# 사진트리 기반 지형렌더링을 위한 GPU기반의 적응형 상세단계 조정 방법

최인지, 신병석
인하대학교 컴퓨터 · 정보 공학과
inji@inhaian.net, bsshin@inha.ac.kr

# GPU-based Adaptive LOD control for Quadtree-Based Terrain Rendering

In-Ji Choi, Byeong-Seok Shin
Dept. of Computer Science and Information Engineering, Inha University

## 요 약

사진트리 기반의 지형 시각화 기법은 많은 응용 프로그램에서 활용되어 왔다. 하지만 전체 과정이 CPU에서 수행되기 때문에 GPU를 사용하는 다른 방법들에 비해 렌더링 성능이 떨어진다. 본 논문에서는 사진트리 기반의 지형 시각화 기법을 GPU에서 수행할 수 있도록 오차텍스처와 LOD텍스처를 제안하고, 상세단계가 적용된 사진트리 블록을 동일한 해상도의 메쉬로 채워서 렌더링 속도를 향상시키는 방법을 제안한다. 전처리 단계에서는 보편 공간에서 사진트리의 연속된 두 단계사이에서 지형의 높이 값 차이를 계산하여 오차텍스처에 저장한다. 렌더링 단계에서는 저장된 오차 값을 이용하여 투영된 오차 값을 계산하고, 그 결과를 LOD텍스처에 저장한다. LOD텍스처에 저장된 값을 이용해서 블록단위로 시각 절두체 선별을 하고 상세단계를 선택한다. 이 방법은 부하가 큰 상세 단계 선택 작업을 GPU에서 수행하고 블록단위 연산을 함으로써 작업량을 줄일 수 있다. 상세 단계가 서로 다른 블록이 인접해 있을 경우 T-정점 때문에 크랙이 발생하는데 원본 고도 데이터의 밉맵을 활용해서 이것을 제거할 수 있다.

## ABSTRACT

Quadtree-based terrain visualization methods have been used in a lot of applications. However, because most procedures are performed on the CPU, the rendering speed is slow in comparison to methods using GPU. In this paper, we present a quadtree-based terrain visualization method working on the GPU with specially designed data structure, error-texture and LOD-texture, and block-based acceleration method. In preprocessing step, we calculate errors in world space and store them to error-texture. In rendering step, we examine projected errors of error-texture and choose the detail level, then store the projected errors to LOD-texture. View frustum culling is performed as block unit using the values of error-texture and LOD-texture. This method reduces CPU load and performs time consuming jobs such as LOD selection and view frustum culling.

**Keyword** : terrain rendering, level-of-detail, GPU

접수일자 : 2008년 7월 10일
심사완료 : 2008년 8월 11일
* 본 연구는 건설교통부 첨단도시기술개발사업 – 지능형국토정보기술혁신사업과제의 연구비지원(07국토정보C05)에 의해 수행되었음.

## 1. 서 론

지형 시각화는 3D컴퓨터 게임, 비행 시뮬레이션 같은 응용 프로그램에서 널리 사용된다. 사진 트리 기반의 연속상세단계 (CLOD : continuous level-of-detail) 기법은 지형 데이터를 실시간 시각화 하는 기법 중 하나이다 [1],[2],[3]. 사진트리는 빠른 상세단계선택과 시각 절두체 선별이 가능하므로 고화질 영상을 실시간으로 생성할 수 있다. 그러나 계층적 자료구조를 사용하는데 필요한 재귀 호출과 포인터 연산은 GPU에서 수행할 수 없기 때문에 GPU의 장점인 고속 병렬연산을 할 수 없고 CPU의 부하를 줄일 수 없다.

본 논문에서는 GPU의 장점을 활용하여 고속으로 지형을 시각화하는 방법을 제안한다. 사진트리를 텍스처로 표현하여 GPU에서 처리할 수 있도록 했다. 전처리 단계에서는 모든 노드에 대해서 연속된 단계들 사이의 차이를 저장하는 오차텍스처를 생성한다. 렌더링 단계에서는 계산된 오차 값들을 투영해서 지형의 거칠기와 관찰자- 지형사이의 거리가 고려된 투영오차 값을 계산하여 LOD텍스처에 저장한다. 이것을 사용자가 지정한 허용오차 값(τ)과 비교해서 렌더링 될 블록들의 목록과 각 블록의 크기를 결정하고, 각 블록에는 미리 정의 해둔 균일한 메쉬들을 채워 렌더링 한다.

2장에서는 기존의 지형 시각화 기법에 대해 소개한다. 3장에서는 오차텍스처와 LOD텍스처를 자세히 설명하고, 제안하는 사진 트리 기반의 지형 시각화 과정을 설명한다. 4장에서는 실험 결과를 기술하고, 결론을 맺는다.

## 2. 관련 연구

Lindstrom은 사진 트리 기반의 지형 시각화 기법을 제시 하였다[1]. 그 방법에서는 지형 데이터 집합을 4개의 균일한 영역으로 재귀적으로 나누어 효율적인 연속상세단계선택과 시각 절두체 선별을 수행한다. Röttger는 사진트리기법에서 하향식방식을 사용하여 간결한 크랙 제거와 기하 모핑을 할 수 있는 방법을 제안하였다.[2] Shin은 시야에 그려지는 다각형의 개수를 조절함으로써 깜박임 없이 일정한 렌더링 속도를 유지하는 방법을 제안하였다[3]. Duchaineau는 지형을 시각화하기 위해 삼각형을 이진트리로 구성하는 기법(ROAM:real-time optimally adaptive meshes)을 제안하였다[4]. 사진트리와 이진트리 모두 계층구조를 사용하여 시점에 따른 최적의 메쉬 모델을 선택할 수 있게 한다. 하지만 계층구조는 CPU에서 처리할 수 밖에 없다.

최근, GPU가 널리 사용되고 있고 그 성능은 계속해서 좋아지고 있다. 이에 따라 GPU를 이용한 지형 시각화 기법들도 많이 소개되고 있다 [5], [6], [7]. Geometry clipmap [5] 기법은 clipmap [8] 을 지형 시각화에 적용한 것이다. 하지만 여기서는 상세단계가 몇 개의 사각 영역들에 의해 보편 공간상에서 선택되기 때문에 정확한 CLOD 계산을 하기에 어렵다.

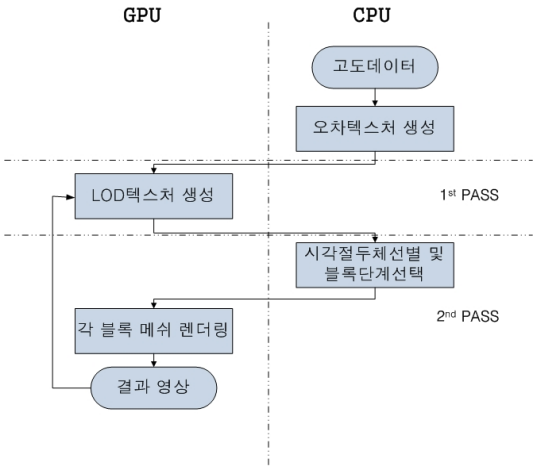
## 3. 텍스처를 이용한 사진트리 기반의 메쉬 생성

이번 장에서는 보편공간에서의 오차 값을 저장하는 오차텍스처와, 투영된 오차 값을 저장하는 LOD텍스처를 설명한다. 이러한 오차값을 정점별로 계산하는 대신에 블록 별로 계산해서 계산량을 줄이고, 블록 안에 미리정의 된 메쉬를 채워 넣는다. 또, 인접한 블록 간에 상세단계가 달라서 생기는 크랙을 원본 고도필드의 밉맵을 활용해서 제거하는 방법을 설명한다.

### 3.1 오차텍스처 생성

이 논문에서는 균일하게 샘플링 된 고도데이터인 DEM(digital elevation map)을 사용한다. 이 고도 필드(height field)는 3차원 위치정보를 가지고 있기 때문에 정점집합으로 쉽게 변환할 수 있다. 여기에서는 사진트리의 모든 단계에 대해서 상위단계의 정점과 하위단계의 변의 중점에 대해 높

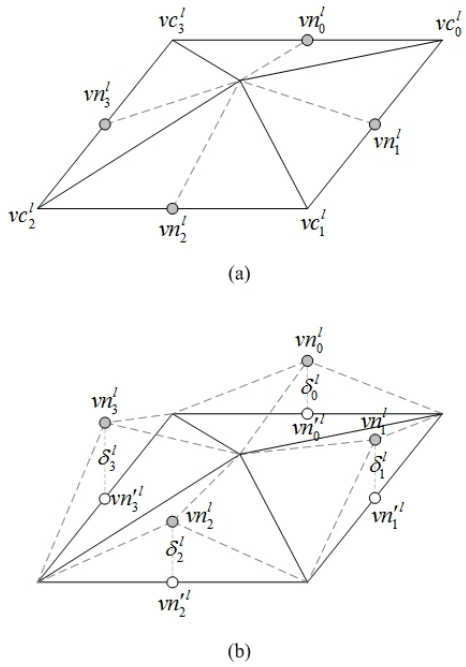
이 차이를 구한다[그림 2참고]. 이 값들을 기하오차 값이라고 부른다. 기하오차 값이 작을수록 지형은 평평한 것이고, 클수록 거친 것이다.




[그림 1] 본 논문에서 제안하는 방법의 처리절차

$\ell$ 단계에서 하나의 블록은 4개의 정점들($vn^l$)을 포함하고 있다. [그림 2]에서와 같이 블록에 포함되어 있는 정점들은 그 블록의 변에 놓이게 된다. $\ell$ 번째 단계의 정점에 대한 오차 값($\delta_k^l$, k∈[0,3])은 실제 정점과 정점에 대응하는 하위단계 변과의 수직거리를 계산함으로써 얻어진다. 블록의 오차 값($\sigma^l$)은 그 블록에 포함되어 있는 정점들의 오차 값 중 최대가 된다[식 1].

한 정점의 오차 값은 더 세밀한 단계의 정점들의 오차 값들을 포함하여야 한다. 그렇지 않으면 더 세밀한 단계가 메쉬 생성 조건을 만족함에도 불구하고 상위 단계에서 사진트리 탐색을 중단할 수 있기 때문이다. 그렇게 되면 원하는 메쉬보다 더 거친 메쉬가 생성된다. 더 거친 상세단계에서 세밀한 단계를 고려하기 위해서 더 세밀한 단계들의 오차 값과 현재 단계의 오차 값 중 최대값을 취하면 된다.[식 2]




[그림 2] (a)는 블록에 포함된 정점들을 보여준다. 각 정점들은 블록의 테두리를 이루는 변들의 중앙에 놓인다. (b)는 각 정점의 오차 값 계산을 보여준다. 오차 값은 정점과 그에 대응하는 하위단계 변과의 수직거리를 계산하여 얻는다. 점선은 더 상세한 단계의 블럭을 나타낸다.

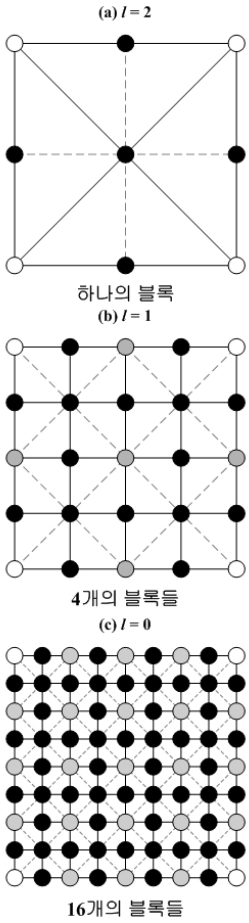
$$\sigma^l = \begin{cases} \max(\delta_k^l) & \text{if } l = 0, \\ \max(\max(\delta_k^l), \sigma^{l+1}) & \text{if } l > 0. \end{cases} \quad \text{where } k \in [0,,3] \qquad \text{[식 1]}$$

$$\delta_k^l = \begin{cases} \| v_k^l - v_k^{l+1} \| & \text{if } l = 0, \\ \max(\| v_k^l - v_k^{l+1} \|, \delta_k^{l+1}) & \text{if } l > 0. \end{cases} \quad \text{where } k \in [0,,3] \qquad \text{[식 2]}$$

이렇게 계산된 기하오차 값들을 GPU에서 처리할 수 있는 2차원 텍스처(오차텍스처)에 저장한다. 오차텍스처의 크기는 지형데이터의 크기와 같다. 오차텍스처를 생성하는 과정은 사진트리의 계층 구

조 탐색이 필요하기 때문에 CPU에서 수행되며, 한 번만 수행하면 된다.

[그림 3]에서는 사진트리 각 단계의 블록오차값을 텍스처에 저장하는 것을 보여준다. 각 단계에서 블록의 테두리변 중점의 오차 값을 계산하고 그 오차 값과 전 단계의 오차 값 중 최대 값을 블록의 중심점에 저장한다. 각 단계에서의 오차 값들의 위치는 다른 단계의 오차 값들의 저장위치와 겹치지 않는다. 그 이유는 특정 단계에서의 정점은 그보다 한 단계 더 세밀한 단계에서 구석 정점이 되기 때문이다. 따라서 전체 상세단계에서의 오차 값들은 하나의 2차원 텍스처에 저장될 수 있다.

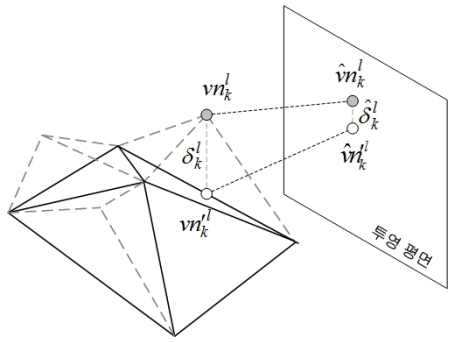



[그림 3] 지형 데이터의 해상도가 9×9일 때 오차 텍스처 생성. 검은색 원은 현재 단계에서의 오차 값을, 회색 원은 현재 단계보다 거친 단계의 오차 값들을 나타낸다.

## 3.2 LOD텍스처 생성

보편 공간상에서 계산된 기하오차 값들을 투영해서 투영평면에서의 오차 값을 계산하고 LOD텍스처에 저장한다. 이 값들(투영오차)을 이용하여 렌더링 될 블록의 크기를 결정한다. LOD텍스처를 만들 때는 계층 구조정보가 필요 없기 때문에 GPU의 프래그먼트 셰이더로 처리할 수 있다.

LOD텍스처는 생성 단계에서는 지형의 거침정도와 시점과의 거리를 고려해서 상세단계를 선택한다. [그림 4]의 $vn_k^l$과 $vn'^l_k$를 각각 월드-뷰-프로젝션-뷰포트 변환 행렬로 변환하면 투영된 정점 $\hat{v}n_k^l$과 $\hat{v}n'^l_k$를 얻어낼 수 있다.




[그림 4] 화면 공간상의 오차 값 계산

이 두 점 사이의 거리 값 $\hat{\delta}_k^l$가 투영오차 값이 된다. 이 투영오차는 연속된 두 단계의 기하오차 값이 최종 영상에 영향을 미치는 픽셀의 개수를 의미한다.

## 3.3 블록 단계 선택

Chang[10]에서 언급된 방법은 정점단위로 상세단계를 계산하고 메쉬를 생성하기 때문에 렌더링 과정에서 병목현상이 발생한다. 이 문제를 해결하

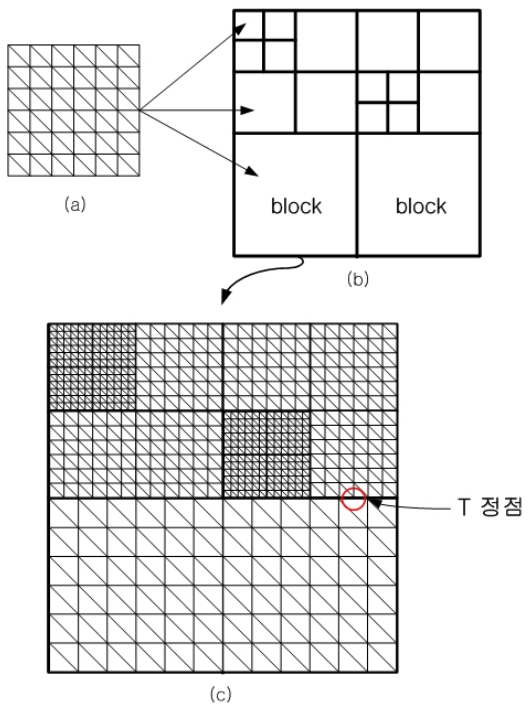
기 위해서 사진트리의 최하위 노드를 정점으로 하는 대신에 여러개의 정점을 포함하는 일정 크기의 블록으로 하고, 블록 단위로 상세 단계를 계산한다. 최종 단계에서 이 블록에 미리 정의된 메쉬를 채워서 렌더링한다.

먼저, 하향식 탐색방법으로 전체 지형 데이터를 재귀적으로 4개의 노드들로 나눠가면서 시각 절두체 선별을 수행하고 선별된 부분에 대해서만 블록의 상세 단계를 계산한다. 시각 절두체 선별은 가장 큰 블록의 네 모서리 점을 이용해서 검사한다. 네 점 중 하나라도 시각 절두체에 포함되어 있으면 해당 블록이 일부 포함된다고 판단하고 자식 블록에 대해 검사한다. 반면에 블록의 네 점 모두가 시각절두체에 포함되어있으면 해당 블록의 투영오차값을 이용해서 블록의 크기를 결정한다. 사용자가 정한 허용오차 값(τ)과 투영오차 값을 비교해서 투영오차 값이 크다면, 하위 단계의 블록을 선택한다. 그렇지 않다면 상위 단계의 블록을 최종 지형 렌더링 목록에 포함 시킨다. 이 논문의 방법에서는 지형의 한 점을 포함하는 최하위 단계의 블록은 필요하지 않으므로 블록의 최소크기에 해당하는 단계까지만 계산한다.

최종 렌더링 목록에 포함된 블록들은 각각 자신의 위치와 크기정보를 가지고 있다. 이 값들을 이용해서 3.5절에서 소개될 크랙제거에서 사용할 레벨텍스처(level texture)를 생성한다. 레벨텍스처에 대해서는 3.5절에서 자세히 설명한다.

## 3.4 각 블록에 동일 해상도의 메쉬 채우기

위 단계를 모두 거치면 렌더링 되어야 할 블록들이 선택된다. 각 블록의 위치와 크기 정보에 따라 정점의 위치를 변경함으로써, 미리 저장해둔 메쉬를 균일하게 적용한다. 이 메쉬의 정점의 개수는 항상 일정하지만 블록의 크기는 각각 다르기 때문에 메쉬가 채워지는 블록의 크기에 따라 개별 삼각형의 크기가 달라진다. 결과적으로 지형의 거칠기와 시점과의 거리가 고려된 메쉬가 생성된다[그림 5]. 생성된 메쉬에 정점 셰이더의 VTF (Vertex Texture Fetch)기능을 이용해서 정점의 높이값을 적용한다.

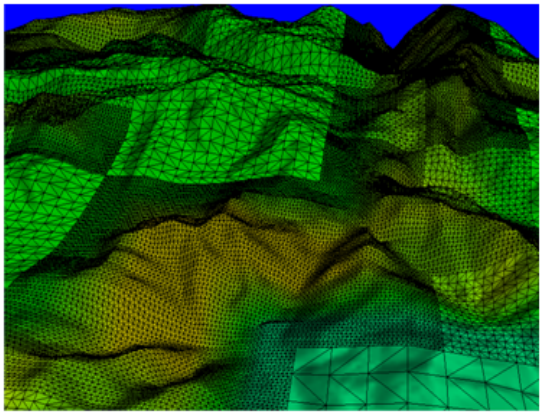



[그림 5] (a) 미리 정의된 메쉬. (b) 각 블록별 LOD 레벨이 결정된 지형데이터. (c) 블록별로 미리 정의된 메쉬가 채워진 지형데이터

## 3.5 크랙 제거

[그림 5]에서 볼 수 있듯이 인접한 블록의 상세 단계가 다를 경우 T-정점이 생겨 크랙이 발생할 수 있다. 이것은 성긴 메쉬의 변과 세밀한 메쉬의 정점이 만날 때 서로의 높이가 다를 수 있기 때문이다. 따라서 이 점에서 정점이 변에 위치하도록 하면 크랙은 제거 된다. 높이를 맞춰 주기 위해서 각 블록의 상세정도를 알 수 있는 레벨텍스처와 입력된 고도필드의 밉맵을 생성한다. 레벨텍스처에는 각 블록의 밉맵레벨 값을 저장하는데, 레벨텍스처의 크기는 최하위 노드의 블록 크기에 반비례한다. 각 정점에대해 레벨텍스처를 이용해서 자신과 자신이 속한 블록에 인접한 블록들의 밉맵레벨을 알아내고, 그 중 최대 값에 해당하는 단계의 밉맵

에서 값을 읽어 정점의 높이 값을 적용한다. 그러면 T-정점들의 높이 값은 인접한 블록과 같은 레벨의 밉맵에서 값을 가져오기 때문에 변의 높이와 같아진다. 미리 정의된 메쉬의 크기는 밉맵의 해상도와 일치하기 위해서 최하위 블록의 크기에 들어가는 메쉬가 원본 높이맵의 값들과 일대일 대응이 되도록 해야한다.

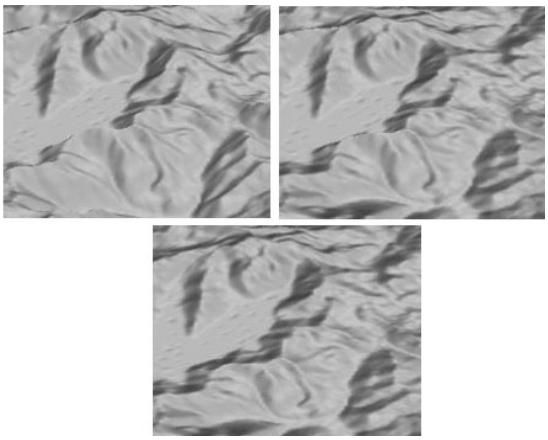


[그림 6] 밉맵을 적용하여 크랙을 제거한 모습 각 색상은 밉맵 레벨을 보여준다. ($\tau$ = 1.0)

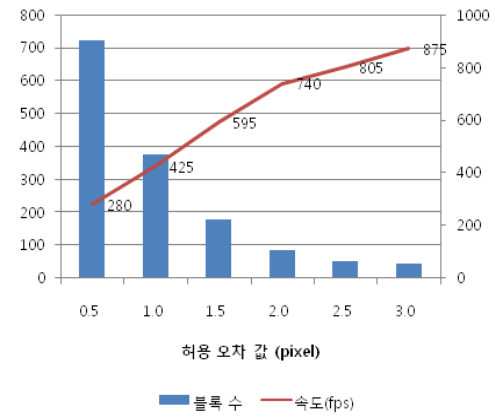
## 4. 실험 결과

실험은 Intel Core2Duo 6400 2.13GHz 에 2GB 주 메모리를 갖는 시스템에서 수행되었다. 그래픽스 가속기는 NVIDIA GeForce™ 8800Ultra를 사용하였고, DirectX9 라이브러리를 사용하였다. 뷰포트의 크기는 800×600로 하였다. 사용된 데이터로는 Puget Sound와 Grand Canyon (512×512)이다. 블록 당 메쉬의 크기는 17×17로 하였다. [그림 8] 에서 보면, 허용오차 값을 크게 할 수록 렌더링되는 블록의 수가 줄어들게 되고, 전체 렌더링 속도는 높아지게 된다. 하지만, 허용 오차가 커질수록 화질이 떨어짐을 확인할 수 있었다. [그림 7]

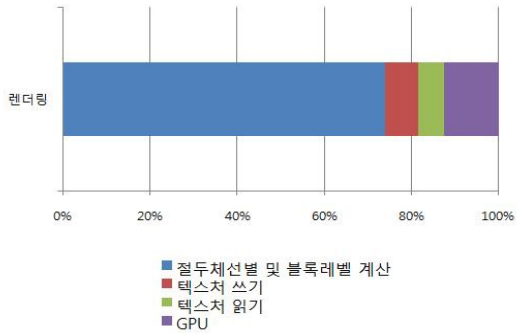
또, 그래픽스 하드웨어는 32-bit텍스처포맷을 사용하기 때문에 8-bit만 사용하는 오차텍스처, LOD 텍스처는 메모리의 낭비를 유발한다. 따라서 이 텍스처들의 한 픽셀 당 4개의 샘플을 저장하였다.



[그림 7] 허용오차에 따른 화질비교 (좌상) $\tau$ =0.5, (우상) $\tau$ =1.0, (하) $\tau$ =1.5




[그림 8] 오차값에 따른 블록 수와 속도

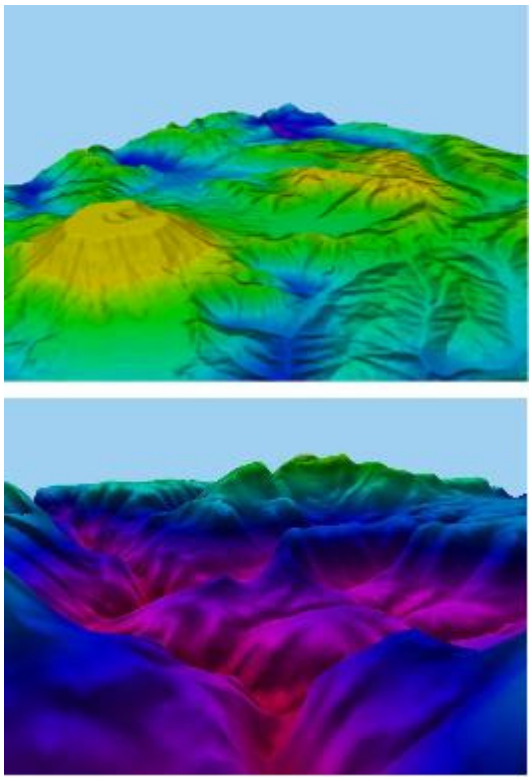



[그림 9] 렌더링 속도 분석

또, 전체 렌더링 속도의 90~95%는 GPU Idle 상태이므로 CPU 병목임을 알 수 있었고, [그림 9]에서 볼 수 있듯이 블록에 대해 절두체선별과 레벨을 결정하는 과정에서 병목이 발생하였다.

## 5. 결 론

본 논문에서는 오차텍스처, LOD텍스처, 레벨텍스처를 이용하여 사진 트리 기반의 지형 시각화 기법에 GPU를 활용하는 방법을 제안하였다. 텍스처는 GPU에서 처리하기에 적합한 데이터 구조이므로 가속효과를 높일 수 있다. GPU에서 수행되는 단계는 LOD텍스처를 갱신하는 과정, 메쉬의 배치와 지형의 크랙을 제거하는 과정이다. 오차텍스처를 이용해서 GPU에서 빠르게 투영오차를 구할 수 있었고, 블록기반으로 처리 하면서 시각 절두체 선별과 허용오차와의 비교연산을 빠르게 수행할 수 있었다. 하지만 시점이 바뀔 때마다 블록의 투영오차 값이 바뀌기 때문에 매 프레임마다 LOD 텍스처를 갱신해야한다. 그래서 갱신된 값으로 블록 단계를 결정하기 위해서 매 프레임마다 CPU에서 LOD 텍스처를 이용해서 시각절두체선별과 블록레벨을 결정해야한다. 이 부분이 렌더링 과정중에 CPU에서 이루어지기 때문에 병목현상이 발생한다. 본 연구에서는 비교연산을 블록단위로 수행함으로써 정점단위로 수행하던 방법에 비해 속도 향상이 있었다. 한 블록에 17*17개의 정점이 있다고 하면 이 정점들의 레벨을 결정하기 위해서 매 프레임마다 수행되는 계산이 한 번에 수행되고, 크랙을 제거하기위해 수행했던 메쉬정제화 과정을 GPU에서 수행하면서 전체 렌더링 속도가 약 4배 향상되었다.



[그림 10] 결과 화면. (위: Puget Sound (512×512), 아래: Grandcanyon(512×512))

블록의 상세단계로 시점과의 거리와 지형의 굴곡을 고려하여 선택하고 미리 정의해둔 메쉬로 그린다. 또, 사진트리 지형 렌더링에서 문제가 되는 크랙 제거를 적은 연산으로 빠르게 수행할 수 있었다.

## 참고문헌

[1] Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Faust, N., Turner, G.: Real-time, Continuous Level-of-Detail Rendering of Height Fields. Proceedings of ACM SIGGRAPH 96, Addison Wesley (1996) 109-118

[2] Röttger, S., Heidrich, W., Slusallek, P., Seidel, H.: Real-Time Generation of Continuous Levels of Detail for Height Fields. Proceedings of 6th International Conference in Central Europeon Computer Graphics and Visualization

(1998) 315-322
[3] Shin, B., Choi, E.: An Efficient CLOD Method for Large-Scale Terrain Visualization. Proceedings of The Entertainment Computing - ICEC 2004. Lecture Notes in Computer Science, Vol. 3166. Springer, Eindhoven, The Netherlands (2004) 592-597
[4] Duchaineau, M., Wolinsky, M., Sigeti, D., Miller, M., Aldrich, C., Mineev-Weinstein, M.: ROAMing Terrain: Real-time Optimally Adapting Meshes. Proceedings of IEEE Visualization 97 (1997) 81-88
[5] Losasso, F., Hoppe, H.: Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids. Proceedings of the 2004 SIGGRAPH Conference, Vol. 23 (2004) 769-776
[6] Schneider, J., Westermann, R.: GPU-Friendly High-Quality Terrain Rendering. Proceedings of Journal of WSCG, Plzen-Bory Czech Republic (2006) 49-56
[7] Agrawal, A., Radhakrishna, M., Joshi, R.C.: Using Programmable GPU Hardware for Increased Visual Effects and Photorealism in Large-Scale Terrain Visualization. Proceedings of Map India (2006)
[8] Tanner, C., Migdal, C., Jones, M.: The clipmap: A Virtual Mipmap. Proceedings of the 25th annual conference on Computer graphics and interactive techniques (1998) 151-158
[9] Pajarola, R.: Large Scale Terrain Visualization Using The Restricted Quadtree Triangulation. Proceedings of IEEE Visualization 98 (1998) 19-26
[10] Chang, H., Shin, B.: Hardware Acceleration of Terrain Visualization Using ef-Buffers ISCIS (2006)
[11] Yotam, L., Zvi, K., Jihad, El. : Seamless Patches for GPU-Based Terrain Rendering WSCG(2007)

**최인지** (In-Ji Choi)

2007년 2월 인하대학교 컴퓨터공학부(학사)
2007년～현재 인하대학교 정보공학부(석사)

관심분야 : 지형렌더링

**신병석** (Byeong-Seok Shin)

1990년 2월 서울대학교 컴퓨터공학과(학사)
1992년 2월 서울대학교 컴퓨터공학과(석사)
1997년 2월 서울대학교 컴퓨터공학과(박사)
2000년～현재 인하대학교 컴퓨터공학부 부교수

관심분야 : 실시간 렌더링, 볼륨 그래픽스, 의료 영상