2005 한국경영과학회/대한산업공학회 춘계공동학술대회
2005년 5월13일~14일, 충북대학교

Regular Grid Method에 기반한 실시간 지형 가시화 알고리즘 개발

Real-time 3D terrain visualization based on Regular Grid Method

정지찬 박준영
동국대학교 산업시스템공학과

Abstract

실시간 가시화/시각화(Visualization)을 위한 지형 가시화 분야에서의 렌더링 속도 향상은 사용자의 현실감에 있어 중요한 역할을 한다. 본 연구는 Height Field 데이터를 사용한 블록 단위 지형 렌더링 방법에서 이전 프레임 정보를 사용하여 지형 가시화 속도를 향상시키는 방법을 제안한다.

Height Field로 표현된 지형을 실시간에 효과적으로 렌더링 할 수 있는 방법으로CLOD(Continuous leveI Of Detail)가 있으며 대표적인 방법으로 Multiresolution TIN(Triangle Irregular Network)과 Regular Grid Method에 기반한 방법들이 있다.

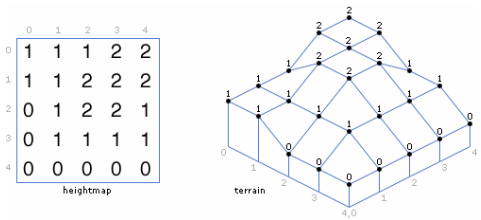
대개의 경우, 지형 데이터는 매우 방대한 크기를 가지고 있어서 실시간으로 렌더링 하는 것이 불가능할 경우가 많다. 따라서 실시간 지형렌더링 에서는 LOD(Level of Detail)관리와 뷰 프러스텀 컬링이 실시간 렌더링 속도 향상을 위한 핵심 사항이 된다.

본 연구에서는 PC 기반에서 효과적으로 지형을 표현하기 위해 기존 Regular Grid Method에 기반한 방법의 가시영역 추출(View Frustum Culling)을 수정하여 실시간 지형 렌더링의 가시 영역 추출(View Frustum Culling)시 기존의 쿼드트리 방식과 함께 이전 프레임에서 블록 단위로 저장된 컬링 정보를 혼용하여 속도를 향상시키는 방법을 상세히 기술 한다.

1. 서론

최근의 3D 렌더링 하드웨어의 발전 덕분에, 이제는 많은 VR 개발자들이 야외 환경 즉 지형분야에 관심을 많이 기울이고 있다. 현실과 유사한 이러한 3차원 가상 물리적 공간은 사용자로 하여금 더욱 게임에 몰입할 수 있도록 하는 요소가 된다. 3차원 가상 공간 중에서도 폐쇄되어 있는 좁은 공간이 아니라 자연경관을 효과적으로 보여줄 수 있는 방법에 대해서는 VR이나 비행 시뮬레이션 분야에서 많은 연구가 되어 있는데 아직까지 PC환경에 적용하기에는 다소 무리가 있는 부분도 있다. 가상환경에서는 화면에 보여지는 것뿐만 아니라 가상 공간상에 존재하는 많은 객체간에 상호 작용 또한 중요한 요소이기 때문에 대다수 가상 가시화의 경우 보여주는 요소에 많은 자원을 할당하기보다는 다른 요소들을 우선으로 하는 경우가 많다. 따라서 PC기반 가상환경에서는 지형을 그리는데 많은 CPU 자원을 할당하지 않는 방법을 사용해야 한다.

이와같이 PC기반 가상환경에서의 VR엔진은 실시간 렌더링에 필요한 최적화된 라이브러리 제작 툴을 제공해야 한다. 사실적인 영상의 실시간 렌더링에 필수적인 모듈은 그림자 생성, 물/불/연기 등 자연현상을 위한 파티클 시스템, 포털 및 BSP 컬링, 출동 탐지, levels of detail(LOD)생성 및 처리, 지형 렌더링 등이 있다. 이같은 실시간 렌더링은 알고리즘을 통하여 최적화된 구현물을 제공해야 하고 그 가운데 가장 중요한 이슈중의 하나가 지형렌더링일 것이다.



(그림 1) Height Field 데이터

지형 표현 기법으로 가장 많이 쓰이는 것은, 그림 1에 도시된 바와 같이 광활한 지형을 효과적으로 표현하기 위해 일정한 간격으로 지형의 높이값을 저장한 Height Field 데이터 이다. Height Field 데이터는 지형의 높이를 일정한 간격으로 샘플링한 2차원 형태의 데이터로 저장되어 있다. 지형의 모든 버텍스(vertex) 정보를 저장하는 방법보다 단순하면서도 메모리를 절약할 수 있는 방법이다.

이러한 Height Field 데이터를 사용하여 지형을 현실감 있게 생성해 내는데 있어 문제가 되는 것은 폴리곤을 지형의 특정에 따라 적절하게 사용하여 현실감을 높이는 것이다. 거리에 따라 폴리곤의 수를 조절하는 LOD(Lelvel Of Detail)는 메쉬 형태로 미리 정의된 폴리곤을 사용하여 거리가 먼 경우 적은 폴리곤을 사용하는 모델을 사용하고, 가까울 경우 폴리곤이 많은 모델을 사용하여 선택적으로 렌더링 하는 방법이다. 이것은 거리에 따라 이산적으로 선택하는 방법이라고 할 수 있다. 하지만 Height Field 지형은

연속적으로 분포되어 있기 때문에 거리에 따라 이산적으로 선택할 수가 없다. 따라서 연속적인 지형 데이터에 적용할 수 있는 CLOD(Continuous LOD)라는 방법을 사용하게 된다. 이 방법은 Height Filed를 사용하여 거리와 지형의특성에 따라 적절한 폴리곤을 실행 시간에 만들어 내고 렌더링 하는 방법이다.

CLOD 기반 지형 렌더링 시스템은 VR이나 비행 시뮬레이션에서 많은 연구가 되어 있지만 이를 PC에 적용하기에는 아직 무리가 있다. 이를 해결하기 위한 대표적인 방법으로 Tin(Triangular Irregular Network)에 기반한 방법[14]과 RGM(Regular Grid Method)에 기반한 방법[13]으로 나눌 수 있다. 이 방법은 지형상에서 관측된 Height Field 데이터로부터 컴퓨터 그래픽스의 시각화 기법인 표면 모델링 기법을 이용하여 지형을 형상화 하는 것을 말한다.

1.1 연구의 목적

따라서 본 연구는 기존의 Regular Grid Method에 기반한 방법의 가시영역 추출(View Frustum Culling)을 사용자가 정한 블록 레벨에서부터 지형 폴리곤을 생성하도록 수정하여 매 프레임을 그릴 때마다 반복적으로 수행해야 하는 컬링(Culling)작업을 이전 프레임에서 결정되었던 컬링(culling)정보를 사용하여 실시간 지형 렌더링 시 속도를 향상시키는 것이 본 연구의 목적이다.

2. CLOD 지형 렌더링

지형 데이터를 표현하기 위해 Height Field 데이터를 사용하는 것은 아주 단순하면서도 메모리를 절약할 수 있는 방법이다. 그러나 예전 지형 가시화에서는 Height Field 데이터를 사용하더라도 지형의 특성이나 거리에 상관없이 샘플링 된 모든 점들을 버텍스로 사용하여 많은 폴리곤을 생성하고 있다. 그러나 지형은 특성상 평평한 지역도 있고 굴곡이 심한 지역도 있기 때문에 이러한 특성을 살리지 못하고 일괄적으로 폴리곤을 생성하는 방법은 낭비라고 할 수 있다. 따라서 지형의 특성과 거리에 따라 폴리곤의 수를 적절히 사용하도록 하는 LOD를 적용한다면 더 넓은 지형을 효과적으로 보여줄 수 있다.

LOD의 기본적인 개념은 물체가 멀리 있는 경우 그 물체를 대표하는 특징 이외에 별로 중요하지 않은 요소들을 제거하여 렌더링 파이프라인에 걸리는 부하를 줄이는 것이다.[1] 이것은 적절한 수의 폴리곤으로 구성된 모델을 미리 여러 개 만들어 놓고 거리에 따라 이산적으로 선택하는 방법이다.그러나 전체 모델을 하나의 대상으로 보고 적용하는 방법이기 때문에 전체 지형 모델에 적용할 수는 없다.

CLOD(Continuous Level Of Detail)는 앞에서 언급한 지형 표현에 LOD를 적용할 때 발생하는 문제점을 해결하는 방법으로 Height Field 지형 데이터로부터 지형의 특성과 거리에 따라서 서로 연속성을 가지는 폴리곤을 부드럽게 생성해 내는 방법이다.

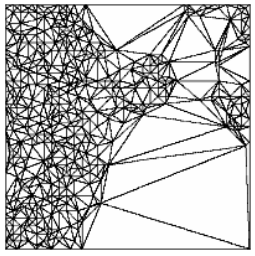
3. Height Field 지형 렌더링 관련연구

Height Field를 사용하여 지형을 표현하는 다양한 방법이 연구되어 있다. 대표적인 방법으로 Multiresolution TIN(Triangular Irregular Network) 방법과 Regular Grid Method에 기반한 CLOD방법이 있다.

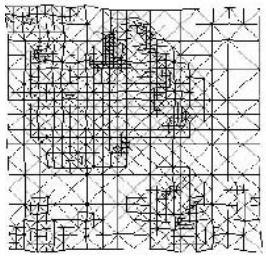
Multiresolution TIN 방법은 일반적으로 전처리 과정을 거쳐 서로 중복되지 않는 다양한 해상도의 폴리곤 집합(이것을 TIN이라고 한다)을 계층적으로 구성하고 실행 시간에 카메라와의 거리에 따른 적절한 해상도를 결정하여 지형 폴리곤의 해상도를 결정하는 방법이다. 그림 2는 TIN방법에 의해 생성된 지형 폴리곤을 표현한 것으로 보는 바와 같이 불규칙적인 폴리곤을 생성해 낸다.

Regular Grid Method에 기반한 방법은 그림 3에서 보는 바와 같이 전체 지형을 균일한 크기의 블록으로 보고 해상도가 낮아지는 경우 큰 블록을 사용하여 폴리곤을 생성하고 해상도가 높을 경우는 더 작은 블록으로 재분할하여 표현하는 방법이다. 이 방법은 TIN 방법보다는 폴리곤을 더 많이 생성하기 때문에 그리 주목을 받지 못했지만 텍스쳐 맵핑의 용이성이나 그 외에 여러 가지 면에서 유용한 점이 많다.

따라서 RGM 접근 방법이 가상현실에서의 지형



(그림 2) TIN방법에 의해 생성된 지형 폴리곤



(그림 3) RGM방법에 의해 생성된 지형 폴리곤

렌더링을 위해서는 더 나은 면을 많이 가지고 있다. PC기반 시스템의 경우 가속기에서 텍스쳐 크기가 일정 크기로 제한되는 경우가 있다. 그리고 텍스쳐 메모리의 크기도 제한되어 있기 때문에 될 수 있으면 텍스쳐의 사용량을 줄여야 한다.

전체 지형 데이터가 클 경우 지형에 입혀질 텍스쳐 한 장만 사용하게 되면 충분한 해상도를 얻기가 어렵다. 따라서 지형렌더링에서는 작은 블록 단위로 쪼개어 텍스쳐를 재사용 하는 경우가 많은데 RGM 방법이 여기에 적당한 방법이라고 할 수 있다. 또한 전체 지형 텍스쳐의 카메라 자유도를 제한하고 현재 보이는 부분에 대해서만 쪼개어 입혀는 방식을 사용하거나, 텍스쳐 소스를 연속적으로 사용할 수 있도록 타일로 제작하여 해당 블록에 입히는 방식을

사용한다. 이러한 방법은 텍스쳐 소스의 사용량을 줄이면서도 아주 효과적인 방법들이다.

그리고 지형과의 상호 작용도 중요한 요소인데 TIN 접근 방법은 전처리과정을 통해 미리 폴리곤 집합을 만들어 놓기 때문에 지형이 동적으로 변할 경우 실시간으로 처리하기가 어렵다.이 경우에도 역시 RGM 방식이 구현하기가 쉽다.

따라서 본 연구에서 구현하고자 하는 Height Field 지형 렌더링 시스템은 RGM 접근 방법을 기반으로 한다. 그리고 텍스쳐 맵핑을 용이하도록 기존 방법을 수정하여 블록 단위로 렌더링 하는 방법을 사용할 것이다.

4. 블록단위의 Height Field 지형렌더링

본 연구에서는 Stefan[17] 방법을 기본으로 하였으며 블록 단위로 텍스쳐를 입히기 쉽도록 전체 영역을 작은 블록으로 분할하는 방법을 제안한다. 블록으로 분할할 경우 지형 폴리곤을 생성하는 방법은 Stefan[17]의 기본적인 방법과 별로 차이는 없지만 사용자가 정한 블록 레벨에서부터 지형 폴리곤을 생성하도록 수정하였다. 그리고 속도를 향상시키기 위해 보이는 부분만을 추출하는 컬링 방법은 일반적으로 쿼드트리를 사용한 계층적인 방법을 사용한다.

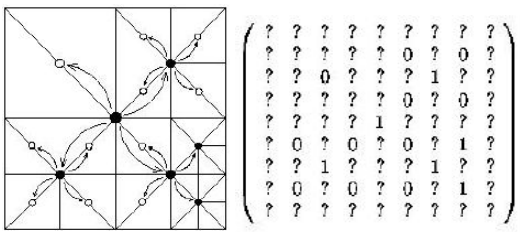
그러나 컬링시 속도의 향상을 위해 이전 프레임에서 결정된 컬링 정보를 재사용 하여 컬링 시 발생하는 부하를 줄이는 방법을 사용하였다.

4.1 기본적인 지형 폴리곤 렌더링 알고리즘

Height Field 지형 데이터는 지형의 높이를 2차원 배열에 저장해 놓은 형태이다.

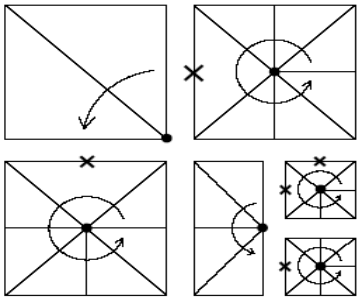
Stefan[17]에서는 전체 Height Field 데이터에 대해 쿼드트리를 사용하여 트라이앵글 팬을 생성한다.

기본적인 알고리즘의 데이터 구조는 그림 4 에서 보는 바와 같이 쿼드트리이며 데이터 사이즈는 $(2^n+1)\times(2^n+1)$로 n은 쿼드트리 레벨의 깊이이다. 쿼드트리의 표현은 블리안 매트릭스로 표현하며 쿼드트리의 각 중앙값이 1이면 그 노드는 더 작은 노드로 분할 되어야 함을 의미한다.



(그림 4) 블리안 매트릭스로 표현한 쿼드트리(n=3)

물음표 기호로 표시된 매트릭스 원소는 삼각형 구성을 계산하는 동안 셋팅시킬 필요가 없다. 왜냐하면 이러한 값들은 삼각형 구성시 하향식 알고리즘에서는 참조되지 않기 때문이다. 각 프레임에서 접근해야 하는 노드의 수는 Height Field 사이즈가 아니라 렌더링 질에만 의존하게 되며 필요로 하는 메모리 대역은 워하는 이미지의 질로 제한된다.



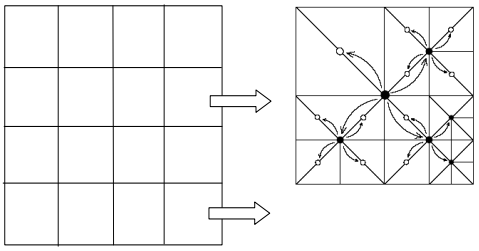
(그림 5) 트라이앵글 팬 구성

Height Field 데이터를 사용하여 폴리곤을 생성하는 방법은 그림 5와 같다. 삼각형 구성이 된 하이트필드는 대응되는 매트릭스 요소가 1로 셋팅되어 있는 부분만을 재귀적으로 쿼드트리를 순회함으로써 그려진다. 쿼드트리의 leaf에 도달할 때마다 전체 또는 부분적인 트라이앵글 팬이 그려진다. 트라이앵글 팬은 변화하는 해상도를 가지는 삼각형그리기에 적합하다. 인접한 블록이 다른 해상도를 가지고 있는 곳에서의 갭을 없애기 위해서 이러한 에지의 가운데 버텍스를 건너뜀으로써 적절한 메쉬가 생성된다 이 방법은 인접한 서브노드(그림 5)의 레벨이 단지 1만큼만 차이가 있을 경우만 동작한다.

트라이앵글 팬을 생성하는 동안 인접한 노드가 같은 레벨로 분할되는지 그렇지 않은지에 대해서 결정할 필요가 있다. 만약 이웃한 노드가 같은 레벨로 분할되지 않으면 우리는 공유된 에지의 가운데 버텍스를 건너뛸 수 있다. 이러한 경우는 이웃하는 노드에 대응되는 매트릭스 원소를 체크함으로써 찾아지는데 이것은 0으로 셋팅되어 있다. (값이 설정되어 있지 않은 매트릭스 원소는 제외됨을 주의하라 레벨간의 차이는 작거나 1과 같아야 한다.)

4.2 블록 단위 렌더링

앞에서 언급한 기본적인 알고리즘은 전체 지형 데이터에 대한 쿼드트리를 사용하여 계산하는 방법이다. 하지만 전체 지형에 대해 텍스쳐를 사용할 경우 텍스쳐 메모리의 한계를 넘어설 수도 있고 블록 단위로 텍스쳐를 사용한다 해도 맵핑에 필요한 적절한 버텍스 확보를 설정할 수가 없다.



(그림 6) 전체 지형의 블록화

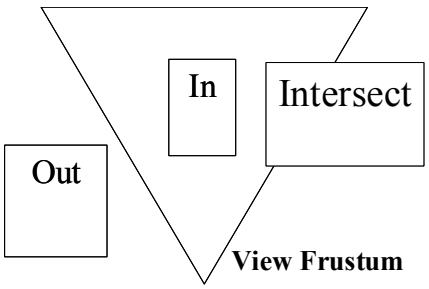
따라서 본 연구에서는 전체 지형에 LOD를 적용할 수 있는 그림 6과 같이 가장 작은 단위의 블록으로 나누어 텍스쳐 맵핑이 쉽도록 수정을 하였다. 이것은 특별히 추가적인 작업 없이 수행이 가능하며 쿼드트리의 루트 레벨에서부터 LOD의 레벨을 결정하기 위한 계산을 수행하는 대신 블록 단위의 서브 쿼드트리로부터 계산하여 결정할 수 있다.

블록의 크기는 쿼드트리르 구성하는 레벨이 n일 경우 $(2^n+1)\times(2^n+1)$가 된다. 불록의 크기
는 사용자가 정의 할 수 있지만 텍스쳐의 크기와 해상도 그리고 LOD로 얻을 수 있는 폴리곤 생성 이 점을 고려해서 선택해야 한다.

9x9크기의 블록을 사용하는 경우 가장 높은 LOD 레벨과 낮은 LOD 레벨과의 폴리곤 생성의 비율은 16:1이고 17×17 크기를 사용할 경우 64:3이다. 본 연구에서는 9X9 크기의 블록을 사용하였다.

4.3 가시 영역 추출

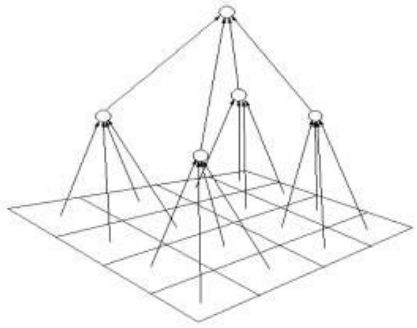
성능을 향상시키기 위해서 보이는 부분만을 추출하는 가시 영역추출(View Frustum Culling)은 Stefan[17]에서는 쿼드트리에 기반한 계층적인 방법을 사용하였다.



(그림 7) 가시영역 추출

가시 영역 추출이란 그림 7에서 보는 바와 같이 카메라의 뷰 볼륨안에 들어오는 물체만 렌더링 파이프라인에 집어넣기 위해 분류해 내는 것을 말한다. 카메라의 뷰 볼륨과 지형의 바운딩 볼륨과의교차 테스트를 하여 교차되거나 포함되는 경우만 렌더링 하도록 함으로써 렌더링 속도를 향상시키는 방법이다. 이 때 지형의 바운딩 볼륨은 AABB(Axis Aligned Bounding BOX)로 구성하였으며 뷰 볼륨과의 교차 테스트는 카메라 뷰 볼륨 상하 좌우의 평면 그리고 카메라와 멀리 떨어진 평면 이렇게 5개와 지형 바운딩 볼륨과의 교차 테스트를 하여 절정하였다[1].

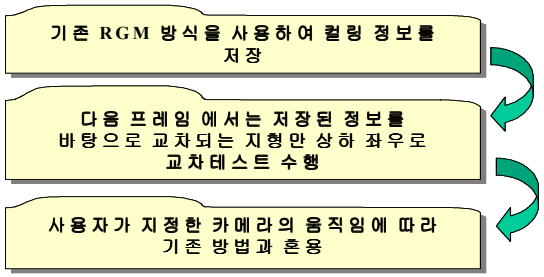
Stefan[17]의 쿼드트리를 사용한 방법은 그림 8에서 보는 바와 같이 가장 큰 블록 즉,



(그림 8) 쿼드트리 기반 컬링

상위 레벨에서부터 하위 레벨로 검사해 나가면서 가장 작은 블록인 폴리곤 단위까지 컬링하는 방법이다. 이 작업은 매 프레임을 그릴 때마다 반복적으로 수행해야 하는 작업이다. 이는 지형이 커질 수록 이러한 컬링 작업도 많아지기 때문에 지형 렌더링 시 프레임 저하의 가장 직접적인 원인이 될 수 있다. 따라서 이전 프레임에서 결정되었던 컬링정보를 사용한다면 효율을 높일 수 있다. 그러나 계층적으로 트리를 따라가면서 결정하는 쿼드트리를 사용하기 때문에 이전 정보를 활용할 마땅한 방법이 없으므로 추가적으로 2차원 배열 형태로 이전 정보를 저장하고 카메라의 움직임이 그렇게 크지 않을 경우 이 정보를 사용하여 가시영역 추출의 오버헤드를 줄이는 방법을 사용한다.

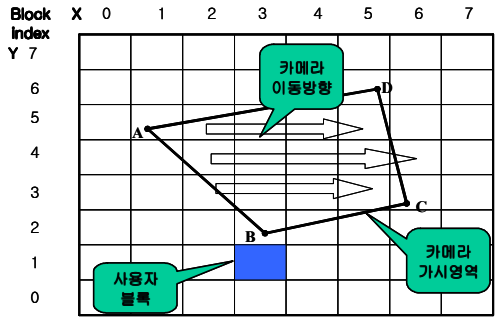
따라서 수정된 지형 렌더링 알고리즘은 그림 9와 같은 방법으로 구현될 것이며 구현과정은 다음과 같다.



(그림 9) 수정된 지형 렌더링 구현과정

처음 시작할 때에는 저장된 정보가 없기 때문에 쿼드트리를 사용하여 컬링 정보를 저장한다. 저장되는 정보는 사용자가 정의한 블록을 하나의 요소로 하고 이 블록은 2차원 XY축의 인덱스를 사용한다. 컬링 정보를 저장하는 방법은 그림 10에서 보는 바와 같이 2차원 배열 형태이지만 저장공간을 절약하기 위해 $(Y_{index}\ (X_{start}, X_{end}))$ 형태로 배열의 한 축에

대해 지형이 포함되는 시작점과 끝나는 점만을 저장한다. 그리고 다음 프레임에서는 쿼드트리를 사용하지 않고 여기에 저장되어 있던 정보를 바탕으로 교차되는 지형에 대해서 상하 좌우로 블록 영역을 확장 또는 축소하면서 교차 테스트를 수행한다.



(그림 10) 이전 프레임 컬링 정보를 사용하는 컬링

이 방법의 단점은 이전 정보가 없을 경우 사용할 수 없고 프레임간에 카메라의 움직임이 많을 경우 쿼드트리 방법보다 효율이 떨어진다는 것이다. 따라서 카메라의 움직임이 적을 경우에만 사용하고 카메라가 다른 곳으로 이동하거나 너무 많이 움직였을 경우에는 쿼드트리 방법을 혼용하였다. 이때 카메라가 어느 정도 움직여야 하는지에 대한 기준을 정해야 하는데 이것은 실험을 통해 결정했다.

다음 장에서는 카메라와 XYZ축 이동과 XYZ축 회전에 대해 어느 정도의 이동 및 회전을 허용할 것인지에 대한 실험을 하고 그 실험 결과를 바탕으로 쿼드트리만 사용한 경우와 쿼드트리와 이전 컬링 정보를 혼합환 방법에 대한 성능 테스트를 했다.

5. 실험결과 및 분석

5.1 카메라 이동 및 회전시 한계치 측정

현재 프레임에서 다음 프레임을 렌더링 할 때 카메라가 이동하는 경우 이전 프레임의 정보를 사용하기 위한 한계치를 결정해야 한다. 이것은 Height Field 지형 데이터의 샘플링 포인트를 어느 정도 까지 이동하는 것을 허용할 지를 결정하는 것이다. 실험에서는 샘플링 포인트를 기준으로 하지 않고 사용자가 정의한 블록을 기준으로 실험을 했으며 블록의 크기는 9X9를 사용하였다. 비교는 쿼드트리만 사용한 경우와 쿼드트리로 생성한 컬링 정보를 저장하고 일정 간격으로 이동이나 회전을 하여 컬링에서 수행하는 교차 테스트 횟수로 비교하였다. 비교 결과 한계치는 쿼드트리만 사용한 경우에 비해 교차테스트 횟수가 비슷해지는 수치로 결정하였다. 이동의 경우 카메라의 far 한계치를 달리 하여 실험하였으며 결과는 표 1과 같다.

<표 1> 이동 및 회전에 대한 한계치

(9X9블록, 카메라의 FOV 60° 기준)

	이동	회전
X축	2.5 block	30°
Y축	2.75block	27°
Z축	1.5block	거의일정

5.2 렌더링 실험결과

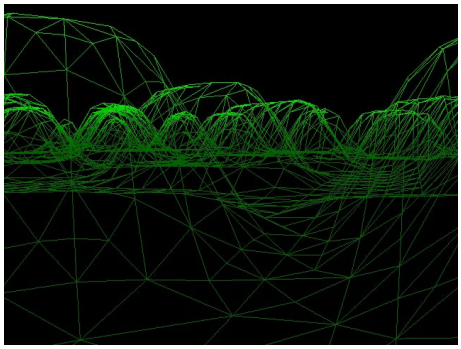
이 시험은 쿼드트리만 사용한 경우와 혼용한 경우에 대해 실제로 렌더링을 수행하여 평균 교차 테스트 횟수 및 초당 프레임 수를 비교하였다. 실험 시스템은 인텔 팬티엄 1.6Mhz, RAM 512M, 가속기 Riva GeForce4 mx에서 OpenGL을 사용하였으며 실험에 사용한 Height Field 데이터의 크기는512X512, 샘플링 포인트간 스케일 50, 카메라의 far 한계 7000, 카메라의 FOV는 60°이다. 카메라 이동에 대한 한계치는 XYZ 모두 1.5블록을 사용하였고 회전에 대한 한계치는 20°이다.

실험 방법은 매초간 지형 위를 카메라가 움직이면서 임의의 축으로 회전하는데 이동 속도나 회전 속도는 가변적이다. 그리고 카메라는 움직이는 동안 3번의 이동을 통해 반드시 쿼드트리를 사용하도록 하였다. 표2는 실험 결과이다.

<표 2> 지형 렌더링 실험결과

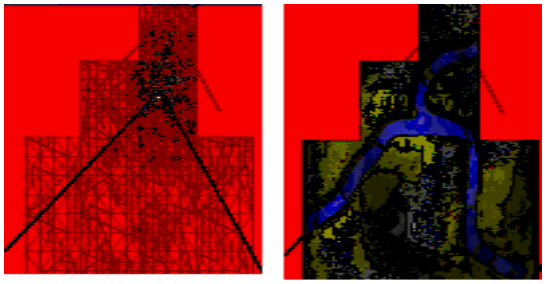
(9X9블록, 카메라의 FOV 60° 기준)

	기존방법		혼용	
	FPS	Intersection	FPS	Intersection
CLOD off	54.247	372.7	68.92	218.1
CLOD on	44.82	372.7	59.17	218.1



(그림 11) 카메라 시점 렌더링 화면

실험 결과 초당 프레임 은 약 25% 정도 향상되었



(그림 12) 위에서 바라본 컬링 지형

고 교차 테스트 횟수는 약 60% 감소하였다. 그림10은 실험을 실시한 렌더링 화면이고 그림 11은 또 다른 카메라로 지형 위에서 바라본 컬링 결과를 보여주는 화면이다.

5.3 실험결과 분석

이전 컬링 정보를 사용하여 블록의 영역 확장을 통한 교차테스트의 수행은 카메라의 이동이 심하면 심할수록 성능이 현저히 떨어진다. 카메라의 이동에 상관없이 일정한 쿼드트리의 방식보다는 안 좋은 방법이다. 그러나 매 프레임마다 카메라의 이동은 그렇게 크지 않은 경우가 대부분이기 때문에 실험결과에서 볼 수 있듯이 렌더링 성능을 향상시킬 수 있었다.

5.1 절에서 설정한 한계치는 실제로 화면상에서 얼마만큼의 이동인지 잘 알 수가 없다. 실험한 방법에서 움직임의 기준이 되는 블록은 9×9 크기를 사용했을 경우 지형 데이터의 샘플링 포인트 9칸에 해당한다. 매 프레임 마다 카메라가 한 블록을 이동한다면 평균적인 초당 프레임 수가 30인 경우 1초에 30 블록 즉, 지형 샘플링 포인트의 270칸을 이동한 것이 된다. 실제로 이러한 수치는 가상공간상에서 순간 이동에 해당하는 결과를 보여준다. 따라서 샘플링 포인트 간격을 50으로 하여 샘플링 포인트간 거리를 50m라고 보았을 때 270칸은 13,500m에 해당하며 초당 13.5km를 이동한 것이다. 이것은 음속의 약 40배에 달하는 속도이다. 이 결과로 볼 때 실제 가상현실이나 게임에서의 카메라의 이동은 현실세계의 객체들의 이동과 그리 다르지 않다. 예를 들어 비행시뮬레이션의 경우에도 음속의 40배 이상의 움직임은 존재하지 않는다. 따라서 실험에서의 카메라 이동의 한계치의 범위가 충분히 효과적이라고 말할 수 있다.

6. 결론

본 연구에서는 PC 기반 가상가시화에서 지형을 좀더 현실적으로 보여주기 위해 기존에 연구되어 있는 Height Field 기반 지형 렌더링 시스템을 비교·분석하였고, PC 기반 가상가시화의 특성을 고려하여 텍스쳐 맵핑이 용이하도록 블록 단위로 렌더링 할 수 있도록 수정하였다. 그리고 가시영역 추출 시 이전 프레임의 컬링 정보를 재사용 하도록 하여 기존 방법에 비해 속도를 향상시킬 수 있었다. 이전 컬링 정보를 사용할 경우 카메라의 이동 속도나 회전 속도에 크게 영향을 받게 되는데 이에 대한 실험을 하여 적정한 한계치를 찾고 이를 용용하여 렌더링 속도를 향상시켰다.

이 한계치를 실제 지형가시화에 적용할 경우 지형 샘플링 포인트의 스케일 비율과 전체 지형의 크기 그리고 카메라가 주로 움직이는 고도에 따라 영향을 받게 된다. 비행 시뮬레이션 같은 고도가 높은 곳에서 카메라 워킹을 하는 게임이라면 샘플링 포인트간 스케일을 더 크게 잡을 것이다. 반대로 지형 바로 위를 움직이는 자동차나 사람과 같이 속도가 비행기에 비해 훨씬 느리게 카메라 워킹이 이루어진다면 그만큼 샘플링 포인트간 간격을 줄여서 지형을 섬세하게 표현할 수 있다. 샘플링 포인트 간격을 줄이는 것은 전체 지형의 크기를 줄어들게 하기 때문에 적절하게 스케일을 해 주어야 한다.

References

[1] 김용준, *3D 게임프로그래밍*, 한빛미디어, pp. 348-472, 2003.
[2] 류광, *Game Programming Gems 1*, 정보문화사, 2001.
[3] 류광, *Game Programming Gems 2*, 정보문화사, 2002.
[4] 류광, *OpenGL Game Programming*, 정보문화사, 2001.
[5] *마이크로 소프트웨어 2002년 3월호*, pp. 372-381.
[6] *마이크로 소프트웨어 2002년 4월호*, pp. 395-404.
[7] David C. Taylor and William A. *Barrett. An algorithm for continuous resolution polygonalizations of a discrete surface,* In Proceedings of Graphics Interface '94, pp. 33-42, 1994.
[8] Gross M.H., Gatti R. and Staadt O. *Fast multiresolution surface meshing*, In Nielson G. and Silver D. , editors, Pro-ceedings Visualization’95, pp. 135-142. IEEE Computer Society Press, 1995.
[9] Hugues H. , *Efficient Implementation of Progressive Meshes*, Computers&Graphics, 1998.
[10] Hugues H. , *View-dependent refinement of progressive meshes,* In Computer Graphics(Proceedings of Siggraph’97), pp.189-198, 1997.
[11] Luebke D. , *Level of Detail for 3D Graphics*, Morgan Kaufmann, pp. 185-228, 2002.
[12] Leclerc, Y. G. and Lau, S. Q. TerraVision:*A Terrain Visualization System,* Technical Report Technical Report 540. SRI International. Menlo Park, CA. April 1994.
[13] Lindstrom P. and Koller D. and Ribarsky W. and Hodges L. F. , and Faust, N. and Turner, G. A. *Real-Time, Continuous Level of Detail Rendering of Height Fields*, Proceedings of ACMSIGGRAPH 96, pp. 109-11

8, August 1996.
[14] Lindstrom P. , *Visualization of Large Terrains M ade Easy*, IEEE Visualization, 2001.
[15] Mortensen J. , *Real-time rendering of height field s using LOD and occlusion culling*, Computer Science Department, University College London, London, 2000.
[16] *OpenGL Architecture Review Board,* OpenGL Ref erence Manual. Addison Wesley, 1992.
[17] R¨ottger S. , Heidrich W. , Slusallek P. , Seidel H.P. , *Real-Time Generation of Continuous Levels of Detail for Height Fields*, In Rroc.6th International Conf erence in Central Europe on Computer Graphics and V isualization '98, l998, pp. 3l5-322.
[18] Suter M. and Niesch D. , *Automated generation o f visual simulation databases using remote sensing and GIS*, In Nielson G. and Silver D., editors, Proceedings Visualization'95, pp. 135-142. IEEE Computer Society Press, 1995.
[19] Watt A. and Policarpo F. , *Real-time Rendering a nd Software Technology*, Addison-Wesley, 2002.