쿼드트리와 웨이블릿 변환을 이용한 실시간 지형 렌더링

Real-time Terrain Rendering using Quadtree and Wavelet Transform

한정현*, 박헌기*, 정문주*
JungHyun Han, HunKi Park, Moonju Jung

Abstract

Rendering of 3D terrain data in real-time is difficult because of its large scale. So, it is necessary to use level-of-detail(LOD) that uses fewer data, but makes almost similar image to the original. We present an algorithm for real-time LOD generation and rendering of 3D terrain data. The algorithm applies wavelet transform to the terrain data, and then generates quadtree based view-dependent LOD using wavelet coefficients that are the output of wavelet transform. It also uses frame-to-frame coherence and view culling for high frame rates.

* 성균관대학교 전기전자 및 컴퓨터공학부

1. 서론

지형 데이터를 3차원으로 가시화하고자 하는 요구는 지금까지 꾸준히 있어 왔다. 그러나 대부분의 지형 데이터는 그 크기가 상당히 크기 때문에 이를 실시간에 렌더링하는 것에 어려움이 있었다. 현재 그래픽 하드웨어의 성능 향상에도 불구하고 거대한 크기의 3차원 지형 데이터를 실시간에 렌더링하고 그 지형을 탐색하는 데는 여전히 무리가 따른다. 그러므로 더 적은 수의 데이터로도 원래의 데이터 그대로를 렌더링했을 때의 결과 이미지와 거의 차이가 없는 이미지를 만들어 낼 수 있는 LOD 기법의 적용이 필요하고, 지금까지 이에 관한 많은 연구가 있었다.

본 논문에서는 지형 데이터에 적용할 LOD 기법으로서 쿼드트리와 웨이블릿 변환을 이용한 새로운 기법을 제안한다. 본 기법은 여러 종류의 지형 데이터들 중 DEM(Digital Elevation Model)을 사용하고, 이 지형 데이터에 웨이블릿 변환을 적용한다. 지형 데이터를 나타내기 위한 자료 구조로 쿼드트리를 사용하고, 쿼드트리와 웨이블릿 변환 결과인 웨이블릿 계수값들을 이용하여 뷰(view)에 종속적인 LOD를 만들어 낸다. 또한, 이어지는 프레임간의 유사성을 이용한 렌더링 블록 리스트라는 것을 사용하고, 쿼드트리의 블록들을 기준으로 뷰 컬링을 수행함으로써 프레임률을 높인다.

본 논문의 구성은 다음과 같다. 2절은 지형에 관련된 LOD 기법에 대한 기존의 연구들을 다루고, 3절에서는 웨이블릿 변환에 대한 간단한 설명을 한다. 4절에서는 쿼드트리와 삼각화에 대한 내용을, 5절에서는 LOD를 만드는 과정인 복원에 대한 내용을 다룬다. 6절에서는 프레임률을 높이기 위한 기법들인 렌더링 블록 리스트의 사용과 뷰 컬링에 대한 설명을 하고, 7절에서는 구현 및 실험 결과를 보여 주고, 8절에서 결론을 맺는다.

2. 관련 연구

지형 데이터에 적용할 LOD 기법에 관한 연구는 지금까지 많이 있었다. Gross et al.[1]은 웨이블릿 변환을 지형 데이터에 적용하여 웨이블릿 계수값들을 정점을 제거하는데 이용하였다. 그리고 남아 있는 정점들을 사용하여 미리 만들어 둔 Look-Up 테이블을 이용한 삼각화를 하였다. 그러나 이들의 방법은 정점을 제거함에 있어서 뷰 요소가 그다지 영향을 미치지 않는다. Lindstrom et al.[2]은 쿼드트리를 자료 구조로 사용하고, 정점을 제거할 때 그 정점을 제거함으로써 생기는 스크린 상의 에러를 기준으로 삼았다. Duchaineau et al.[3]은 triangle bintree를 자료 구조로 사용하고, 역시 스크린 상의 에러를 이용하지만 정점을 제거하는 방법이 아니라 정점을 삽입하는 방법을 제안하였다. 이들의 방법은 삼각형의 병합(merge)과 분할(split)을 위한 큐를 하나씩 두어 프레임간의 유사성을 이용함으로써 프레임률을 높였다.

[2],[3],[4],[5]의 방법 모두는 크랙을 방지하기 위해서 모든 삼각형을 직각 이등변 삼각형으로 유지하는 방법을 사용하고 있다.이 방법은 인접한 삼각형 간의 레벨 차이를 1이하로 제한한다. 이로 인해 각자의 에러 기준에 의해 제거되어도 된다고 결정된 정점이 인접한 다른 삼각형의 정점으로 인해서 제거되지 못하고, 결국 세밀한 복원이 필요하지 않은 부분이 세밀하게 복원이 되는 문제가 생긴다.

이에 대한 대안으로 본 논문에서는 쿼드트리를 사용하며 인접한 블록간의 레벨 차이를 두지 않고 triangle fan을 이용하여 크랙없이 삼각화하는 방법을 제안한다. 또한 실시간의 렌더링을 위해 뷰 컬링 및 프레임간의 유사성을 이용한 렌더링 블록 리스트 등을 사용한다.

3. 웨이블릿 변환

3.1 웨이블릿 변환 기초

1차원 이미지를 예로 1차원 함수에 1차원 Haar 웨이블릿 변환을 적용하는 방법을 알아보도록 한다[6]. 1차원 이미지의 픽셀값들을 두개씩 쌍을 지어 평균을 내고, 원래의 픽셀값과 평균값과의 차이인 웨이블릿 계수값을 계산하여 평균값과 웨이블릿 계수값을 저장한다. 위의 과정을 평균값에만 반복적으로 적용하여 전체의 평균값 하나와 웨이블릿 계수값들이 나오게 되면, 그것이 바로 1차원 이미지의 Haar 웨이블릿 변환이 된다. 다음과 같은 픽셀값을 갖는 1차원 이미지를 예로 들어 보면,

[9 7 3 5]

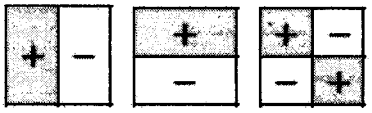
이 이미지의 웨이블릿 변환 과정과 결과는 다음과 같다.

해상도	평균값	웨이블릿 계수값
4	[9 7 3 5]	
2	[8 4]	[1 -1]
1	[6]	[2]

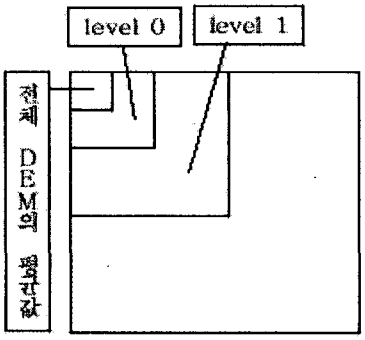
결과 ⇒ [6 2 1 -1]

웨이블릿 변환한 결과로부터 원래의 이미지로의 복원은 각 단계의 평균값들에 그 평균값에 대응되는 웨이블릿 계수값을 더하거나 뺌으로써 이루어진다.

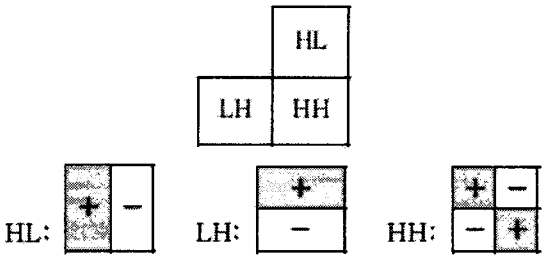
다음으로 2차원 이미지를 예로 2차원 이미지에 2차원 Haar 웨이블릿 변환을 적용하는 방법을 알아보도록 한다[6]. 2차원 Haar 웨이블릿 변환에는 standard와 nonstandard의 두 가지 방법이 있다. 먼저, standard 방법은 2차원 이미지의 각 행의 픽셀값에 1차원 Haar 웨이블릿 변환을 적용한 후, 그 결과물의 각 열에 1차원 Haar 웨이블릿 변환을 적용하는 것이다. Nonstandard 방법은 2차원 이미지의 각 행의 픽셀값을 두 개씩 쌍을 지어 평균을 내고 웨이블릿 계수값을 저장하는 과정을 한 번 수행한 후, 그 결과물의 각 열을 다시 두 개씩 쌍을 지어 평균을 내고 웨이블릿 계수값을 저장한다. 이 과정을 본래의 이미지가 전체의 평균값 하나와 웨이블릿 계수값들로 바뀌어질 때 까지 평균 값들에만 반복적으로 수행한다. 2차원 Haar 웨이블릿 변환의 웨이블릿 기저 함수들은 <그림 1>과 같다. <그림 1>에서 '+'로 표시



<그림 1> 2차원 Haar 웨이블릿 기저 함수



<그림 2> HWT이 적용된 DEM의 형태



<그림 3> 각 레벨의 영역별 기저 함수의 형태

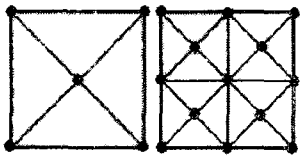
된 부분은 함수의 support 영역 중 그 부분에는 웨이블릿 계수값을 더하라는 의미이고, '-'로 표시된 부분은 함수의 support 영역 중 그

부분에는 웨이블릿 계수값을 빼라는 의미이다.

3.2 웨이블릿 변환을 이용한 전처리 단계

앞 절에서 설명한 nonstandard HWT(Haar 웨이블릿 변환)을 DEM에 적용시키면 <그림 2>와 같은 2차원 배열의 형태를 가지게 된다.

또한 각 레벨은 <그림 3>과 같은 형태를 가지고 있게 된다. <그림 3>의 각 영역 HL, LH, HH의 웨이블릿 계수값들은 <그림 1>의 웨이블릿 기저 함수를 확장 또는 축소, 평행 이동한 함수들의 계수가 된다.



레벨 0 레벨 1

<그림 4> 쿼드트리의 블록

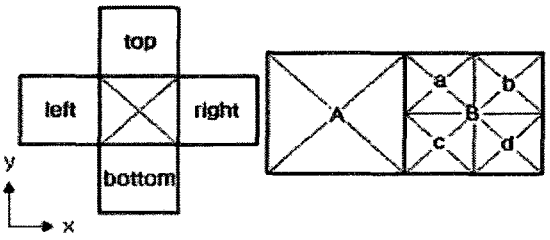
3.3 웨이블릿 계수값을 이용한 복원

일반적으로 HWT로부터 본래의 데이터를 복원하기 위해서는 모든 레벨의 웨이블릿 계수값들을 각각에 해당하는 웨이블릿 기저 함수의 영역에 더하거나 빼는 방법을 사용한다. 그러나 본래의 데이터 그대로 복원하기보다는 영역마다 다른 레벨로의 복원이 필요하기 때문에 [7]에서 제안된 방법을 사용하여 특정 영역을 특정 레벨로 복원하기 위해 사용되는 웨이블릿 계수값들을 추출하여 복원하도록 한다.

4. 쿼드트리

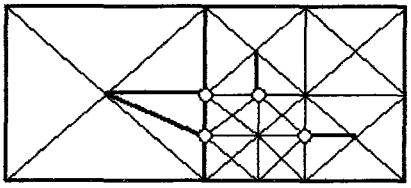
기본적으로 지형을 나타내기 위한 자료구조는 쿼드트리를 사용한다. 쿼드트리의 각 노드를 블록이라 하고 각 레벨의 블록은 <그림 4>와 같다. <그림 4>에서 진한 색으로 나타내진 영역이 각 레벨의 단위 블록이 된다. 한 블록은 <그림 4>처럼 4개의 삼각형으로 삼각화되고, 레벨 k의 블록은 레벨 k+1의 블록 4개를 자식 블록으로 갖는다.

지형의 경계를 이루는 블록을 제외한 모든블록은 각각 4개의 인접한 블록을 갖게 되고, 인접한 블록과의 관계는 <그림 5>와 같다. 어떤 블록과 인접한 블록이 반드시 그 블록과 같은 레벨일 필요는 없으며, 같은 레 벨이거나 더 낮은 레벨일 수 있다. <그림 5>에서 블록 a의 left neighbor 블록은 블록 A가 되고, right neighbor 블록은 블록 b가, bottom neighbor 블록은 블록 c가 된다. 그리고 블록 A의 right neighbor 블록은 블록 B가 된다. 이와 같은 블록간의 관계는 인접한 블록의 레벨 차이로 인해 삼각화시 생기게 되는 크랙을 제거하는데 사용된다.



<그림 5> 쿼드트리 블록의 인접 블록

쿼드 트리의 각 블록은 triangle fan으로 삼각화 되며, 인접한 블록간의 레벨 차이로 생기게 되는 크랙은 <그림 6>과 같이 서로 인접한 블록 중 레벨이 더 낮은 블록에, 레벨이 상대적으로 높은 블록에서만 가지고 있는 정점을 삽입함으로써 방지한다.

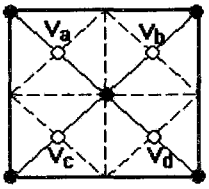


<그림 6> 크랙을 방지하기 위한 삼각화

5. LOD 생성

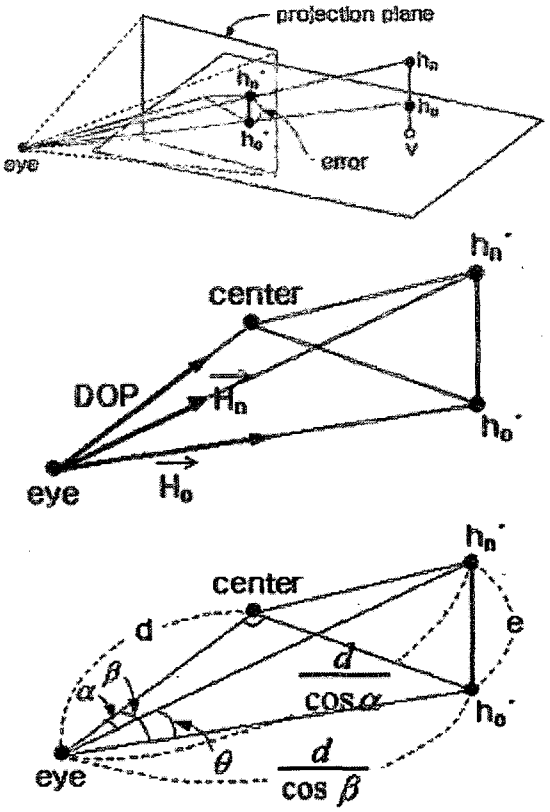
5.1 복원

지형 데이터의 HWT 결과를 사용해서 복원을 수행한다. 전체 평균값만으로 복원한 상태가 레벨 0의 블록이 된다. 전체 평균값을 avg라 하고, 웨이블릿 계수값 중 레벨 k에 속하는 계수값들을 WC_k라고 할 때 $\sum_{l=0}^{k-1} WC_l$ 과 avg를 사용해서 복원한 상태가 레벨 k의 블록이 된다. 레벨 k의 블록에서 레벨 k+1인 블록으로의 복원에는 WC_k가 사용된다. 복원은 레벨 0의 블록에서부터 시작해서 레벨을 증가시켜 나가면서 수행한다. 레벨 k의 블록에서 레벨 k+1의 블록으로 복원을 더 진행할지를 결정하기 위해서 레벨 k의 블록 내에서 레벨 k+1의 블록들의 중심 정점이 위치하게 되는 부분에서 생기게 되는 높이값의 변화량으로 에러를 계산해서 결정한다. 에러가 어떻게 계산되는지에 대한 구체적인 내용은 다음 절에서 설명한다.



<그림 7> 레벨 k의 블록에서 레벨 k+1의 블록으로 복원 시 고려 대상이 되는 정점들

<그림 7>의 v_a, v_b, v_c, v_d 에서 각각 에러를 계산하는데, 이 때 4개의 에러 중 어느 하나라도 일정 임계치 이상이면 복원을 진행하고, 4개의 에러가 모두 임계치 미만인 경우에는 더 이상 복원을 진행하지 않는다. 그리고 자신의 에러는 임계치 미만이었지만 인접한 블록들로 인해 복원이 된 블록에 대해서는 다음 레벨로의 복원을 결정할 때 고려 대상에서 제외한다. 여기서 복원을 중지하는 의미는 현재 블록의 자손 블록들에서 임계치 이상의 에러가 나오지 않게 된다는 것이고, 그것은 곧 자손 블록들에서의 높이값 변화가 현재 블록보다 작음을 나타낸다고 볼 수 있다. 대부분 웨이블릿 변환 결과인 웨이블릿 계수값들은 레벨이 높아질수록 그 크기가 작아지기 때문에([8],[9]), 부모 블록보다 자손 블록들에서의



<그림 8> 변화되는 높이값의 투영 평면에서의 에러 계산

높이값의 변화가 작다는 것이 거의 보장된다. 그러나 자손블록들에서 나타나는 높이값의 변화량들이 여러 레벨에 걸쳐서 누적될 경우, 그 값이 오히려 부모 블록에서의 변화량보다 커질 수가 있게 된다. 그러므로 이러한 것을 방지하기 위해서 각 블록마다 자신의 자손 블록들에서 생기는 높이값 변화량의 누적치 중 가장 큰 값을 저장해 둠으로써, 자신의 변화량보다 그 값이 더 클 경우는 자신의 변화량

으로 생긴 에러가 임계치 미만이라 하더라도 복원을 한 레벨 더 진행한다.

5.2 에러 계산

지형 내의 임의의 위치에서 현재의 높이값에서 변화되는 높이값에 대한 에러는 그 높이값의 변화가 투영 평면에 맺히는 길이로써 계산한다. <그림 8>에서 h_o는 현재의 높이값을, h_n은 변화된 후의 높이값을 나타낸다. <그림 8>에서 눈으로부터 투영 평면의 중심점까지를 잇는 벡터의 단위 벡터를 $\overrightarrow{DOP}$, 눈으로부터 h_n까지를 잇는 벡터의 단위 벡터를 $\overrightarrow{H_n}$이라 하고, 눈으로부터 h_o까지를 잇는 벡터의 단위 벡터를 $\overrightarrow{H_o}$라 한다. 또한 $\overrightarrow{DOP}$와 $\overrightarrow{H_n}$이 이루는 각을 α, $\overrightarrow{DOP}$와 $\overrightarrow{H_o}$가 이루는 각을 β라 하고, $\overrightarrow{H_n}$과 $\overrightarrow{H_o}$가 이루는 각을 θ라 한다. 이 때 투영 평면에 맺히는 높이 변화량의 에러 e는 다음 식에 의해서 구해진다.

$$e^2 = d^2(1/\cos^2\alpha + 1/\cos^2\beta - 2\cos\theta/\cos\alpha\cos\beta)$$

$$\cos\alpha = \overrightarrow{DOP}\cdot\overrightarrow{H_n}$$

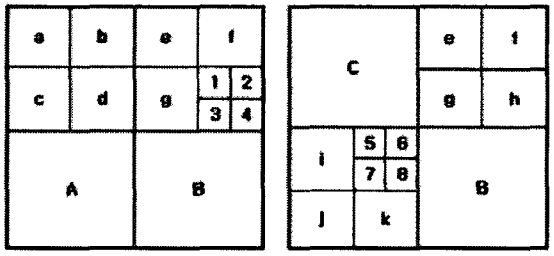
$$\cos\beta = \overrightarrow{DOP}\cdot\overrightarrow{H_o}$$

$$\cos\theta = \overrightarrow{H_n}\cdot\overrightarrow{H_o}$$

6. 프레임률을 높이기 위한 방법들

6.1 렌더링 블록 리스트

레벨 k의 블록들 중 렌더링되어질 블록들의 집합을 B_k라 하고, 쿼드트리의 최상위 레벨이 m일 때 $\bigcup_{k=0}^{m} B_k$을 렌더링 블록 리스트라 한다. 첫번째 프레임에서 렌더링 블록 리스트는 B_0를 제외하고는 모두 공집합인 상태에서 시작한다. 그리고 레벨 0에서부터 레벨 m까지 차례대로 복원을 진행해 가며 렌더링 블록 리스트가 수정된다. 블록 $b_i \in B_k$가 복원에 의해 레벨 k+1의 블록들 b_{i1}, b_{i2}, b_{i3}, b_{i4}로 대체되면, B_k에서 블록 b_i를 제거하고, 블록 b_{i1}, b_{i2}, b_{i3}, b_{i4}를 B_{k+1}에 추가한다. 첫번째 프레임이 아닌 경우에는 이전 프레임에서의 렌더링 블록 리스트를 사용해서 복원을 수행한다. 이는 이어지는 프레임의 경우 이전 프레임에서 렌더링되었던 블록이 현재 프레임에서도 렌더링될 가능성이 높다는 프레임간의 유사성을 이용하기 위함이다. 한 프레임 진행되었을 때 렌더링 블록 리스트의 변화의 예를 <그림 9>에 나타내었다.



프레임 1

B_0={ }
B_1={A, B}
B_2={a, b, c, d, e, f, g}
B_3={1, 2, 3, 4}

프레임 2

B_0={ }
B_1={B, C}
B_2={e, f, g, h i, j, k}
B_3 = {5, 6, 7, 8}

<그림 9> 렌더링 블록 리스트의 수정

첫번째 프레임에서는 더 높은 레벨로의 복원만이 고려 대상이었지만, 그 이후 프레임에서는 더 낮은 레벨로의 간략화 또한 고려 대상이 된다.

레벨 k+1의 블록 4개가 레벨 k의 블록 하나로 간략화되기 위해서는 레벨 k의 블록에서 레벨 k+1의 블록으로 복원할 때의 방법을 반대로 사용한다. 즉, 레벨 k+1 블록들이 레벨 k의 블록으로 간략화되었을 때 레벨 k+1의 블록들의 중심 정점에서 나타나는 높이값의 변화량에 대해서 에러를 계산한다. 계산된 4개의 에러가 모두 임계치 미만이면 레벨 k

의 블록으로 간략화를 수행하고, 1개의 에러라도 임계치 이상이면 간략화를 하지 않는다. 이러한 과정은 렌더링 블록 리스트에서 B_m에서 시작해서 B_0까지의 순서로 진행된다. 레벨 k+1의 블록들 b_{i1}, b_{i2}, b_{i3}, $b_{i4} \in B_{k+1}$이 레벨 k의 블록 b_i로 간략화가 된다면, B_{k+1}에서 블록 b_{i1}, b_{i2}, b_{i3}, b_{i4}을 제거하고, B_k에 b_i를 추가한다. 이 때 현재 프레임에서 B_k에 새로 추가된 b_i에 대해서는 B_k에 대해서 복원 또는 간략화를 수행할지에 대한 검사를 할 때 간략화할 수 있는지에 대한 여부만 조사한다.

6.2 뷰 컬링

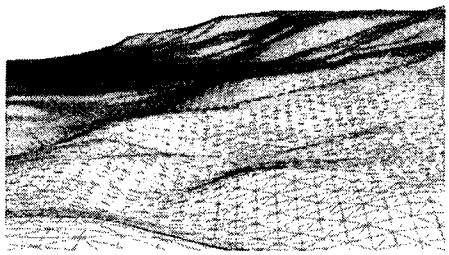
뷰 프러스텀(view frustum) 내에 들어오지 않는 블록에 대해서 컬링을 수행한다. 뷰 프러스텀을 z=0 평면 상에 투영시켰을 때 뷰 프러스텀을 구성하는 8개의 점이 만들어 내는 convex hull과 블록이 겹치는 부분이 있는지를 검사하여, 겹치는 부분이 있는 블록에 대해서만 위에서 설명한 복원 과정을 수행한다.

7. 구현 및 실험

전처리 단계에서 DEM의 HWT를 사용해서 쿼드트리를 생성한다. HWT의 웨이블릿 계수값 3개를 사용해서 4개의 값을 만들어 내기 때문에 DEM의 크기가 $2^k+1 \times 2^k+1$일 경우 완전한 쿼드트리를 생성하면 원래 약 4^k였던 데이터량이 약 $(4^{k+1}-1)/3$으로 늘어나게 된다. 그러나 굴곡이 아주 심한 산악지대를 제외한 대부분의 지형의 경우 웨이블릿 계수값들의 거의 절반 정도가 0이기 때문에 쿼드트리에서 0인 노드들을 제외하면, 데이터량이 $(2 \cdot 4^k-1)/3$ 정도로 줄어든다. 지형이 평지에 가까울수록 0인 웨이블릿 계수값들이 늘어나서 쿼드트리를 나타내기 위한 데이터량은 줄어들게 된다. 그러므로 본 논문에서 제안한 쿼드트리를 사용하면, 지형이 평지에 가까울수록 원 DEM에 비해서 더 적은 양의 데이터로도 같은 지형을 나타낼 수 있는 이득이 있다.

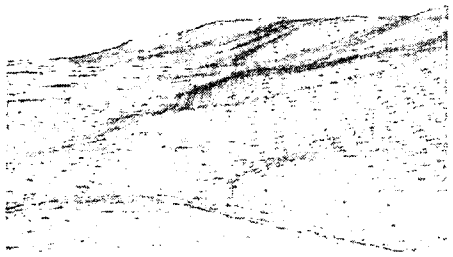
렌더링 블록 리스트는 전처리 단계에서 생성한 쿼드트리 블록의 포인터들을 가지며, 첫 프레임에 생성된 후, 매 프레임마다 수정된다.

x와 y 방향으로의 해상도가 30인 365 X 475 크기의 DEM을 $513(2^9+1) \times 513(2^9+1)$ 크기로 보정한 후 본 논문에서 제안한 기법들을 적용한 결과는 <그림 10>과 같다. <그림 10> (c),(d)는 투영 평면에서 계산되는 에러의 임계치를 0.00001로 한 결과이다. 원 DEM을 그대로 렌더링한 경우 삼각형의 개수는 약 345000개였고 평균 프레임률은 8fps(frames per second)가 나왔다. 뷰에 종속적인 LOD를 생성하여 렌더링한 경우 삼각형의 개수는 프레임당 평균 4000개 정도였고, 평균 프레임률은 20fps가 나왔다. 더 적은 수의 삼각형으로도 원 DEM과 거의 차이가 없는 결과 이미지를 만들어 냄을 볼 수 있다. [5]의 결과와 비교를 해보면 [5]에서는 86000개의 삼각형을 8000개로 줄여서 프레임률을 20fps에서 38fps로 높였는데, 본 논문에서는 실험 데이터의 삼각형 개수가 더 많으므로 처리해야 할 양이 더 많아서 프레임률이 더 낮은 것으로 생각된다.

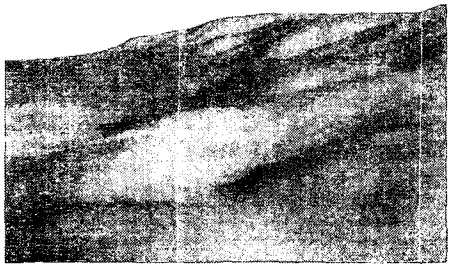


(a) 원 DEM(wireframe)

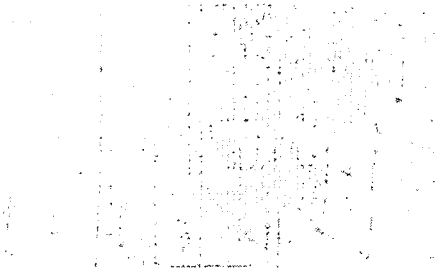
(b) 원 DEM(음영처리된 이미지)



(c) 뷰에 종속적인 LOD를 생성하여 렌더링한 지형



(d) 뷰에 종속적인 LOD를 생성하여 렌더링한 지형



(e) (c)의 top 뷰

<그림 10> 실험 결과

8. 결론 및 향후 과제

쿼드트리와 웨이블릿 변환을 이용한 LOD 기법으로 실시간의 지형 렌더링을 구현할 수 있었다. 현재 실시간 렌더링을 위해 쿼드트리와 렌더링 블록 리스트 등 유지해야 할 데이터의 양이 많은데, 이를 좀 더 줄이기 위한 연구를 앞으로 수행해야 할 것이다.

참고 문헌

[1] Gross, M., Gatti, R., and Staadt, O., "Fast Multiresolution Surface Meshing", IEEE Visualization '95, Oct. 1995, pp. 135-142.

[2] Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L. F., Faust, N., and Turner, G. A., "Real-Time, Continuous Level of Detail Rendering of Height Fields", Proceedings of SIGGRAPH '96, Aug. 1996, pp. 109-118.

[3] Duchaineau, M. A., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C., and Mineev-Weinstein, M. B., "ROAMing Terrain: Real-time Optimally Adapting Meshes", IEEE Visualization '97, Nov. 1997, pp. 81-88.

[4] Röttger, S., Heidrich, W., Slussallek, P., and Seidel, H.-P., "Real-Time Generation of Continuous Levels of Detail for Height Fields", Proceedings of the 6th International Conference in Central Europe on Computer Graphics and Visualization, Feb. 1998, pp. 315-322.

[5] Pajarola, R. B., "Large Scale Terrain Visualization Using the Restricted Quadtree Triangulation", IEEE Visualiza-tion '98, Oct. 1998, pp. 19-26.

[6] Stollnitz, E. J., DeRose, T. D., and Salesin, D. H., *Wavelets for Computer Graphics: theory and applications*. MK, 1996.

[7] 박헌기, 정문주, 한정현, "웨이블릿 변환을 이용한 지형 데이터의 전송 및 렌더링", HCI 2001 HCI.CG.VR 학술대회 발표 논문집, 2001, pp. 833-838.

[8] Goswami, J. C., and Chan, A. K., *Fundamentals of Wavelets: theory, algorithms, and applications*. John Wiley & Sons, Inc., 1999.

[9] Averbuch, A., Lazar, D., and Israeli, M., "Image Compression using Wavelet Transform and Multiresolution Decom-position" IEEE Transactions on Image Proc., 1996, 5(1):4-15.

저자소개

한정현

1988 서울대학교 컴퓨터공학과 학사
1991 University of Cincinnati Computer Science 석사
1996 University of Southern California 박사
1996~1997 National Institute of Standards and Technology(NIST)
1997~현재 성균관대학교 전기전자 및 컴퓨터공학부 조교수
관심분야: CAD/CAM, 컴퓨터 그래픽스, 컴퓨터 비전

박현기

2000 성균관대학교 전기전자 및 컴퓨터공학부 학사
2000~현재 성균관대학교 전기전자 및 컴퓨터공학부 석사과정

정문주

2001 성균관대학교 전기전자 및 컴퓨터공학부 학사
2001~현재 성균관대학교 전기전자 및 컴퓨터공학부 석사과정