바운딩 박스 세분화를 통한 지형 렌더링의 가속화※

이은석, 이진희, 조인우, 신병석
인하대학교 컴퓨터 · 정보공학과
elflee77@nate.com, jhlee07@inhaian.net, joins@inha.edu, bsshin@inha.ac.kr

Acceleration of Terrain Rendering Using Bounding Box Subdivision

Eun-Seok Lee, Jin-Hee Lee, Inwoo Jo, Byeong-Seok Shin
Dept. of Computer Science and Information Engineering, Inha University

요 약

최근의 3D게임이나 가상현실을 위한 지형 시각화 응용에서는 사실적인 장면을 렌더링 하기 위해 고화질 영상을 실시간에 제공하는 GPU기반의 광선투사법을 이용한다. 이 방법은 지형데이터의 크기가 증가할수록 샘플링 해야 하는 텍셀의 개수가 증가하기 때문에 렌더링 속도가 저하된다. 이러한 문제점을 해결하기 위해서 본 논문에서는 GPU에서 사진트리를 기반으로 수행되는 바운딩 박스 세분화를 이용하여 빈 공간이 제거된 바운딩 박스를 생성하고 이를 이용하여 광선투사법을 가속화하는 방법을 제안한다. 이 방법은 각 광선마다 빈 공간 도약을 위해 트리를 탐색하여 중복된 탐색연산을 수행해야 했던 기존의 방법과 달리 바운딩 박스를 이용하여 탐색 연산을 단 1번만 수행하도록 하여 수행속도를 가속화 하였다.

ABSTRACT

Recent terrain rendering applications such as 3D games and virtual reality, use GPU-based ray-casting method for rendering high-quality scenes in realtime. As the size of terrain dataset grows bigger, the rendering speed will be decreased by the increase of the number of texture samplings. To accelerate the conventional ray-casting, we propose an efficient ray casting method with subdivided bounding boxes which are based-on GPU quadtree traversal. The subdivision of the terrain's bounding box can reduce the empty spaces effectively. By performing the ray-casting with this compact bounding box, we can efficiently reduce computation with empty space skipping. Unlike the recent quadtree-based empty space skipping techniques which perform the tree traversal at each ray, our method traverses the tree only once per frame. Therefore, we can save much computational time.

Keywords : Height-field(고도필드), Terrain rendering(지형 렌더링), Ray-casting (광선투사법), Realtime rendering(실시간 렌더링)

접수일자 : 2011년 11월 10일 심사완료 : 2010년 12월 05일
교신저자(Corresponding Author) : 신병석
※ 이 논문은 2011년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임.(2011-0003842)
※ 이 논문은 2011년도 인하대학교의 지원에 의하여 연구되었음.

1. 서 론

고도필드(height field)는 높이값을 저장한 이미지 데이터로서 지형을 표현하는데 널리 사용된다. 고도필드를 통한 지형 시각화는 GIS(Geographic Information System)나 3D게임, 비행 시뮬레이션, 가상현실 등의 다양한 응용프로그램들에서 사용된다. 이들은 가상 환경의 기반이 되는 사실적인 장면을 실시간에 표현해야 한다.

GPU의 발달로 인하여 지형 시각화에 광선투사법을 적용하여도 실시간에 사실적인 영상을 만들어 낼 수 있게 되었다[1,2,3,4]. 이 방법은 별도의 메쉬 없이 렌더링이 가능하여 많은 그래픽 메모리가 필요하지 않다. 또한 GPU만을 이용하기 때문에 CPU가 다른 연산에 전념할 수 있도록 해준다. 하지만 지형 데이터가 대용량화 되고 디스플레이 기술의 발전으로 HD급 고화질 영상이 요구되면서 실시간 렌더링을 위한 새로운 가속화 기법이 필요하게 되었다.

사진트리를 이용한 빈 공간 도약기법[2,3]은 광선투사법의 대표적인 가속화 방법이다. 이것은 광선의 경로 위에 있는 모든 텍셀에서 광선과 지형표면과의 교차여부를 검사해야했던 기존 방법의 연산량을 줄이기 위하여 불필요한 텍셀을 건너뛰는 방법이다. 하지만 사진트리기반의 빈 공간 도약 기법은 최악의 경우에는 빈 공간을 도약하지 않는 광선투사법보다 비효율적이다. 이는 일반 광선투사법에서 한번의 샘플링만으로 찾을 수 있는 표면도 루트노드부터 트리를 하향식으로 탐색하여 찾아야 하기 때문이다. 이러한 단점을 해결하기 위하여 본 논문에서는 사진트리를 이용한 바운딩 박스 세분화 기법으로 광선투사를 가속화하는 방법을 제안한다.

기존의 방법이 프래그먼트 쉐이더(fragment shader)에서 각 광선별로 트리탐색 연산이 이루어진 반면 본 논문에서 제안하는 방법은 기하 쉐이더(geometry shader)를 이용하여 트리 탐색을 한다. 기하 쉐이더의 스트림 출력(stream output)기능은 입력된 정점 집합을 다시 메인 메모리로 돌려보내는 것이 가능하다[5]. 따라서 바운딩 박스를 기하 쉐이더에서 세분화하고 스트림 출력을 이용하여 다시 입력값으로 바인딩 시키는 방법을 이용하여 트리를 탐색의 재귀연산을 수행할 수 있다. 트리를 탐색할 때 기존 방법에서는 각 광선별로 탐색해야 했던 반면에 제안하는 방법은 루트부터 말단노드까지 단 한번의 연산으로 트리 탐색을 완료할 수 있는 장점이 있다. 또한 기존 방법이 광선과 각 트리 레벨의 바운딩 박스들의 교차점을 계산해야 하는 반면 제안하는 방법은 래스터화 단계를 이용하여 광선의 시작점만 찾아주면 되기 때문에 기존의 방법에 비하여 전체적인 연산량을 줄일 수 있다.

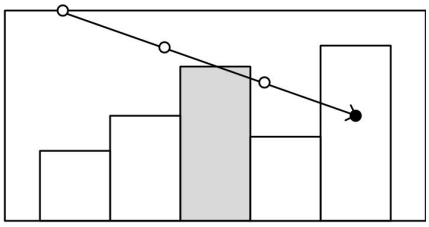
2장에서는 관련 연구들을 소개하며 3장에서는 본 방법에 대한 구체적인 설명을 한다. 4장에서는 실험결과를 보이며 5장에서 결론을 맺는다.

2. 관련 연구

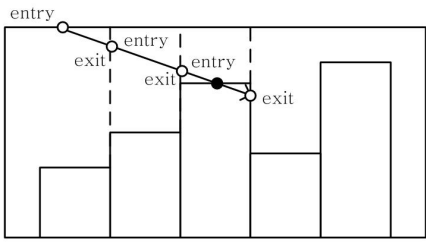
기존 지형 시각화에서는 폴리곤 기반의 메쉬 재구성(reconstruction) 기법을 사용하였다[6,7]. 하지만 이 방법들은 메쉬 간략화 등의 작업을 CPU에서 수행해야 했으며 처리 가능한 프리미티브 개수에 제한이 있다. 이러한 문제점들은 광선투사법을 이용하여 해결할 수 있다. 광선투사법은 GPU만으로 수행할 수 있으며 기하 정보를 재구성하지 않기 때문에 원본 데이터를 저장하는 공간 외에 추가적인 메모리공간을 필요로 하지 않는다. 따라서 GPU가 처리할 수 있는 다각형의 개수를 초과하는 대용량의 데이터를 광선투사법으로 렌더링 할 수 있다.

지형 렌더링을 위한 광선투사법에는 등간격으로 샘플링하는 방법과 각 텍셀마다 교차 검색을 수행하여 찾아내는 방법이 있다. 등간격 샘플링을 이용한 방법은 [그림 1]의 (a)와 같이 광선의 시작점부터 종료점까지 일정간격으로 샘플링하며 지형과의 교차점을 찾는 방법이다. [그림 1]에서 막대는 하나의 텍셀을 의미한다. 이 방법은 비교적 연산이 간단하고 수행속도가 빠르나 샘플링 간격이 넓을수록

부정확한 결과가 나온다. [그림 1]의 (a)에서 광선과 처음으로 교차하는 회색 막대에 해당하는 지형 표면은 찾지 못하고 검은색 샘플점에 도달해서야 지형표면과 교차함을 발견할 수 있다. 이것은 결과 영상에 잘못된 색상이 나오거나 깜빡임 현상이 나타날 수 있다. 반면 [그림 1]의 (b)처럼 텍셀마다 교차점을 검색을 수행하는 방법은 광선의 경로 위에 있는 모든 텍셀과의 교차점을 검사해야하므로 연산량은 많으나 정확한 교차점을 탐색할 수 있다.



(a)



(b)

[그림 1] (a)정규 샘플링을 이용한 광선투사법과 (b)텍셀별 교차점 검색을 통한 광선투사법

하지만 위의 광선투사법은 간략화작업을 하지 않기 때문에 지형데이터가 클수록 연산량이 많아 실시간처리가 어렵다. 이러한 문제점을 해결하기 위해 여러 가지 가속화 기법들이 제안되었다. 이진 탐색 기법[8]은 속도의 향상은 크지만 등간격 샘플링을 이용한 광선투사법과 같이 정확한 교차점을 찾기 힘들다는 문제점을 가지고 있어 정확한 영상을 만들어내기 어렵다. 시차차단매핑(parallax occlusion mapping)은 선형 보간된 시차의 변위를 이용하여 렌더링 하는 기법이다[9]. 하지만 이 방법은 높이차가 심한 험준한 지역에서 많은 오차를 보인다. Tatarchuk는 이 방법을 확장한 동적시차차단매핑(dynamic parallax occlusion mapping)[10]을 통하여 화질을 개선하였으나 완벽하게 오차를 제거하지 못하였다.

원뿔스텝매핑(Cone Step Mapping)은 지형에 수직으로 접하는 원뿔과 광선의 교차점을 탐색하는 기법으로 오차 없이 빠른 속도로 렌더링 할 수 있다[11]. Donnelly가 제안한 거리맵(distance map)을 이용한 방법[12]은 구추적기법(sphere tracing)을 이용하여 빠른 렌더링이 가능하다. 하지만 이 방법들은 데이터 생성과정의 연산량이 많아 전처리 시간이 오래 걸리는 단점이 있다.

Tevs는 이러한 단점을 해결하기 위하여 사진트리를 이용한 Oh의 방법[2]에 기반하여 GPU에서 사진트리를 생성하고 탐색하는 방법을 제안하였다[3]. 이 방법에서 사진트리의 각 노드들은 하위노드들의 최대 높이값을 저장하고 있으며 각 광선이 루트노드에 해당하는 바운딩 박스와 교차할 경우 다음 레벨에 해당하는 세분화된 바운딩 박스들과의 교차여부를 검사하는 과정을 재귀적으로 반복하여 지형 표면과 광선의 최종 교차 지점을 구한다. 이 모든 과정을 GPU에서 수행하기 위해 사진트리를 텍스쳐 피라미드 형식으로 저장한다. Dick은 이 가속기법에 몇가지 최적화 방법을 추가하여 수백 기가바이트급의 대용량 지형을 외부 메모리에서 처리하여(out-of-core) 실시간에 렌더링 할 수 있게 하였다[1].

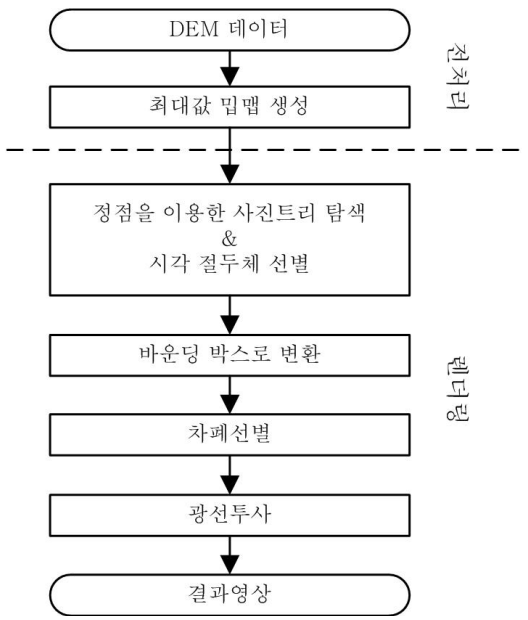
기존의 CPU를 활용한 방법으로는 지형의 바운딩 박스를 사진트리를 이용하여 세분화 한 후 그 깊이정보를 이용하여 빈 공간을 도약하는 방법이 있다[13]. 이것은 GPU로 집중된 연산량을 CPU를 이용하여 줄여준다. 하지만 이 방법은 순차적인 처리를 해야 하기 때문에 GPU가 발달됨에 따라 병렬 프로세싱을 사용한 GPU기반 가속화 기법들에 비하여 처리속도가 느리다.

본 논문에서 제안하는 방법은 기존의 CPU에서 메쉬를 생성하여 가속화 하는 방법들과는 달리 GPU에서 사진트리를 탐색하고 탐색한 결과를 이용

하여 빈 공간을 줄인 바운딩 박스를 생성한다. 그리고 이 바운딩 박스의 깊이값을 이용하여 효과적으로 빈 공간을 도약한다.

3. 바운딩 박스의 세분화를 이용한 빈 공간 도약기법

본 절에서는 대용량 DEM데이터를 렌더링하기 위하여 정점 증식을 이용한 빈 공간 도약 기법에 대해 설명한다.



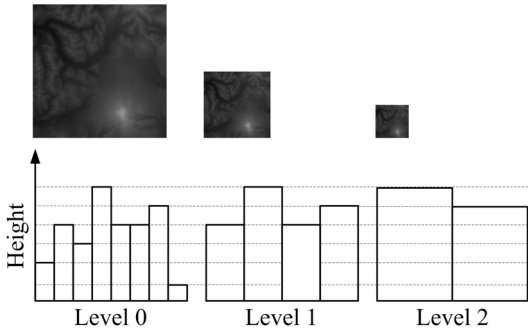
[그림 2] 제안하는 방법의 처리절차

우선 전처리 과정에서 공간 도약을 위한 자료구조로 사용되는 사진트리를 최대값 밉맵[3]으로 생성한다. 이것은 GPU에서도 트리구조를 사용할 수 있도록 텍스쳐 계층으로 사진트리를 저장한 것이다. 이 사진트리는 렌더링 과정에서 빈 공간 도약을 효과적으로 하기 위해 자식 노드들의 높이값들 중 최대값을 각 노드에 저장한다.

렌더링 과정에서는 GPU를 이용하여 효과적으로 트리를 탐색하기 위하여 정점 증식을 이용한다. 지형의 중앙에 위치한 루트노드에 해당하는 한 정점은 시점과의 거리기반 LOD와 시각 절두체 선별과정을 거쳐 하위노드 탐색이 필요하다고 판단되면 자식노드에 해당하는 4개의 정점으로 증식된다. 이러한 과정을 말단노드까지 반복한 후 각 정점들을 최대 높이값을 갖는 바운딩 박스로 변환한다. 이 바운딩 박스들을 이용하여 효과적으로 빈 공간을 도약하는 광선투사를 수행할 수 있다. 하지만 기존의 단일 패스 광선투사법을 사용할 경우 여러 바운딩 박스들의 뒷면이 하나의 픽셀에 중복하여 매핑되기 때문에 중복된 광선투사 연산을 한다는 문제점이 있다. 이러한 문제점을 해결하기 위해 차폐선별을 수행하는 패스를 추가하였다.

3.1 최대값 밉맵

최대값 밉맵은 각 노드에 자신의 영역을 표현하는 바운딩 볼륨에 대한 정보를 저장하여 광선의 진입점과 종료점을 계산할 수 있도록 한 자료구조이다[3]. 트리의 각 노드들은 자식 노드들의 최대값만을 저장하여 [그림 3]의 하단과 같이 바운딩 볼륨의 높이를 저장한다. Level 0은 원본 고도필드 데이터이며 bottom-up방식으로 상위레벨을 생성한다. 최대값만을 저장하는 이유는 지형 데이터는 뒷면을 렌더링 할 필요가 없고 광선과 표면의 교차점을 찾는 동시에 렌더링 연산이 종료되므로 최소값은 연산에서 필요로 하지 않기 때문이다.



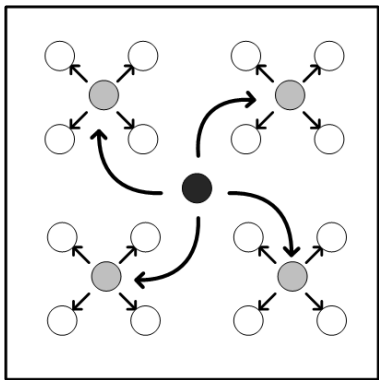
[그림 3] 텍스쳐 피라미드로 구현된 사진트리

제안하는 사진트리는 GPU에서 사용하기 위하여 [그림 3]의 상단과 같이 텍스쳐 피라미드 형태로 저장된다. 최대값 밉맵은 GPU에서 상향식(bottom-up)으로 생성할 수 있기 때문에 CPU에서 순차적으로 생성하던 사진트리보다 전처리 시간이 짧은 장점이 있다[3].

3.2 정점을 이용한 사진트리의 탐색

여기서는 지형의 바운딩 박스를 세분화하여 광선투사법을 가속화하는 방법을 제안한다. 바운딩 박스를 세분화 하면서 최대값 밉맵을 참조하여 박스에 높이값을 적용하면 빈 공간이 줄어들게 되고 이것은 광선투사에서 불필요한 연산을 줄여준다.

GPU를 이용하여 바운딩 박스를 세분화하기 위해서 본 논문에서는 사진트리를 이용하여 메쉬 생성을 가속화하는 알고리즘인 정점증식기법[14]을 이용한다. 이 방법은 [그림 4]와 같이 하나의 정점을 4개로 증식시키며 트리를 탐색하는 방법으로 GPU의 병렬처리를 통하여 단 1번의 트리탐색으로 원하는 결과를 얻을 수 있다. 또한 바운딩 박스 대신 정점을 사용하기 때문에 메모리 소모가 적으며 연산속도가 빠른 장점이 있다.



[그림 4] 정점 증식 기법 (중앙의 검정색 정점이 루트노드이며 4개씩 증식하여 하위레벨을 탐색한다)

본 방법은 정점의 특정 위치가 곧 트리의 노드를 의미한다. 루트노드인 정점은 바운딩 볼륨의 중심에 위치하고 트리 탐색시 그 점을 기준으로 4분할된 영역의 중점에 자식노드에 해당하는 정점들을 증식시킨다. 자식노드로 증식한 부모정점은 증식 후에 자신을 제거하여 자식정점들만을 남겨 바운딩 박스를 세분화 한다. 이러한 과정을 반복하여 탐색을 마친 후 최종적으로 각 정점을 바운딩 박스로 변환하여 빈 공간을 줄인 바운딩 박스집합(bounding box set)을 생성한다.

3.3 사진트리를 이용한 LOD

트리의 탐색은 다음의 두가지 조건을 만족할 경우 수행된다. 첫 번째는 본 논문에서 제안하는 지형과 시야와의 거리가 특정 조건에 부합하는지를 검사하며 두 번째는 시각 절두체 안에 바운딩 박스가 있는지를 검사하여 결정한다.

3.3.1 거리기반의 LOD

시점과 지형과의 거리를 이용한 방법은 3.1절에서 설명한 텍스쳐 피라미드를 이용해서 효과적인 LOD를 수행할 수 있다. 지형이 시점에서 멀어질수록 화면 공간상의 한 픽셀에 지형데이터의 여러 텍셀들이 매핑될 수 있다. 이 경우 모든 텍셀을 샘플링하는 것보다 간략화된 상위레벨의 밉맵을 사용하면 샘플링할 텍셀 수를 줄일 수 있으므로 속도를 향상시킬 수 있다. 제안하는 방법에서는 4개의 텍셀이 화면공간상의 한 픽셀에 매핑 될 경우 트리를 탐색시켰다.

이 LOD방법을 사진트리 탐색에 적용하기 위해서는 우선 한 바운딩 박스당 얼마만큼의 해상도를 갖는 지형데이터를 표현해 줄 것인지를 정해야한다. 바운딩 박스당 표현할 지형데이터의 해상도를 τ라고 할 때 제안하는 방법에서는 아래와 같은 수식으로 트리를 탐색한다. 이 사진트리는 하향식(top-down)으로 탐색한다.

$$f = \begin{cases} 1 & S(a') \geq 4^{*} S(\tau_p) \\ 0 & S(a') < 4^{*} S(\tau_p) \end{cases}$$

여기서 f는 사진트리의 탐색 여부를 결정하는 플래그이고 S는 특정영역의 넓이를 구하는 함수이다. 이 수식은 바운딩 박스의 바닥면(a)을 화면공간상으로 투영하여 투영된 사각형(a')의 넓이($S(a')$)와 τ와 같은 크기의 픽셀 영역 τ_p의 넓이($S(\tau_p)$)와 비교하여 탐색여부를 결정한다. 루트노드는 단 1개의 텍셀을 갖는 텍스쳐 피라미드부터 시작한다. 이 루트노드의 $S(a')$가 $S(\tau_p)$보다 4배 이상 크다면 화면공간상에서 한 텍셀이 4개 이상의 픽셀에 매핑되는 것이므로 사진트리를 계속 탐색하여 더 정교한 데이터를 쓰도록 바운딩 박스를 세분화 한다. 반대로 $S(a')$가 $S(\tau_p)$의 4배 미만일 경우 화면공간상에서 인접한 4개의 텍셀이 한 픽셀에 매핑되지 않는 경우이기 때문에 더 이상 탐색 할 필요 없이 트리 탐색을 종료한다.

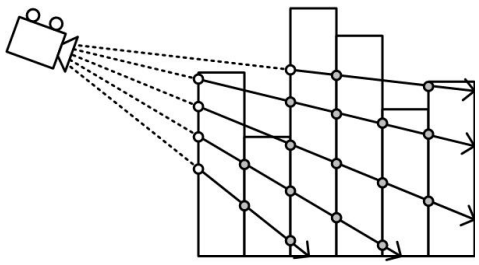
3.3.2 시각 절두체 선별

3D게임이나 가상현실에서는 일반적으로 지형 전체를 한 번에 바라보는 경우가 드물며 보통 지형 표면 가까이에서 지평선 방향으로 바라보는 시점을 갖는 경우가 대부분이다. 따라서 시야 바깥쪽에 있는 바운딩 박스들은 삭제를 하여 전체적인 렌더링 속도를 향상시킬 수 있다.

제안하는 방법에서는 3.3.1절에서 설명한 f에 대한 연산을 수행하기 전에 해당 바운딩 박스가 시각절두체 내부에 있는지에 대한 판단을 하여 바운딩 박스가 겹치는 부분 없이 외부에만 존재할 경우 해당 노드를 삭제하였다. 이것은 시야 밖에 존재하는 노드들에서 수행할 연산을 사전에 차단하여 효과적으로 가지치기(pruning)을 수행하였다.

3.4 차폐 선별을 이용한 광선투사법

정점 증식을 통하여 생성한 바운딩 박스를 이용하여 단일패스 광선투사법[1,7]을 진행할 경우 [그림 5]와 같은 문제가 발생한다. 광선투사법은 GPU에서는 바운딩 박스들을 프래그먼트 단위로 래스터화한 후 각각의 프래그먼트에서 광선을 투사한다. [그림 5]의 흰 점들은 정상적인 광선의 진입점이며 회색점들은 실제로는 투사할 필요가 없는데 투사가 되는 광선들의 진입점이다. 따라서 박스가 많아질수록 중복된 광선의 숫자도 많아지고 이로 인해 속도가 크게 저하된다. 이러한 문제를 해결하기 위하여 본 논문에서는 차폐된 프래그먼트에 대해서 연산을 수행하지 않도록 프래그먼트의 차폐여부를 검사하는 단계를 추가하였다.



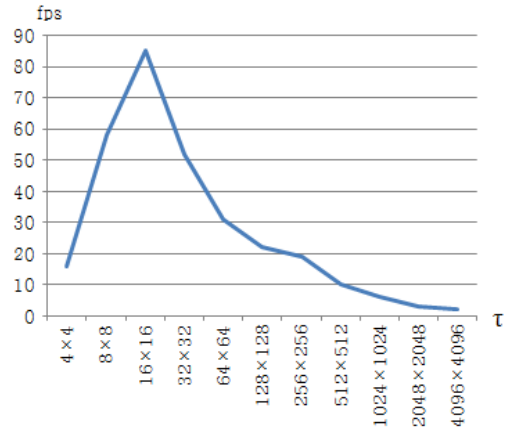
[그림 5] 바운딩 박스를 적용한 경우의 문제점

첫 패스는 깊이 테스트가 가능하도록 설정한 후. 기존의 광선투사법과 마찬가지로 박스들의 뒷면을 렌더링 한다. 이때 뒷면엔 고도필드의 u,v,w 좌표값과 해당 바운딩 박스의 레벨을 R,G,B,A 채널에 각각 저장한다. u,v,w 좌표값은 시각좌표와의 차이를 통하여 광선의 방향을 알아내는데 사용된다. 그리고 바운딩 박스의 레벨값은 해당 u,v,w 값을 둘러싼 박스의 앞면을 계산하여 광선의 진입점을 구하는데 사용된다. 이렇게 하면 깊이 테스트를 통해 차폐된 프래그먼트들은 렌더링 되지 않는다. 두 번째 패스에서는 차폐 선별된 이전 패스의 결과물을 이용하여 렌더링하기 위하여 루트노드에 해당하는 바운딩 박스의 뒷면을 렌더링한다. 루트 노드만 사용할 경우 단 하나의 바운딩 박스만 생성하며 이것은 픽셀당 하나 이하의 프래그먼트들만 매핑이 되며 이것은 각 픽셀당 단 하나만의 광선을 투사할 수 있게 해준다. 그러나 루트노드만 사용할 경우 빈 공간에 해당하여 연산을 수행할 필요가 없는 프래그먼트들 또한 생성된다. 따라서 이전 패스의 결과 영상에서 값이 0인 프래그먼트들은 광선을 투사하기 전에 연산을 종료시킨다.

4. 실험 결과

실험은 AMD PhenomTM II X2 545 Processor CPU 3.30GHz에 DDR3 4GB의 주 메모리를 갖는 시스템에서 수행하였다. 그래픽 장치는 1GB의 비디오메모리를 갖는 ATITM Radeon HD5870을 사용하였고 그래픽 라이브러리는 DirectX 11를 사용하였다. HD급 고화질 렌더링을 위해 뷰포트 크기는 1600×900으로 설정하였다. 데이터셋은 40962의 해상도를 갖는 16비트 DEM데이터인 Puget Sound를 사용하였다.

제안하는 방법의 효율성을 입증하기 위해서 비교 대상으로 가속화 기법이 적용 되지 않은 GPU 기반 광선투사법[4]과 프래그먼트 쉐이더에서 최대값 밉맵을 이용하여 빈 공간을 도약하는 방법[3]을 선택하였다. 보다 정확한 비교를 위해 트리탐색을 수행할 때 제안하는 방법에서 사용하는 LOD방법과 같은 방법을 프래그먼트 쉐이더를 이용한 빈 공간 도약기법에 적용하였다.

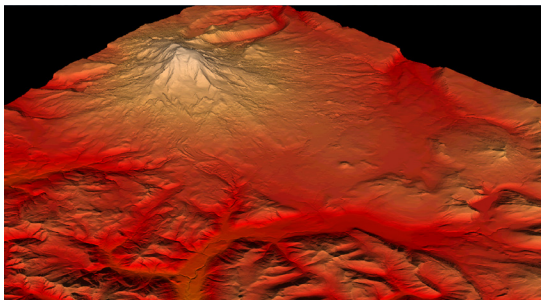


[그림 6] 제안하는 방법의 렌더링 속도

[그림 6]은 한 바운딩 박스가 표현할 지형 데이터의 해상도인 τ에 따라 제안하는 방법의 성능을 측정한 결과를 보인다. 실험결과 제안하는 방법은 하나의 바운딩 박스에서 16×16의 데이터를 처리할 경우 가장 우수한 성능을 보였다. 하나의 바운딩 박스에서 처리하는 데이터가 작을수록 트리를 깊게 탐색하기 때문에 속도가 빨라졌으나 8×8 이하의 데이터를 처리할 경우엔 느려졌다. 이것은 데이터가 작아질수록 바운딩 박스를 구성하는 기하 데이터의 양이 증가하여 GPU의 병렬처리 성능을 초과하였기 때문이다.

[표 1] 제안하는 방법과 비교실험과의 fps비율

τ	A:B	A:C
4×4	8.0	0.89
8×8	29.0	3.22
16×16	42.5	4.72
32×32	26.0	2.89
64×64	15.5	1.72
128×128	11.0	1.22
256×256	9.5	1.07
512×512	5.0	0.55
1024×1024	3.0	0.33
2048×2048	1.5	0.16
4096×4096	1.0	0.111



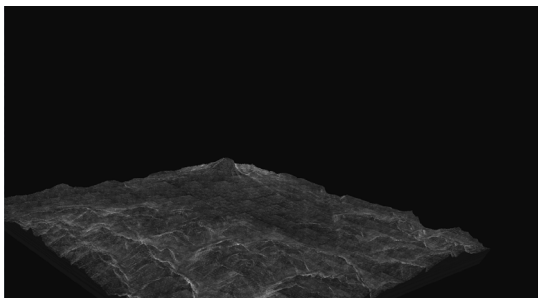
[그림 7] 실험에서 사용된 조건으로 렌더링한 결과영상

[표 1]은 제안하는 방법과 기존 방법들과의 성능을 비교한 것이다. 여기서 A는 제안하는 방법이며 B는 기본적인 광선투사법 C는 최대값 밉맵을 프래그먼트 쉐이더에서 사용한 방법이다. 가속화 기법이 적용되지 않은 단일 패스 광선투사법과 프래그먼트 쉐이더 기반의 빈 공간 도약기법의 렌더링 속도는 같은 조건에서 각각 2fps와 18fps로 측정되었다. 이 실험은 모두 동일한 조건에서 측정한 것으로서 [그림 7]과 같은 결과영상이 만들어진다. 4096×4096일 경우 지형데이터와 같은 크기이기 때

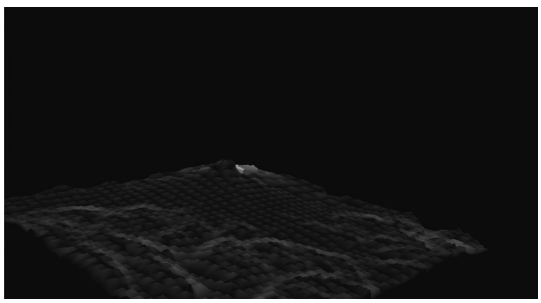
문에 트리 탐색을 하지 않아 가속화가 적용되지 않은 단일 패스 광선투사법인 B와 속도가 같으며 가속화가 적용된 방법인 C보다 느리다. 작은 τ를 사용하여 트리 탐색을 할수록 B를 최대 42.5배 가속화 하며 기존의 가속화 기법인 C에 비하여 최대 4.72배 빠르다. 하지만 GPU가 병렬로 한 번에 처리할 수 있는 기하데이터양을 초과하면 4×4와 같이 C보다 느려진다. 최적의 속도를 보인 16×16에서는 시점을 이동하였을 때 최대 190fps의 속도를 보였으며 평균 92fps 의 속도를 보였다. 이는 동일 측정 조건에서 B의 방법보다 평균 약 46배, C의 방법보다 평균 약 6배정도 빠른 속도이다.

[그림 8]은 비교실험들과 제안하는 방법의 도약 횟수이다. 흰색이 강할수록 큰 도약횟수를 보인다. 여기서 τ는 최적의 성능을 보이는 16×16으로 하였다. 가속화기법을 수행하지 않았을 경우에는 광선과 지형데이터와의 교점을 찾기 전까지 광선의 경로상에 있는 모든 샘플점들을 한 텍셀단위로 도약하며 샘플링 해야 하기 때문에 시야에서 멀어질수록 도약횟수가 많다. 그러나 제안하는 방법과 프래그먼트 쉐이더를 이용한 빈 공간 도약기법의 경우에는 불필요한 부분은 사진트리 탐색을 통하여 건너뛰기 때문에 [그림 8]의 (b)와 (c)같이 도약횟수가 전체적으로 많이 감소하였다. 하지만 프래그먼트 쉐이더를 이용한 가속화 방법의 경우엔 각각의 도약에 바운딩 박스와 광선과의 교점을 구하는 연산과 LOD연산이 추가되기 때문에 한번 도약할 때마다 수행해야 하는 연산이 많다. 제안하는 방법의 도약은 가속화 기법을 사용하지 않는 방법과 동일하게 진행이 되기 때문에 프래그먼트 쉐이더 기반의 빈 공간 도약기법보다 연산량이 적으며 트리탐색 또한 전체적으로 광선투사과정 이전에 한번만 수행되기 때문에 도약횟수도 적다.

(a)



(b)



(c)

[그림 8] 수행된 광선의 도약횟수. (a)는 가속화 기법없이 광선투사법만 사용했을 경우 (b)는 프래그먼트 쉐이더를 이용한 빈 공간 도약기법을 적용한 경우 (c)는 제안하는 방법을 이용한 경우이다.

5. 결 론

본 논문에서는 지형 광선투사법을 시점기반의 LOD를 기반으로 한 바운딩 박스의 세분화를 이용하여 가속화 하는 방법을 소개하였다. 이 방법은 기존의 방법들이 빈 공간 도약을 위해 각 광선단위로 트리탐색을 여러 번 수행하던 것을 기하 쉐이더를 이용하여 단 1번만 수행하여 연산을 가속

화 하였다. 또한 광선투사 도중 바운딩 박스와의 교차점을 구하기 위한 추가적인 연산 없이 GPU의 래스터화 단계를 이용하여 광선의 시작점과 종료지점을 측정하였고 추가적인 렌더링 패스로 중복된 광선들이 생기지 않도록 하였다. 이것은 최적의 바운딩 박스를 사용했을 경우 평균 46배 빠른 속도를 보이며 기존의 가속화 방법보다도 평균 6배정도 빠른 속도를 보였다.

하지만 제안하는 방법은 트리를 너무 깊게 탐색을 하여 GPU의 성능을 초과하는 기하 데이터를 생성할 경우 처리속도가 느려지는 단점이 있다. 추후 이러한 단점을 극복하기 위해 바운딩 박스의 세분화와 프래그먼트 쉐이더 기반의 빈 공간 도약 기법을 혼합하여 적은 기하 데이터로 효과적인 빈 공간을 도약을 수행하는 방법을 연구할 것이다.

감사의 글

이 논문은 2011년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임.(2011-0003842)

이 논문은 2011년도 인하대학교의 지원에 의하여 연구되었음.

참고문헌

[1] C. Dick, J. Krüger, R. Westermann, “GPU Ray-Casting for Scalable Terrain Rendering”, In Proc. of Eurographics 2009, pp. 43-50, 2009.

[2] K. Oh, H. Ki, C. Lee, “Pyramidal displacement mapping: a GPU based artifacts-free ray tracing through an image pyramid”, In Proc. of the ACM symposium on Virtual Reality Software and Technology, pp 75-82, 2006.

[3] A. Tevs, I. Ihrke, H. P. Seidel, “Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering”, In Proc. of the ACM Symposium on Interactive 3D Graphics and Games, pp. 183 - 190, 2008.

[4] Microsoft: DirectX Software Development Kit. http://www.microsoft.com/directx, Nov 2008. “RaycastTerrain Sample”.

[5] D. Blythe, “The Direct3D 10 System”, ACM, 2006.

[6] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, G. Turner, “Real-time, Continuous Level-of-Detail Rendering of Height Fields”, In Proc. of ACM Siggraph 96, pp. 109-118, 1996.

[7] M. Duchaineau, M. Wolinsky, D. Sigeti, M. Miller, C. Aldrich, M. Mineev-Weinstein, “ROAMing terrain: real-time optimally adapting meshes”, In Proc. of Visualization ’97, pp.81-88 ,1997.

[8] F. Policarpo, M. M. Oliveira, J. Comba, “Real-Time Relief Mapping on Arbitrary Polygonal Surfaces”, In Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 359-368. 2005.

[9] Z. Brawley and N. Tatarchuk, “Parallax occlusion mapping: Self-shadowing, perspective-correct bump mapping using reverse height map tracing”, In ShaderX3: Advanced Rendering with DirectX and OpenGL, pp 135-154, 2004.

[10] N. Tatarchuk, “Dynamic parallax occlusion mapping with approximate soft shadows”, In Proc. of the ACM Symposium on Interactive 3D Graphics and Games, pp. 63-69, 2006.

[11] J. Dummer, “Cone step mapping:An iterative ray-heightfield intersection algorithm”, http://www.lonesock.net/files/ConeStepMapping.pdf, 2006.

[12] W, Donnelly, “Per-Pixel Displacement Mapping with Distance Functions”, GPU Gems, vol. 2. Addison-Wesley, 2005,

[13] Z. Fehér, “Terrain Rendering with the Combination of Mesh Simplification and Displacement Mapping”, In Proc. of CESCG 2010: The 14th Central European Seminar on Computer Graphics, 2010.

[14] E.S. Lee, B.S. Shin, “Geometry Splitting: An Acceleration Technique of Quadtree-Based Terrain Rendering Using GPU”, IEICE TRANSACTIONS on Information and Systems, Vol. .E94-D, No.1 pp.137-145, 2011.

이 은 석 (Lee, Eun-Seok)

2008년 2월 인하대학교 컴퓨터공학부 (학사)
2010년 8월 인하대학교 컴퓨터정보공학부 (석사)
2011년 2월-현재 인하대학교 컴퓨터정보공학부 (박사)

관심분야 : 실시간렌더링, 상세단계선별, 차세대컴퓨팅

이 진 희 (Lee, Jin-Hee)

2005년 2월 한국방송통신대학교 컴퓨터과학과 (학사)
2007년 2월 인하대학교 컴퓨터정보공학부 (석사)
2007년-현재 인하대학교 컴퓨터정보공학부 (박사)

관심분야 : HCI, 차세대컴퓨팅, 실시간렌더링

조 인 우 (Jo, Inwoo)

2010년 2월 인하대학교 컴퓨터공학부 (학사)
2010년-현재 인하대학교 컴퓨터정보공학부 (석사)

관심분야 : 지형렌더링, 상세단계선별

신 병 석 (Shin, Byeong-Seok)

1990년 2월 서울대학교 컴퓨터공학과 (학사)
1992년 2월 서울대학교 컴퓨터공학과 (석사)
1997년 2월 서울대학교 컴퓨터공학과 (박사)
2000년-현재 인하대학교 컴퓨터정보공학부 교수

관심분야 : 볼륨그래픽스, 차세대컴퓨팅, 실시간렌더링