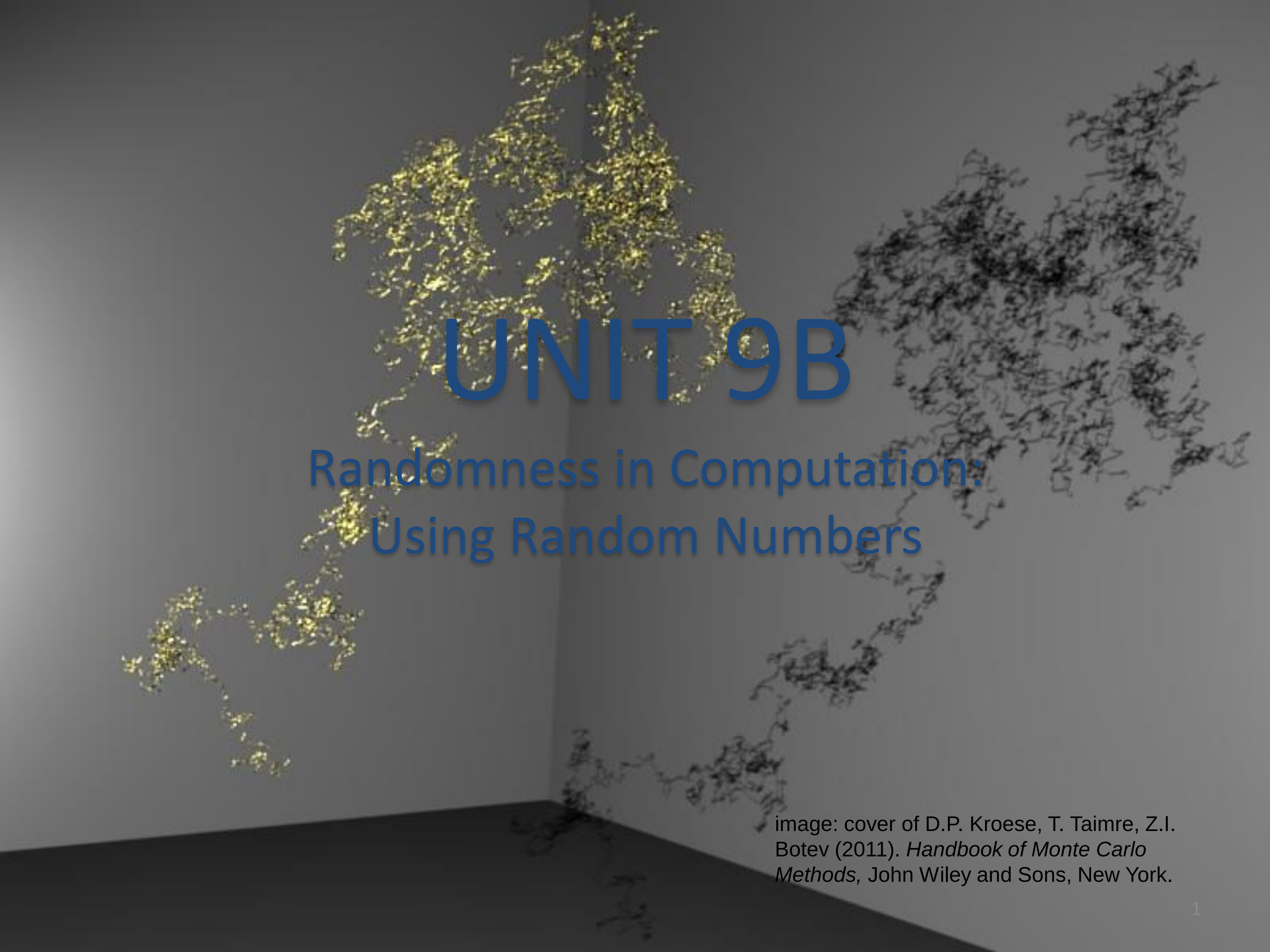


UNIT 9B

Randomness in Computation: Using Random Numbers

image: cover of D.P. Kroese, T. Taimre, Z.I. Botev (2011). *Handbook of Monte Carlo Methods,* John Wiley and Sons, New York.

Last Lecture: Pseudorandom Number Generation

- Linear Congruential Generators (LCGs)
  - We can generate a series of numbers, all different, that ***looks* random** even though it isn't

- If we choose appropriate constants for our LCG, then we can generate a very long sequence before numbers begin to repeat. The length of the sequence is its ***period***

- To generate random numbers in Python we can use randint(x,y), which generates a random integer between x and y.

Uses of Random Numbers

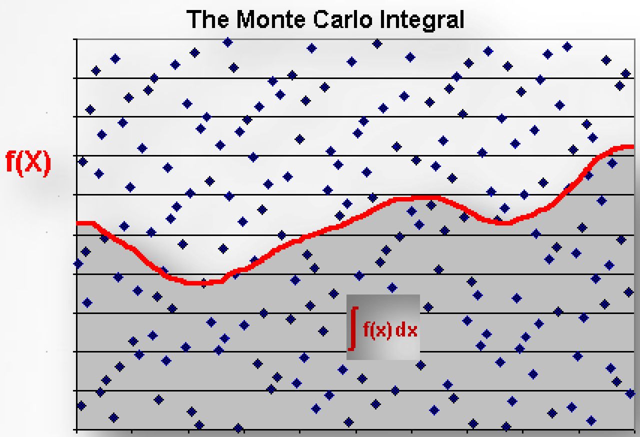
- Many!
  - Cryptography
  - Simulation
  - Gambling
  - Games

- This Lecture -
Monte Carlo methods

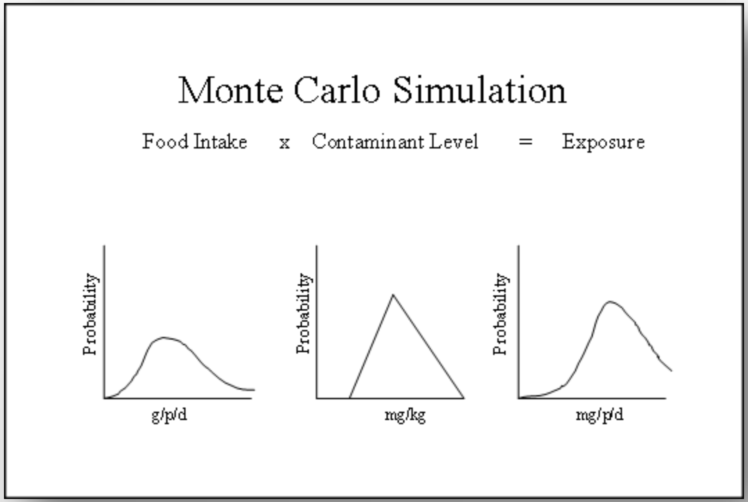
Idea: run many experiments with random inputs to approximate an answer to a question.

We might be unable to answer the question any other way, or an *analytical* (logical, mathematical, exact) solution might be too expensive.

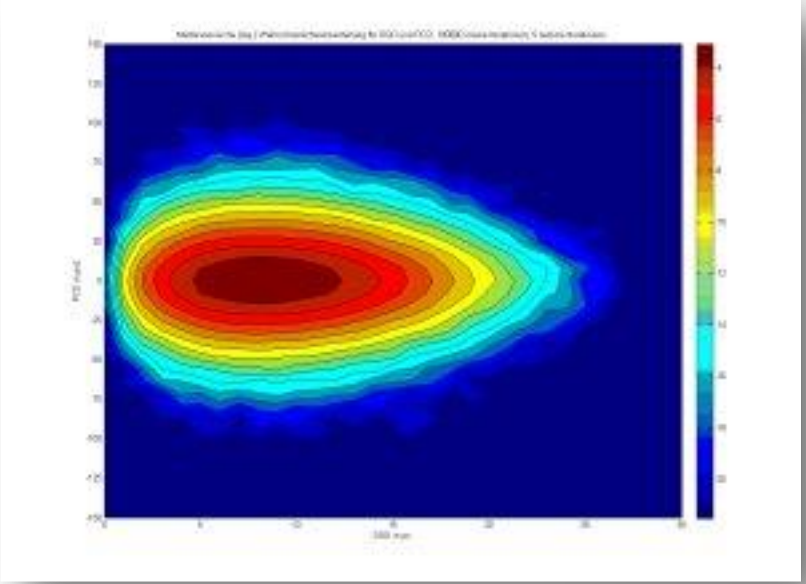
Some Applications



http://marcoagd.usuarios.rdc.puc-rio.br/quasi_mc.html



US Food and Drug Administration



Dr.-Ing. Matthias Westhäuser. Statistical Analysis of Fiber Optical Systems using Multicanonical Monte Carlo Methods (http://www.hft.e-technik.tu-dortmund.de/forschung/projekt.php?id=18&lang=en)

Monte Carlo Methods

- The hungry dice player
- The clueless student*
- The umbrella quandary*
- A survey of applications

* Source: *Digital Dice* by Paul J. Nahin

What is a Monte Carlo Method?

- An algorithm that **uses** a source of (pseudo) **random numbers**
- **Repeats an "experiment"** many times and calculates a statistic, often an average
- **Estimates** a value (often a probability)
- … usually a value that is ***hard* or *impossible*** to calculate analytically

A simple Monte Carlo method

• • •

(no computer needed!)

Simple example: dice statistics

- We can **analyze** throwing a pair of dice and get the following probabilities for the sum of the two dice:

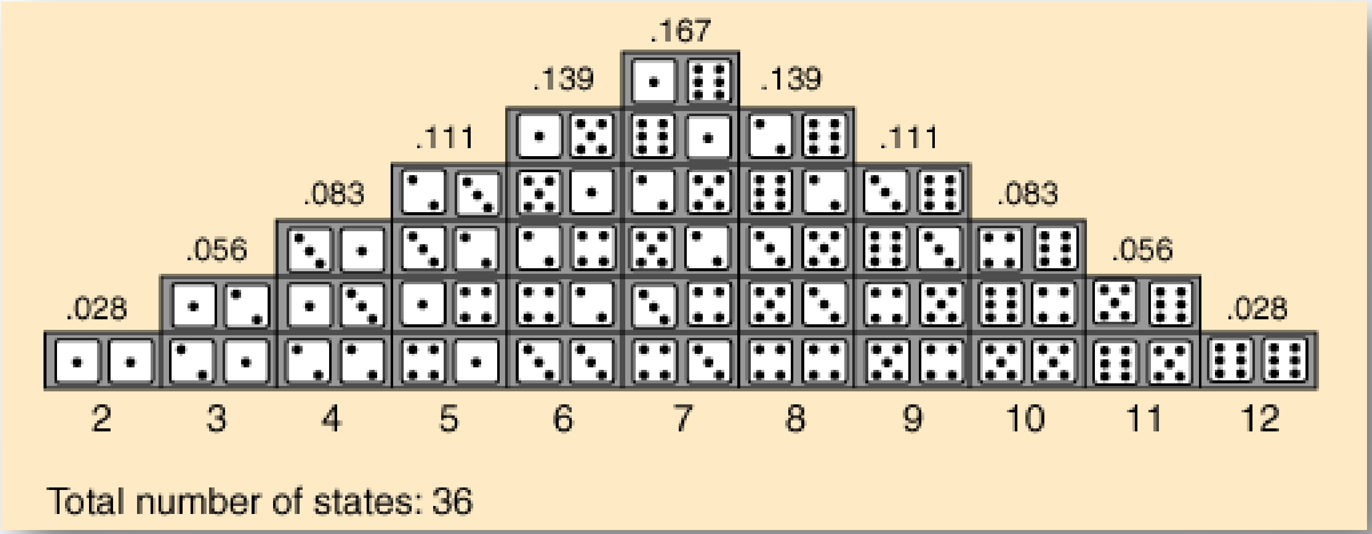
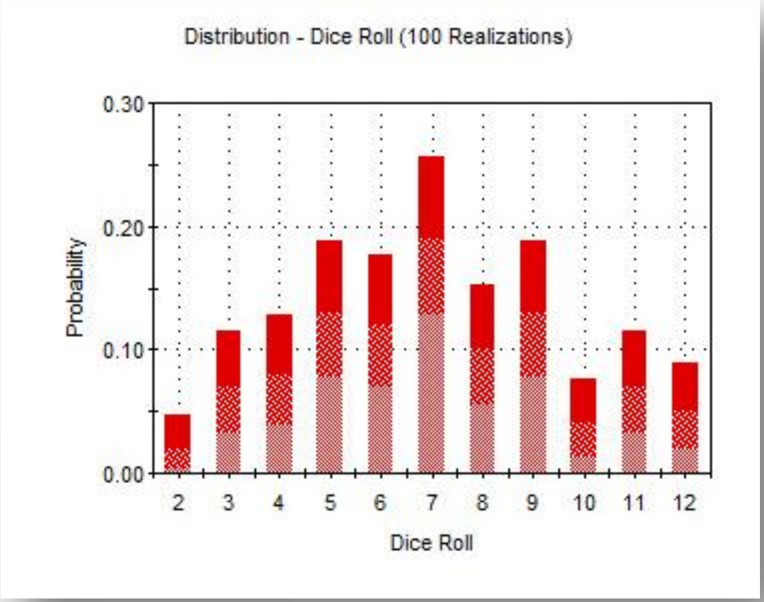


image source: http://hyperphysics.phy-astr.gsu.edu/hbase/math/dice.html
via http://www.goldsim.com/Web/Introduction/Probabilistic/MonteCarlo/

Simple example: dice statistics

- … **or** we can throw a pair of dice 100 times and record what happens, or 10000 times for a more accurate estimate.



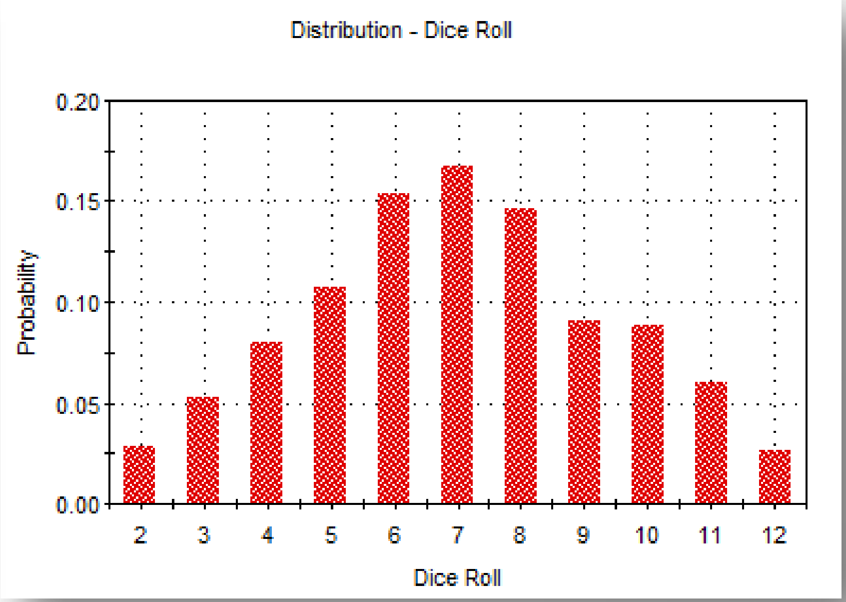


image source: http://www.goldsim.com/Web/Introduction/Probabilistic/MonteCarlo/

The Hungry Dice Player

estimating the expected value of a simple game

A game of dice

```
def dice_game() :
    strikes = 0
    winnings = 0
    while strikes < 3 :   # 3 strikes and you're out
        # get 2 random numbers (1…6)
        die1 = roll()
        die2 = roll()
        # play: strike or win?
        if die1 == die2 :
            strikes = strikes + 1
        else :
            winnings = winnings + die1 + die2
    return winnings       # in centsz
```

The Hungry Dice Player

- In our simple game of dice:
  Can I expect to make enough money playing it to buy lunch?
- That is, what is the expected (average) value won in the game?
- We could figure it out by applying laws of probability
- …or use a Monte Carlo method

Monte Carlo method for the hungry dice player

```
def average_winnings(runs) :
    # runs is the number of experiments to run
    total = 0
    for n in range(runs) :
        total = total + dice_game()
    return total/runs
```

```
>>> [round(average_winnings(10),2) for i in range(5)]
[85.8, 94.8, 120.7, 123.3, 90.0]

>>> [round(average_winnings(100),2) for i in range(5)]
[105.97, 102.95, 107.74, 134.4, 114.54]

>>> [round(average_winnings(1000),2) for i in range(5)]
[106.84, 107.11, 105.59, 104.28, 106.41]

>>> [round(average_winnings(10000),2) for i in range(5)]
[104.94, 105.71, 105.81, 105.74, 104.62]
```

The Clueless Student

a famous matching problem

The Clueless Student

A clueless student faced a pop quiz:

a list of the 24 Presidents of the 19^{th} century and
another list of their terms in office, but scrambled.
The object was to match the President with the term.

If the student guesses a random one-to-one matching,
how many matches will be right out of the 24, on average?

The quiz

1. Monroe	a. 1801-1809
2. Jackson	b. 1869-1877
3. Arthur	c. 1885-1889
4. Madison	d. 1850-1853
5. Cleveland	e. 1889-1893
6. Jefferson	f. 1845-1849
7. Lincoln	g. 1837-1841
8. Van Buren	h. 1853-1857
9. Adams	i. 1809-1817
etc.	etc.

Solving the problem

- The problem (1710, Pierre de Montmort) was important in development of probability theory

- The mathematical analysis is, um, interesting (see http://www.math.uah.edu/stat/urn/Matching.html)

- Let's just simulate the situation, randomly selecting guesses and checking to see how many correct match-ups they contain.

Representing a guess

values

0	1	2	3	4	5	6	7	8	...
1801-09	1809-17	1817-25	1825-29	1829-37	1837-41	1841-41	1841-45	1845-49	...

6	0	2	3	1	4	5	7	8	...

0	1	2	3	4	5	6	7	8	...
Jefferson	Madison	Monroe	Adams	Jackson	Van Buren	Harrison	Tyler	Polk	...

indexes

Representing a guess

- **Representing a guess:** A guess is just a permutation (shuffling) of the numbers 0 … 23.

  E.g. [0, 1, 2, 3, 4, 5, …, 23] represents a completely correct guess
  [1, 0, 2, 3, 4, 5, …, 23] represents a guess that is correct except that it gets the first two presidents wrong.

- Let's define a ***match*** in a guess to be any number *k* that occurs in position *k*. (*E.g.,* 0 in position 0, 10 in position 10)

- With this representation, our question becomes:

  if I pick a random shuffling of the numbers 0…23,
  how many (on average) matches occur?

Randomly permuting a list

To get a random shuffling of the numbers 0 to 23 we use the `shuffle` function from module `random`:

```
>>> nums = list(range(10))
>>> nums
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> shuffle(nums)
>>> nums
[4, 5, 3, 2, 0, 9, 6, 1, 8, 7]
```

```
>>> shuffle(nums)
>>> nums
[3, 6, 1, 4, 5, 8, 2, 9, 0, 7]
```

Algorithm

We will solve a more general problem

- **Input:** *pairs* (number of things to be matched),
  samples (number of samples to test)
- **Output:** average number of correct matches per sample
- **Method:**

1. Set *num_correct* = 0
2. Do the following *samples* times:

    a. Set *matching* to a random permutation of the numbers 0…*pairs*-1

    b. For ***k*** in 0…*pairs*, if *matching*[***k***] = ***k*** add one to *num_correct*

3. The result is *num_correct* / *samples*

Code for the clueless student

```
from random import shuffle
# pairs is the number of pairs to be guessed
# samples is the number of samples to take
def cl_student(pairs, samples) :
    num_correct = 0
    matching = list(range(pairs))
    for i in range(samples):            # experiment samples times
        shuffle(matching)               # generate a guess
        # if generated guess is a match, increment the num_correct
        for j in range(pairs):
            if matching[j] == j :
                num_correct = num_correct + 1
    # return the average value
    return num_correct / samples
```

Running the code

The mathematical analysis says the expected value is exactly 1 (**no matter how many matches are to be guessed**).

```
>>> cl_student(24, 10000)
0.9924

>>> cl_student(24, 10000)
1.0071

>>> cl_student(10, 10000)
1.0224
```

```
>>> cl_student(10, 10000)
0.9999

>>> cl_student(5, 10000)
1.0039

>>> cl_student(5, 10000)
0.9826
```

More samples – smaller error

```
>>> 1 - cl_student(5, 1000)
0.036000000000000003

>>> 1 - cl_student(5, 10000)
0.0059000000000000016

>>> 1 - cl_student(5, 100000)
0.0014100000000000223

>>> 1 - cl_student(5, 1000000)
-0.00066799999999998909
```

The Umbrella Quandary

simulating a system

The Umbrella Quandary

- Mr. X walks between home and work every day
- He likes to keep an umbrella at each location
- But he always forgets to carry one if it's not raining

- If the probability of rain is *p*, how many trips can he expect to make before he gets caught in the rain?

  (Assuming that if it's not raining when he starts a trip, it doesn't rain during the trip.)

The trivial cases

- What if it always rains?
- What if it never rains (ok, that was too easy)
- So we only need to think about a probability of rain greater than zero and less than one

Solving the umbrella quandary

- Analysis of the problem can be done with Markov chains

- But we’re just humble programmers,

  → we’ll simulate and measure

Simulating an event with a given probability

- In contrast to the clueless student problem we're given a probability of an event
- We want to simulate that the event happens, with the given **probability *p*** (where *p* is a number between 0 and 1)
- **Technique:** get a random float between 0 and 1;
  if it's less than *p* simulate that the event happened

```
if random() < p :
        raining = True
```

Representing home, work, and umbrellas

- Use **0** for **home**,
  1 for **work** as **location**
- A list for the **number of umbrellas** at each location (2 locations)
- How should we initialize?

```
location = 0
umbrellas = [1, 1]
```

Remember that he likes to keep an umbrella at each location

Figuring out when to stop

We want to count the number of trips before Mr. X gets wet, so we want to keep simulating trips until he does.

- To keep track:

```
wet = False
trips = 0
while (not wet):
     …
```

Changing locations

Mr. X walks between home (0) and work (1)

- To keep track of where he is:

```
location = 0    #start at home
```

- To move to the other location:

```
location = 1 – location
```

- To find how many umbrellas at current location:

```
umbrellas[location]
```

Putting it together

```
from random import random
def umbrella(p) :            # p is the probability of rain

    wet = False
    trips = 0
    location = 0
    umbrellas = [1, 1]       # index 0 stands for home, 1 stands for work

    while (not wet) :
        if random() < p :    # it's raining

            if umbrellas[location] == 0 :  # no umbrella at current loc.
                wet = True
            else:
                trips = trips + 1
                umbrellas[location] -= 1           # take an umbrella
                location = 1 - location            # switch locations
                umbrellas[location] += 1           # put umbrella

        else:     # it's not raining, leave umbrellas where they are
            trips = trips + 1
            location = 1 – location                # switch locations

    return trips
```

Running simulations

```
>>> umbrella(.5)
22
>>> umbrella(.5)
4
>>> umbrella(.5)
13
>>> umbrella(.5)
2
>>> umbrella(.5)
2
```

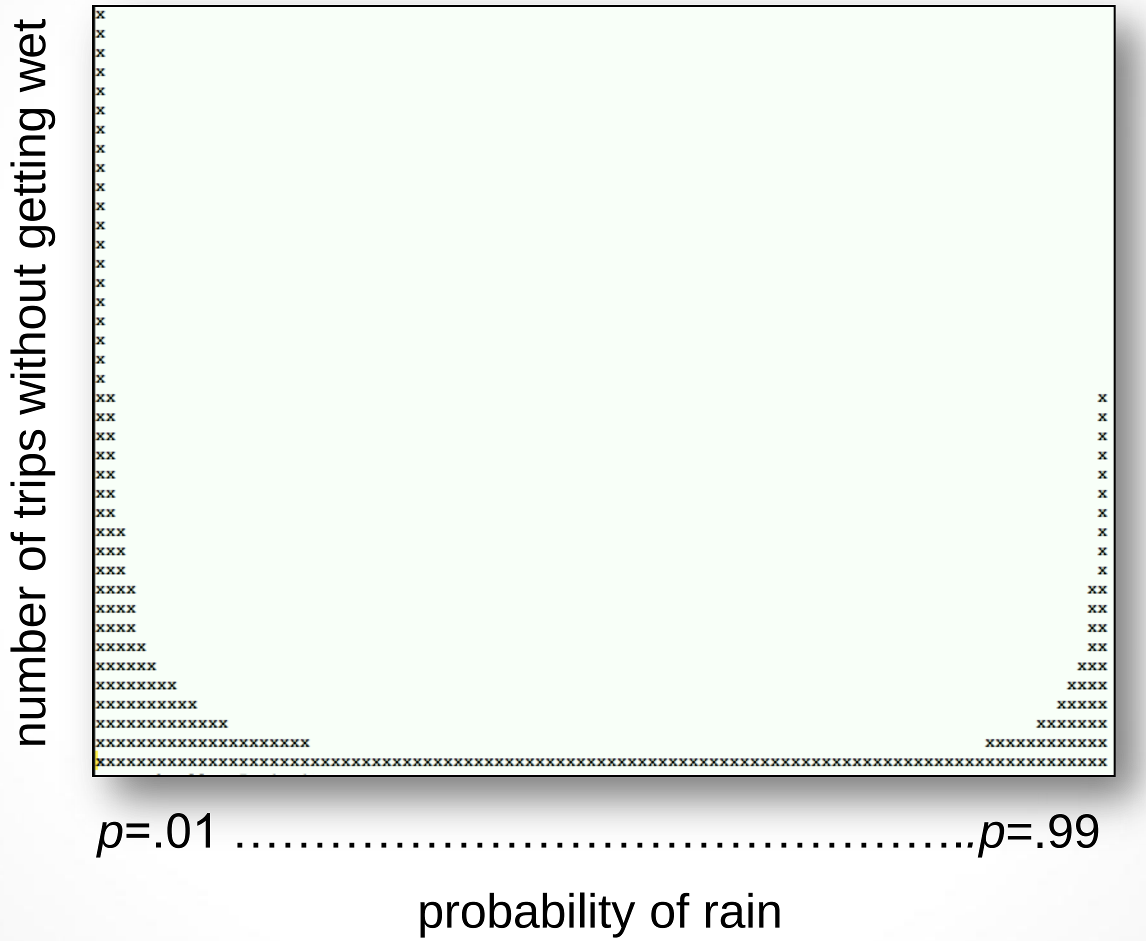
Great, but we want averages

- One experiment doesn't tell us much–we want to know, **on average**, if the probability of rain is *p*, how many trips can Mr. X make without getting wet?
- We add code to run `umbrella(p)` 10,000 times for different probabilities of rain, from $p = .01$ to .99 in increments of .01
- We accumulate the results in a list that will show us how the average number of trips is related to the probability of rain.

Running the experiments

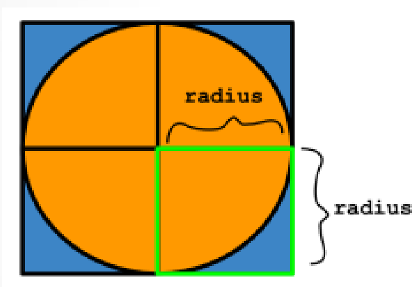
```
# 10,000 experiments for each probability from .01 to .99
# Accumulate averages in a list

def test() :
    results = [None]*99          # Initialize list
    p = .01                      # probability starts 0.1

    for i in range(99) :
        trips = 0

        # find average of 10000 experiments
        for k in range(10000) :
            trips = trips + umbrella(p)
        results[i] = trips/10000

        p = p + .01              # inc. for next probability

    return results
```

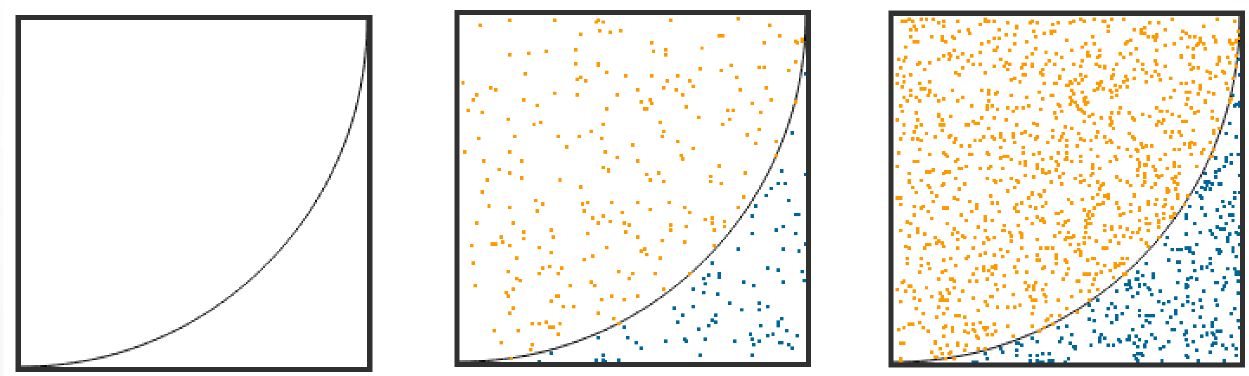
Crude plot of results



Other Samples and Uses

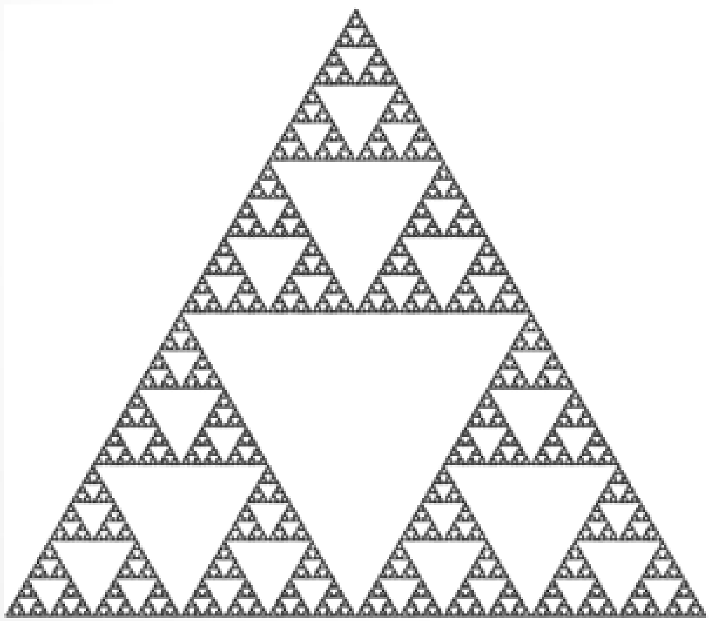
- Calculation of PI using the Monte Carlo Method (http://pumpkinprogrammer.com/2015/03/14/approximating-pi-monte-carlo-method/)

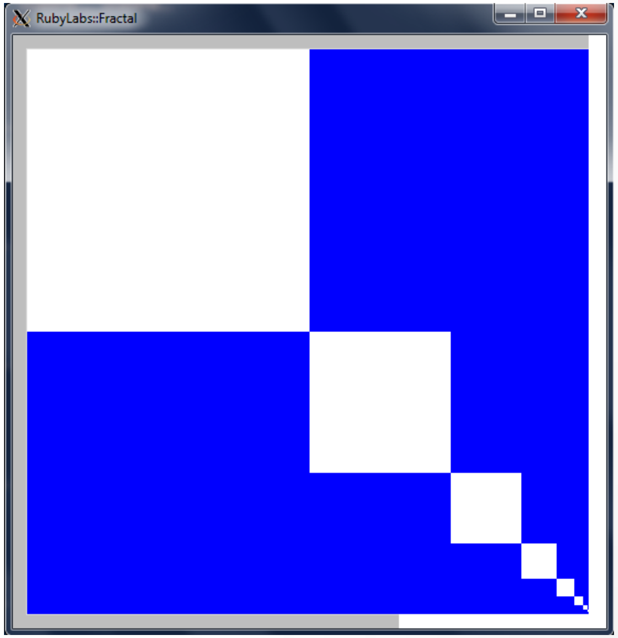




Fractals and Randomness

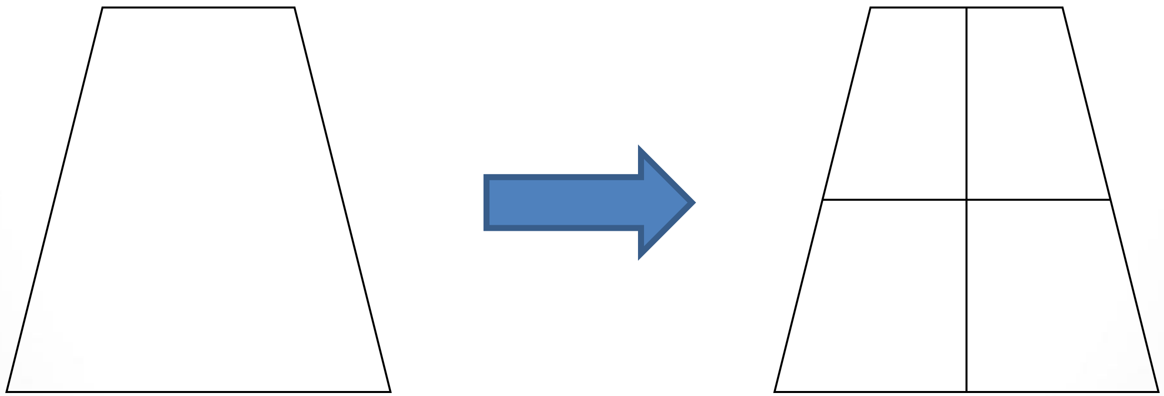
- Recall: A fractal is an image that is self-similar.
- Fractals are typically generated using recursion.





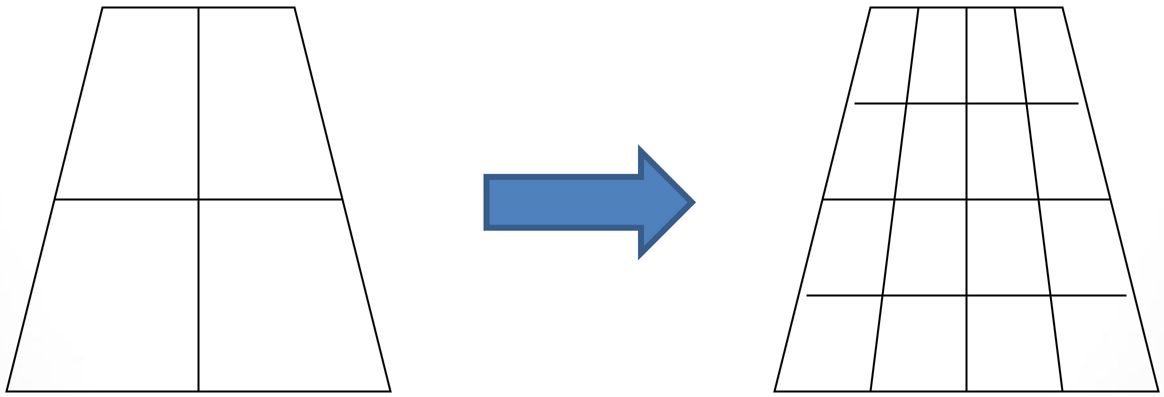
Simple Fractal

Connect midpoints of the quadrilateral



Simple Fractal

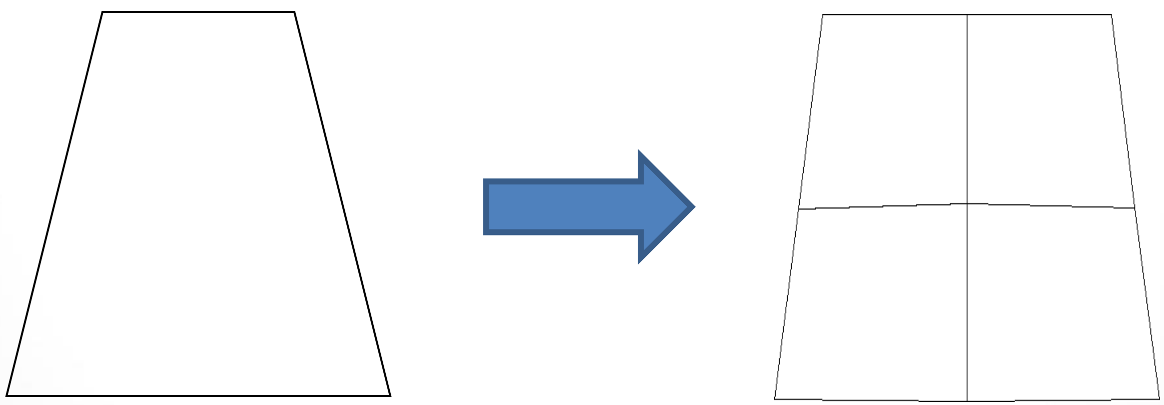
- Connect midpoints of each quadrilateral recursively



It makes a disco floor!

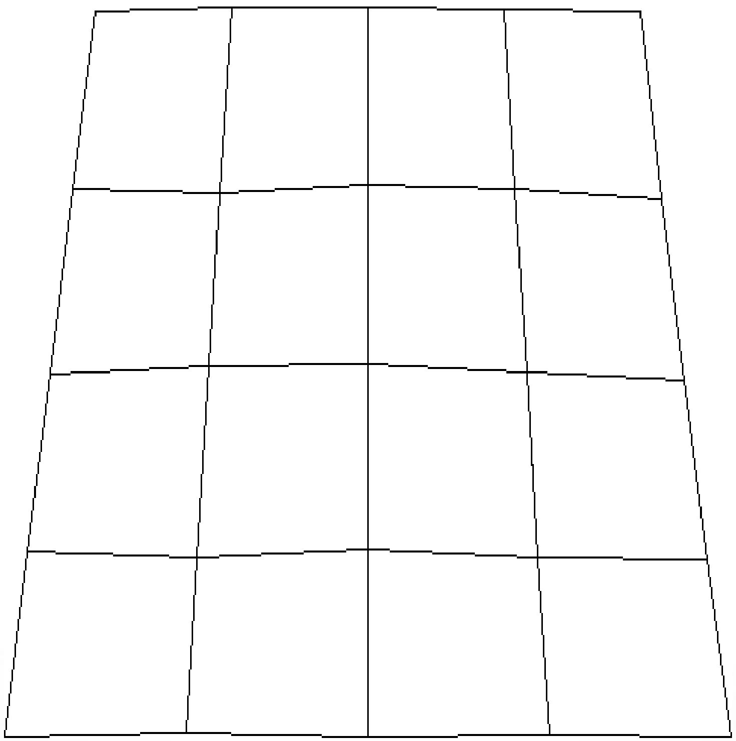
Fractal with Randomness

- Randomly move midpoints slightly and then connect.



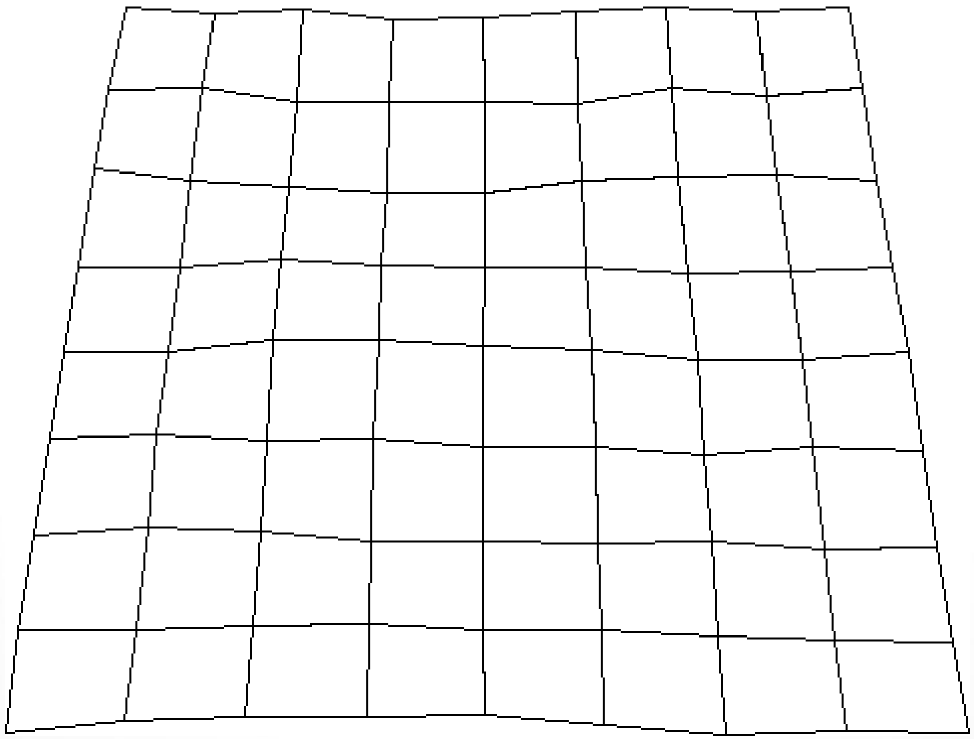
Fractal with Randomness

- Randomly move midpoints slightly and then connect.



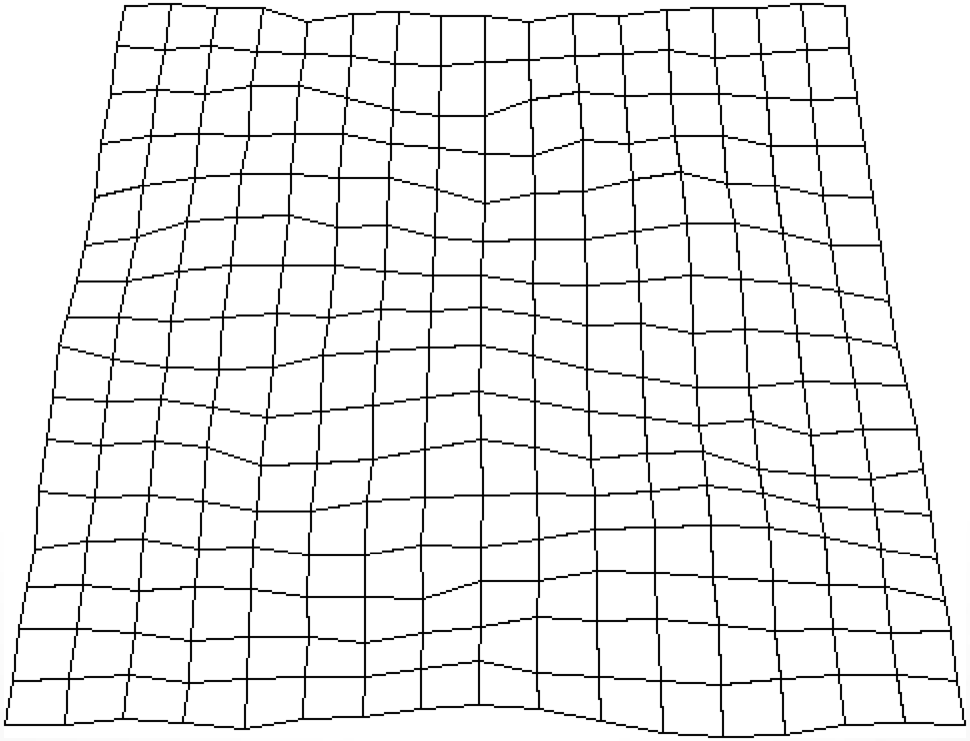
Fractal with Randomness

- Randomly move midpoints slightly and then connect.



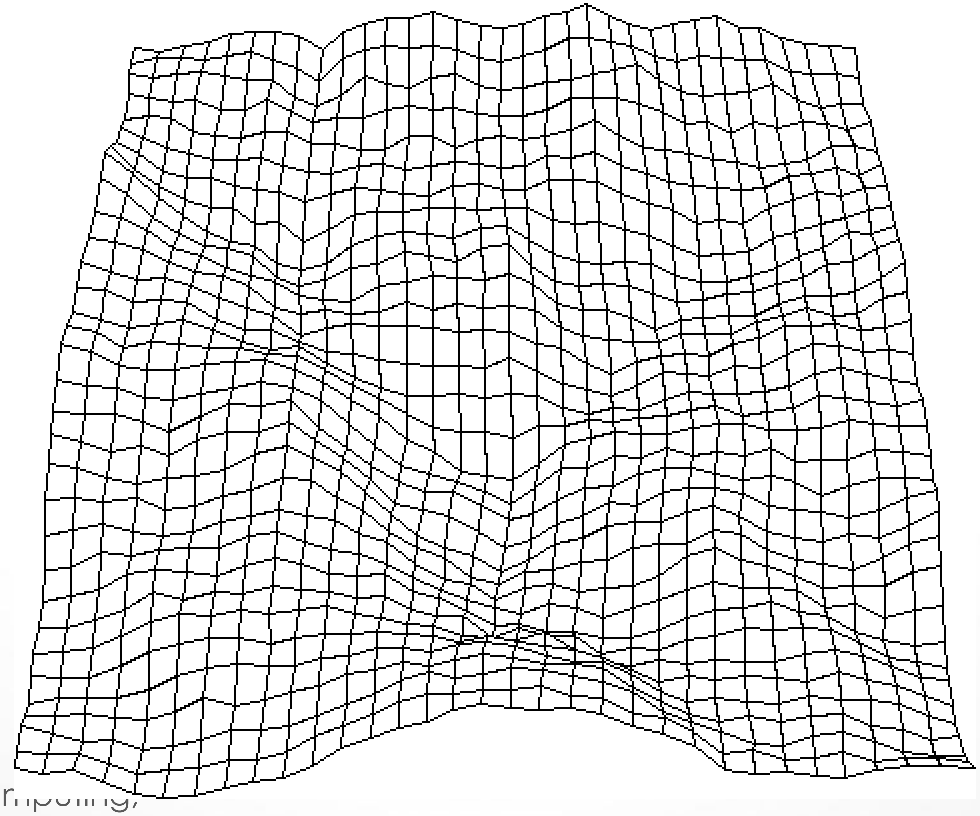
Fractal with Randomness

- Randomly move midpoints slightly and then connect.



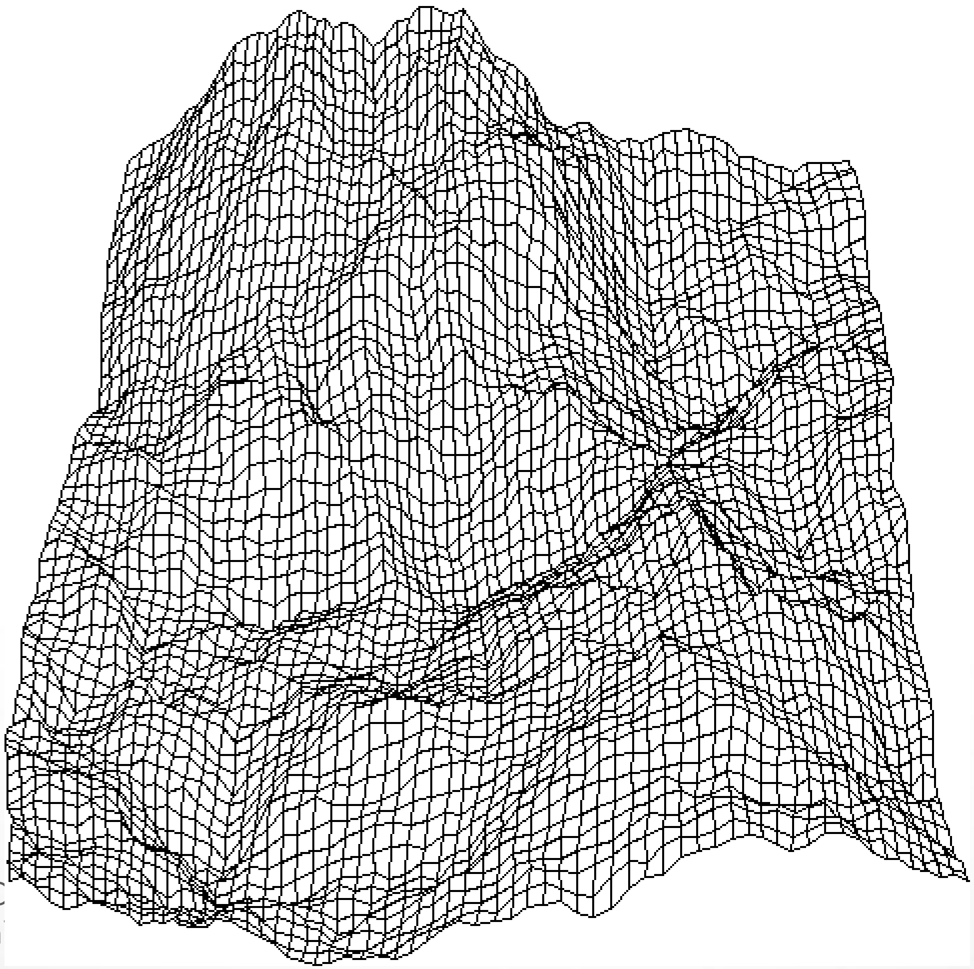
Fractal with Randomness

- Randomly move midpoints slightly and then connect.



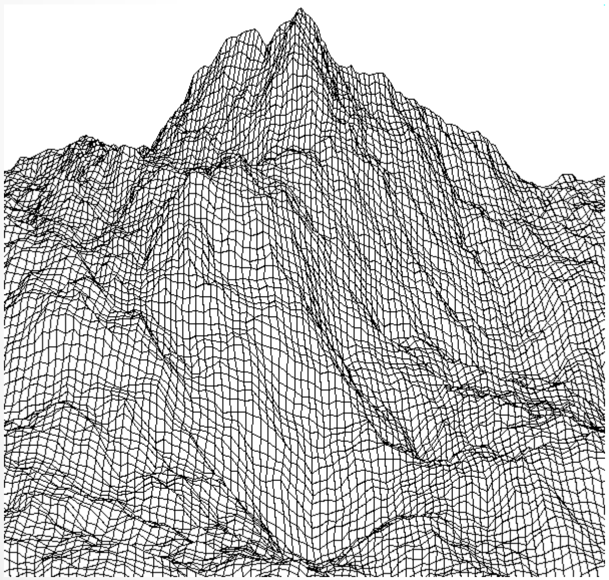
Fractal with Randomness

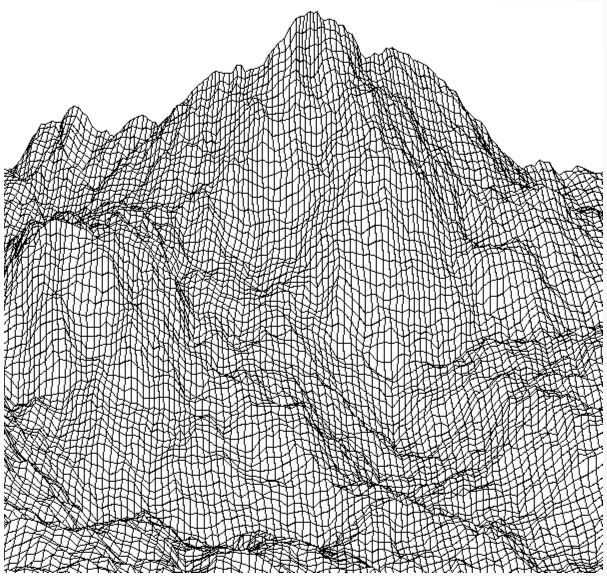
- Randomly move midpoints slightly and then connect.



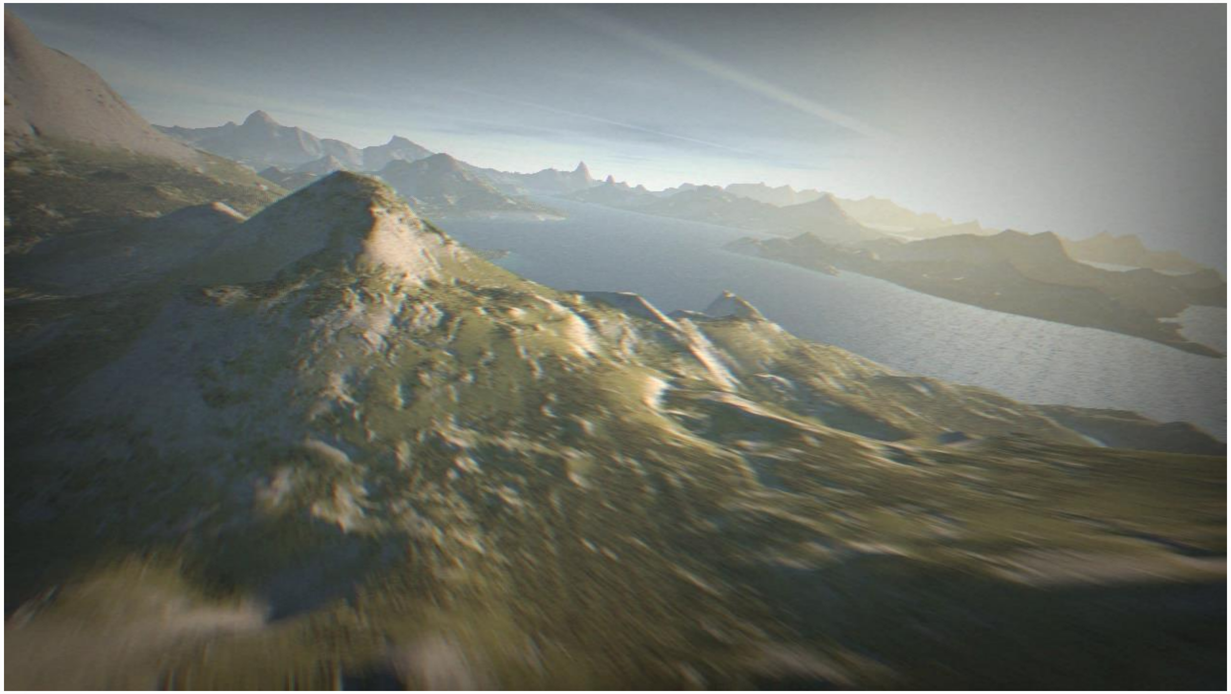
Fractal with Randomness

- This technique can be used to create some realistic mountain ranges.





“Elevated”



- Was produced from somebody's 4 kilobyte computer program.

Applications

• • •

many, many, many

Finance

- Investment portfolio analysis
- Stock option analysis
- Personal financial planning

Engineering

- Reliability engineering
- Wireless network design
- Wind farm yield prediction
- Fluid dynamics
- Robotics

Mathematics and physics

- Multi-dimensional partial differentiation and integration
- Optimization
- Simulating quantum systems (pioneered by Fermi in 1930)

Many others

- Computational biology
- Physical chemistry
- Applied statistics where data distributions are difficult to analyze
- Game playing

Graphics: path tracing

image: http://www.graphics.cornell.edu/~eric/thesis/images.html

image: http://2.bp.blogspot.com/-cUQu1ym3krA/UPYw6qhsZPI/AAAAAAAADeU/YnqtyJjBJJc/s1600/cubecity9b.png

Summary

- Monte Carlo methods use random number generator to “run experiments” in software

- Operations we used:
    - get random integer in a given range
    - get a random permutation of a list
    - use random float between 0 and 1 to decide if an event with probability `p` happens

```
if random() < p : # event happened
```

Next time: Simulation

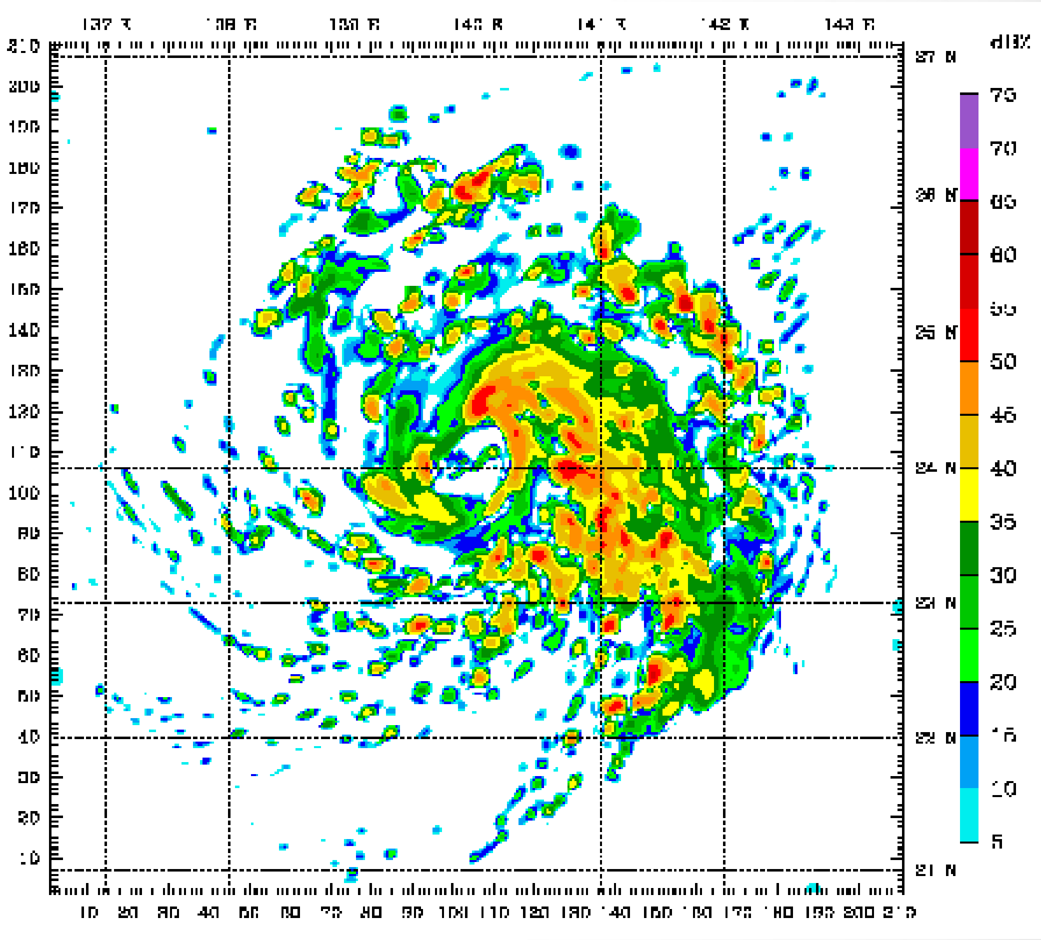


Image: Wikipedia