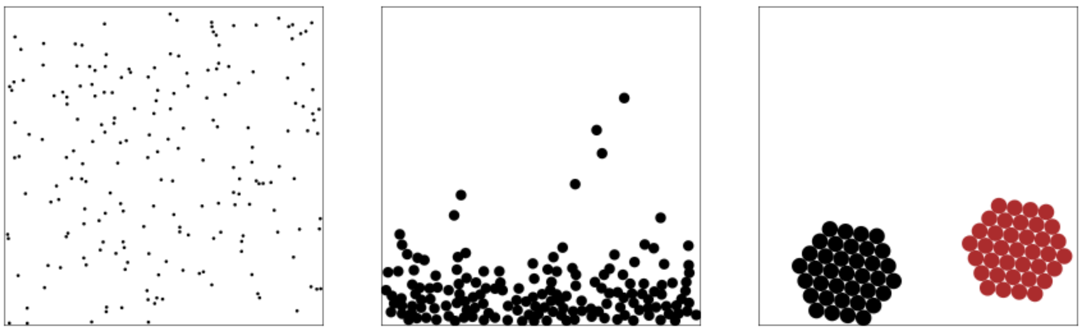
CHALMERS
UNIVERSITY OF TECHNOLOGY



Many Tiny Things

Explorable Explanations on Statistical Physics

Master's thesis in Complex Adaptive Systems

PONTUS GRANSTRÖM

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016

Many Tiny Things

Explorable Explanations on Statistical Physics

PONTUS GRANSTRÖM

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2016

Many Tiny Things
Explorable Explanations on Statistical Physics
PONTUS GRANSTRÖM

Supervisor and Examiner: Göran Wahnström, Department of Physics

Master's Thesis 2016
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Screenshot of interactive simulations of a gas, a liquid and a solid. These simulations are featured in the explorable explanations described in the report.

Typeset in LaTeX
Gothenburg, Sweden 2016

Many Tiny Things
Explorable Explanations on Statistical Physics
PONTUS GRANSTRÖM
Department of Physics
Chalmers University of Technology

Abstract

This thesis report describes the development of *Many Tiny Things*, a series of *explorable explanations* on basic statistical physics. The *explanations* guide the reader through a molecular dynamics model, which is simulated and visualised on the reader's computer. The reader can interact with the simulated model, and *explore* concepts such as energy, temperature, friction, entropy and state transitions. The target audience is people with little physics or mathematics background, and the focus is on conveying an intuitive understanding, not a quantitative one. The goal is for the reader to play and experiment with the model, and hopefully learn something new about the world in the process.
In the report I detail the design of the simulated model and the reader's interaction with it, as well as the way the guiding explanations are presented. I cannot offer any conclusive general wisdom about explorable explanations, but the discussion on my design process could prove interesting for future projects similar to this one. The explanations are implemented as web pages, which has both benefits and drawbacks. A web page is easy to access and reference, but it is ill-suited for making performant programs, which is needed for many real-time simulations. There are many challenges in writing interactive texts, among them a lack of tools and experience. I suggest possible solutions to issues I have encountered and directions for further developing explorable explanations.

Keywords: teaching, learning, pedagogical, simulation, explorable explanation, statistical physics, thermodynamics, molecular dynamics

Acknowledgements

I want to thank my supervisor and examiner, Göran Wahnström, for his interest in the project and his excellent advice on the physics and simulations. Moreover, I want to thank everyone who has helped me test the explanations, providing invaluable feedback. Finally, I'd like to thank my family and friends for supporting and encouraging me in this work.

Pontus Granström, Gothenburg, December 2016

Contents

1. Introduction

Physics, like most academic subjects, is usually taught by textbooks and lectures. The written word is a very efficient form for communicating knowledge en masse. A book is written once and can be read by millions of people, each at their own pace. While books are great as a complete and detailed reference, they can struggle at giving readers an intuitive understanding. Pictures usually help, but because the medium is inherently static, it is not ideal for representing motion, change or procedures. Furthermore, books are a form of one-way communication, where the reader is passively absorbing the author's words.

In most courses, textbooks are complemented by lectures. Lectures, in contrast to books, are not static, and tend to be better at conveying an intuition for the subject. A lecturer has access to many forms of expression impossible in a book, such as drawing pictures, gesticulating, using sound, showing video, or even performing experiments on stage. Additionally, a lecture is a two-way communication, where the audience can ask questions or request clarification, and the lecturer can check if the audience is following the argument.

Modern computers and the internet open up new possibilities for teaching, and both textbooks and lectures are represented online. The internet lets readers download textbooks as digital e-books, and some are presented in web-page form. Wikipedia, one of the most visited websites on the internet, contains vast amounts of knowledge, connected together with hyperlinks. Video of lectures from many universities can be watched on websites like YouTube, and Massive Open Online Courses (MOOCs) hope to bring knowledge to even more people. These initiatives make good use of the internet for mass communication and ease of access. However, they are mostly very similar to a traditional textbook or lecture course, and do not take full advantage of the interactivity offered by a modern personal computer.

Dynamical systems, and physical systems in particular, can be simulated on a computer. Historically, this has been one of the main uses of supercomputers or big computer clusters, but today, even personal computers can be used to simulate non-trivial systems. Consider a simulation of a dynamic model running on a personal computer, for example a model of the solar system. Visualising the simulated model with computer graphics is a great way of aiding human understanding of the system, and videos of such visualisations are commonly used in teaching.

However, dynamic models are capable of more than just being shown as a video. If a student is given the ability to interact with the simulation, she can *explore* the model. In the case of a solar system, she might be able to change the size of planets, change their spin, perturb the trajectories, add more planets, and so on. She can then ask questions and answer them by experimenting with the simulation, or just play around and maybe discover something on her own. An explorable model such as this has the potential to be a useful tool for both teaching and learning.

This thesis project is my attempt at creating a series of **explorable explanations** with the name *Many Tiny Things*, based on a simulation of the *many-tiny-particles* model of statistical physics. In other words, I combine the *explorable* simulation with *explanatory* text that guides the reader through the model and presents the fundamental concepts of statistical physics, such as energy, entropy and pressure. The hope is to leverage the interactive model to explain such abstract concepts in a simple way, and keep the reader active, exploring and experimenting.

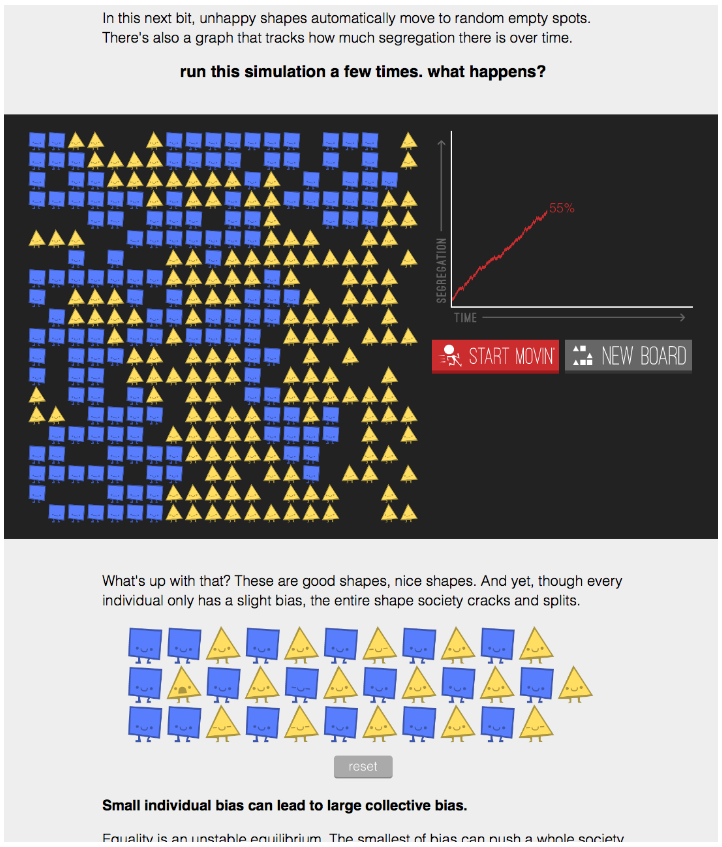
1.1. Previous work

The term *explorable explanation* was coined by Bret Victor in his essay *Explorable Explanations* [1], where he presents some ideas and examples on how interactivity can be used to make the reader of a text more active. Furthermore, Victor has a number of essays, talks and prototypes about media for learning and thinking, available on his website [2]. Victor's essays has inspired others to make explorable explanations, two of which are particularly relevant to this project.

Parable of the Polygons [3] by Vi Hart and Nicky Case (figure 1.1a), is about how systemic segregation can arise from individual choices, and is based on Thomas Schelling's 1971 paper *Dynamic Models of Segregation* [4]. The explanation uses an abstract and very simple model, with a discrete grid of individuals that move randomly depending on who their neighbors are. The reader is invited to explore the model and its rules by moving around the individuals manually. After establishing the basic rules, the simulation is run automatically, and the reader can instead influence the rules and see what effect it has.

Considerably larger in scope is *Earth: A Primer* by Chaim Gingold [5], an interactive book on geography, primarily aimed at school children. It is based on a phenomenological model of the earth, accounting for topology, temperature, water, erosion, cloud formation, and vegetation. In contrast to *Parable*, which starts with a very simple model, and explores its consequences, *Earth Primer* seems created to explain some certain phenomena, with a model tailored to fit those phenomena. It presents its reader with many tools (making rain, raising or lowering terrain, moving sediment, changing temperature) - about one for each phenomenon explained. This makes the explanations very concrete, but not as suprising and interesting as *Parable.*

Both *Parable* and *Earth Primer* serve as inspiration for Many Tiny Things, and it shares a number of similarities with them. I will refer to both of them as I discuss issues of presentation in chapter 4 and in the final discussion in chapter 6.


In this next bit, unhappy shapes automatically move to random empty spots. There's also a graph that tracks how much segregation there is over time.

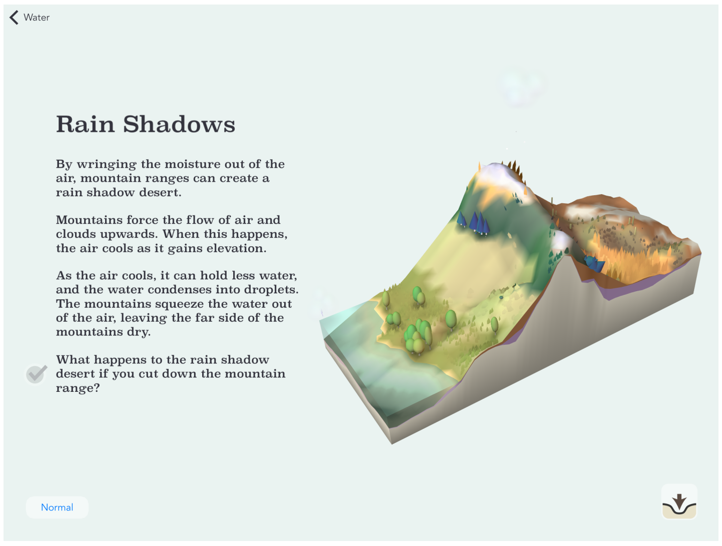
run this simulation a few times. what happens?

What's up with that? These are good shapes, nice shapes. And yet, though every individual only has a slight bias, the entire shape society cracks and splits.

Small individual bias can lead to large collective bias.

Equality is an unstable equilibrium. The smallest of bias can push a whole society

(a)



Rain Shadows

By wringing the moisture out of the air, mountain ranges can create a rain shadow desert.

Mountains force the flow of air and clouds upwards. When this happens, the air cools as it gains elevation.

As the air cools, it can hold less water, and the water condenses into droplets. The mountains squeeze the water out of the air, leaving the far side of the mountains dry.

What happens to the rain shadow desert if you cut down the mountain range?

(b)

Figure 1.1.: Excerpts (screenshots) of (a) *Parable of the Polygons*, and (b) *Earth: A Primer*, two explorable explanations that inspired *Many Tiny Things*.

1.2. Purpose and Scope

The purpose of this thesis project is to **create a series of interactive explanations on statistical physics**. More specifically: How everything is made from many tiny things and the consequences thereof. I had a number of goals for the project:

- **Give the reader an intuition for the microscopic world.** For example: how heat and sound spreads, or what melting or evaporating really means.
- **Use a single, simple model with interesting consequences.** The more phenomena can be explained with a single scientific theory, the more powerful that theory is. I want to convey this by using a single model as much as possible.
- **Encourage curiosity, experimentation and play.** If I can make reader active, hopefully she will learn more.
- **Aimed at curious people with little background in physics or mathematics.** Not specifically aimed at children, but might work for those that can read. No mathematical formulas.
- **Easy to access.** *Getting to* the explanations should be as simple as possible.

Practically, this means designing and implementing a suitable molecular dynamics model, designing how the model is presented, and writing explanations making use of model and presentation. The project merges several disciplines: physics, programming, user interface design and pedagogy. However, there are limits to the scope of the project. It is **not** about:

- **Measuring the effectiveness of the explanations.** The project is only concerned with the *creation* of the explanations.
- **Comprehensibly covering the subject of statistical physics.** The project is to be seen as a prototype perhaps complementing a course, more than as a textbook to base a course on.
- **Creating a framework for making more explorable explanations.** Although, it might be able to serve as a starting point for such a framework.

The result of the project should be seen as a **prototype**, more than a **finished product**, and I aim to keep working on it after the project is done.

1.3. Reading this Report

Before you continue reading, I *highly advise you* to **try Many Tiny Things** before reading this report. The explanations that are finished are available at

http://manytinythings.github.io

Make sure to use a computer with a mouse and a modern web browser.

This report describes the final design of the explorable explanations, the choices I have made, as well as the reasons behind them. Throughout the report, I will refer to the intended audience of the *explanations* as **the reader** (it does not refer to you who is reading this).

Chapter 2 describes the technical details of the simulated model, and is intended for people with knowledge of statistical physics. It is followed by chapter 3, where I talk about how the reader interacts with simulation. These two chapters are specific to the subject of Many Tiny Things, and are probably not applicable to explorable explanations on other subjects.

Chapter 4 is about presentation: how the exploratory and the explanatory fits together and is presented to the reader. It is the chapter most interesting to a general audience. Chapter 5 describes the implementation of the explanations, and is primarily written for someone who plans to make their own explorable explanation. Finally, I discuss the project as a whole, and talk about how it could be improved further in chapter 6.

2. Model

The explorable explanations are based on a molecular dynamics model, which is simulated and visualised on the reader's computer. A few examples is shown in figure 2.1. The model is designed with a few goals in mind. It should be:

- **Qualitatively physically correct**, meaning it should exhibit real world behavior, e.g. state transitions, energy is conserved, entropy increases, etc. so that these concepts can be explained properly. However, it does not have to be *quantitatively accurate*, since the point is not to, as an example, predict the boiling temperature of aluminium from its microscopic properties.
- **As simple as possible**, so that a concept can be explained with minimal potential for confusion, but not any simpler than is needed to represent the concept accurately.
- Able to simulate **many particles in real-time**. The more particles that can be simulated, the better I can demonstrate collective behavior. This requires the simulation to be optimised to run fast on the reader's computer.

To summarize: the model should be **simple**, allow **many particles**, and be **qualitatively accurate**. However, there is one qualitative inaccuracy: the simulations are two-dimensional (2D), not three-dimensional (3D) like the real world. This sacrifices some interesting properties such as 3D crystal structure, but a 2D model has two significant advantages over a 3D model. First, since computer screens are 2D, a 2D model is easier to visualize, and much more legible. In a 3D simulation, particles overlap, and they are hard to visualize spatially without moving the camera around. Requiring the user to control the camera in turn increases the complexity of the interface, taking focus off

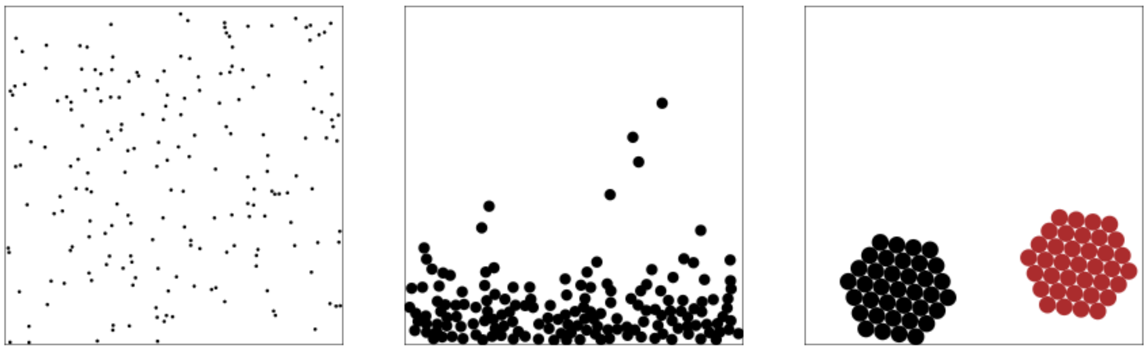


Figure 2.1.: Visualisations of the model described in chapter 2, simulating a gas, a liquid and two solids.

the important part: interacting with the particles. Second, filling 2D space requires much fewer particles than filling 3D space, which facilitates simulating (seemingly) larger systems.

2.1. Particles and Walls

The model consists of *particles* and *walls*, as seen in figure 2.1. Particles are represented by discs that move around in 2D space, interacting with other particles, and bouncing off walls. Walls are stationary line segments, used to restrict the movement of the particles.

There are $N_{\text{particles}}$ particles, each described by

- an index k, $0 \leq k < N_{\text{particles}}$,
- a position vector $\mathbf{x}_k$,
- a velocity vector $\mathbf{v}_k$,
- a radius r_k,
- a mass m_k,
- a type T_k,
- and a color c_k, to identify different types of particles.

There will be many identical particles, differing only in index and dynamic variables ($\mathbf{x}_k$, $\mathbf{v}_k$). The type T_k distinguishes between different types of identical particles, which is primarily used for interactions between particles. Instead of having to specify the interaction between every pair of particles, I specify it between every pair of types. When two particles interact, we look up their types, and use the types to use the corresponding interaction.

Note that the particles are rotationally symmetric, which lets me ignore the orientation of the particles. The model is thus only concerned with the linear momentum of the particles, and does not incorporate their angular momentum (spin).

A wall is an (infinitely thin) line segment, described by

- a starting point $\mathbf{a}$,
- a vector $\mathbf{b}$ describing the extent of the wall.

Walls do not move, which means they have infinite mass $m_{\text{wall}} = \infty$ and zero velocity $\mathbf{v}_{\text{wall}} = 0$.

2.2. Numerical integration

The simulation evolves in time according to Newton's second law, $\mathbf{F} = m\mathbf{a}$, approximated using numerical integration. I use the *velocity Verlet* integration method [6] with fixed timestep Δt. It is implemented as follows:

Algorithm 1: Velocity Verlet

```
for each particle do
    $\mathbf{v}_k \leftarrow \mathbf{v}_k + \mathbf{a}_k \Delta t/2$
    $\mathbf{x}_k \leftarrow \mathbf{x}_k + \mathbf{v}_k \Delta t$
Update the accelerations $\mathbf{a}_k \leftarrow \mathbf{F}_k/m_k$
for each particle do
    $\mathbf{v}_k \leftarrow \mathbf{v}_k + \mathbf{a}_k \Delta t/2$
```

where $\mathbf{F}_k$ is the force on each particle. The force is not necessarily calculated independently for each particle, since it can come from particle-particle interactions, a global field, or user interaction. More on this later.

The Verlet method is the preferred method in molecular dynamics for several reasons [7]:

- It is simple to implement.
- The global error is $\mathcal{O}(\Delta t^2)$, which is small considering how simple the algorithm is.
- It needs only one evaluation of the acceleration for each timestep. Calculating the acceleration usually takes the majority of the simulation time.
- It is time-reversible, which means the energy does not drift for periodic motion.
- It is symplectic, which means it conserves energy well for conservative systems in general.

Any continuous-time conservative system is symplectic, but discrete-time integration methods generally are not. A symplectic system preserves areas in phase space for each pair of position-momentum coordinates, (x, p_x), (y, p_y), etc. Consider several copies of the system in different states, each represented by a point in phase space. These points enclose an area, and if the system is symplectic, that area will remain the same (but the shape might change). A symplectic integrator, such as velocity Verlet, preserves this property as well, even though it is discrete in time rather than continuous. It also implies the conservation of a discrete analogue to the energy, which is not exactly the same as the real energy, but stays within Δt^k, for $k \in \mathbb{N}$. Appendix A describes a simple method for deriving symplectic integrators, including the velocity Verlet integrator.

Because of the time-reversibility of the velocity Verlet method, errors in energy cancel out over one period in an oscillating system, and there is no energy drift over time. However, the period of a molecular dynamics simulation is, for all intents and purposes, infinite. Nonetheless, the energy is approximately conserved, because it has to stay close to its discrete analogue, implied by symplecticity.

Note that, while the *algorithm* is time-reversible and conserves energy *in exact arithmetic*, in practice, the finite precision of floating point arithmetic has to be taken into account. The floating point errors in the energy average out, so energy conservation still holds approximately. However, time-reversibility is broken, because floating point arithmetic does not give the exact starting result when inverted, and the errors compound quickly.

2.3. Units

The model is used to illuminate *qualitative* concepts, and is not concerned with quantitative results. Therefore, I choose simple units with no relation to real-world measurements. The standard radius of particles is $r = 1$ and the standard mass is $m = 1$. Constants, such as Boltzmann's constant k_B, are also set to unity. Choosing units so that most values are close to unity have the benefit of improving floating point accuracy, by keeping calculations well within the floating point range. The size of the particle box, strength of forces, etc., are adjusted for each scenario, to facilitate interaction by the reader.

The visualisation of the simulation runs with a variable framerate, with a target of 60 frames per second (matching most computer monitors). For each frame of animation, the simulation is run a number of timestep decided by the simulation speed and the timestep Δt. Simulation time typically runs at a rate of 5 simulation units per real-world second, but can be changed to slow down or speed up the time perceived by the reader, or to make sure that the simulation can keep up with real-time visualisation. The timestep Δt is chosen to ensure stability (energy conservation), but has to be balanced so that the simulation can run in real-time without stuttering.

2.4. Measurements

In explaining energy, temperature, pressure and entropy, I want to show how they vary over time. Consequently, I need to measure these properties. To compare and contrast different values of, say, energy, I need to be able to measure the energy of two different parts of the simulation. Thus, I need to make measurements not only globally, but also locally. Local measurements are done by saving data for each individual particle and wall. Then, for example, if I want to measure the energy in a certain area, or for all particles of a certain type, I simply add up the contributions of all particles fulfilling the criteria.

2.4.1. Energy

The kinetic energy of each particle is calculated from its velocity as

$$E_{\text{kin},k} = \frac{1}{2} m \mathbf{v}_k^2.$$

The potential energy is not usually attributed to a specific object, but since all interactions are pairwise and symmetric, I attribute half the potential energy of the interaction to each involved particle. The potential energy attributed to particle k is

$$E_{\text{pot},k} = \sum_{l \neq k} \frac{V_{kl}}{2}$$

2.4.2. Temperature

By the equipartition theorem, each quadratic energy term contributes a factor $k_B T/2$ to the average energy. For the forces I use, only the kinetic energy terms are quadratic, and there is one for each spatial dimension, so in 2D

$$T = \frac{1}{k_B N_{\text{particles}}} \sum_k E_{\text{kin},k}$$

which I use as the instantaneous temperature. The instantaneous temperature tends to fluctuate, so when it is presented, I smooth it by using a rolling average.

2.4.3. Pressure

In 3D, pressure is the perpendicular force on a surface per unit area. In 2D, there are no surfaces, only curves, so the definition is changed to read "per unit length" instead of "per unit area".

I calculate the instantaneous pressure P on the walls by summing the perpendicular parts of the forces $\mathbf{F}_i$ acting on each wall, and dividing by the length L of the wall.

$$P = \frac{1}{L} \sum_i (\mathbf{F}_i \cdot \hat{\mathbf{n}}),$$

where $\hat{\mathbf{n}}$ is the wall normal. In a hard collision (section 2.8.2), the force on the wall is $\mathbf{F}_i = \mathbf{I}_i/\Delta t$, where $\mathbf{I}$ is the impulse of the collision.

The pressure fluctuates even more than the temperature, especially in macroscopic mode where it consists of only impulses. To smooth it, long rolling averages are needed.

2.4.4. Entropy

The entropy S is defined as

$$S = -k_B \sum_i p_i \ln p_i, \tag{2.1}$$

where p_i is the probability that microstate i is occupied. If I know the probability of each microstate, and assume that there are very many such states, the entropy can be calculated by integrating over phase space.

In a numerical simulation, it is hard to count the number of microstates this way. Instead, I estimate the entropy using a histogram method, by defining a number of *classes*, or *bins*, e.g. by dividing the simulation space into regions, or by assigning the particles different colors. The p_i in equation (2.1) are then interpreted as the probability that a particle belongs to class i, and estimated by

$$p_i = N_i / N_{\text{particles}}$$

where N_i is the number of particles belonging to that class, and $N_{\text{particles}}$ is the total number of particles. This estimation of the entropy behaves qualitatively just like the proper entropy, and works well for demonstrating the concept.

2.5. Thermostat

To explore heat and temperature, the reader needs to be able to control the temperature of the simulation. A simple way is to let the reader control a velocity scaling parameter α

$$\mathbf{v}'_k = \alpha \mathbf{v}_k$$

but this is problematic. The scaling is exponential, which works well for $\alpha < 1$, where it behaves just like friction. However, for $\alpha > 1$, the temperature increases exponentially, which is unintuitive and quickly leads speeds beyond what the numerical integration can handle.

A better solution is using a thermostat, where the user sets a target temperature and the velocities are gradually changed to reach that temperature, in a more realistic way. Having the reader control a target temperature instead of the scaling also has the benefit of letting me specify a maximum temperature below which I know the system is stable.

2.5.1. Deterministic thermostat

One way of implementing a thermostat is letting the temperature difference $\Delta T = T_{\text{desired}} - T$ decay exponentially

$$\Delta T(t) = \Delta T(0) e^{-t/\tau}$$

which can be realised by scaling the the velocities like this every timestep

$$\mathbf{v}'_k = \sqrt{1 + \frac{\Delta t}{\tau} \frac{\Delta T}{T}} \, \mathbf{v}_k$$

This thermostat is not physical, since we are precisely adjusting the individual velocities of the particles. Nonetheless, it works well in situations where the random walks of the Langevin method below is undesirable.

2.5.2. Langevin thermostat

A more physically accurate way of changing the temperature is by introducing two new force terms acting on the particles: a random-fluctuation term and a dissipation term [8]. The forces give us a differential equation, Langevin's equation, for the velocity

$$\frac{d}{dt}\mathbf{v}(t) = -\eta\mathbf{v}(t) + \sqrt{\frac{2\eta k_B T_{\text{desired}}}{m}}\,\mathbf{g}(t)$$

where η is the damping constant (viscosity), $\mathbf{g}(t)$ is a Gaussian-distributed random process with zero mean, unit variance and uncorrelated in time. This roughly emulates being in contact with an infinite thermal reservoir with temperature T_{desired}. The viscosity η controls how fast the simulation reaches equilibrium with the reservoir and the size of the fluctuations around that equilibrium.

Langevin's equation can be implemented by adding a stochastic step before and after the velocity Verlet algorithm. The derivation can be found in appendix B. For both of these steps, the velocity of each particle is modified as follows

$$\mathbf{v}'_k = c\mathbf{v}_k + v_{\text{thermal}}\sqrt{1-c^2}\,\mathbf{G}$$

where

$$c = e^{-\eta\Delta t/2}, \qquad v_{\text{thermal}} = \sqrt{k_B T/m}$$

and $\mathbf{G}$ is a vector whose components are independent Gaussian random numbers with zero mean and unit variance.

2.6. Modes

The model has two modes, dubbed *microscopic* and *macroscopic*. The *microscopic mode* is the standard simulation technique in molecular dynamics [7]. It describes the interactions between particles as smooth potentials, and uses small timesteps. The *macroscopic mode* describes the particles as hard discs – a discontinuous potential. It solves for exact collision between the particles, changing the velocities by an impulse at the instant of impact.

There are several reasons for having two modes:

- In the macroscopic mode the particles behave like macroscopic objects – billiard balls. This is useful when comparing the tiny particles to familiar macroscopic objects like billiard balls.
- The large timestep of the macroscopic mode can simulate a larger number of particles in real time.
- The large timestep and exact collision-solving of the macroscopic mode is incompatible with the potentials used in the microscopic mode.

- If velocities get too large, the microscopic mode will violate energy conservation, usually resulting in an explosion with particles flying off in different directions. In contrast, the exact collision-solving means the macroscopic mode conserves energy at much higher temperatures, and breaks down only because of floating-point inaccuracies.

2.7. Microscopic mode

The microscopic mode is simpler than the macroscopic, but also more computationally demanding. This is because it uses a much smaller timestep, typically in the range $10^{-3} < \Delta t < 10^{-2}$ in simulation time units, which corresponds to ~ 1000 iterations per second on the computer.

2.7.1. Potentials

In the micro mode, the particles and walls only interact through potentials. The most important kinds of potentials in the explanations are a simple repulsive potential, and a simple attractive potential. Both of these are modeled using the Lennard-Jones potential.

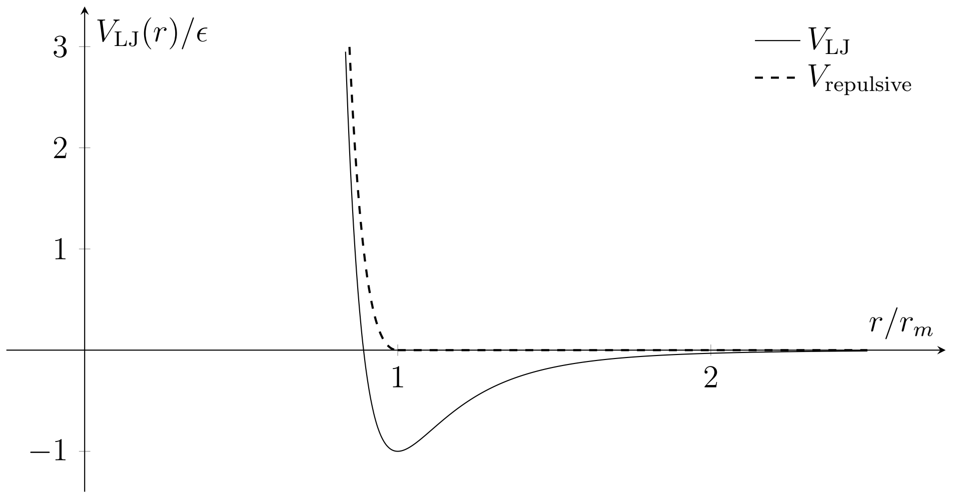


Figure 2.2.: The Lennard-Jones potential with shape parameter $s = 6$.

Lennard-Jones potential

The Lennard-Jones Potential describes the interaction of two electrically neutral atoms, and is a good model for noble gases. It is spherically symmetric, and thus depends only on the distance r between two particles, and is written

$$V_{\text{LJ}}(r) = \epsilon\left(\left(\frac{r_m}{r}\right)^{2s} - 2\left(\frac{r_m}{r}\right)^{s}\right)$$

where ϵ is the bond energy (depth of the well), and r_m is the distance to the potential minimum. In the standard formulation $s = 6$, but I have chosen to make s into a parameter, controlling the width of the potential well.

The potential has two terms, one positive (repulsive) and one negative (attractive). The positive r^{-2s} term models Pauli-repulsion, and dominates when the particles are close ($r < r_m$). The negative r^{-s} term models van der Waals-attraction, and dominates at long range ($r > r_m$).

The Lennard-Jones force between particles k and l, acting on particle l, is computed as

$$\mathbf{F}_{\text{LJ}}(r) = -\nabla V_{\text{LJ}} = (\mathbf{x_k} - \mathbf{x_l})\frac{2s\epsilon}{r^2}\left(\frac{r_m}{r}\right)^{s}\left(\left(\frac{r_m}{r}\right)^{s} - 1\right)$$

with $r = |\mathbf{x_k} - \mathbf{x_l}|$. If s is even, r will only appear squared in this formula, and no square roots need to be computed. The force on particle k is $-\mathbf{F}_{\text{LJ}}(r)$, as per Newton's third law.

Repulsive potential

To make a smooth, repulsive potential, I simply truncate the Lennard-Jones potential at $r = r_m$ and shift it to get

$$V_{\text{repulsive}}(r) = \begin{cases} V_{\text{LJ}}(r) + \epsilon, & \text{if } r < r_m \\ 0, & \text{if } r \geq r_m \end{cases}$$

also shown in figure 2.2 At r_m, the bottom of the well of V_{LJ}, the force vanishes, so $F_{\text{repulsive}} = -\nabla V_{\text{repulsive}}$ is continuous. The shift by $V_{\text{LJ}}(r_m) = \epsilon$ makes sure $V_{\text{repulsive}}$ is continuous too.

Truncation

The Lennard-Jones potential is weak at large distances: particles far apart have negligible influence on each other. To save computational time, I would like to ignore these long-range interactions. Therefore, I cut off the potential at $r_{\text{cutoff}} = 2.5r_m$, where $|V_{\text{LJ}}(2.5r_m)| < 0.01\epsilon$.

$$V_{\text{truncated}}(r) = \begin{cases} V_{\text{LJ}}(r), & \text{if } r < r_{\text{cutoff}} \\ 0, & \text{if } r \geq r_{\text{cutoff}} \end{cases}$$

This potential is not continuous, and neither is its derivative (the force). This can be fixed by shifting the non-zero part ($r < r_{\text{cutoff}}$) of the potential by $-V_{\text{LJ}}(r_{\text{cutoff}})$ and $(r - r_{\text{cutoff}})F_{\text{LJ}}(r_{\text{cutoff}})$, respectively. However, I did not need this correction to make the explanations work, so I did not use it.

2.7.2. Lookup grid

With a truncated potential, computing the force can be avoided for distant particles. However, the *distance* between every pair of particles still has to be computed, so the time complexity of finding the accelerations is $O(N^2_{\text{particles}})$. The complexity can be reduced by putting the particles in an appropriate data structure.

I use the most basic data structure: a lookup grid. The simulation is divided into a grid with n_x columns and n_y rows. I want to find out which grid cell each particle belongs to (i.e. which cell contains the particle's center). The grid coordinates (x_g, y_g) of the cell are calculated from each particle's coordinates $\mathbf{x}_k = (x, y)$ as

$$x_g = \left\lfloor \frac{x - x_{\min}}{x_{\max} - x_{\min}} n_x \right\rfloor, \qquad y_g = \left\lfloor \frac{y - y_{\min}}{y_{\max} - y_{\min}} n_y \right\rfloor \tag{2.2}$$

The algorithm for calculating the accelerations is then:

Algorithm 2: Lookup grid

```
Initialize $L$ as an $n_x \times n_y$ array of grid cells.
for each particle do
    Find grid coordinates of the particle using equation (2.2).
    Store grid coordinates with particle.
    Append a pointer to the particle to the grid cell at $L[x_g + n_x y_g]$.
for each particle do
    for each grid cell within cutoff radius of particle do
        for each particle in grid cell do
            Calculate forces and update acceleration of involved particles.
```

Because particles cannot overlap much, there is a maximum number of particles that fit in each grid cell. Therefore, the innermost loop in the above algorithm is bounded by a constant, and the time complexity is reduced to $O(N_{\text{particles}})$.

2.8. Macroscopic mode

In the macroscopic mode, the particles are all hard spheres, corresponding to the two-particle potential

$$V(r) = \begin{cases} \infty, & r \leq R \\ 0, & r > R \end{cases} \tag{2.3}$$

where R is the sum of the particle radii. This potential cannot be numerically integrated because of the infinite potential, and the infinite slope at $r = R$. However, because the potential is constant everywhere else, the only time this potential exerts a force is at exactly $r = R$, where a collision occurs.

If there are only such hard collisions, the velocities of the particles will not change except at the exact time of impact. Therefore, we can move all the particles in straight lines until the next pair of them collide. The basic algorithm is then

Algorithm 3: Basic collision algorithm

```
repeat
    Find the time until the next collision
    Move all particles until the next collision
    Handle the collision, changing the velocities of the involved particles
until end of timestep
```

This algorithm does not benefit from smaller timesteps, so it is run with one timestep per frame. The following sections go through this algorithm in more detail, and discuss edge cases and optimizations.

2.8.1. Finding the next collision

Figure 2.3a depicts two particles a and b, with discs as collision hulls, moving at constant velocities $\mathbf{v}_a$ and $\mathbf{v}_b$. I want to find if and when they will collide, which is the same as finding the smallest future time t at which the two particles touch. Even at constant velocity, is not always obvious if they will collide or not, as demonstrated in figure 2.3b.

I simplify the problem by measuring the velocities relative to one of the particles (I pick particle a). Then $v'_a = 0$ and $v'_b = v_b - v_a$, as seen in figure 2.3c. In this system, only one particle is moving, and it is rather obvious whether the two particles will collide, figure 2.3d. Here, I am trying to find the time of intersection for two circles, which is straight-forward to solve. However, I would like a more general solution, that can apply to walls as well, and maybe even arbitrary collision hulls.

The problem would be much simpler if the moving object was a point instead of a circle, figure 2.3e. A moving point sweeps out a line, so the problem would reduce to finding the intersection between an object and a line. I can achieve this similarly to the relative velocity change above. There, I transformed the system so that one particle had all the velocity. Here, I will transform the system so that one particle has all the collision hull, and the other is just a point.

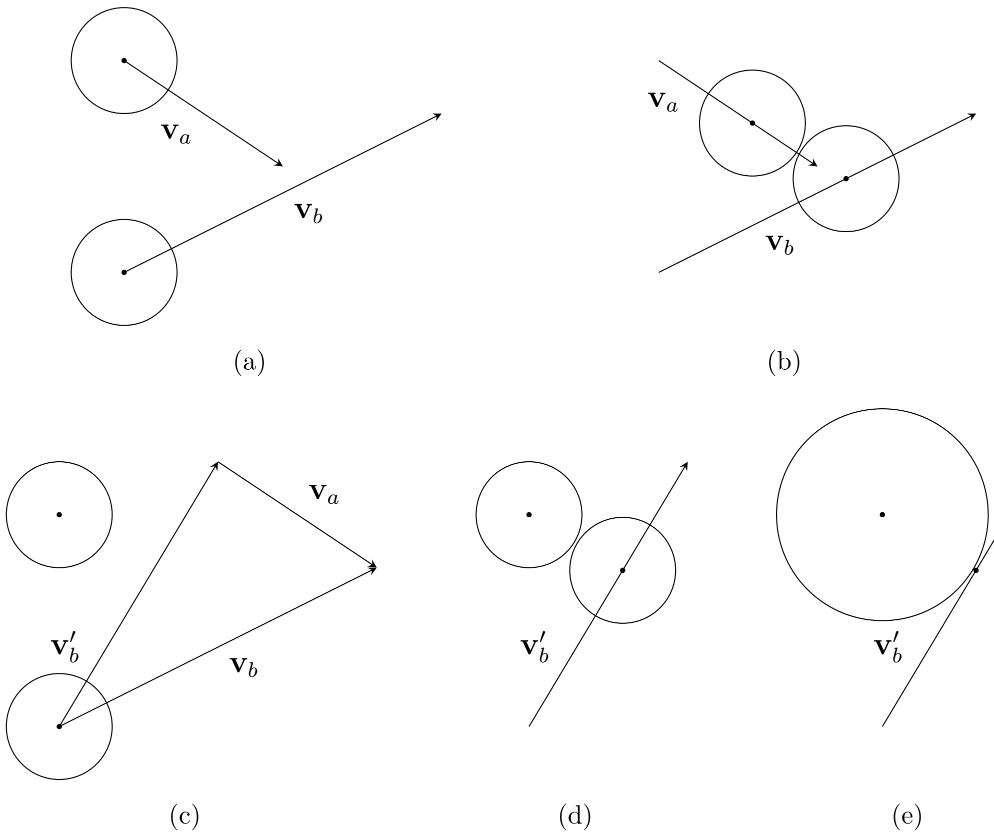


Figure 2.3.: Collision between two particles a and b. Figure (a) is the general case, and together with (b) demonstrates that it can be unclear if the particles will collide (here, they barely miss each other), (c) defines the relative velocity $\mathbf{v}'_b = \mathbf{v}_b - \mathbf{v}_a$, (d) shows that in the relative system, the question of collision is straight-forward, and (e) depicts the equivalent collision of a point with a combined collision hull.

I move one shape around the perimeter of the other, with no gap between them (figure 2.4a). The shape traced by the origin of the other shape will be their combined collision hull. That is, colliding a point with the newly traced shape is the same as colliding the two original shapes. Technically, this operation is the *Minkowski sum* of two sets. For two discs, the Minkowski sum is just another disc with radius $R = R_1 + R_2$.

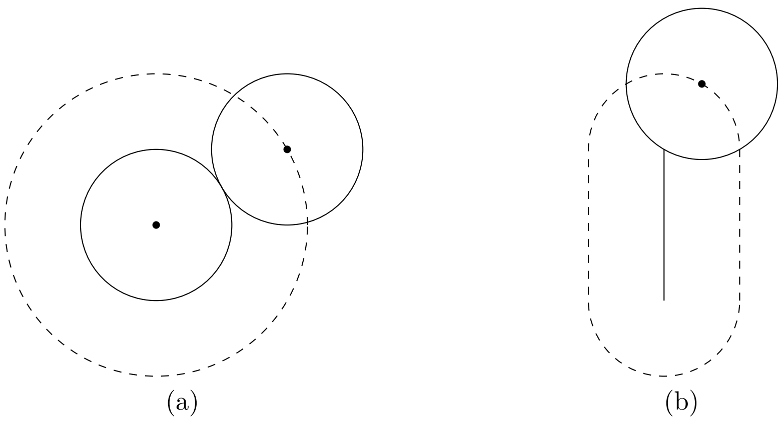


Figure 2.4.: Combined collision hull. A particle is traced along the boundary of (a) another particle or (b) a wall. The dashed shape traced by the particle's center is the combined collision hull of the two objects.

Particle–particle collision

The problem is now reduced to the intersection between a line segment and a circle. I write the line segment on parametric form $\mathbf{x} = \mathbf{a} + t\mathbf{b}, t \in [0, 1]$, and let the circle have radius R and center at $\mathbf{c}$. Then, a short derivation (appendix C.1) arrives at the answer

$$t_{+,-} = \frac{\mathbf{d}\cdot\mathbf{b} \pm \sqrt{(\mathbf{d}\cdot\mathbf{b})^2 + \mathbf{b}^2(R^2 - \mathbf{d}^2)}}{\mathbf{b}^2}$$

where $\mathbf{d} = \mathbf{c} - \mathbf{a}$.

However, we are only interested in the earliest intersection at $t_{\text{collision}} = t_-$ (negative sign in front of the square root), which corresponds to collision from the outside. (If the particles collide from inside, we let them move out of overlap without any collision.)

The normal of the collision is the vector connecting the particle centers at the time of collision $\mathbf{n} = \mathbf{a} + t_{\text{collision}}\mathbf{b} - \mathbf{c}$, normalized to $\hat{\mathbf{n}} = \mathbf{n}/|\mathbf{n}|$.

Particle–wall collision

A collision between wall and particle is more complicated than the particle-particle case. Here the combined collision hull is a rectangle capped by two halfcircles at opposite ends (figure 2.4b). I have to check for intersection between the line of movement and the two straight sides, as well as with the two circles at the ends.

First, I test for intersection with the sides of the rectangle, using the line-line intersection formula in appendix C.2, and making sure that the intersection is from the outside. If there is a collision with the sides, there can be no collision with the semicircles. If not, I can test the semicircles as if they were circles, as with the particle-particle collision.

The collision normal is the normal of the combined hull at the point of contact. If the collision is at the sides, it is simply the outward normal of the sides, and for the circles it is calculated the same way as in the previous section.

2.8.2. Handling a collision

Because of the discontinuous potential, equation (2.3), the collision is an *impulse*, changing the momenta of the particles instantaneously. The direction of the impulse is normal to the line of collision, so the momenta will only change along the normal $\hat{\mathbf{n}}$.

Given two bodies with velocities $\mathbf{v}_1$ and $\mathbf{v}_2$, the relative velocity in the normal direction is

$$\mathbf{v}_n = ((\mathbf{v}_1 - \mathbf{v}_2) \cdot \hat{\mathbf{n}})\hat{\mathbf{n}}.$$

The strength of the impulse is such that momentum and energy is conserved. Using this fact, the formulae for updating the velocities of the bodies are

$$\begin{aligned}\mathbf{v}_1' &= \mathbf{v}_1 - \frac{2m_2}{m_1 + m_2}\mathbf{v}_n \\ \mathbf{v}_2' &= \mathbf{v}_2 + \frac{2m_1}{m_1 + m_2}\mathbf{v}_n\end{aligned}$$

For non-elastic collisions, we can introduce a coefficient of restitution C_R such that:

- If $C_R = 1$, the collision is completely elastic and energy is conserved.
- If $C_R = 0$, the collision is completely inelastic. The objects will stick together and move with the same velocity after the collision.
- With $0 < C_R < 1$, interpolate between these two scenarios.

The velocity update formulae then become

$$\begin{aligned}\mathbf{v}_1' &= \mathbf{v}_1 - \frac{(1 + C_R)m_2}{m_1 + m_2}\mathbf{v}_n \\ \mathbf{v}_2' &= \mathbf{v}_2 + \frac{(1 + C_R)m_1}{m_1 + m_2}\mathbf{v}_n\end{aligned}$$

Walls are immovable, which is modeled by giving them infinite mass $m_{\text{wall}} = \infty$ and zero velocity $\mathbf{v}_{\text{wall}} = 0$. In this case the velocity update formulae are

$$\mathbf{v}'_{\text{particle}} = \mathbf{v}_{\text{particle}} - \frac{(1+C_R)m_{\text{wall}}}{m_{\text{particle}} + m_{\text{wall}}}\mathbf{v}_n = \mathbf{v}_{\text{particle}} - (1+C_R)\mathbf{v}_n$$
$$\mathbf{v}'_{\text{wall}} = \mathbf{v}_{\text{wall}} + \frac{(1+C_R)m_{\text{wall}}}{m_{\text{particle}} + m_{\text{wall}}}\mathbf{v}_n = \mathbf{v}_{\text{wall}}$$

which correctly bounces the particle off the wall, with the wall remaining in place.

2.8.3. Detailed collision algorithm

Here, I express the collision algorithm in more detail. We will have to find the collisions of all pairs of particles and walls, advance time until the next collision, and resolve that collision. It will be useful to introduce an object C representing a potential collision, containing the collision time, the colliding objects, and the collision normal. We can then state the naive algorithm as follows:

Algorithm 4: Detailed collision algorithm

```
$t_{\text{remaining}} \leftarrow \Delta t$
repeat
    Reset recorded collisions
    for all particle pairs do
        if this pair collides with $0 \leq t_{\text{collision}} < t_{\text{remaining}}$ then
            Record the collision and its time
    Find the earliest collision $C_{\text{earliest}}$, with time $t_{\text{earliest}}$
    Move all particles: $\mathbf{x}_k \leftarrow \mathbf{x}_k + t_{\text{earliest}}\mathbf{v}_k$
    $t_{\text{remaining}} \leftarrow t_{\text{remaining}} - t_{\text{earliest}}$
    Resolve the collision $C_{\text{earliest}}$
until no recorded collisions
Move all particles to end of timestep: $\mathbf{x}_k \leftarrow \mathbf{x}_k + t_{\text{remaining}}\mathbf{v}_k$
```

It seems wasteful to look at all pairs of particles after each collision. Only the two particles involved in the collision have changed their trajectories, so only the collisions involving them needs to be updated. If we store the potential collisions so that they persist between collisions, we can avoid recalculating the information. This idea leads us to the next iteration of the algorithm (new or moved lines are shown with a +):

Algorithm 5: Final collision algorithm

```
  $t_{\text{remaining}} \leftarrow \Delta t$
+ **for** all particle pairs **do**
+     **if** this pair collides with $0 \leq t_{\text{collision}} < t_{\text{remaining}}$ **then**
+         Record the collision and its time
  **repeat**
      Find the earliest collision $C_{\text{earliest}}$, with time $t_{\text{earliest}}$
      Move all particles: $\mathbf{x}_k \leftarrow \mathbf{x}_k + t_{\text{earliest}}\mathbf{v}_k$
      $t_{\text{remaining}} \leftarrow t_{\text{remaining}} - t_{\text{earliest}}$
      Resolve the collision $C_{\text{earliest}}$
+     **for** all recorded collisions $C$ **do**
+         $t(C) \leftarrow t(C) - t_{\text{earliest}}$
+         **if** $C$ shares an involved particle with $C_{\text{earliest}}$ **then**
+             Remove $C$ from the recorded collisions
+     **for** all particle pairs containing a particle involved in $C_{\text{earliest}}$ **do**
+         **if** this pair collides with $0 \leq t_{\text{collision}} < t_{\text{remaining}}$ **then**
+             Record the collision and its time
  **until** no recorded collisions
  Move all particles to end of timestep: $\mathbf{x}_k \leftarrow \mathbf{x}_k + t_{\text{remaining}}\mathbf{v}_k$
```

3. Interactions

The simulation is an important part of an explorable explanation. However, it is not very *explorable* unless the reader can interact with it. The reader should be able to perform a number of actions:

- **Move particles**.
- **Add or remove particles to the simulation**.
- **Change global parameters such as temperature and friction**.
- Set up initial conditions.
- Add or remove walls.
- Modify particle attributes such as size or type.
- Select multiple particles to move or modify.

Not all of these actions are fully implemented as I am writing this, and this chapter will only cover the **ones in bold**.

While all of these actions effectively means changing some numbers in computer memory, I do not want the reader to just be typing numbers in boxes. An abstract interaction like that runs counter to the goal of the project, which is to use the reader's intuition. Instead, these actions are much more natural if performed by direct manipulation of the simulation, using a mouse or other pointing device.

For manipulations of non-spatial parameters, such as friction, I create a visual representation of that parameter that can be manipulated directly. Numerical parameters are represented by sliders, dragged by the mouse. Each slider has a range specified by the author, and can optionally use a non-linear scale. Boolean parameters are represented by checkboxes, which can be enabled or disabled. There are also buttons, that can be used for actions (such as resetting the simulation).

3.1. Tools

A simple way to allow each action to be performed is the concept of *tools*, familiar to most people from almost any computer application. There are a number of tools, one of which is active. The active tool dictates the behavior of the mouse pointer, so that many different actions can be performed using a device with only a few buttons. To keep the interactions as simple as possible, only the primary mouse button is used. This means each tool corresponds to exactly one kind of interaction.

3.1.1. The Drag Tool

Dragging things with the mouse is a common action on a computer. We drag files, windows, scrollbar handles, and more. Therefore, it is natural to want to drag the particles in the simulation, which is what the drag tool enables.

However, there is a complication. Normally when dragging, the object being dragged will follow the mouse pointer immediately. The coordinates of the object and the pointer are the same at all times. This works in the pseudophysical desktop metaphor of most operating systems, but it is problematic in a physical simulation such as this one.

For example, if the mouse drags a particle through a wall, the particle will follow the mouse through the wall. Furthermore, there is no clear way to define the momentum or acceleration of the particle, since it can move as erratically as the mouse. These issues are avoided if we instead act on particles with a *force* and let the simulation take care of moving it. This approach is more realistic, and also allows the reader to create a force in a static situation, such as pushing a particle towards a wall.

I chose to have the drag tool act as a spring with damping,

$$\mathbf{F}_{\text{drag}} = -k(\mathbf{x}_k - \mathbf{m}) - c\dot{\mathbf{x}}_k$$

where $\mathbf{m}$ is the pointer position, and k and c are the strength of the spring and damping, respectively. Note that the damping is not restricted to the line of the spring. With more realisitic spring damping, as with no damping, the particle oscillates around the mouse when held still, which is unintuitive.

3.1.2. The impulse tool

In the explanations, the game of billiards is used as a familiar analogue to tiny particles. Billiards is not played by dragging the balls around, but by aiming and shooting, also called an *impulse*. This action is also a useful way to give the particle system a more controlled energy increase than is possible with the drag tool.

An impulse is an instantaneous change in momentum, or a force acting during a very short time. This is exactly what happens in the hard collision in section 2.8.2. The simulation uses a finite time step Δt, and so the most instantaneous force is one that acts for one timestep. To achieve an impulse of strength I, the force is

$$\mathbf{F}_{\text{impulse}} = \frac{I}{\Delta t}(\mathbf{m} - \mathbf{x}_k)$$

where $\mathbf{m}$ is the pointer position.

The tool allows precise aiming, and is used like this:

1. Press the mouse button on top of a particle.
2. Move the mouse while holding the button down. An arrow is drawn from the particle to the pointer, indicating the direction and magnitude of the impulse.
3. Release the mouse button. The impulse is applied and the particle's velocity changes instantly.

3.1.3. The Create and Destroy Tools

These tools creates or destroys particles at the pointer if the mouse button is held down. Particles will only be created if nothing would intersect the particle. Without this rule, the new particle might appear too close to walls or particles. In the microscopic mode, very close means a very high potential, and the particle might get a speed too high for the simulation too handle. In the macroscopic mode, the particle might intersect walls and other particles.

The type, mass and radius of the created particle is specified by the author, and cannot be changed by the reader. This feature was not prioritised, but is not hard to add.

4. Presentation

An interactive simulation can be explored and played with, but it is not an *explanation*. An explanation implies an *author* who is presenting ideas, guiding the reader to better understanding. Presentation is an important part of any explanation, such as a traditional textbook or lecture. Since this report is about *interactive* explanations, I will focus on possibilities and considerations unique to the interactive medium.

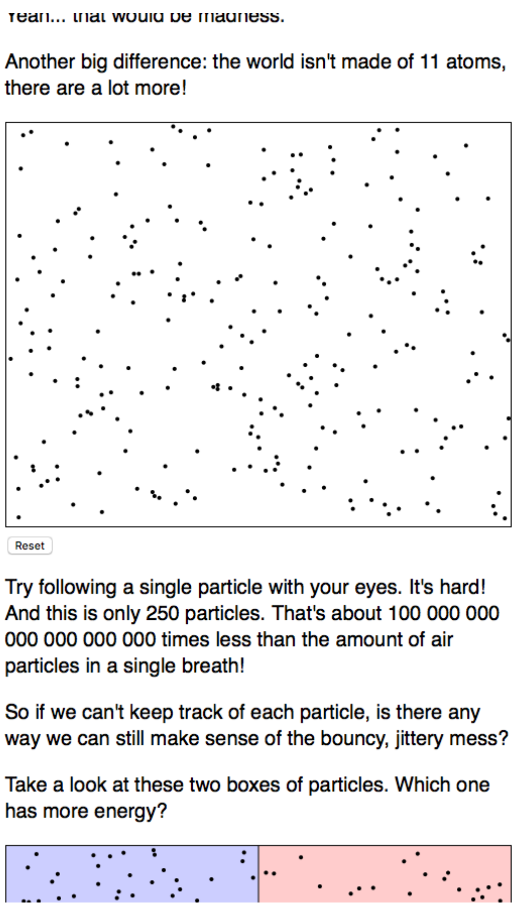
A caveat: while some of the ideas in this section might be applicable to explorable explanations on other subjects, I do not have enough experience to claim any general principles. Therefore, I will center the discussion around the development of Many Tiny Things.

There are a number of goals for the presentation. It should:

- **Guide the reader** through the simulations.
- **Invite curiosity and play**. The reader should be encouraged to play with the simulation and try things out.
- **Provide context** to the model, comparing it to real world situations.
- **Make sure the reader follows**. If the reader misses some important idea, she might feel lost in the further explanations.

Excerpts from an early design and the final one can be seen in figure 4.1, and can be compared with the excerpts in figure 1.1. My initial idea was to present the explanations like a textbook, but with more, moving, and interactive pictures – the simulations. This is essentially the presentation used in *Parable of the Polygons* [3], although I would have more than one page of explanations. However, for several reasons, that format did not work very well, and I ended up with something closer to *Earth: A Primer* [5]. That is, there are many short pages, each centered around a single instance of the simulation. Most pages have a small task to accomplish by interacting with the simulation.

The design process consisted mainly of me implementing various prototypes, trying them out, and iterating on the inevitable flaws I would discover. I made several shorter explanations for each presentation prototype, to make sure it was thouroughly tested. Because the content is dynamic, it is hard to predict if an idea will work or not. It is not enough to look at it, it has to be interacted with. As the author, I know how everything is supposed to work, and what it is supposed to convey. Therefore, I cannot myself fully test whether the explanations are working as intended. An essential part of the development process was to test the explanations on friends and family. Letting someone from the intended audience test the explanations uncovered issues and gave me insight


Yeah... that would be madness.

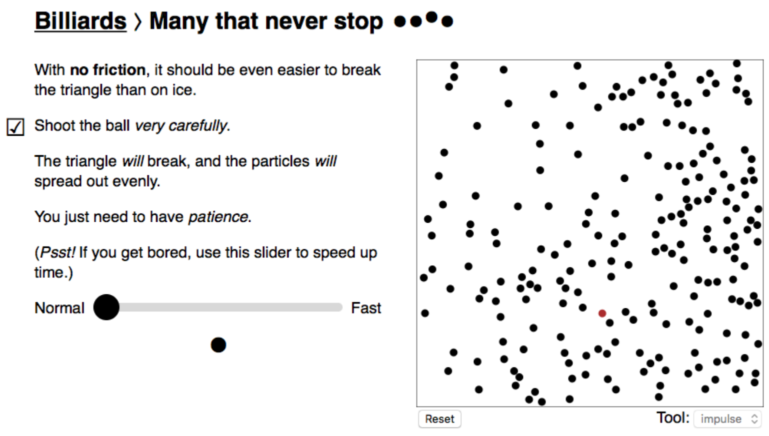
Another big difference: the world isn't made of 11 atoms, there are a lot more!

Try following a single particle with your eyes. It's hard! And this is only 250 particles. That's about 100 000 000 000 000 000 000 times less than the amount of air particles in a single breath!

So if we can't keep track of each particle, is there any way we can still make sense of the bouncy, jittery mess?

Take a look at these two boxes of particles. Which one has more energy?

(a) Early design


Billiards 〉 **Many that never stop**

With **no friction**, it should be even easier to break the triangle than on ice.

☑ Shoot the ball *very carefully*.

The triangle *will* break, and the particles *will* spread out evenly.

You just need to have *patience*.

(*Psst!* If you get bored, use this slider to speed up time.)

(b) Final design

Figure 4.1.: Examples of part of an explanation. The early design screenshot is taken from the middle of an explanation, and continues upward and downward. The screenshot of the final design shows one page in the middle of an explanation.

in how to fix them.

The rest of this chapter goes into more detail on the issues I encountered and how I resolved them. I describe the development from a few different angles:

- **Exploratory vs Explanatory**, putting emphasis on the simulations or the text.
- **Gating**, hindering the reader's progress to make sure they are following the argument.
- **Branching**, using the interactive medium to express non-linear arguments.

4.1. Exploratory vs Explanatory

The relationship between the explanatory text and exploratory simulations is an important part of the design. In the initial designs, the text was the main part of the explanations, with accompanying simulations that illustrated points in the text (figure 4.1a). In digital media, there are no page boundaries to account for, so I put the simulations inline in the text, right where they are mentioned. With the simulations as part of the flow of the argument, the text could **guide the reader** through the explanation. This design is common on the web and familiar to both author and audience. However, the familiarity turned out to be a detriment. With the focus on the text, readers treated it like they would treat a standard, static text, and often unintentionally skipped interacting with the simulations.

I experimented with removing the text completely, and instead communicating wordlessly. The idea was to set up the initial conditions, perhaps have some visual indicators telling the reader what to interact with, and hope that the concept (e.g. "entropy always increases") came across. To the surprise of no one, it did not work out. While the explained phenomena are more fundamental than our words describing them, having a name for an abstract idea nonetheless helps make it more concrete, and names are crucial in communicating with other people. It is also much less effort for the author to tell the reader in words "give the particles a kick with the impulse tool and see what happens", than it is to program, draw and animate the same idea without words.

The final design, figure 4.1b, is nevertheless inspired by the wordless experiment. I exchanged the roles of text and illustration, letting the simulations take center stage. This means less text per simulation, which together with the gating puts focus on the simulation. This way, instead of having the voice of the author dominate, I emphasize the reader's exploration, with some guiding words from the author. The design is similar to the panels of a comic, where the picture is in the spotlight, with the words in a supporting role.

Furthermore, I let each page contain just one simulation. This had several advantages. It presents one idea at a time, which makes the arguments clearer. It also lets the reader keep exploring and thinking until she is ready to continue, with no distractions. Since the text only talks about what is currently on screen, it avoids ambiguities and makes it easier for the author to set up the right conditions. And as a side benefit, it requires less computer resources, since only one simulation is run at a time.

4.2. Gating

In a traditional textbook, the author sometimes asks the reader to do an exercise before continuing. In an interactive medium, the author can *require* that the reader does the exercise, or she cannot continue reading. I call this **gating**, and the required exercise a **lock**. For example, in an explanation on matrix multiplication, a gate could require the reader to multiply two given matrices. Filling in the correct result would unlock the gate and let her keep reading. The point of such a gate is making sure that the reader is following the argument. If she gets the answer wrong, the computer can suggest that she rereads the explanation. A more sophisticated program might even analyze the error and suggest possible misunderstandings.

My initial design had no gating, as seen in figure 4.1a. The idea was to let the reader explore at their own pace. This way, someone new to the subject would naturally go through the explanations thoroughly, rereading things until she *got it*. On the other hand, an experienced reader could skim through quickly, with nothing slowing her down.

Unfortunately, this approach did not work very well. The main issue is that readers are not used to interactivity. When presented with what looks like a normal text with pictures, many treat it as such, and scroll past interactive elements (out of habit, I assume). I had to remind the reader to interact with the simulations, but even then, they would forget that interacting was part of the explanation.

If a reader misses the point of a paragraph in a book, she usually rereads it and tries to make sense of it. So here would the reader reread the *text*, but still miss the interactive part. This presentation did **not guide the reader well**, and if she missed the interactive parts, she would have **problems following the argument**.

I decided to implement some kind of gating, that made sure that the reader had interacted with the simulation. The basic idea was to have one or more tasks for the reader to finish before she could keep reading.

In my first gating prototypes, I would put gates at every step of the explanation. This **made sure that the reader followed**, but had a number of drawbacks. Instead of *exploring* the simulation, the reader was just doing what she was told. With every finished task, a new task would pop up, leaving **little room for exploration** or contemplation. Instead, the reader's attention kept jumping between the simulation and the new task. I wanted the tasks to be friendly suggestions **guiding the reader**, and but instead they became boxes to tick, chores to do.

Part of the idea with putting many tasks in a row was to guide the reader, step-by-step, through an argument. However, in a precise argument, steps might require specific initial conditions, such as having the particles be arranged a certain way. With many tasks in a row, the conditions became less and less predictable for each task. This made **guiding the reader hard** for me, the author.

All of this led me to the final design, shown in figure 4.1b, which uses the same gates, but more sparingly. The gates are used for two main reasons:

- To **make sure the reader does not miss how to *interact* with the simulation**. For example, the first time a new tool is introduced, a gate makes sure that the reader understands how to use it before moving on.
- To let the reader build an hypothesis on what is going to happen. The gate hides the text that would otherwise reveal the result of the little experiment. This **invites exploration** and hopefully encourages active reading and thinking ahead.

Furthermore, the gates only stop progress on that particular page, and the reader is free to browse ahead or go back and reread something at her leisure. There are small indicators showing which pages have been interacted with, so that she can see where she has been and what is left to explore.

4.3. Structure and branching

In a textbook or a lecture, the author presents ideas as a sequence of words with accompanying pictures. I call this kind of structure **linear**, in the sense that the information is layed out in a straight line (figure 5.1a). If the ideas presented are nodes in a graph, the book or lecture would be a *path graph*.

Note that, even though the information is presented in a specific order, the reader will not necessarily learn the ideas in that order. She can jump around in a book, rereading parts she found difficult, skipping over parts that seem less important, and even look up other texts referenced from the book. In a lecture, the reader can ask questions or request that something be repeated. Nonetheless, we are talking about *presentation*, and the information is *presented* in a linear fashion.

In contrast, an interactive medium allows the presention to **branch**, with hyperlinks that instantly move the reader to another page. This is what the World Wide Web is built on. With this model, information can be presented as an arbitrary graph, figure 5.1b, where each page is a node, and the links to other pages are directed edges. This way of structuring information is very flexible, with no restriction on the amount of **branching**.

On the other hand, the more branching occurs, the harder it becomes to keep the bigger picture in one's head. Furthermore, increased branching also means decreased authorship, since the reader can read the information in any order. Therefore there is usually a smallest unit of linear structure. On Wikipedia, it is an article, on a blog it is a blog post, and on the web in general it is a webpage.

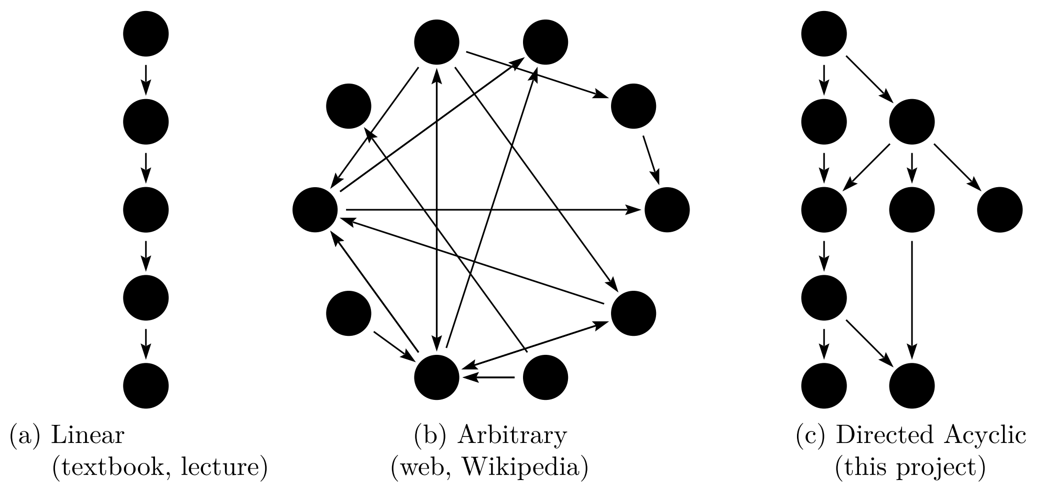


Figure 4.2.: Examples of graphs representing the information flow in different media.

In Many Tiny Things, I have chosen to restrict myself to a structure roughly equivalent to a *Directed Acyclic Graph* (DAG). A DAG is a graph where all the nodes can be put in a sequence, where all directed edges only point to nodes later in the sequence. This maintains the sense of direction needed for an argument, while allowing some amount of branching.

In practice, the branching can be used to highlight parallel tracks of an argument, or show that two concepts are independent. Additionally, a DAG allows me to express that the explanations of some concepts, say brownian motion, depends on explanations of earlier concepts, in this case maybe heat and friction.

My hope is that some branching can help **guide the reader** by showing the structure of the argument more explicitly than in a linear medium. However, there is a risk that splitting up the argument introduces confusion, making it **harder for the reader to follow**.

An argument against branching is that, no matter how the text branches, the reader will always read the explanation in *some* order. If the parts are put in a sequence, the author can choose the order that is deemed closest to optimal, whereas otherwise it is up to the reader. However, learning is not linear, and it is impossible to find an optimal order that fits all readers. Also, readers of traditional textbooks read sections out of order anyway, if they do not understand something.

5. Implementation

Many Tiny Things is implemented as a series of web pages, which at the time of this writing is available at

http://manytinythings.github.io

and the source code is available at

https://github.com/ManyTinyThings/ManyTinyThingsWeb

The purpose of the *World Wide Web* (*web* for short) is mainly to present documents with links between them, which is reflected in its design. It uses HTML (HyperText Markup Language) to describe the contents of a document, CSS (Cascading Style Sheets) to describe the layout of said document, and URLs (Uniform Resource Locators) to link documents together. An increasingly important part of the web is dynamic content that the reader can interact with (such as this project!). For this, the programming language JavaScript is used.

Using the web for this project has both benefits and drawbacks. The main benefits are ease of access and distribution, while the main drawbacks are performance problems and lack of control. This means that the web is a good fit for presentation aspects, but ill-suited for simulation.

5.1. Distribution

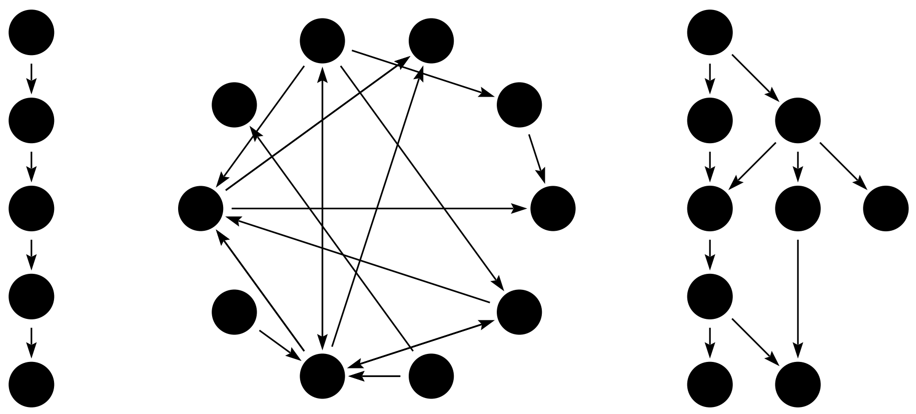
The main reason I choose to use the web format is that it is easily accessible. To access a web page, all the reader needs to do is click a link or type in a web address. There is nothing to install, and nothing to download. Moreover, each page of the explanation has its own web address, allowing, for example, a teacher to integrate parts of the simulation in their course.

Another benefit of the web is the promise of “write once, run anywhere”, that is, that the explanations should work the same in any web browser on any operating system. This is not entirely true, as there are differences both in available hardware and in implementation details of different browsers. I have chosen to target the most popular modern web browsers (Chrome, Firefox, Safari) on desktop operating systems (Windows, macOS, Linux), and on these, almost everything works the same.

The alternative to a web implementation would be a binary application to download and install. This is both harder to access for the reader, and harder for someone like a teacher to integrate. Binary applications also have to be adapted to every operating system, and require more work to do layout and text well. The big upside of a binary application is that it has direct access to the computer, with no layers of abstraction between. This primarily means better performance through less overhead, ability to manually manage memory, and ability to make better use of the computer's CPU (Central Processing Unit) and GPU (Graphics Processing Unit).

5.2. Presentation

The interlinked-document model of the web is a good fit for the presentation of Many Tiny Things. Each page of the explanation is an HTML document, using CSS for page layout, and URLs to link to the other pages. JavaScript is used to program all dynamic and interactive elements, such as graphs and sliders. The HTML contains the content unique to each page, while common CSS and JavaScript is shared between all pages.



(a) Linear (textbook, lecture) (b) Arbitrary (web, Wikipedia) (c) Directed Acyclic (this project)

Figure 5.1.: Examples of graphs representing the information flow in different media.

The pages are served as static HTML documents, stitched together from templates containing shared elements. Static pages have the benefit of being very simple and efficient to host, requiring no custom code to be run on the web server.

5.2.1. Authoring

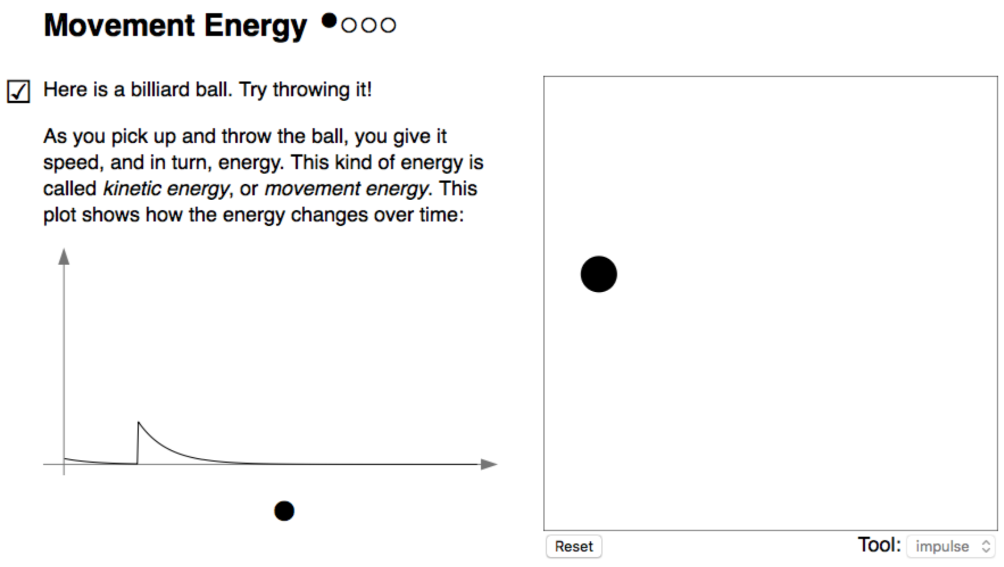
Writing a static document, such as this report, is something most people are comfortable doing. The tools may vary, and getting the details of typesetting right can be tricky, but on the whole, writing a static document is common knowledge.

```
Here is a billiard ball. Try throwing it!

<script>
    lock(function () {
        var energy = getTotalEnergy(sim);
        return (energy > 2);
    });
    gate();
</script>

As you pick up and throw the ball, you give it speed,
and in turn, energy.
This kind of energy is called _kinetic energy_,
or _movement energy_.
This plot shows how the energy changes over time:

<script>
    createTimeSeriesHere({
        timeRange: 50,
        yMax: 10,
        update: function() {
            var energy = getTotalEnergy(sim);
            return {time: sim.time, data: [energy]};
        },
    });
</script>
```


Movement Energy ●○○○

☑ Here is a billiard ball. Try throwing it!

As you pick up and throw the ball, you give it speed, and in turn, energy. This kind of energy is called *kinetic energy*, or *movement energy*. This plot shows how the energy changes over time:

Figure 5.2.: An example showing some of the features of the authoring tools. At the top is the source code, and below it the resulting page (which is interactive, although that is hard to convey in this static text).

On the other hand, how to produce text with interactive parts is not well known. There are almost no tools tailored for this purpose available, so most authors of explorable explanations create their own ad hoc tools. One tool specifically for this purpose is Bret Victor's *Tangle* [9], a "JavaScript library for authoring reactive documents". However, it is mainly for presenting numbers embedded in text, which did not suit this project.

I ended up creating my own basic authoring tool, with a few design goals:

- **Easy to edit**. Minimal effort should be required to change the content.
- **Easy to understand**. Everyone with a little programming experience should be able to use the tool.
- **Not tied to this particular project**. Someone else should be able to use the tool to explain something else.

Some example source code and the produced result are shown in figure 5.2. Simulations, graphs and interactive elements are created inline in the document with JavaScript, and then appear at the corresponding place in the document. This way, the source code mirrors the final result as closely as possible.

Each element has an `update` procedure that is run once every time the screen refreshes (normally at 60 Hz). For an interactive (input) element such as a slider, the `update` procedure receives the value of the slider as an argument. Conversely, the `update` procedure of a reactive (output) element, such as a graph, is used to change the appearance of the graph to reflect new data. In the `update` procedure, the author can write any code that she wishes, and thus easily interoperate with any other code she has written (such as her own simulation).

Gates and locks work similarly to the other elements. A gate hides all text that comes after it in the document until all its locks have been opened. Locks belong to the closest gate following them. A lock is created with a "key function", such that the lock opens when the function returns true. This allows locks to unlock on any condition that the author can write down, similarly to the `update` functions.

5.3. Drawing

To draw something dynamic on a web page, like my simulations, there are three main alternatives.

- **SVG** (Scalable Vector Graphics) [10], a markup language describing shapes with vectors (similar to PDF). It is mainly designed for static pictures and is not well suited for animating with JavaScript.
- **WebGL** (Web Graphics Library) [11], the web version of OpenGL, which is a direct interface to the GPU (Graphics Processing Unit) of the computer. It is a lower level library and is more complicated to use than the other methods, but has the best potential performance.

- The **Canvas** API (Application Programming Interface) [12], which is a number of JavaScript functions used for drawing to a bitmap area of a webpage. This method is simpler to use than the others, and is specifically designed for dynamic drawing.

I experimented with both WebGL and Canvas, and decided to go with Canvas, since it was supported in a wider range of web browsers. Drawing to the screen was never the performance bottleneck, so the performance gain from using WebGL had no overall impact.

5.4. Simulation and Performance

The simulations are embedded in a web page, but runs on the reader's computer. This means they have to be written in JavaScript, the programming language for user interaction on web pages. JavaScript was designed to perform simple tasks, such as hiding an element on a web page, or generating some text dynamically. It was not made to run computationally intensive molecular simulations.

5.4.1. Just-in-time compilation

JavaScript is an *interpreted* language, which means the source code is read (interpreted) by another program while it is being run, executing it line by line. In contrast, a language like C is *compiled* directly into machine code, which is executed by the processor, with no other program in between. The overhead of having an interpreter means a JavaScript program is much slower than a similar C program.

Browser vendors (Apple, Google, Mozilla) have put a lot of effort into improving the performance of JavaScript. This is mainly done using a technique called just-in-time compilation (JIT). While the program is being interpreted, if the interpreter detects that a part of the program is run many times, it compiles that part of the program straight to machine code. JIT compiled code runs about as fast as code compiled ahead-of-time, and greatly improves the performance of JavaScript.

Without JIT, it would not be feasible to run the simulations in the browser. Unfortunately, JIT is harder to work with than normal, ahead-of-time compilation. Most importantly, it is less predictable and harder to reason about. When the code is JIT compiled, it might be optimized, but if the state of the program changes, that optimization might not still be valid. Then the code has to be recompiled without optimizations, resulting in pauses or stutters. Since all of this is done as the program is running, it is hard for the programmer to know how to influence it.

5.4.2. Garbage collection

In a language like C, the programmer is responsible for allocating memory when needed, and freeing it when done. In JavaScript, on the other hand, memory is managed by the interpreter. Automatic memory management can be done in several ways. JavaScript employs *garbage collection*, where the interpreter pauses the program at regular intervals, inspects all memory and frees any memory that is no longer in use.

Garbage collection works well as long as there are natural pauses in the program, where it can run without interrupting anything. However, in the case of a dynamic simulation that runs continuously, the garbage collector causes undesirable pauses and stutters. These issues can be avoided by keeping references to objects not in use, and then reuse these objects instead of allocating new ones. While this approach does work, the JavaScript language is not designed to be used this way, and it leads to more complicated code.

5.4.3. WebAssembly

Just-in-time compilation and garbage collection does not work well with real-time simulation and animation. Therefore, JavaScript is unsuitable for making computationally intensive real-time simulations. Currently, JavaScript is the only way to run such a simulation in a web page, but the web browser vendors are working on a new standard, WebAssembly [13]. As the name suggests, WebAssembly is an assembly language, or byte code, for the web.

The intention is to let the programmer write code in C, for instance, and compile it to WebAssembly. The WebAssembly is sent with the web page, and when it reaches the user, it is compiled, ahead-of-time, to machine code. Garbage collection is optional and will not even be supported in the first version of WebAssembly. WebAssembly is designed to solve the issues I've encountered with JavaScript when developing the simulations, and would be a perfect fit for this project, if it was available.

5.4.4. Emscripten

While WebAssembly is a few years away from being ready for use, there is a related project available as of this writing. The Emscripten compiler [14] compiles C or C++ code to optimized JavaScript. JavaScript generated by Emscripten is easier for the interpreter to just-in-time compile, and it sidesteps the garbage collector by keeping all memory in a huge array, simulating the stack and heap of the C language.

I rewrote parts of the simulation in C, and used Emscripten to compile it to JavaScript. The result performed better, and importantly, completely without pauses or stutters. A prototype is available at

http://manytinythings.github.io/emscripten with source code at

https://github.com/ManyTinyThings/ManyTinyThingsCPP

Unfortunately, replacing the JavaScript simulation with the Emscripten one is not a simple task, and will not be a part of this thesis project.

5.4.5. Parallelism

Much of the simulation code can be run in parallel. Specifically, each part of the velocity Verlet algorithm is independent for each particle. The program could potentially take advantage of the multiple processor cores most computers have, or even run the simulation on the GPU (Graphics Processing Unit), which usually has hundreds of cores.

Running the simulation on multiple CPU cores in JavaScript would require the use of the Web Worker API, which allows limited multicore processing. This would not be too hard, but because of the other shortcomings of JavaScript detailed above, I decided to not spend more time optimizing code that would still retain more severe problems.

Using the GPU to run the simulation has the potential to be the most performant method. However, adapting an algorithm to run on the GPU is a lot of work, and is outside the scope of this project. Furthermore, there is currently no web API for doing general, non-graphics GPU programming.

6. Discussion

There is potential in explorable explanations in general, and in Many Tiny Things in particular. However, that potential was not fully realised during this project. I was unable to create full explanations on the subjects I had planned, such as entropy, pressure and friction. Finished and currently available at

http://manytinythings.github.io

is the *introduction* to the explanations, which introduces the model using billiard balls. The introduction does not fully explain the phenomena, but it demonstrates most of the work described in this report, and should give a good idea of how the finished work will turn out. I plan to keep working on Many Tiny Things, adding more explanations, improving the presentation and fixing implementation problems.

6.1. Effectiveness

It is hard to gauge the effectiveness of the explanations – if they help learning compared to a traditional book. Perhaps the interactivity distracts more than it helps build intuition, and using static pictures would work better. Maybe the explanations require so little effort to read that they make the reader *less* active. Answering questions like these is outside the scope of this project, but could be the basis of a scientific study.

Nevertheless, I can provide an anecdotal, non-scientific account of the testing I *did* perform. The people testing the explanations tended to enjoy the experience, moreso later on in the project as I improved the presentation. As I write this, I have unfortunately not finished enough explanations to be able to confidently assess if the testers understood the many-tiny-particles model. Nonetheless, after only trying the introduction, a few testers were asking questions such as "is this like the wind blowing from high to low pressure?" and "they never stop moving, but they are changing speed, how does that work?".

6.2. Obstacles

The reason that I was not able to finish the explanations as planned was mostly due to me underestimating the amount of work required. I expected to spend more time writing explanations and less on presentation, authoring tools and trying to get JavaScript to run fast. Even writing the explanations was more time-consuming than I had foreseen. However, now that much of the foundational work has been completed, writing new explanations is now much easier than it was to make the first prototypes.

6.2.1. Writing Explorable Explanations

The presentation needed several iterations, as is detailed in chapter 4, more than I initially expected. I did make explanations for energy, entropy and pressure in earlier prototypes, but as the presentation changed, I did not update all of them. Updating them to the final format essentially requires a complete rewrite, which I did not have time to do as part of the thesis.

Writing interactive explanations requires much more work than writing static text. Interactivity means the author has to account for the behavior of the reader, including edge cases such as waving the mouse around furiously. It also requires a different approach to explaining, which I have only begun to learn.

I created the authoring tools to make it as simple as possible to create new explanations. Making gates, graphs and interactive elements work well was time-consuming, and left little time to actually write the explanations. However, this time-investment will hopefully pay off as future explanations are written. The authoring tools are not tied to the specific simulation used here, and could be used to make explanations on other subjects as well.

6.2.2. Performance

The choice to make Many Tiny Things as a website had all the expected benefits of being easily accessible. However, the performance drawbacks were greater than I had anticipated. Some of the planned explanations were impossible to realize, since not enough particles could be simulated in real-time. And the realizable simulations still occasionally pause and stutter because of just-in-time compilation and garbage collection.

In hindsight, I put too much effort into trying to make JavaScript run fast. While certainly possible, it would require an inordinate amount of work, since such code is hard to read and even harder to debug. A better solution would have been to use Emscripten to program the simulations in C, but I was not aware of this possibility until too far into the project.

6.3. Future work

The most obvious extension of this project is to continue developing Many Tiny Things. Moreover, there are other subjects for which explorable explanations might be a good fit.

Another avenue of development is improving the authoring tools, and packaging them for use by other authors. Finally, there is the question of the effectiveness of explorable explanations.

6.3.1. Continued development

Many Tiny Things has room for improvement in several areas: more phenomena can be explained using the same model, using other models can open up new possible explanations, and the explanations could be adapted for a different audience. Furthermore, the simulation could be implemented differently to improve performance.

I plan to at least finish the explanations I have prototypes of, on energy, heat, friction, pressure, sound and entropy. There are more advanced and abstract phenomena that might be explained using the same model:

- Thermodynamic concepts like heat capacity, adiabatic processes, enthalpy and other potentials, engines, cycles and efficiency.
- The idea of keeping certain properties fixed, canonical ensembles.
- More on flow, heat transport, pumping, turbulence, fluid dynamics.
- More complex potentials, bonds and chemical reactions.

Some of these would require more particles, and some would probably work better with a different model. The Ising model, with particles in a lattice, might help explain other aspects of state transitions. A continuum model, describing the system with densities in a grid instead of positions of discrete particles, will fit explanations on flow, and could also be combined with the particle model, to show how they fit together. A model with discrete particles *and* discrete time could be used to demonstrate how microscopic time-reversibility still gives macroscopic irreversibility. A quantum mechanical model could be used to better explain the interactions, chemical bonds, and touch on condensed matter physics.

Another avenue of further development is adapting and extending the explanations for a different audience. Targeting undergraduate physics students, for instance, would call for a different approach. One could then assume more prior knowledge and expect that the readers are comfortable with more complicated visualisations and arguments. The model should be more correct, able to quantitive predictions, and the explanations should be more precise and make use of mathematics.

Rewriting the simulation in a more suitable programming language than JavaScript should allow more particles and more stable performance. The same goes for any other dynamic model that might benefit from having a large number of particles or cells. I **strongly** advise against using JavaScript in these cases. Outside this project, I will attempt to rework Many Tiny Things to use Emscripten, and in the future WebAssembly.

6.3.2. Explanations on Other Subjects

While the molecular dynamics model seems particularly well suited for an explorable explanation due to its tactile nature, explorable explanations can be used to explain other subjects as well. A few ideas:

- **Quantum mechanics** is notoriously hard to grasp intuitively. Perhaps interactivity could help make sense of quantum weirdness?
- **Algorithms**, such as the ones employed in this project, seem like a good fit. Exploring an algorithm or carrying it out herself might help a reader understand better. Mike Bostock has a *sligthly* interactive essay on *visualizing* algorithms [15], which might be inspiring.
- **Statistics** is tricky and unintuitive in the abstract. Exploring probability distributions and sampling through interactive, concrete examples, might make it more tangible.

6.3.3. Presentation and Tools

The most generalizable part of this thesis project is the presentation. Gating, branching and the overall format is not tied to Many Tiny Things in particular, and neither are the tools to make them. What *is* particular to this project is the degree to which the different presentation tools were used. Other projects might, for example, use no gates or have more text. Nonetheless, I think having these tools at one's disposal should prove valuable when making explorable explanations. A natural project, then, is to improve the tools and package them so that other people can use them.

There are a few ideas that were left unexplored concerning the presentation, because they fell outside the scope of the thesis. One is *exercises*, which are common in textbooks, and one of the best ways to learn a subject. The gates in Many Tiny Things serve as mini-exercises, but they are not intended to be challenging. I am not sure if it is possible to make a challenge that gives insight, without using the analytical tools of mathematics and physics, but it seems worth trying.

Another idea is to have a sandbox, where the reader can set up her own initial conditions, interactions, geometry, etcetera and create her own experiments. This is mostly a question of making a user interface for all the possible parameters, without making it too complicated.

The branching structure uses hyperlinks, which instantly move the reader to another page. Sudden jumps like this make pages seem disconnected, resulting in less coherence. One way to solve this problem is to juxtapose the pages spatially, and have the connections between them be animated. *Earth: A Primer* [5] does not have any branching, but uses animations in this way to good effect.

A. Symplectic Integrators

Consider a dynamical system of a single particle with position x and momentum p. The energy of the system is a function of x and p called a Hamiltonian $\mathcal{H}(x,p)$. From the Hamiltonian we can get the equations of motion through Hamilton's equations

$$\dot{x} = \frac{\partial \mathcal{H}}{\partial p}, \qquad \dot{p} = -\frac{\partial \mathcal{H}}{\partial x}. \tag{A.1}$$

For the most common form of Hamiltonian

$$\mathcal{H}(x,p) = \frac{p^2}{2m} + V(x) \tag{A.2}$$

where m is the mass and $V(x)$ is a potential, this gives

$$\dot{x} = \frac{\partial \mathcal{H}}{\partial p} = \frac{p}{m} = v, \qquad \dot{p} = -\frac{\partial \mathcal{H}}{\partial x} = -\frac{\partial V}{\partial x} = F$$

where v is the velocity and F is the force. Put together, this is Newton's second law

$$m\ddot{x} = \dot{p} = F$$

We can write equation (A.1) as a single equation if we introduce $z = (x,p)$ and the operator $D_{\mathcal{H}}$, defined as

$$\begin{aligned} D_{\mathcal{H}} z &= \frac{\partial z}{\partial x}\frac{\partial \mathcal{H}}{\partial p} - \frac{\partial z}{\partial p}\frac{\partial \mathcal{H}}{\partial x} \\ &= \left(\frac{\partial \mathcal{H}}{\partial p}, -\frac{\partial \mathcal{H}}{\partial x}\right). \end{aligned} \tag{A.3}$$

(This is the same as the *poisson bracket* of $\mathcal{H}$ and z: $D_{\mathcal{H}} z = \{z, \mathcal{H}\}$.) Hamilton's equations then read simply

$$\dot{z} = D_{\mathcal{H}} z$$

and the formal solution is an operator exponential

$$z(t) = \exp(t D_{\mathcal{H}}) z(0)$$

defined as a Taylor expansion

$$\exp(t D_{\mathcal{H}}) = 1 + t D_{\mathcal{H}} + \frac{t^2 D_{\mathcal{H}}^2}{2!} + \frac{t^3 D_{\mathcal{H}}^3}{3!} + \dots \tag{A.4}$$

The simplest numerical integration scheme is just truncating this expansion after two terms and using a small timestep Δt

$$\begin{aligned} z(t+\Delta t) &= \exp(\Delta t D_{\mathcal{H}}) z(t) \\ &\approx (1+\Delta t D_{\mathcal{H}}) z(t) = (x(t), p(t)) + \Delta t \left(\frac{\partial \mathcal{H}}{\partial p}(t), -\frac{\partial \mathcal{H}}{\partial x}(t) \right) \end{aligned} \tag{A.5}$$

This is simple Euler integration, which is even more clear if we use the Hamiltonian in equation (A.2).

$$\begin{aligned} x(t+\Delta t) &= x(t) + \Delta t \, \frac{p(t)}{m} \\ p(t+\Delta t) &= p(t) + \Delta t \, F(x(t)) \end{aligned} \tag{A.6}$$

The common Hamiltonian in equation (A.2) was written on the form

$$\mathcal{H}(x,p) = V(x) + T(p)$$

where the p and x dependendencies are separate. Since differentiation is linear, the corresponding operator is separable too.

$$D_{\mathcal{H}} = D_V + D_T$$

Applying D_T twice gives

$$\begin{aligned} D_T^2 z &= D_T \left(\frac{\partial z}{\partial x} \frac{\partial T}{\partial p} - \frac{\partial z}{\partial p} \overbrace{\frac{\partial T}{\partial x}}^{=0} \right) \\ &= \overbrace{\frac{\partial}{\partial x} \left(\frac{\partial z}{\partial x} \frac{\partial T}{\partial p} \right)}^{=0} \frac{\partial T}{\partial p} - \frac{\partial}{\partial p} \left(\frac{\partial z}{\partial x} \frac{\partial T}{\partial p} \right) \overbrace{\frac{\partial T}{\partial x}}^{=0} \\ &= 0 \end{aligned} \tag{A.7}$$

and similarly, $D_V^2 z = 0$. This means that all higher order terms in the taylor expansion in equation (A.4) vanish, and

$$\begin{aligned} \exp(\Delta t D_T) &= 1 + \Delta t D_T \\ \exp(\Delta t D_V) &= 1 + \Delta t D_V \end{aligned}$$

These operator exponentials look like the simple truncation in equation (A.5), and can be straight-forwardly computed in the same way. Because they are computable, we can approximate the full time evolution operator by splitting the exponential into two parts

$$\exp(\Delta t D_{\mathcal{H}}) = \exp(\Delta t (D_V + D_T)) \approx \exp(\Delta t D_V) \exp(\Delta t D_T)$$

where each part can be applied independently. Applying the approximated operator gives us

$$\begin{aligned}\exp(\Delta t D_V)\exp(\Delta t D_T)(x,p) &= \exp(\Delta t D_V)(x + \Delta t \frac{\partial T}{\partial p}(p), p) \\ &= \left(x + \Delta t \frac{\partial T}{\partial p}(p), p + \Delta t \frac{\partial V}{\partial x}\left(x + \Delta t \frac{\partial T}{\partial p}(p)\right)\right)\end{aligned} \tag{A.8}$$

or, written with separated components for the common Hamiltonian

$$\begin{aligned} x(t+\Delta t) &= x(t) + \Delta t\, \frac{p(t)}{m} \\ p(t+\Delta t) &= p(t) + \Delta t\, F(x(t+\Delta t)) \end{aligned} \tag{A.9}$$

which looks very similar to euler integration, equation (A.6). The difference is that the position and momentum are updated one at a time instead of simultaneously. I call this method *symplectic Euler integration.*

We can improve the approximation by splitting up the operator a different way. In general, the approximation of order k is

$$\exp(\Delta t D_{\mathcal{H}}) = \prod_i \exp(b_i \Delta t D_V)\exp(a_i \Delta t D_T) + \mathcal{O}\left(\Delta t^k\right), \qquad \sum_i a_i = \sum_i b_i = 1 \tag{A.10}$$

for some coefficients a_i and b_i. The first-order approximation uses $a_1 = b_1 = 1$, giving us the symplectic Euler method in equation (A.9). The second-order approximation is

$$\exp(\Delta t D_{\mathcal{H}}) = \exp\left(\frac{1}{2}\Delta t D_V\right)\exp(\Delta t D_T)\exp\left(\frac{1}{2}\Delta t D_V\right) + \mathcal{O}\left(\Delta t^2\right) \tag{A.11}$$

which is exactly the *velocity Verlet* method in section 2.2. The coefficients a_i and b_i of higher-order approximations can be found by comparing the taylor expansions of $\exp(\Delta t(D_V + D_T))$ and $\exp(\Delta t D_V)\exp(\Delta t D_T)$.

I have claimed that this method results in symplectic integrators, now let me argue for why that is. A continuous Hamiltonian system is by definition symplectic, and so is the corresponding time evolution operator $\exp(tD_{\mathcal{H}})$. The operators $\exp(tD_T)$ and $\exp(tD_V)$ are therefore symplectic, since they correspond to the Hamiltonians of a free particle and a massless particle. It follows that any integration method we construct using equation (A.10) is symplectic as well.

In addition to improved stability, a benefit of symplectic integrators is that they update position and momentum one at a time. Hence, there is no need to keep data for more than one timestep at a time in memory.

B. Solving Langevin's Equation

Langevin's equation has the form

$$\frac{d}{dt}v(t) = -\eta v(t) + g(t)$$

and we want to find an algorithm for solving this equation for each timestep Δt. We rearrange the equation and multiply by the integrating factor $e^{\eta t}$ so that we can use the product rule backwards

$$e^{\eta t} g(t) = e^{\eta t}\frac{d}{dt}v(t) + \eta e^{\eta t} v(t) = \frac{d}{dt}\Big(e^{\eta t} v(t)\Big)$$

This allows us to integrate for one timestep

$$\int_0^{\Delta t} \mathrm{d}t\ e^{\eta t} g(t) = e^{\eta \Delta t} v(\Delta t) - v(0)$$

and write the solution

$$\begin{aligned} v(\Delta t) &= v(0)e^{-\eta\Delta t} + \int_0^{\Delta t} \mathrm{d}t\ e^{\eta(\Delta t - t)} g(t) \\ &= v(0)e^{-\eta\Delta t} + R(t) \end{aligned} \tag{B.1}$$

Since $g(t)$ is a random process, $R(t)$ is too, and we need to know its mean and variance. The stochastic force $g(t)$ has zero mean and is uncorrelated in time

$$\begin{aligned} \langle g(t) \rangle &= 0 \\ \langle g(t) g(t') \rangle &= \delta_{t,t'} \frac{2\eta k_B T}{m} \end{aligned}$$

With this, we can compute the mean

$$\langle R(t) \rangle = \sqrt{\frac{2\eta k_B T}{m}} \int_0^{\Delta t} \mathrm{d}t\ e^{\eta(\Delta t - t)} \langle g(t) \rangle = 0 \tag{B.2}$$

and the variance

$$\begin{aligned} \Big\langle R(t)^2 \Big\rangle &= \int_0^{\Delta t} \mathrm{d}t\ e^{\eta(\Delta t - t)} \int_0^{\Delta t} \mathrm{d}t'\ e^{\eta(\Delta t - t')} \langle g(t) g(t') \rangle \\ &= \frac{2\eta k_B T}{m} e^{2\eta\Delta t} \int_0^{\Delta t} \mathrm{d}t\ e^{-2\eta t} \\ &= \frac{2\eta k_B T}{m} e^{2\eta\Delta t} \frac{e^{-2\eta\Delta t} - 1}{2\eta} \\ &= \frac{k_B T}{m}\Big(1 - e^{2\eta\Delta t}\Big) \end{aligned} \tag{B.3}$$

Using equations (B.1) to (B.3), we can now write down an algorithm for solving Langevin's equation,

$$v(t+\Delta t) = cv(t) + \sqrt{\frac{k_B T}{m}}\sqrt{1-c^2}\,G, \quad \text{where } c = e^{\eta\Delta t} \tag{B.4}$$

and G is an independent gaussian random number with zero mean and unit variance. For multidimensional systems, the algorithm is applied for each dimension independently.

The Langevin equation introduces a stochastic force and a damping, which destroys energy conservation and instead drives the system to the temperature T. Nevertheless, the formalism used to derive the symplectic integrators in appendix A can be used to find a particularly accurate way of applying equation (B.4), see [8].

Consider a system with a hamiltonian operator $D_{\mathcal{H}}$, consisting of kinetic and potential terms, D_T and D_V, but now also a third term D_L, corresponding to the Langevin forces.

$$\exp(\Delta t D_{\mathcal{H}}) = \exp(\Delta t(D_T + D_V + D_L))$$

This operator can be split like before, and a simple and accurate split is

$$\exp(\Delta t D_{\mathcal{H}}) \approx \exp\left(\frac{1}{2}\Delta t D_L\right)\exp\left(\frac{1}{2}\Delta t D_T\right)\exp(\Delta t D_V)\exp\left(\frac{1}{2}\Delta t D_T\right)\exp\left(\frac{1}{2}\Delta t D_L\right)$$

which means using one iteration of equation (B.4) with timestep $\Delta t/2$ before and after one step with the velocity Verlet method in equation (A.11).

C. Intersections

The simplest way of solving for intersections is to represent lines parametrically

$$\mathbf{x} = \mathbf{a} + t\mathbf{b}$$

and solve for the parameter t. Representing lines on parametric form is not only simple, but also flexible, since it can represent not only infinite lines, but also half-lines (rays) and line segments. The point $\mathbf{a}$ is then the starting point of a line segment or a ray, and $\mathbf{b}$ describes the extent of a segment. There is an intersection

- For a line segment, if $t \in [0, 1]$.
- For a ray, if $t \geq 0$.
- For an infinite line, if $t \in \mathbb{R}$.

Not only is this method simple, it is also coordinate free, meaning it uses only vectors, and works in any coordinate system. This is not true for any representation, for example, the function representation $y = kx + m$ cannot represent lines parallel to the y-axis of the coordinate system. Another benefit of coordinate-free expressions is that they are easier to generalize to higher dimensions.

C.1. Line-circle intersection

We have a line

$$\mathbf{x} = \mathbf{a} + t\mathbf{b}$$

and a circle

$$(\mathbf{x} - \mathbf{c})^2 = R^2$$

with center point $\mathbf{c}$ and radius R. Here $\mathbf{x}$ is the potential intersection point, that lies on both line and circle. Solving these equations for t gives us a point on the line where they intersect, or a criterion for when they do not intersect. Using $\mathbf{d} = \mathbf{c} - \mathbf{a}$:

$$R^2 = ((\mathbf{a} + t\mathbf{b}) - \mathbf{c})^2 = (t\mathbf{b} - \mathbf{d})^2 = t^2\mathbf{b}^2 - 2t\mathbf{d} \cdot \mathbf{b} + \mathbf{d}^2$$

Solving for t we get

$$t_{+,-} = \frac{\mathbf{d} \cdot \mathbf{b} \pm \sqrt{(\mathbf{d} \cdot \mathbf{b})^2 + \mathbf{b}^2(R^2 - \mathbf{d}^2)}}{\mathbf{b}^2}$$

When the argument of the square root is negative, there are no real-valued solutions and no intersections. If there are solutions, they generally come in pairs, since a line usually intersects a circle two times. The only exception is when the square root vanishes, in which case the line is tangent to the circle.

Here we have assumed that the vectors are two-dimensional, but the derivation works just as well with vectors of arbitrary dimension. In particular, it works just as well with a sphere in 3D as it does with a circle in 2D.

C.2. Line-line intersection

Line-line intersection can be solved in a coordinate-free way using the outer product, denoted by a wedge $\wedge$. The outer product is the 2D version of the cross product in 3D. I define it as

$$\mathbf{a} \wedge \mathbf{b} = |\mathbf{a}||\mathbf{b}| \sin\theta$$

where θ is the angle between the vectors. The outer product is anticommutative

$$\mathbf{a} \wedge \mathbf{b} = -\mathbf{b} \wedge \mathbf{a}$$

which implies the useful property

$$\mathbf{a} \wedge \mathbf{a} = 0.$$

Using orthonormal coordinates, the outer product is calculated

$$(a_x, a_y) \wedge (b_x, b_y) = a_x b_y - a_y b_x.$$

We write both lines in parametric form

$$\mathbf{x}' = \mathbf{x} + t\mathbf{v}, \quad \mathbf{x}' = \mathbf{a} + s\mathbf{b}.$$

Intersection occurs when these are equal

$$\mathbf{x} + t\mathbf{v} = \mathbf{a} + s\mathbf{b} \quad \Longrightarrow \quad \mathbf{x} - \mathbf{a} = s\mathbf{b} - t\mathbf{v}.$$

Outer multiplying by $\mathbf{v}$ and $\mathbf{b}$, we can eliminate t and s, respectively

$$\begin{aligned}(\mathbf{x} - \mathbf{a}) \wedge \mathbf{v} &= s\mathbf{b} \wedge \mathbf{v} - \cancel{t\mathbf{v} \wedge \mathbf{v}} \\ (\mathbf{x} - \mathbf{a}) \wedge \mathbf{b} &= \cancel{s\mathbf{b} \wedge \mathbf{b}} - t\mathbf{v} \wedge \mathbf{b}\end{aligned}$$

and we can solve for s and t

$$\begin{aligned}s &= \frac{(\mathbf{x} - \mathbf{a}) \wedge \mathbf{v}}{\mathbf{b} \wedge \mathbf{v}} \\ t &= \frac{(\mathbf{x} - \mathbf{a}) \wedge \mathbf{b}}{\mathbf{b} \wedge \mathbf{v}},\end{aligned}$$

where we used $\mathbf{v} \wedge \mathbf{b} = -\mathbf{b} \wedge \mathbf{v}$.

References

[1] Victor B. Explorable explanations 2011. http://worrydream.com/ExplorableExplanations (accessed September 14, 2016).

[2] Victor B. Bret victor, beast of burden n.d. http://worrydream.com (accessed December 15, 2016).

[3] Hart V, Case N. Parable of the polygons 2014. http://ncase.me/polygons/ (accessed September 14, 2016).

[4] Schelling TC. Dynamic models of segregation†. Journal of Mathematical Sociology 1971;1:143–86.

[5] Gingold C. Earth: A primer. 2015.

[6] Swope WC, Andersen HC, Berens PH, Wilson KR. 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 1982;76:637–49.

[7] Thijssen J. Computational physics. Cambridge university press; 2007.

[8] Bussi G, Parrinello M. Accurate sampling using langevin dynamics. Physical Review E 2007;75:056707.

[9] Victor B. Tangle 2011. http://worrydream.com/Tangle (accessed September 22, 2016).

[10] Scalable vector graphics (svg) 1.1 (second edition) 2011. https://www.w3.org/TR/SVG11/ (accessed January 4, 2017).

[11] WebGL specification 2014. https://www.khronos.org/registry/webgl/specs/1.0/ (accessed January 4, 2017).

[12] HTML5, 4.11.4 the canvas element 2014. https://www.w3.org/TR/html5/scripting-1.html#the-canvas-element (accessed January 4, 2017).

[13] WebAssembly 2016. https://webassembly.github.io (accessed September 26, 2016).

[14] Emscripten 2015. https://kripken.github.io/emscripten-site/ (accessed September 26, 2016).

[15] Visualizing algorithms 2014. https://bost.ocks.org/mike/algorithms/ (accessed January 4, 2017).