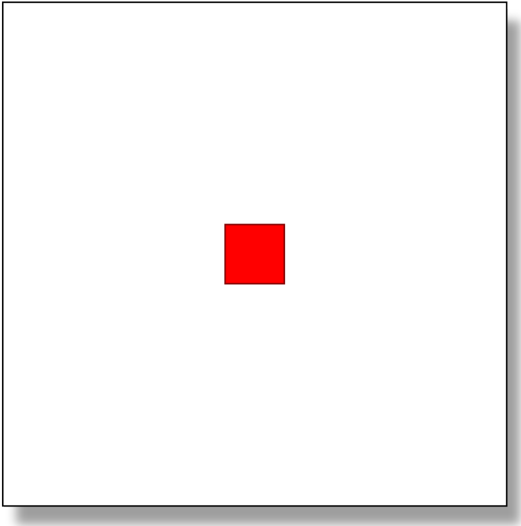
Fractal Growth

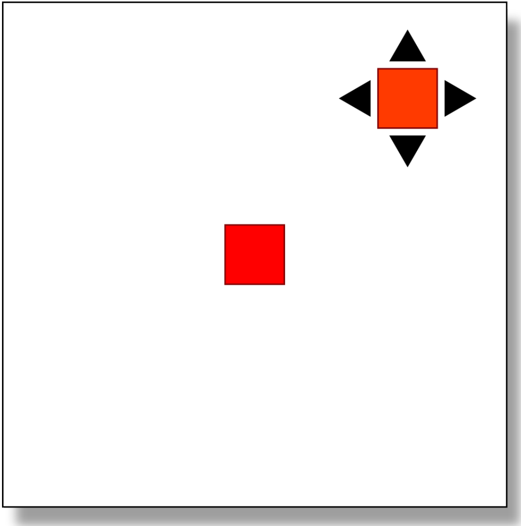
Seminar on Computational Physics

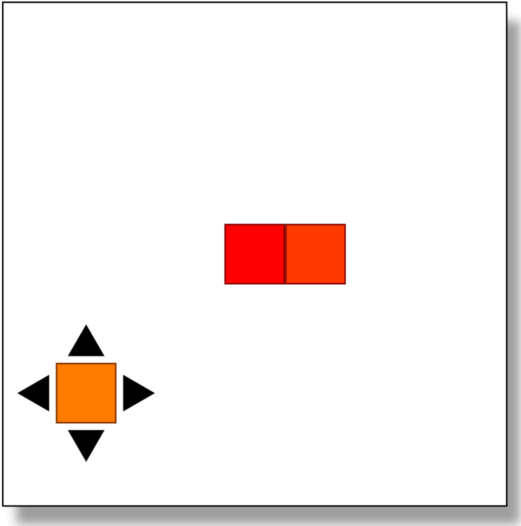
Benedikt Sauer, Alexander Schroer

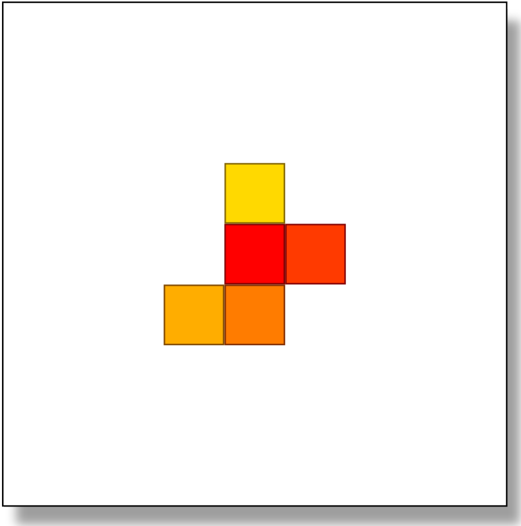
HISKP
Universität Bonn

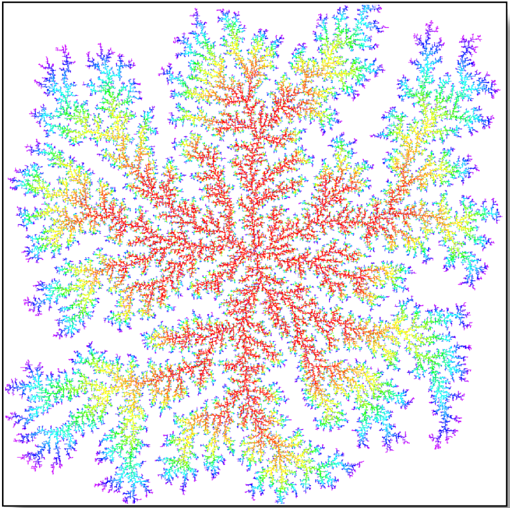
2nd March 2011











T. A. Witten, L. M. Sander, 1981

Outline

Fractal Dimension

Definition (Hausdorff measure)

The (outer) *d*-dimensional Hausdorff measure is defined as:

$$H_\varepsilon^d(S) = \inf_{\substack{\bigcup_{i=1}^\infty U_i \supset S \\ \mathrm{diam}(U_i) < \varepsilon}} \sum_{i=1}^\infty \mathrm{diam}(U_i)^d \tag{1}$$

The *d*-dimensional measure is (modulo some measure theory) the limit $\varepsilon \to 0$.

Definition (Hausdorff dimension)

The Hausdorff dimension of a set is defined as:

$$d_H = \sup\{d \in \mathbb{R}_0^+ \mid H^d(S) = \infty\} \tag{2}$$

Fractal Dimension

Definition (Hausdorff measure)

The (outer) d-dimensional Hausdorff measure is defined as:

$$H_\varepsilon^d(S) = \inf_{\substack{\bigcup_{i=1}^{\infty} U_i \supset S \\ \mathrm{diam}(U_i) < \varepsilon}} \sum_{i=1}^{\infty} \mathrm{diam}(U_i)^d \tag{1}$$

The d-dimensional measure is (modulo some measure theory) the limit $\varepsilon \to 0$.

Definition (Hausdorff dimension)

The Hausdorff dimension of a set is defined as:

$$d_H = \sup\{d \in \mathbb{R}_0^+ \mid H^d(S) = \infty\} \tag{2}$$

Discrete Calculations

Definition (Simpler working definition)

The fractal dimension of a point set is defined as the following:

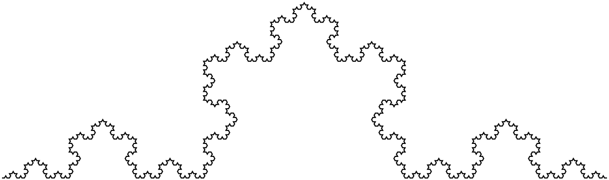
$$D = -\lim_{R \to 0} \frac{\log(N)}{\log(R)}$$

Discrete Calculations

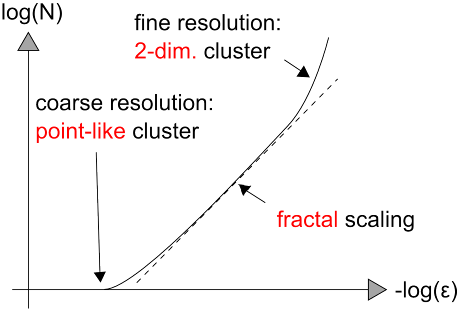
Definition (Simpler working definition)

The fractal dimension of a point set is defined as the following:

$$D = -\lim_{R\to 0} \frac{\log(N)}{\log(R)}$$



Fractal dimension of finite structures



Box-counting:
$\log(N) = -d \log(\epsilon)$

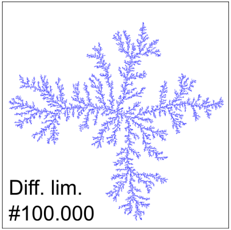
Other scaling power-laws:

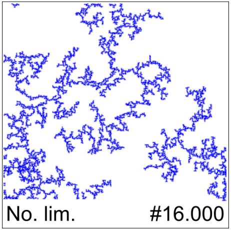
- radius of gyration $R_g \propto N^{1/d}$
- density-density-correlation $C(r) \propto r^{D-d}$
- ...

The Meakin model

1.
- Set seed in the center of the lattice
- Generate particles at $R + 10$ one by one
- Let particle diffuse against the cluster
- Particles further than $2R$ are removed

2.
- Generate many particles once
- Let them diffuse and clump together
- Steady state is reached

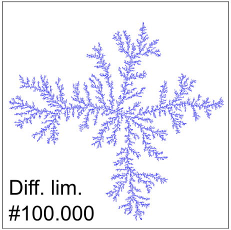


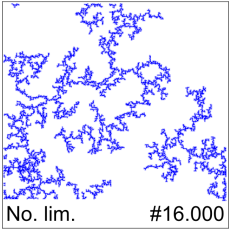


The Meakin model

1.
   - Set seed in the center of the lattice
   - Generate particles at $R + 10$ one by one
   - Let particle diffuse against the cluster
   - Particles further than $2R$ are removed

2.
   - Generate many particles once
   - Let them diffuse and clump together
   - Steady state is reached





Implementation: Outline

Two types of objects:

- Particles (which carry interaction information)
- Clusters (specialized containers for particles)

The main program step works as follows:

- Update the environment
  - Remove particles that are too far away or to old
  - Create new particles or clusters
- Let particles and clusters interact
  - If they are near enough merge them
  - Else let them move randomly by probabilities calculated in the interaction

If we want we output graphics or calculate statistics afterwards.

Implementation: Outline

Two types of objects:

- Particles (which carry interaction information)
- Clusters (specialized containers for particles)

The main program step works as follows:

- Update the environment
  - Remove particles that are too far away or to old
  - Create new particles or clusters
- Let particles and clusters interact
  - If they are near enough merge them
  - Else let them move randomly by probabilities calculated in the interaction

If we want we output graphics or calculate statistics afterwards.

Implementation: Outline

Two types of objects:

- Particles (which carry interaction information)
- Clusters (specialized containers for particles)

The main program step works as follows:

- Update the environment
    - Remove particles that are too far away or to old
    - Create new particles or clusters
- Let particles and clusters interact
    - If they are near enough merge them
    - Else let them move randomly by probabilities calculated in the interaction

If we want we output graphics or calculate statistics afterwards.

Implementation: Outline

Two types of objects:

- Particles (which carry interaction information)
- Clusters (specialized containers for particles)

The main program step works as follows:

- Update the environment
  - Remove particles that are too far away or to old
  - Create new particles or clusters
- Let particles and clusters interact
  - If they are near enough merge them
  - Else let them move randomly by probabilities calculated in the interaction

If we want we output graphics or calculate statistics afterwards.

Implementation: Outline

Two types of objects:

- Particles (which carry interaction information)
- Clusters (specialized containers for particles)

The main program step works as follows:

- Update the environment
  - Remove particles that are too far away or to old
  - Create new particles or clusters
- Let particles and clusters interact
  - If they are near enough merge them
  - Else let them move randomly by probabilities calculated in the interaction

If we want we output graphics or calculate statistics afterwards.

Implementation: Outline

Two types of objects:

- Particles (which carry interaction information)
- Clusters (specialized containers for particles)

The main program step works as follows:

- Update the environment
  - Remove particles that are too far away or to old
  - Create new particles or clusters
- Let particles and clusters interact
  - If they are near enough merge them
  - Else let them move randomly by probabilities calculated in the interaction

If we want we output graphics or calculate statistics afterwards.

Implementation: Outline

Two types of objects:

- Particles (which carry interaction information)
- Clusters (specialized containers for particles)

The main program step works as follows:

- Update the environment
  - Remove particles that are too far away or to old
  - Create new particles or clusters
- Let particles and clusters interact
  - If they are near enough merge them
  - Else let them move randomly by probabilities calculated in the interaction

If we want we output graphics or calculate statistics afterwards.

Implementation: Outline

Two types of objects:

- Particles (which carry interaction information)
- Clusters (specialized containers for particles)

The main program step works as follows:

- Update the environment
  - Remove particles that are too far away or to old
  - Create new particles or clusters
- Let particles and clusters interact
  - If they are near enough merge them
  - Else let them move randomly by probabilities calculated in the interaction

If we want we output graphics or calculate statistics afterwards.

Implementation: Outline

Two types of objects:

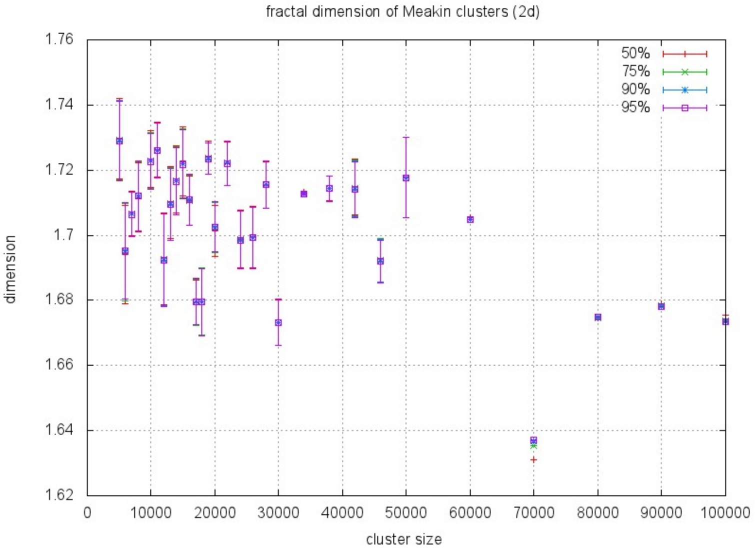
- Particles (which carry interaction information)
- Clusters (specialized containers for particles)

The main program step works as follows:

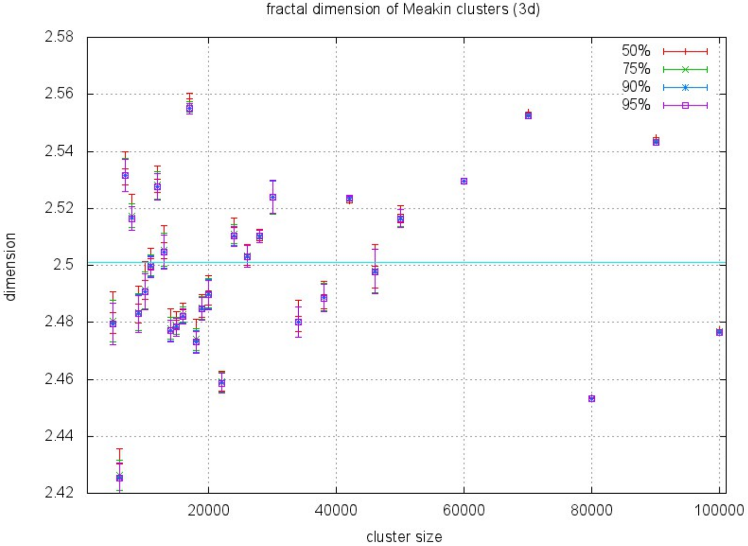
- Update the environment
  - Remove particles that are too far away or to old
  - Create new particles or clusters
- Let particles and clusters interact
  - If they are near enough merge them
  - Else let them move randomly by probabilities calculated in the interaction

If we want we output graphics or calculate statistics afterwards.

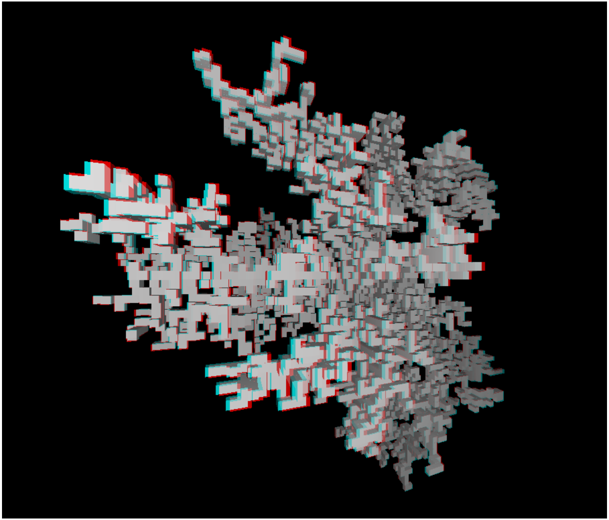
Fractal dimension of 2-d Meakin clusters



Fractal dimension of 3-d Meakin clusters

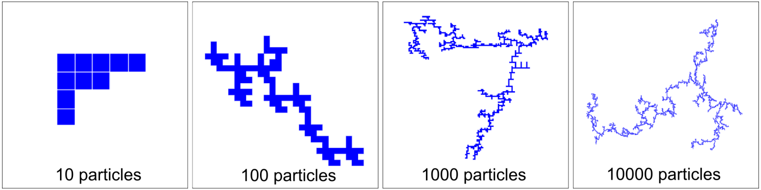


Anaglyph of 3-d Meakin cluster (8000 particles)



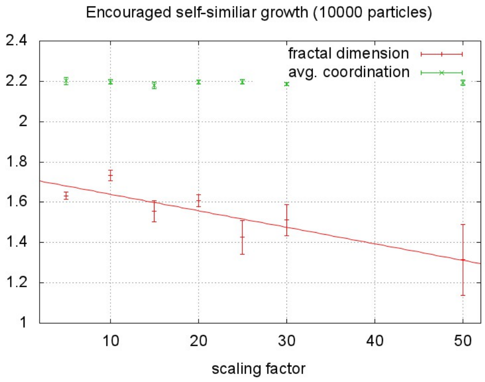
Encouraged self-similar growth

- Want to create structures with tailor-made dimensionality while maintaining basic building principle (physically motivated!)
- Idea: Encourage certain scaling behaviour (remember $D \approx -\frac{\log N}{\log \epsilon}$)
- Diffuse clusters instead of particles



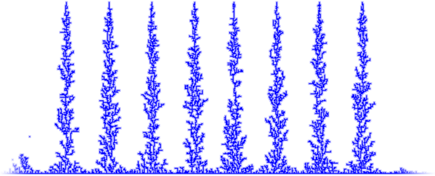
A theoretical treatment is difficult:

- Hybridization of IFS and statistical process, so no simple scaling law
- Non-negligible finite-size effects

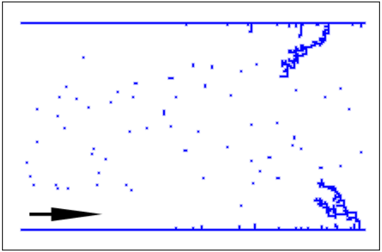


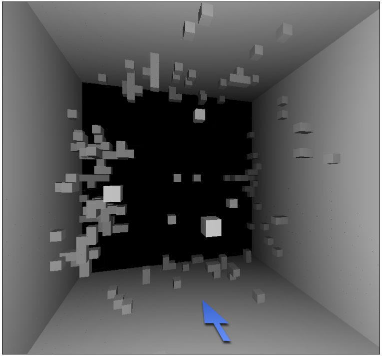
Dendrites

Pyrolusite
Jonathan Zander, 2008

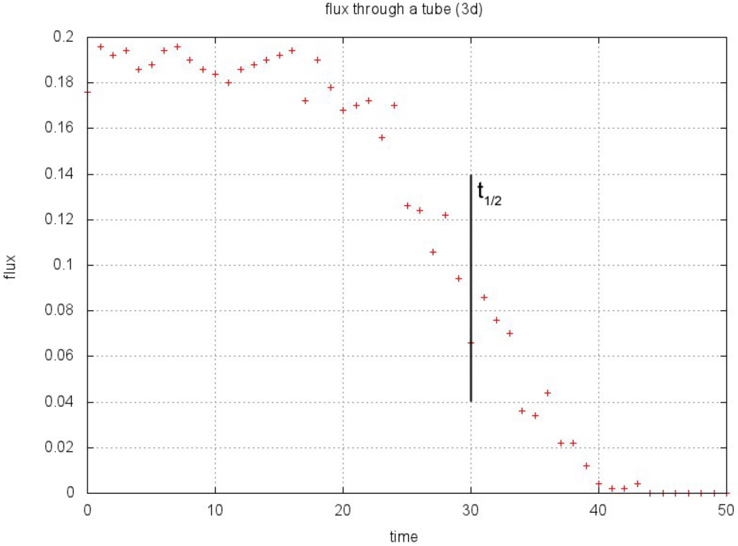


Tubes

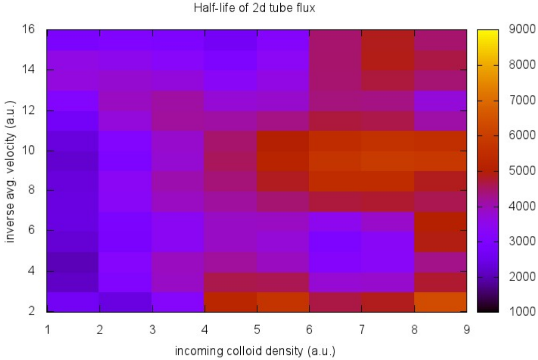




Tubes: flux


flux through a tube (3d)
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
flux
0
10
20
30
40
50
time
$t_{1/2}$

Tubes: flux

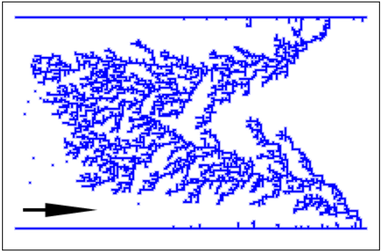


2-dimensional simulation, length/radius: 10

Tubes: real-world physics?

Obstacles are fixed unphysically:

- require breaking of bonds
- leads to MD-like simulations

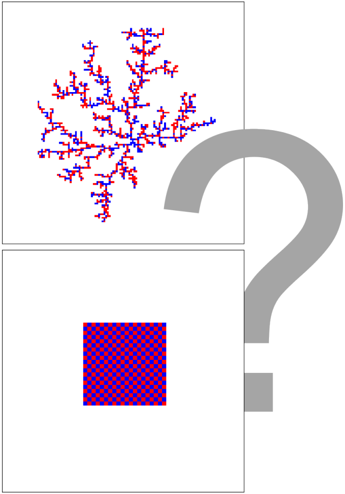


Crystals vs. Dendrites

Idea: Add Coulomb-like interaction

- Introduce particle charge
- Set preferred directions in random walk depending on other particles
- Try to observe a transition from $D = 1.6$ to $D = 2$

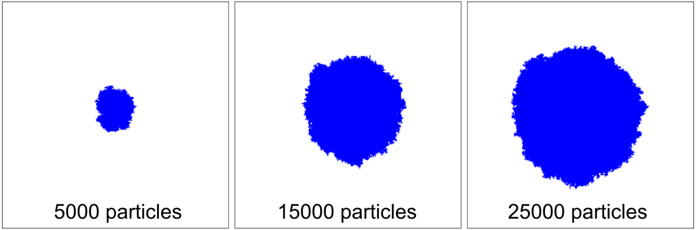
This is work in progress.



The Eden-Meakin model

Idea

Describe organic growth (rather than anorganic):
Add new clusters size at nearest neighbours

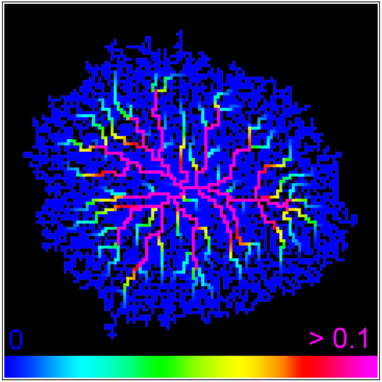


M. Eden, 1961, P. Meakin, 1991

The Eden-Meakin model

Scoring system:

- For each new particle, randomly choose an adjacent "parent"
- Add a score $\Delta S = (1 + l)^{-\eta}$ to each of the l ancestors
- Reduce all scores S_i by $1/N_m$
- Remove particles of non-positive score

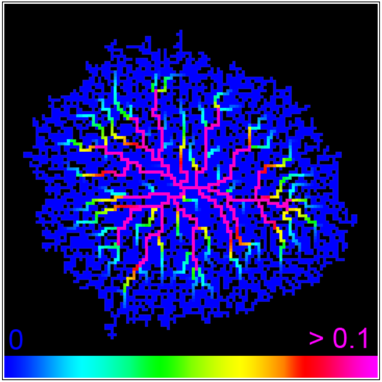


$\eta = 1$, $N_m = 102400$

The Eden-Meakin model

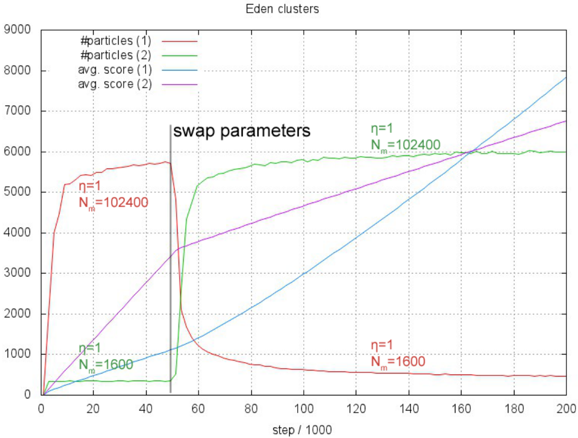
Features:

- fractal backbone with $D = D(S_{\text{thres}})$
- finite "equilibrium" size
- adaptivity, memory

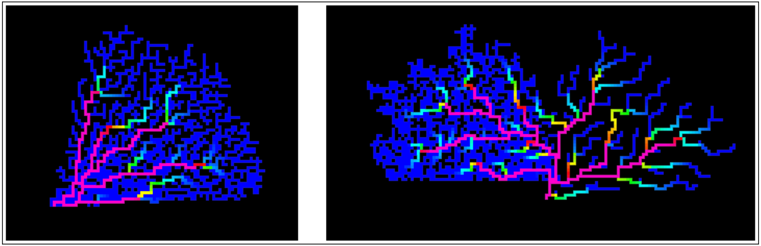


$\eta = 1$, $N_m = 102400$

The Eden-Meakin model: examples



The Eden-Meakin model: examples



Geometrical scoring: $\Delta S \propto \frac{\vec{x} \cdot \hat{e}_{\text{sun}}}{|\vec{x}|}$

Summary

- Combining simple methods motivated by natural processes generates fractal structures, e.g. Brownian motion and molecular adhesion.
- It is possible to simulate these efficiently on a lattice
- The fractal dimension can be calculated as $D = -\frac{\log N}{\log \epsilon}$ or estimated on discretized structures via the radius of gyration.
- Fractal structures describe anorganic and organic real world phenomena.

References

- T. A. Witten and L. M. Sander, Phys. Rev. Lett. 47, 1400 (1981)
- H. E. Stanley, J. Phys. A 10, L211 (1977)
- P. Meakin, Phys. Rev. A 27, 1495 (1983)
- P. Meakin, Phys. Rev. Lett. 51, 13 (1983)
- M. Eden, Proc. 4th Berkeley Symp. on Mathematics, Statistics and Probability, vol. 4, F. Neyman, ed., (1961)
- P. Meakin, Physica A, 179 (1991)

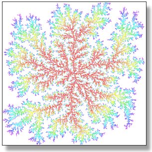
Fractal growth is fun!

source code

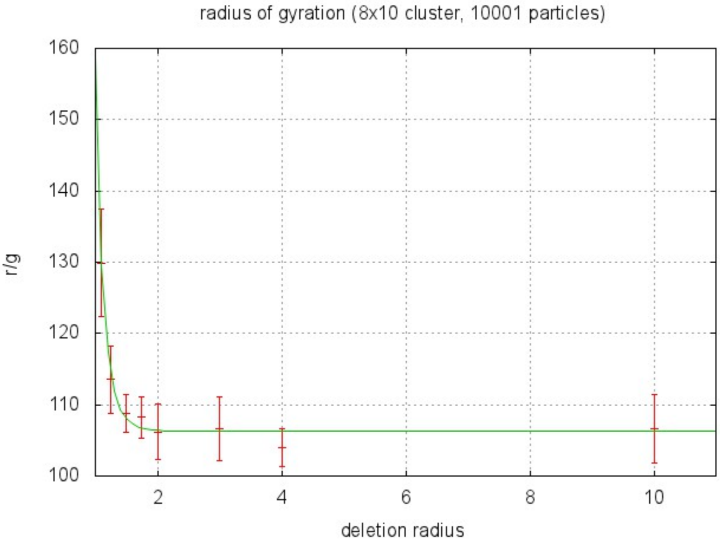
`http://github.com/filmor` or `http://github.com/palmstroem`

(may not always be in working state, requires gcc $\geq$ 4.5.2)

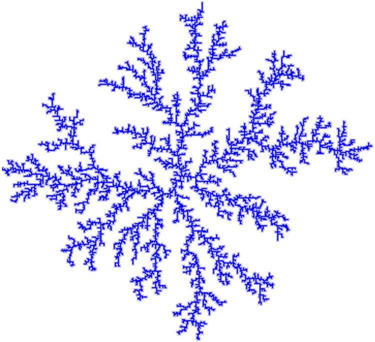
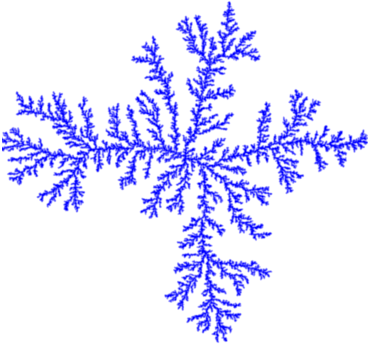
Thank you for listening!



"Diffusion from infinity"



Self-similiarity

Clusters

Cluster objects need to do some tricks to be efficient.

- Memory efficiency:
  - Save only a cube surrounding the cluster, not a full lattice
  - Let this cube grow on purpose (in 2^d steps) and reset its center
- Runtime efficiency:
  - Save also a vector of all particles
  - Save a radius (for fast collision checks)

Clusters

Cluster objects need to do some tricks to be efficient.

- Memory efficiency:
  - Save only a cube surrounding the cluster, not a full lattice
  - Let this cube grow on purpose (in 2^d steps) and reset its center
- Runtime efficiency:
  - Save also a vector of all particles
  - Save a radius (for fast collision checks)

Clusters

Cluster objects need to do some tricks to be efficient.

- Memory efficiency:
  - Save only a cube surrounding the cluster, not a full lattice
  - Let this cube grow on purpose (in 2^d steps) and reset its center
- Runtime efficiency:
  - Save also a vector of all particles
  - Save a radius (for fast collision checks)

Clusters

Cluster objects need to do some tricks to be efficient.

- Memory efficiency:
  - Save only a cube surrounding the cluster, not a full lattice
  - Let this cube grow on purpose (in 2^d steps) and reset its center
- Runtime efficiency:
  - Save also a vector of all particles
  - Save a radius (for fast collision checks)

Clusters

Cluster objects need to do some tricks to be efficient.

- Memory efficiency:
  - Save only a cube surrounding the cluster, not a full lattice
  - Let this cube grow on purpose (in 2^d steps) and reset its center
- Runtime efficiency:
  - Save also a vector of all particles
  - Save a radius (for fast collision checks)

Clusters

Cluster objects need to do some tricks to be efficient.

- Memory efficiency:
  - Save only a cube surrounding the cluster, not a full lattice
  - Let this cube grow on purpose (in 2^d steps) and reset its center
- Runtime efficiency:
  - Save also a vector of all particles
  - Save a radius (for fast collision checks)

Clusters

Cluster objects need to do some tricks to be efficient.

- Memory efficiency:
  - Save only a cube surrounding the cluster, not a full lattice
  - Let this cube grow on purpose (in 2^d steps) and reset its center
- Runtime efficiency:
  - Save also a vector of all particles
  - Save a radius (for fast collision checks)

Clusters

Cluster objects need to do some tricks to be efficient.

- Memory efficiency:
  - Save only a cube surrounding the cluster, not a full lattice
  - Let this cube grow on purpose (in 2^d steps) and reset its center
- Runtime efficiency:
  - Save also a vector of all particles
  - Save a radius (for fast collision checks)

Additional Notes

- The whole program is build of templates (compile-time polymorphy)
    - Gives similar flexibility as runtime-polymorphy (virtual functions)
    - Greatly improves performace on good compilers (ca. handcoded C)
    - Lets us use the same framework for Meakin and Eden
- In a later version will unify clusters and particles
    - They have similar movement and interaction characteristics
    - $\Rightarrow$ Implementation of the above is currently tedious
- Sadly we haven't found a method to parallelize, if someone finds one, tell us :)

Additional Notes

- The whole program is build of templates (compile-time polymorphy)
    - Gives similar flexibility as runtime-polymorphy (virtual functions)
    - Greatly improves performace on good compilers (ca. handcoded C)
    - Lets us use the same framework for Meakin and Eden
- In a later version will unify clusters and particles
    - They have similar movement and interaction characteristics

  $\Rightarrow$ Implementation of the above is currently tedious
- Sadly we haven't found a method to parallelize, if someone finds one, tell us :)

Additional Notes

- The whole program is build of templates (compile-time polymorphy)
  - Gives similar flexibility as runtime-polymorphy (virtual functions)
  - Greatly improves performace on good compilers (ca. handcoded C)
  - Lets us use the same framework for Meakin and Eden
- In a later version will unify clusters and particles
  - They have similar movement and interaction characteristics
  - $\Rightarrow$ Implementation of the above is currently tedious
- Sadly we haven't found a method to parallelize, if someone finds one, tell us :)

Additional Notes

- The whole program is build of templates (compile-time polymorphy)
  - Gives similar flexibility as runtime-polymorphy (virtual functions)
  - Greatly improves performace on good compilers (ca. handcoded C)
  - Lets us use the same framework for Meakin and Eden
- In a later version will unify clusters and particles
  - They have similar movement and interaction characteristics

  $\Rightarrow$ Implementation of the above is currently tedious
- Sadly we haven't found a method to parallelize, if someone finds one, tell us :)

Additional Notes

- The whole program is build of templates (compile-time polymorphy)
  - Gives similar flexibility as runtime-polymorphy (virtual functions)
  - Greatly improves performace on good compilers (ca. handcoded C)
  - Lets us use the same framework for Meakin and Eden
- In a later version will unify clusters and particles
  - They have similar movement and interaction characteristics
  - $\Rightarrow$ Implementation of the above is currently tedious
- Sadly we haven't found a method to parallelize, if someone finds one, tell us :)

Additional Notes

- The whole program is build of templates (compile-time polymorphy)
  - Gives similar flexibility as runtime-polymorphy (virtual functions)
  - Greatly improves performace on good compilers (ca. handcoded C)
  - Lets us use the same framework for Meakin and Eden
- In a later version will unify clusters and particles
  - They have similar movement and interaction characteristics

  $\Rightarrow$ Implementation of the above is currently tedious
- Sadly we haven't found a method to parallelize, if someone finds one, tell us :)