# 지형형상화를 위한 객체 클래스 설계 및 구현

## Design and Implementation of Object Classes for Terrain Simulation

노용덕*
Noh Yong Deok

**Abstract**

In 3D computer graphics, fractal techniques have been applied to terrain models. Even though fractal models are convenient way to get the data of terrain models, it is not easy to gain the final results by manipulating the data of terrain model. However, by using the object oriented programming techniques, we could reduce the effort of programming job to find the final result. In this paper, a set of classes made by object oriented programming technique is presented. To show the results, the data of a terrain model were made by a fractal technique, namely, the midpoint displacement methods with square lattices of points.

## 1. 서 론

컴퓨터 그래픽은 매우 다양한 분야에서 사용되고 있으며, 시뮬레이션 분야에서 컴퓨터 그래픽이 사용된 것은 매우 오래전의 일이다. 특히, 최근 들어 멀티미디어, 게임, 및 가상현실 등의 분야에서 사실적인 그래픽이 각광을 받고 있으며 컴퓨터 분야의 지속적인 기술의 발전으로 고가의 그래픽 워스테이션에서만 가능했던 3차원 그래픽 가속 기능들이 가격이 저렴한 PC용으로 개발, 발표되고 있다. 비록 이런 그래픽 가속기가 성능면에서는 고가의 장비들에 비해 떨어지는 것이 사실이지만, PC용으로 개발되면서 보다 깨끗하고 빠른 3차원 그래픽을 이용할 수 있는 층이 일반 사용자에게로 까지 넓혀지게 되었다.

하드웨어의 발전에 따라서 컴퓨터 그래픽스 분야에서도 3차원 물체를 보다 빠르고 효과적으로 표현하고자 하는 연구가 계속되어 왔다. 특히, 산이나 구름, 나무와 같은 비규칙적으로 보이는 자연적인 물체에 대한 모델화 작업은 Mandelbrot [4] 에 의한 프랙탈 기법이 만들어진 이후에 급속도로 발전한 전형적인 한 분야이다. 프랙탈 기법은 2차원 및 3차원 지형 형상화 작업에 적용되는 것은 물론 자연적으로 존재하는 많은 물체들을 컴퓨터상에서 재창조하고 묘사하는데 매우 효과적으로 사용되고 있다[5, 6].

지형과 같은 경우에는 구체적으로 데이터를 준비하고 이를 하나씩 그려 나가는 방식이 일반적이나, 문제는 지형을 나타내는 데이터를 준비하는 작업이 매우 많은 노력과 시간을 필요로 한다는데 있다. 이러한 것은 프랙탈 기법에 의하여 여러 가지 다양한 지형에 관한 데이터를 쉽게 만들어 낼 수가 있다. 이때 만들어

* 세종대학교 컴퓨터공학과

지는 지형은 여러 가지 주어지는 기본값에 따라 다양한 형태를 이루지만, 이러한 지형은 일반적으로 실제의 특정지형을 나타내는 것은 아니다. 그러나, 특정 지형에 대한 최소한의 데이터를 제공함으로써 해당 지형의 모습과 유사한 결과를 얻을 수 있는 데이터를 생성할 수도 있다[1].

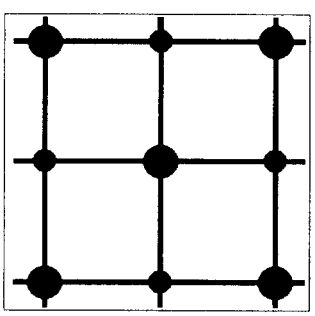
프랙탈 기법에 의하여 지형에 관계된 데이터를 쉽게 생성할 수는 있지만, 이를 그래픽으로 구현하여 우리가 원하는 지형의 형태를 만들기 위하여는 별도의 많은 프로그래밍 작업을 거쳐야 한다. Mandelbrot 이 만든 한 혹성의 모습은 그 데이터를 프랙탈 기법에 의하여 만든 것이기는 하지만, 다시금 많은 데이터 재처리 과정을 거쳐서 혹성의 그림을 완성한 것이다[13]. 또는 기존의 프로그램을 사용하기 위하여는 그에 맞는 형식의 데이터를 생성하고 해당 프로그램만을 사용하여 제한된 형식의 출력만을 기대할 수가 있다. 이 논문의 목적은 하나의 프랙탈 기법에 의하여 얻는 데이터를 객체지향방식에 의하여 생성하고 저장하고 그에 따라서 형상화에 적합한 유사 객체언어를 사용할 수 있게 함으로써, 사용자로 하여금 복잡한 프로그래밍 과정을 거치지 않고 보다 쉽게 지형을 형상화하는 객체지향방식을 구현하고자 하는데 있다. 이를 위하여 여기서는 마이크로 소프트사에서 비교적 최근에 발표한 OLE 기술을 기반으로 하는 DirectX 의 3차원 가속기 Direct3D 의 기능에 맞추어 코드화 작업을 수행하였으며, 논문의 순서는 다음과 같다. 먼저, 여기서 사용하는 격자형 프랙탈 기법을 소개하고, 이 방식에 의하여 만들고 사용하는 객체지향 언어를 사용한 프로그래밍의 기본 설계방식과 내용을 제시한다. 그리고, 이 방식에 의하여 얻어진 3 차원 지형의 결과를 보이고, 클래스 정의나 사용에 있어서의 더 고려하여야 할 분야와 한계에 대한 평가를 결론에서 보인다.

## 2. 격자형 중간점 이동 알고리즘

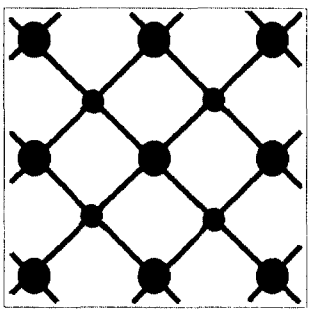
중간점 이동 알고리즘 (midpoint displacement algorithm) 은 Fournier [2,3] 에 의하여 개발된 것으로 연속분할 (subdivision) 알고리즘이라고도 한다. 이 알고리즘의 특징은 하나의 선분을 계속적으로 분할하여 나가서 결과적으로 해안선이나 산의 능선과 같은 비규칙적인 선의 형태를 만든다. 이러한 기법을 3차원상에서 그대로 적용함으로써, 하나의 3차원 지형과 유사한 데이터를 무한히 만들어 낼 수가 있는 것이다. Fournier의 알고리즘을 기본으로 여러 사람들에 의하여 중간점 이동 알고리즘으로 분류될 수 있는 알고리즘들이 만들어졌다[3, 4, 9, 10, 11, 12].

위 알고리즘의 한 방식으로 격자형 중간점 이동 알고리즘 (midpoint displacement methods with square lattices of points) 이 있다. 이 방법은 사각형 모양의 격자로 된 점을 이용하여 모든 사각형에 대해 중간점을 추가해 작은 사각형들을 계속 만들어가면서 프랙탈 지형 데이터를 생성하는 알고리즘이다. 먼저, 한 변의 길이가 $\delta$인 초기 사각형이 있고 이를 기본으로 연속적으로 중간점을 만들어가는 경우를 생각한다〈그림 1〉. 여기서, 모든 사각형에 대해 사각형내의 중간점을 삽입하면, 시작점이 원래의 사각형에 대해서 45° 회전한, 결과적으로 한 변의 길이가 $\delta/\sqrt{2}$ 인 사각형들이 새로 생기게 된다〈그림 2〉. 이때, 다시 새로 생성된 사각형들에 대해 새로운 중간점, 즉, 사각형내의 중간점을 삽입하게 되면 변의 길이가 $\delta/\sqrt{2}$이면서 원래 사각형과 동일한 형태의 사각형들이 보다 촘촘하게 배치된 상태로 생성되게 된다〈그림 3〉.

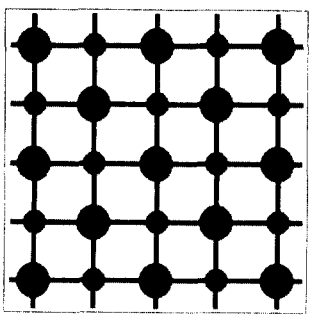
이런 과정을 반복하면서 새로운 점들을 추가해 나가게 되는데, 이 각각의 단계에서 변의 길이는 $r = 1/\sqrt{2}$ 의 비율로 줄어들게 된다. 그리고, 새로 생성된 점들에 대해서는 전 단계에서 만들어진 변의 길이 $\delta$에 $r^{2H}$를 곱한 값과 정규분포 난수 값을 곱한 결과를 높이 값으로 부여함으로써 3차원 데이터를 생성하게 한다. 결과적으로, 각 단계마다 사각형의 네 모퉁이 값에 의해 새



〈그림 1〉 초기상태의 사각형



〈그림 2〉 첫 중간점 삽입후의 사각형



〈그림 3〉 두 번째 중간점 삽입후의 사각형

로운 점이 $\sigma^2$만큼 증가하면, $n$번째 단계에서는 새로 생성된 점에 대해서는 $\sigma^2 r^{2Hn} = \sigma^2 \left|\frac{1}{2}\right|^{nH}$의 값을 높이로 부여하게 된다. 이 수식에서, H의 값은 초기 값으로 부여되는 값으로 지형의 복잡도를 나타내며, H의 값이 작아질수록 지형의 변화가 심해진다. 또한, 새로 생성된 점에 대해서만 값을 부여하지 않고, 각 단계마다 모든 점에 대해 난수 값을 더하여 가는 방식을 사용할 수도 있는데 이것은 하나의 알고리즘을 이용하여 쉽게 구현될 수 있다[13].

## 3. 지형 데이터 생성 및 저장 클래스

격자형 중간점 이동 알고리즘을 사용하면, 원하는 정도에 따라서 무한한 양의 데이터를 만들 수가 있다. 이런 데이터를 사용하여 지형을 형상화하려면, 기존의 프로그램 패키지나 개발 툴을 사용하거나, 또는 완전한 재처리 과정을 거쳐야 한다. 기존의 패키지나 개발 툴을 사용하는 경우에는, 형상화 작업에 사용되는 프로그램의 형식에 맞게 데이터를 생성하는 방식으로 프로그래밍 작업이 수행하는 것이 일반적이다. 여기서는 이러한 관점에서 MS-Windows 용 프랙탈 지형 데이터 뷰어 (이하 LandV) 를 제작하고 Direct3D 에서 사용하기 위한 3차원 지형 데이터를 생성, 이를 모니터상에서 구현하였다. LandV 에서는 지형 데이터 생성 부분의 모듈화와 코드의 재사용성을 높이기 위해서 객체지향 기법을 적용하여 C++ 클래스를 제작하였다. 이를 위해, LandV는 격자형 중간점 이동 알고리즘 구현에 관한 모든 정보와 함수를 포함하고 있는 객체 CMidPoint 클래스와 생성된 데이터를 저장하는 2차원 배열 클래스 CLandScape를 정의한다.

### 3.1 CLandScape 클래스의 설계 및 구현

CLandScape 클래스는 2차원 배열의 생성과 파괴, 그리고 지형 데이터 결과의 저장 및 참조를 용이하도록 하는 지형 데이터 저장객체이다. 이 클래스의 제작 목적은 단순하고 복잡한 작업을 자동화시키데 있다. 따라서 그 작업의 대부분은 사용자로부터 은폐되어 있고, 작업수행을 위한 일부 함수들만을 공개하고 있다 〈그림 4〉. 공개된 함수는 초기화 함수 FInit를 비롯하여 배열의 크기를 알아내는 GetSize 함수와 두 개의 연산자 Overload 함수가 있다.

CLandScape 객체를 사용하기 위해서는 초기화 작업을 거쳐야 하는데, CLandSacpe :: FInit 함수를 이용해 그 크기에 맞는 메모리를 실제로 할당할 수 있다. CLandScape는 다소 복잡한 방식으로 2차원 배열 구조를 구현하게 되는데 〈그림 4〉에서 볼 수 있듯이, CLandScape의 멤버 데이터로 C_X class를 이용한다. 이 C_X class는 외부에서는 전혀 알 수 없는 비공개 객체로 2차원 배열의 가로행 역할을 하는 객체이다. CLandScape 클래스는 이 C_X class를 포인터 배열을 이용해 할당하는 방식으로 2차원 배열을 구현하고, 이 데이터의 접근을 위해서 CLandScape는 물론 C_X class에서도 [ ]연산자를 Overload 한다. 이러한 접근 방식은 C++의 연산자 Overload에 의해서 가능하다. 또한 이 클래스의 초기화 및 사용은 모두 CMidPoint 클래스에서 이루어지고 사용자는 단지 [ ]연산자를 이용해 접근하기만 한다.

```
/* gen_tbl.h - Landscape data array class definition */
#ifndef _GENTABLE_H_
#define _GENTABLE_H_
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "type.h"

enum LS_RETURN {
        LSERR_OUTOFMEM = -1,
        LSERR_FAIL,
        LS_OK
} ;

// Horizontal Data Line
class C_X {
        float *x ;
        UINT  xSize ;
public :
        C_X () {x = NULL ;};
        ~C_X() {if (x) delete x ;};
        float & operator[] (int index) {return x[index];}
        C_X   & operator=  (C_X &obj)
                        {memcpy (x, obj.x, sizeof(float)*obj.xSize);
                         return *this;}
        LS_RETURN FInit (UINT N)
        {
                if (N <= 0) return LSERR_FAIL ;
                if ( x != NULL)       delete x ;
                if ((x = new float[N]) == NULL) return LSERR_OUTOFMEM ;
                memset(x, 0, sizeof(float)*N) ;
                xSize = N ;
                return LS_OK ;
        }
} ;

// Vertical Data Line
class CLandScape {
        C_X  *y ;
        UINT ySize ;
public :
        CLandScape (const CLandScape &obj) ;
        CLandScape () {y = NULL ;} ;
        ~CLandScape() {if (y) delete []y;}
        LS_RETURN FInit (UINT N) ;
        UINT      GetSize() const {return ySize;} ;
        C_X       & operator[] (int index) {return y[index] ;}
        CLandScape & operator=  (const CLandScape &obj)
        {       if (FInit (obj.ySize-1) == LS_OK)
                for (UINT i = 0 ; i < obj.ySize; i++) y[i] = obj.y[i] ;
                return *this;
        } ;
} ;
__inline CLandScape :: CLandScape (const CLandScape &obj)
{
        y = NULL ;
        *this = obj ;
}
__inline LS_RETURN CLandScape :: FInit (UINT N)
{
        if (N <= 0) return LSERR_FAIL ;
        N ++ ;
        if ( y != NULL)       delete []y ;
        if ((y = new C_X[N]) == NULL) return LSERR_OUTOFMEM ;
        for (UINT i = 0 ; i < N && y[i].FInit(N)==LS_OK ; i++) ;
        if (i < N) { delete []y ;
                y = NULL ;
                return LSERR_OUTOFMEM ; }
        ySize = N ;
        return LS_OK ;
}
#endif
```

〈그림 4〉 LandV의 지형 데이터 저장 2차원 배열 클래스의 정의

### 3.2 CMidPoint 클래스의 설계 및 구현

CMidPoint 클래스 설계의 기본 목표는 클래스 내부에서 생성되는 데이터와 그 생성 과정에 대한 정보를 공개하지 않음으로써 코딩 중에 발생할 수 있는 오류를 최소화하여 프로그램의 안정성을 확보하는 데 있다. 이런 구조적 특징으로 인해 CMidPoint는 지형 생성을 위한 객체 인터페이스를 제외한 모든 정보들은 비공개로 선언하고 있다〈그림 5〉.

따라서 이 클래스를 이용하여 데이터를 생성하기 위해서는 CMidPoint 클래스에서 제공하는 인터페이스를 이용하여야 한다. 이 인터페이스는 FInit, GenLandScape, GetLandData의 3가지 공개함수로 구성하였으며, 그 호출 순서를 단계별로 보면 다음과 같다. 첫 번째 단계는 FInit 함수를 호출하여 지형 생성에 필요한 초기화 작업을 하는 과정이다. 격자형 중간점 이동 알고리즘에서 사용하는 정규분포 난수 생성자의 초기값과 반복회수 n을 초기화한다. 또한 반복회수 n에 의해 결정되는 지형의 크기에 따라 데이터 저장객체 CLandScape를 초기화한다. 이 초기화에 실패할 경우에 FInit함수는 false 값을 돌려주므로, 안전한 수행을 위해서는 FInit 함수의 return 값을 확인해야 한다〈그림 6〉.

두 번째 단계에서는, FInit가 성공할 경우 GenLandScape 함수를 호출하여 데이터를 생성하는 단계이다. GenLandScape 함수는 지형 데이터를 생성하기 위해 격자형 중간점 이동 알고리즘을 구현한다. 즉, 이 함수는 지형의 복잡도를 결정하는 H와 생성을 시작하는 초기 사각형의 한 변의 길이 $\delta$, 그리고 생성의 각 단계에서 모든 점에 대해 난수값을 더할 것인가를 결정하는 Boolean 인자를 받는다. 그리고 데이터의 생성 과정 중에 사용하는 정규분포 난수발생을 위해 첫 단계에서 초기화한 seed값을 사용하며 생성된 데이터는 FInit함수에서 초기화한 CLandScape 객체에 저장된다〈그림 7〉.

```
/* midpoint.h - MidPoint Landscape generate algorithm class definition */

#ifndef _MIDPOINT_H_
#define _MIDPOINT_H_
#include <math.h>
#include "gaussian.h"
#include "type.h"
#include "gen_tbl.h"

class CMidPoint {
        CRand       gauss ;
        CLandScape X ;
        int         maxLevel ;
        int         N ;
        float       delta ;
protected :
        float     F3 (float x0, float x1, float x2) ;
        float     F4 (float x0, float x1, float x2, float x3) ;
public :
        BOOL FInit (int maxLevel, UINT seed);
        BOOL GenLandScape(float sigma, float H, BOOL addition) ;
        BOOL GetLandData (CLandScape &obj, INT scale) const;
} ;
#endif
```

〈그림 5〉 LandV의 지형 생성 클래스의 정의

```
BOOL CMidPoint :: FInit (int mL, UINT seed)
{
        if (mL <= 0) return FALSE ;
        maxLevel = mL ;
        N        = 1 ;
        while (mL--) N *= 2 ;
        if (X.FInit(N) != LS_OK) return FALSE ;
        gauss.InitGauss (seed) ;
        return TRUE ;
}
```

〈그림 6〉 FInit 함수 리스트

```
BOOL CMidPoint :: GenLandScape(float sigma, float H, BOOL addition)
{
int       stage ;
int       x, y, D = N, d = N/2 ;  // Array indexing variables
delta = sigma ;
X[0][0] = delta*gauss.Gauss() ;
X[N][0] = delta*gauss.Gauss() ;
X[0][N] = delta*gauss.Gauss() ;
X[N][N] = delta*gauss.Gauss() ;
for (stage = 0 ; stage < maxLevel ; stage++)
{
          /* going from grid type I to type II */
          delta = delta*(float)pow(0.5, 0.5*H) ;
          // interpolate and offset points
          for (x = d ; x <= N-d ; x += D)
          {
            for (y = d ; y <= N-d ; y += D)
                X[y][x]=F4(X[y+d][x+d],X[y+d][x-d],X[y-d][x+d],X[y-d][x-d]);
          }
          // displace other points also if needed
          if (addition)
          {
            for (x = 0 ; x <= N ; x += D)
                for ( y = 0 ; y <= N ; y += D)
                      X[y][x] = X[y][x] + delta*gauss.Gauss() ;
          }
          /* going from grid type II to type I */
          delta = delta*(float)pow(0.5, 0.5*H) ;
          // interpolate and offset boundary grid points
          for (x = d ; x <= N-d ; x += D)
          {
             X[0][x] = F3 (X[0][x+d], X[0][x-d], X[d][x]) ;
             X[N][x] = F3 (X[N][x+d], X[N][x-d], X[N-d][x]) ;
             X[x][0] = F3 (X[x+d][0], X[x-d][0], X[x][d]) ;
             X[x][N] = F3 (X[x+d][N], X[x-d][N], X[x][N-d]) ;
          }
          // interpolate and offset interior grid points
          for (x = d ; x <= N-d ; x += D)
          {
            for (y = D ; y <= N-d ; y += D)
                X[y][x] = F4(X[y+d][x], X[y-d][x], X[y][x+d], X[y][x-d]) ;
          }
          for (x = D ; x <= N-d ; x += D)
          {
            for (y = d ; y <= N-d ; y += D)
                X[y][x] = F4(X[y+d][x], X[y-d][x], X[y][x+d], X[y][x-d]) ;
          }
          //displace other points also if needed
          if (addition)
          {
            for (x = 0 ; x <= N ; x+=D)
                  {
                for (y = 0 ; y <= N ; y+=D)
                    X[y][x] = X[y][x]+delta*gauss.Gauss() ;
                  }
                for (x = d ; x <= N-d ; x+=D)
                  {
                     for (y = d ; y <= N-d ; y+=D)
                         X[y][x] = X[y][x]+delta*gauss.Gauss() ;
                  }
          }
          D/=2 ;
          d/=2 ;
}
return TRUE ;
}
```

〈그림 7〉 GenLand Scape 함수 리스트

마지막 단계는 GenLandScape에서 생성된 데이터의 복사본을 받아내는 단계이다. GetLandData 함수는 인자로 받는 scale 값을 이용하여 불규칙한 데이터를 -scale/2 에서 + scale/2 사이의 값으로 scaling하고, 그 결과를 CLandScape의 새로운 Instance에 저장해 return 한다. 데이터 저장 객체를 참조형태(reference)로 되돌리지 않고 복사본을 되돌리는 것은 객체은닉 개념에 기초한 것으로 임의로 데이터가 파괴되는 것을 방지한다 〈그림 8〉.

작업을 수행하였다. 이를 위하여 먼저 격자형 중간점 이동 알고리즘을 이용하여 지형 데이터를 생성하는데, 이때, 프랙탈 지형 데이터를 생성하기 위한 시그마와 같은 몇 가지 초기 값이 필요하다. 따라서 지형 데이터를 생성해내는 CMidPoint :: GenLandScape 에 이러한 인자를 넘겨주어야 하는데, LandV에서는 이 초기 값을 사용자로부터 용이하게 받아들이기 위해서 〈그림 9〉와 같은 Windows 대화상자를 사용한다. 〈그림 9〉에서 보는 바와 같이 대화상자의 입력항목은 모두 디폴

```
BOOL CMidPoint :: GetLandData (CLandScape &obj, INT scale) const
{
        int i, j, size ;
        float min, max ;
        obj = X ;
        size= obj.GetSize() ;
        min = obj[0][0] ;
        max = obj[0][0] ;
        for (i = 0 ; i < size ; i++)
                for (j = 0 ; j < size ; j++)
                {
                        if (min > obj[i][j]) min = obj[i][j] ;
                        if (max < obj[i][j]) max = obj[i][j] ;
                }
        max -= min ;
        for (i = 0 ; i < size ; i++)
                for (j = 0 ; j < size ; j++)
                {
                        obj[i][j] -= min ;
                        obj[i][j] /= max ;
                        obj[i][j] *= scale ;
                        obj[i][j] -= scale / 2.0f ;
                }
        return TRUE ;
}
```

〈그림 8〉 GetLandData 함수 리스트

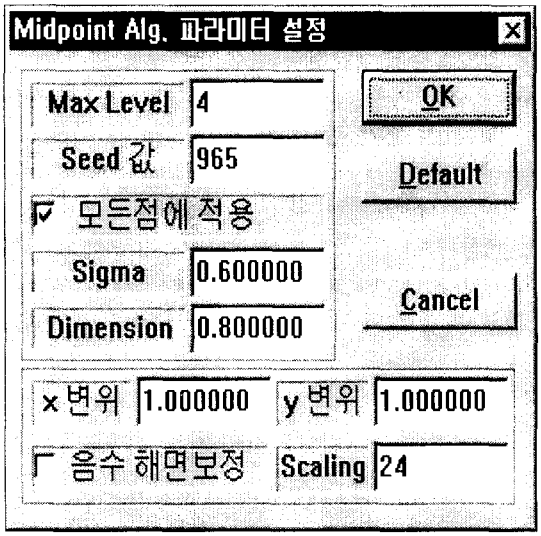
이외에도, CMidPoint 클래스에서 특기할 사항으로는 메모리 관리에 관한 부분이다. FInit 함수에서 자동으로 초기화되는 지형 데이터 저장 객체는 CMidPoint 클래스의 destructor에 의하여 모두 자동으로 정리되기 때문에, CMidPoint 클래스를 이용해 생성한 데이터는 클래스의 객체 Instance가 해제되기 전까지 유효하다. 따라서 한번 GenLandScape 함수를 호출하여 생성된 같은 데이터에 대하여 여러 번에 걸쳐 다른 scaling 값을 이용하여 데이터를 받아내는 것이 가능함은 물론, 생성과 파괴가 자동으로 이루어진다.

## 4. 지형 형상화 작업

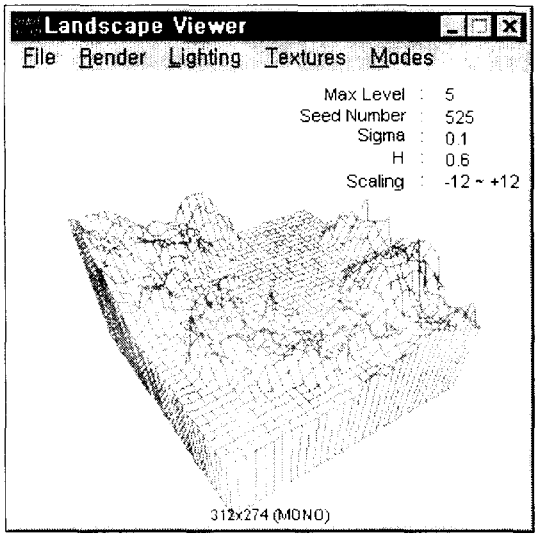
앞에서 설명한 클래스들을 기본으로 만들어진 프랙탈 지형 데이터 뷰어 LandV 를 사용하여 지형 형상화

트 값을 갖고 있으며, 사용자의 필요에 따라 손쉽게 초기 값을 변화시킬 수 있는 기능을 제공한다. LandV는 사용자로부터 받아들인 초기 값을 이용해 CMidPoint 클래스로부터 격자형 프랙탈 지형 데이터를 생성해 낸다.

〈그림 9〉에서의 Max Level 은 중간점을 구해나가는 단계를 나타낸다. 위의 예에서 4 의 값의 경우는 한 변에 2**4 = 16 개의 점이 생기고 결과적으로 모두 256 개의 점이 사각형내에 만들어진다. Seed 는 지형 데이터 생성에 필요한 난수의 초기 값이며, Sigma 는 사각형의 한 변의 길이를 나타낸다. X 변위와 Y 변위는 사각형 분할시에 분할된 사각형의 한 변의 기본길이로써, 이 값을 조정함으로써 뷰어내에 지형의 모습이 적당한 크기로 출력할 수가 있다. 또한, Scaling 은 지형의 높이를 조정하기 위한 값으로 값이 커질수록 평평



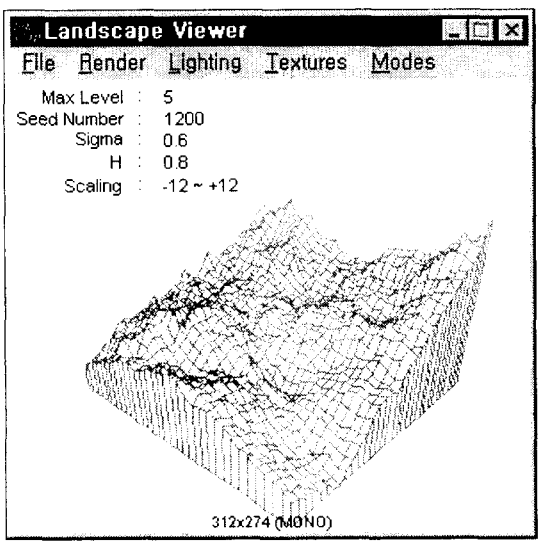

〈그림 9〉 초기값 입력 대화상자




〈그림 11〉 음수해면보정을 한 결과

한 지형이 만들어진다. 음수해면보정은 지형의 높이를 구하였을 때, 음수의 값은 모두 0 의 값으로 지정하는 것을 의미한다.

마지막으로 〈그림 10〉 및 〈그림 11〉은 LandV 를 사용하여 만들어진 결과를 보여주고 있다. 〈그림 10〉은 음수해면보정을 하지 않아서 지표면이 전부 다 울퉁불퉁한 모습을 보여주고 있으며, 〈그림 11〉은 음수해면보정의 결과로 바다위의 섬의 모양, 또는 해변가의 모습을 보여주고 있다.




〈그림 10〉 음수해면보정을 하지 않은 결과

## 5. 요약 및 결론

하나의 물체를 3차원으로 컴퓨터상에서 형상화하기 위한 많은 기법이 개발되어 왔으며, 프랙탈 기법은 특히 자연적인 물체, 즉, 산이나 구름, 숲, 나무, 등과 같은 것을 표현하는데 효과적으로 사용되고 있다. 프랙탈 기법을 사용함으로써, 간단한 수식에 의하여 원하는 양의 데이터를 무한정 만들 수 있고, 또한 반복적으로 생성 할 수 있는 장점이 있다. 그러나 이때 만들어진 많은 양의 데이터를 사용하여 하나의 물체를 형상화하기 위하여는 많은 프로그래밍 작업을 필요로 하고 있다. 여기서는 객체지향언어를 사용하여 보다 쉽게 하나의 지형을 형상화하기 위한 클래스의 한 예를 보이고 그 결과를 보였다.

그러나, 이 방법은 나름대로의 단점이 있다. 즉, 클래스에 대한 정의에 주의를 요하고, 초기 프로그래밍 작업이 쉽지 않으며, 이와 같은 클래스들을 활용하기 위하여는 객체지향언어로 작성된 Direct3D와 같은 그

래픽 지원 프로그램을 사용하여야 보다 효과적이라는 문제가 있다. 실제로 LandV는 1000 라인에 가까운 프로그래밍 작업에 의하여 완성된 것이다. 이러한 문제점은 그래픽이나 멀티미디어 작업에 편리한 표준화된 개발환경이 점점 발전됨에 따라서 점차로 해소될 것으로 기대한다.

## 참고문헌


[1] 노용덕, "프랙탈 기법에 의한 울퉁도 형상화 사례 연구", 한국 시뮬레이션학회 논문지, 4권,1호(1995), pp113-119

[2] Fournier, A. and Reeves W.T., "A Simple Model of Ocean Waves", Computer Graphics, Vol.20, No.4, pp. 75-84

[3] Fournier, A. and Fussell D. and Carpenter L., "Computer rendering of stochastic models", Communications of the ACM, Vol.25, No.6, pp.371-84

[4] Hearn,D., Computer Graphics, pp.205-210, Prentice-Hall, 1987

[5] Mandelbrot, B.B., "Comment on Computer Rendering of Fractal Stochastic Models", Communications of the ACM, Vol.25, No.8, pp.581-584, 1982.

[6] Mandelbrot, B.B., "The Fractal Geometry of Nature", W.H.Freeman, San Francisco, 1982

[7] Perlin, K., "An Image Synthesizer", Computer Graphics, Vol.19, No.3, pp.287-296

[8] Watt,L., Fundamentals of Three-Dimensional Computer Graphics, pp.255-261, Addison Wesley, 1989.

[9] Musgrave,F.K., Kolb,C.E., and Mace, R.S., "The Synthesis and Rendering of Eroded Fractal Terrains", Computer Graphics, Vol 23, No.3, pp.41-50, July, 1989.

[10] Lewis,J.P., "Generalized Stochastic Subdivision", ACM Transactions on Graphics, Vol.6, No.3, pp. 167-190, July, 1987.

[11] Miller, Gavin S.P., "The Definition and Rendering of Terrain Maps", Computer Graphics, Vol.20, No. 4, pp.39-48, 1986.

[12] Szeliski, R., and Demetri Terzopoulos, "From Splines to Fractals", Computer Graphics, Vol.23, No.3, pp. 51-60, July, 1989.

[13] Peitgen, H.O. and Dietmar Saupe (eds.), "The Science of Fractal Images", Springer-Verlag, New York, 1988


## ● 저자소개 ●


노용덕

1976년 서울대학교 산업공학과 졸업(학사)

1984년 Auburn Univ. I.E. (MS)

1987년 Auburn Univ. I.E. (Ph.D)

1976～1980년 국방과학연구소

1988～현 재 세종대학교 전산과학과 부교수

관심분야 컴퓨터 그래픽스, 시뮬레이션, 및 성능분석