Korean Journal of Remote Sensing, Vol.20, No.6, 2004, pp.409~418

메모리 효율 향상을 위한 고정격자기반 실시간 지형 LOD 알고리즘에 관한 연구

황보택근* · 양영규* · 문민수**
경원대학교 소프트웨어대학*, (주)케이연구소**

A Study on Regular Grid Based Real-Time Terrain LOD Algorithm for Enhancing Memory Efficiency

Taeg-keun Whangbo*, Young-Kyu Yang*, and Min-soo Moon**
College of Software, Kyungwon University*, k Research Inc**

Abstract : LOD is a widely used technique in 3D game and animation to represent large 3D data sets smoothly in real-time. Most LOD algorithms use a binary tree to keep the ancestor information. A new algorithm proposed in this paper, however, do not keep the ancestor information, thus use the less memory space and rather increase the rendering performance. To verify the efficiency of the proposed algorithm, performance comparison with ROAM is conducted in real-time 3D terrain navigation. Result shows that the proposed algorithm uses about 1/4 of the memory space of ROAM and about 4 times faster than ROAM.

Key Words : Level-of-detail, 3D Terrain, Memory Space, Rendering Speed.

요약 : LOD(Level of Detail)는 대용량의 데이터를 갖는 게임이나 애니메이션에서 실시간에 자연스러운 처리를 위해서는 사용되는 기법으로, 3차원 게임이나 애니메이션에 자주 사용되고 있으며 3차원을 표현하기 위한 중요한 기술로 부각되고 있다. 본 논문에서는 기존의 2진 트리 방식에서 부모 노드의 정보를 배제하여, 널리 사용되고 있는 2진 트리 방식인 ROAM(Realtime Optimally Adapting Meshes) 알고리즘 보다 메모리의 양을 적게 사용하며 성능은 오히려 향상된 새로운 LOD 알고리즘을 제시하고자 한다. 본 알고리즘의 성능을 검증하기 위하여 3차원 지형에서 실시간 네비게이션 시 기존의 ROAM알고리즘과 제안된 알고리즘을 비교하였으며 그 결과 메모리는 제안 알고리즘이 기존의 알고리즘에 약 1/4정도를 사용하고 수행 시간은 약 4배가량 빠른 것으로 조사되었다.

2004년 10월 8일 접수 : 2004년 11월 12일 수리.

1. 서 론

넓은 영역을 포함하는 3차원 지형을 컴퓨터에서 표현하는 것은 컴퓨터 그래픽 기술의 발달에도 불구하고 방대한 데이터의 양으로 인하여 실시간에 자연스러운 처리가 어렵다. 그래서 지금까지 게임이나 애니메이션 등에서 지형을 표현하는 방법은 2차원의 영상을 주로 사용하였으나 사실감 등이 떨어지므로 3차원 지형 표현에 대한 요구가 높아지고 있다.

그동안 3차원의 지형을 효율적으로 표현하기 위한 연구가 많이 진행되었고 그 중에서 인간의 시각적인 특성을 이용한 LOD (Level Of Detail)방법이 널리 사용되고 있다. LOD의 기본 개념은 3차원 모델의 모양을 최대한 유지하면서, 멀리 있거나 단순한 물체를 기하학, 위상 정보를 이용하여 데이터의 양을 줄이고 가까워지면 다시 데이터의 양을 늘려서 자세하게 표현 방법이다.

본 논문에서는 3차원의 지형을 표현할 때 적은 메모리를 사용하여 보다 빠르게 화면에 출력해 주는 새로운 LOD알고리즘을 제시한다.

LOD 방법은 지형의 표현 방법에 따라 다른 방법이 사용되는데 일반적으로 지형을 표현하는 방법에는 불규칙 삼각망(Triangulated Irregular Networks)을 이용하는 방법과 고정 격자(Regular Grid)를 이용하는 방법이 있다.

불규칙하게 분포된 위치에서 표고를 추출하여 이들 위치를 삼각형의 형태로 연결하여 전체 지형을 표현하는 방식이 불규칙 삼각망인데, 이러한 방식은 세 개의 위치를 가지고 하나의 삼각형을 이루며 각각의 삼각형 내에서 경사의 크기 및 방향이 결정된다[Garland et al, 1995; 류승택 외, 2000; 김양수, 2000). 고정 격자 방식에서 고저차이에 의한 방법은 지형 데이터를 효율적으로 저장하고 거리에 따라 적절하게 폴리곤의 수를 조절할 수 있는 데이터 저장 방법으로 지형데이터를 표현할 때 지형의 높이 정보만을 저장하고 폴리곤은 내부적인 알고리즘으로 생성해 내는 방법이다.

본 논문의 구성은 2장에서 대표적인 LOD 알고리즘에 대하여 살펴보고 3장에서 고정 격자 방식에서 메모리의 효율성과 실행 속도를 향상시킨 새로운 LOD 알고리즘에 대하여 설명한다. 4장에서는 제안한 알고리즘의 성능을 평가하기 위하여 대표적인 ROAM방식과 제안한 방식을 실험을 통하여 비교분석하였으며 5장에서 결론은 맺는다.

2. 관련 연구

Lindstrom *et al.* (1996)은 쿼드트리 구조에서 bottom-up 방식으로 점에 기반한 간략화와 블록에 기반한 간략화 방법을 사용하여 높은 렌더링 속도를 보장하는 방법을 제시하였다. 이 방식은 실시간 LOD의 초기 연구로서 간략화 시 모든 점에서 높이차를 계산하므로 시간이 많이 소요되며 이로 인해 프레임 전환 시 지형이 튀는(poping)현상이 발생하는 문제점이 있다.

Rottger *et al.* (1997)는 Lindstrom의 문제점을 해결하기 위하여 쿼드트리 구조의 top-down 방식을 사용하여 렌더링 되는 각 프레임에서 전체 지형 중 일부분만을 다루게 하여 많은 양의 지형에서도 높은 프레임 속도를 가능하게 하는 방식을 제안하였다. 이 방식은 쿼드트리 행렬에 의해 구성된 삼각형이 지형에 대응되는 행렬의 요소가 1로 되어 있는 부분만을 재귀적으로 쿼드트리를 순회하면서 트라이앵글 팬(triangle fan)을 사용하여 랜더링하였다.

Duchaineau *et al.* (1997)는 2진 트리 방식을 이용하여 만들어진 트리구조에서 얻어지는 삼각형들을 bottom-up과 top-down방식으로 세분화 하고 단순화하여 지형을 형성하는 ROAM알고리즘을 발표하였다. ROAM방식은 Fig. 1에서와 같이 직각삼각형의 빗변을 공유하는 두 삼각형을 세분화하고 단순화하여 LOD를 수행하는 방식으로, 상당히 우수한 알고리즘으로 평가되고 있으며 게임 등에서 널리 사용되고 있다. Blow(2000)는 초기의 ROAM 방식이 조밀하게 형성된 지형데이터에서는 효과적이지 못한 것을 발견하고 등가곡면(isosurface)를 사용하여 ROAM의 속도를 개선하였다.

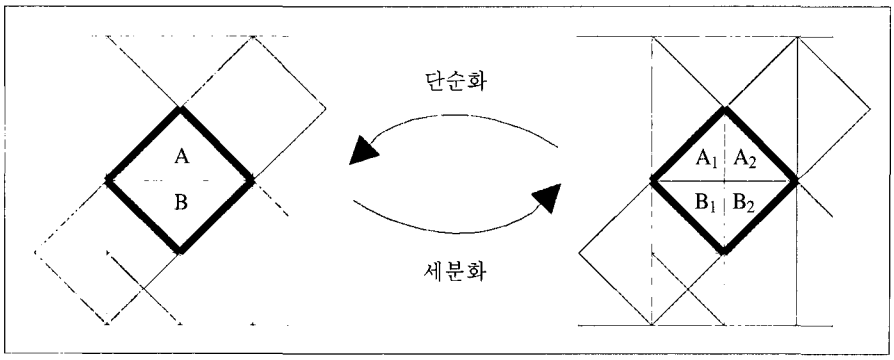

Fig. 1. Split and merge operation on a binary triangle tree used by the ROAM algorithm.

Levenberg (2002)는 CABTT (Cached Aggregated Binary Triangle Trees)라는 기존의 2진트리 방식을 확장한 저장된 집합 2진 삼각형을 사용하여 bottom-up과 top-down방식으로 세분화 하거나 단순화 하여 LOD를 수행하였다. CABTT는 세분화 될 삼각형을 미리 가지고 있는 방식으로 메모리가 많이 소요될 뿐 아니라 프레임 변환 시 지형이 튀는(poping)현상이 발생할 수도 있다.

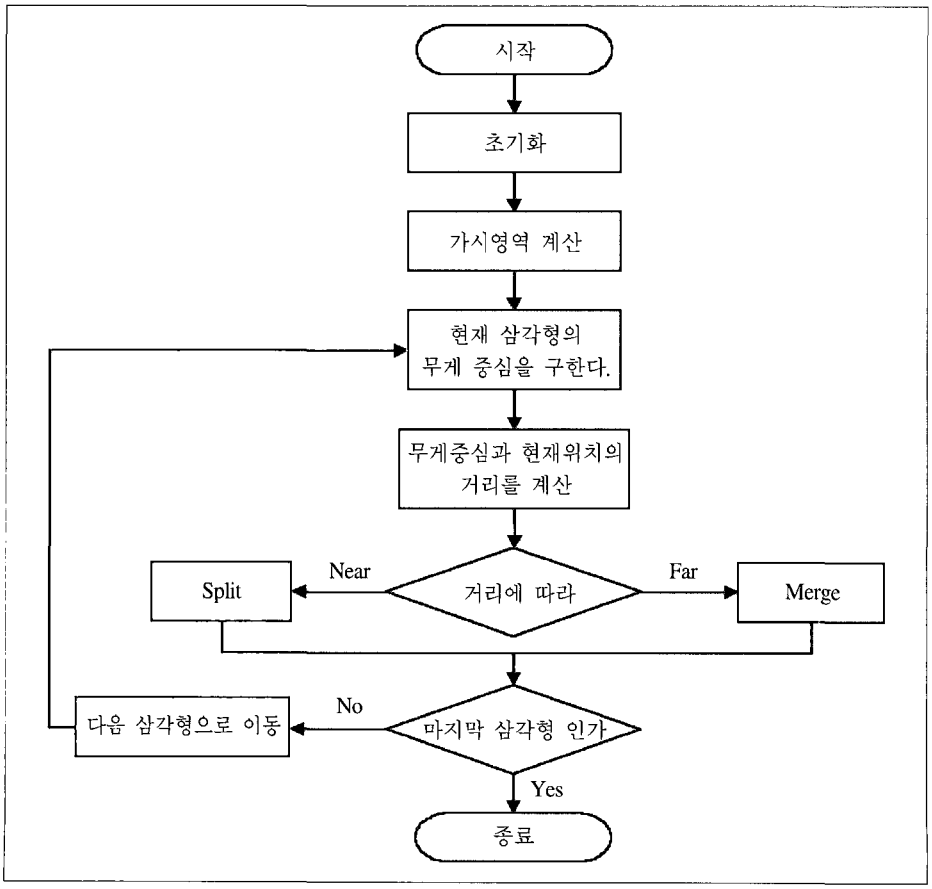

Fig. 2. Flowchart of LOD algorithm.

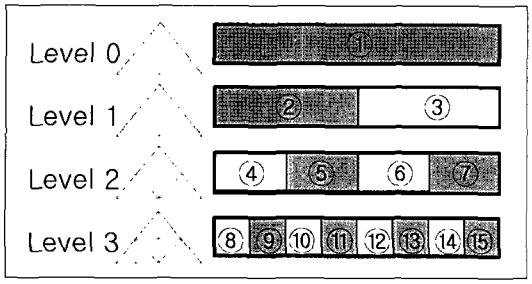

Fig. 3. Data Structure for binary tree.

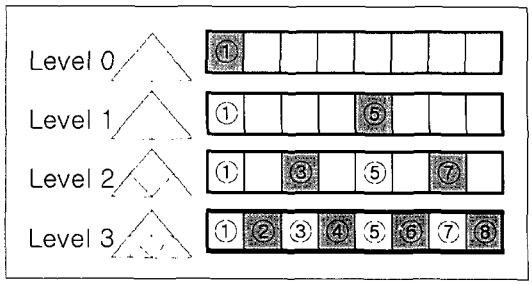

Fig. 4. Proposed data structure.

일반적으로 불규칙 삼각망에서의 LOD는 실시간 처리가 어려운 것으로 알려져 있으며, 불규칙 삼각망을 사용한 알고리즘 중 수행속도 개선을 위해 전체 지형을 구역화 하고 각 구역별로 세밀화 갈망삽입알고리즘을 적용하여 단순화를 수행하는 알고리즘이 제안되었다(강윤식 외, 2000).

지금까지 제안된 대부분의 고정격자 LOD 알고리즘은 각 단계별로 트리구조를 사용하여 삼각형을 관리하므로 가장 정밀한 단계의 삼각형의 수가 2^n개라고 하면 노드의 개수는 2^{n+1}-1가 되고, 또한 각 노드들은 연결리스트로 연결되어 몇 단계 위의 자식이나 조상을 찾는데 시간이 많이 필요하게 된다.

본 논문에서는 삼각형의 세분화와 단순화 알고리즘보다는 새로운 자료구조를 제안하여 기존의 트리구조 방식의 메모리의 비효율성과 탐색에 소요되는 시간을 개선하여 보다 빠른 속도로 LOD를 수행할 수 있는 새로운 방법을 제안하고 자 한다.

3. 제안 알고리즘

제안하는 LOD 알고리즘의 전체 구성은 크게 3단계로 나눌 수 있는데, 초기화 단계는 지형 데이터로부터 높이 값을 읽어온다. 두 번째 단계는 가시영역 계산을 위하여 현재위치에서 지형의 가시영역을 설정 하는 단계이다. 마지막단계는 세분화와 단순화 단계로서 가시영역 계산이후 부분은 모두 마지막 단계에 포함된다. 세분화와 단순화 단계에서는 현재 시점의 위치와 방향, 삼각형 무게중심과 실제높이의 차이값 및 거리 등에 따라서 세분화와 단순화를 수행한다. Fig. 2는 본 논문에서 제안하는 알고리즘의 전체적인 구성도이다.

1) 제안한 자료구조

일반적인 지형세분화에서는 2진 트리, 쿼드 트리 등을 사용하는데 제안하는 LOD의 자료구조는 메모리의 효율을 높이기 위하여 부모의 정보를 가지는 트리구조를 사용하지 않고 배열을 사용하여 트리구조의 부모 정보를 생략한 삼각형의 정보를 가지도록 한다.

본 연구에서 사용한 지형은 전체 지형을 16×16개의 패치로 나누고 각 패치는 8단계로 세분화되는 것으로 하여 각 패치에서 가장 세밀한 단계에선 256개의 삼각형으로 세분화 된다. 지형 전체는 총 216개의 삼각형을 가지게 되며 2진 트리를 사용하여 구성하면 $2^0+2^1+2^2+\cdots+2^{15}(=2^{16}-1)$ 개의 부모의 정보를 가지게 된다. 반면 제안된 자료구조에서는 부모의 정보를 생략하므로 부모 정보만큼의 메모리의 효율을 극대화하게 된다.

부모의 정보를 생략하는 대신 현재 가지고 있는 지형의 정보를 이용하여 세분화와 단순화를 병행함으로써 LOD를 수행하도록 한다.

Fig. 3는 하나의 패치가 최대 8개의 삼각형으로 나뉜다고 가정했을 때의 2진 트리 방식의 자료구조이고, Fig. 4는 제안한 자료구조이다. Fig. 3의 2진 트리구조의 경우에는 Triangle ①부터 Triangle ⑮까지 24-1 개의 삼각형 정보를 저장해야 한다. Fig. 4의 제안한 자료구조는 부모의 정보를 저장하지 않고 최하위 노드의 개수인 23개의 삼각형 정보만을

저장한다. 제안한 방법의 경우 단계 0에서 단계 1로 삼각형이 세분화 되면 Triangle ①은 단계 1의 삼각형으로 갱신되고, Triangle ⑤를 추가하게 된다. 마찬가지로 단계 1에서 단계 2로 삼각형이 세분화 될 때에도 Triangle ①, ⑤는 단계 2의 새로운 삼각형으로 갱신되고, Triangle ③, ⑦을 추가한다.

본 논문에서는 지형을 16×16의 패치를 나누고 각 패치는 최대 256개의 삼각형으로 분할하여 256×16×16(=65536)만큼의 배열을 설정하여 LOD 알고리즘을 수행하도록 하였다.

2) 세분화(Split)

본 논문에서는 최초 지형을 16×16의 패치로 나누어 지형의 시점에서 가까운 부분부터 지형을 세분화하여 LOD를 수행한다.

(1) 세분화 알고리즘

중점의 높이와 실제 높이의 차이 값이 임계 값보다 크면 세분화 알고리즘을 수행한다. 이때 세분화에 따라 인접삼각형과 균열(crack)일 발생할 수 있는데 이때는 인접삼각형도 함께 강제세분화(forced split)를 시행하여야한다(Lindstrom, 1996). 전체 세분화 알고리즘은 다음의 6단계를 따른다.

Tbn : Triangle BaseNeighbor
hypotenuse : (PATCH_SIZE / (maxLevel+1))

step 1. Split 하면 최대 단계 보다 커지는가?
1.1 yes 이면 step 6
step 2. 삼각형 빗변의 길이가 최소인가?
2.1 yes 이면 step 6
2.2 no 이면 삼각형 빗변의 중점의 높이 계산
step 3 중점의 높이와 실제 높이가 차이가 있는가?
3.1 No 이면 step 6
3.2 yes 이면 현재 삼각형의 Tbn을 구한다.
step 4. Tbn의 단계가 현재 삼각형보다 작은가?
4.1 yes 이면 Tbn의 단계가 현재 단계보다 클때 까지 Tbn을 Split한다.
step5. 현재삼각형을 분할한다.
step6. 종료

3)단순화(Merge)

세분화 알고리즘으로 분할된 지형정보를 다시 상위의 삼각형으로 합쳐주는 알고리즘이 단순화 알고리즘이다. 단순화는 시점으로부터 먼 거리의 지형에 대해 삼각형을 합쳐주어서 LOD를 수행한다.

(1) 단순화 알고리즘

단순화하려는 삼각형의 거리가 임계 값보다 크게 되면 단순화 알고리즘을 수행한다. 단순화의 경우도 세분화와 마찬가지로 균열이 발생할 경우 강제 단순화가 필요하며 전체 알고리즘은 다음의 8단계를 따른다.

Tcur : Triangle Current (현재 삼각형)
Tbn : Triangle BaseNeighbor
BPn : Merge가 되었을 때 기준이 되는 삼각형
NPn : Merge가 되었을 때 정보를 상실하는 삼각형

step 1. 현재 단계가 0-maxLevel 사이인가?
1.1 no 이면 step 8.
step 2. 현재 삼각형이 Bpn 인가?
2.1 yes 이면 step 3
2.2 no 이면 step 4.
step 3
3.1 Npn을 구한다.
3.2 Bpn의 단계보다 Npn의 단계가 더 큰가?
3.2.1 no 이면 step 5
3.2.2 yes 이면 Npn을 Merge한다.
step 4
4.1 Bpn을 구한다.
4.2 Npn의 단계보다 Bpn의 단계가 더큰가?
4.2.1 no 이면 step 5
4.2.2 yes 이면 Bpn을 Merge한다.
step 5 Tbn을 구한다.
step 6 Tbn을 Merge한다.
step 7 현재 삼각형을 Merge 한다
step 8 종료

4) BaseNeighbor 탐색

BaseNeighbor탐색은 Fig. 5에서 삼각형 D를 세분화할 경우 D만을 세분하면 d에서 균열이 발생할 우려가 생기므로 빗변을 공유하고 있는 A를 찾아 함께 세분하게 되는데, A와 D는 단계가 다르므로 A는 2번 세분화를 시행하여야 한다. 우선 A를 이등분하고 다시 D와 인접한 삼각형을 세분한다. 이때 A를 이등분하면서 B와의 균열 발생가능성이 생기며 A와 B는 단계가 다르므로 B는 다시 2번 세분화를 실시하게 된다. 이와 같은 단계로 세분하면 삼각형 D의 세분화의 최종 결과는 Fig. 5의 오

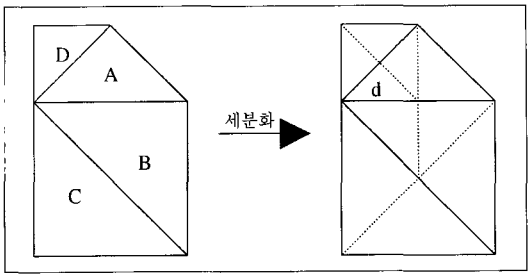

Fig. 5. Forced split of triangle D.

른쪽 그림과 같은 형태로 된다. 이와 같이 하나의 삼각형을 세분할 경우 세분화로 인하여 빗변을 공유하는 삼각형에서 균열이 발생할 가능성이 있으면 지속적으로 세분화하는 과정을 강제세분화(forced split)라고 한다. 단순화의 경우도 세분화의 역으로 강제단순화를 수행하게 된다. 이러한 과정에서 빗변을 공유한 인접한 삼각형 즉 BaseNeighbor를 탐색하는 과정이 필요하고, 이과정이 LOD에서 가장 시간이 많이 소요되는 요소이다.

Fig. 5의 세분화 경우 트리구조를 사용하였다면 D의 BaseNeighbor를 찾는 과정이 필요하고 A를 세분화하였을 경우 다시 A의 자식노드로의 연결이 필요하며 A의 BaseNeighbor를 탐색하는 과정 등이 필요하다. 이 경우 연결리스트를 사용하므로 BaseNeighbor 탐색에 많은 시간이 소요되며, 본 연구에서는 이러한 BaseNeighbor탐색을 일정한 규칙에 의해 쉽게 탐색할 수 있도록 함으로서 전체 LOD에 소요되는 시간의 단축이 가능하도록 하였다.

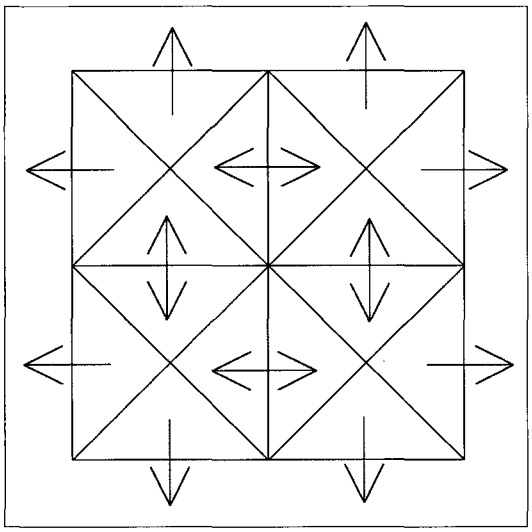

Fig. 6. BaseNeighbor in Level 3.

BaseNeighbor탐색은 현재 삼각형이 속한 단계에 따라 탐색하는 방법이 달라지는데 짝수 단계인 경우는 삼각형이 속한 패치 안에서 BaseNeighbor를 탐색하고 현재 삼각형이 홀수 단계인 경우에는 같은 패치에 있을 수도 있고 인접한 패치에서 BaseNeighbor를 찾아야 하는 경우가 발생한다. Fig. 6에 단계 3에서 삼각형들의 BaseNeighbor를 표현하고 있는데 패치의 경계부분에 있는 삼각형들은 인접한 패치에서 BaseNeighbor를 찾아야 하고 내부에 있는 삼각형들은 같은 패치안에서 BaseNeighbor를 찾을 수 있다.

(1) 같은 패치 내에서의 탐색

같은 패치 내에서 BaseNeighbor를 탐색하는 방법은 현재 삼각형의 번호에 $2^{maxlevel}$ 또는 $2^{maxlevel-currentlevel}$을 더하거나 빼서 Base-Neighbor를 찾는다. $2^{maxlevel}$는 삼각형이 그림 Fig. 7에서와 같이 패치의 대각선(AB를 잇는 대각선)을 빗변으로 사용하는 삼각형일 경우에 해당되고 $2^{maxlevel-currentlevel}$는 그 외의 삼각형에 사용된다. 패치의 대각선을 빗변으로 사용하는 삼각형은 짝수 단계에서 발생하고 이러한 삼각형은 일정한 패턴으로 생기므로 쉽게 판단할 수 있다. 홀수 단계의 경우 같은 패치에서 BaseNeighbor을 찾을 경우 $2^{maxlevel-currentlevel}$을

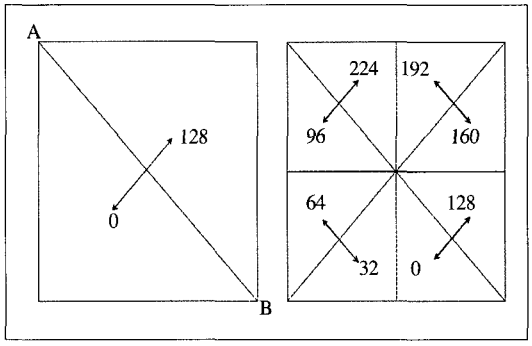

Fig. 7. BaseNeighbor search in the same patch.

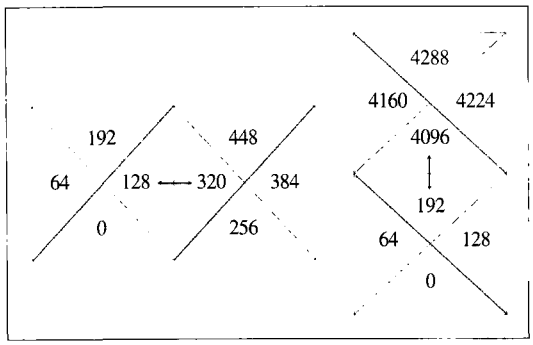

Fig. 8. BaseNeighbor search in the adjacent patch.

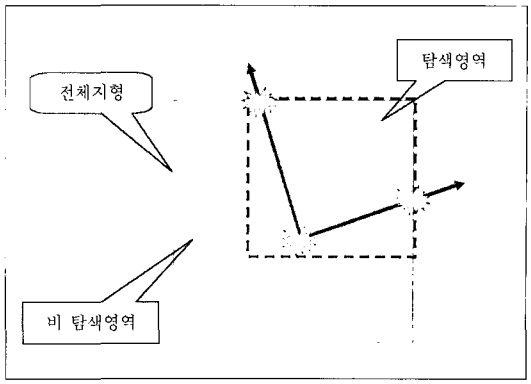

Fig. 9. Visible Space and non-visible space.

사용하면 된다.. Fig. 7는 짝수 단계의 경우 같은 패치 내에서 BaseNeighbor를 찾는 것을 보여준다.

(2) 인접 패치에서 탐색

현재 삼각형이 홀수 단계일 경우 패치의 경계선을 빗변으로 사용하고 있는 삼각형인 경우 BaseNeighbor는 인접한 패치에서 찾아야 하는데 q3 또는 np 값을 현재 삼각형의 번호에 더하거나 빼서 찾는다. 여기서 q3는 Fig. 8과 같이 좌우에 인접한 패치에서 BaseNeighbor를 찾을 경우에 사용되며 하나의 패치에서 가질 수 있는 최대 삼각형의 개수의 3/4로서, 단계를 8단계로 하였을 경우에는 최대 삼각형의 개수가 256이므로 이 값의 3/4는 192가 된다. np는 아래위로 인접한 패치에서 BaseNeighbor을 찾을 경우에 사용되며, 가로 패치 수에 따라 값이 결정된다. 본 논문에서와 같이 패치의 수가 16×16이며 단계를 8단계로 하면 np=(16×256)-q3=16×256-192=3094가 된다. 패치의 경계선을 빗변으로 사용하는 삼각형은 홀수 단계에서 발생하고 일정한 패턴으로 발생하므로 쉽게 판단할 수 있다. Fig. 8은 인접한 패치에서 BaseNeighbor를 찾아주는 것을 보여주고 있다.

(3) 예외 처리

예외 처리는 단계 3 이후부터, 즉 단계 4부터 예외 처리가 필요한 삼각형이 생겨난다. 예외 처리는 (1)과 (2)절에서 설명한 방법으로 BaseNeighbor을 찾을 수 없는 삼각형으로서 이러한 삼각형의 위치는 단계가 증가하여도 동일한 위치에서 예외가 발생한다. 단계 4이상의 각 패치에서 삼각형이 생성될 때 삼각형 4, 11, 16, 19, 20, 27, 28, 35, 36, 43, 44, 47, 52, 59의 배수에서 예외가 발생하며, 이러한 삼각형의 BaseNeighbor는 순서대로 삼각형 4의 BaseNeighbor는 삼각형 11, 삼각형 16의 BaseNeighbor는 삼각형 19의 순으로 BaseNeighbor가 결정된다.

5) 가시성 판단(Visibility Test)

실시간으로 지형을 렌더링하는데 중요한 측면의 하나가 view영역의 설정이다. 카메라가 특정위치, 특정 방향에 배치되어 있을 때, 그 카메라로부터 보일 수 있는 부분을 효율적으로 판단할 수 있어야 한다. 지형의 어떤 부분이 보이는지 보이지 않는지를 빠르고 정확하게 판단하여 보이지 않는 부분은 무시하므로 보다 빠르게 지형을 표현할 수 있다.

본 논문에서 지형의 가시영역 설정은 탐색 영역의 설정단계, 가시영역 설정단계의 2단계로 나누어서 지형의 가시영역을 판단하였다.

(1) 탐색영역의 설정

Fig. 9과 같이 전체 지형을 탐색 영역과 비 탐색 영역으로 나누어서 비 탐색 영역에 대해서 계산 량을 줄여서 시야영역을 확보하는 방법으로 가시성을 판단하도록 한다.

탐색 영역 설정을 위해 교차점 탐색은 직선의 방정식을 이용하여 Fig. 10에서와 같이 지형의 외곽선과 시야각이 교차하는 점을 찾아낸다. 직선의

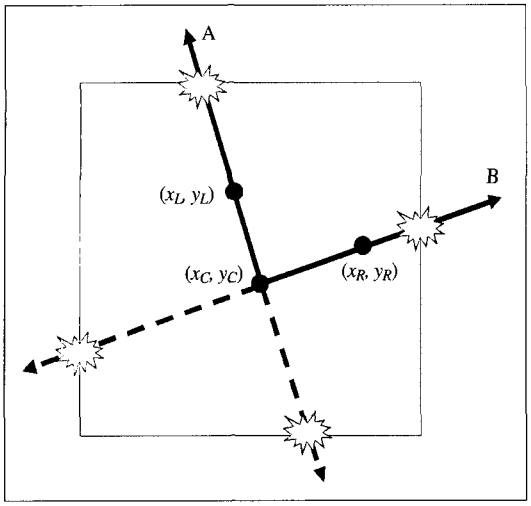

Fig. 10. Intersection points in visible space.

경우 두개의 점으로 직선의 방정식을 구할 수 있는데, 직선 A와 직선 B의 경우 두 직선의 교점 (x_C, y_C)과 각 직선 위의 한 개의 점(x_L, y_L),(x_R, y_R)을 가지고 직선의 방정식을 구한 후 전체 지형의 외곽선과의 교차점을 찾아줄 수 있다.

앞에서 찾은 지형의 외각선과 교차하는 2 점과 시점을 이용하여 최소 x, y좌표와 최대 x, y 좌표를 구하여 탐색영역을 만든다.

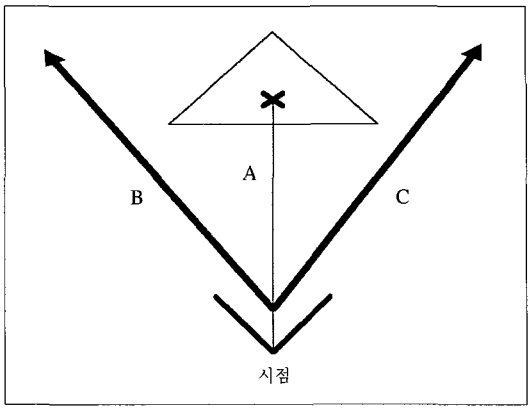

Fig. 11. Visible space set-up.

(2) 가시영역 설정

가시영역 설정은 위에서 결정된 탐색 영역 안의 삼각형들에 대해서 삼각형의 무게중심을 연결하는 A벡터와 시야를 결정하는 B벡터, C벡터의 외적을 구함으로 구해진다. Fig. 11의 A벡터와 B벡터의 외적과 C벡터와 A벡터의 외적을 구하여 두 외적의 부호가 같다면 삼각형은 가시영역 안에 존재하고 두 외적의 부호가 같지 않다면 그 삼각형은 가시영역에 존재하지 않는다.

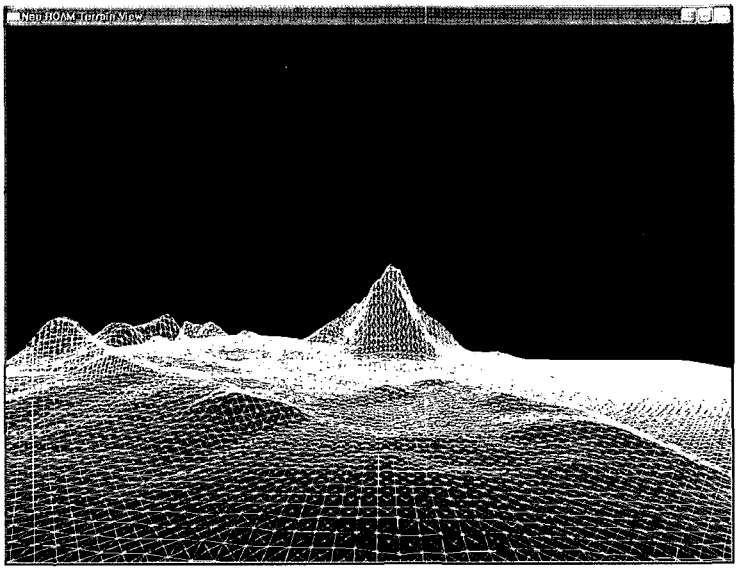

Fig. 12. Original terrain.

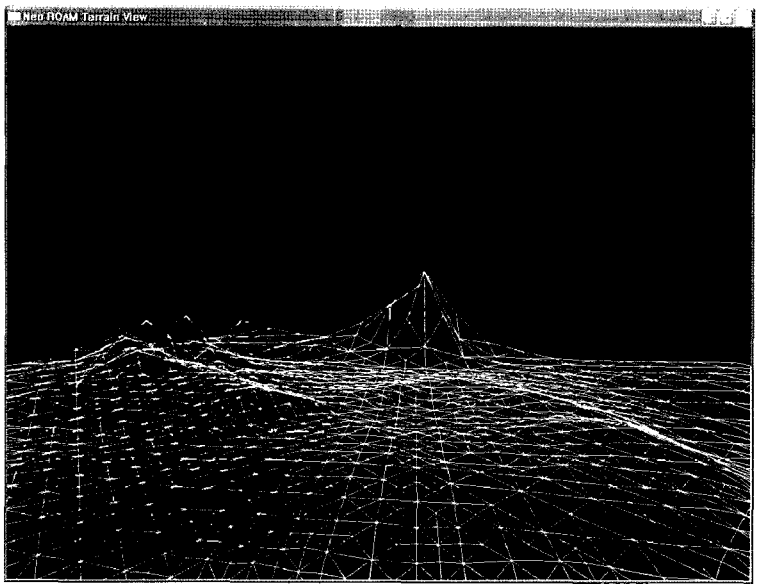

Fig. 13. Terrain with LOD.

4. 실험 및 고찰

지형을 렌더링함에 있어서 트리를 사용한 자료구조는 세분화와 단순화시에 지형을 표현하는데 불필요한 부모의 정보를 가지게 된다. 본 논문은 트리구조의 불필요한 부모정보를 배제하고 세분화와 단순화시에 필요한 현재의 지형정보만을 가지고 지형을 렌더링한다는 생각에서 출발하게 되었다. 2진 트리의 자료구조 대신에 2진트리의 최하위 노드의 수만큼의 메모리를 할당하여 분할되어있는 현재의 상태 및 위치를 저장하고, 트리의 부모정보를 저장하지 않음으로써 트리구조를 사용하는 방식보다 메모리의 양을 줄이도록 하였다.

본 논문은 Visual C++ 6.0 과 지형 표현을 위하여 OpenGL 함수를 이용하여 구현하였으며 시스템의 사양은 다음과 같다.

- CPU: Pentium-3 800MHz
- Memory : 256 M
- OS : Windows XP

제안한 자료구조와 기존의 2진 트리 자료구조를 총 9개의 Height Map을 사용하여 비교실험을 하였다. 실험에 쓰여진 지형데이터는 1024×1024크기의 Height Map으로써 0-255의 높이 값을 가진다. 삼각형 메쉬는 최대 단계 7까지 세분화되며

Table 1. Performance Evaluation (단위:ms)

	제안한 알고리즘		기존의 ROAM	
	Turn	Forward	Turn	Forward
Map A	6,529	6,780	27,660	27,669
Map B	6,500	6,780	27,689	27,710
Map C	6,519	6,780	26,177	26,238
Map D	7,020	7,100	26,158	26,238
Map E	7,031	7,090	27,780	27,750
Map F	6,469	6,790	27,740	27,709
Map G	6,729	7,041	27,830	27,850
Map H	7,080	7,120	27,860	27,900
Map I	7,080	7,130	26,197	26,276

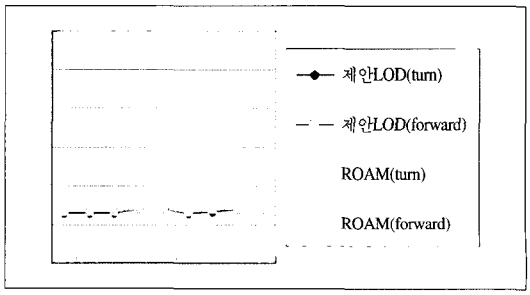

Fig. 14. LOD performance evaluation.

총 10,000프레임을 비교하였다. Fig 12과 Fig 13에 실험에 사용된 원본 지형의 일부와 LOD가 적용되었을 때의 지형을 보여주고 있다.

본 논문에서는 두 가지 방법으로 렌더링 속도를 비교하였다. 첫 번째는 Map 주위를 회전하는 방식(Turn)으로 Open GL로 렌더링하여 실행속도를 비교하였다. 두 번째는 Map을 전진과 후진하는 방식(Forward)으로 Open GL로 렌더링하여 실행속도를 비교하였다.

제안한 방법과 기존의 ROAM구조를 사용한 방법과의 속도를 비교한 값을 Table 1과 Fig. 12에서 나타내었다.

5. 결 론

기존의 2진 트리 방식은 3차원 지형을 렌더링 시에 부모 노드의 정보를 가지고 있어서 렌더링에 불필요한 자료구조를 가지는 문제점이 있다. 기존의 2진 트리 방식의 문제점을 개선하고자 본 논문에서는 트리구조의 부모정보를 제거한 자료구조를 제안하였다.

제안한 알고리즘으로 실험한 결과 메모리 효율성을 비교해보면 2진트리구조의 경우하나의 노드는 두개의 자식노드의 정보를 가지고 있으므로 하나의 자식노드의 포인터 정보를 4 Byte라고 할 때 단계 7까지의 메모리양을 계산하면 (2-1)(삼각형의 총개수)×2(자식노드의 수)×4Byte(자식노드의 포인터정보)=2040Byte의 메모리를 사용하지만 제안한 자료구조의 경우 2(하위 삼각형의 개수)×4 Byte(삼각형의 단계정보)=512Byte의 메모리만을 사용한다. 이는 기존 알고리즘에 비해 메모리는 약 1/4정도 줄인 것이다. 렌더링 속도 또한 실험에 의해서 4배 정도 향상된 것으로 조사 되었다.

향후 연구 과제로는 더 자연스러운 지형 생성을 위해서 삼각형의 분할 또는 단순화 할 때의 삼각형의 갑작스런 변화에 대응할 수 있는 geomorphing 기법이나 정점보간 등을 연구할 필요성이 있다.

사 사

본 논문은 2004년 대학 IT연구센터 육성지원사업비의 지원을 받아 연구되었음.

참고문헌

강윤식, 박우찬, 양성봉, 2000. 구역화를 이용한 디지털 격자지형데이터의 단순화 알고리즘, 정보처리학회논문지, 7(3): 935-942.

김양수, 2000. 지형데이터를 위한 효율적인 단계별 상세(LOD) 표현방법, 부산대학교

류승택, 최윤석, 안충현, 윤경현, 1996. 3차원 지형 모델링을 위한 래디얼 스위프 알고리즘, 기술과학연구소 논문집, 26(1): 133-140.

Blow, J., 2000. Terrain Rendering at High Levels of Detail, *Proceedings of Game Developers Conference 2000.*

Duchaineau, M., M. Wolinski, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. 1997. ROAMing Terrain: Real-time, Optimally Adapting Meshes, *Proceedings of the Conference on Visualization* '97, pp.203-211.

Garland M. and P. S. Heckbert, 1995. *Fast Polygonal Approximation of Terrains and Height Fields*, CMU-CS-95-181, CS Dept., Carnegie Mellon Univ.

Levenberg, J., 2002. Fast View - Dependent Level - of - Detail Rendering Using Cached Geometry, *IEEE VISUALIZATION 2002*, pp.78-82

Lindstrom, P. and V. Pascucci. 2001. Visualization of Large Terrains Made Easy, 2001, *Proceedings of IEEE Visualization 2001*, pp.363-370.

Röttger, S., 1997. *Real-Time Generation of Continuous Levels of Detail for Height Fields*. Technical Report, Universitat Erlangen-Nurnberg.