MSc Computer Games and Entertainment

Maths & Graphics Unit 2011/12

Lecturer: Gareth Edwards

Rendering Triangular Terrain Mesh

Breckon, Chenney, Hobbs, Hoppe, Watts

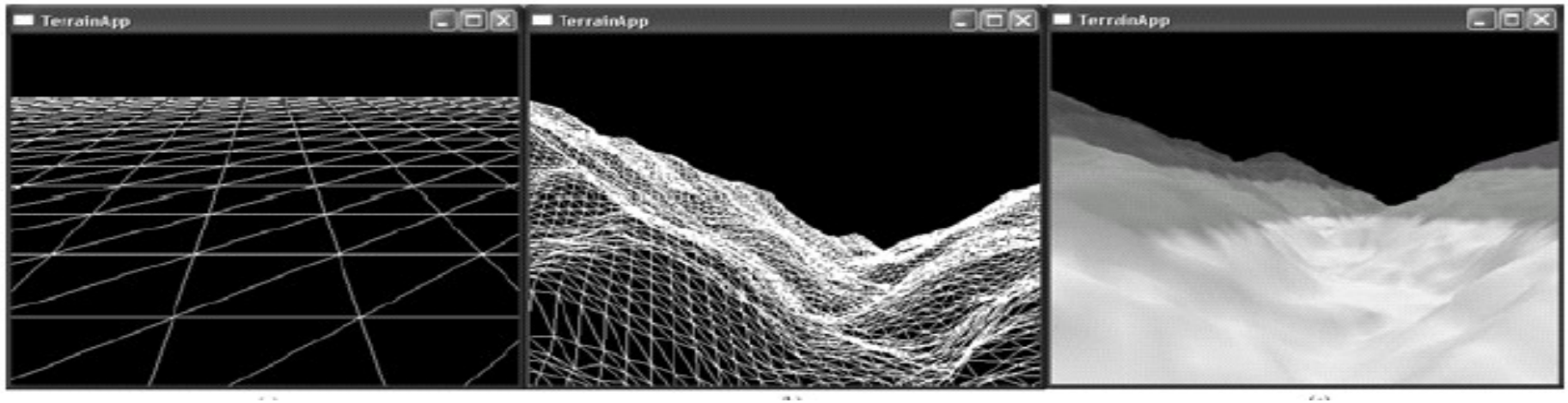
Terrain

- Terrain is important to many games
- As a model, it is very large
- Creating every point explicitly by hand is not feasible, so automated *terrain generation* methods are common
- When rendering, some of the terrain is close, and other parts are far away, leading to *terrain LOD* algorithms

Terrain

- A terrain mesh is nothing more than a triangle grid, with heights of each vertex in the grid specified in such a way that the grid models a smooth transition from mountain to valley, simulating a natural terrain.

- To make it realistic we apply a nice texture showing sandy beaches, grassy hills, and snowy mountains



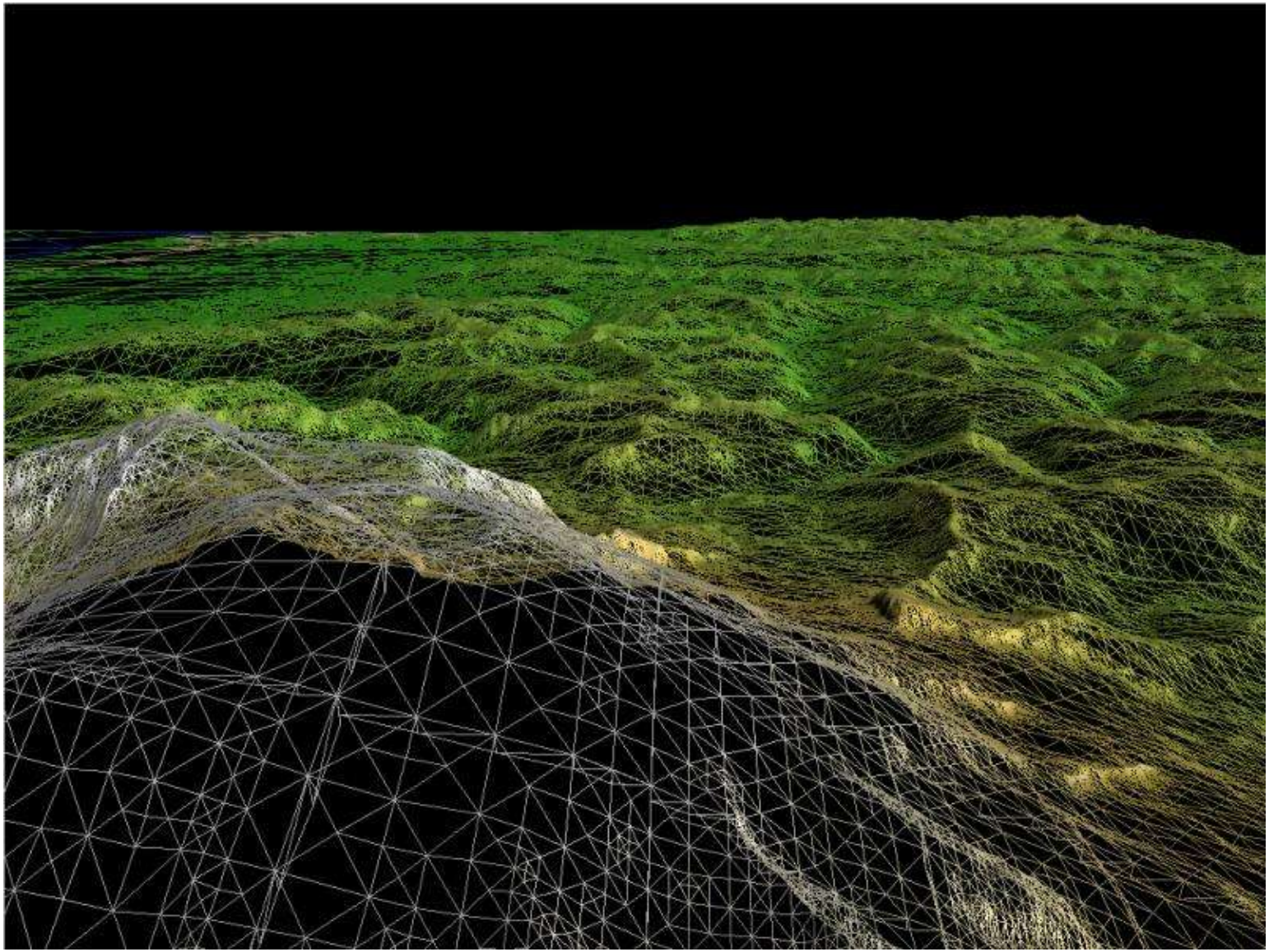
Terrain

- To generate terrain a simple solution consists in using a brute force approach:
  - It simply stores the entire terrain vertex/index data and then renders it.
  - For games requiring a small terrain, this approach is workable with modern graphics cards that support hardware vertex processing.
- However, for games requiring larger terrains, you have to do some kind of level of detail or culling because the enormous amount of geometry data needed to model such huge terrains can be overwhelming for a brute force approach.

What are the challenges?

- Terrains tend to be huge
- Visualizing a terrain of 16384 x 16384 samples (164 x 164 km, samples 10 m apart) requires drawing 536,805,378 triangles (268,435,456 vertices)
- Adding a 32 bit RGBA texture and having 16 bit heights, the total memory consumption is above 1.5 Gb!

What are the challenges?



Terrain Generation Models

- Paint the height field (artist generated)

- Various pseudo-random processes:
  - Independent random height at each point
  - Fault generation methods (based on random lines)
  - Particle deposition
  - Fractal terrain from meshes

- Triangulated point sample methods

- Plenty of research and commercial tools for terrain generation:

  - http://www.vterrain.org/LOD/Implementations/

Surface Attributes

- Rather than painting texture and color directly, paint attributes
  - e.g. Grass, Trees, Water, …
  - Features can have game-play characteristics: speed of motion, impassable, …

- Then automatically generate colors/textures
  - Care must be taken at the boundaries of the regions

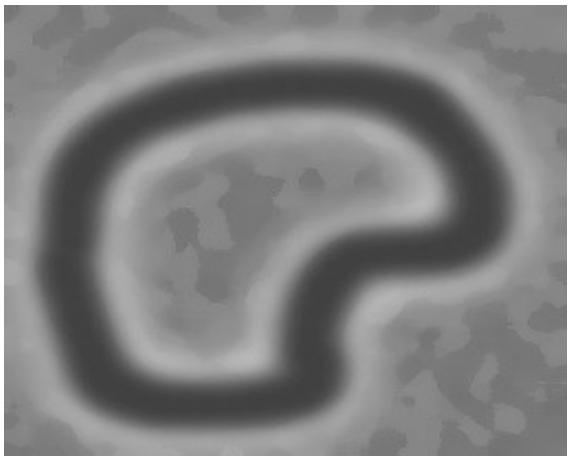
- Can also work for the terrain itself
  - e.g. Maps that have a finite number of possible heights, with ramps between them

Representing Terrain

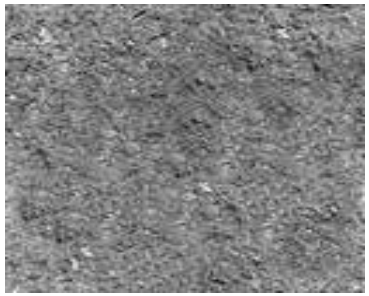
- Terrains are often represented using *elevation maps*

- An elevation map is a 2D array of regularly spaced height samples

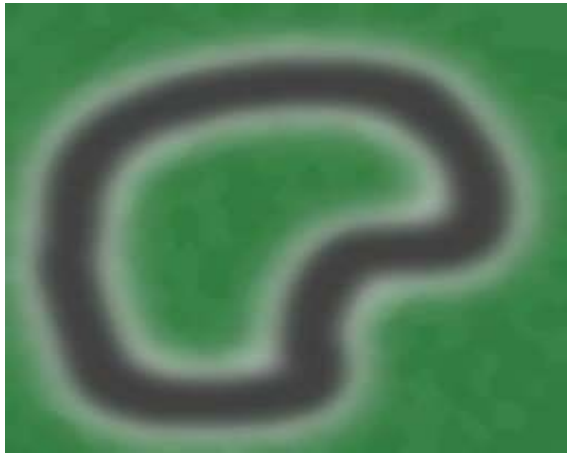
Painted Terrain Example



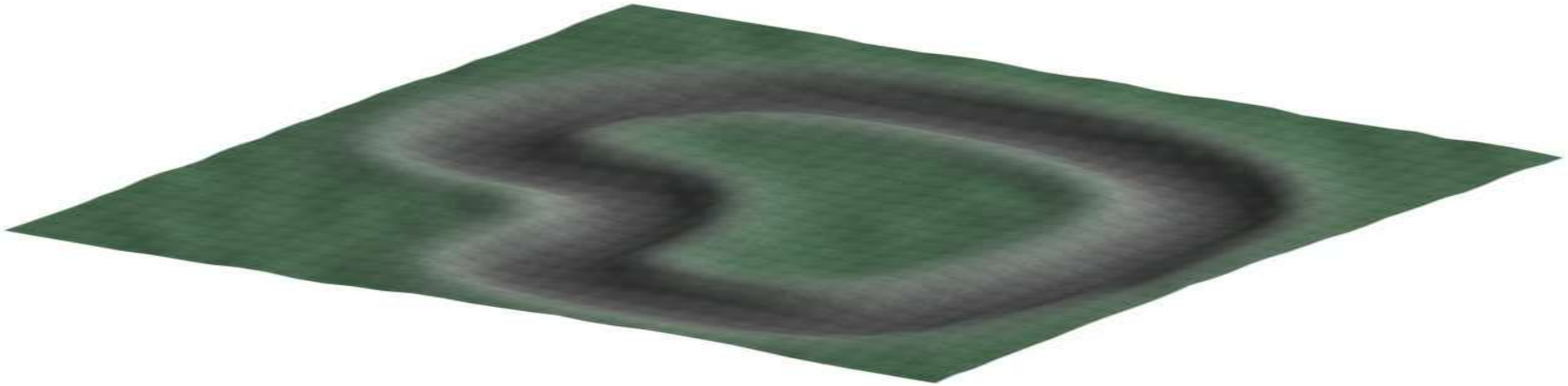
Height



Texture



Color



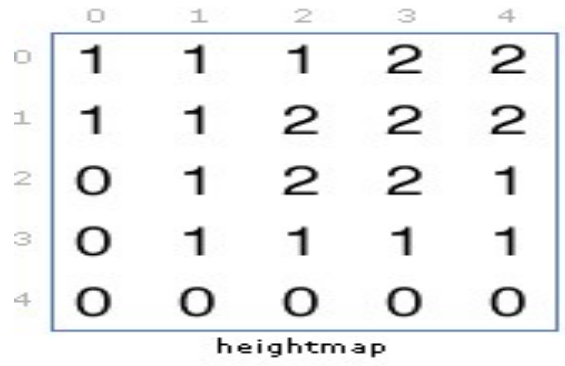
Representing Terrain

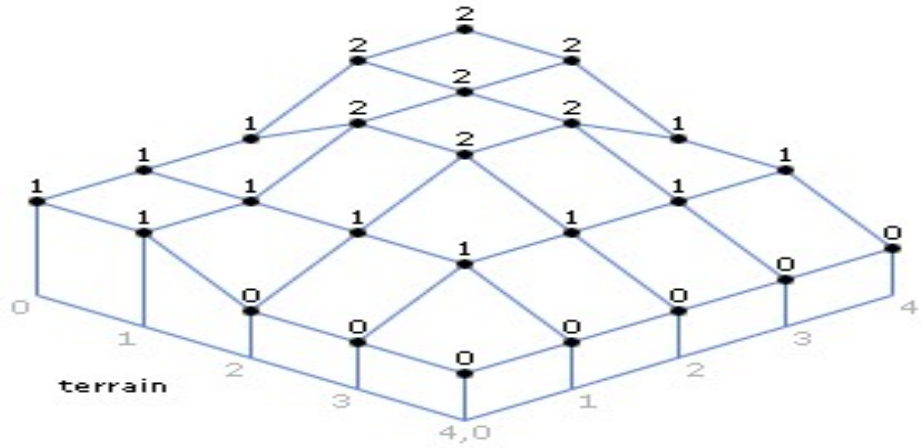
- The base representation for terrain is usually a height field
  - $z=f(x,y)$ for (x,y) within the limits of the space

- There are two common ways to represent the function $f(x,y)$
  - Explicitly store the value of $f(x,y)$ for a discrete grid of (x,y) locations
    - Generally interpolate or triangulate to get points not on the grid
    - Easy to figure out what the height of the terrain is at any given (x,y)
    - Expensive to store the entire terrain
  - Store a polygonal mesh
    - Cheaper if the terrain has large flat areas
    - Harder to figure out what the height is under the player (have to know which triangle they are in)

Heightmaps

- We use a heightmap to describe the hills and valleys of the terrain.
- A heightmap is an array where each element specifies the height of a particular vertex in the terrain grid.





- One of the possible graphical representations of a heightmap is a greyscale map, where darker values reflect portions of the terrain with low altitudes and whiter values reflect portions of the terrain with higher altitudes.

Create Heightmap as Greyscale

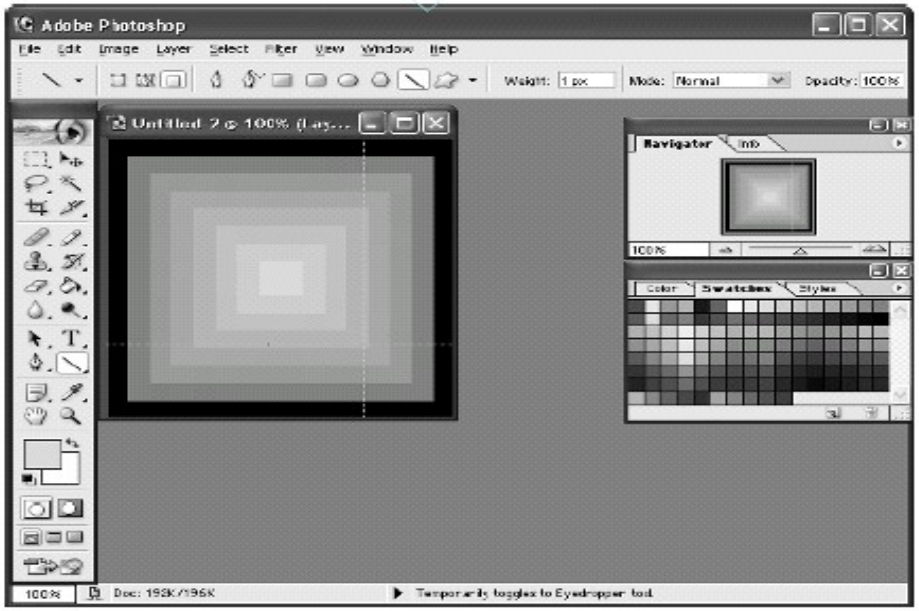
Heightmaps can be generated either :

- procedurally

or

- In an image editor such as Adobe Photoshop. Using an image editor is probably the easiest way to go, and it allows you to create the terrain interactively and visually as you want it.

In addition, you can take advantage of your image editor features, such as filters, to create interesting heightmaps.



Random Process Generation

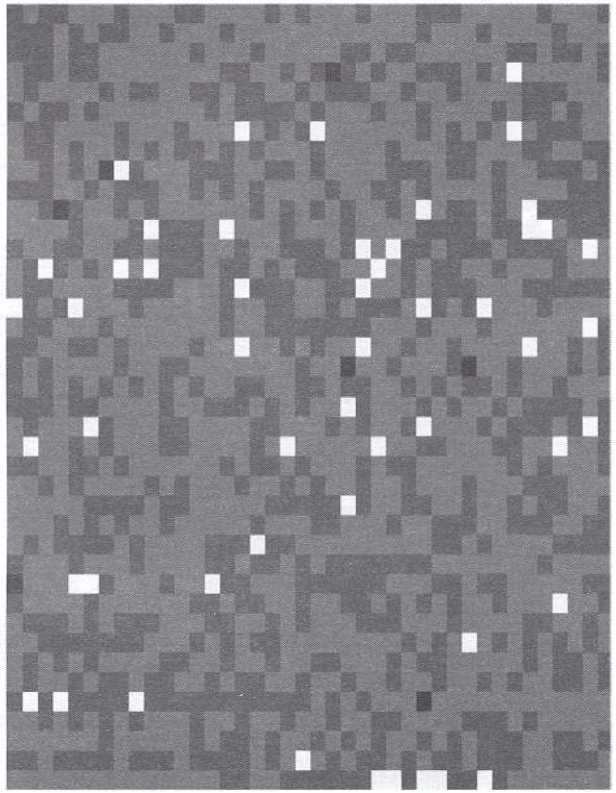
- The claim is that real terrain looks “random” over many scales
- Hence, random processes should generate realistic terrain
  - The catch is knowing *which* random processes
- Some advantages:
  - Lots of terrain with almost no effort
  - If the random values are repeatable, the terrain can be generated on the fly, which saves on storage
- Some disadvantages:
  - Very poor control over the outcome
  - Non-intuitive parameter settings

Random Points

- Randomly choose a value for each grid point
  - Can make each point independent (don't consider neighbors)
  - Can make points dependent on previous points - a *spatial Markov process*

- Generally must smooth the resulting terrain
  - Various smoothing filters could be used

- Advantage: Relatively easy to generate on the fly



Fault Formation

- Claimed to model real fault lines, but not at all like real faults

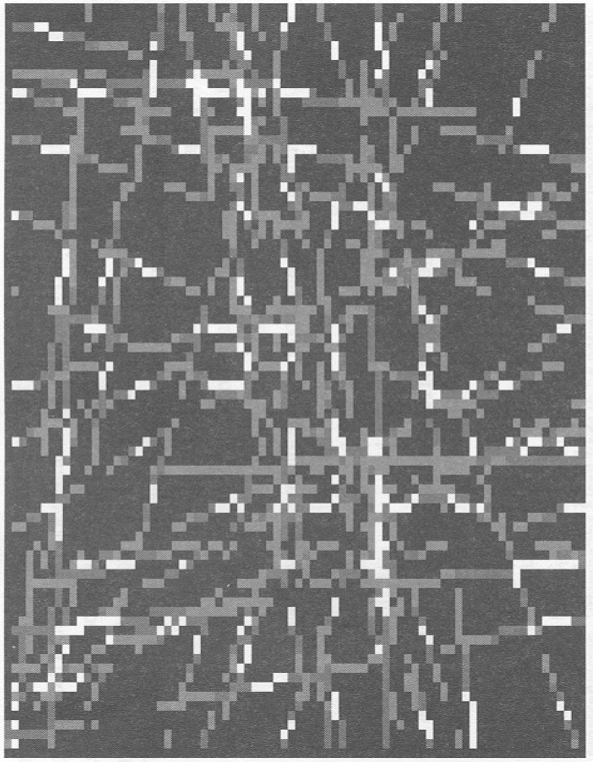
- Process 1:
  - Generate a random line and lift everything on one side of the line by *d*
  - Scale *d* and repeat

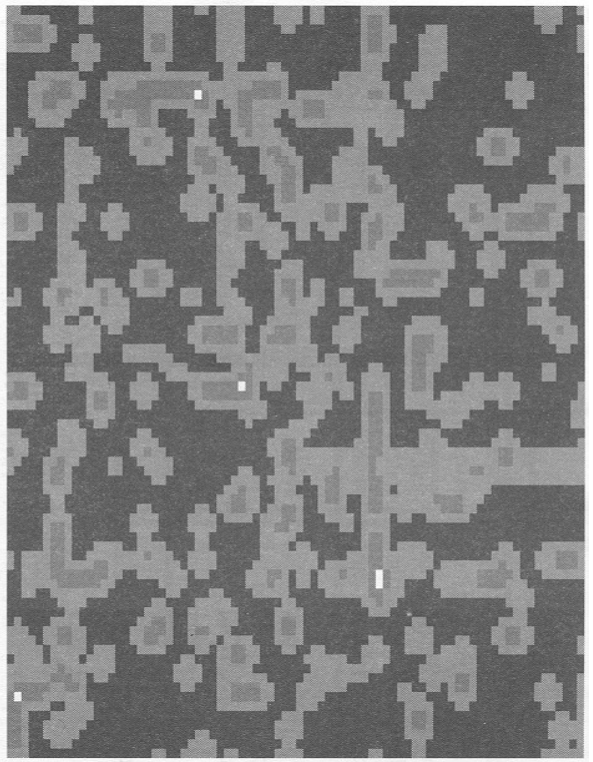
- Process 2:
  - Generate a random line and lift the terrain adjacent to the line
  - Repeat (maybe with a scale function)

- Smoothing is important

Fault Formation Example

Initial





Smoothed

Particle Deposition

- Supposed to model lava flow (or glacial drumlins!)

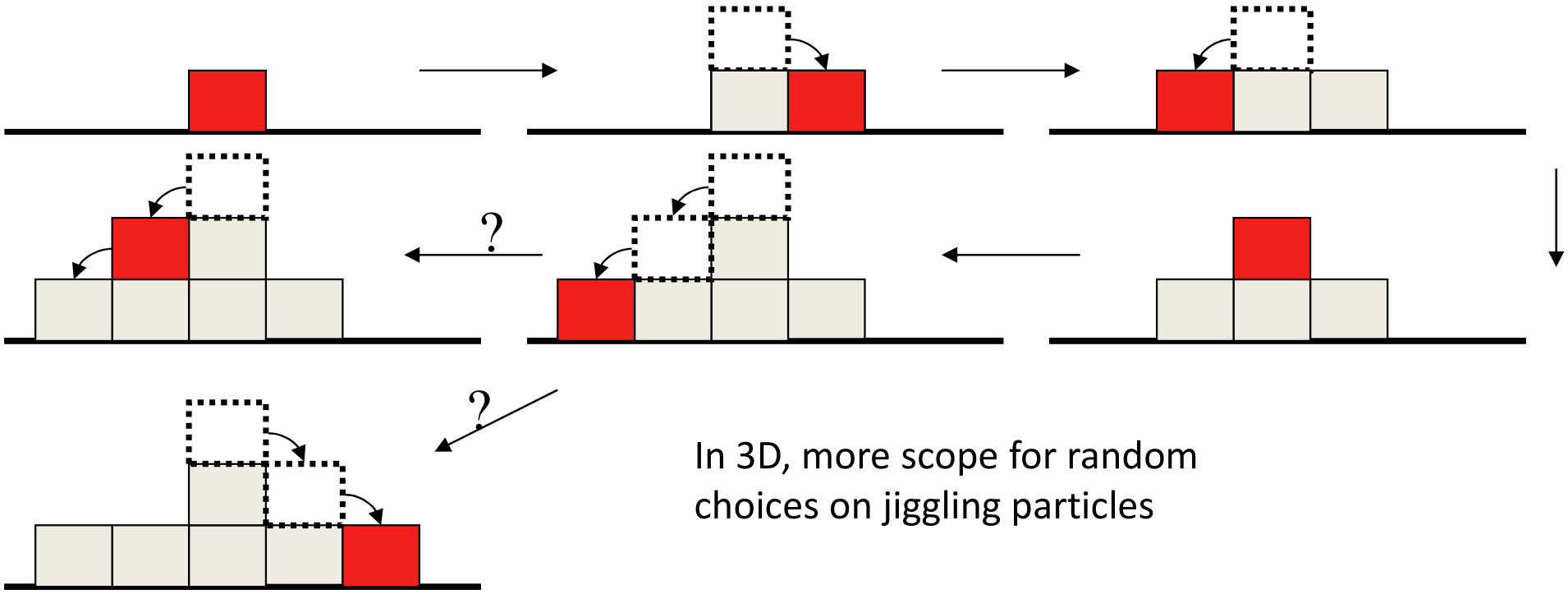
- Process:
  - Drop a particle onto the terrain
  - Jiggle it around until it is at most one unit higher than each of its neighbors
  - Repeat
  - Occasionally move the drop point around

- To do volcanoes, invert everything above a plane

- To do sinkholes, invert the hill

- To do rows of hills, move the drop point along a line

Particle Deposition Process



In 3D, more scope for random choices on jiggling particles

Fractal Terrain

- Based on subdivision of a course polygon mesh
- subdivision adds detail to the mesh in a random way
- Algorithm (starting with a triangular mesh):
  - Split each edge, and shift the new vertex up or down by a random amount
  - Subdivide the triangles using the new vertices
  - Repeat
- Also algorithms for quadrilateral meshes
- http://www.gameprogrammer.com/fractal.html

Required For Next Week

Required reading for next week....

You should read up to the end of Lengyl Chapters 4.

Please also read:

- Chapter 13 on Fluid Simulation and
- Chapter 15 on Curves and Surfaces

You should read up to the end of Dunn & Parberry Chapter 9.

Please also read:

- Chapter 14 on Triangle Meshes
- Chapter 15 on 3D Math for Graphics

You will also be required to hand in your first Assignment!

There are five parts worth 10, 40, 20, 10 and 20 marks respectively:

- Read a configuration file containing L-System parameters (10).
- Display the resulting L-System model such that using Hotkeys successive iterations can be manually stepped through (40).
- Hotkey the read configuration file process such that different hotkey can load and display at least four different configuration files (20)
- Hotkey the increment/decrement of at least three of the L-System parameters which are then used to recreate the L-System model (10).
- Write an illustrated description (name it "*lsystem.doc*" and it must be 1200 words or more) of your project detailing your objectives, data classes and other structures (20).

Please Note.....

- That the marks awarded will not depend solely on satisfying the implementation requirements detailed above, but also on the elegance of your solution, the techniques used, the speed of your application, quality of code documentation and the quality of the images realised.

- That your implementation should be able to render the L-Systems illustrated in the assignment document.

Software requirements....

- It will be implemented in C++ using the supplied DirectX framework RTVS_Lite.
- Your project should be created in Microsoft Visual Studio 2008 or 2010.
- There must be a clear distinction between your application framework and your application (i.e. between those parts of your application that provide ‘housekeeping’, and those that implement your L-System).
- You must include a complete working example in your submission.
- You must provide all dependent files.
- You must ensure that the Lecturer can compile and run your application on the Lecturer’s PC.

Submission requirements

- When you have completed your assignment, make a copy of your project and the written description of your project, other project files, output images and other associated files in to a directory called *lsystemproject* in your home directory. Zip this file.

- Write a short *README.txt* plain text file, describing what (and why) you have included within your *lsystemproject* project folder.

- Upload the *lsystemproject.zip* and the *README.txt* file to your assignment area.

End