굴곡을 표현하였지만, 3차원 텍스처는 메모리 소비가 크며 속도를 저하시킨다. Baboud[14]는 이와 유사하지만 2차원 텍스처에 안전 거리를 저장하는 방법을 사용하였다. 이들의 기법은 실시간 렌더링이 가능하지만, 전처리가 매우 오랜 시간을 소요하여, 완전한 상호 작용 시스템에는 적합하지 않다.

3. 피라미드 변위 매핑 알고리즘

우리의 기법은 쿼드트리 구조의 피라미드 변위 맵을 GPU 기반으로 탐색하여, 변위 맵을 빠르고 정확하게 렌더링할 수 있게 한다. 정확하고 강건한 광선과 변위 맵의 교차 검사 알고리즘을 사용하여, 기존의 기법들[10-12,15]에서 발생하는 심각한 문제를 해결한다.

이미지 피라미드는 $2^n \times 2^n$의 픽셀을 가진 0 레벨부터, $2^0 \times 2^0$개의 픽셀을 가진 n 레벨까지의 계층적인 이미지들의 집합을 의미한다. 하위 레벨 텍스처의 각 픽셀(i, j)에는, 4개의 픽셀들($2i$, $2j$), ($2i$, $2j$+1), ($2i$+1, $2j$), ($2i$+1, $2j$+1) 중 가장 작은 변위값을 저장한다(그림 6). 이렇게 만들어진 쿼드트리를 루트(root)부터 리프(leaf)까지 계층적으로 탐색해 나간다.

[4]는 CPU 상에서 덧셈과 시프트 연산만을 사용하는 midpoint 래스터라이즈 기법을 사용하여 광선을 전진시켜 나간다. 이러한 방식은 GPU 프로그래밍에서는 사용할 수 없다. 따라서, 우리는 기존의 GPU 기반의 시차 폐색 매핑[10], 릴리프 매핑[11] 등의 기법에서와 유사하게, 읽어온 변위에 해당하는 높이까지 광선을 직선으로 직접 이동시킨다. 노드의 경계를 지나칠 때에는 "노드 넘어가기"라는 특수한 방법을 사용하며, 이는 3.2절에서 자세히 다룬다.

먼저, 탄젠트 공간에서의 시점 벡터, E를 계산한다. 광선의 시작점, P의 x, y값은 현재 텍스처 좌표가 되며, z값은 0이 된다. 피라미드 텍스처의 평면에 투영된 광선의 z 값이, 광선의 x, y를 텍스처 좌표로 사용하여 현재 레벨의 피라미드 변위 맵에서 읽어온 z값보다 크면, 광선을 읽어온 z에 해당하는 만큼 전진시키고(P'), 다음 레벨로 내려간다; 그렇지 않으면 전진없이 다음 레벨로 내려가기만 한다. 이 작업을 리프 노드에 이를때까지 반복한다(그림 8). 이렇게 계산된 광선의 최종 위치에서의 x, y 좌표를 사용하여 텍스처에서 읽어온 법선, 색 정보 등을 사용하여 조명을 계산한다.

이러한 피라미드 변위 맵을 사용한 계층적 탐색은 기존의 기법들에서 표현 가능한 완만한 변위 맵 뿐만 아니라, 날카로운 변위 맵을 예각에서 보았을 때도 정확하게 수행될 수 있다. 또한, 변위 맵의 해상도가 증가함에 따른 렌더링 속도의 저하 정도가 작다(O(logN), N은 변위 맵의 가로 또는 세로 해상도).

3.1 피라미드 변위 맵

먼저, 이미지 피라미드를 생성해야한다. 밉맵과 유사한 형태로 상향식으로 각 하위 레벨 텍스처의 픽셀의 값을 결정하되, 하위 레벨의 4개의 픽셀(사분면) 중 가장 작은 변위값(즉, 가장 높은 높이값)을 저장한다(그림 7). 중간 노드 픽셀에는 지역적으로 가장 작은 변위값이 저장되며, 루트 노드 픽셀에는 전역적으로 가장 작은 변위값이 저장된다. 이러한 피라미드 변위 맵의 형태는 [4]와 동일하다.

피라미드 변위 맵은 CPU 상에서 생성하거나, 픽셀 쉐이더를 사용하여 GPU 상에서 생성할 수 있다. PCI Express의 그래픽 카드에서는 CPU 상에서도 매우 빠르게(256×256 해상도의 변위 맵에서 약 2.3ms) 생성할 수 있다. 이러한 피라미드 변위 맵의 생성은 변위 맵이 변경될 때 단 한 번만 수행한다.

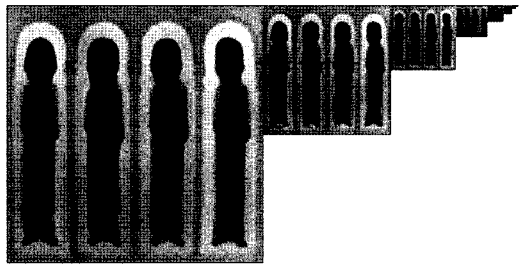

그림 7 사분면 픽셀 (자식 노드) 중 가장 작은 변위값을 저장하는 피라미드 변위 맵의 예

3.2 광선과 변위 맵의 교차점 찾기

각 픽셀에 대하여, 정점의 텍스처 좌표로부터 보간된 텍스처 좌표를 출발점으로 하여, 쿼트트리 구조의 피라미드 변위 맵을 사용하여 지형과의 교차점을 계산한다. 루트부터 리프를 향하여 계층적으로 탐색해 나간다(그림 8과 10). 먼저, 시선 벡터를 탄젠트 공간으로 변환해서 광선의 방향 벡터를 구한 뒤, 다음의 과정을 통해 시선 방향으로 나아가는 광선과 변위 맵과의 교차점을 찾는다.

먼저, 현재 텍스처 좌표, P에서 피라미드 변위 맵의 루트 레벨의 변위값, d를 읽어온 뒤, 광선의 현재 위치를 d의 변위에 해당하는 위치로 전진시킨다(그림 8(a)). 다음으로, 전진된 광선의 위치, P'에서 피라미드 변위 맵의 다음 레벨의 변위값, d'을 읽어온다. 만일, d'이 d보다 크면, 광선을 d'의 변위에 해당하는 위치로 전진시킨다. 그렇지 않은 경우에는 광선을 이동시키지 않고 다음 레벨로 진행한다(그림 8(b)). 이 과정을 리프 레벨에 이를 때까지 반복한다. 리프 노드에서의 광선의 위치는

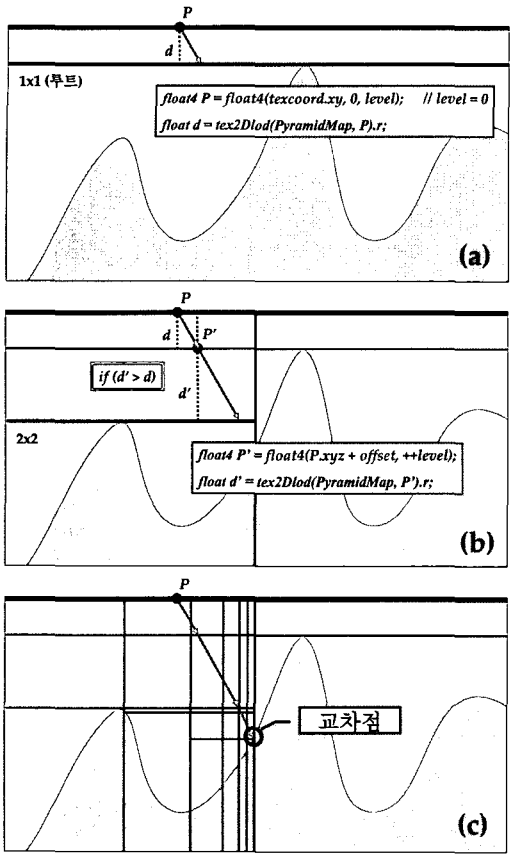

그림 8 피라미드 변위 맵을 사용한 광선 추적 알고리즘: 광선이 현재 노드 내부에서 전진할 경우. 현재 레벨의 현재 노드에서 읽어온 변위 에 해당하는 높이까지 광선을 이동시킨다(a). 이 작업을 리프 노드에 이를 때까지 수행하며, 리프 노드에서의 광선의 위치가 올바른 교차점이 된다(b와 c).

광선과 변위 맵의 정확한 교차점이 된다(그림 8(c)).

피라미드 변위 맵의 각 픽셀에는 픽셀이 차지하고 있는 표면의 가장 높은 값을 저장하고 있다. 따라서, 이 값을 사용하여 높은 곳에서부터 순차적으로 내려가면, 교차점을 지나쳐서 더 낮은 곳으로 내려가는 문제를 막을 수 있다.

불행하게도, 복잡하고 날카로운 변위 맵에서 또는 예각에서 바라보았을 때, 앞서 설명한 탐색법은 교차점을 놓칠 수 있다(그림 9(a)). 이러한 문제를 피하기 위하여, 전진한 광선의 위치가 전진하기 전과 같은 노드에 있는지 여부를 검사한다. 만일, 현재 노드를 넘어간다면, 광선을 전진 시키지 않고, 광선 방향으로의 노드의 경계에서 임의의 작은 값, δ만큼 이동한 위치로 광선을 전진시킨다(그림 10(b)). 여기서 δ는 현재 하위 레벨 텍스처의

```
level = ROOT_LEVEL;
d = tex2dlod(PyramidMap, float4(P.xyz, level));
P' = P + E * d;
level--;

while (level > 0)
{
    d = tex2Dlod(PyramidMap, float4(P'.xyz, level));

    if (P'.z < d)
        P' = P + E * d;

    level--;
}
```

그림 9 피라미드 변위 맵을 사용한 광선 추적 의사코드 I: 광선이 현재 노드 내부로 전진할 경우

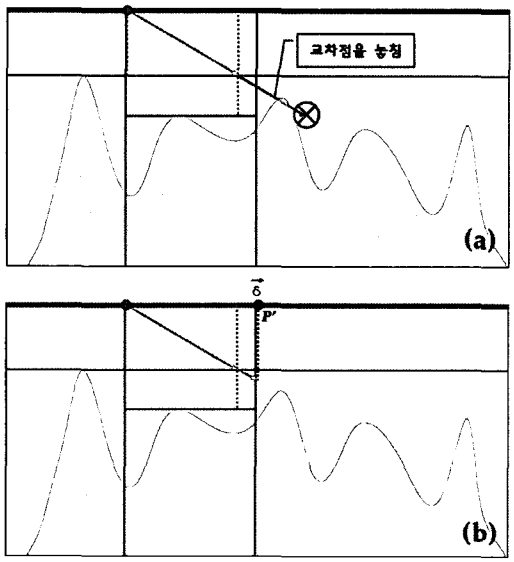

그림 10 노드 넘어가기: 전진한 광선이 현재 노드를 넘어가면 교차점을 놓칠 수 있다(a). 이에 따라, 우리는 광선이 현재 노드를 넘어갈 경우, 전진시키지 않고 노드의 경계에서 임의의 작은 값, δ만큼 전진한 위치로 이동시킨다(b). 이후, 이동한 위치에서 다시 탐색을 수행한다.

크기에 따라 적절한 값으로 설정한다(0.01 / 2^n, n은 레벨). 우리는 이 과정을 "노드 넘어가기(node crossing)"라고 부른다. 노드 넘어가기를 수행한 뒤에는 다음 레벨로 진행하지 않는다. 따라서, 같은 레벨에서 다시 탐색을 수행한다.

노드 넘어가기를 수행한 광선은 두 가지 상황으로 나뉜다. 첫째로, 넘어간 곳의 지형이 높아서 광선의 높이와 일치하거나 더 높은 경우로, 이러한 경우는 그림 11과 같이 광선이 벽과 교차한다. 이러한 광선은 이후에 더 이상 전진을 하지 않게 되므로, 결국 정확한 교차점

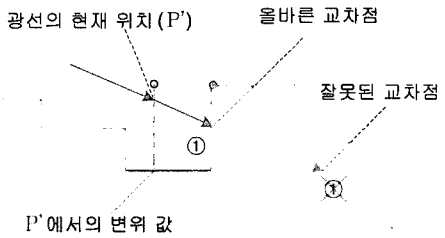

그림 11 노드 넘어가기는 정확하게 광선과 벽과의 교차점을 찾을 수 있게 한다.

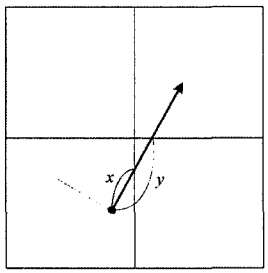

그림 12 x와 y 축의 노드 경계를 모두 넘어갈 경우, 경계까지의 거리가 짧은 쪽(x 축)을 선택한다.

```
level = ROOT_LEVEL;
d = tex2Dlod(PyramidMap, float4(P.xyz, level));
P' = P + E * d;
level--;

while (level > 0)
{
    d = tex2Dlod(PyramidMap, float4(P.xyz, level));

    if (P'.z < d)
    {
        P_tmp = P + E * d;

        // 두 광선이 같은 노드에 있는지 검사한다
        if (IsWithinSameNode(P_tmp, P))
        {
            P' = P_tmp;
            level--;
        }
        else
        {
            // 광선이 노드의 경계를 살짝 넘어가도록한다.
            P' = CrossNode(P', E, level);
        }
    }
    else
        level--;
}
```

그림 13 피라미드 변위 맵을 사용한 광선 추적 알고리즘 II: 노드 넘어가기 포함

을 찾게 된다. 둘째로, 지형이 광선보다 낮은 경우이다. 이 경우는 탐색을 계속 진행하면 광선이 변위를 따라 전진하므로 결국 올바른 교차점을 찾을 수 있게 된다.

광선이 전진한 위치가 피라미드 변위 맵 상의 x, y 두 축의 노드를 동시에 넘어가는 경우가 생길 수 있다. 이 경우 현재 위치에서 광선의 방향으로의 경계까지의 거리가 더 짧은 축의 노드를 넘어가도록 한 뒤, 광선 추적을 계속 진행한다.

3.3 확대를 위한 선형 보간

지금까지 점 샘플링(point sampling)에 의한 교차점 검사 문제를 다루었다. 하지만 부드러운 굴곡을 표현하기 위하여, 일반적으로는 선형 보간(linear interpolation)을 선호한다(그림 14). 불행하게도, 피라미드 변위 맵을 사용한 우리의 알고리즘은 선형 보간된 텍스처를 사용할 수 없다(그림 15). 이러한 문제를 완화하기 위하여 Cohen[4]은 다중 샘플링을 사용하였지만, 이는 속도에 많은 희생을 따르고 선형 보간된 변위 맵과의 교차점을 보장할 수 없다.

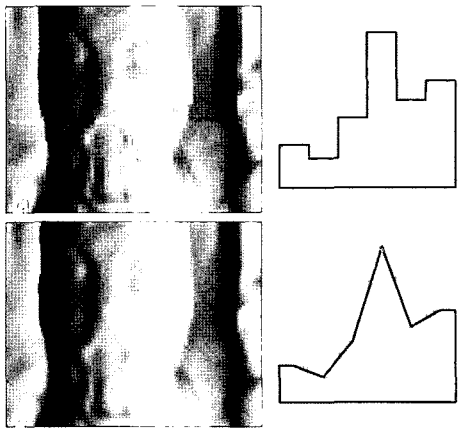

그림 14 점 샘플링에 의한 렌더링(위)과 제안하는 선형 보간 방법을 사용한 렌더링(아래)의 비교

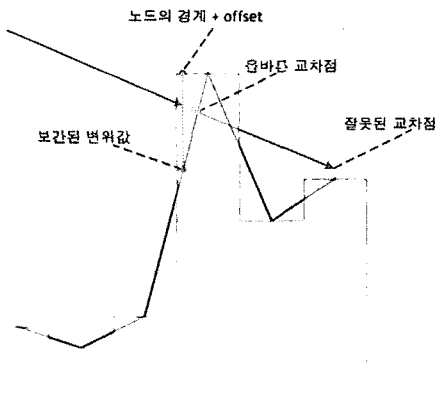

그림 15 보간된 변위 맵을 사용하여 광선 추적을 수행할 경우, 오류가 발생하게 된다.

이 문제를 해결하기 위하여, 우리는 새로운 방법을 제안한다. 우선 리프 레벨에 도달할 때까지는 3.2절에서 설명한 방식대로 점 샘플링으로 탐색한다. 현재 광선의 위치, P에서 앞/뒤 한 텍셀 안에 보간된 교차점이 있다고 가정하고, 다음의 추가적인 작업을 통해 보간된 교차점, P'을 찾는다.

그림 16에서와 같이, 반 텍셀 크기만큼 전진하여 P_b 그리고 후진하여 P_a를 구한다. 그 후, 이들 좌표에서 변위값 d_a와 d_b를 읽어오고, 거리 a와 b를 구한다. 만일 a가 b보다 크거나 같다면, P_a와 P_b를 아래의 식을 통해 적절히 선형 보간하여 P'을 계산하고 탐색을 종료한다.

$$P' = lerp(P_a, P_b, a/(a+b))$$

그렇지 않은 경우, P_b에서 보간된 변위값, d_i을 읽어온 뒤, d_i에 해당하는 위치까지 광선을 전진시킨다. 그 후 다시 점 샘플링을 통한 탐색을 진행한 뒤, 앞서 설명한 보간을 수행한다. 이 과정을 현재 광선의 높이가 d_i와 같거나 낮을 때까지 반복한다(그림 17).

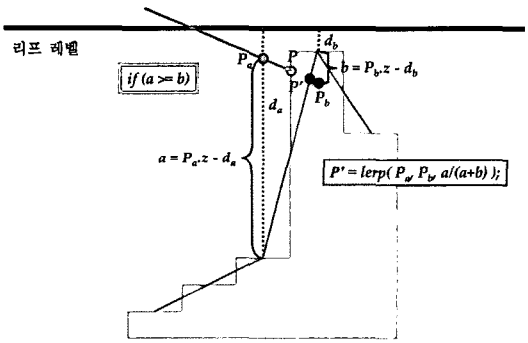

그림 16 광선이 반 텍셀 크기만큼 전진한 위치와 후진한 위치를 적절한 비율로 선형 보간하여, 보간된 교차점을 찾는다.

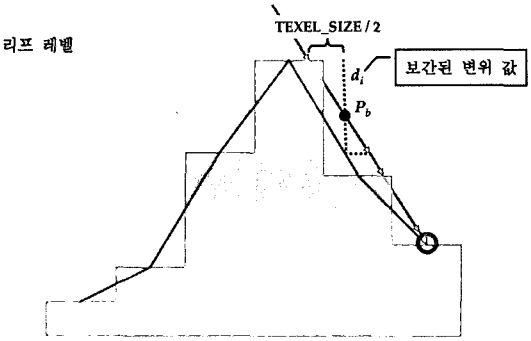

그림 17 P_b에서 보간된 변위값, d_i를 읽어온 뒤, d_i에 해당하는 위치까지 광선을 전진시킨 후, 점 샘플링과 보간을 반복하여 교차점을 찾아나간다.

3.4 축소를 위한 밉맵 필터링

이번 하위 섹션에서는 물체가 멀리 있거나 작아졌을 때, 화질을 개선하고 속도를 가속화하기 위한 축소 필터링(minification filtering)을 기술한다. 우리는 이미지 피라미드를 사용하며 루트(1x1)부터 하향식으로 쿼드트리를 탐색한다. 따라서 간단하게, 거리가 멀어짐에 따라 쿼드트리의 최대 탐색 레벨을 제한(즉, 중간 노드를 리프 노드로 취급)함으로써 쉽게 축소 필터링을 수행한다. [16,12]는 픽셀 쉐이더에서 실시간에 LOD 레벨을 결정하는 방법을 소개하였다.

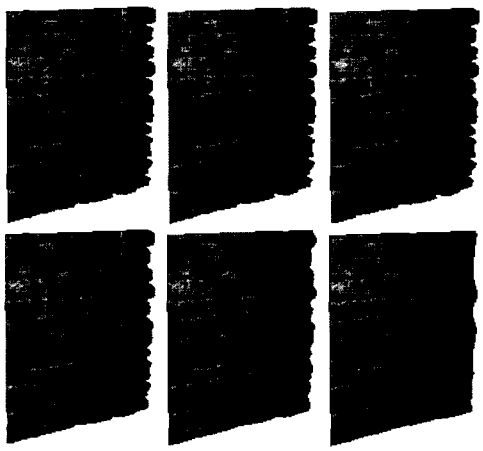

그림 18 최대 탐색 레벨의 제한을 사용한 LOD의 예. 왼쪽 위부터 오른쪽 아래로 가면서 최대 탐색 레벨이 8, 7, 6, 5, 4, 3으로 낮아진다.

4. 구 현

우리의 주 목표 중 하나는 [4]의 기법을 GPU 친화적으로 구현하는 것이다. 이번 절에서는 구현 문제를 다룬다. 우리는 본 논문에서 제안하는 기법을 DirectX 9.0의 HLSL 3.0을 기반으로 구현하였다. 우리의 알고리즘은 이미지 공간 상에서 수행하기 때문에 대부분의 작업은 픽셀 쉐이더(pixel shader)를 사용한다. 광선 설정과 조명 계산은 릴리프 매핑[11]에서와 일치하며, 광선 탐색 방식만 다르다.

정점 쉐이더(vertex shader)에서는 시점과 광원 벡터를 탄젠트 공간으로 변환한다. 이 작업은 픽셀 쉐이더에서 수행할 수도 있지만, 우리는 적은 개수의 정점을 가진 물체에 상세도를 높이는 것이 목표이기 때문에 상대적으로 처리할 양이 적은 정점 쉐이더에서 이러한 변환을 수행함으로써 속도를 향상시킬 수 있다.

픽셀 쉐이더에서, 먼저 탄젠트 공간 상으로 변환된 시점과 광원 정보를 사용하여, 시점 방향과 광원 방향을

구한다. 앞서 설명한 것과 같이, 광선의 시작점의 x, y 값은 현재 텍스처 좌표가 되며, z값은 0이 된다. 광선은 직선으로 전진하며, quadrics[3] 등을 사용하여 휘도록 하지 않는다. [17]은 릴리프 매핑에 quadric의 근사를 적용하여 실루엣을 표현하였으며, 비록 우리는 구현하지 않았지만 본 논문의 기법에도 적용할 수 있을 것이다.

트리를 탐색하기 위한 유용한 전략 중 하나는 재귀적(recursive) 알고리즘이다. 하지만 현재 쉐이더 버전 3.0에서는 모든 함수를 인라인화하며 스택이 존재하지 않기 때문에 재귀 호출을 지원하지 않는다. 따라서 우리는 반복적(iterative) 알고리즘을 사용한다. 그림 13은 이러한 알고리즘의 의사코드를 보여준다.

[4]에서는 쿼드트리의 탐색을 위해 덧셈과 시프트 연산만을 사용하는 midpoint 기법을 사용했지만, 쉐이더 프로그래밍에서는 시프트 연산을 지원하지 않는다. 이에 따라, 그림 13의 의사코드처럼 광선의 전진이 현재 노드 내부에서 일어나는지 여부 파악(IsWithinSameNode 함수)과 노드 넘어가기 작업(CrossNode 함수)을 수행하며, 이는 현재 노드의 x, y 축 상의 범위를 직접 계산함으로써 수행한다.

먼저 현재 광선의 x, y 상의 위치에 현재 레벨에서의 해상도(가로 또는 세로 크기)를 곱한 뒤, floor 연산을 통해 현재 노드의 ID를 파악한다. 다음으로, 전진 후의 노드의 ID 또한 같은 방법으로 파악한다. 각 좌표축마다 두 노드의 ID가 서로 일치하는 지를 파악한 뒤, 일치하지 않을 경우 노드 넘어가기를 수행한다. 다음은 앞서 기술한 내용에 대한 쉐이더 코드이다.

```
float2 currentID = floor(P2.xy * resolution);
float2 nextID = floor(tmpP2.xy * resolution);
int isSameNodeX = (currentID.x == nextID.x) ? 1 : 0;
int isSameNodeY = (currentID.y == nextID.y) ? 1 : 0;
```

이러한 광선 추적을 통해 변위 맵과의 교차점을 찾으면, 광선의 현재 위치에서 법선 정보를 읽고, 범프 매핑과 동일하게 이 값을 사용하여 조명을 계산한다. 자기 그림자(self shadow)를 표현하기 위해서는, 빛의 위치에서 광선과 변위 맵의 교차점 방향으로 광선 추적을 수행한 뒤, 빛에서 출발한 광선과 변위 맵의 교차점이 눈에서 출발한 광선과 변위 맵과의 교차점이 서로 다른 경우 그림자로 판단한다.

5. 결 과

우리는 ATI의 Radeon X1900 512MB 그래픽 카드 상에서 실험하였다. 거의 모든 과정을 GPU 상에서 처리하기 때문에 CPU의 사양은 알고리즘의 성능에 영향을 주지 않는다. 모든 실험에서 화면 해상도는 512×512를 사용하였다. 그림 4와 21을 제외한 모든 결과 이미지는 256×256 RGBA 32비트(A8R8G8B8) 변위 맵을 사용하였다. 그림 4는 512×512, 그림 21은 256×256, 512×512, 1024×1024, 2048×2048의 해상도로 변화시키며 렌더링한 결과이다. 그림 1의 e에서 사용된 얼굴 이미지는, 실시간에 획득한 사람의 얼굴에 대한 깊이 맵[18]을 변위 맵으로 사용하여 렌더링한 결과이다.

우리의 기법은 마이크로 폴리곤을 사용하였을 때와 유사한 화질을 보인다(그림 1의 c와 d, 그림 20). 마이크로 폴리곤을 사용한 결과는 131,072개의 다각형을 사용한 결과이며, 우리의 기법은 오직 2개의 다각형만을 사용한다. 그림 20의 이미지를 512×512 해상도의 화면에 렌더링하기 위하여, 우리의 기법은 약 0.35MB의 메모리를 필요로 하였고, 480.06fps로 렌더링할 수 있다. 이에 반해 높이 매핑은 많은 개수의 폴리곤으로 인하여 약 2.1MB 이상의 메모리를 필요로 하였고, 50.13fps로 렌더링되었다. 보다 상세도가 높은 모델을 사용할 경우, 이러한 차이는 극대화된다.

그림 19와 22는 우리의 기법이 [1,10,11]의 기법에서 심각한 문제를 발생시키던 날카로운 변위 맵와 예각에서도 좋은 화질로 렌더링할 수 있음을 보여준다. 이러한 문제를 해결한 기법들 [13,14]은 텍스처의 해상도가 증가함에 따라 큰 성능 저하를 보인다. 그림 21은 우리의 기법이 계층적인 접근 방법을 채택하여 변위 맵의 해상도가 증가하여도 속도가 크게 감소하지 않는 것을 보여준다. 256×256 해상도에서 480.06fps, 512×512 해상도에서 437.29fps, 1024×1024 해상도에서 421.35fps, 그리고 2048×2048 해상도에서 375.53fps의 속도로 렌더링할 수 있었다.

표 1은 본 논문에서 사용된 변위 맵에서의 렌더링 속도는 나타내며, 표 2는 시차 폐색 매핑, 릴리프 매핑 그리고 제안된 기법의 속도를 비교한 것이다.

비록 우리의 기법이 약 2배 정도 느린 성능을 보였지만, 그림 19와 22에서와 같이 기존의 기법들에 비해 월

표 1 렌더링 속도(단위: 초당 프레임 수)

	점 샘플링	선형 보간
지형 (그림 1)	482.27	285.88
얼굴 (그림 1)	494.89	292.84
자갈 벽 (그림 22)	521.56	306.80

표 2 렌더링 속도 비교(단위: 초당 프레임 수)

	시차 폐색 매핑	릴리프 매핑	우리의 기법
지형	1129.59	1014.04	482.27
얼굴	1121.54	994.12	494.89
자갈 벽	1131.58	1160.47	521.56

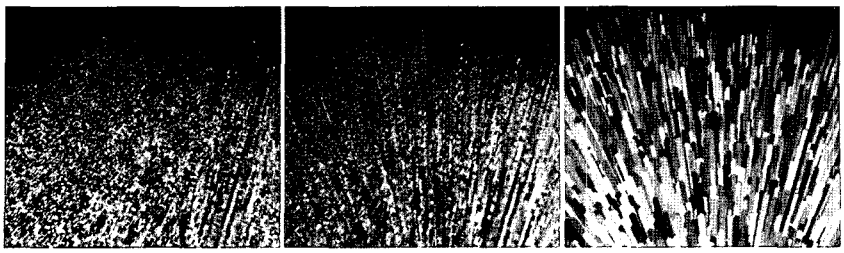

그림 19 날카로운 변위 맵에서 심각한 오류를 발생하는 시차 폐색 매핑(좌), 이 문제를 완화하지만 해결하지 못하는 릴리프 매핑(중간), 이에 반해 깨끗한 영상을 생성하는 우리의 기법(우)의 화질 비교

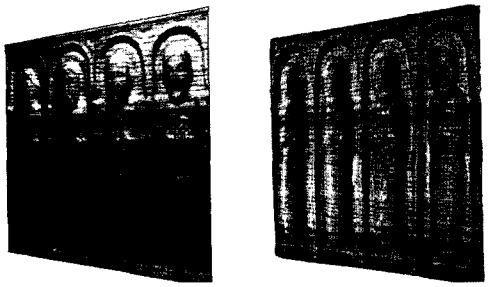

그림 20 좌: 우리의 기법(법선 맵+변위 맵+피라미드 변위 맵: 256×256×5.3 bytes; 480.06fps), 우: 높이 매핑(다각형: 256×256×8×4+변위 맵: 256×256×1 bytes; 50.13fps)

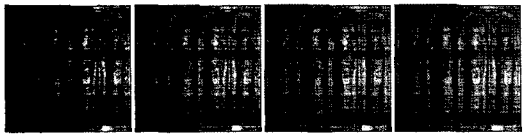

그림 21 텍스처의 해상도에 따른 렌더링 결과. 좌측부터 순서대로, 256×256(480.06fps), 512×512(437.29fps), 1024×1024(421.35fps), 2048×2048(375.53fps)

등히 우수한 화질을 제공하였다. 기존의 기법들이 우리의 기법과 동일한 화질을 제공하기 위해서는 매우 많은 샘플을 사용해야 하며(기본적으로 선형 탐색을 사용하기 때문), 이 경우 속도가 현저하게 감소한다. 필요한 샘플의 개수가 변위 맵의 형태와 시점에 극히 민감하며, 필요한 개수를 정확하게 파악하기 매우 어렵다.

6. 결론과 향후 과제

우리는 시각적 오류없이 변위 맵을 실시간에 렌더링하기 위한 GPU 기반의 기법을 소개하였다. 이를 위하여 쿼드트리의 중간 노드에 가장 작은 변위값을 저장한 이미지 피라미드 형태의 피라미드 변위 맵을 사용하여, 높은 곳에서 낮은 곳으로 계층적으로 탐색하며 광선과 변위 맵의 교차 검사를 수행하였다. 이 기법은, 기존의

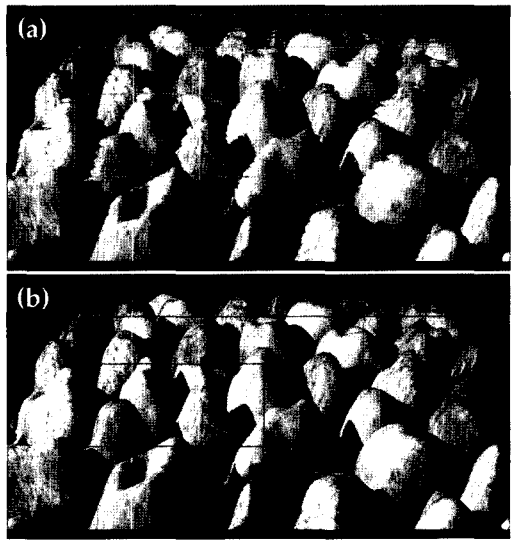

그림 22 릴리프 매핑은 굴곡과 그림자 표현이 가능하지만, 교차점을 지나쳐서 잘못된 이미지를 생성한다(a). 반면에, 우리의 기법은 정확하게 표현할 수 있다(b).

GPU 기반 기법들에서 심각한 오류를 발생하는 날카로운 변위 맵이나 예각에서도 정확한 굴곡의 표현이 가능하였다. 또한, 변위 맵의 해상도가 증가함에 따른 속도의 손실이 적었다. 우리는 또한, 선형 보간된 부드러운 변위 맵을 렌더링하기 위한 방법을 소개하였다. 밉맵과 LOD를 적용하여 화질을 향상시키고, 렌더링 속도를 가속화하였다. 제안된 기법은 가상 현실, 게임, 방송 시스템 등에서 사각형 등의 단순한 폴리곤만으로 복잡한 기하를 표현하는데 유용하게 사용될 수 있을 것이다.

이 기법은 거칠게 근사하는 기존의 GPU 기반 기법들[10-12]에 비해 훨씬 정확하지만, 속도에서는 약간의 희생을 따른다. 이는 보간된 굴곡을 표현할 때 더욱 커진다. 우리는 향후 연구에서, 광선과 변위 맵의 교차 검사와 보간된 변위 맵의 렌더링을 보다 가속화하고, 향상된 보간 방법을 연구할 것이다. 또한, [18]과 같이 굽은 광선을 사용하여 실루엣을 표현하도록 개선할 것이다.

참 고 문 헌

[1] Blinn, J. F. 1978. Simulation of Wrinkled Surfaces. *In Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ACM Press, pp. 286-292.

[2] Cook, R. L. 1984. Shade trees. *In Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ACM Press, 223-231.

[3] Patterson, J.W., Hoggar. S. G., and Logie, J.R. 1991. Inverse Displacement Mapping. In EUROGRAPHICS Conference Proceedings. *Computer Graphics Forum* Volume 10, Issue 2 (1991) pp. 129-139.

[4] Cohen, D. and Shaked, A., Photo-Realistic Imaging of Digital Terrain, *Computer Graphics Forum*, 12, 3 (September 1993), 363-374.

[5] Wright, J. R., and Hsieh, J. C. L. A Voxel-Based, *Forward* Projection Algorithm for Rendering Surface and Volumetric Data. *Proc. IEEE Visualization '92*. IEEE Computer Society, Boston, MA, 1992, pp. 340-348.

[6] Cohen, D. and Shaked, A., Photo-Realistic Imaging of Digital Terrain, *Computer Graphics Forum*, 12, 3 (September 1993), 363-374.

[7] Lee, C., and Shin, Y. G., An Efficient Ray Tracing Method for Terrain Rendering, *In proceedings of Pacific Graphics '95*, 1995, pp. 180-193.

[8] Musgrave, F. K. Grid tracing: Fast Ray Tracing for Height Fields. *Technical report*, Department of Mathematics, Yale University, December 1991.

[9] Coquillart, S., and Gangnet, M. 1984. Shaded display of digital maps. *IEEE Computer Graphics and Applications*, pages 35-42, July 1984.

[10] Brawley, Z., and Tatarchuk, N. 2004. Parallax Occlusion Mapping: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing. *In ShaderX3: Advanced Rendering with DirectX and OpenGL*, Engel, W., Ed., Charles River Media, pp. 135-154.

[11] Policarpo, F., Oliveira, M. M., and Comba, J. 2005. Real-Time Relief Mapping on Arbitrary Polygonal Surfaces. *In ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games Proceedings*, ACM Press, pp. 155-162.

[12] Natalya, M. 2006. Dynamic Parallax Occlusion Mapping with approximate soft shadows. *In Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM Press, pp.63-69.

[13] Donnelly, W. 2005. Per-Pixel Displacement Mapping with Distance Functions. *In GPU Gems 2*, M. Pharr, Ed., Addison-Wesley, pp. 123-136.

[14] Baboud, L., and Decoret, X. 2006. Rendering Geometry with Relief Textures. *In Graphics interface Conference Proceedings*.

[15] Kaneko, T., Takahei, T., Inami, M., Kawakami, N., Yanagida, Y., Maeda, T., and Tachi, S. 2001. Detailed Shape Representation with Parallax Mapping. *In Proceedings of the ICAT 2001*, 205-208.

[16] Shreinder, D., Woo, M., Neider, J., and Davis, T.. 2005. *OpenGL® Programming Guide: The Official Guide to Learning OpenGL®*, version 2, Addison-Wesley.

[17] Oliveira, M. M., and Policarpo, F. 2005. An Efficient Representation for Surface Details. *UFRGS Technical Report RP-351*, January 26, 2005.

[18] Tsalakanidou, F., Forster, F., Malassiotis, S., and Strintzis, M. G. 2005. Real-time acquisition of depth and color images using structured light and its application to 3D face recognition. *Real-Time Imaging* 11, 5-6 (Oct. 2005), 358-369.

오 경 수

정보과학회논문지 : 시스템 및 이론
제 34 권 제 4 호 참조

기 현 우

정보과학회논문지 : 시스템 및 이론
제 34 권 제 4 호 참조