FRESNO STATE
Discovery. Diversity. Distinction.

Algorithms and Data Structures (CSci 115)

California State University Fresno
College of Science and Mathematics
Department of Computer Science
H. Cecotti

Learning outcomes

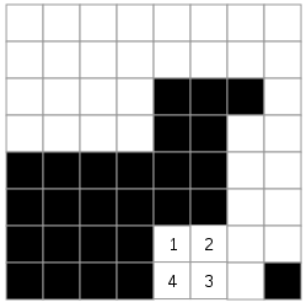
- Other types of tree based data structures
  - K-D tree
  - Quadtree
    - 2D
  - Octree
    - 3D
  - BSP
- Applications
  - Terrain Generation

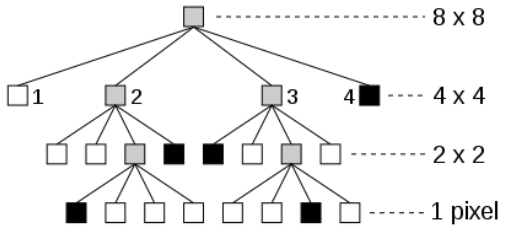
Rationale

- Previous trees
  - How to organize a set of elements
    - 1D: each element is represented by a single key

Quadtree

- Tree data structure
  - Each internal node has exactly 4 children
  - To efficiently store data of points on a 2D space
- Applications
  - 2D compression of images
    - where each node contains the average color of each of its children
    - deep in the tree → more details of the image.
  - Mesh generation



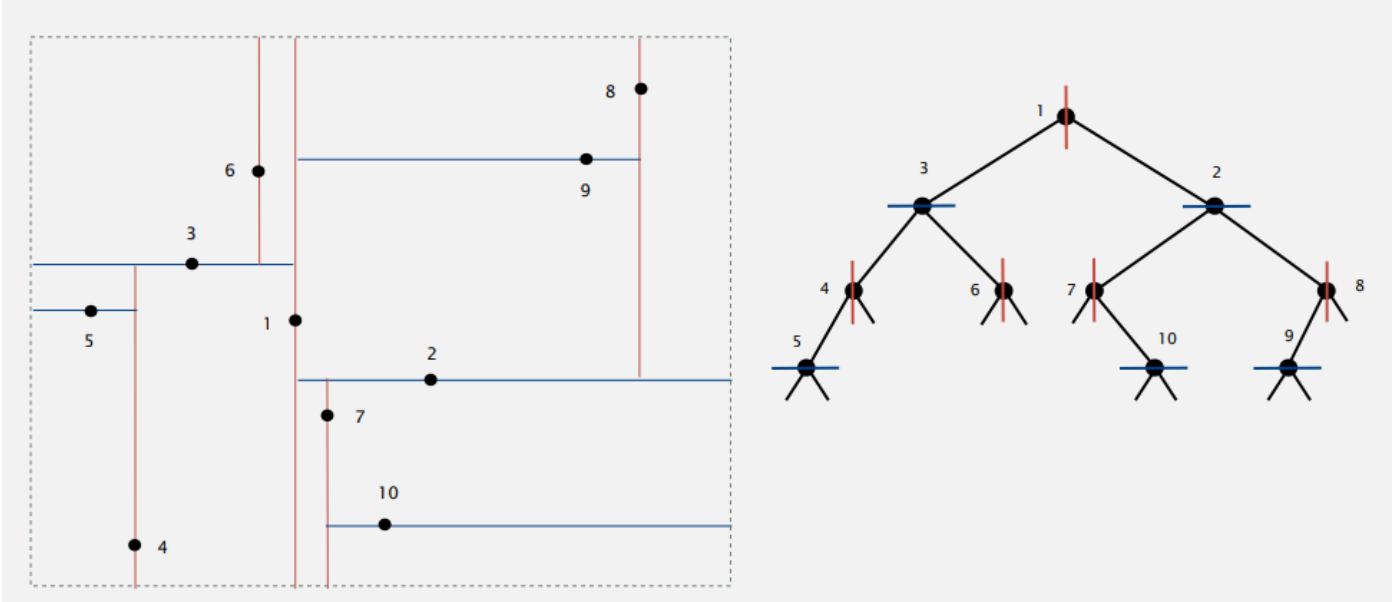


Quadtree

- Construction
  1. Divide the current 2D space into 4 boxes.
  2. If (a box contains 1 or more points in it)
     - Then create a child object, storing in it the 2D space of the box
     - Else a box does not contain any points: do not create a child for it
  3. Recurse for each of the children.

K-D tree

- Type of partitions

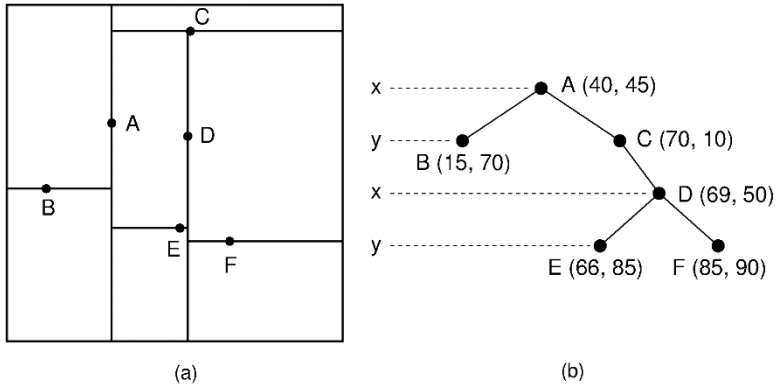


K-D tree

- K-D tree (John Bentley – 1975)
- Binary tree
  - Every node is a k-dimensional point
- Each non-leaf node
  - Generation of a splitting hyperplane
    - dividing the space into 2 parts known as half-spaces
  - Points to the **left** of this hyperplane: **left** subtree of that node
  - Points **right** of the hyperplane: represented by the **right** subtree.
- Hyperplane direction
  - Every node in the tree: associated with one of the k-dimensions
    - with the hyperplane **orthogonal** to that dimension's axis.

K-D tree

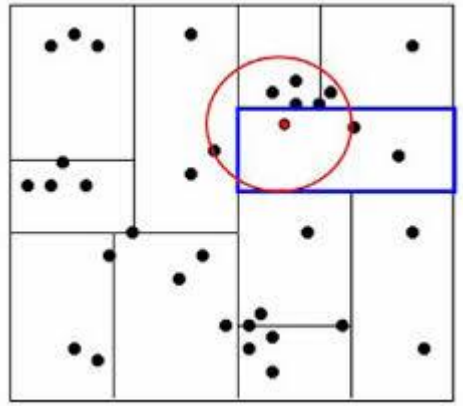
- The k-d tree
  - ➢ It differs from the BST because in each level of the kd tree makes **branching decisions** based on a particular search key associated with that level
    - the discriminator
  - ➢ It could be used to unify key searching across any arbitrary set of keys
    - Example: name + location
  - ➢ It is used to support search on **multi-dimensional** coordinates
    - locations in 2D or 3D space.



(a) The kd tree decomposition for a 128×128-unit region containing 7 data points.
(b) The kd tree for the region of (a).

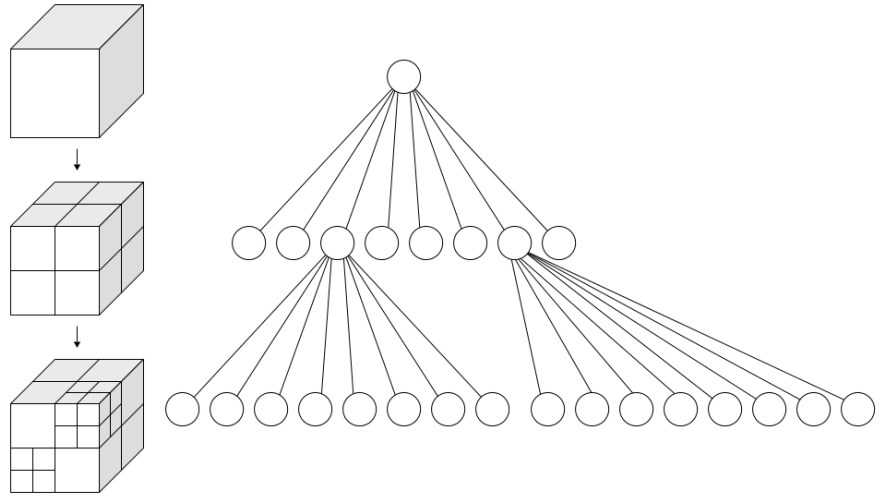
K-D tree

- Typical functions in data structures
  - Search/Insert/Delete
- For pattern recognition problems
  - Search the closest neighbor
- **Main steps**
  - Search for the leaf that contain the query point
  - Get the distance between query point and the leaf node
  - Go back up in the tree to find if you can minimize this distance
    - **If** (the circle (hypersphere) crosses the plane)
    - **Then**
      - There could be nearer points on the other side of the plane, so we must move down the other branch of the tree from the current node looking for closer points (same recursive process as the entire search)
    - **Else**
      - The algorithm continues walking up the tree, and the entire branch on the other side of that node is eliminated.



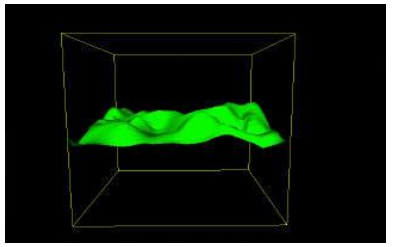
Octree

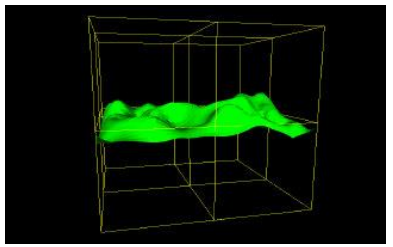
- Tree data structure
  - Each internal node has exactly 8 children
- 3D version of quadtree
- Applications
  - Space partitioning
  - 3D representation of meshes
  - If you want to do your own minecraft

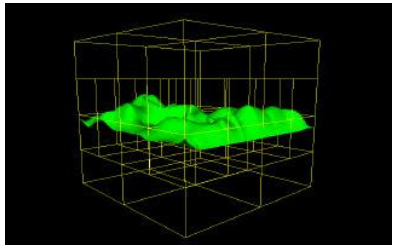


Octree

- Method:
  - Octrees are a space partitioning data structure
    - Octrees in the context of modeling
      - In modeling the leaf nodes were either **filled** or **empty**
      - The leaf nodes contain a list of triangles that are in that sub-section of space
  - The process:
    1. A single cube is placed around the entire world
    2. If (the cube contains too many triangles)
    3. Then the cube is sub-divided into 8 smaller axis-aligned cubes
    4. Recurse until we hit the stopping condition

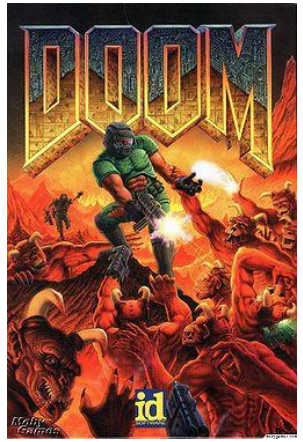


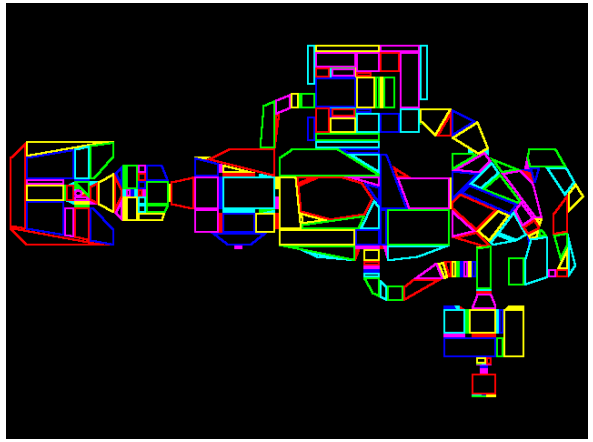




BSP

- Binary Space Partitioning (BSP)
  - recursively subdividing a space into convex sets by hyperplanes
- Applications
  - clever way of sorting the subsectors into the right order for rendering.
  - Example: DOOM





BSP

- BSP divides the level up into a binary tree
  - Each location in the tree is a node
    - Representing a particular area of the level
      - the root node represents the entire level
    - At each branch of the tree:
      - a **dividing line** that splits the area of the node into 2 subnodes.
      - at the same time, the dividing line divides linedefs into line segments called segs.
- At the leaves of the tree are convex polygons
  - where it is not useful to divide the level up any further (convex polygons == subsectors, bound to a particular sector)
  - each subsector has a list of segs associated with it.
- Algorithm
  - Start at the root node.
  - Draw the child nodes of this node recursively. The child node closest to the camera is drawn first. (This can be found by looking at which side of a given node's dividing line the camera is on.)
  - When a subsector is reached, draw it.
- When a level is revised using an editor → BSP data must be updated **only if** a structural change has been made
  - the level can still be used without rebuilding its nodes, but only non-structural changes will be taken into account by the engine).

Terrain generation

- Environments get larger, more complex.
- Difficult task for game designers to create a coherent world.
  - ➢ Zelda 3 (Nintendo, 1991)
    - 220 yards x 220 yards
- You cannot place manually each rock, each tree…

Terrain generation

- The algorithms to generate terrain
  - Valid for 2D games
  - Automatic level creator
- Challenges in Terrain generation
  - To be pretty only - consistent
  - To respect gameplay rules
    - Problem of sequence breaking (e.g. Metroid Prime, Half Life2, …)

Type of terrains

- The choice of the method:
  - Lots of peaks?
  - Lots of hills?
- Main algorithms
  - Midpoint displacement
  - Fault formation
  - Hill algorithm

Artificial image

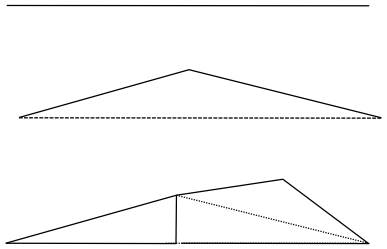
Giant causeway
“Random but with a structure”
→ particular patterns

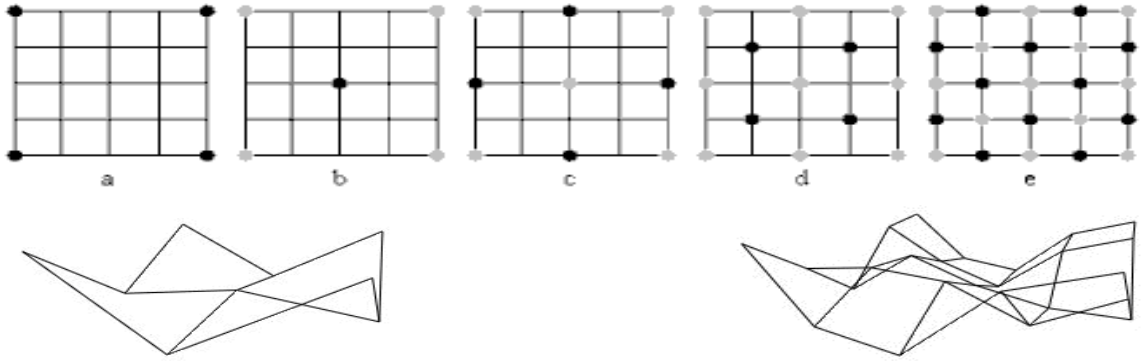
Death Valley

Bryce Canyon

Midpoint displacement

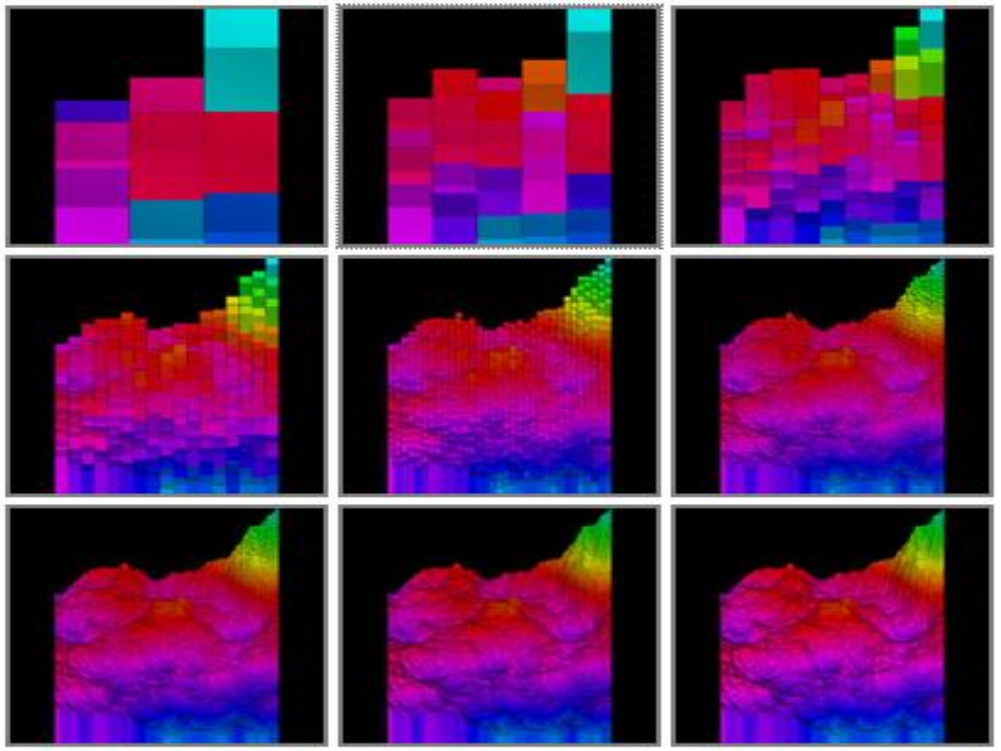
- Creates terrain from 4 seed height values
  - Each corner of the terrain is given a seed height value
- Height of the midpoint between the four points is calculated by averaging the 4 seed values
  - Random value between –d/2 and d/2 is added to the mid point
- Algorithm
  - recursively performs the same process on sub-squares until desired level is reached





Midpoint displacement

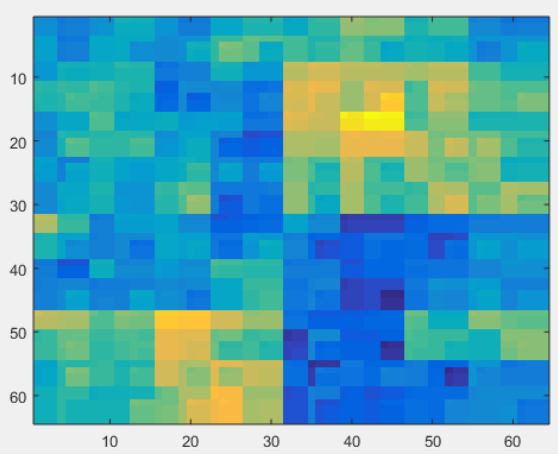
- Example:
  - ➢ Used for 2D games
    - ➢ Dwarf fortress
      - http://www.bay12games.com/dwarves/

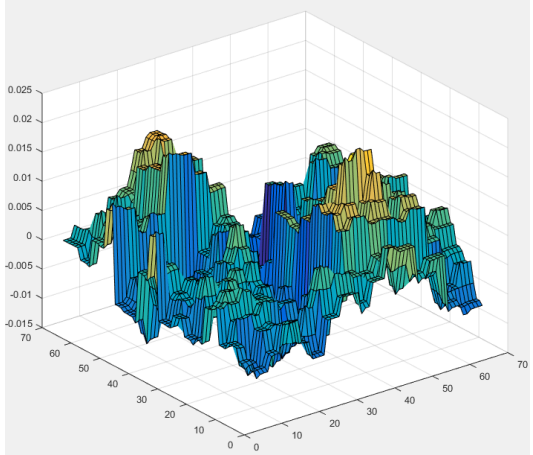


Midpoint displacement

- Algorithm

- Example
  - Map 64x64
  - 2D 3D





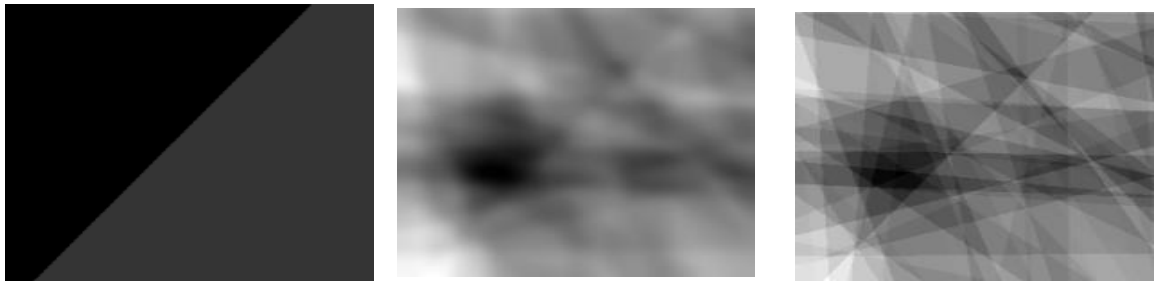

```
function [ x ] = midpoint(x,d,dmax)
% midpoint displacement
if d<dmax
    [n,m]=size(x);
    if n>=2 && m>=2
        midn=floor(n/2);
        midm=floor(m/2);

        v1=x(1,1);
        v2=x(n,1);
        v3=x(1,m);
        v4=x(n,m);
        r=(rand()*2-1)/20; % random value
        v5=(v1+v2+v3+v4)/4+r;

        x(midn,midm)=v5;
        x(midn,1)=(v1+v2)/2;
        x(1,midm)=(v1+v3)/2;
        x(midn,m)=(v3+v4)/2;
        x(n,midm)=(v2+v4)/2;

        x(1:midn,1:midm)=(x(1,1)+x(midn,1)+x(1,midm)+x(midn,midm))/4;
        x(1:midn,midm:m)=(x(1,midm)+x(1,m)+x(midn,midm)+x(midn,m))/4;
        x(midn:n,1:midm)=(x(midn,1)+x(midn,midm)+x(n,1)+x(n,midm))/4;
        x(midn:n,midm:m)=(x(midn,midm)+x(midn,m)+x(n,midm)+x(n,m))/4;


        x(1:midn,1:midm)=midpoint(x(1:midn,1:midm),d+1,dmax);
        x(1:midn,midm:m)=midpoint(x(1:midn,midm:m),d+1,dmax);
        x(midn:n,1:midm)=midpoint(x(midn:n,1:midm),d+1,dmax);
        x(midn:n,midm:m)=midpoint(x(midn:n,midm:m),d+1,dmax);


    end
end
end
```

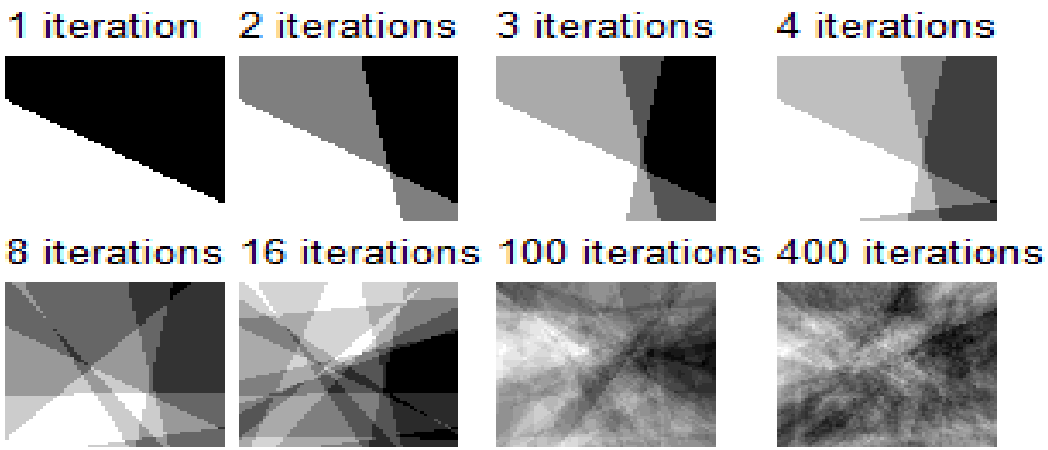
Fault formation

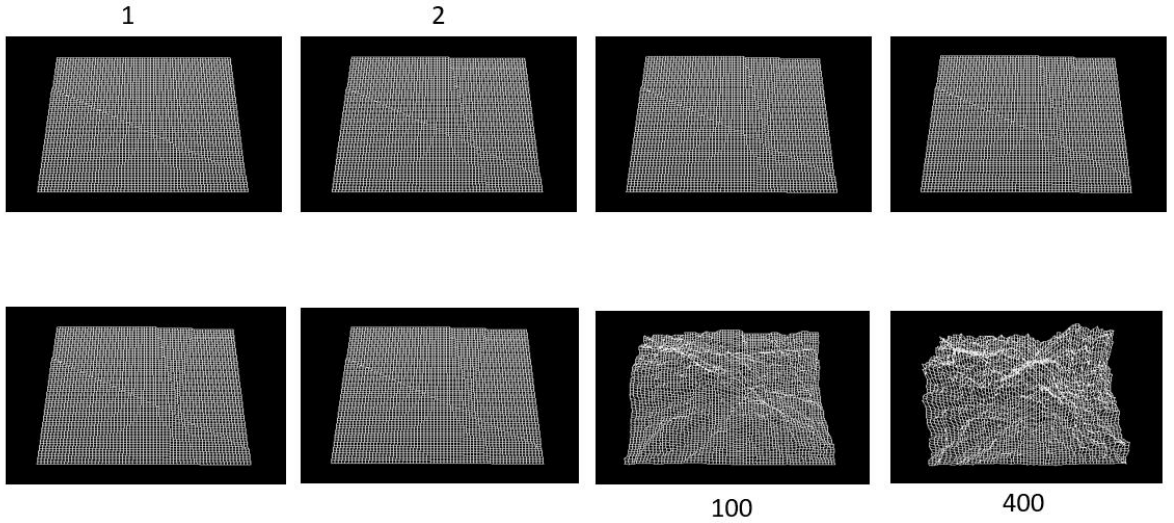
- Terrain grid recursively sub-divided by drawing lines between two random points on the edge of the terrain
  - Vertices to the right – raised by *d*
  - Vertices to the left - lowered by *d*
  - *d* represents the distance between 2 extreme points on the grid
  - Linear decrease in *d* after each pass
    - Should never <= 0!
- The more passes the rougher the generated terrain will appear

Fault formation

- Examples:

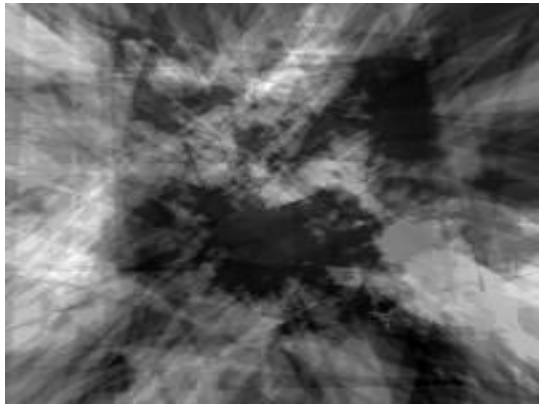


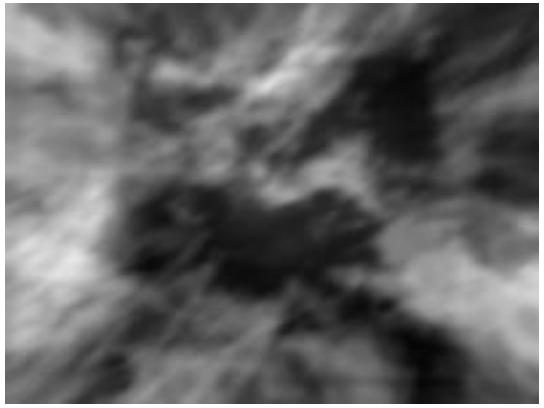


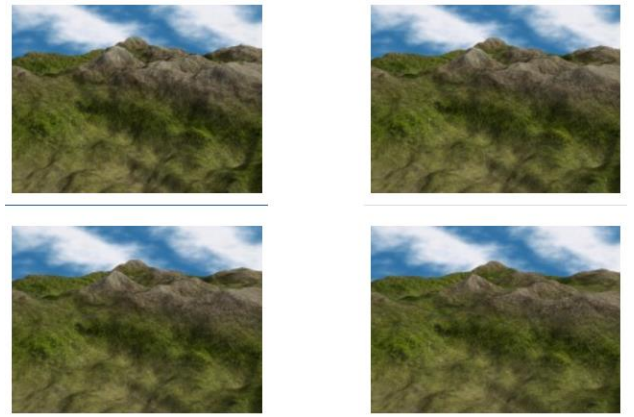


Erosions

- Fault formation
  - For significant differences in neighbouring vertex height values
  - Extreme variations in height between vertices will result in unrealistic results
- Solution:
  - Pass the height value for each vertex through a smoothing filter





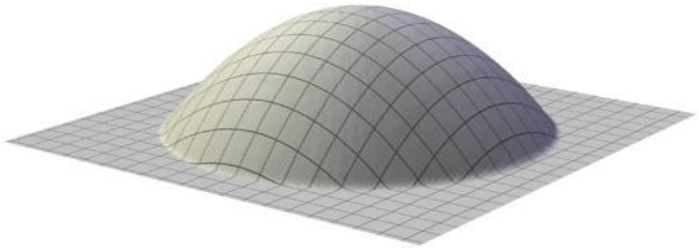


Hill Algorithm

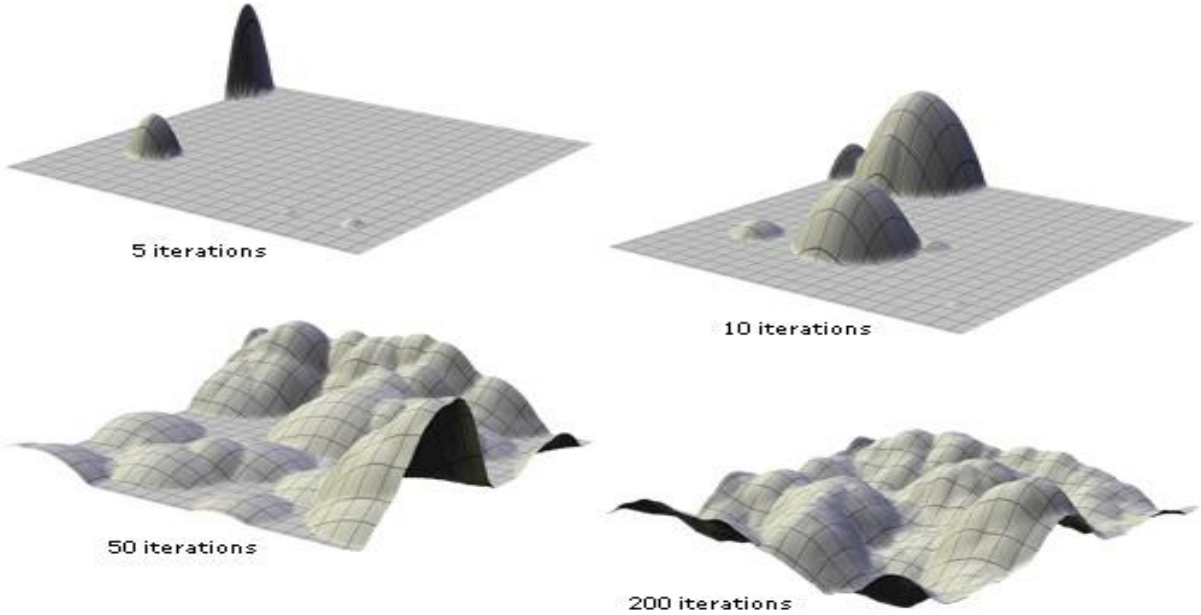
- Start with a flat terrain
  - initialize all height values to 0
- Pick
  - a random point + a random radius between some predetermined min and max
    - Carefully choosing this min and max will make a terrain rough and rocky or smooth and rolling.
- Raise the point a random amount,
  - Points radiating out form the central point of the hill also raised.
- Repeat as many times as required
- Normalize the terrain
- Flatten out the valleys

Hills

- Example:



1 hill



Multiple hills

Conclusion

- Tree Data structures
  - Pandora box
    - BST, AVL, B-trees, 2-3 trees, Red-Black trees…
    - K-D tree, Quadtree, Octree, BSP, …
      - Searches involving a multidimensional search key
      - Binary space partitioning
- Applications
  - From databases, OS, to 3D rendering, pattern recognition, video games…
    - Different types of tree-based data structures
      - What function to stress?
        - Search, Insert,…

Questions ?

- See links on Canvas
- Reading
  - Noor Shaker, Julian Togelius, and Mark J. Nelson (2016). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. ISBN 978-3-319-42714-0.