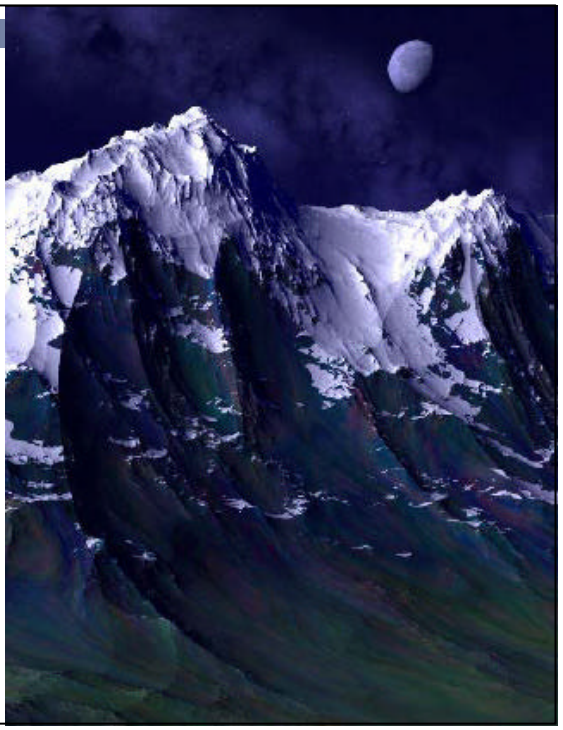
Fractal Terrains

Sheelagh Carpendale
Fall 2003

Image: ‘Emile’ by Ken Musgrave

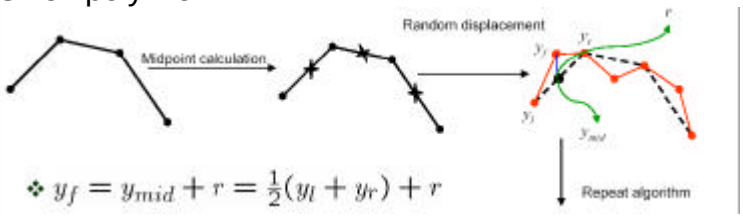


Terrain Modeling by Fractals

- We can obtain realistic models of terrain by using random fractals
- Von Koch, Sierpinsky, Menger and Julia fractals are determinist
- Adding randomness for natural fractals
- Random walk and Brownian motion (from statistical mechanics1827)
- Zigzag and erratic movement of particles in a gas or other fluid

Simple Algorithm

- How can we generate the silhouette of mountains?
- midpoint displacement (adding noise to the Faber subdivision)
- Given polyline :

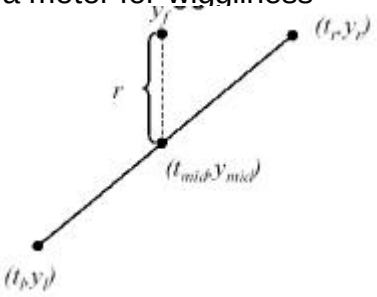


$$y_f = y_{mid} + r = \frac{1}{2}(y_l + y_r) + r$$

- repeat this process several times
- What are related factors for r ?

Related Factors

- The dimension of final fractal (D): a meter for wiggliness
- 1 < D < 2
- Distance between t_l and t_r
- Type of randomness

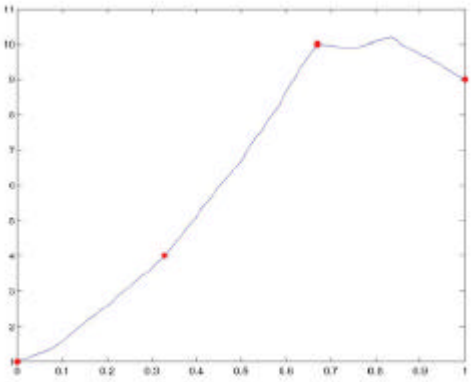


- Gaussian (Normal) distribution with :
  - mean of zero
  - variance relates to fractal dimension
- Check statistical software for this distribution

Example

- t = 0 0.3300 0.6700 1.0000
- y = 1.0000 4.0000 10.0000 9.0000

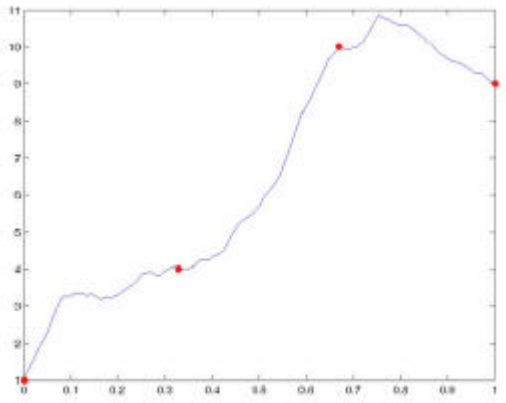
- Levels of subdivision = 6
- Fractal dimension = 1.1
- Noise factor = 0.1



Example 2

- t = 0 0.3300 0.6700 1.0000
- y = 1.0000 4.0000 10.0000 9.0000

- Same initial points
- Levels of subdivision = 6
- Fractal dimension = 1.4
- Noise factor = 0.4

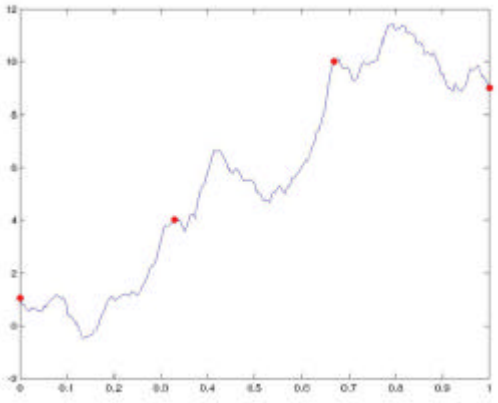


Example 3

- t = 0 0.3300 0.6700 1.0000
- y = 1.0000 4.0000 10.0000 9.0000

- Same initial points

- Fractal dimension = 1.5
- Standard Brownian motion
- Noise factor = 0.5

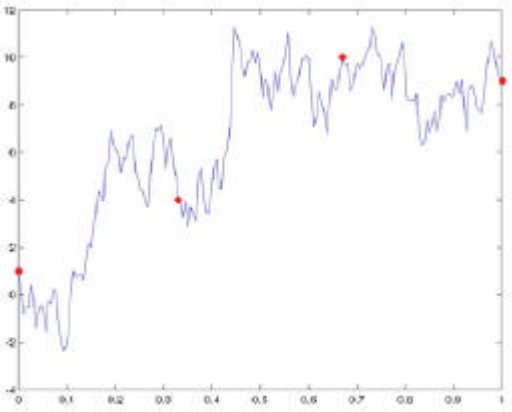


Example 4

- t = 0 0.3300 0.6700 1.0000
- y = 1.0000 4.0000 10.0000 9.0000

- Same initial points

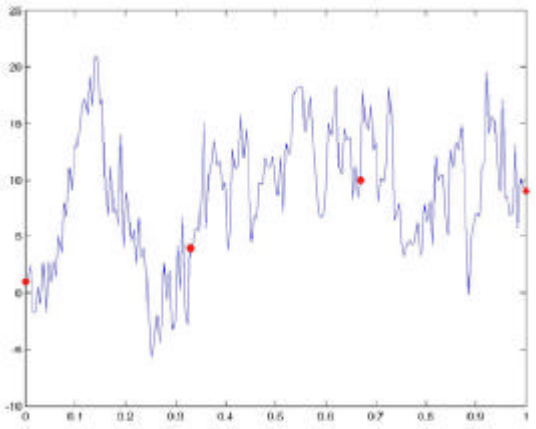
- Fractal dimension = 1.7



Example 5

- t = 0 0.3300 0.6700 1.0000
- y = 1.0000 4.0000 10.0000 9.0000

- Same initial points
- Fractal dimension = 1.9



Determining The Variance at Each Level

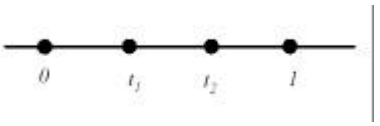
- The variance of the random number can express the fractal dimension of the curve
- In all fractals we have a self similarity
- Scaling of some part of the object has the same visual appearance as the complete object
- Let start with y(0) and y(1)

$$var(y(1) - y(0)) = \sigma^2$$

- Self similarity gives

$$var(y(t_2)) - y(t_1)) = |t_2 - t_1|\sigma^2$$
$$\text{for } 0 \leq t_1 \leq t_2 \leq 1$$

- If we apply the same variance for any portion, after magnification we obtain more deviation



General Case

- Decreasing the variance for a given general D ?

$$var(y(t_2) - y(t_1)) = |t_2 - t_1|^{2H}\sigma^2$$

- where H=2-D

$$\Delta_n^2 = \frac{\sigma^2}{(2^n)^{2H}}(1 - 2^{2H-2})$$

Simplification

- Not perfect
- Use uniform points at the beginning
- Scale variance by a noise_factor instead of using H and D
- let u=normal(0,1) and j be the level of the subdivision then we have:

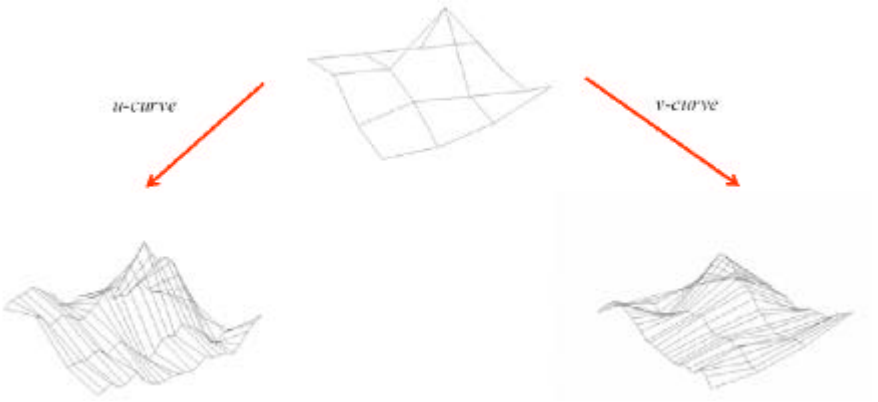
$$r = \frac{1}{2^{j-1}} * noise\_factor * u$$

Algorithm

```
Input Y, m, noise_factor
//Y : input poly-line, m : level of subdivision
for j=1 to m
    n=size(Y);
    K=1;
    for i=2 to n
        Z(k)=Y(i-1);
        u=normal(0,1);
        r=noise_factor*(1/2^(j-1))*u;
        Z(k+1)=0.5*(Y(i-1)+Y(i)) + r; // midpoint + noise
        k=k+2;
    endfor;
    Z(k)=Y(n);
    Y=Z;
endfor;
```

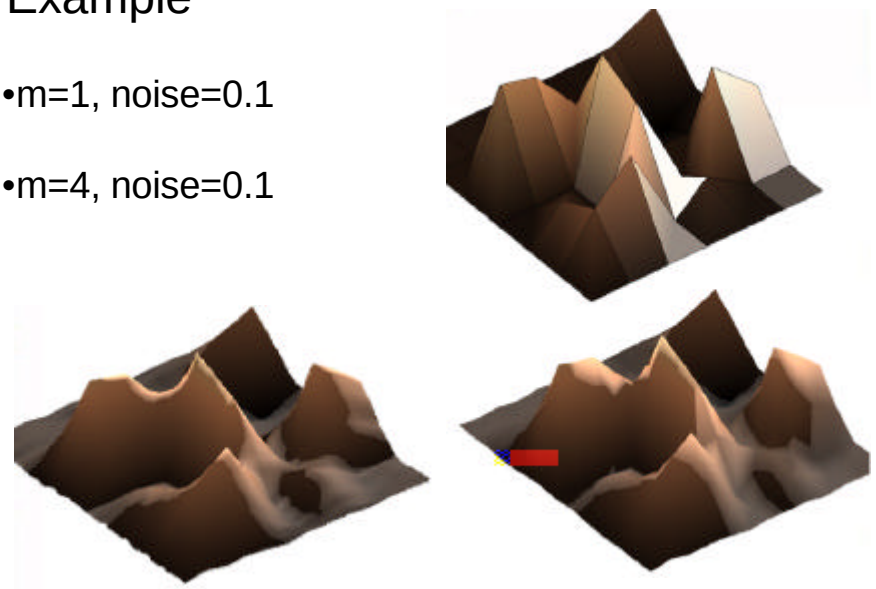
Extension to 3D

- The same as tensor product surface algorithm
- Repeat for each u-curve and then repeat for each v-curve
- Displace new inserted points be a random noise (r)



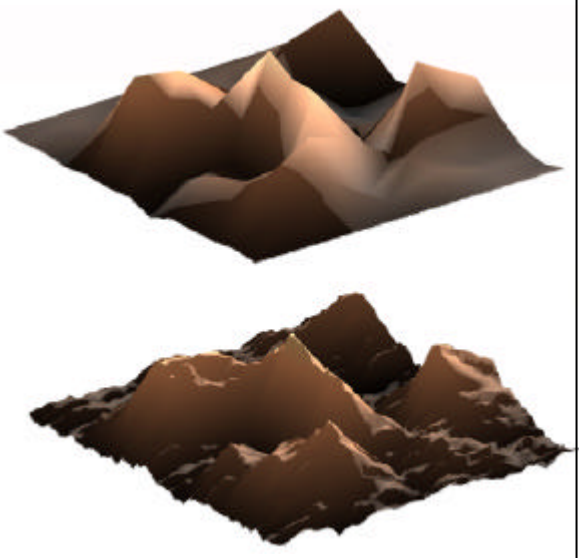
Example

- m=1, noise=0.1
- m=4, noise=0.1

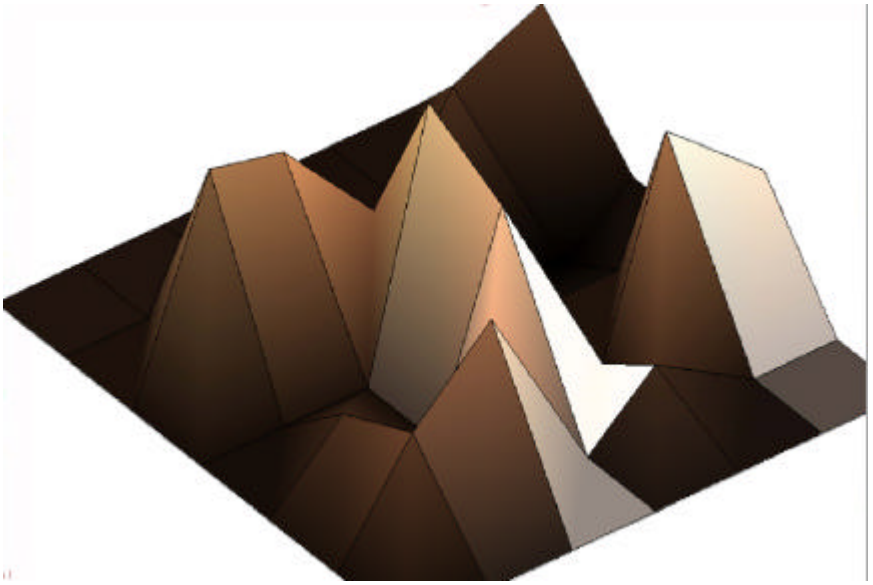


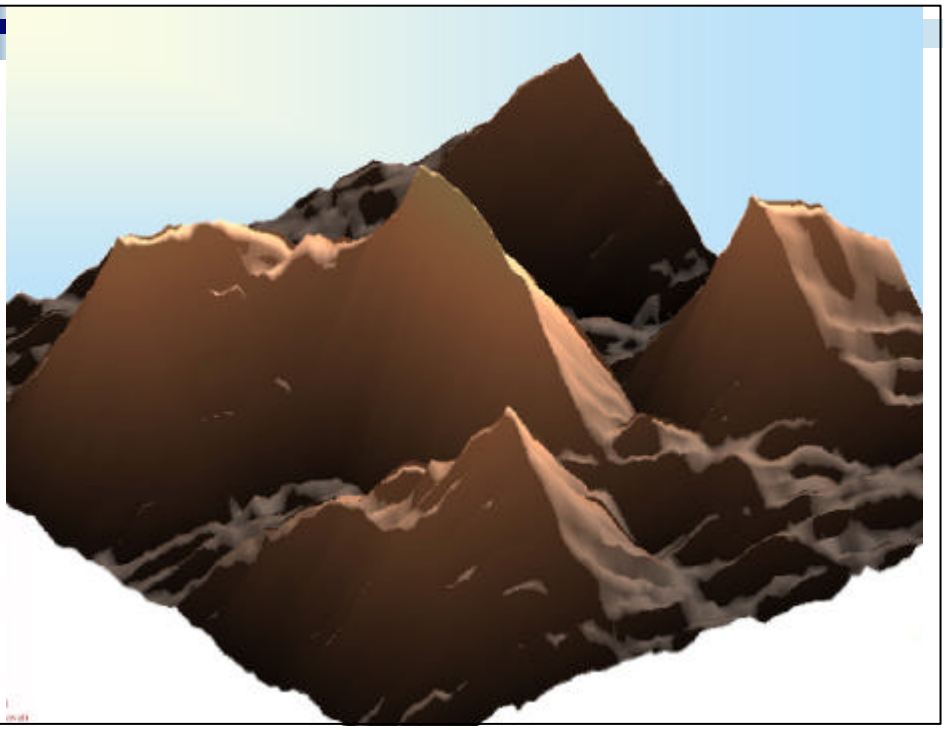
Example

- m=1, noise=0.4
- m=4, noise=0.4



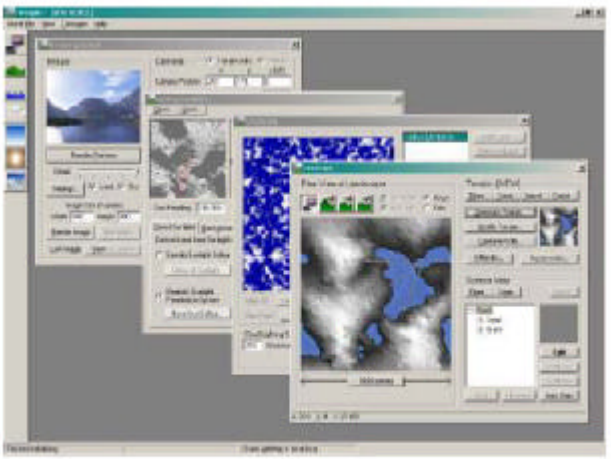
Close view (0.4)





Terrain software

Terragen software demo



http://www.planetside.co.uk/terragen/

The following algorithm shows the main part of the Z-Buffer algorithm

a) What is the best candidate for c in line 2?

b) What are the missing commands in lines 9 and 10?

```
1- for all positions (x,y) in the view screen
2-          frame(x,y)=c
3-          depth(x,y)=max_distance
4- end
5- for each polygon in the mesh
6-          for each point(x,y) in the polygon-fill algorithm
7-                    compute, z, the distance of corresponding 3D-point from cap
8-                    if depth(x,y) > z // a closer point
9-                              ??????????????
10-                             ??????????????
11-                   endif
12-         endfor
13-endfor
```

The following algorithm shows the main part of the Z-Buffer algorithm

a) What is the best candidate for c in line 2?

b) What are the missing commands in lines 9 and 10?

```
1- for all positions (x,y) in the view screen
2-          frame(x,y)=background-colour
3-          depth(x,y)=max_distance
4- end
5- for each polygon in the mesh
6-          for each point(x,y) in the polygon-fill algorithm
7-                    compute, z, the distance of corresponding 3D-point from cap
8-                   if depth(x,y) > z // a closer point
9-                              depth(x,y) = z; // update the z buffer
10-                             frame(x,y) = I(p); // shading – the intensity 'I' at point p
11-                   endif
12-         endfor
13-endfor
```

This figure shows one of the triangles in a 3D mesh. For this triangle, we have some given information, which is provided, in the table fields.

Point	**A**	B	C	D	E	F
Coordinates	(0,0,0)	(0,4,2)	(2,2,2)			
Intensity	80	40	160			

In addition, D is the midpoint of A and C, E is the midpoint of B and C, and F is 1/4 of the way from D to E

a)Find intensity value of F using Gouraud shading

b)Find z coordinate of F.