MOLECULAR DYNAMICS

by

Filip Sund

THESIS

for the degree of

MASTER OF SCIENCE

Faculty of Mathematics and Natural Sciences
University of Oslo

June 2012

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus

quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

To someone

This is a dedication to my cat.

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus

quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Contents

Part I

Introduction

Part II

Molecular dynamics

Chapter 1

Simulations

1.1 Potential

- Which of the parameters given below are constants?
- Does the potential care about number of neighbors (for example via adjustment of some parameters) ?

The interatomic potential[14] we use for for both silica and water consists of two-body and three-body terms, and has the form

not exactly same as vashishta, see nanobubble supplements

$$E_{\text{tot}} = \sum_{i<j} V_{ij}^{(2)}(r_{ij}) + \sum_{i<j<k} V_{ijk}^{(3)}(\boldsymbol{r}_{ij}, \boldsymbol{r}_{ij}),$$

for

$$1 \leq \{i, j, k\} \leq N.$$

$V_{ij}^{(2)}$ is the two-body term, which consists of four terms that take into account steric repulsion, charge-charge (Coulomb), charge-dipole, and dipole-dipole (van der Waals) interactions, and has the form

$$V_{ij}^{(2)}(r) = \underbrace{\frac{H_{ij}}{r^{\eta_{ij}}}}_{\text{steric repulsion}} + \underbrace{\frac{Z_i Z_j}{r} e^{-r/r_{1s}}}_{\text{Coulomb}} - \underbrace{\frac{D_{ij}}{2r^4} e^{-r/r_{4s}}}_{\text{charge-dipole}} - \underbrace{\frac{w_{ij}}{r^6}}_{\text{van der Waals}},$$

where r is the distance between two atoms i and j, H_{ij} and η_{ij} are the strengths of the steric repulsion, Z_i is the charge associated with atom i,

D_{ij} controls the charge-dipole interaction, w_{ij} controls the dipole-dipole interaction, and r_{1s} and r_{4s} are the screening lengths for the Coulumb and charge-dipole interactions respectively.

$V_{ijk}^{(3)}$ is the three-body term, which take into account bending and stretching of covalent bonds, and has the form

split into $f(\boldsymbol{r}_{ij}, \boldsymbol{r}_{ik})$ and $p(\theta_{ijk}, \theta_0)$ as in vashista 1990?

$$V_{jik}^{(3)}(\boldsymbol{r}_{ij}, \boldsymbol{r}_{ik}) = B_{ijk} \underbrace{\exp\left(\frac{\xi}{r_{ij} - r_0} + \frac{\xi}{r_{ij} - r_0}\right)}_{\text{bond-stretching}} \underbrace{\frac{\left(\cos\theta_{ijk} - \cos\theta_0\right)^2}{1 + C_{ijk}\left(\cos\theta_{ijk} - \cos\theta_0\right)^2}}_{\text{bond-bending}},$$

for

$$\{r_{ij}, r_{ik}\} \leq r_0,$$

where $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$, $r_{ij} = |\boldsymbol{r}_{ij}|$, r_0 is the cutoff distance for the three-body interaction, θ_{ijk} is the angle between $\boldsymbol{r}_{ij}$ and $\boldsymbol{r}_{ik}$, B_{ijk} is the strength of the three-body interaction, and θ_0 is a parameter that controls the angle at which the three-body term vanishes.

ξ and C_{ijk} ???

cuts off interaction at r_0 with no discontinuities in the derivatives with respect to r (vashista 1990)

1.2 Integration

TODO:

- Truncation error Verlet/velociy Verlet
- Numerical stability?
- Memory?
- Self starting, symplectic, reversible

The equations of motion are integrated using the velocity Verlet algorithm:

$$\boldsymbol{r}(t+\Delta t) = \boldsymbol{r}(t) + \boldsymbol{v}(t)\Delta t + \frac{\boldsymbol{F}(t)}{2m}\Delta t^2$$
$$\boldsymbol{v}(t+\Delta t) = \boldsymbol{v}(t) + \frac{\boldsymbol{F}(t+\Delta t) + \boldsymbol{F}(t)}{2m}\Delta t.$$

This algorithm has a glocal/accumulated error of $\mathcal{O}(\Delta t^2)$.

either [13] sec. 8.4.1-8.4.3 or [7] sec. 4.3.3

1.2.1 Derivation of Verlet algorithm

The Verlet algorithm[15] is a simple method for integrating second order differential equations of the form

why do we use velocity Verlet

$$\frac{\mathrm{d}^2\boldsymbol{r}(t)}{\mathrm{d}t^2} = \boldsymbol{F}\Big[\boldsymbol{r}(t), t\Big] = \boldsymbol{F}(t).$$

We first let

$$\frac{\mathrm{d}\boldsymbol{r}(t)}{\mathrm{d}t} = \boldsymbol{v}(t),$$

and

$$\frac{\mathrm{d}\boldsymbol{v}(t)}{\mathrm{d}t} = \boldsymbol{a}(t) = \frac{\boldsymbol{F}(t)}{m}.$$

We then do a Taylor expansion of $\boldsymbol{r}(t \pm \Delta t)$ around time t

$$\boldsymbol{r}(t+\Delta t) = \boldsymbol{r}(t) + \boldsymbol{v}(t)\Delta t + \boldsymbol{a}(t)\frac{\Delta t^2}{2} + \frac{\mathrm{d}^3\boldsymbol{r}(0)}{\mathrm{d}t^3}\frac{\Delta t^3}{6} + \mathcal{O}(\Delta t^4), \tag{1.1}$$
$$\boldsymbol{r}(t-\Delta t) = \boldsymbol{r}(t) - \boldsymbol{v}(t)\Delta t + \boldsymbol{a}(t)\frac{\Delta t^2}{2} - \frac{\mathrm{d}^3\boldsymbol{r}(0)}{\mathrm{d}t^3}\frac{\Delta t^3}{6} + \mathcal{O}(\Delta t^4). \tag{1.2}$$

By summing these two equations we get

$$\boldsymbol{r}(t+\Delta t) + \boldsymbol{r}(t-\Delta t) = 2\boldsymbol{r}(t) + \boldsymbol{a}(t)\Delta t^2 + \mathcal{O}(\Delta t^4),$$

which by rearranging can be written as

$$\boldsymbol{r}(t+\Delta t) \approx 2\boldsymbol{r}(t) - \boldsymbol{r}(t-\Delta t) + \boldsymbol{a}(t)\Delta t^2,$$

Which is the equation used to update the positions in the regular Verlet algorithm. We see that the estimate of the new position contains an truncation error for one timestep Δt of the order $\mathcal{O}(\Delta t^4)$.

The Verlet algorithm does not use the velocity to compute the new position, but we can find an estimate of the velocity by taking the difference between eqs. (1.1) and (1.2)

$$\boldsymbol{r}(t+\Delta t) - \boldsymbol{r}(t-\Delta t) = 2\boldsymbol{v}(t)\Delta t + \mathcal{O}(\Delta t^3),$$

which by rearranging can be written as

$$\boldsymbol{v}(t) = \frac{\boldsymbol{r}(t+\Delta t) - \boldsymbol{r}(t-\Delta t)}{2\Delta t} + \mathcal{O}(\Delta t^2).$$

Something about lower precision

We see that this estimate of the velocity has a truncation error of the order $\mathcal{O}(\Delta t^2)$, compared to the error in the position $\mathcal{O}(\Delta t^4)$.

something about why velocity Verlet is good

A modification of the Verlet algorithm usually called the velocity Verlet algorithm[12] can be derived in a similar way. We have the same Taylor expansion of $\boldsymbol{r}(t+\Delta t)$ around t as before

$$\boldsymbol{r}(t+\Delta t) = \boldsymbol{r}(t) + \boldsymbol{v}(t)\Delta t + \boldsymbol{a}(t)\frac{\Delta t^2}{2} + \mathcal{O}(\Delta t^3), \tag{1.3}$$

and now we also expand $\boldsymbol{v}(t+\Delta t)$ around t

$$\boldsymbol{v}(t+\Delta t) = \boldsymbol{v}(t) + \boldsymbol{a}(t)\Delta t + \frac{\mathrm{d}^2\boldsymbol{v}(t)}{\mathrm{d}t^2}\frac{\Delta t^2}{2} + \mathcal{O}(\Delta t^3). \tag{1.4}$$

We now need an expression for $\frac{\mathrm{d}^2\boldsymbol{v}(t)}{\mathrm{d}t^2}$, which can be found by a Taylor expansion of $\frac{\mathrm{d}\boldsymbol{v}(t+\Delta t)}{\mathrm{d}t}$

$$\frac{\mathrm{d}\boldsymbol{v}(t+\Delta t)}{\mathrm{d}t} = \frac{\mathrm{d}\boldsymbol{v}(t)}{\mathrm{d}t} + \frac{\mathrm{d}^2\boldsymbol{v}(t)}{\mathrm{d}t^2}\Delta t + \mathcal{O}(\Delta t^2),$$

which by rearranging and multiplying with $\frac{\Delta t}{2}$ gives

$$\begin{aligned}\frac{\mathrm{d}^2\boldsymbol{v}}{\mathrm{d}t^2}\frac{\Delta t^2}{2} &= \left(\frac{\mathrm{d}\boldsymbol{v}(t+\Delta t)}{\mathrm{d}t} - \frac{\mathrm{d}\boldsymbol{v}(t)}{\mathrm{d}t}\right)\frac{\Delta t}{2} + \mathcal{O}(\Delta t^3) \\ &= \Big[\boldsymbol{a}(t+\Delta t) - \boldsymbol{a}(t)\Big]\frac{\Delta t}{2} + \mathcal{O}(\Delta t^3).\end{aligned}$$

Inserting this into eq. (1.4) we get

$$\begin{aligned}\boldsymbol{v}(t+\Delta t) &= \boldsymbol{v}(t) + \boldsymbol{a}(t)\Delta t + \Big[\boldsymbol{a}(t+\Delta t) - \boldsymbol{a}(t)\Big]\frac{\Delta t}{2} + \mathcal{O}(\Delta t^3) \\ &= \boldsymbol{v}(t) + \Big[\boldsymbol{a}(t) + \boldsymbol{a}(t+\Delta t)\Big]\frac{\Delta t}{2} + \mathcal{O}(\Delta t^3).\end{aligned} \tag{1.5}$$

So the total velocity Verlet algorithm with truncation of the higher-order terms is

$$\boldsymbol{r}(t+\Delta t) = \boldsymbol{r}(t) + \boldsymbol{v}(t)\Delta t + \boldsymbol{a}(t)\frac{\Delta t^2}{2}, \tag{1.6}$$

$$\boldsymbol{v}(t+\Delta t) = \boldsymbol{v}(t) + \Big[\boldsymbol{a}(t) + \boldsymbol{a}(t+\Delta t)\Big]\frac{\Delta t}{2}, \tag{1.7}$$

with the truncation error for one timestep Δt being of the order $\mathcal{O}(\Delta t^3)$ for both the position and the velocity.

The algorithm is usually rewritten in the following way, to optimize the implementation on a computer. We see that the new velocities can be written as

$$\boldsymbol{v}(t+\Delta t) = \tilde{\boldsymbol{v}}(t+\tfrac{1}{2}\Delta t) + \boldsymbol{a}(t+\Delta t)\frac{\Delta t}{2}, \tag{1.8}$$

where

$$\tilde{\boldsymbol{v}}(t+\tfrac{1}{2}\Delta t) = \boldsymbol{v}(t) + \boldsymbol{a}(t)\frac{\Delta t}{2}. \tag{1.9}$$

We see that eq. (1.9) can be used in updating the positions, so we rewrite eq. (1.6) to

$$\boldsymbol{r}(t+\Delta t) = \boldsymbol{r}(t) + \tilde{\boldsymbol{v}}(t+\tfrac{1}{2}\Delta t)\Delta t. \tag{1.10}$$

Which leads us to the usual way of implementing the algorithm[2]:

- Calculate the velocities at $t+\frac{1}{2}\Delta t$ using eq. (1.9) (repeated here)

$$\tilde{\boldsymbol{v}}(t+\tfrac{1}{2}\Delta t) = \boldsymbol{v}(t) + \frac{\boldsymbol{F}(t)}{m}\frac{\Delta t}{2}.$$

- Calculate the new positions at $t+\Delta t$ using eq. (1.10) (repeated here)

$$\boldsymbol{r}(t+\Delta t) = \boldsymbol{r}(t) + \tilde{\boldsymbol{v}}(t+\tfrac{1}{2}\Delta t)\Delta t.$$

- Calculate the new forces $\boldsymbol{F}(t+\Delta t)$.
- Calculate the new velocities at $t+\Delta t$ using eq. (1.8) (repeated here)

$$\boldsymbol{v}(t+\Delta t) = \boldsymbol{v}(t+\tfrac{1}{2}\Delta t) + \frac{\boldsymbol{F}(t+\Delta t)}{m}\frac{\Delta t}{2}.$$

This implementation minimizes the memory needs, as we only need to store one copy of $\boldsymbol{r}$, $\boldsymbol{v}$ and $\boldsymbol{F}$ at all times, compared to implementing eqs. (1.6) and (1.7) which needs to store the values of both $\boldsymbol{F}(t)$ and $\boldsymbol{F}(t+\Delta)$ to calculate the new velocities.

do we want this?

Pseudocode:

```
v += F*dt/(2*m);
r += v*dt;
F = calculate_forces(r, v);
v += F*dt/(2*m);
```

1.2.2 Global error in the Verlet algorithm

Sources:

- `http://math.stackexchange.com/questions/668707/verlet-method-global-erro`
- `http://www.saylor.org/site/wp-content/uploads/2011/06/MA221-6.1.pdf` (same as Wikipedia)

I don't know if the calculations below are correct

The global error in position can be derived from the local error for one timestep Δt. We see from eq. (1.3) that

$$\text{error}\Big(\boldsymbol{r}(t_0+\Delta t)\Big) = \mathcal{O}(\Delta t^3),$$

and from eq. (1.5)

$$\text{error}\Big(\boldsymbol{v}(t_0+\Delta t)\Big) = \mathcal{O}(\Delta t^3).$$

The error for two timesteps is

$$\boldsymbol{r}(t_0+2\Delta t) = \boldsymbol{r}(t_0+\Delta t) + \boldsymbol{v}(t_0+\Delta t)\Delta t + \boldsymbol{a}(t_0+\Delta t)\frac{\Delta t^2}{2} + \mathcal{O}(\Delta t^3),$$

Error in $\boldsymbol{a}(t_0+\Delta t) = \mathcal{O}(\Delta t^3)$??

which gives

$$
\begin{aligned}
&\text{error}\Big(\boldsymbol{r}(t_0 + 2\Delta t)\Big) \\
&= \text{error}\Big(\boldsymbol{r}(t_0 + \Delta t)\Big) + \text{error}\Big(\boldsymbol{v}(t_0 + \Delta t)\Big)\Delta t + \text{error}\Big(\boldsymbol{a}(t_0 + \Delta t)\Big)\frac{\Delta t^2}{2} + \mathcal{O}(\Delta t^3) \\
&= \mathcal{O}(\Delta t^3) + \mathcal{O}(\Delta t^3)\Delta t + \mathcal{O}(\Delta t^3)\frac{\Delta t^2}{2} + \mathcal{O}(\Delta t^3) \\
&= 2\mathcal{O}(\Delta t^3).
\end{aligned}
$$

Similarly we find

$$
\begin{aligned}
\text{error}\Big(\boldsymbol{r}(t_0 + 3\Delta t)\Big) &= 3\mathcal{O}(\Delta t^3) \\
\text{error}\Big(\boldsymbol{r}(t_0 + 4\Delta t)\Big) &= 4\mathcal{O}(\Delta t^3) \\
\text{error}\Big(\boldsymbol{r}(t_0 + 5\Delta t)\Big) &= 5\mathcal{O}(\Delta t^3),
\end{aligned}
$$

$$
\text{error}\Big(\boldsymbol{r}(t_0 + n\Delta t)\Big) = n\mathcal{O}(\Delta t^3) = \mathcal{O}(\Delta t^2)
$$

1.3 Ensemble, observables etc.

1.4 Initialization

To initialize the system we generate a silica crystal in the crystalline form β-cristobalite.

1.5 Passivation

Since we don't take into consideration molecular bonds in the silica when removing atoms to create pores, we get dangling unsaturated bonds in the system. We rectify this by passivating the system by inserting atoms on the dangling bonds, turning them into stable silanol groups.

why silanol?

In the passivation procedure we do some basic assumptions, based on the chemical nature of silica and water. In the thermodynamically stable form, silica should have the following properties:

source?

- Silicon atoms should have tetrahedral coordination, with four oxygen atoms surrounding each silicon atom in a tetrahedral shape.

something about bonds? bonded to four oxygen atoms?

- Oxygen atoms should have two silicon neighbors.
- The Si-O distance should be in the range 1.5-1.9 pm(depending on the crystalline form).

source?

When removing atoms to create pores we don't care about these properties, which leads us to the following cases

- Silicon atoms with less than four oxygen neighbors.
- Oxygen atoms with one missing silicon neighbor.
- Silicon and oxygen atoms with no neighbors.
- Silicon and oxygen atoms with too many neighbors?

When inserting oxygen and hydrogen we must make sure to inject neutrally, meaning twice as much hydrogen as oxygen (H_2O)

Silicon atoms with less than four oxygen atoms bound to them get $(4 - n_O)$ hydroxide (OH^-) groups attached to them, where n_O is the numer of oxygen atoms bound to the silicon. Oxygen atoms with a missing silicon neighbor get a hydrogen attached.

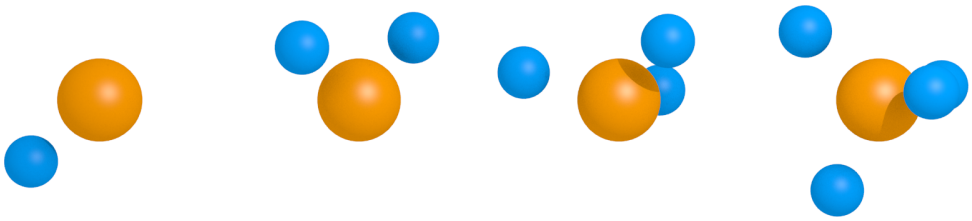
When passivating a silicon atom with missing oxygen neighbors by simply filling in the missing atoms to complete the SiO_4-tetrahedra. We then put one hydrogen atom on each new oxygen atom, to avoid dangling bonds on the inserted oxygen atoms.

When passivating a oxygen atom with a missing silicon neighbor, we put one hydrogen atom on the opposite side of the oxygen atom compared to the silicon atom.

All silicon and oxygen atoms with no neighbors we remove, since they aren't really part of the silica.

To find the number of neighbors/bonds/bonded atoms for each silicon and oxygen atom, we create what we call *neigbor lists*, which is a list of atoms within a chosen radius, for each atom.

- Tetrahedra



(a) (b) (c) (d)

Figure 1.1: Caption

- Neighbor lists – see base_code/passivate_using_tetrahedra/passivator.cpp near line 700
  - Create list of atoms in each voxel
  - Create neighbor lists for each atom by looping through neighbor voxels for each atoms
- Count number of neighbors of different types – find number of missing neighbors, Si - 4 Oxygen, Oxygen 2 Si
- Insert OH on Si with missing O neighbors, insert H on Oxygen with missing Si neighbors
  - Insert O/H at good angles
- Improvement: find the atoms near surface using voxels, only passivate those atoms

1.6 Injecting water

To fill the pore we have made (after passivating the system) we use the technique of *voxelation* (see section 3.1), and put one water molecule in each unoccupied voxel. The water density is then controlled by the size of the voxels.

If we want to inject water with density ρ [kg/m^3], we can find the voxel size we need using the molar mass of water, $M_{H_2O} = M = 0.0180158$ kg/mol. We find the "volume" of a water atom in Ångström, the unit used in the MD

integrator/program and output files, as follows

$$\begin{aligned} V &= \frac{M\ [\mathrm{kg/mol}]}{\rho\ [\mathrm{kg/m^3}]} \\ &= \frac{M\ [\mathrm{kg/mol}] \times \dfrac{1}{N_A\ [\mathrm{mol^{-1}}]}}{\rho\ [\mathrm{kg/m^3}] \times \left(10^{-10}\ [\mathrm{Å/m}]\right)^3} \\ &= \frac{M}{\rho} \times \frac{10^{-30}}{N_A}\ [\mathrm{Å^3}], \end{aligned}$$

from which we find the size we need our voxels to be as

$$L = \left(\frac{M}{\rho} \times \frac{10^{30}}{N_A}\ [\mathrm{Å^3}]\right)^{1/3}\ [\mathrm{Å}].$$

We then divide the system into voxels of length L. And put one water molecule with random orientation in the center of each empty voxel. The naive way of finding the empty voxels is to just find which voxel each existing silicon and oxygen atom is in, and mark those as occupied. But the amorph structure of solid silica means that we have a lot of very small pores inside the matrix, which ends up as empty voxels.

something about definition of a pore?

What we do is to assign a radius to each atom type , and mark all voxels with it's center within this radius as occupied.

which is hard, Si-O, ???

- Voxelize system – size depends on wanted density
- Mark all voxels within distance from other atoms as occupied
- Fill other voxels with H2O with random O-H orientation, but correct angle
- Improvement: Use one voxel size in the beginning (to avoid one-voxel pores), and then use a smaller voxel size when injecting water

Chapter 2

Fractures

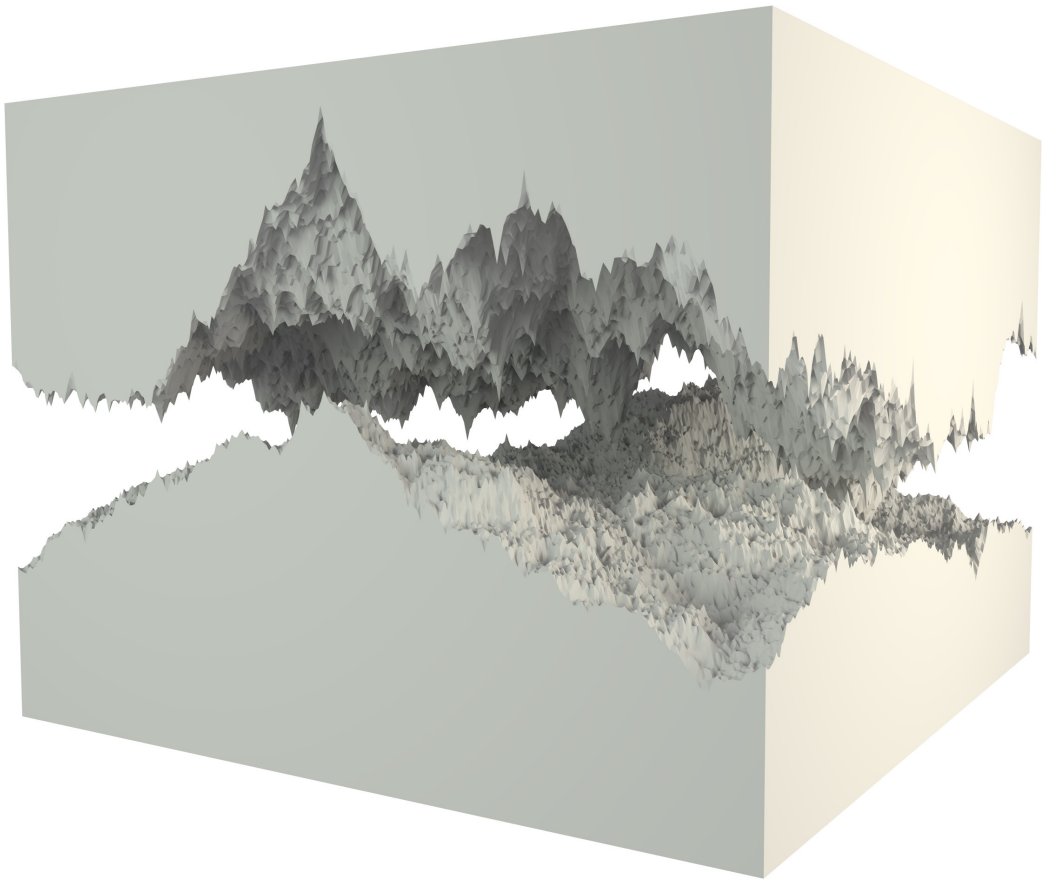


Figure 2.1: Caption

We want to study the behaviour of water trapped in nanoscale (pores and)

fractures in silica, so need want a way to generate and characterize such a structure. (Several methods of characterizing a fracture could be imagined (SOURCES, examples), and we will use several of them.)

terrain == heightmap??, finn bra ord her

2.1 Characterization

change wording? copied from Fractals...

An *affine transformation* transforms a point $\boldsymbol{x} = (x_1, \ldots, x_n)$ into new points $\boldsymbol{x}' = (r_1 x_1, \ldots, r_n, x_n)$, where the scaling rations $r_1, \ldots, r_n$ are *not* all equal.

A bounded set $\mathcal{S}$ is *self-affine* if $\mathcal{S}$ is the union of N non-overlapping subsets $\mathcal{S}_1, \ldots, \mathcal{S}_N$, each of which is congruent to the set $\boldsymbol{r}(\mathcal{S})$ obtained from $\mathcal{S}$ by the affine transform defined by $\boldsymbol{r}$. Here *congruent* means that the set of points $\mathcal{S}$ is identical to the set of points $\boldsymbol{r}(\mathcal{S})$ after possible translations and/or rotations of the set[5].

A set $\mathcal{S}$ is *statistically self-affine* if $\mathcal{S}$ is the union of N non-overlapping subsets each of which is scaled down by $\boldsymbol{r}$ from the original, and is identical in all statistical respects to $\boldsymbol{r}(\mathcal{S})$.

2.1.1 Hurst exponent

- define fractal dimension

To characterize a fractal one can use the Hurst exponent, usually called H[1], which comes from a statistical method developed by Hurst[9][8]. The Hurst exponent is related to the fractal dimension D by

$$D = d - H,$$

where d is the spatial dimension of the fractal's domain[5].

[1] The name used by Hurst in his work where he first describes the exponent was actually K[9][8]

The statistical method developed by Hurst is called *rescaled range analysis* and was designed for use on 1-dimensional time series $f(t)$. The method has been generalized to higher dimensions[4], but the original 1D form is shown here.

Rescaled range analysis

First the time series is divided into intervals of length τ. The average over each interval of length τ is

overlapping/non-overlapping?

$$\langle f \rangle_\tau = \frac{1}{\tau} \sum_{t=1}^{\tau} f(t).$$

We let F be the accumulated deviation from the mean

$$F(t,\tau) = \sum_{t'=1}^{t} \Big(f(t') - \langle f \rangle_\tau \Big).$$

The difference between the maximum and minimum of the accumulated deviation from the mean is the *range* R

$$R(\tau) = \max_{1 \le t \le \tau} \Big(F(t,\tau) \Big) - \min_{1 \le t \le \tau} \Big(F(t,\tau) \Big).$$

The standard deviation S of the time series is estimated using

$$S^2 = \frac{1}{\tau} \sum_{t=1}^{\tau} \Big(f(t) - \langle f \rangle_\tau \Big)^2.$$

Hurst found that the observed *rescaled range*, R/S, for many time series is described by the empirical relation[5]

$$\frac{R}{S} = \left(\frac{\tau}{2} \right)^H \sim \tau^H.$$

We now see that we can estimate the Hurst exponent by a linear fit of the form

$$\log \left(\frac{R}{S} \right) \sim H \log \tau,$$

where we find H as the slope of the linear fit.

Something about that it's hard to measure, and why.

Detrending moving average

To estimate the Hurst exponent of a surface we use a method called detrending moving average (DMA) first described by E. Alessio, A. Carbone et al.[1], and later generalized to higher dimensions by A. Carbone [3]. We use this method because ???

good results, easy to implement?

We define a self-affine) surface as $f(i,j)$, where f is the height in the point (i,j), defined in a discrete 2-dimensional domain with size $N \times N$, and with $i,j = 1,\ldots,N$. We divide the surface into square subsurfaces of size $n \times n$, and find the average $\tilde{f}_n$ of each subsurface by[2]

$$\tilde{f}_n(i,j) = \frac{1}{n^2} \sum_{k=-m}^{n-1-m} \sum_{l=-m}^{n-1-m} f(i-k, j-l) \tag{2.1}$$

$$= \frac{1}{n^2} \sum_{k=(i+m)-n+1}^{i+m} \sum_{l=(i+m)-n+1}^{i+m} f(k,l), \tag{2.2}$$

where

$$m = \lfloor n\theta \rfloor \qquad \text{for } 0 \leq \theta < 1,$$

which means that $0 \leq m \leq (n-1)$. θ is a parameter that controls the position of the square relative to the point (i,j) in $\tilde{f}_n(i,j)$. There are three extreme cases for θ, which are listed below, and are illustrated in fig. 2.2.

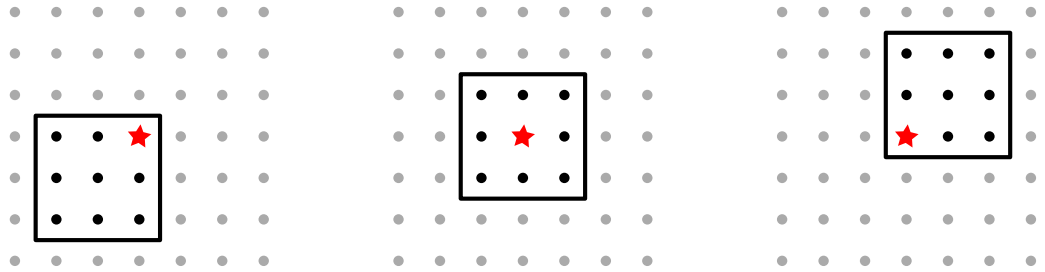
$\theta = 0$:
We have $m = 0$ and we average over a square with the point (i,j) in the square's upper right corner ($(i-n+1) \leq k,l \leq i$). See fig. 2.2a.

$\theta = 1/2$:
We average over a square centered on the point (i,j). See fig. 2.2b.

$\theta = (n-1)/n$:
We have the maximum value for m, $m = n-1$, and average over a square with the point (i,j) in the square's lower left corner ($i \leq k,l \leq (i+n-1)$). See fig. 2.2c.

~~THESE LIMITS ARE WRONG??~~ fixed

[2] eq. (2.1) is how the average $\tilde{f}_n$ is stated in the article[3], but we rewrite it to eq. (2.2) so it's easier to understand. The two forms are equivalent.



(a) $\theta = 0$ **(b)** $\theta = 1/2$ **(c)** $\theta = (n-1)/n$

Figure 2.2: Illustration of what the parameter θ controls in the detrending moving average method in 2 dimensions. The circles are points where the surface is defined, the red star is the point (i, j), and the black square encompasses the points averaged over to calculate $\tilde{f}_n(i, j)$ in eq. (2.2). The illustration uses $n = 3$.

We then define the generalized variance σ^2_{DMA}

$$\sigma^2_{\mathrm{DMA}} = \frac{1}{(N-n)^2} \sum_{i=n-m}^{N-m} \sum_{j=n-m}^{N-m} \left(f(i,j) - \tilde{f}_n(i,j) \right)^2,$$

where we see that $f(i, j) - f_n(i, j)$ is the difference between the point (i, j) and the average of a square of points (including the point itself) of size $n \times n$, as explained above (see fig. 2.2). The summation limits $(n - m) \leq i, j \leq N - m$ are set so that the averages $f_n(i, j)$ don't exceed the domain with size $N \times N$.

The generalized variance goes as

$$\sigma^2_{\mathrm{DMA}} \sim \left(2n^2\right)^H,$$

which we can use to find the Hurst exponent H, by calculating σ^2_{DMA} for different sizes of the squares, n. We estimate H by a linear fit of $\log \sigma^2_{\mathrm{DMA}}$ against $\log 2n^2$, where H is the slope.

In the paper that generalizes DMA to higher dimensions[3] they use different parameters for each spatial dimension d, $\boldsymbol{\theta} = \theta_1, \ldots, \theta_d$ and $\boldsymbol{n} = n_1, \ldots, n_d$, but for simplicity and to avoid strange results, we use $\theta_1 = \theta_2 = \theta$ and $n_1 = n_2 = n$.

implemented in Matlab/Octave

2.1.2 Surface area

2.1.3 Distance to nearest atom

- Fractals
- Fractional Brownian Motion
- The Hurst Exponent

2.2 Generating fractures

Stuff to define?

- fBm surfaces
- Hurst exponent
- Gaussian random variable ?
- non-stationary
- self-similar
- isotropic
- lacunarity

2.2.1 Generating terrains

To generate our random terrains we use an iterative midpoint displacement method usually called successive random additions (SRA). The method is based on a method proposed by Fourner in 1982[6], but with some modifications suggested by Voss[16, 17]. The method has further been discussed by Saupe[11], amongst others. We choose this method mainly because it's possible to generate periodic terrains with it, because it generates very good approximations to fBm surfaces[18], and because the Hurst exponent of the generated terrains is easy to control. The method is also easy to understand, easy to implement, and generates terrains with high resolution very fast. The

method is widely used in scientific applicationscite?? because of these properties, and is also used for generating terrains in computer graphics, since the terrains look very realistic with only some minor artefacts?? (high, narrow peaks).

Midpoint displacement method in 1 dimension

The method we use to generate random terrains is very similar to the standard midpoint displacement method, which in 1 dimension goes as follows. See fig. 2.3 for a visual presentation.

1. Give the values at the endpoints of the interval, y_0 and y_n, random values from a Gaussian random variable with mean $\mu = 0$ and variance σ_0^2. This initial standard deviation σ_0 can be chosen freely.

2. Generate the value in the center of the interval, $y_{n/2}$, by averaging over the two endpoints and adding a Gaussian random number with mean $\mu = 0$ and variance

$$\sigma_1^2 = (1/2)^{2H} \sigma_0^2,$$

   where H is the wanted Hurst exponent.

3. Continue generating the values in the center of each sub-interval until you reach the desired number of points, while reducing the variance of the random number by a factor $\frac{1}{2}$ each iteration. For iteration i we have

$$\sigma_i^2 = (1/2)^{i2H} \sigma_0^2.$$

Why 1/2? (needed to create good fBm, see [16])

This method generates a 1-dimensional line, with a Hurst exponent close to the input H. But since we only add random numbers to the new values, the result is non-stationary for $H \neq 0.5$[16], and it is neither self-similar or isotropic, as noted by Mandelbrot[10]. To mitigate this we implement the addition suggested by Voss[16], which consists of adding a random number to *all* new and old points in each iteration. Voss called this modified method *successive random additions*.

how close?

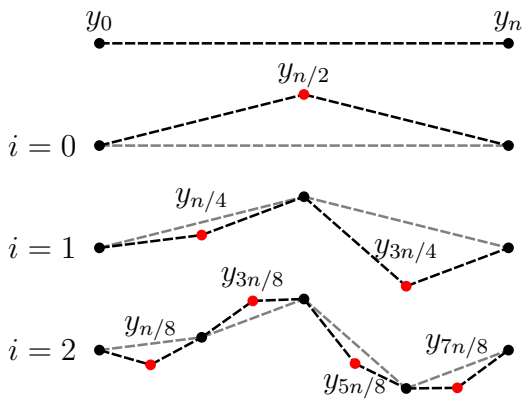


Figure 2.3: Illustration of the midpoint displacement method in 1 dimension.

Successive random additions on an infinite grid in 2 dimensions

As mentioned above, the method called *successive random additions* is a modification of the regular midpoint displacement method first described by Richard F. Voss[16], where we add random numbers to *all* points in each iteration, compared to just the *new* points in the regular midpoint displacement method.

Voss has geneneralized the the method of successive random additions to higher dimensions[16], which is the algorithm we use to generate fractures. We use it to generate terrains in the form of heightmaps, meaning a 2-dimensional grid of points (i, j), with a value for the height in each point $(z_{i,j} = z(i, j))$ which is generated by the algorithm.

To explain the algorithm we start with a simple case of an infinite grid of evenly spaced points, all with known z-values. This allows us to first focus on the central part of the algorithm which consists of two steps; often called the *diamond-step* and the *square-step*. We start with a set of 2-by-2 points in the grid, as seen in the leftmost square of fig. 2.4a.

The *square-step*: We have four points that make up a square. The z-value in the center of the square is generated by averaging the z-values of the four corners of the square, as indicated by the red dots in fig. 2.4a, and adding a random Gaussian number with mean $\mu = 0$ and variance σ_0^2 to both the new point and the old points. This initial standard deviation σ_0 can be chosen freely.

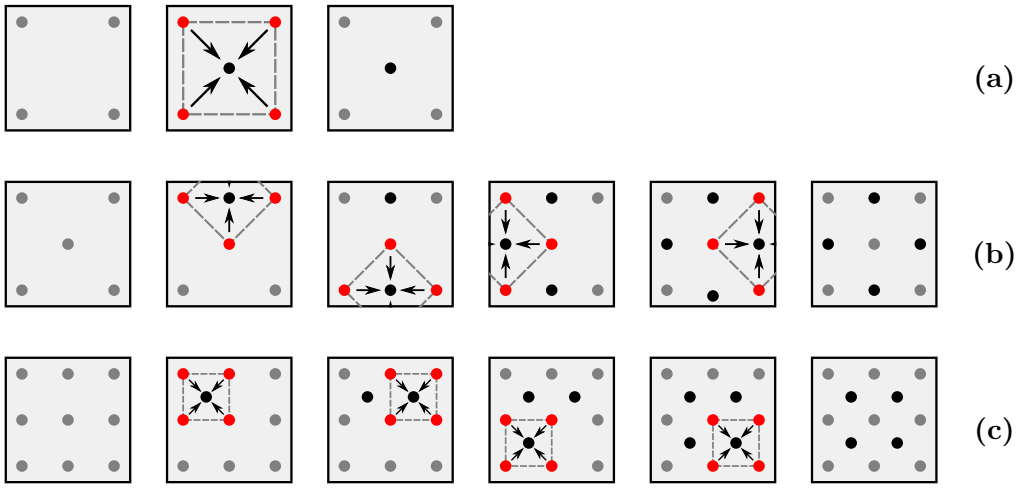


Figure 2.4: Illustration of the different steps of the diamond-square algorithm. New points are black, old points are grey, and points used in the generation of the new points are red. **(a)** The initial square-step, **(b)** the diamond-step, **(c)** the square-step.

The *diamond-step*: The z-values in the middle of the four edges of the square is generated by averaging the z-values of the points above, below, to the right of, and to the left of the points, as indicated by the red dots in fig. 2.4b, and adding a random Gaussian number with mean $\mu = 0$ and a *reduced* variance $\sigma_1^2 = \sigma_0^2 r^{2H}$, where H is the wanted Hurst exponent, and $r < 1$ is a parameter that controls the *lacunarity* of the fractal, to all new and old points.

We now see that we can apply the square step four times on the resulting grid (see the rightmost square in fig. 2.4b), as illustrated in fig. 2.4c, and then keep increasing the resolution by alternating between the two methods. For step i of the algorithm the variance of the Gaussan variable becomes

$$\sigma_i^2 = \sigma_0^2 \left(r^n\right)^{2H}, \tag{2.3}$$

where σ_0^2 is the initial variance.

Since we add a random number to all points in each step of this algorithm, we can change the variance of the random number with any factor $r < 1$ each step.

Successive random additions on a finite grid in 2 dimensions

We obviously can't generate infinite terrains, instead we generate a terrain on a finite grid of size $n \times n$.

This means that we get some special cases that need to be addressed near the boundaries of the grid. We also want the generated terrains to be periodic, meaning that they should behave nicely when repeated in the x- and y-direction.

When generating the points on the edges of the grid using the diamond-step as in fig. 2.4b we see that some of the neighbors we need to average over will be outside our finite grid. Since the system is periodic, we solve this by wrapping around the edges to reach the neighbor. For example, when generating the point in the middle of the top edge $z_{0,n/2}$, the neighbor above the point, $z_{0-n/2,n/2}$, will be outside the grid in the x/i-direction. Using periodic boundary conditions we find that the point we want is $z_{n/2,n/2}$.

This leaves us with the following algorithm for generating a periodic terrain which approximates a 2-dimensional fractional Brownian motion with Hurst exponent H

1. Start with a grid of size $n \times n$, where $n = 2^p + 1$, and p is a positive integer.

2. Generate the z-value in the center of each 2-by-2 square of assigned points by using the square-step, as in fig. 2.4c. Add a random Gaussian value with mean $\mu = 0$ and variance σ_0^2 to all new and old points. The initial standard deviation σ_0 can be chosen freely.

3. Generate the z-value in the center of each

4. Generate the z-value in the center of the grid using the square step by averaging over the four corners, as shown in fig. 2.4a. Add a random Gaussian number with mean $\mu = 0$ and variance $\sigma_1^2 = \sigma_0^2 r^{2H}$ to the new point, and to the four corners (add the same number to all four corners to keep the system periodic).

define r

5. Generate the values in the middle of the top and left edge of the grid using the diamond step, as shown in fig. 2.4b. Add a random Gaussian number with mean $\mu = 0$ and variance $\sigma_2^2 = \sigma_1^2 r^{2H}$ to all new and old

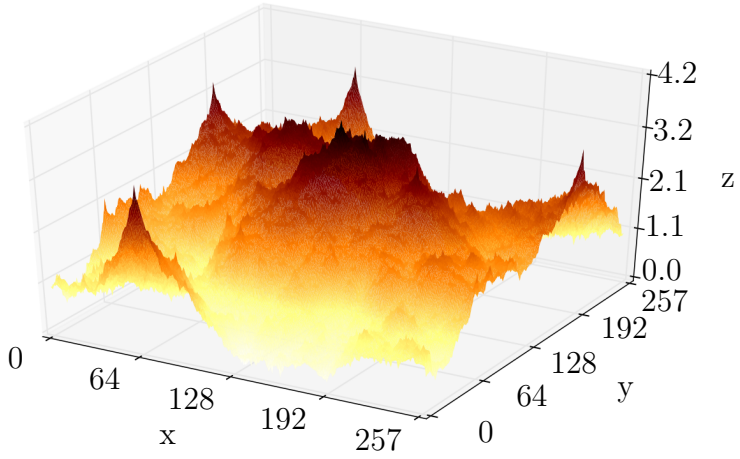


Figure 2.5: Caption.

points. Set the point in the middle of the right and bottom edges equal to

When calculating the averages in the diamond-step Since the terrain is periodic, we can wrap around the edges to reach the points beyond the edges of our grid (the points outside the grid in fig. 2.4b). Set values in the middle of the right and bottom edge equal to the corresponding values in the middle of the left and top edge, to keep the terrain periodic[3].

6. Continue generating new points and increasing the resolution of the terrain/grid until you reach the desired number of points/resolution. We see in fig. 2.4c that the square step doesn't need any changes, since we don't generate any points at the edges of the grid, and we never have any neighbors that are beyond the edges of the grid. For the diamond step we need to make sure we wrap around the edges to find neighbors when calculating points that are close to the edges of the grid, and that we update the left and bottom edges as needed.

   For step i the variance of the random Gaussian number is

$$\sigma_i^2 = \sigma_0^2 (r^i)^{2H}.$$

See fig. 2.5 for a terrain generated by the algorithm.

[3] We could have done this in a slightly different way, by making the grid one unit smaller, and starting with only one point in one of the corners, but by doing it the way we have we can keep the square step the same as before.

Notes:

- Size of surface $(2^n + 1)$
- Edges / bottom/top edge

Stuff:

- Diamond-square – why and how
- Periodic/non-periodic
- Lacunarity

Implementation (in Matlab/C++)?

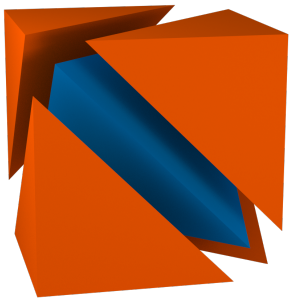
2.2.2 Generating fractures from terrains

To generate a realistic fracture we use the method of successive random additions, as described previously, to generate random heightmaps with a known Hurst exponent. If we generate two of these terrains we can make a fracture by putting one terrain above the other, and letting either the space between or outside the heightmaps be the void. Since we are using periodic systems the two different approaches are equivalent, but we chose to let the space between the heightmaps be the void, for easier visualization.

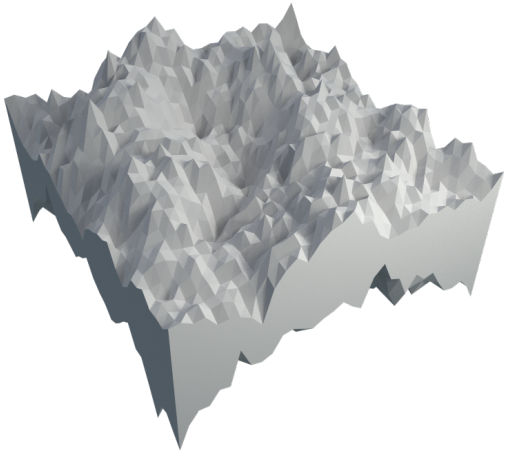
In practice we make a fracture using the following procedure

- Generate two heightmaps.
- Rescale the x- and y-positions of all points in the heightmaps, so they span the size of the atomic system.
- Rescale the z-values of the heightmaps so all points are inside the system.
- Remove all atoms between the upper and lower heightmap.

Removing all atoms between the two heightmaps isn't a trivial task. We do this by utilizing the fact that the points in each heighmap lie on a regular grid in the x-y-plane, to divide the volume into tetrahedra. If the two heightmaps



(a) Illustration of how to divide a convex hexahedron into five tetraheda.



(b) A random fracture made from two periodic heightmaps.

are not intersecting, we see that we can divide the volume between them into convex hexahedra, one for each set of points (x_i, y_i), (x_i, y_{i+1}), (x_{i+1}, y_i), and (x_{i+1}, y_{i+1}) in the x-y-grid. Each of these hexahedra can then be divided into five tetrahedra, as illustrated in fig. 2.6a.

Finding a point inside a tetrahedron

A tetrahedron consists of four points $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{d}$ and four faces, spanned by the four possible combinations of the four points. For a face spanned by the points $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ we can find if a point $\boldsymbol{P}$ is on the same side of the face as the point $\boldsymbol{d}$ (the point not used to construct the face) by doing some geometry. We calculate the cross product

$$\boldsymbol{n} = (\boldsymbol{a} - \boldsymbol{c})(\boldsymbol{b} - \boldsymbol{c}),$$

which gives us the normal vector to the surface $\boldsymbol{n}$. We then find the sign of dot products

$$\boldsymbol{n} \cdot (\boldsymbol{P} - \boldsymbol{c}),$$
$$\boldsymbol{n} \cdot (\boldsymbol{d} - \boldsymbol{c}).$$

If the sign of these dot products is the same, we know that the point $\boldsymbol{P}$ is on the same side of the face as the point $\boldsymbol{d}$. We now see that if we do this for all four faces of the tetrahedra, we know that the point $\boldsymbol{P}$ is inside the tetrahedra if the signs of all pairs of dot products are equal.

Chapter 3

Measurements

3.1 Voxelation, calculating distances, finding neighbors, neighbor lists, periodicity tricks

3.2 Mean square displacement

3.3 Density

3.4 Distance to atom

3.5 "Generation matrix"

3.6 "Voxel counter"

A histogram of the fraction of voxels that has one or more atom in them vs. the voxel size in x-, y-, and z-direction.

3.7 Cage cage correlation

3.8

Part III

Results

3.9 Mean square displacement

3.10 Distance to atom

3.11 Area

3.12 Volume?

Part IV

Discussion

Part V

Appendices

Bibliography

[1] E. Alessio, Anna Carbone, G. Castelli, and V. Frappietro. Second-order moving average and scaling of stochastic time series. *The European Physical Journal B - Condensed Matter*, 27(2):197–200, May 2002.

[2] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Clarendon Press, 1989.

[3] Anna Carbone. Algorithm to estimate the Hurst exponent of high-dimensional fractals. *Physical Review E*, 76(5):056703, November 2007.

[4] Jie Fan. Rescaled Range Analysis in Higher Dimensions. *Research Journal of Applied Sciences, Engineering and Technology*, 5(18):4489–4492, 2013.

[5] J. Feder. *Fractals*. Physics of Solids and Liquids. Springer, 1988.

[6] Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, 1982.

[7] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation*. Academic Press, Inc., Orlando, FL, USA, 2nd edition, 2001.

[8] H. E. Hurst. Long-term storage capacity of reservoirs. *American Society of Civil Engineers*, 116:770–808, 1951.

[9] H. E. Hurst, R. P. Black, and Y. M. Simaika. *Long-Term Storage: An Experimental Study*. Constable, 1965.

[10] Benoit B. Mandelbrot. Technical correspondence: Comment on computer rendering of fractal stochastic models. *Commun. ACM*, 25(8):581–583, August 1982.

[11] Dietmar Saupe. Algorithms for random fractals. In Heinz-Otto Peitgen and Dietmar Saupe, editors, *The Science of Fractal Images*, chapter 2,

pages 71–113. Springer-Verlag New York, Inc., New York, NY, USA, 1988.

[12] William C. Swope. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics*, 76(1):637, 1982.

[13] J. M. Thijssen. *Computational Physics*. Cambridge University Press, 1999.

[14] Priya Vashishta, Rajiv K. Kalia, José P. Rino, and Ingvar Ebbsjö. Interaction potential for SiO2: A molecular-dynamics study of structural correlations. *Physical Review B*, 41(17):12197—-12209, 1990.

[15] Loup Verlet. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, 159(1):98–103, 1967.

[16] Richard F. Voss. Random fractal forgeries. In Rae A. Earnshaw, editor, *Fundamental Algorithms for Computer Graphics*, volume 17 of *F*, chapter 8, pages 805–835. Springer-Verlag, 1985. Proceedings of the NATO Advanced Study Institute on Fundamental Algorithms for Computer Graphics held at Likley, Yorkshire, England, March 30 - April 12, 1985. See also [17].

[17] Richard F. Voss. Fractals in nature: From characterization to simulation. In Heinz-Otto Peitgen and Dietmar Saupe, editors, *The Science of Fractal Images*, chapter 1, pages 21–70. Springer-Verlag New York, Inc., New York, NY, USA, 1988. See also [16].

[18] Guiyun Zhou and Nina S.-N. Lam. A comparison of fractal dimension estimators based on multiple surface generation algorithms. *Computers & Geosciences*, 31(10):1260–1269, December 2005.