# GPU를 이용한 영상 재투영

Image Reprojection Using GPU

김효원, Hyowon Kim*, 기현우, Hyunwoo Ki*, 이호현 Hohyun Lee*, 오경수, Kyoungsu Oh*
*숭실대학교 미디어 학과

**요약** 영상 재투영이란, 깊이 맵을 투영하여 임의의 시점에서 본 이미지를 생성해내는 기법을 말한다. 기존의 CPU를 이용한 영상 재투영 기법들의 가장 큰 단점은 CPU와 GPU 간의 데이터 복사가 일어나고 재투영 연산 자체의 속도가 느리기 때문에 실시간 렌더링이 불가능 하다는 것이다. 따라서 본 논문에서는 GPU를 이용하여 영상 재투영을 구현하고 실시간에 이미지를 렌더링하는 기법을 소개한다.
우리의 기법은 입력으로 참조 이미지와 해당 이미지의 깊이 맵이 주어졌을 때, 임의의 시점에서 보이는 새로운 이미지를 실시간으로 생성한다. 임의의 시점에서 이미지를 생성하기 위해, 각 픽셀에서 참조 이미지에 해당하는 평면을 렌더링하여 시점 반대 방향의 광선을 생성한다. 이 광선을 참조 이미지의 투영 공간으로 변환한 후, 광선과 깊이 맵 간의 교차점을 찾는다. 이렇게 찾아낸 깊이 맵의 교차점과 일치하는 참조 이미지의 픽셀 색으로 새로운 시점의 이미지를 만들어 낼 수 있다. 이와 같은 기법은 기하 정보의 복잡도와 관계없이 수십 프레임의 속도로 실시간 렌더링이 가능하다.

***핵심어: Reprojection, GPU, Image-based Rendering, Real-time Rendering***

## 1. 서론

영상 재투영 (image reprojection) 이란, 깊이 맵을 투영하여 임의의 시점에서 본 이미지를 생성해내는 기법을 말한다. 이 기법은 연산량이 기하 정보량과 무관하기 때문에 어떠한 복잡한 장면도 속도 저하 없이 렌더링이 가능하다. 또한 컴퓨터로 생성해 낸 이미지뿐만 아니라 깊이 정보를 가지고 있는 실사 이미지를 적용할 경우, 임의의 시점에서의 또 다른 실사 이미지도 생성해 낼 수 있다.

우리의 기법은 이미 존재하는 이미지를 바탕으로 다른 각도에서의 정보나 새로운 시각적 효과들을 만들어 낼 수 있다. 따라서 시각 디자인이나 인터렉티브 아트 작품 등에서 우리의 기법이 하나의 솔루션이나 아이디어를 제공할 수 있을 것으로 기대된다.

기존의 연구들 중에 텍스처의 깊이 정보를 표현하기 위해 변위 맵을 사용하는 여러 가지 기법들이 존재한다. 역변위 매핑 (inverse displacement mapping) [1]은 기하 정보의 변경 없이 변위 맵에 저장된 높이 값들과 각 픽셀에서 출발한 광선의 교차점을 구해서 각 픽셀에 해당하는 텍셀의 위치를 결정하는 방법이다 (그림 2). 여기서 변위 맵

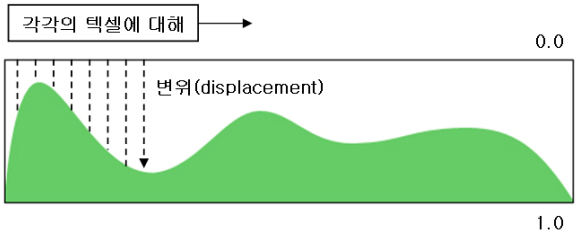


그림 1. 변위 맵은 각 픽셀에서 물체 표면의 고도에 따른 높이 차이를 저장한다.

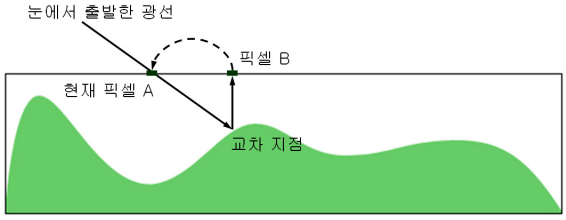


**그림 2.** 역변위 매핑은 픽셀 A를 렌더링하기 위해 광선의 교차 지점에 해당하는 픽셀 B의 정보를 사용한다.

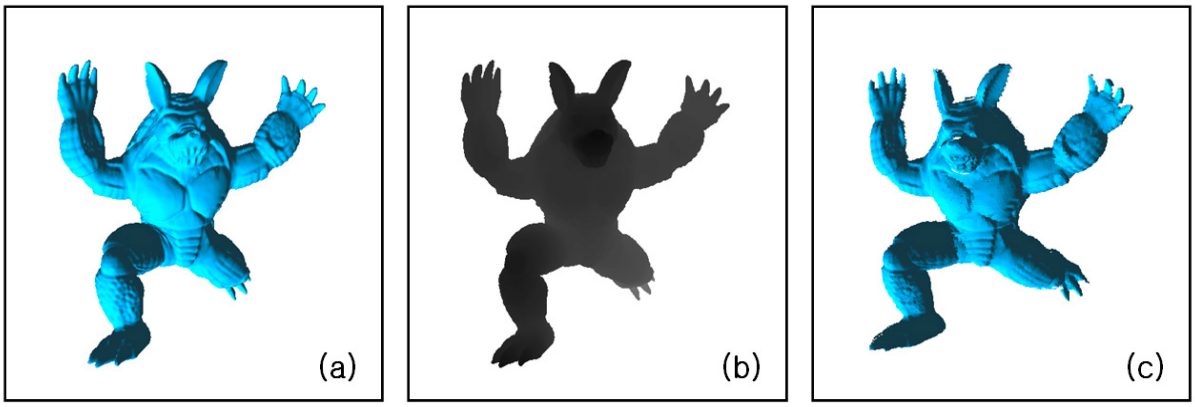


**그림 3.** (a)는 참조 이미지, (b)는 (a)의 깊이맵이다. 우리의 방법은 (a)와 (b)를 입력 받아 (c)와 같이 다른 시점에서 본 새로운 이미지를 생성한다.

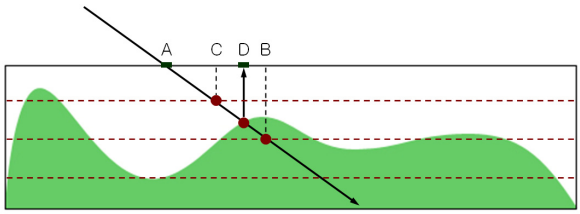


**그림 4.** 릴리프 매핑 (relief mapping) 은 이진 탐색 (binary search) 를 통해 광선과 변위 맵 간의 교차점을 찾는다. 픽셀 A를 렌더링 하기 위해, 광선의 중간 지점인 B, B와 A의 중간 지점인 C, B와 C의 중간 지점인 D를 차례로 탐색한다.

(displacement map) [2]은 물체 표면의 고도에 따른 높이 차이를 저장한 맵을 말한다 (그림 1).

여기서 교차점을 찾는 것은 일종의 광선 추적 (ray tracing) 방식인데, 기존의 방식들은 주로 CPU 기반 이었다 [6, 7, 8]. 변위 맵에 대한 이미지 피라미드를 사용하는 [6]은 변위 맵을 쿼드트리 형태로 계층화한다. [9, 10]는 점진적인 알고리즘 (incremental algorithm) 형태의 광선 추적을 사용하여 속도를 가속화한다. 하지만 이러한 CPU 기반의 기법들은 실시간 렌더링이 불가능하다는 치명적인 단점을 가지고 있다.

최근에 등장한 광선과 변위 맵의 교차점을 구하기 위한 GPU 기반의 실시간 알고리즘으로는 시차 폐색 매핑 (parallax occlusion mapping) [3], 릴리프 매핑 (relief mapping) [4]등이 있다. 이 기법들은 교차 검사를 위해 선형 탐색 (linear search) 또는 이진 탐색 (binary search) (그림 4) 을 수행하는데, 이는 종종 시각적 오류를 유발할 수 있다.

굴곡을 정확하게 표현하는 실시간 변위 맵핑 [5]은 변위 맵을 쿼드트리 형태의 이미지 피라미드로 만들고, 이 트리를 하향식으로 탐색한다. 이 기법은 기존의 기법들과 달리 예각에서도 시각적 오류 없이 정확한 이미지를 생성한다. 따라서 우리는 광선과 깊이 맵 간의 교차점을 찾기 위해 같은 방식의 교차점 검출 알고리즘을 사용하였다.

본 논문에서는 영상 재투영 기법의 전 단계를 GPU 상에서 구현하는 방법을 제안한다. 우리의 기법은 입력으로 참조 이미지와 해당 이미지의 깊이 맵이 주어졌을 때, 임의의 시점에서 보이는 새로운 이미지를 실시간으로 생성한다. 알고리즘은 각 픽셀에서 광선을 생성하는 부분, 깊이 맵과 광선과의 교차점을 검출하는 부분으로 크게 나누어진다.

우선 새로운 시점에서 각 픽셀의 광선을 생성하기 위해 참조 이미지에 해당하는 평면을 렌더링 한다. 각 픽셀에서는 그 픽셀에 해당하는 평면의 위치로부터 시점 반대 방향의 광선을 생성한다. 이 광선을 참조 이미지의 투영 공간으로 변환한 후, 참조 이미지의 깊이 맵 간의 교차점을 찾는다. 최종적으로 교차점에 해당하는 참조 이미지의 픽셀 값으로 렌더링하여 새로운 시점에서의 이미지를 생성할 수 있다.

그림 3 은 참조 이미지와 해당 깊이 맵, 그리고 이를 이용하여 새로운 시점에서 렌더링한 결과 이미지를 보여주고 있다. 우리의 기법은 기하 정보량과 관계없이 수십 프레임의 속도를 얻었다.

이 후 본문에서는 전체 시스템의 입출력 정보에 대해 서술하고 알고리즘을 두 단계로 나누어 설명한다. 첫 번째 단계는 각 픽셀에서의 광선을 결정하는 방법이며, 두 번째 단계는 광선과 깊이 맵 간의 교차점 검출에 대한 알리즘이다. 마지막으로 우리의 기법을 활용하여 렌더링한 결과 이미지들과 수행 속도에 대한 결과를 제시 할 것이다.

## 2. GPU 기반의 영상 재투영 방법

GPU를 이용하여 영상 재투영을 구현하는 우리의 방법에 대해 세 부분으로 나누어 소개할 것이다. 실험을 위해 3차원

메쉬를 임의의 시점에서 렌더링하여 참조 이미지와 깊이 맵을 얻는다. 이를 이용하여 새로운 시점에서 어떻게 새로운 이미지를 생성해내는지 설명한다.

## 2.1 입력과 출력

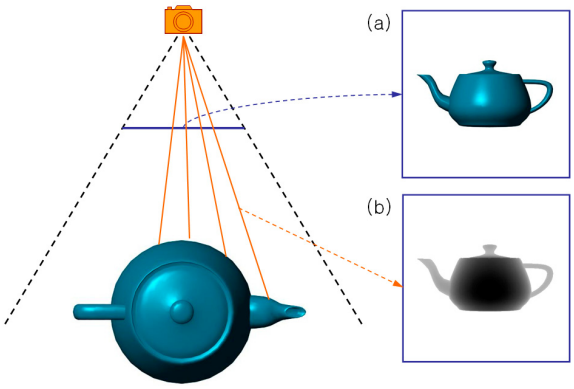


**그림 5.** 입력 데이터는 임의의 시점에서 물체를 렌더링한 참조 이미지 (a)와 카메라에서 물체까지의 z 값을 저장한 깊이맵 (b)이다.

우리의 방법은 깊이 정보를 가지고 있는 이미지를 입력 받아 임의의 시점에서 본 새로운 이미지를 출력 결과로 얻을 수 있다.

실험을 위해 그림 5에서와 같이 3차원 물체를 임의의 시점에서 렌더링하여 참조 이미지와 깊이 맵을 얻는다. 깊이 맵은 카메라 위치에서 물체까지의 z 값을 저장하여 쉽게 생성할 수 있다.

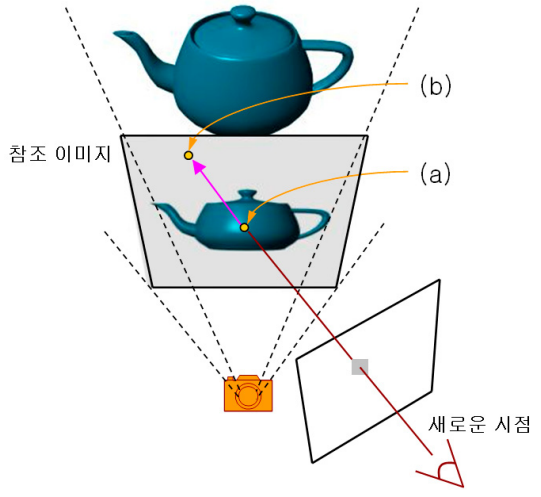


**그림 6.** 새로운 시점에서 참조 이미지에 해당하는 3차원 평면 (plane) 을 렌더링한다. 각 픽셀에서 출발한 광선이 평면과 만나는 지점 (a)는 광선의 시작 지점이다. (b)는 (a)에서 같은 방향으로 임의의 크기만큼 전진한 지점이다. 우리는 (a)와 (b)를 월드 상에서의 3차원 좌표로 변환한 뒤, 참조 이미지의 투영 공간 좌표로 변환한다.

```
VS_OUTPUT RaySetupVS(VS_INPUT In)
{
   OUTPUT Out;
   float4pos = float4(In.Position, 1.0f);
   Out.Position = mul(pos, g_mWVP);

    float4 a = mul(pos, g_mWV);
    float4 b = a * 1.5;
    b.w = 1.0f

   float4 trans_a = mul(a, g_mEyeToRefProj);
    float4 trans_b = mul(b, g_mEyeToRefProj);
    trans_a /= trans_a.w;
    trans_b /= trans_b.w;

   float4 RayDirection = trans_b - trans_a
   RayDirection = normalize(RayDirection)
   RayDirection /= RayDirection.z;

   Out.start = trans_a.xyz;
   Out.direction = RayDirection.xyz

   return Out;
}
```

**그림 7.** 각 픽셀에서 광선을 생성하기 위한정점 쉐이더

## 2.2 광선 생성

참조 이미지를 렌더링한 시점과 다른 지점의 시점이 주어졌을 때, 새로운 시점에서의 이미지를 생성하기 위해 각 픽셀에서 시점 반대 방향으로의 광선을 생성한다. 각 픽셀의 광선은 새로운 시점에서 참조 이미지에 해당하는 평면을 렌더링하여 얻을 수 있다. 여기서의 평면은 참조 이미지가 렌더링 된 3차원 공간 상의 사각형으로 생각 할 수 있다.

새롭게 주어진 시점에서 각 픽셀의 광선이 출발하여 참조 이미지 평면과 만나는 지점이 우리가 찾고자 하는 광선의 시작 지점이다. 이는 그림 6 에서 (a)에 해당한다. 우리는 정점 쉐이더에서 입력 데이터로 들어오는 (a)의 좌표에 월드 행렬을 곱하여 월드 좌표계로 변환하였다. 그리고 월드 좌표계 상의 (a)를 임의의 크기만큼 전진시켜 또 다른 좌표인 (b)를 구한다.

각 픽셀에서 얻은 광선과 교차 검사를 수행할 깊이맵은 참조 이미지와 동일한 카메라 투영 변환을 거친 값이다. 따라서 우리가 얻고자 하는 최종 광선은 새로운 시점의 투영 공간이 아닌, 참조 이미지를 렌더링한 시점의 투영 공간에 있어야 한다.

월드 좌표계 상의 (a)와 (b)에 참조 이미지의 카메라 투영 행렬을 곱하여 투영 좌표계로 변환한다. 변환을 거친 좌표 (a)는 광선의 시작 지점이 되고 좌표 (b)와 (a)의 차는 광선의 방향 벡터가 된다. 마지막으로 참조 이미지의 깊이 맵은 0~1 사이의 값을 가지므로 광선의 z 값 또한 0~1 사이로 조정해야 한다.

이와 같이 각 픽셀의 광선 정보는 정점 쉐이더에서 계산

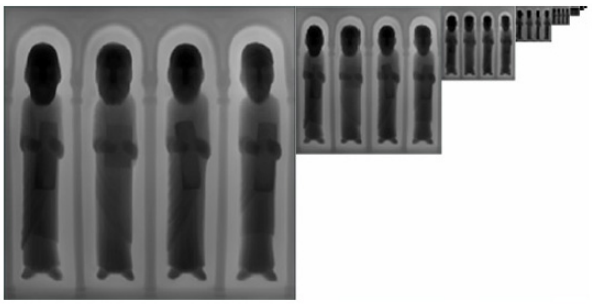

**그림 8.** 사분면 픽셀 (자식 노드) 중 가장 짧은 깊이 값을 저장하는 피라미드 깊이 맵의 예

하여 픽셀 쉐이더로 넘겨진다. 광선을 생성하기 위한 정점 쉐이더 코드는 그림 7 에 자세히 소개하였다.

픽셀 쉐이더에서는 해당 광선과 참조 이미지의 깊이 맵 간의 교차 검사를 수행한다. 교차점 검출에 대한 알고리즘은 다음 절에서 설명한다.

## 2.3 교차점 검출

영상 재투영을 위한 첫 번째 단계로 정점 쉐이더에서 각 픽셀의 광선을 생성하였다. 우리는 정점 쉐이더에서 얻은 광선의 시작 좌표와 방향 정보를 가지고 픽셀 쉐이더에서 교차점 검출을 수행한다. 여기서 얻고자 하는 교차점은 광선과 참조 이미지의 깊이 맵 간의 교차 지점이다. 현재 광선과 깊이 맵은 모두 참조 이미지의 투영 공간의 상에 있으므로 깊이 값은 투영면과의 수직 거리이다.

우리는 [5] 의 방법을 써서 깊이 맵과 광선 사이의 교차점을 찾아낸다. [5] 의 방법은 피라미드 변위 맵핑 알고리즘을 사용하여 어떤 각도에서도 오류 없이 정확한 교차점을 찾아낸다. 이 기법은 기존의 시차 폐색, 릴리프 매핑 등에서 사용한 광선 탐색 기법과 유사하지만, 쿼드트리 구조의 피라미드 깊이 맵을 GPU 기반으로 탐색하기 때문에 빠르고 정확하게 렌더링할 수 있다.

이미지 피라미드는 2n x 2n 의 픽셀을 가진 0레벨부터, 20 x 20 개의 픽셀을 가진 n 레벨까지의 계층적인 이미지들의 컬렉션을 의미한다 (그림 8). 우리는 깊이 맵을 이미지 피라미드로 만들어 사용한다. 깊이 이미지 피라미드에서 서브 레벨 텍스처의 각 픽셀 (i, j) 에는 4개의 픽셀들(2i, 2j), (2i, 2i+1), (2i+1, 2j), (2i+1, 2j+1) 중 가장 작은 깊이 값을 저장한다. 이렇게 만들어진 쿼드트리를 루트부터 리프까지 계층적으로 탐색해 나간다.

정점 쉐이더에서 계산하여 각 픽셀에 보간되어 전달된 광선의시작점의 x,y 값은 현재 텍스처 좌표가 되며, z 값은 0이 된다. 그림 9 에서 피라미드 텍스처의 평면에 투영된 광선의 z 값인 d’ 가, 광선의 x, y 를 텍스처 좌표로 사용

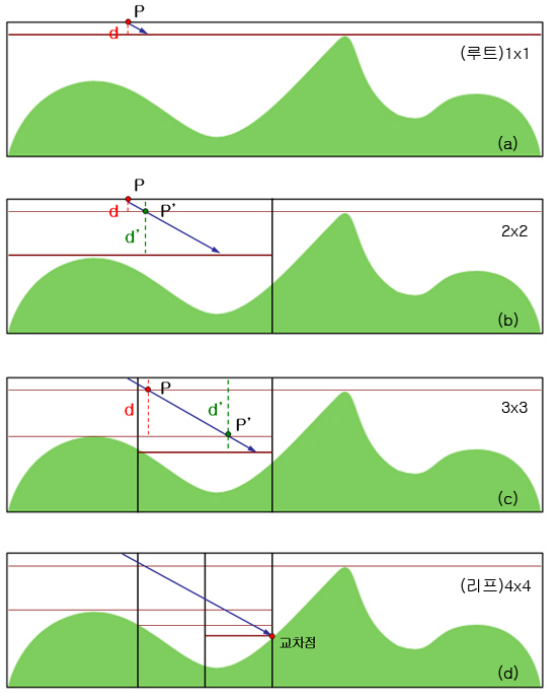


**그림 9.** 깊이 맵 피라미드의 루트에는 가장 작은 깊이 값이 저장되어 있으므로 해당 위치까지전진한다.(a) 현재 광선의 깊이 값인 d 와 전진한 광선 위치 P’ 에서 다음 레벨의 깊이 값인 d’ 를 읽어와 비교한다. d’ 의 값이 d 의 값보다 크면 광선을 d’ 깊이까지 전진 시킨다 (b, c). 이 과정을 반복하여 리프에 도달하면 교차점을 찾을 수 있다 (d).

하여 현재 레벨의 피라미드 깊이 맵에서 읽어온 z 값인 d 보다 크면, 광선을d' 값만큼 전진시키고 다음 레벨로 내려간다. 그렇지 않으면 전진하지 않고 다음 레벨로 내려가기만 한다. 이 작업을 리프 노드에 도착할 때까지 반복한다. 이렇게 계산된 광선의 위치에서의 x, y 좌표를 사용하여 참조 이미지를 참조하고 해당 픽셀의 색을 결정한다.

이러한 피라미드 깊이 맵을 사용한 계층적 탐색은 기존의 방식들에서 표현 가능한 완만한 깊이 맵 뿐만 아니라, 날카로운 깊이 맵을 예각에서 보았을 때도 정확하고 빠르게 수행될 수 있다. 또한 텍스처의 해상도가 증가함에 따른 렌더링 속도의 저하가 작다.

## 3. 결과

우리는 ATI의 Radeon X1900 512MB 그래픽 카드 상에서 실험하였다. 거의 모든 과정을 GPU 상에서 처리하기 때문에 CPU 의 사양은 알고리즘의 성능에 영향을 주지 않는다.

그림 10 의 몬스터 모양의 모델(아래)는 약 33만 개의 폴리곤을 가진 모델이며, 남자 흉상의 모델(위)는 약 60만 개의 폴리곤을 가진 모델이다. 우리의 기법으로 마이크로 폴리곤을 사용한 경우와 유사한 이미지를 얻을 수 있음을 알

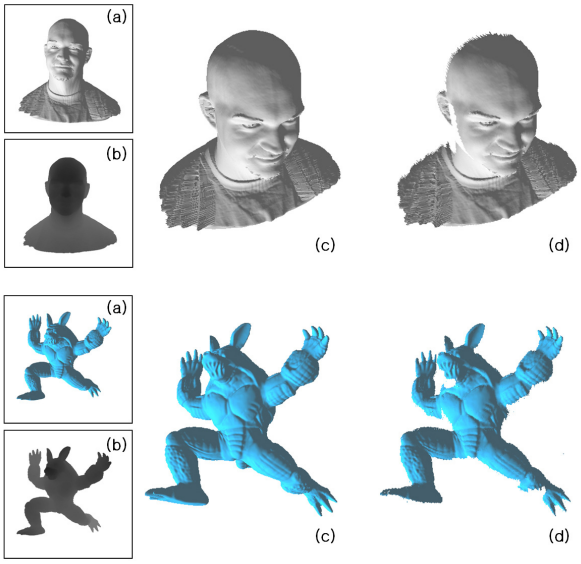


**그림 10.** (a) 는 참조 이미지, (b) 는 깊이 맵, (c) 는 현재 시점에서 실제 메쉬를 이용한 렌더링 결과, (d) 는 현재 시점에서 우리의 기법으로 렌더링한 결과이다.

수 있다.

우리의 기법에 해당하는 이미지 (d)는 실제 메쉬를 렌더링한 결과 (c)에 비해 렌더링 되지 않은 부분이 많이 나타난다. 이는 참조 이미지에 해당하는 (a)에서 찾을 수 없는 정보가 렌더링 되는 부분은 비어 있도록 처리하였기 때문이다. 기존의 변위 맵 렌더링 알고리즘 [3, 4, 5] 의 경우는 주변 정보를 이용한 보간을 통해 비어있는 영역을 채우는 방법을 사용하였다. 우리의 기법은 광선과 깊이 맵이 교차한 지점이 주변의 깊이 값과 일정 기준보다 크게 차이나는 경우 정보를 찾을 수 없는 구멍으로 판단하여 알파 값을 0으로 주어 렌더링 하였다.

또한 표 1 의 결과를 통해 대용량 기하 정보로 인해 실시간 렌더링이 불가능한 경우, 우리의 기법이 대안으로 사용될 수 있음을 알 수 있다. 우리의 기법이 시점에 따라 다른 렌더링 속도를 보이는 것은 교차점 검출 알고리즘에서 시점에 따라 광선이 탐색해 들어가는 깊이가 달라지기 때문이다. 광선이 탐색해 들어가는 깊이가 길어질수록 피라미드 깊이 맵을 읽는 횟수가 많아져야 한다. GPU에서 텍스처를 읽는 연산은 매우 느린 연산 중에 하나이므로 속도 저하에 영향을 많이 미친다. 따라서 예각에서 렌더링 할수록 교차점을 찾는 과정이 길어져 렌더링 속도가 저하된다.

표 1. 렌더링 속도

| | 메쉬 렌더링 | 우리의 기법 (최적의 시점에서) | 우리의 기법 (최악의 시점에서) |
|---|---|---|---|
| 남자 (위) | 약 5 fps | 약 200 fps | 약 50 fps |
| 몬스터 (아래) | 약 9.5 fps | | |

## 5. 결론 및 향후 과제

우리는 GPU를 이용하여 영상 재투영을 실시간 렌더링하는 기법을 소개하였다. 이를 위하여 각 픽셀에서 시점 반대 방향의 광선을 생성하여 참조 이미지의 투영 공간으로 변환시켰다. 픽셀 쉐이더에서는 생성한 광선과 참조 이미지의 깊이 맵 간의 교차점을 검출하였다. 교차 검사 알고리즘으로는 이미지 피라미드를 이용한 계층적 탐색 방식을 사용하였다.

우리의 기법은 기존의 CPU를 이용한 영상 재투영 방식보다 빠르며 실시간 렌더링이 가능하다. 또한 어떠한 각도에서도 교차점 검출 오류로 인한 잘못된 결과 없이 정확한 이미지를 생성한다.

그림 10 에서와 같이 우리의 결과 이미지는 마이크로 폴리곤을 사용한 경우와 유사한 화질을 보인다. 우리의 기법은 장면 내의 기하 정보의 크기와 관계없이 오직 2개의 다각형만으로 빠르게 렌더링할 수 있다.

또한 컴퓨터로 생성한 이미지가 아닌 실사 이미지를 활용하여 새로운 시점의 이미지를 생성할 수 있다. 이는 기존의 이미지로부터 새로운 시각적 효과나 다른 각도에서의 해석을 요구하는 예술적 작업들을 가능하게 한다. 따라서 시각 디자인이나 인터렉티브 아트 작업 등에서 우리의 기법이 하나의 솔루션으로 활용될 수 있을 것이라 기대한다.

우리는 앞으로의 과제에서 여러 장의 참조 이미지와 깊이 맵을 입력 받아 좀 더 다양한 각도에서 정확도가 높은 결과 이미지를 만들어내기 위한 방법들을 연구 할 것이다. 또한 대용량의 기하 정보로부터 가장 적합한 참조 이미지와 깊이 맵을 생성하기 위한 알고리즘을 개발할 것이다. 마지막으로 주어지는 시점에 따라 조명을 다시 계산하여 물체의 재질과 밝기를 정확하게 표현하기 위한 기법을 연구 할 것이다.

## 감사의 글

본 연구는 한국게임산업개발원의 게임리서치센터 (GRC)로부터 지원을 받았습니다.

## 참고문헌

[1] Patterson, J.W., Hoggar.S.G., and Logie, J.R. "Inverse Displacement Mapping". In EUROGRAPHICS Conference Proceedings. Computer Graphics Forum Vol. 10, Issue 2, pp.129~139, 1991.

[2] Cook, R.L. "Shade Tree". In Proceedings of the

11th annual conference on Computer graphics and interactive techniques, ACM Press, pp. 223-231, 1984.

[3] Brawley,Z., and Tatarchuk,N. "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, 2004.

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

[5] 오경수, 기현우, "이미지 피라미드를 이용한 변위 맵의 실시간 렌더링", 하계 한국게임학회 학술발표대회, p.209-215, 2006.

[6] Cohen, D. and Shaked, A., "Photo-Realistic Imaging of Digital Terrain". Computer Graphics Forum, 12, pp. 363-374, 1993.

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

[8] Lee, C. and Shin, Y. G., "An Efficient Ray Tracing Method for Terrain Rendering, In proceedings of Pacific Graphics 95, pp. 190-193, 1995.

[9] Musgrave, F. K., "Grid Tracing: Fast Ray Tracing for Height Fields". Technical report, Department of Mathematics, Yale University, 1991.

[10] Coquillart, S., and Gangnet, M. "Shaded display of Digital Maps". IEEE Computer Graphics and Applications, pp. 35-42, 1984.