항법정보와 실시간 업데이트 지형 데이터를 사용한 3D 지형 재구축 시스템

백인선*, 엄기현**, 조경은**
동국대학교 영상대학원 멀티미디어학과*, 영상미디어대학 멀티미디어공학과**
bt2103@naver.com, {khum, cke}@dongguk.edu

A 3D Terrain Reconstruction System using Navigation Information and Realtime-Updated Terrain Data

In-Sun Baek*, Ky-Hyun Um**, Kyung-Eun Cho**
Dept. of Multimedia, Graduate School of Digital Media, Dongguk University**
Dept. of Multimedia Engineering, Dongguk University**

요 약

게임 캐릭터와 객체들이 상호작용하는 지형은 가상세계를 구성하기 위한 필수 요소이다. 지형을 제작할 때 많은 갱신 작업과 시간이 들어가는 문제점이 발생한다. 본 논문에서는 실제 지형을 촬영한 데이터로부터 가상 공간의 3D 지형을 생성하기 위한 3D 지형 재구축 시스템을 제안한다. 제안 시스템에서는 스테레오 카메라로 촬영하고, 레이저 스캐너로 실측한 3차원 지형 데이터를 기반으로 생성된 그리드 기반의 높이 맵(Height Map)과 로봇의 항법정보 중 z축과 x축 방향 벡터를 이용해 가상공간의 중심인 월드좌표계에 맞게 로테이션을 수행하여 축의 방향을 일치시키고, 로봇 중심의 좌표계에서 월드 좌표계로의 이동 벡터를 각 포인트에 더하여 최종적으로 월드좌표계에 맞게 변환한다. 이후 무방향성 그래프를 사용하여 지형 데이터를 관리하면서 가상공간에서 필요한 부분에만 동적으로 지형 메쉬를 생성한다. 이때 지형 데이터의 오류를 보정하여 메쉬를 올바르게 갱신한다. 실험에서는 제안 시스템이 지형 재구축을 완료할 때까지 일정한 주기로 FPS를 확인하고, 완성된 지형을 가시화하여 품질을 검토하였다. 지형의 전체 크기를 알 수 없거나, 실시간으로 지형의 크기가 변화하는 환경에서는, 제안된 시스템이 쿼드트리를 사용한 지형 관리보다 지형 크기가 작을 때 3배정도의 높은 FPS를 보이나, 지형이 아주 클 때는 평균 40% 정도 나은 실행 성능을 가진다. 최종적으로는, 실측한 지형의 모양을 그대로 유지하면서 가시화하고 있다. 본 연구에서 제안한 시스템을 이용하여 게임에 이용할 지형 데이터를 실시간으로 자동 생성하여 게임에 이용하거나, 실제 지형을 배경으로 필요한 영화의 CG 작업에 활용하는 등의 응용 방안을 고려해 볼 수 있다.

ABSTRACT

A terrain is an essential slement for constructing a virtual world in which game characters and objects make various interactions with one another. Creating a terrain requires a great deal of time and repetitive editing processes. This paper presents a 3D terrain reconstruction system to create 3D terrain in virtual space based on real terrain data. In this system, it converts the coordinate system of the height maps which are generated from a stereo camera and a laser scanner from global GPS into 3D world using the x and z axis vectors of the global GPS coordinate system. It calculates the movement vectors and the rotation matrices frame by frame. Terrain meshes are dynamically generated and rendered in the virtual areas which are represented in an undirected graph. The rendering meshes are exactly created and updated by correcting terrain data errors. In our experiments, the FPS of the system was regularly checked until the terrain was reconstructed by our system, and the visualization quality of the terrain was reviewed. As a result, our system shows that it has 3 times higher FPS than other terrain management systems with Quadtree for small area, improves 40% than others for large area. The visualization of terrain data maintains the same shape as the contour of real terrain. This system could be used for the terrain system of realtime 3D games to generate terrain on real time, and for the terrain design work of CG Movies.

Keywords : virtual world(가상세계), 3D terrain reconstruction system(3D 지형 재구축 시스템), undirected graph(무방향성 그래프)

접수일자 : 2010년 10월 04일, 일차수정 : 2010년 11월 08일, 심사완료 : 2010년 11월 30일
교신저자(Corresponding Author) : 조경은

1. 서 론

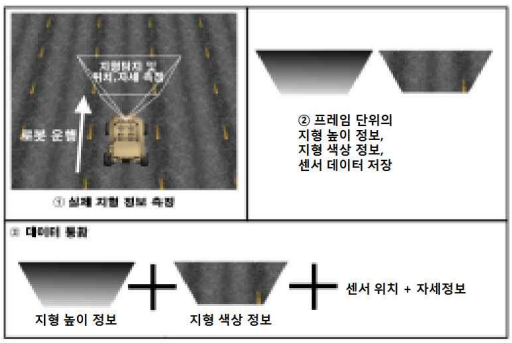
최근 개발되는 게임에서는 실물 도시나 마을을 모델로 삼아 실제라고 느낄 만한 게임상의 월드를 만드는 경향이 커지고 있다. 이러한 게임상의 지형은 실제 지형의 사진을 찍어 원화를 그리고, 그것을 바탕으로 지형 폴리곤(Polygon) 모델을 만든다.

이러한 지형 제작 방식들은 공통적으로 많은 반복 작업과 경우에 따라 장기간의 시간이 필요하다. 특히 지형을 세밀하게 표현할수록 작업량이 늘어나게 된다. 가상공간의 지형 생성 방법 중에 실측영상으로부터 모델을 만드는 방법들은 실제 지형의 모습을 재현하는 것에만 초점이 맞추어져 있어서 게임상에 그대로 사용하기에는 지형 크기 등의 문제로 적합하지 않다[1,2,3].

본 논문에서는 이러한 반복 작업과 시간을 들이는 불편함, 그리고 실데이타를 이용한 지형 재구성상의 기존 문제점들을 개선하기 위하여, 실측 지형 데이터로부터 가상 공간상의 지형 모델을 생성하는 기법을 제안한다. 이 기법은 실제 지형을 측정한 지형 데이터를 실시간에 프레임 단위로 적재하여 가상공간에 렌더링(rendering)하기 위한 3D 메쉬(mesh)를 생성하고, 갱신한다. 이 논문에서는 지형 데이터로부터 가상공간의 3D모델을 생성해내는 방안을 설명한다. 색상 정보와 지형 형태 정보를 통합하여 가상공간에서 볼 수 있는 완전한 지형 메쉬를 만들어 렌더링한다.

본 논문에서 지형 데이터는 [그림 1]과 같이 스테레오 카메라와 레이저 스캐너를 부착한 자동차가 움직이면서 실제 지형을 촬영하고 주사한(scan) 것을 실측 데이터로 이용한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문과 관련된 지형 메쉬를 생성하는 기법과 관리하는 기법을 살펴보고, 현재의 지형 관리 방법들이 본 연구에 그대로 적용되었을 때의 문제점을 설명한다. 3장에서는 실시간에 지형 데이터를 로드하여 각 센서 데이터의 위치 좌표를 계산하는 지형 재배치 방안을 설명한다. 제4장에서는 지형 관리 방법과 메쉬를 생성하고 갱신하는 방법을 설명한다. 5장에서는 본 논문에서 제시한 방법과 기존 방법과의 성능을 실험을 통하여 비교 분석한다. 6장에서는 결론을 서술한다.



[그림 1] 지형 데이터의 실측 방법

2. 관련 연구

가상공간의 지형을 만드는 방법과 만든 지형을 관리하기 위한 방법은 여러 가지가 있다. 이 장에서는 본 논문과 관련된 지형을 생성하는 방법과 생성된 지형을 관리하는 방법에 대한 기존 연구를 설명한다.

2.1 지형 생성 기법

가상공간에서는 점, 선, 면 등으로 물체를 표현한다. 그중 가장 많이 쓰이는 것은 면이다. 게임 같은 가상공간에서 지형은 대체로 무수히 많은 삼각형들의 집합으로 이루어진다. 삼각형을 구성하는 방법은 프로그래머가 위치 값을 직접 입력하는 방식, 3DS Max나 Maya같은 폴리곤 모델링 툴을 이용하는 방식, 영상이나 이미지로부터 계산을 통하여 모델의 형태를 계산해내는 방식[4] 등 여러 가지가 있다.

지형을 생성하는 방식이나 지형을 관리하는 방식에 따라 실외지형과 실내지형으로 나뉜다. 실외지형을 생성하는 방식은 주로 높이맵(Heightmap)

방식을 쓴다[5]. 그 외에도 이미 만들어진 지형에 변화를 가하여 새로운 지형을 생성하는 방식으로 프렉탈(fractal) 지형 생성 방식이 있다[6].

가상공간의 지형을 생성하는 방법 중에 실측영상으로부터 모델을 만드는 방법이 있다. 한 단위 프레임의 지형은 실제 지형을 촬영하는 센서의 측정 거리에 따라 지형의 크기와 정점의 개수, 넓이 등이 달라진다. 그래서 프레임이 진행됨에 따라 지형 모델 메쉬를 누적시키고 알맞은 값을 추출하여 하나의 완전한 메쉬로 만드는 과정과 만들어진 메쉬를 관리하는 방법에 대한 연구가 필요하다.

2.2 지형 관리 기법

지형을 관리하는 기법도 실외지형과 실내지형에서 쓰이는 기법들로 분류된다. 실내지형을 관리하기 위해 쓰이는 대표적인 자료구조로는 BSP 트리를 들 수 있다[7]. 실외지형에 대해서는 쿼드트리(Quadtree)나 옥트리(Octree)가 많이 쓰인다.

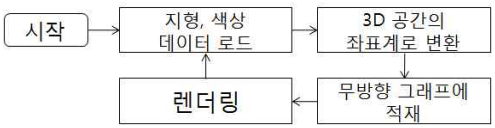
쿼드트리는 지형의 최대 크기를 알고 있을 때 공간을 4분면으로 분할하는 방식이다. 각각의 셀은 4개의 자식 노드를 갖게 되며 각 자식 노드가 분할 기준이 되는 최소 크기가 될 때까지 재귀분할을 한다[8,9]. 지형의 최대 크기와 쿼드트리 분할을 위한 최소 (2^n+1)의 크기가 있어야 한다[9]. 이 쿼드트리는 특정 영역을 탐색할 때 유용하다. [그림 4]와 같이 방향성 그래프 구조는 탐색할 때, 탐색 시작 루트 노드 R이 정해져 있다. 옥트리는 각각의 셀을 8개로 분할한다. 옥트리는 쿼드트리보다는 더 많은 메모리 공간을 차지하며 좀 더 정확한 탐색이나 공간 구성을 위해 사용된다.

이 지형 관리 구조들은 지형이 미리 만들어지나 지형 데이터를 실시간에 갱신하는 것을 염두에 두고 만든 것이 아니기 때문에, 본 논문의 연구 환경에서처럼 가변적인 지형관리 기법으로는 적합하지 않다. 쿼드트리 구조를 그대로 이용할 경우 노드의 탐색이나 메쉬의 정점을 갱신하는 부하가 커진다. 옥트리를 이용해도 지형 크기를 알아야 하는 조건에 맞지 않고, 갱신을 위한 탐색 시간은 쿼드트리보다 더 오래 걸리게 된다.

지형 렌더링 성능을 높이기 위한 대표적인 방법으로는 세부수준(LOD) 표현이나 실시간 최적화 적응 메쉬(ROAM) 등이 있다[10]. 이 기법들은 쿼드트리, 옥트리 등의 구조와 결합되어 사용되어야 성능이 극대화된다.

3. 가상 공간상의 지형 재배치

실측된 지형 데이터로부터 가상공간의 3D 지형 모델을 만드는 방법은, 먼저 센서로 주사하는 실제 지형 데이터를 생성하여 프레임 단위로 저장하고, 이 지형 데이터를 가상공간의 3D 메쉬로 재구성한다. 가상공간의 지형 모델을 만드는 작업은, 실측된 지형 데이터를 저장한 지형 데이터의 재구성, 가상공간에서의 지형 데이터의 위치 계산, 가상공간 상에의 배치, 렌더링되는 지형 메쉬 갱신 등이다. 이 과정들은, 한 단위 프레임의 지형 데이터가 가상공간의 지형에 배치되는 과정이다. 이들을 반복하여 실행한다. [그림 2]는 3D 지형을 재구축하는 과정을 흐름도로 나타낸 것이다. 이 장에서는 지형 데이터를 읽어와 메모리에 적재한 후 진행되는 재배열과 위치 좌표 변환 과정을 설명한다.



[그림 2] 3D 지형 재구축 과정의 흐름도

3.1 지형 데이터의 로드

지형 데이터는 300×300의 크기인 프레임이 실측되는 순서대로 순차적으로 저장된다. 지형의 높이 값만 저장한다. 높이 값이 없는 부분은 렌더링에서 제외하기 위해 실측 데이터에서 생생될 수 없는 값인 0x8000으로 저장한다. 300×300개의 요소가 1차원 배열로 저장된 초기 지형 데이터를 읽

어들여 2차원 배열의 형식으로 저장한다. 이들중에서 유효 값을 걸러내고 지형 데이터와 색상 데이터의 프레임 매칭을 실행한다.

한 프레임의 지형 데이터는 센서정보, 지형 형태 정보, 지형 색상 정보로 구성된다. 센서정보는 실제 지형을 촬영하는 센서의 실제 위치와 자세 정보이다. 지형 형태 정보는 센서가 탐지한 지형의 높이정보이다. 지형 색상 정보는 탐지된 높이에 매치되는 각 지형의 탐지된 부분의 색상 값이다. 색상정보는 스테레오 이미지로부터 추출한 것이며 지형 데이터를 계산할 때 지형 형태정보와 통합한다.

읽어들인 데이터는 다시 2차원 배열의 형태로 처리하게 된다. 가로 x축 세로 z축을 기준으로 데이터가 들어있는 부분만 y값으로 처리하여 일반적인 지형생성과정인 높이맵(Heightmap) 방식으로 지 형데이타를 계산한다. 지형 데이터는 탐지된 지형의 높이값과 해당 높이값에 맞는 색상정보의 2가지로 구성되어 있다. 각각의 정보를 메모리에 적재 한 후에 가상월드에 배치한다.

지형 데이터를 읽어 들일 때 지형 데이터의 분석이 필요하다. 센서를 부착한 자동차의 이동 속도에 따라 같은 위치나 비슷한 위치에 지형 정보가 중복으로 저장될 수 있기 때문이다. 지형 데이터는 (식 1)을 사용한 좌표계의 가상공간으로 옮겨진다. 옮겨진 지형 데이터를 클러스터링 과정을 통하여 비슷한 위치의 값들 중 올바른 값을 판별한다. (식 3)을 이용하여 센서가 탐지한 정보도 가까운 곳을 탐지한 정보가 더 정확하다고 판단한다.

P가 현재 자동차의 위치일 때, $R_x(\Phi)$은 pitch 값, $R_y(k)$는 yaw값, $R_z(w)$는 roll값이고 각각의 R()은 (식 2)로 표현된다[11].

$$R_x(-\Phi)R_z(-w)R_y(-k)(W-P)=D \quad \text{(식 1)}$$
$$W=R_y(k)R_z(W)R_x(\Phi)D+P$$

$$R_x(\Phi)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix},\ R_y(k)=\begin{bmatrix}\cos k & 0 & \sin k\\ 0 & 1 & 0\\ -\sin k & 0 & \cos k\end{bmatrix}, \text{(식2)}$$
$$R_z(w)=\begin{bmatrix}\cos w & -\sin w & 0\\ \sin w & \cos w & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$h_m=\begin{bmatrix}h_1 & m=1\\ \frac{1}{2}(h_{m-1}+h_m)=\sum_{i=1}^{m}(\frac{1}{2})^{m-i+1} & m>1\end{bmatrix} \quad \text{(식 3)}$$

3.2 센서 데이터의 위치 좌표 계산

본 논문에서 제안하는 시스템에서는 가상 공간상의 원점 (0,0,0)을 기준으로 지형을 생성한다. 맨 처음 프레임의 센서 위치 값을 원점으로 정하고, 다음 프레임부터 이동한 값을 처음 프레임과 (식 4)와 같이 계산하여 위치정보로 이용한다[11].

$$P_f=P_{w1}-P_{w1} \quad \text{(식 4)}$$
$$P_{current}=P_{wc}-P_{w1}$$

P_f : 실세계에서 맨 처음 측정된 위치 정보를 가상공간의 좌표 정보로 변환한 값

P_{w1} : 실세계에서 맨 처음 측정된 값

$P_{current}$: 현재 프레임에 대해 측정 정보를 가상공간의 좌표 정보로 변환한 값

P_{wc} : 현재 프레임에 대해 측정된 위치 값

자세정보는 위치 정보를 이용하여 가상 공간상에서의 회전(자세)값을 정한다. 위치 정보와 마찬가지로 측정한 맨 처음의 값을 가상 공간상에서 맨 처음에 xyz축 기준 (0,0,0)만큼 회전한다. 현재 프레임의 회전 값은 "직전 프레임의 위치값 - 현재 프레임의 위치값"으로 구한다. 이를 표현한 식이 다음의 (식 5)이다.

$$R_{f(xyz)}=R_{curxrot}=R_{curyrot}=R_{curzrot}=0 \quad \text{(식 5)}$$
$$R_{curxrot}=atan2(\sqrt{(P_{oldx}-P_{curx})^2+(P_{oldz}-P_{curz})^2},\ P_{cury}-P_{oldy})$$
$$R_{curyrot}=atan2(P_{curx}-P_{oldx},\ P_{curz}-P_{oldz})$$

$R_{f(xyz)}=R_{curxrot}=R_{curyrot}=R_{curzrot}=0$: 첫 프레임에서의 회전 정보, 첫 프레임에서는 0으로 초기화하여 가상공간상에서 Z축+ 방향을 바라보게 한다.

$R_{curxrot}$: 현재 프레임의 x축 회전 값
$R_{curyrot}$: 현재 프레임의 y축 회전 값
$P_{oldx}, P_{oldy}, P_{oldz}$: 이전 프레임에서 센서의 x,y,z 위치 값
$P_{curx}, P_{cury}, P_{curz}$: 현재 프레임에서 센서의 x,y,z 위치 값

3.3 지형 데이터와 센서 정보의 통합계산

로드된 지형 데이터는 이제 높이정보와 색상정보를 통합한다. 2차원의 전체 임시 배열이 존재한다. 지형데이터의 처음부터 끝까지 탐색하며 높이정보가 존재하는 부분을 찾고 존재한 부분만 임시배열에 저장한다. 이 지형 정보는 센서 기준으로 배열 전체 범위 내에 저장된 것이기 때문에 가상월드에 그대로 적용시키는 경우, 가상 월드의 규모 내에서만 변화가 일어난다.

지형 데이터에 센서 정보를 통합하는 과정은 지형 데이터를 임시 배열에 저장하는 도중에 진행된다. 배열을 순환하며 매번 몇 번째의 배열에 탐지된 지형이 있다면 그 지형 데이터에 센서의 위치값을 더하는 것이다. [그림 3]은 한 프레임에 대한 지형 데이터를 계산하는 의사코드이다.

지형 데이터의 통합 계산과 2차원 배열 구성이 끝나면 가상공간에서 눈에 보이기 위한 3차원 메쉬로 표현하기 위해 지형구조에 할당하기 위한 작업이 필요하다.

```
for(int 세로인덱스 = 0; 세로인덱스 < 지형맵세로크기;세로인덱스++)
{
    for(int 가로인덱스 = 0; 가로인덱스 < 지형맵가로크기;가로인덱스++)
    {
        int 지형인덱스 = 세로인덱스 * 지형맵가로크기 + 가로인덱스;
        if( 지형데이터[지형인덱스] == 유효 )
        {
            지형형태정보 = 지형데이터[지형인덱스].형태정보 + 지형데이터[지형인덱스].계산된센서정보;
            지형색상정보 = 지형데이터[지형인덱스].색상정보;

            현재프레임의지형정보 = 지형형태정보 + 지형색상정보;
        }
    }
}
지형관리자.업데이트(현재프레임의지형정보);
```

[그림 3] 지형데이터를 통합계산하는 의사코드

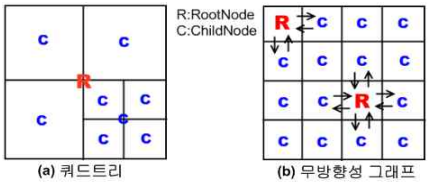
4. 지형 데이터의 동적 재구축

실시간으로 변화하는 지형 데이터에 대해 데이터를 로드하고, 계산하고, 렌더링하기 위해서는 안정적인 지형 관리 구조가 필요하다. 이 장에서는 지형 관리에 적합한 자료 구조를 선정하고, 이를 토대로 지형 데이터의 갱신, 오류 보정 과정 등을 설명한다.

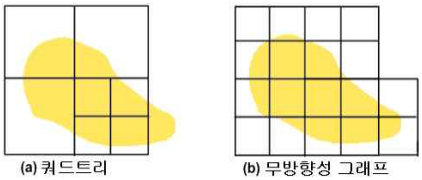
4.1 무방향성 그래프를 이용한 지형 관리

본 논문에서 쓰이는 지형 데이터는 그 크기가 실시간으로 변화하기 때문에 지형의 최대 크기를 알아야 하는 쿼드트리 조건에는 맞지가 않다. 만약 임의로 최대 크기를 정하더라도 실시간 업데이트 환경에서는 지형 데이터의 높이 정보를 매 프레임마다 갱신해야 하기 때문에, 갱신되는 지형 정보의 모든 버텍스에 대한 탐색을 수행해야 하는 부하가 발생한다[12]. 본 논문에서는 실시간으로 갱신하는 부하를 줄이고, 지형의 최대 크기를 알 수 없을 때 사용이 가능한 무방향성 그래프를 이용하여 지형 관리를 설계한다.

무방향성 그래프는 탐색 시작 노드에 제한을 받지 않고 노드간 이동도 자유자재로 가능하다. [그림 4]는 무방향 그래프의 이동 자유도가 쿼드트리와 다름을 표현한 것이다. 지형 관리를 위해 실시간에 지속적으로 지형 데이터 갱신이 이루어지는 환경에서, [그림 5]의 (a)와 같은 쿼드트리는 관리하는 범위 외로 지형 데이터를 갱신할 때 쿼드트리를 재구성하거나 초기에 쿼드트리의 크기를 크게 설정해야 한다. 하지만 [그림 5]의 (b)와 같은 무방향성 그래프를 사용한 지형 구조는 관리범위를 벗어나면 그 부분에 맞는 노드를 추가하면 된다.



[그림 4] 쿼드트리와 무방향성 그래프의 이동 자유도의 차이



[그림 5] 쿼드트리와 무방향성 그래프의 범위를 벗어나는 경우의 지형관리

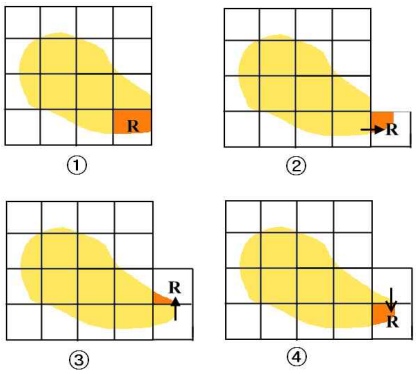
그래프 구조의 노드가 가상 공간상에서 렌더링 되는 최소 단위이다. 각 노드는 사각 형태의 격자 모양으로 높이맵 형태의 메쉬로 구성된다. 사각 메쉬 형태의 노드를 쓰는 이유는 메쉬 구성이 용이하기 때문이다. 사각 형태의 노드를 쓸 경우, 각 노드의 가로, 세로 길이만 알면 높이 값은 지형 데이터를 입력하여 사용하므로 별도로 계산할 필요가 없다.

무방향성 그래프를 사용한 지형 관리 구조에서 각 노드는 그려지는 삼각형 개수, 이웃 노드 포인터, 노드의 탐색 여부 변수 등으로 구성되어 있다. 이외 렌더링을 위한 정점 버퍼와 인덱스 버퍼가 포함된다. 가상공간상에서 어느 부분에 위치하는지 알기 위한 노드의 중앙 x,z값이 있고, 지형 데이터의 노드 포함 여부를 알기 위한 노드의 최소, 최대 x,z값이 있다. 중앙, 노드 최소, 최대 x,z값들은 전부 노드의 로컬 좌표 기준이 아니라 월드 좌표 기준이다. 이웃 노드 포인터는 STL(Standard Template Library)을 사용하여 구현하며, 그 수에 제한은 없다.

4.2 지형 데이터의 갱신

갱신 작업을 시작할 때 가상공간의 원점 (0,0,0) 좌표를 기준으로 하는 노드를 하나 생성한다. 그 후에 매 프레임이 진행될 때마다 지형 데이터 동적 할당 단계에서 마지막으로 갱신된 노드를 기준으로 지형 데이터 렌더링을 위한 지형 메쉬를 갱신한다. 맨 첫 프레임에서 한 프레임 단위의 모든 지형 데이터가 한 노드 안에 포함된다면 새로운 노드의 생성은 이루어지지 않고, 지형 메쉬를 갱신하기 위한 임시 정점 큐에 지형 데이터를 저장한다.

모든 지형 데이터에 대해서 다음과 같은 과정이 동일하게 수행되고 갱신된다. 만약 노드의 범위(최소, 최대)를 벗어난다면, 벗어난 지형 데이터를 임시 큐에 저장하고 어느 방향으로 벗어났는지를 검사한다. 한 노드에 포함되는지에 대한 범위 검사가 끝나고 나면 임시 정점 큐에 있는 나머지 지형 데이터를 하나씩 꺼내서 노드의 메쉬를 갱신한다. 노드의 갱신 작업이 완료되면, 벗어난 지형 데이터에 대해 노드를 추가하고 갱신을 수행한다.



[그림 6] 지형 데이터가 마지막으로 갱신된 노드의 포인터가 이동되는 과정

각 노드는 그래프 노드 관리자에 의해 관리된다. 그래프 노드 관리자는 너비탐색함수와, 최초로 생성된 노드 포인터, 마지막으로 갱신이 수행된 노드의 포인터 lastNode 등의 요소로 운영된다. 매

프레임마다 갱신이 이루어질 경우 [그림 6]과 같이 지형 데이터의 동적 할당 단계에서 노드에 대해 벗어났는지 검사하는 것이 필요하다. 이 때 마지막 노드 포인터 lastNode를 사용함으로써 모든 노드를 검사하지 않고 바로 그 마지막 노드만 검사함으로써 연산량을 줄일 수 있다.

4.3 오류 보정

실제 지형을 센서에 가깝게 촬영하고 좋은 환경에서 데이터를 만들더라도 지형 데이터에는 오류 값이 존재할 수 있다. 이 절에서는 본 연구에서 확인한 회전 정보의 오류와 지형 색상 정보의 오류 등의 보정 방법을 설명한다.

4.3.1 회전 정보의 오류 보정

회전 정보의 오류 $\Delta\Theta_b$는 실제 회전 라디안 값 Θ_r과 탐지된 라디안 값Θ_s 차이값 범위내에 존재한다. 이 회전 라디안 값이 (식 6)으로 표현될 때 자동차의 전방과 같은 방향이 되어야 함을 고려해야 한다. 만약 속도가 낮다면 자동차의 회전 각도 θd는 신뢰성이 없지만, $(|\Delta y| \gg \epsilon, |\Delta x| \gg \epsilon)$와 다른 $\theta s(t)$와 $\theta s(t-1)$사이의 $\Delta\theta s$가 주어지면 (식 7)로 측정된 값은 믿을만하다.

$$\theta d = \arctan((\Delta y + \epsilon y)/(\Delta x + \epsilon x)) \quad \text{(식 6)}$$

$$\Delta\theta s = \theta s(t) - \theta s(t-1) \quad \text{(식 7)}$$

회전 정보의 오류 보정은 자동차가 정지, 저속, 고속으로 회전할 때에 따라 각각 (식 8,9,10) 등의 오류보정을 기준으로 센서의 자세정보를 보정한다.

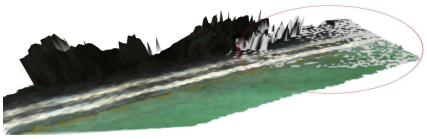
$$\theta_r = \theta_s + \Delta\theta_b \quad \text{(식 8)}$$

$$\begin{aligned} \theta_r &= \theta_d \quad \text{if} |d\Delta\theta_d/dt - \epsilon| < d\Delta\theta_s/dt \\ \theta_s &= \theta_r(t-1) + \Delta\theta_s \ \text{if} |d\Delta\theta_d/dt - \epsilon| > d\Delta\theta_s/dt \end{aligned} \quad \text{(식 9)}$$

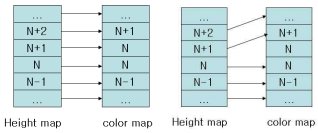
$$\theta_r = \theta_d \quad \text{(식 10)}$$

4.3.2 색상정보와 지형형태 정보의 오류 보정

높이맵과 컬러맵의 프레임 매칭 에러 때문에 일어나는 현상의 예를 [그림 7]의 빨간 원부분에서 볼 수 있다. 이러한 현상을 유발하는 오류는 3가지가 있을 수 있다. 하나는 [그림 8]처럼 각 프레임 간의 색상 정보가 지형 형태 정보보다 먼저 저장되거나 지형 형태 정보가 먼저 저장되는 경우이다. 둘째는 [그림 9]에서와 같이 프레임 사이에 아예 저장이 되지 않아서 정보가 없는 경우이다. 셋째는, [그림 10]처럼 저장된 데이터에 잘못된 정보가 저장되는 경우 등이다. 이들 모두 문제를 유발한 높이 맵이나 컬러 맵의 프레임을 무시하여 해결한다.

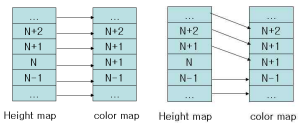


[그림 7] 색상 정보와 지형 형태 정보의 프레임 매칭 에러



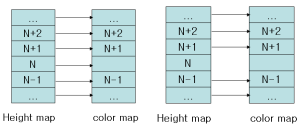
(a) 오류1 (b) 해법1

[그림 8] 데이터가 먼저 저장되는 오류와 해법



(a) 오류2 (b) 해법2

[그림 9] 데이터가 저장되지 않는 오류와 해법



(a) 오류3 (b) 해법3

[그림 10] 저장 데이터가 잘못된 오류와 해법

5. 실험 및 분석

이 장에서는 지금까지 설명한 지형 재구축 시스템을 적용하여 실측된 지형 데이터로부터 가상 공간의 3D 지형을 생성하는 실험을 진행한다.

제안한 시스템의 성능을 검증하기 위하여, 현재 지형 데이터의 효과적인 관리를 위해 가장 보편적으로 사용하는 쿼드트리(Quad Tree)와의 성능을 비교한다. 실험을 통해 제안 시스템의 각 단계가 바르게 수행되고 이 과정에서 성능 개선과 지형 가시화 수준을 검토한다.

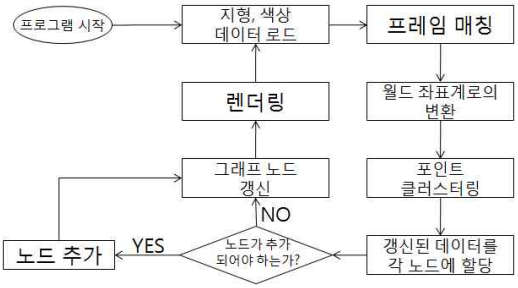
실험은 쿼드코어 2.40GHz, 3.2G램, NVIDIA Geforce 9800GTX+ 1GB의 PC 사양에서 마이크로소프트 社의 Visual Studio 2005와 DirectX SDK 9.0c (Aug. 2007) 버전을 사용하여 진행하였다.

5.1 실험 방법

그리드 기반의 높이맵(Height Map)과 컬러맵으로 실시간 갱신되는 지형 데이터를 관리하기 위한 성능 분석을 위하여 무방향 그래프와 쿼드트리의 성능을 비교한다. 쿼드트리를 적용한 시스템과 무방향 그래프를 적용한 시스템을 구현하여 동일한 데이터로 FPS를 측정하였다. 두 시스템 모두 지형 데이터의 관리를 제외한 처리 방법은 동일하게 진행하였다.

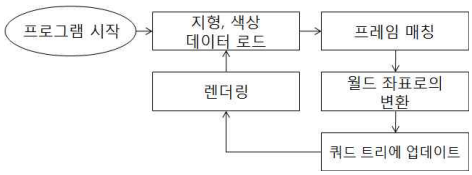
본 연구에서 제안한 3D 지형 재구축 방안을 토대로 시스템을 구현하였다. [그림 11]은 [그림 2]에 표현한 흐름도를 무방향 그래프를 적용하여 상세화시킨 것이다. 무방향 그래프는 필요에 따라 동적으로 노드를 할당한다.

실험 데이터는 [그림 1]과 같이 차량을 이용하여 스테레오 카메라를 통해 획득한 실제 영상과 스캔한 지형 데이터를 사용해 생성된 그리드 기반의 높이 맵(Height Map)을 사용한다. [그림 11]에서 지형, 색상 데이터 로드 단계에서 1개의 프레임을 읽어와 나머지 과정을 반복 수행한다. 하나의 프레임은 높이 값만을 저장하고 있는 하나의 높이 맵과 높이 맵에 대응되는 컬러 맵으로 구성되어 있다.



[그림 11] 무방향 그래프를 사용한 3D 지형 재구축 시스템의 상세 흐름도

무방향 그래프의 성능을 비교하기 위하여 본 연구에서는 쿼드트리를 사용한 지형 관리 시스템을 별도로 구현하였다. [그림 12]에서 적용된 쿼드트리를 사용한 시스템은 [그림 11]에서 사용한 시스템과 동일한 순서로 진행되는 것을 확인할 수 있다. [그림 11]의 포인트 클러스터링 단계부터 그래프 노드 갱신까지의 처리 단계를 쿼드트리로 대체한 형태를 취하고 있다.



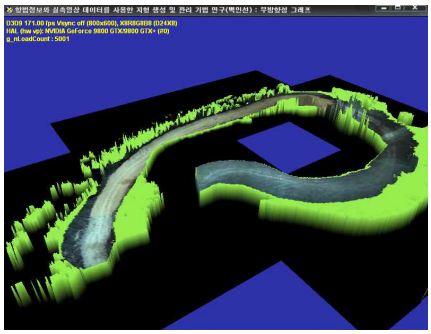
[그림 12] 쿼드 트리를 사용한 3D지형 재구축 흐름도

쿼드트리는 지형의 크기가 (2^n+1)이어야 하기 때문에, 먼저 3D 지형 재구축 시스템을 통한 실험을 먼저 진행하여 5,000개의 프레임을 진행한 후 생성된 실험 지형의 최대, 최소 크기를 알아내어 쿼드트리의 전체 크기를 추정하였다. 그래서, 그 크기를 1025x1025의 크기로 설정하였다. 지형 메쉬를 동적으로 생성하지는 않았고, 1025x1025에 맞는 크기의 지형 메쉬를 갱신하였다.

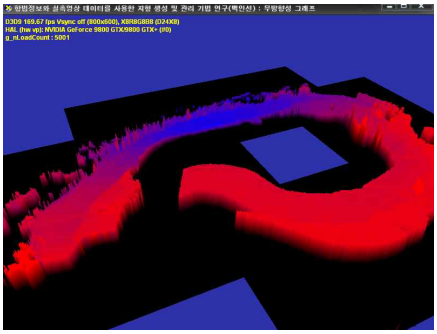
실험은 지형 데이터를 총 5,000개의 프레임을 채집하였으며, 프레임을 진행함에 따라 데이터를 로드하고 계산하고 지형 메쉬를 생성, 갱신하는 작업과 동시에 지형 메쉬의 렌더링을 수행하였다. 그리고 프레임이 진행될 동안 500 프레임마다 FPS를 저장하여 5,000 프레임까지 총 10개의 FPS 값 중 최저 FPS와 최고 FPS를 구하고 평균 FPS를 계산하였다.

5.2 실험 결과 및 분석

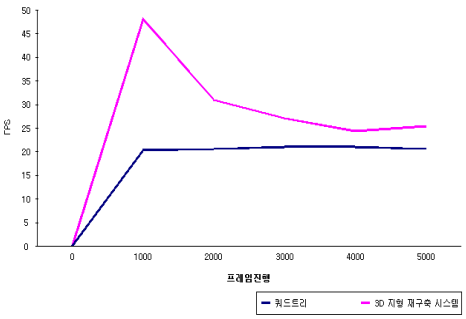
실험은 본 연구에서 구현한 3D 지형 재구축 시스템과 쿼드트리를 이용가한 지형 재구축 방식을 수행하여 데이터 로드가 끝날 때까지의 FPS(Frame Per Second)를 측정한다. FPS는 최고 수치를 기록한 FPS와 전체 프레임을 진행함에 따른 평균 FPS를 측정한다. 측정한 FPS 결과를 그래프로 나타낸 것이 [그림 15]이다. [그림 13]은 지형 데이터의 형태정보와 색상정보를 통합하여 하나의 지형데이터로 만들고 제안 시스템으로 완성하여 가시화한 지형의 모습이다. [그림 14]는 지형데이터의 형태정보와 색상정보를 통합하는 과정에서, 지형의 높이를 특정 기준에 따라 색상을 정하여 지형 데이터를 만들고 제안 시스템으로 완성하여 가시화한 지형의 모습이다.



[그림 13] 스테레오 이미지 색상을 이용한 지형의 가상 모델 재구성 결과

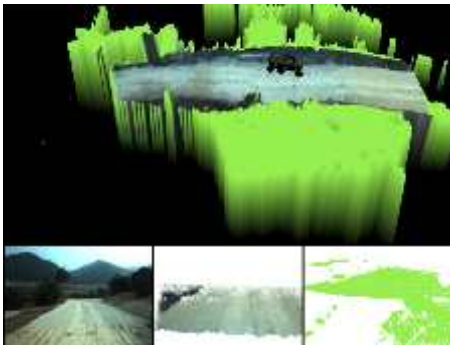


[그림 14] 높이값 색상을 이용한 지형의 가상 모델 재구성 결과



[그림 15] 쿼드트리와 3D지형 재구축 시스템의 FPS비교

[그림 16] 지형 데이터 취득 당시에 스테레오 카메라를 통해 획득한 전방 영상

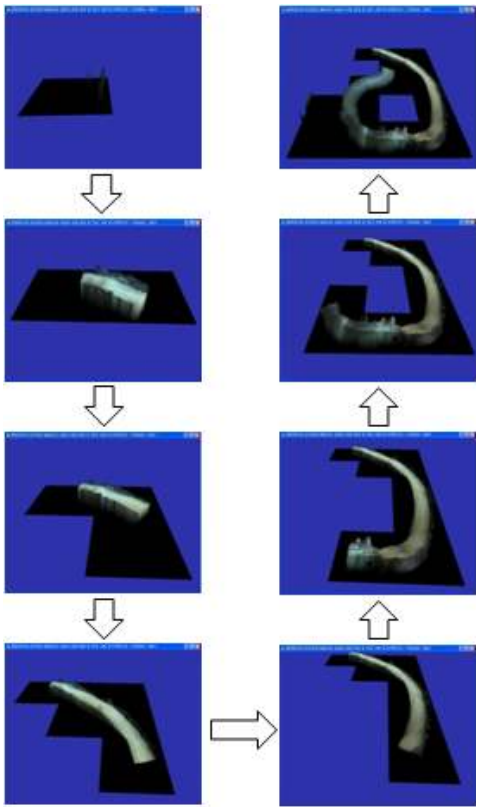


[그림 17] 전체 시스템 화면인터페이스

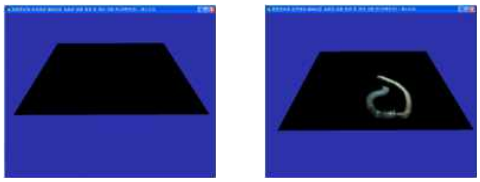
[그림 16]은 스테레오 카메라를 이용해 촬영된 전방의 이미지이다. 하나의 이미지와 그에 대응되는 높이 맵(Height Map)과 색상 정보 맵이 존재하기 때문에, 실제 지형이 정확하게 재구성되고 있는지 확인할 수 있다. [그림 17]은 [그림 16]을 전체 시스템에 적용한 그림이다. 좌측 하단에 실제 지형 데이터 획득 당시의 전방 촬영 영상을 지형 재구성과 동시에 확인할 수 있다.

[그림 15]의 그래프를 보면, 쿼드트리를 이용하는 경우와 제안 시스템의 경우, 최저 FPS는 각각 17.4와 23.2, 최고 FPS는 21.0과 78.7, 평균 FPS는 20.3과 34.1이다. 모두 쿼드트리를 사용한 지형 구조보다 높은 FPS를 가진다. 이것은 지형 데이터가 동적으로 변화하는 환경에서 제안 시스템이 동적인 환경에 더 적합하다는 것을 입증한다. [그림 15]에서 초기에 500 프레임까지는 제안 시스템이 쿼드트리에 비해 월등한 FPS를 보이지만, 시간이 지남에 따라 쿼드트리보다 약간 더 높은 수준의 FPS를 보여주었다. 쿼드트리는 처음부터 지형 메쉬를 생성하고 지형 메쉬의 갱신만을 수행하지만, 제안 시스템은 지형 데이터를 로드한 각 프레임에 따른 최소의 지형 메쉬만을 생성한다. 이것은 초기에 렌더링되는 지형 메쉬는 작지만, 프레임 누적이 되면서 지형이 더 생성되며 지속적으로 갱신하는 과정에서 렌더링되는 삼각형의 수가 늘어남에 따라 렌더링 속도가 더 느려짐에 따른 것이다.

지형이 클수록 지형 메쉬가 늘어날 가능성이 크기 때문에 매우 큰 지형이 렌더링되는 경우는, 지형이 커짐에 따라 공간 소요 양이 필요한 만큼 증가하면서 사용하게 된다. 이는 초기부터 지형 전체 크기만큼의 공간을 요구하는 쿼드트리를 사용하는 경우보다 공간 성능이 개선되는 것이다. 또한 실제 공간 크기 정보를 미리 알아야만 하는 쿼드트리의 단점을 개선한 방법이기도 하다. [그림 18]과 [그림 19]에서 검은색 사각형은 메모리에 할당되어 있는 공간을 의미한다. [그림 18]에서 보여지듯이, 무방향 그래프는 지형 데이터가 업데이트 될 때에 필요한 노드를 할당하지만, [그림 19]의 쿼드트리를 적용한 시스템은 좌측의 최초 할당된 공간과, 우측에서 보여지는 어느정도의 지형을 생성한 시점의 할당량이 같은 것을 확인할 수 있다.



[그림 18] 무방향 그래프를 적용한 시스템의 진행 과정



[그림 19] 쿼드트리를 적용한 시스템

쿼드트리를 사용한 지형은 지형 데이터 로드 초기에서 5,000 프레임까지의 FPS가 일정한 수준의 FPS를 보인다. 이것에서, 쿼드트리가 만들어진 지형에 대해서는 렌더링 속도를 유지하는 안정적인 구조라는 것을 알 수 있다.

6. 결론 및 향후 연구

본 논문에서는 스테레오 카메라를 통해 실측된 영상의 지형 데이터로부터 가상공간의 지형을 생성하는 3D 지형 재구축 프레임워크를 제안하였다. 제안 시스템은 실측영상으로부터 저장한 지형 데이터 중 센서데이터를 로드하여 가상공간의 원점을 기점으로 각 프레임별로 자세정보를 계산하고, 1차원으로 수집되는 지형 데이터를 2차원 형식으로 변환하여 높이맵 형식의 지형 메쉬를 생성하면서 센서데이터와 통합하였다. 올바른 지형 데이터를 정하여 지형 메쉬를 생성하기 위해 클러스터링 과정을 통해 비슷한 위치의 지형 데이터 중 올바른 값을 판별하였다. 또한 렌더링되는 지형 메쉬의 삼각형 개수를 줄이기 위하여 무방향성 그래프를 사용하여 노드 단위로 지형을 관리하는 방안을 설명하였다.

본 논문은 실시간으로 센싱된 지형 데이터를 가상 세계의 3D 지형으로 실시간으로 재구축하는 방법을 제안하였다. 본 연구에서 제안한 시스템을 이용하여 게임에 이용할 지형 데이터를 실시간으로 자동 생성하여 게임에 이용하거나, 실제 지형을 배경으로 필요한 영화의 CG 작업에 활용하는 등의 응용 방안을 고려해 볼 수 있다.

참고문헌

[1] A. Griesert, “Interactive Realtime Terrain Visualization for Virtual Set Applications.”, International Conference on Human and Computer, Aizu-Wakamatsu/Tokyo, August, 2003, pp. 55-61.

[2] C. Dai and X. Deng, “Fast rendering of massive textured terrain data”, Journal of Communication and Computer, Mar. 2007, Vol.4, No.3, ISSN 1548-7709, USA, pp. 63-68

[3] R. Sukumar, “Multi-sensor Integration for Unmanned Terrain Modeling.”, in Proc. SPIE Unmanned Systems Technology VIII, Vol. 6230, Orlando, FL, April 2006, pp. 65-74.

[4] A. McKenzie, “Terrain Geometry from Monocular Image Sequences.”, Journal of Computing Science and Engineering, March 2008.

[5] F. D. Luna저, 최현호 역 “DirectX9를 이용한 3D Game 프로그래밍 입문”, 정보문화사, 2004.

[6] M. Deloura외 공저, 류광 역, “Game Programming Gems”, 정보문화사, 2001.

[7] S. Ranta, “Binary Space Partitioning Trees and Polygon Removal in Real Time 3D Rendering”, Uppsala Master's Theses in Computing Science Examensarbete 179, Jan. 2001.

[8] R. Pajarola, “Overview of Quadtree-based Terrain Triangulation and Visualization”, UCI-ICS Technical Report No. 02-01 Dept. of Information & Computer Science, Univ. of California, Irvine, Jan. 2002.

[9] 김용준 저, “IT EXPERT 3D 게임 프로그래밍”, 한빛미디어, 2003.

[10] M. Duchaineau, “ROAMing Terrain (Real-time Optimally Adapting Meshes)”, Institute of Electrical and Electronics Engineers Visualization '97 Phoenix, AZ, October 1997, pp 19-23.

[11] Eric Lengyel, Mathmatics for 3D game Programming & Computer Graphics, Charles River Media, 2004

[12] Thomas H. Cormen, “Introduction to Algorithms”, 한빛 미디어, 2009

백 인 선 (Insun Baek)

2007.2 한국IT전문학교 게임프로그래밍 학사
2009.2 동국대학교 영상대학원 멀티미디어학과 공학석사

관심분야 : 지형엔진, 게임프로그래밍

엄 기 현 (Kyhyun Um)

1975.2 서울대학교 공과대학 응용수학과 공학사
1977.2 한국과학기술원 전산학과 석사
1994.2 서울대학교 대학원 컴퓨터공학과 공학박사
1978.3-2007.6 동국대학교 컴퓨터멀티미디어공학과 교수
2007.7-현재 동국대학교 영상미디어대학 멀티미디어 공학과 교수
1995.3-1999.2 동국대학교 정보관리처장 역임
2001.3-2003.2 동국대학교 정보산업대학 학장 역임
2009.9-현재 동국대학교 영상미디어대학 학장 겸 영상대학원 원장
2005.3-현재 한국게임학회 자문위원
1998.12-현재 한국멀티미디어학회 부회장, 자문위원, 수석부회장, 학회장 역임

관심분야 : 게임시스템 및 구조 설계, 멀티미디어응용 시스템, 멀티미디어데이터베이스

조 경 은 (Kyungeun Cho)

1993.2 동국대학교, 전자계산학과 공학사
1995.2 동국대학교, 컴퓨터공학과 대학원 공학석사
2001.8 동국대학교, 컴퓨터공학과 대학원 공학박사
2003.9-2005.8 동국대학교 정보산업대학 컴퓨터멀티미디어공학과 전임강사
2005.9-현재 동국대학교 영상미디어대학 멀티미디어 공학과 부교수
2003-현재 한국멀티미디어학회 이사
2003-현재 한국게임학회 부회장

관심분야 : 컴퓨터 게임 알고리즘, 게임 인공지능, 멀티미디어 정보처리