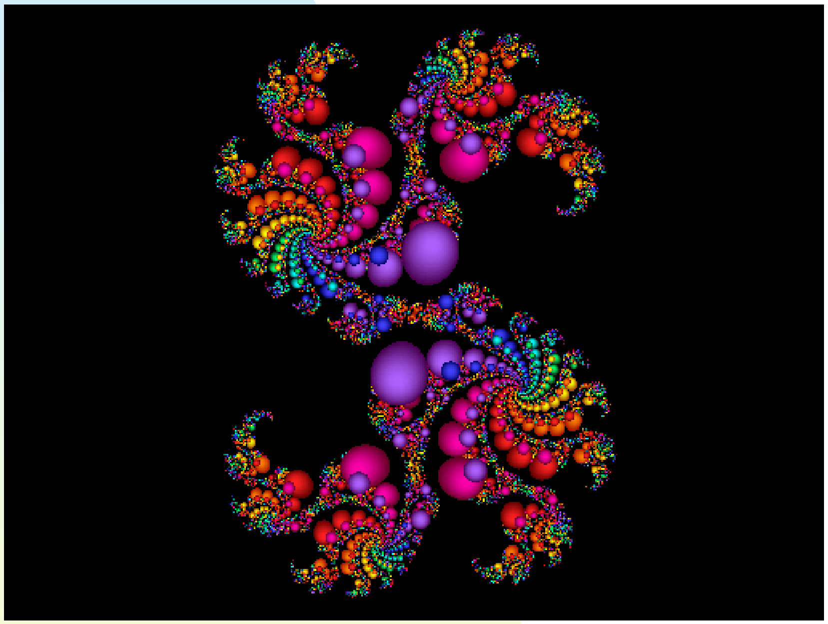
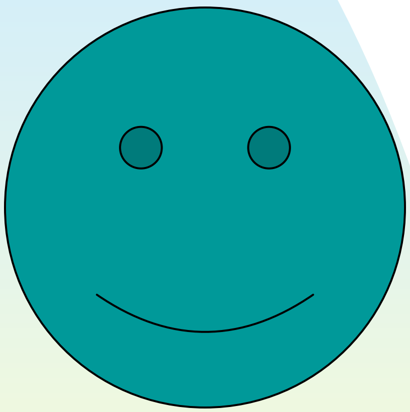
Session overview



- Iterated function systems
- Announcements:
  - Pass in project 1 paper now
  - Commit Sierpinski to your SVN repository so we can run it.
  - Homework 1 due Thurs.



Multiple Reduction Copy Machine (MRCM)

- Based on a collection of contractions
- The reduction lenses are similarity transformations, which preserve angles
- Note that more general transformations may use reductions of different amounts in different directions
- Demo

Linear transformations

- Reduction transformations apply a scaling
- When combined with shearing and/or rotation and/or reflection we are said to be applying a linear transformation (or mapping)

Linear mappings

- A linear mapping, F, is a transformation which associates with every point P in the plane a point $F(P)$ such that
  - $F(P_1 + P_2) = F(P_1) + F(P_2)$ for all points P_1 and P_2
  - $F(sP) = sF(P)$ for any real number s and all points P

Matrix representation

- A linear mapping F can be represented by a matrix

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

- If $P = (x, y)$ and $F(P) = (u, v)$ then
  - $u = ax + by$
  - $v = cx + dy$

Affine linear transformations

- Affine linear transformations are simply the composition of a linear mapping together with a translation
- If F is linear and Q is a point then $w(P) = F(P) + Q$ is said to be ***affine linear***
- Allow us to describe contractions which involve positioning in the plane

Matrix representation

- Since *F* is given by a matrix and *Q* is given by a pair of coordinates, say (*e f*), an affine linear transformation is given by six numbers, represented in an augmented matrix

$$\left(\begin{array}{cc|c} a & b & e \\ c & d & f \end{array}\right)$$

- If $P = (x, y)$ and $w(P) = (u, v)$ then
  - $u = ax + by + e$
  - $v = cx + dy + f$

Modeling the MRCM

- Define each lens of a MRCM by an affine linear transformation
- If there are n lenses, there are n affine transformations: w_1, w_2, … w_n
- For a given initial image A small affine copies $w_1(A)$, $w_2(A)$, …, $w_n(A)$ are produced

Hutchinson operator

- The MRCM overlays all these copies into one new image, the output $W(A)$ of the machine:

$$W(A) = w_1(A) \cup w_2(A) \cup \ldots \cup w_n(A)$$

- We call W the Hutchinson operator, after the Australian mathematician
- Running the MRCM in feedback mode corresponds to iterating the operator W

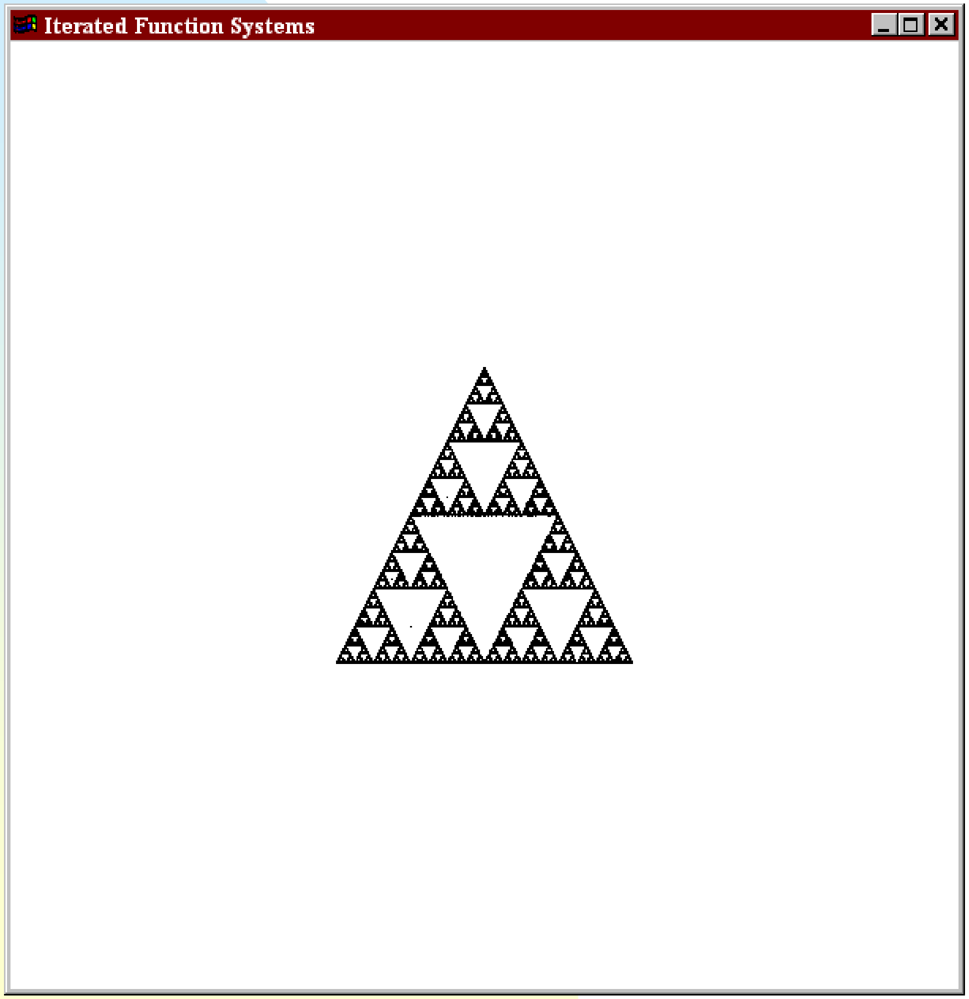
Iterated function systems

- Iterating the Hutchinson operator is the essence of an ***iterated function system (IFS)***
- An IFS generates a sequence which tends towards a final image A_∞, which we call the attractor of the IFS
- A_∞ is left invariant by the IFS, which means $W(A_\infty) = A_\infty$

Example program

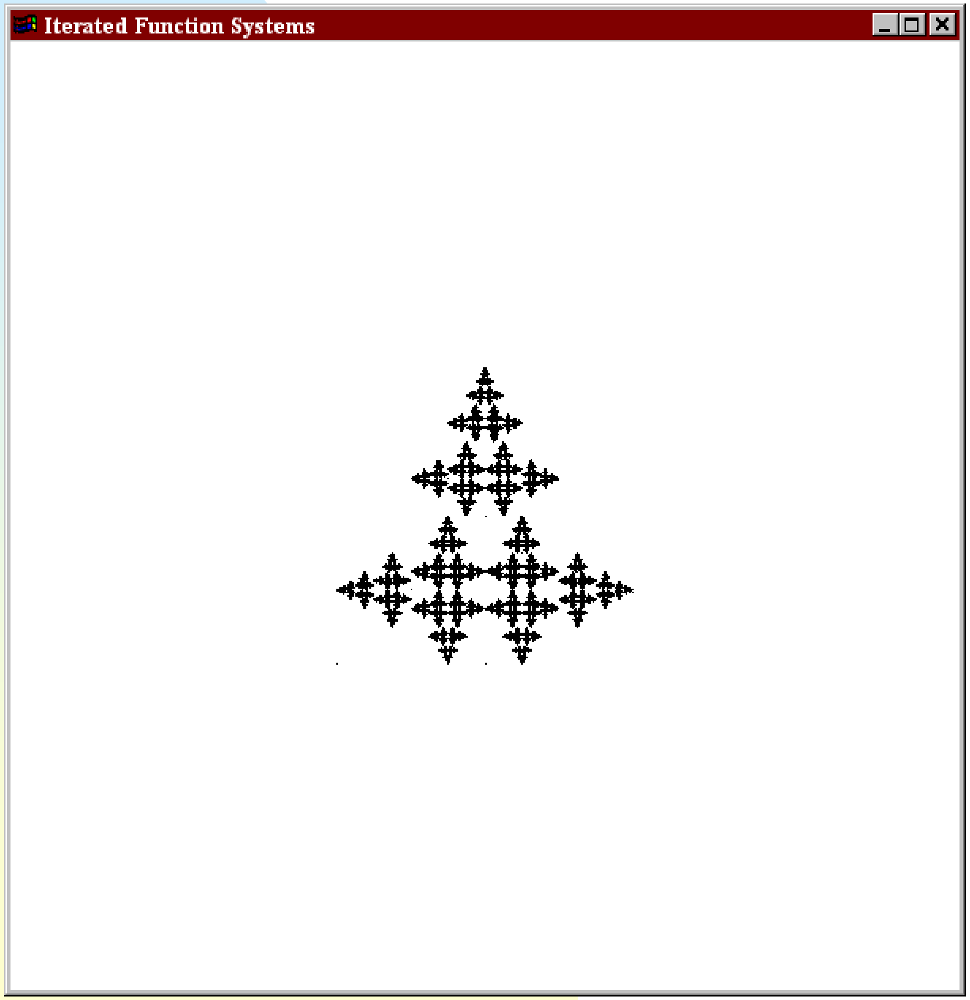
- Program `ifs.cpp` generates fractal images via iterated function systems
- Values for a, b, c, d, e, and f for several images in your text can be found online on the course ANGEL website
- Some are repeated on the following slides with images
- Upload it and play with it to get the shapes on the following slides.

Sierpinski triangle



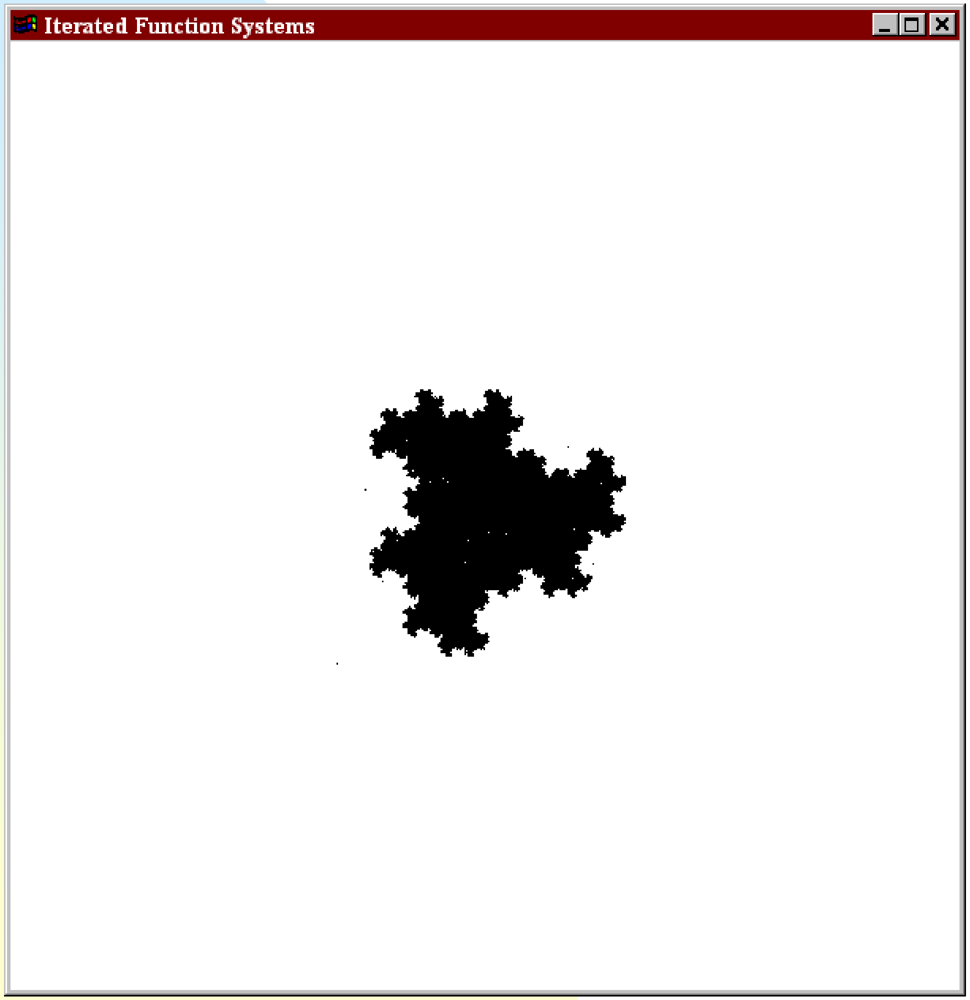
- #1: 0.5, 0.0, 0.0, 0.5, 0.0, 0.0
- #2: 0.5, 0.0, 0.0, 0.5, 0.5, 0.0
- #3: 0.5, 0.0, 0.0, 0.5, 0.25, 0.5
- n = 100,000

Twin Christmas tree



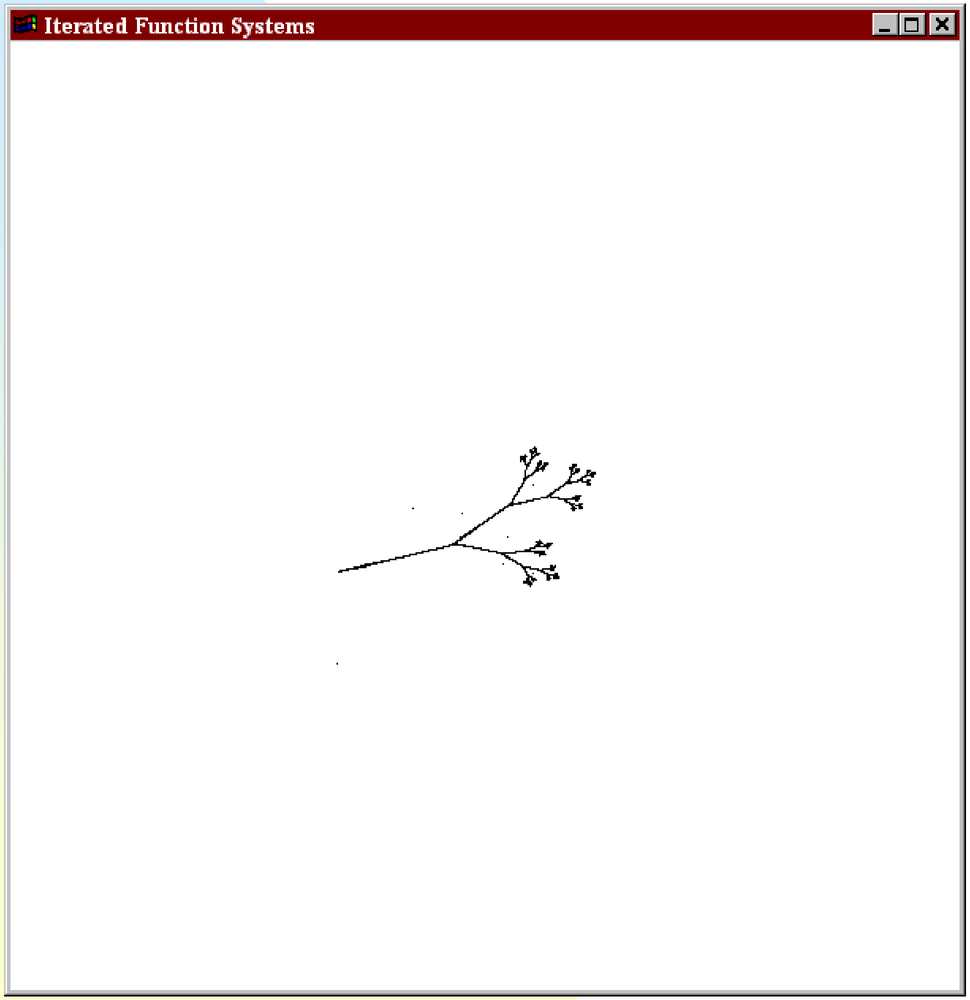
- #1: 0.0, -0.5, 0.5, 0.0, 0.5, 0.0
- #2: 0.0, 0.5, -0.5, 0.0, 0.5, 0.5
- #3: 0.5, 0.0, 0.0, 0.5, 0.25, 0.5
- n = 100,000

Dragon with threefold symmetry



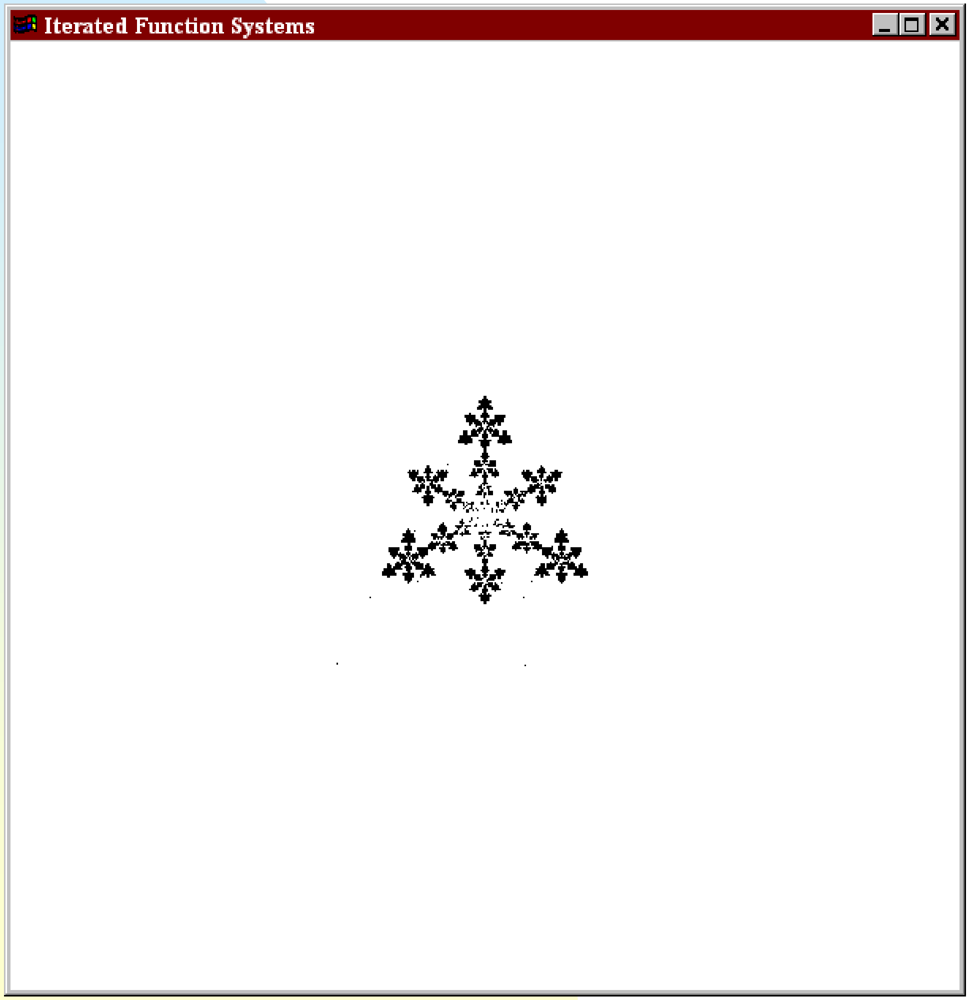
- #1: 0.0, 0.577, -0.577, 0.0, 0.0951, 0.5893
- #2: 0.0, 0.577, -0.577, 0.0, 0.4413, 0.7893
- #3: 0.0, 0.577, -0.577, 0.0, 0.0952, 0.9893
- n = 100,000

Twig



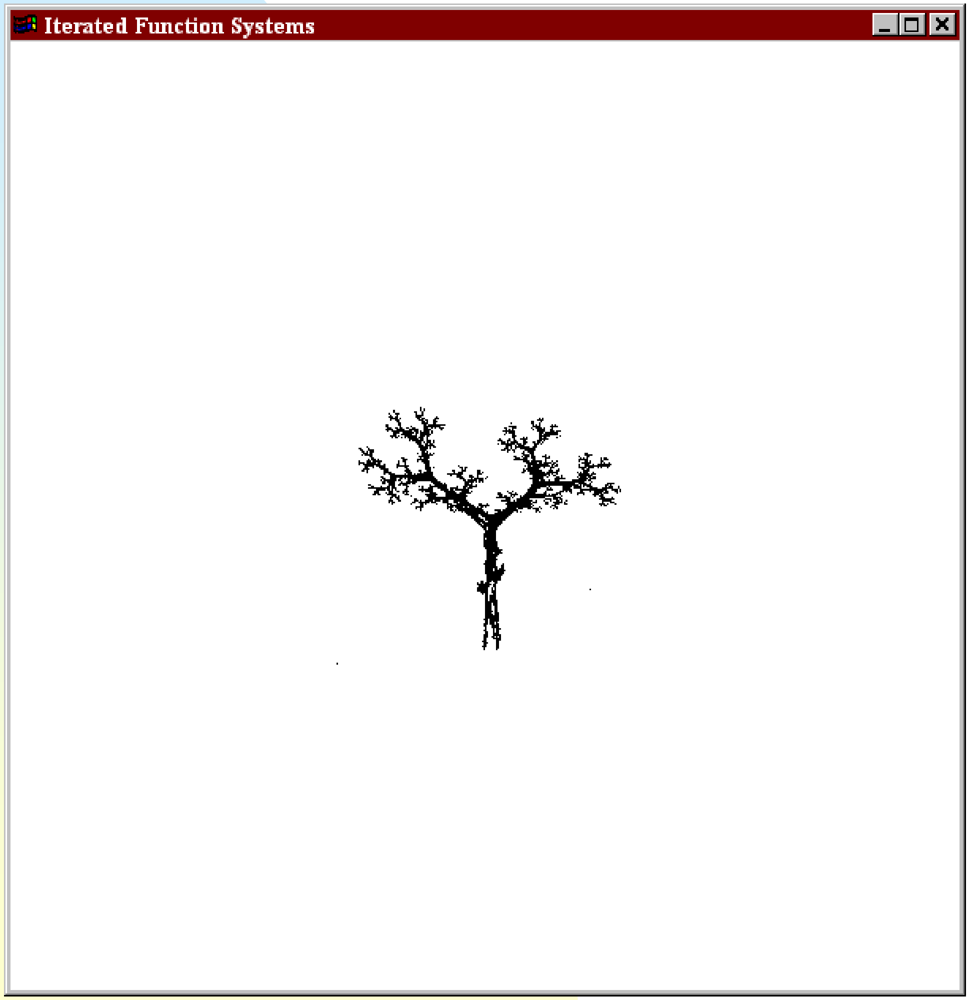
- #1: 0.387, 0.43, 0.43, -0.387, 0.256, 0.522
- #2: 0.441, -0.091, -0.009, -0.322, 0.4219, 0.5059
- #3: -0.468, 0.02, -0.113, 0.015, 0.4, 0.4
- n = 100,000

Crystal



- #1: 0.255, 0.0, 0.0, 0.255, 0.3726, 0.6714
- #2: 0.255, 0.0, 0.0, 0.255, 0.1146, 0.2232
- #3: 0.255, 0.0, 0.0, 0.255, 0.6306, 0.2232
- #4: 0.37, -0.0642, 0.642, 0.37, 0.6356, -0.0061
- n = 250,000

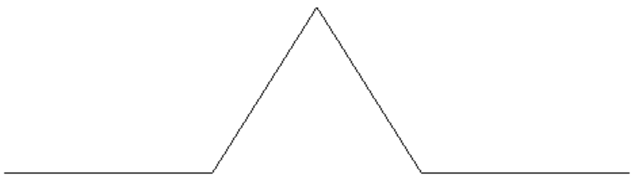
Tree



- #1: 0.195, -0.488, 0.344, 0.443, 0.4431, 0.2452
- #2: 0.462, 0.414, -0.252, 0.361, 0.2511, 0.5692
- #3: -0.058, -0.07, 0.453, -0.111, 0.5976, 0.0969
- #4: -0.035, 0.07, -0.469, -0.022, 0.4884, 0.5069
- #5: -0.637, 0.0, 0.0, 0.501, 0.8562, 0.2513
- n = 100,000

Koch curve IFS exercise

- Here is the generator for the Koch curve:



- Consider generating the Koch curve via an IFS
  - How many contraction mappings are there?
  - What are the parameter values for each of the mappings?

Koch snowflake via IFS

- How can the Koch curve be extended to the snowflake with an IFS?
- Apply a rotation and translation transformation to the generated points
- Program code is in `Kochifs.cpp`