제18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호 (2002.11)

# 절차적인 지형 모델링과 텍스쳐링

강호철*, 한정현*
*성균관대학교 정보 통신 공학부
e-mail : kang5725@hanmail.net

# Procedural Terrain Modeling and Texturing

Ho-Chul Kang*, JungHyun Han*
*Information and Communications Engineering
Sung Kyun Kwan University

요 약

본 논문에서는 절차적인(procedural) 방법으로 지형을 모델링하고 텍스쳐링 하는 기법을 제안하고 구현하였다. 절차적인 기법에서 널리 쓰이는 노이즈 함수(noise function)를 이용하여 지형 데이터를 생성한다. 동시에 지형의 기본적인 텍스쳐 자료를 입력받아 역시 노이즈 함수를 이용하여 텍스쳐를 생성한다. 이렇게 절차적인 기법으로 모델링과 텍스쳐링을 수행할 경우, 데이터를 파일이나 데이터베이스에 따로 저장할 필요가 없으므로, 거대한 또는 무한한 크기의 지형을 손쉽게 생성할 수 있다. 이 기법은 야외 지형을 필요로 하는 3D 온라인 게임(Online Game)에 응용될 수 있다. 또한, 인터넷상에서 가상 공간을 구현할 때 지형 데이터를 서버로부터 다운 받을 필요가 없기 때문에 무한한 야외의 공간을 쉽게 구현할 수 있다.

## 1. 서론

기존의 지형 렌더링 방식은 비트맵이나 DEM (Digital Elevation Model) 파일 등에서 지형의 높이 값을 얻어와 렌더링 하는 방식이었다. 비트맵 파일을 예로 들면, 각 격자의 높이 값을 그레이 레벨의 이미지로 저장하여 지형의 데이터를 표현한다. 하지만 이러한 하이트 맵(height map)은 그 데이터가 한정되어 있기 때문에 무한한 지형을 표현할 수 없고, 무한하지는 않더라 할지라도 생성하고자 하는 지형의 크기가 클 때, 그 크기가 커질수록 지형의 높이 데이터를 저장하는 파일의 크기도 커지기 때문에 데이터를 저장하는 공간도 커지고 이들 데이터를 읽어들이는 시간도 오래 걸리게 된다. 이러한 문제점을 해결하기 위해 본 논문에서는 절차적인 방법으로 지형의 높이 데이터를 생성하는 방법에 대해서 논의할 것이다.

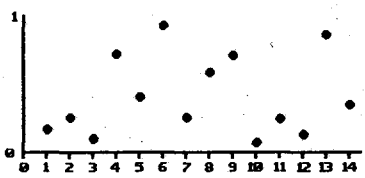
3D 장면을 구성하는 지형이나 오브젝트들을 렌더링할 때 이들에게 현실감을 부여하기 위한 작업이 바로 텍스쳐 맵핑 작업이다. 지금까지 사용된 텍스쳐 맵핑 방식은 물체의 실제 사진을 스캔한 이미지를 사용하거나 디자이너들이 그래픽 툴로 그린 이미지를 사용하여 맵핑하게 된다. 이를 절차적인 텍스쳐 맵핑과 구별하여 이미지 텍스쳐 맵핑 방식이라고 하는데, 지형을 맵핑할 때도 이런 이미지를 이용하여 맵핑을 하게 된다. 하지만 이러한 이미지도 하이트 맵과 마찬가지로 유한한 데이터를 가지고 있기 때문에 넓은 지형을 맵핑할 때는 문제가 생긴다. 이러한 문제를 해결하기 위해 사용되는 방법 중에 하나로 타일을 이용하기도 하는데, 이 방식은 작은 텍스쳐 이미지를 지형 전체에 반복해서 맵핑해주는 방법이다. 타일을 이용하면 무한한 지형도 맵핑이 가능하긴 하지만 똑같은 이미지를 계속 반복해서 맵핑하기 때문에 단편적인 지형이 생성되게 된다. 본 논문에서는 단편적인 지형의 생성을 막기 위해 지형의 절차적인 텍스쳐를 생성하는 방법에 대해 논의할 것이다.

## 2. 노이즈 함수

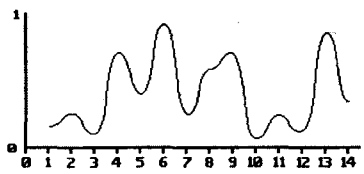
2.1 의사 난수

노이즈 함수는 절차적인 기법에서 가장 많이 사용되는 함수로 Ken Perlin [4]이 제안하였다. 노이즈 함수란 무작위의 수를 생성하는 것으로 어떤 수열이 입력으로 주어 졌을 때 무작위 적인 수열을 생성하는 것이다. 단, 노이즈 함수에서는 의사난수를 사용하는데 의사난수란 난수 값을 구할 때마다 값이 바뀌는 난수와는 달리 같은 파라미터가 주어졌을 때 같은 값을 생성하는 난수를 말한다.

노이즈 함수는 정수 값을 파라미터로 이용하여 의사 난수를 생성한다. 다음 그림은 노이즈 함수를 1차원에서 구현한 것이다.




(그림 1)




(그림 2)

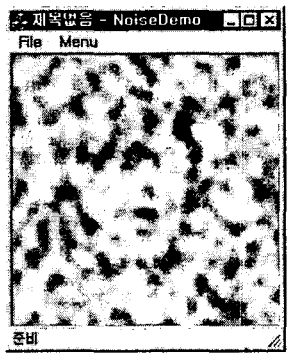
(그림 1)에서 x축은 정수 값 파라미터를 의미하고 y축은 정수 파라미터에 의해 생성된 의사 난수 값을 나타낸다. (그림 2)는 (그림 1)로부터 얻은 난수 값 사이를 보간 해준 것으로 노이즈 함수를 나타낸다.

2.2 알고리즘

다음은 노이즈 값을 구하는 노이즈 함수의 알고리즘이다.

```
function Noise(x, y)
    total = 0, p = persistence, n = Number_Of_Octaves - 1
    loop i from 0 to n
      frequency = 2^i, amplitude = p^i
      total = total + InterpolatedNoise(x * frequency, y *
              frequency) * amplitude
    end of i loop
    return total
end function
```

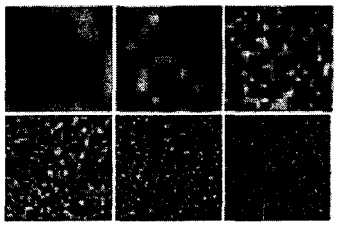
위의 persistence는 amplitude를 p'로 나타낼 때 p값을 말하고 octave는 합성할 노이즈의 개수를 말한다. 노이즈 함수의 구현 결과는 (그림 3)과 같다.




(그림 3)

## 3. 절차적인 지형 모델링

2절에서 설명한 노이즈 함수를 이용하여 여러 가지 노이즈 값을 구할 수 있다. 파라미터들을 변형하여 (그림 4)와 같은 노이즈 값들을 구할 수 있다.



(그림 4)

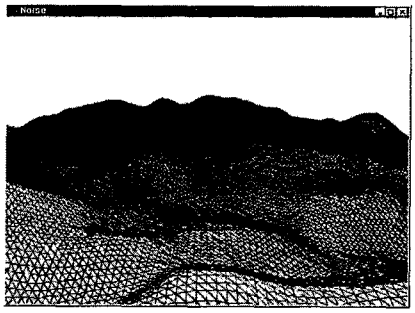
(그림 4)의 노이즈 값들을 모두 합성하면 좀더 부드러운 노이즈 값을 구할 수 있다. (그림 5)가 합성한 노이즈의 결과이다.


(그림 5)

본 문에서 제안하는 기법에서는 이를 하이트 맵으로 인식하고 지형의 높이 데이터를 얻어 렌더링을 한다. 단, 기존의 하이트 맵과는 달리 하이트 값을 무한하게 생성할 수 있기 때문에 무한한 지형 렌더링이 가능하게 된다. 현재 카메라의 위치에 따라 지형을 실시간으로 계속 생성하게 된다. 이를 의사코드로 나타내면 다음과 같다.

```
function CreateTerrainData(Camera c)
  integer ii = jj = 0
  loop i from c.x-(width/2) to c.x+(width/2)
    loop j from c.y-(height/2) to c.y+(height/2)
     height[ii][jj] = Noise(i, j)
     jj++
    end of j loop
    ii++
  end of j loop
end function
```

width와 height는 생성할 지형의 크기를 나타내고, 코드 안에서 임의의 상수로 지정해 놓는다. 지형의 높이 데이터를 구하기 위해 카메라의 위치 값을 파라미터로 받게 된다. 카메라의 위치를 기준으로 전, 후, 좌, 우로 width/2 만큼 지형 데이터를 생성하게 되고 카메라의 위치가 변할 때마다 실시간으로 지형의 데이터를 구하여 렌더링 한다. (그림 6)은 실제로 구현한 지형의 와이어 프레임을 나타낸다.




(그림 6)

## 4. 절차적인 지형 텍스처링

노이즈 함수를 이용하여 지형의 모델링을 한 후, 지형에 텍스쳐 맵핑을 해야 하는데 텍스쳐 역시 노이즈 함수를 이용하여 생성한다. 본 논문에서 지형의 텍스쳐를 생성하는 방법은 다음과 같다.

· 서로 보간하지 않은 의사 난수의 값과 풀의 색깔, 흙의 색깔을 합성하여 텍스쳐의 데이터를 생성한다.

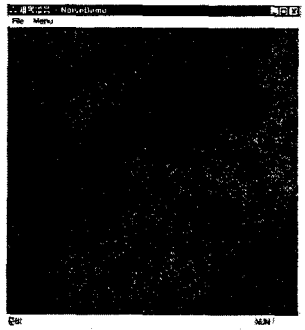
· 노이즈 함수로 생성된 난수의 값과 풀의 색깔, 흙의 색깔을 합성하여 텍스쳐의 데이터를 생성한다.

· 위의 과정에서 얻은 두 개의 데이터에 가중치를 주어 합성하여 최종 텍스쳐를 생성한다.

첫 번째 과정을 의사코드로 나타내면 다음과 같다.

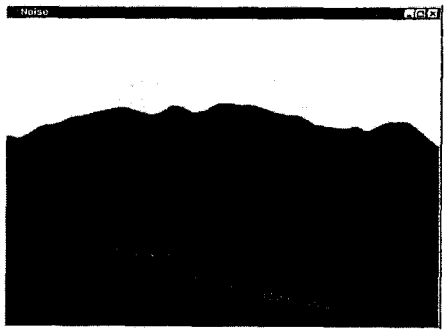
```
loop i from 0 to texture_width
  loop j from 0 to texture_height
    float f = Random(i, j)
    if f > threshold then texture_1[i][j] = f * GREEN
    else texture_1[i][j] = f * BROWN
  end of j loop
end of j loop
```

texture_width와 texture_height는 각각 생성할 텍스쳐의 가로 길이와 세로 길이를 나타내고 threshold는 높이의 기준 값을 주어 그 값보다 크면 풀의 색깔을 할당하고, 작으면 흙의 색깔을 할당하게 하였다. 두 번째 과정은 첫 번째 과정과 거의 유사한데, 랜덤 함수 대신 노이즈 함수를 쓰고 각 색깔의 가중치를 다르게 한다. 두 개의 텍스쳐 데이터를 합성하면 (그림 7)과 같다.



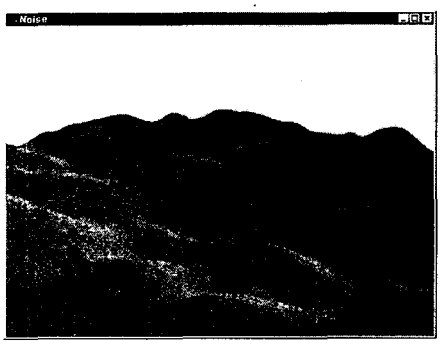
(그림 7)

최종적으로 텍스쳐를 맵핑하면 (그림 8)과 같다.

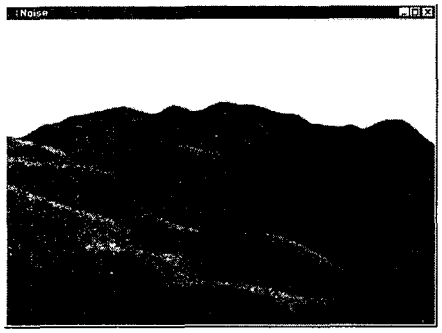



(그림 8)

풀의 색깔과 흙의 색깔의 가중치에 변화를 주면 다음과 같은 여러 가지 지형을 생성할 수 있다. (그림 9)는 풀의 색에 가중치를 많이 준 경우이고, (그림 10)은 흙의 색깔에 가중치를 많이 준 경우이다.




(그림 9)




(그림 10)

## 5. 결론

절차적인 방법으로 지형을 렌더링하면 지형의 높이 데이터와 텍스쳐 데이터를 파일이나 데이터베이스로부터 얻을 필요 없이 소스 코드로부터 지형의 관련된 데이터를 얻어 무한한 지형을 렌더링 할 수 있다. 이는 야외를 배경으로 하는 3D 게임이나 가상 공간에서 효과적으로 이용될 수 있을 것이다.

절차적인 방법의 특성상 지형의 데이터를 시스템에 저장해 놓을 필요가 없지만 항상 그 데이터들을 얻기 위해 소스 코드를 실행하여 처리해야 한다. 그러므로 렌더링의 시간이 오래 걸린다는 단점이 있다. 하지만 하드웨어의 성능은 시간이 지남에 따라 계속 발전하고 있기 때문에 머지 않아 절차적인 방법을 이용하여 실시간으로 렌더링할 수 있을 거라 예상된다.

## 참고문헌

[1] Perlin, K., "In the beginning : The Pixel Stream Editor," *SIGGRAPH '01 Real-Time Shading course notes,* pp.2.1-2.10, August 2001.

[2] Perlin, K., "Noise Hardware," *SIGGRAPH '01 Real-Time Shading course notes,* pp.9.1-9.23, August 2001.

[3] Erik Lindholm, Mark J Kilgard, and Henry Moreton, "A User-Programmable Vertex Engine," *Computer Graphics (SIGGRAPH '01 Proceedings),* pp. 149-158, August 2001.

[4] Kekoa Proudfoot, William R. Mark, Svetoslav Tzvetkov, and Pat Hanrahan, "A Real-Time Procedural Shading System for Programmable Graphics Hardware," *Computer Graphics (SIGGRAPH '01 Proceedings),* pp. 159-170, August 2001.

[5] Moller, T., and Haines, E., *Real-Time Rendering,* A K Peters, 1999.

[6] Ebert, David S., F. Kenton Musgrave, Darwyn Peachey, Perlin, K., and Steven Worley, *Texturing & Modeling: a Procedural Approach,* second edition, AP Professional, San Diego, 1998.

[7] Perlin, K., "An Image Synthesizer," *Computer Graphics (SIGGRAPH '85 Proceedings),* vol. 19, no. 3, pp. 287-296, July 1985.