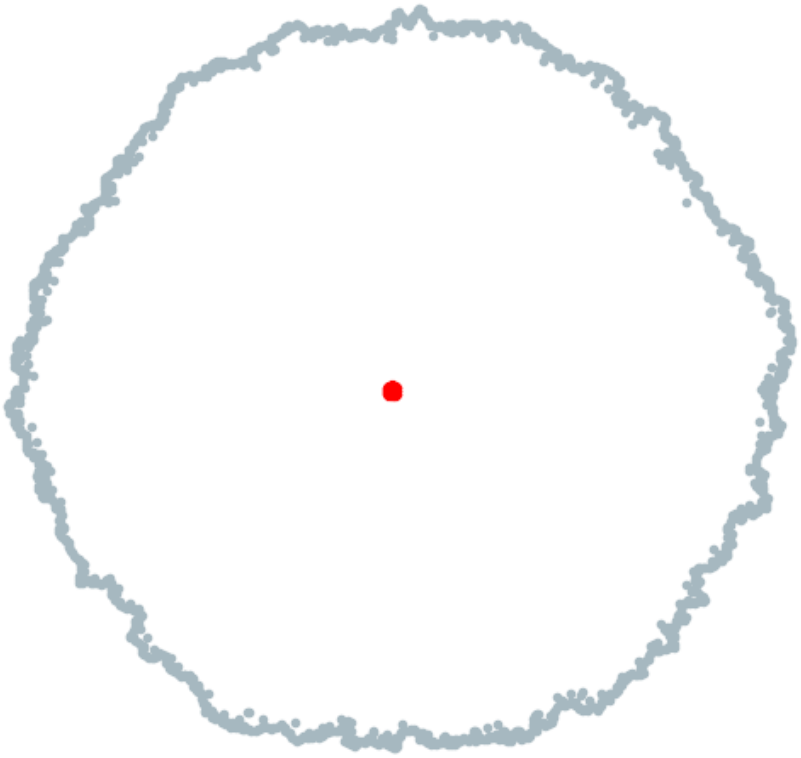


Universal interface growth in substrate etching processes

Master's thesis in Complex Adaptive Systems

TOBIAS WALLSTRÖM

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Universal interface growth in substrate etching processes

TOBIAS WALLSTRÖM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2025

Universal interface growth in substrate etching processes
TOBIAS WALLSTRÖM

Supervisor: Johannes Hofmann, Department of Physics, University of Gothenburg
Examiner: Johannes Hofmann, Department of Physics, University of Gothenburg

Master's Thesis 2025
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A simulated interface between atoms and holes on a honeycomb lattice.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Universal interface growth in substrate etching processes
TOBIAS WALLSTRÖM
Department of Physics
Chalmers University of Technology

Abstract

Etching is a process that has been used both in art and in industry for hundreds of years. Although well understood on larger spatial scales, when acting on a small-scale two-dimensional lattice, more intricate effects seem to take place. This project has aimed at using a Monte Carlo simulation to study etching on a honeycomb lattice and describing the effect of the underlying crystal structure on the evolution of the system with different relative decay rates for solitary atoms.

The implementation shows that at short times the crystal structure very much plays a role in the shapes that develop by creating shapes resembling a hexagon with slightly rounded corners. As time passes, the holes created become more circular, and especially for increasing reactivity for solitary atoms, the shape becomes nearly completely circular at large times. This is quantified by several metrics such as area error measurement as well as radial error measurements. We find that the radius grows linearly with time, indicating that as more atoms become exposed, more reactions occur, but as the interface grows larger, more atoms need to be annihilated to increase the radius. In future work, we would like to study longer time frames for more reactive conditions to see where the hexagonal error will plateau.

Keywords: etching, Gillespie algorithm, annihilation, Monte Carlo, lattice, interface, surface growth

Acknowledgements

I am extremely grateful to the Department of Physics at the University of Gothenburg for having me and making me feel welcome. In particular I would like to thank Johannes Hofmann for supplying the idea and for the guidance he gave, as well Enrique Rozas Garcia for letting me bounce ideas on him. I also want to thank Blaž Pridgar, Ole Fjeldså, and Gustaf Jonasson Johansson for being my proverbial rubber ducks when I was trying to locate bugs in my code.

Lastly, I want to thank friends and family for all the support throughout the years, I would not have been able to do it without them.

Tobias Wallström, Gothenburg, June 2025

Contents

1

Introduction

Typically, chemical removal reactions occur on the boundary between two or more domains of reactants, be it fluids, solids, or a combination of each. This boundary is called the interface. The etching process is part of the latter family, where a fluid or gas reacts with a solid. What differs between etching and other types of destructive reactions is that the goal of the process is to only react superficially, usually creating intricate patterns on the surface [1]. This might be straight lines, like for printed circuit boards, or more artsy patterns like the ones on Damascus steel knives or on paintings.

The important part for us is that etching is a surface reaction where one element dissolves another. It can be modeled as an annihilation process between a surface molecule A and an etchant molecule B described by

$$A + B \to \emptyset \tag{1.1}$$

or, more simply, as

$$A \to \emptyset \tag{1.2}$$

if only one part is of interest, for example such as when the etchant is plentiful [2].

This work is motivated by an interesting discovery made recently in one of the local labs: When etching gold on a graphene substrate with a PMMA mask, microscopic holes in the mask let etchant through and started creating holes. These holes, however, did not grow in circles as can be initially thought to happen, but rather showed a clear hexagonal shape. If this growth could be reasonably predicted, it can be used to engineer more efficient etching processes or help avoid pitfalls when creating designs to be etched.

1.1 Aim of the project

This project aims to use Monte Carlo simulations in order to quantitatively explore the effect on an underlying hexagonal lattice on interface growth. The model used will focus on annihilation of the type

$$A \to \emptyset,$$

where A is a lattice atom at the interface of the etching fluid. The reason for this choice is that we are interested in how the lattice structure plays a role in

the shapes generated, and if it will persist even without phenomena like diffusion or variable etchant concentration. We will also briefly look at how the surface roughness increases by means of the growth factor β in the Kardar-Parisi-Zhang (KPZ) equation [3] for one spatial dimension. As we saw the results on a honeycomb lattice, that is what will be explored in this paper. Most simulations will look at early to medium time scales. A variable reaction rate that depends on the number of remaining nearest neighbors will also be studied to determine their effect on the system's evolution over time.

2

Theory

In many physical models, a very common way to calculate time evolution is by using mean field theory. The advantage is that this is very computationally efficient, since if one knows how the average particle behaves then all that is needed is to apply the same transformation for all of them. For example, looking at radioactive material we have a clear half-life for when half of the atoms will have decayed. This typically works for very large numbers of atoms and for higher dimensions, but the predictability of the system decreases as the number of atoms decreases.

The same principle can be seen in statistical models. Since it is a random and finite process that's usually rather small, there is only a very small probability that a system perfectly matches up with its mean field results, though it will usually be close. This poses another problem, which arises when we don't quite know the mean field of the reaction, or outright assume it to be something other than it is. This is where the theory breaks down and we must actually do the simulations in order to study the evolution of the system over time.

2.1 One-step reaction models and Gillespie algorithm

In decay models, any one atom with a reactivity λ has a decay probability of

$$P(A \to \emptyset) = \lambda dt \tag{2.1}$$

during a small time span dt. In a system of n independent atoms this simply changes to

$$P\left(nA \to (n-1)A\right) = n\lambda dt \tag{2.2}$$

for any sufficiently small dt such that $P \ll 1$. This leads to the probability of k reactions happening in the same time span dt will be

$$P\left(nA \to (n-k)A\right) = \binom{n}{k} \lambda dt^k, \tag{2.3}$$

which is very unlikely for any $k > 1$. This means that the reaction can be simplified and modeled according to a simple one-step master equation, given by

$$\frac{\partial}{\partial t} P(n,t) = \lambda[(n+1)P(n+1,t) - nP(n,t)], \tag{2.4}$$

where λ is the reaction rate and n the number of atoms that are available for the reaction. Discretizing the equation results in

$$P(n, t + dt) = \mu_{n+1} dt P(n + 1, t) + (1 - \mu_n dt) P(n, t), \tag{2.5}$$

with $\mu_n = \lambda n$. To simulate the time evolution, one can therefore simply test every small time step to see if an atom reacts or not. This, however, poses a problem. In order to get accurate results λdt must be sufficiently small to capture at most one reaction. This works reasonably well in the beginning when n is large, but as n becomes smaller and smaller each next decay becomes less and less probable. By extension it will take longer and longer between each decay as time increases, represented by more steps where nothing is happening. Luckily there is a solution to the problem.

Instead of looking at the decay probability at time $t + dt$, we look at the probability q_n of no etching reactions happening in any other time span τ. By looking at the time interval $\tau + d\tau$ we can postulate

$$q_n(\tau + d\tau) = q_n(\tau) P(\text{no decay in } d\tau) = q_n(\tau)(1 - \mu_n d\tau) \Rightarrow \frac{\mathrm{d}}{\mathrm{d}\tau} q_n(\tau) = -\mu_n q_n \tag{2.6}$$

We also know that since the decay rate is finite, at $\tau = 0$ no decays can have occurred, meaning that $q_n(0) = 1$, giving us the solution of an exponential $q_n(\tau) = e^{-\mu_n \tau}$. Going back to the decay probability at time τ it is defined as $p_n(\tau)d\tau$, we see that it is just the probability of pausing until time τ and then a reaction in the next interval $d\tau$, resulting in

$$p_n(\tau) d\tau = e^{-\mu_n \tau} \mu_n d\tau \Rightarrow p_n(\tau) = \mu_n e^{-\mu_n \tau} \tag{2.7}$$

This means that the decay probability of one out of n atoms follows an exponential distribution, which is independent of starting time, allowing us to attack the problem from the other way around. Instead of checking every small time step whether a decay has occurred we can generate the time between two decays, and then at the new time let one atom decay, and repeat until the end of the simulation has been reached. This is known as the Gillespie algorithm.

There is, however, one slight change in this case, as for some shapes the annihilation exposes new atoms that can react with the etchant, see Figure 2.1. If there is about equal parts of etchant compared to atoms or if the etchant moves slowly, the newly exposed atoms will have a lower rate of decay. For simplicity's sake we will assume that the replenishment of etchant is fast, which will let us assume a constant μ_n for each over time, only being possibly dependent on the number of free nearest neighbors, which will be explained more in Section 2.3.

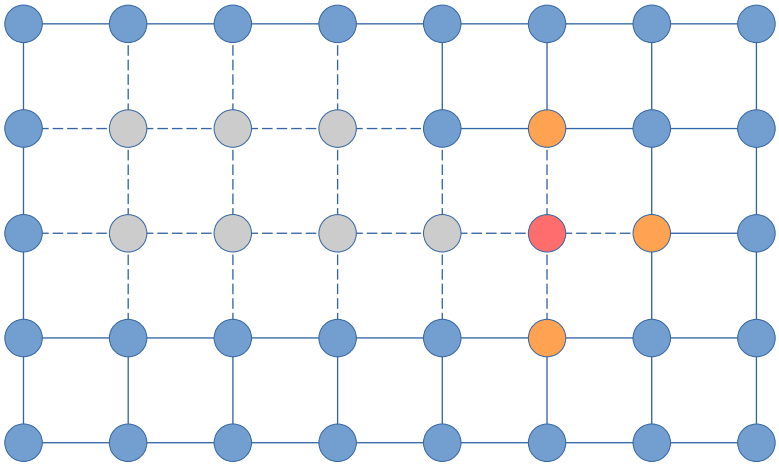


Figure 2.1: Atoms on a lattice. Blue points represent existing atoms, and gray points atoms that have already been removed. An etching event (in red) exposes three new atoms that can be etched in a subsequent step (orange). Since the red atom has been etched away, and as such is no longer part of the interface, the net change in interface size is 2.

2.2 Two-dimensional lattice structures

A lattice is defined by an infinite set of points that perfectly tile a plane. Each point in the lattice is reached from the origin by an integer combination of linearly independent basis vectors [4]. In other words, it is translationally invariant with respect to the discrete basis vectors. Some stipulate that the set of points need to be reachable by a set of primitive basis vectors [5], but in this thesis we will use the former definition as that allows us to call the honeycomb a lattice, whereas the latter definition does not. We will distinguish the two by calling them compact if they can be represented by primitive vectors.

A lattice can always be divided into unit cells such that the unit cells perfectly tile the plane. If the unit cell also happens to contain exactly one lattice point it will be called a primitive unit cell. One such cell is the Wigner-Seitz cell. In the circumstance that the lattice is not a compact one, the analog of the Wigner-Seitz cell is called a Voronoi cell. The Voronoi cell of the honeycomb lattice is triangular as can be seen in Figure 2.2. The given area for the Voronoi cell is

$$A_{Voronoi} = \frac{9}{4\sqrt{3}}c^2, \tag{2.8}$$

with c being the distance to the nearest neighbor. In the case of graphene, for example, we have $c = 1.42$ Å [6], which gives us a Voronoi cell area of roughly $A_{Voronoi} \approx 2.62$ Å^2.

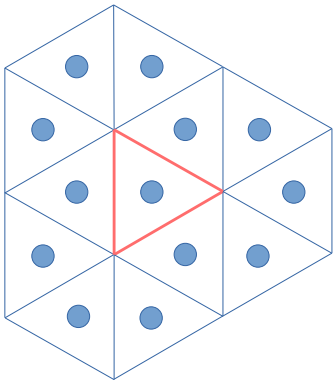
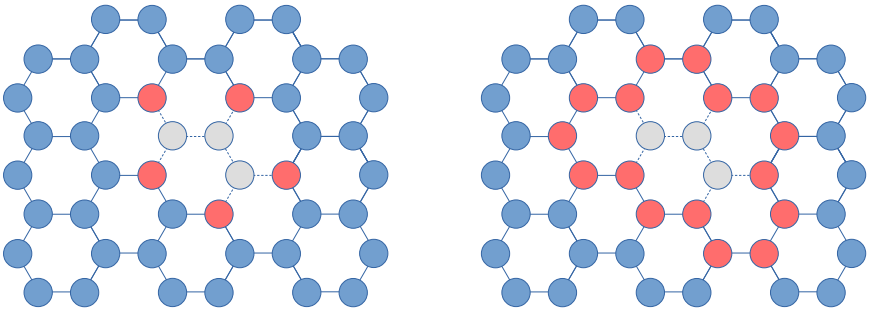


Figure 2.2: The Voronoi cell of the honeycomb lattice. The borders of the center cell have been colored red.

2.3 Interface structures and etching rates

In order to decide which atom is next in line for annihilation, we must first see which ones are exposed to the hole, i.e. which atoms form the interface. By looking at an atom's nearest neighbors we can get a sense of how exposed it is. For this we need to define the nearest neighbors of an atom. The simplest model is the close-packed hexagonal lattice. In it, all neighbors are directly connected and have the same distance between each other. In a close-packed square, the orthogonal neighbors will always be considered nearest neighbors, but the diagonal ones may or may not be. Similarly, in a honeycomb lattice an atom can be considered exposed either if the cell it touches has a free corner or if the lattice edge is free. Choosing one over the other could greatly change the characteristics of the annihilation.



(a) Exposed atoms if lattice edge is free

(b) Exposed atoms if it has a "free corner"

Figure 2.3: Zoomed in version of honeycomb lattice. Blue represents the existing atoms, gray the removed ones (kept to help illustrate lattice structure), and red represents the nearest neighbors which have a possibility of being annihilated.

Once the nearest neighbor system has been chosen, we can begin with the Monte Carlo simulation. There is one more choice needed to be done first, and that is

to decide the relative rates of annihilation. From chemistry we have the general expectation that a larger surface area leads to more reactivity, though it could prove to be different at these scales. Here, we could scale the etching rate linearly by the number of free neighbors, we could square it, or we could give it any other proportions. On the other hand, we could say that every atom with a free neighbor is equally likely to be annihilated. This could, for an atom with η free nearest neighbors, be simplified into

$$\mu_i(\eta) = \begin{cases} 0 & \eta = 0 \\ \eta^\sigma \lambda & \eta > 0,\ \sigma \in \mathbb{R} \end{cases} \tag{2.9}$$

making the uniform probability correspond to $\sigma = 0$, linear to $\sigma = 1$ and so on. In the odd case of punishing many neighbors, σ can be set to a negative value. In that case, the total reaction rate would be

$$\Gamma = \sum_i \mu_i(\eta) \Rightarrow f(\tau) = \Gamma e^{-\Gamma\tau} \tag{2.10}$$

and the probability of any particular atom α being the one to decay would be

$$P(\text{atom } \alpha \text{ removed}) = \frac{\mu_\alpha}{\sum_i \mu_i} = \frac{\eta_\alpha^\sigma}{\sum_i \eta_i^\sigma}. \tag{2.11}$$

This would give us the expected time to decay to be around $\langle\tau\rangle \approx 1/\Gamma$.

2.4 Characterizing the shape of an etched interface

After letting the system develop over time, either by time passed or by number of atoms removed, it will develop a characteristic interface shape. In order to accurately describe how the system evolves, there needs to be a way to quantify the likeness of the different shapes. It would be possible to manually do a fit and "hand-wave" the approximation, but not to a satisfactory degree. In order to quantify the shapes, we will look at the lattice itself, and by individually comparing the radii and the area between the simulated shape and an ideal circle (and hexagon) we can compare the best fit.

2.4.1 Radius-based errors

There are two major ways of calculating the error of the radius. One way is to measure the mean absolute error (MAE) of the radius, and the other is the root mean square error (RMSE) of the radius.

The MAE for N particles is simply

$$E_{MAE}^{(r)} = \min_r \frac{1}{N} \sum_k |d_k - r|, \tag{2.12}$$

where d_k is the distance of particle k from the center, and r is the radius of the shape the interface is compared to. For the same definitions of d_k and r, the RMSE is

$$E_{RMSE}^{(r)} = \min_r \sqrt{\frac{1}{N}\sum_k (d_k - r)^2}. \tag{2.13}$$

Calculating the error from a circle is easy, as the radius is constant. However, the error is more difficult to calculate for the hexagon. First, the radius of the hexagon needs to be parametrized by

$$r_{hex}(\theta) = \frac{\rho\sqrt{3}}{2\sin\left[\theta - \theta_0 - \frac{\pi}{3}\left(\left\lfloor\frac{3(\theta-\theta_0)}{\pi}\right\rfloor - 1\right)\right]}, \quad \theta, \theta_0 \in [0, 2\pi), \tag{2.14}$$

with ρ, θ, and θ_0 as outlined in Figure 2.4. Then, the MAE and RMSE can be calculated with each particle on the interface using their polar coordinates. The growth of these errors over time can be characterized with a factor β, such that $E_{MAE,RMSE} \sim t^\beta$.

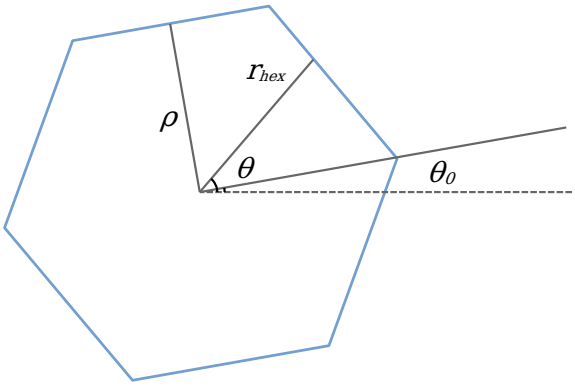


Figure 2.4: Illustration of the variables used in the radius calculation. ρ is the shortest distance from the center to the edge of the hexagon, θ is how far along the hexagon has been traced from the horizontal line (dashed), and θ_0 is the rotational offset of the hexagon.

2.4.2 Area overlap

Another way of calculating the shape is to check how much of the area overlaps. Like before, the hexagon is parametrized by its radius (2.14), and the circle by its radius. One can then see which points are in the circle or hexagon that shouldn't be, and vice versa, i.e. the symmetric difference between the two shapes being compared. After that we can simply plot the error.

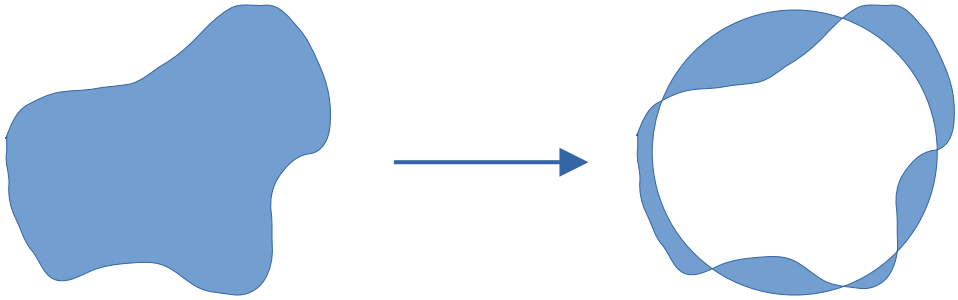


Figure 2.5: Overlaying a circle on an arbitrary shape. The shaded area in the right-hand figure is the total area error, i.e., the parts of the circle that are outside of the shape and the parts of the shape that are outside of the circle.

By minimizing the area overlap with radius and angle sweeping, we get the best possible fit and we can then compare how much the evolved shape differs from the ideal cases.

However, we need to be able to compare how different the simulation is from each shape over time, and that will not be possible as-is, as the area doesn't scale up equally for all simulations over time. Thus a normalization metric is needed. The easiest, and possibly best, is to compare how big of a percentage of the area is incorrect in comparison to the area of the hole. Fortunately, the size of the hole perfectly correlates to the number of removed atoms, thanks to the concept of Voronoi cells as mentioned in Figure 2.2. And since the unit length is constant over the whole lattice (and arbitrary) we can simply just use the number of removed atoms to normalize the error. This finally gives us the normalized error term

$$E_A = \min_s \frac{A_s}{A_h} = \min_s \frac{N_s}{N_r}, \tag{2.15}$$

where A_s is the shaded area, A_h the area of the hole, or in the discrete case N_s and N_r being the number of shaded and removed atoms, respectively. Figure 2.6 shows an illustration of the area error as a function of radius r.

In the case of the circle, this definition would give rise to an error of 1 for radius 0, and an infinite error as r approaches infinity, with a minimum somewhere in between.

The error would only be zero in the case of a perfect circle. This is illustrated in Figure 2.7. The same can be extrapolated to the hexagon, or any other shape that one might want to use for comparison.

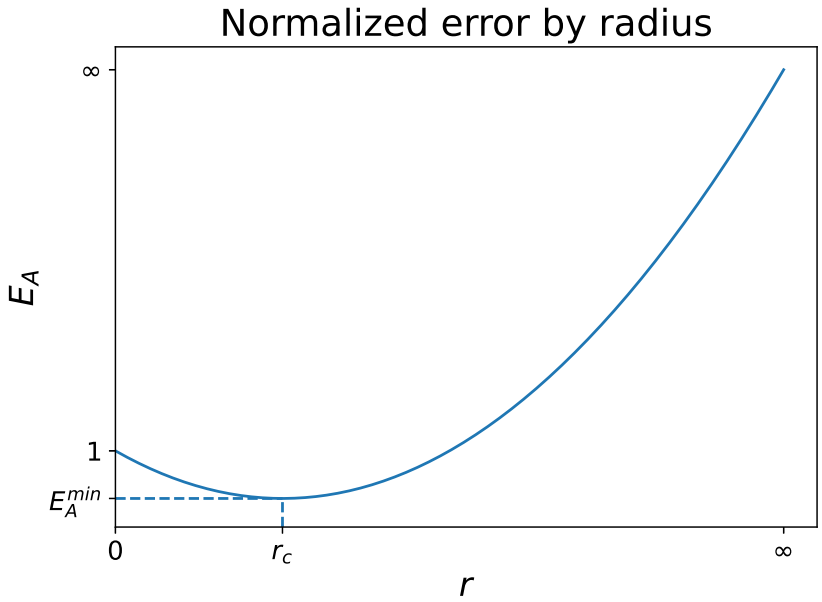


Figure 2.6: Circle area as a function of its radius. At $r = 0$ we will have an error of 1, that decreases until some radius $r_c \in [\min_i d_i, \max_i d_i]$ for which we have the minimum error, after which it increases.

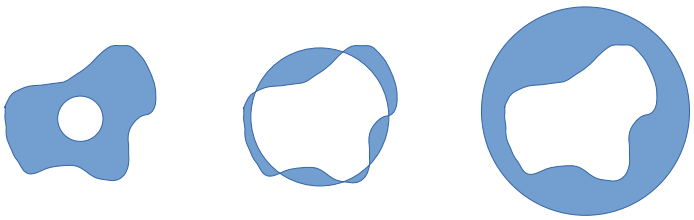


Figure 2.7: Visualizing overlap with different radii for the circle. The shaded area is the smallest for a radius around the one in the middle. As the radius increases past that, the error will increase.

3

Methods

In this section, we discuss the implementation used to study the system. The main task behind the algorithm is to identify the atoms on the lattice that can be annihilated and then choose one of those at random to be removed.

Since the system modeled is two-dimensional and can be seen as a set of equidistant points, some lattice structures can be very easily simulated. The first one that comes to mind is the simple compact square lattice, $\mathbb{Z}^2$: As all atoms are evenly spaced, each element in an array can represent one grid point in the lattice. A value of 1 can represent an atom being present at that point whereas a 0 can represent a hole in the lattice. From this we can easily create mathematical formulae to initialize the lattice in any way we want to, as outlined below.

When simulating the annihilation of atoms, there are a few ways of deciding which one to annihilate next. Since we want the domain to grow from the initial condition, it should only be able to spread via its nearest neighbors. In order to find how many of the nearest neighbors are free we need to look at the atom's surroundings. This can easily be done with a discrete 2D convolution applied on the lattice holes (i.e. $1 - A_{ij}$, where A_{ij} is the atom representation of the lattice) using a 3×3 kernel, as seen in Figure 3.1. If we want all eight adjacent elements (orthogonal and diagonal) to be considered as neighbors we get K_1. If desired, the diagonal nearest elements can also be ignored, giving us K_2.

$$K_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad K_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.1}$$

However, there's still a step missing. We have the number of free neighbors, but in order to get the interface we need to tick one more box: There needs to be an atom at the lattice position. So to get the final interface we need to mask the nearest neighbors with the lattice of atoms to apply it to the right ones. This is simply done with an element-wise multiplication of the atom representation. Only then can we apply the Monte Carlo simulation by repeatedly choosing an atom at random with probabilities proportional to their rates described by (2.9).

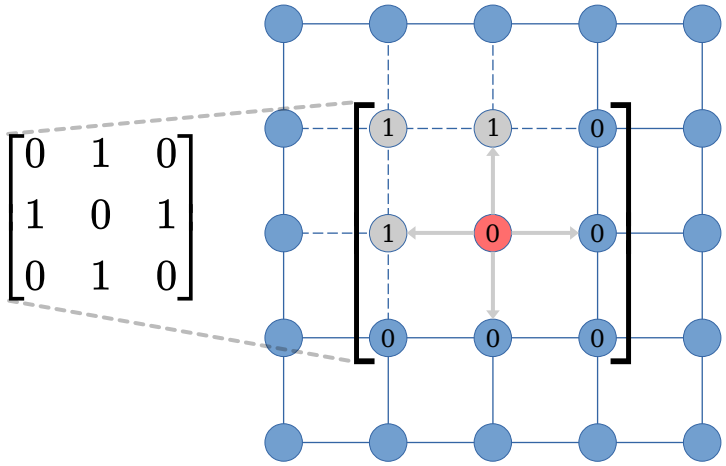


Figure 3.1: Calculating the number of free nearest neighbors for the red atom with the K_2 kernel seen in (3.1). When calculating the number of free nearest neighbors, a hole is considered a 1 and an atom a 0. By summing the element-wise products we get that the number of free nearest neighbors is 2. If we instead use kernel K_1, the diagonals would also be included, giving us 3 free nearest neighbors. Doing this for every atom gives us the nearest-neighbors matrix.

Once the atom is removed, the area surrounding the removed atom is recalculated with respect to free nearest neighbors and interface status. The time passed also needs to be added each iteration, as it is not with the number of atoms removed. Since we are using the Gillespie algorithm we can simply add $\tau \sim \exp(\Gamma)$ for each removed atom. For our intents we can set the scale factor to 1 since that equally changes the time scale for all iterations. Following that, a check can be done to see if a time threshold has been passed, typically 1 time unit but it can be chosen freely, and if so we check the similarity to the different shapes, see Section 3.2 for more details. This is looped until the end time has been reached, after which we save all relevant data from each run to be analyzed later. In the case of a simulation reaching the edge of the lattice, a flag is set in the metadata to signal that the simulation might include errors and should as such be run again, but with a larger lattice. An example can be found in Appendix A of how a simple square system can be implemented in Python using NumPy [7] and SciPy [8], with a fixed $\sigma = 1$.

3.1 Implementing different lattice structures

In order to study different shaped lattices, we need to represent the lattice and its vectors somehow. For example, we can look at the simple hexagonal (aka triangular) kernel and try to represent it on a grid:

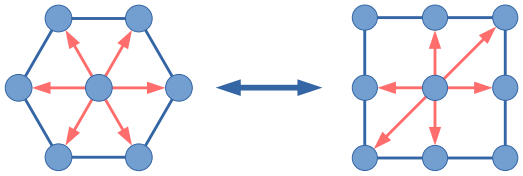


Figure 3.2: One way of representing a hexagonal close-packed (HCP) lattice in an array. Inspiration taken from Darjani, Koplik, and Pauchard [9].

This gives rise to the kernel

$$K_{hex} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \tag{3.2}$$

with a small coordinate change. We can translate each coordinate according to

$$(x, y) \to (x - \frac{y}{2}, \frac{\sqrt{3}}{2} y), \tag{3.3}$$

corresponding to the coordinate shift seen in Figure 3.2. In this scenario we can still represent the lattice as a grid, but with the caveat that the top left and bottom right corners of the array are not used, resulting in inefficiently used memory. This is fine, however, as memory is not the bottleneck in this simulation.

In order to simulate non-primitive lattices, there is still a possibility of using arrays in order to keep track of the crystal structure. What is done then is that a mask shaped like the lattice is put on top of the atom structure. This leads to more memory being used on unavailable points but with the advantage of it being very easy to implement and that it is easy to expand the domain.

Noticing that the honeycomb lattice has two out of every three lattice points occupied with the third one being empty, we can construct the lattice as

$$M_{honeycomb} = \begin{bmatrix} 0 & 1 & 1 & \dots \\ 1 & 0 & 1 & \\ 1 & 1 & 0 & \\ \vdots & & & \ddots \end{bmatrix} \tag{3.4}$$

repeated to tile the grid. This would result in a lattice like in Figure 3.3. In the

same way, a face-centered square structure could be modeled as

$$K_{FCC} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \quad M_{FCC} = \begin{bmatrix} 1 & 0 & \dots \\ 0 & 1 & \\ \vdots & & \ddots \end{bmatrix} \tag{3.5}$$

with K_{FCC} and M_{FCC} being the kernel and the tiled mask, respectively. Finally, if we wanted to use the definition of a free neighbor as in Figure 2.3b that would correspond to the kernel

$$K_{bighex} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}, \tag{3.6}$$

but with the same mask as for the normal honeycomb lattice as specified by (3.2). This one was not used, however, as the small kernel was computationally intensive enough.

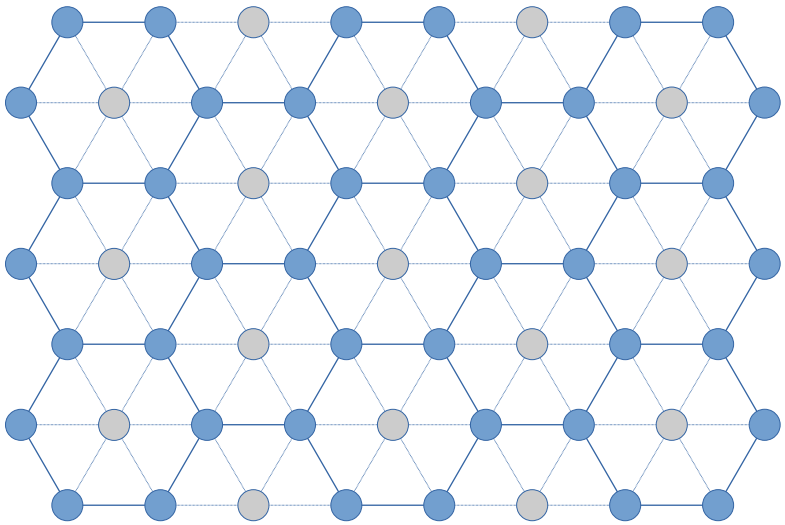


Figure 3.3: The result of combining the kernel from Figure 3.2 and the mask from (3.4). The gray atoms are invalid points and can as such not contribute to either the removal of atoms or to ”free” nearest neighbor count.

This is, however, where we start to encounter problems. For close-packed lattices it is possible to use every single array element in the calculation. On the contrary, when introducing the mask, more and more of the array becomes disused. For the hexagon, the valid cells take up 2/3 of the mask, i.e. we still waste 33% of the grid. This becomes even worse for the FCC structure, as only half is actually used, leaving the other half wasted. Creating even more sophisticated unit cells can easily waste

more and more of the given space. Thus, more creativity is required. Noticing that the diagonal elements of M_{hex} are ones, by rotating it 45 degrees a ones matrix can be obtained, resulting in better efficiency.

3.2 Shape classification

Two main ways of classifying the shapes have been established: the area error and the radial error, with the radial error having two different measurements. For the radial errors we first start with the center point of the initial condition and find the closest interface atom, as well as the furthest one. This becomes the basis of the radius sweep. Then, the distance to each atom along the interface from the center is calculated and compared to the radius of each circle, saving the best fit. The same was done for the hexagon, only with it also being tried for different rotations. To classify the radial error both the MAE and RMSE were calculated.

The area classification was done in much the same way. Initially, the area overlap was a simple XOR operation of a shape mask and the lattice itself. A circular mask was created with a radius $r \in [\min_i d_i, \max_i d_i]$ in units of the nearest-neighbor distance c, with a resolution of $\Delta r = 0.5c$. The same radius sweep was done for the hexagon as well, however for each r, the hexagon was rotated to find the best fit. Using the fact that the hexagon has a sixfold rotational symmetry, the rotation angle was set to $\theta_0 \in [-\pi/6, \pi/6]$ and had a resolution of $\Delta\theta_0 = 5°$. Using a higher resolution would give a more accurate reading, but is slower, which is why a 5 degree rotational resolution was chosen as a tradeoff.

3.3 Time-step analysis

Once the error classifications were defined, all that was needed was to compare each run equivalently. It would not be possible to compare after every single atom has been removed for two reasons; one reason being that it would take too much computation power for such a small change, and the other being that the number of atoms removed isn't constant with regards to time passed. Thus, another metric needs to be used when deciding how to compare each run.

Luckily, there is one thing that progresses predictably, and that is the time. To this end, we set discrete time steps when to perform the shape analysis to get a better understanding of how the system evolves. For this we can simply check whether t and $t + \tau$ are on opposite sides of such a point. If they are, then we know that the point has been passed due to the mean value theorem, and we also know what the system looked like at that time.

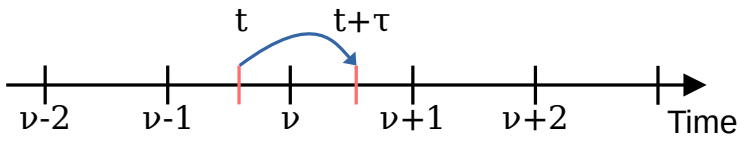


Figure 3.4: Whenever $t + \tau$ would cross a point of analysis, the error comparison would be run and each error saved to a file for later analysis.

Since we have that $t < \nu < t + \tau$, and we know that the next decay happens at $t + \tau$, then the system looks the same at t as it does at ν. This gives us an easy way of doing the analysis, as once τ is generated, we check if an analysis point has been passed before annihilating the atom, and if it has we perform the analysis. Only once the analysis has been performed do we let the next atom decay as the simulation resumes.

3.4 Enhancing performance

The model itself is written in Python, which is inherently a slow language. It is therefore important to make the code run faster in order to be able to gather data in a timely manner. The simplest way of doing this is by making use of (comparatively fast) libraries such as NumPy and SciPy. The more this can be done, the faster the code will run in general. However, using the libraries is only a necessary condition for the code to run somewhat fast, not a sufficient one.

The main problem that arises with creating bigger and bigger lattices is not actually the memory usage, but rather the computing time required for each step. Each annihilation requires a convolution step as well as several masking steps, whose computation time scales with the area of the lattice.

The model designed is building on removing one atom at a time from the lattice, and each iteration is dependent on the one before it. This leads to parallelization not being applicable inside any one simulation. However, there are lots of ways to speed up the process. For one, we can create a small bounding box around the atom being annihilated, and only recalculate the number of nearest neighbors in its surroundings, as the state of the lattice is only altered on a local level. Furthermore, since the hexagon is periodic every 1/6 of a rotation, the rotation sweep only needs to be done in the interval $\theta_0 \in [0, \pi/6)$.

When implementing the area overlap function more performance improvements can be done. One thing that may reduce the quality of the results is to have a lower resolution of the rotation of the hexagon, for example if we test every 5 degrees of rotation instead of every 1 degree we speed that part up by a factor of 5 with the caveat that the best fit might be in between two measurements. Another way of improving performance without sacrificing the results is when comparing the ideal

vs the measured shapes. By creating a bounding box around the union of the shapes, we save precious computational power by only comparing the points that may overlap and not the ones that definitely do not.

Furthermore, the naive way of implementing area overlap would be to create a mask of the ideal shape based on the radius and to then do an XOR of the two, counting however many of the resulting lattice points are non-zero. But, as previously said, mask operations are slow. Instead we can simply get the radius and angle from the center point to each atom (or hole), and check whether it is inside or outside of the ideal shape. Simply count the ones on the wrong side of the border to get the same result but an order of magnitude faster.

The final major improvement that can be made is to run several simulations in parallel. This is easily done within any multiprocessing library. Using one processor core per simulation essentially speeds it up linearly with respect to the number of cores assigned.

4

Results

In this section, we will discuss the results for two initial conditions, one at a time. For each initial condition we will compare the interfaces at their final time, the area error over time, the best-fit angle over time, and the radial errors over time, both the RMSE and the MAE. Then we will compare each error type by their scaling factor σ before finally taking a look at the growth patterns over time.

Each ensemble consists of 1,000 simulations. Before each ensemble, an initial condition was chosen. One run was picked at random whose interface would be shown along with the initial condition for that ensemble. The two starting patterns were a small circle with radius $r = 3.5d$, where d is the nearest neighbor distance. The hexagonal starting condition had a hexagon with $\rho = 30d$ rotated by $\theta_0 = 30°$. Furthermore, the base reaction rate λ was set to 1, allowing us to characterize time as t instead of λt in the plots.

The overarching pattern that can be seen is that the system starts off by growing in a pattern resembling a hexagon, but as time increases the error compared to a circle decreases faster than compared to a hexagon, though both decrease for low times. At time t_{cross}, the magnitudes of the errors cross each other. At late times the hexagonal radial errors start increasing faster, especially for higher σ.

4.1 Small circle starting condition

One randomly chosen run was chosen as a ”representative” of each relative reaction rate. We can see that as σ increases the shape not only gets rounder but there are also fewer solitary atoms inside the overarching shape. This might not hold true for every single run in the ensemble, but it gives a glimpse of what they might look like.

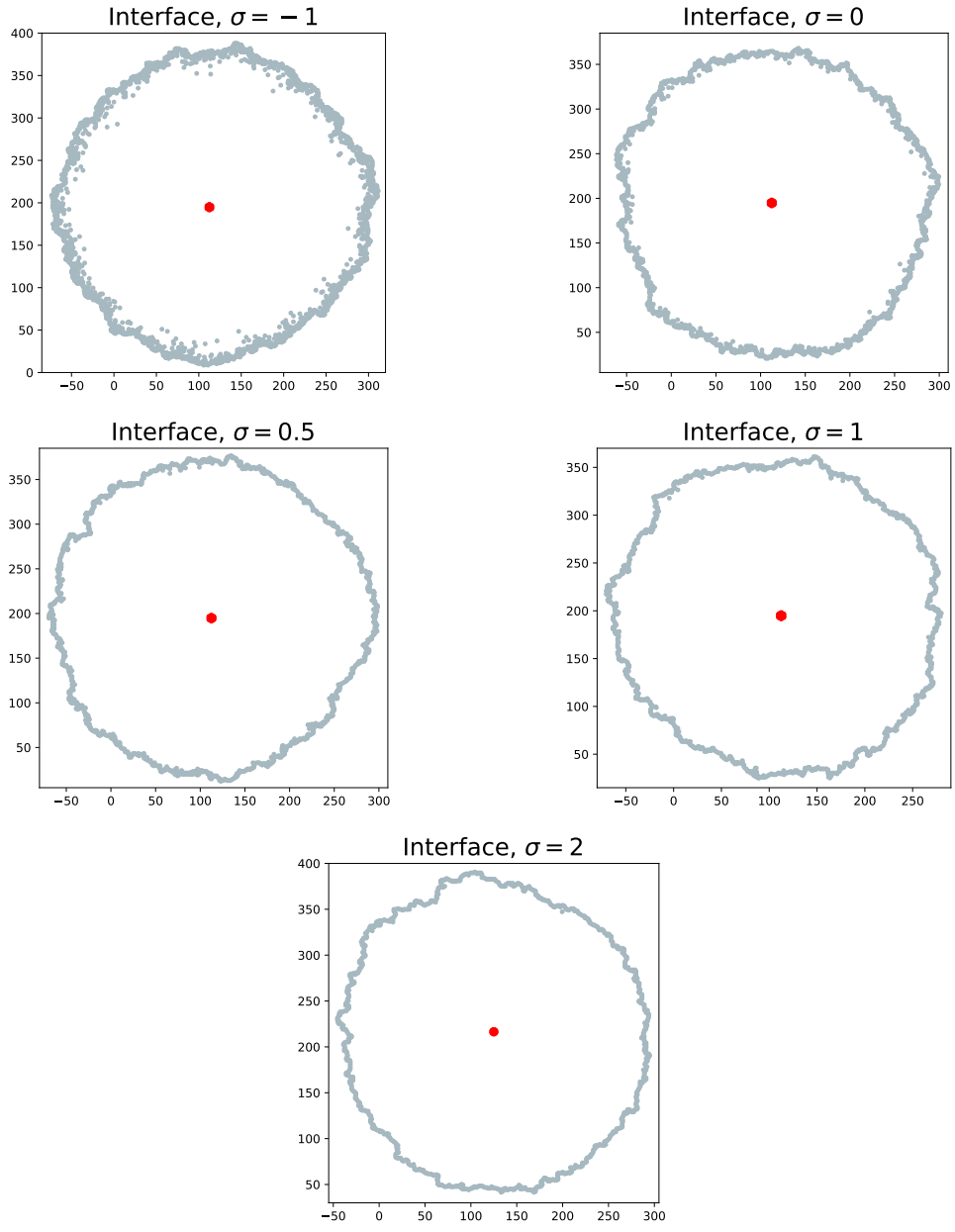


Figure 4.1: Interface of a random run for each given σ. Gray represents atoms on the boundary of the hole, and the red dot in the middle is the starting condition of the hole.

Looking at the area error over time, we can easily see that at the start the error of the hexagon is lower than that of the circle, but still within error of each other. As time passes it becomes more and more like both shapes, but the circular error drops faster. The errors cross at t_{cross}, the values of which can be found in Table 4.1

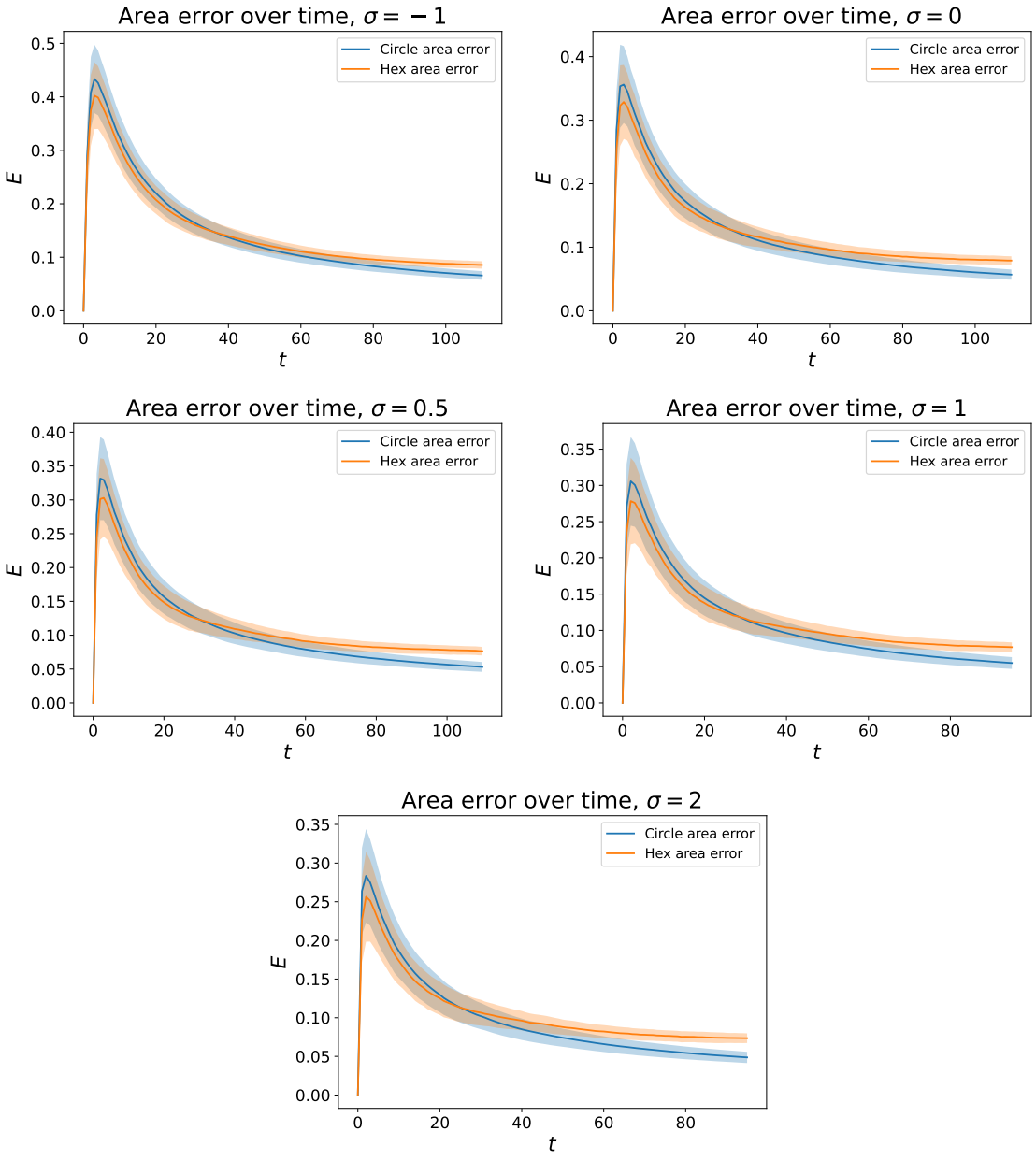


Figure 4.2: Area error over time. The line is the mean error for each ensemble and the shaded area is one standard deviation above or below the mean.

When comparing the crossover time and the etched area at the crossing time, we find that the higher scaling factors decrease t_{cross} quickly enough that the area at the crossing time decreases as σ increases. Plotting the crossover time for the different scaling factors we find a nearly perfect linear relationship, with the slope being -4 and the error of the line fit is on the order of 10^{-29}.

Table 4.1: Crossing time t_{cross} and area at time $t = t_{cross}$ for the different values of σ.

σ	-1	0	1/2	1	2
t_{cross}	36	32	30	28	24
$A(t = t_{cross})$	6770	5953	5594	5232	4512

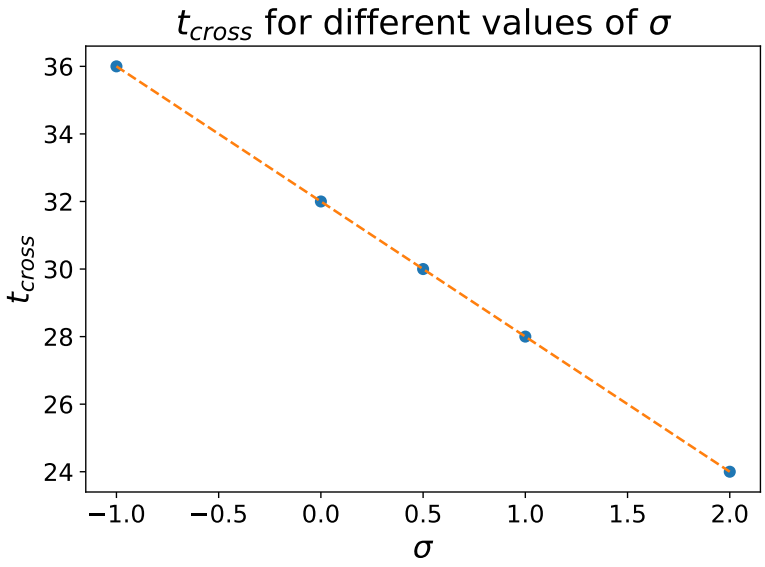


Figure 4.3: Crossover time t_{cross} for different values of σ. Blue dots are the data points and the dashed line is a linear fit.

Studying the best-fitting angle, we can see a slight positive bias over time, but the standard deviation is so large that it is essentially random. What is interesting, however, is that the mean offset angle is always slightly positive at later times.

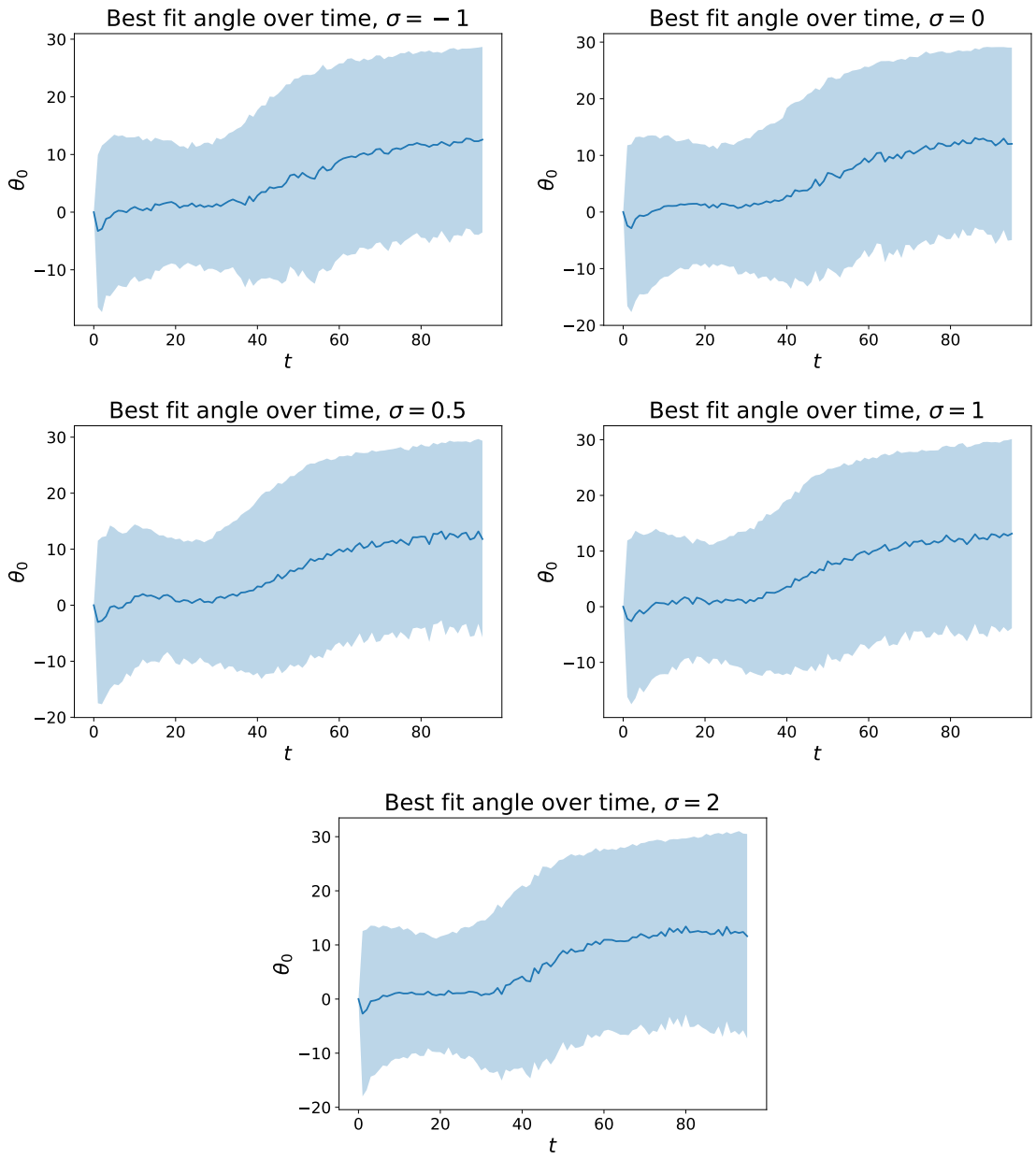


Figure 4.4: Best-fitting rotation angle for the hexagon over time. The solid line is the mean and the shaded area is within one standard deviation of the mean in each direction

For the RMSE, the error grows in a less-than-linear fashion for both errors, with the exact polynomial coefficient found in Table 4.2. Although generally the same shape, we see a larger separation of the errors as σ increases.

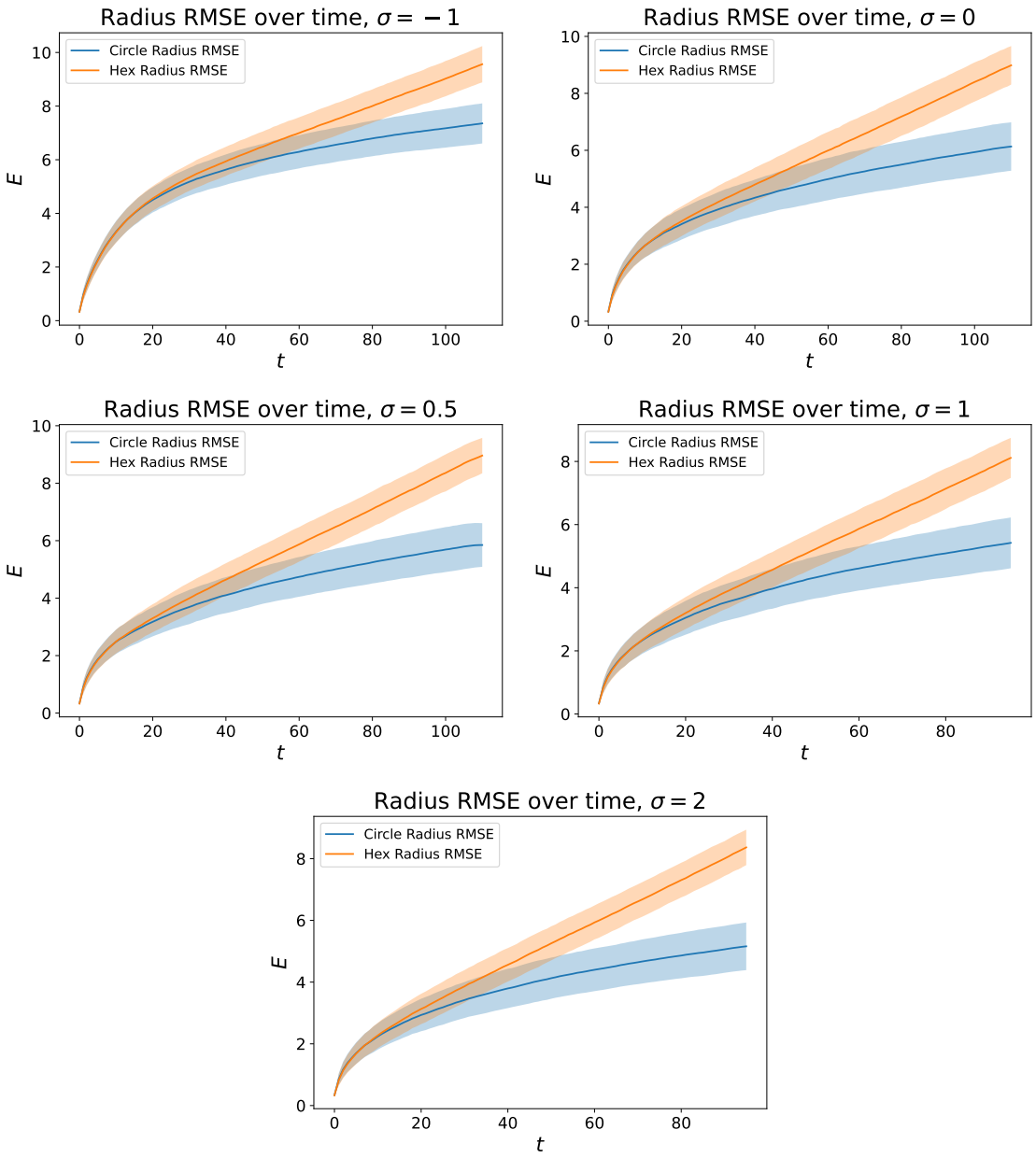


Figure 4.5: Root mean square error over time. Solid lines are mean ensemble errors, and the shaded area is within one standard deviation of the mean.

The mean absolute error's time evolution is nearly identical to the root mean square error, only slightly lower in magnitude. Similarly to Figure 4.5, both grow less than linearly over time, but the hexagonal error grows faster.

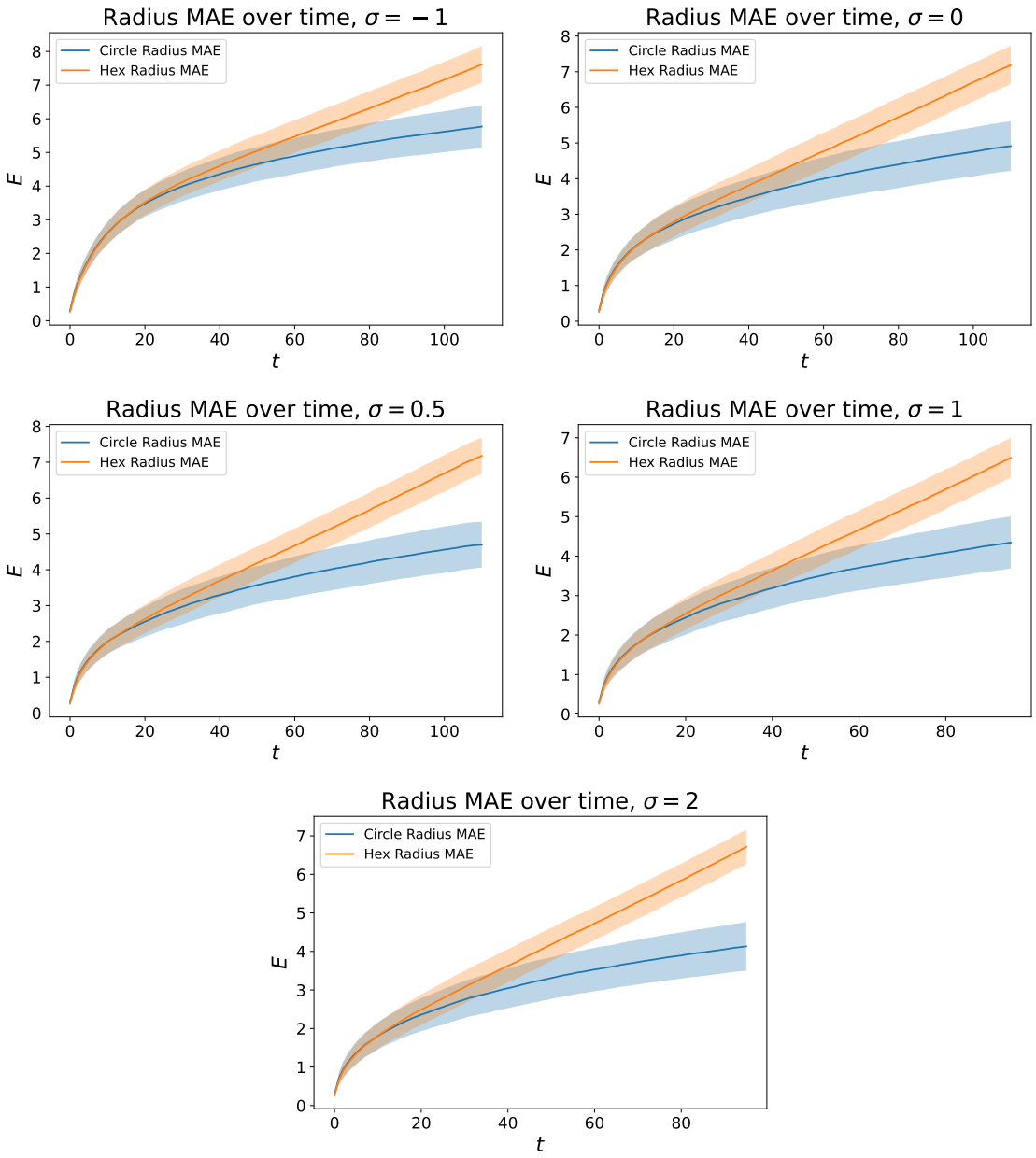


Figure 4.6: Mean absolute error over time. Solid lines are mean ensemble errors, and the shaded area is within one standard deviation of the mean.

4.2 Hexagonal starting condition

If we instead started with a hexagon rotated by 30 degrees we get the following results. Overall the results are similar in character but different in magnitude.

For the randomly chosen interface, we see that the interface for $\sigma = 0$ is rougher than it is for $\sigma = 1$, as well as more internal atoms still existing, similar to Figure 4.1

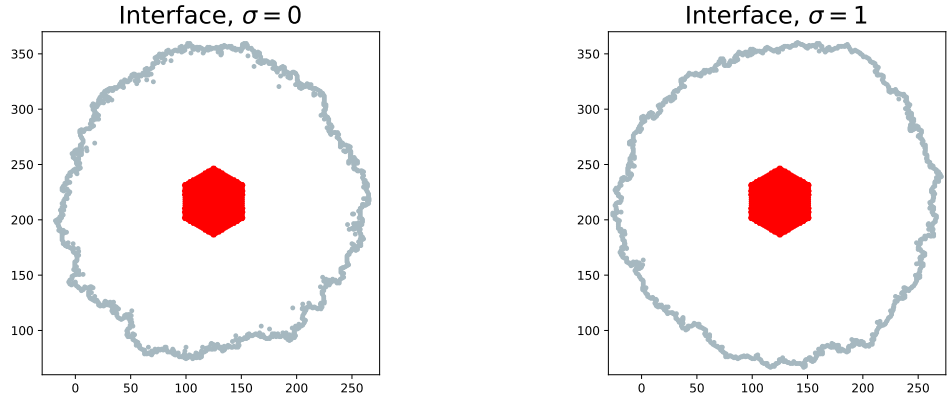


Figure 4.7: A randomly chosen run for each σ. Apart from there being fewer solitary atoms when $\sigma = 1$, the right shape is rounder just like for the small circle starting condition.

The area error now has different initial values, as the shapes' differences are larger than the lattice distance c. They never reach the same magnitude as for the small circle initial condition, most likely because a small number of atoms diverting from the ideal shapes corresponds to a lower relative error as the etched area is larger to start with.

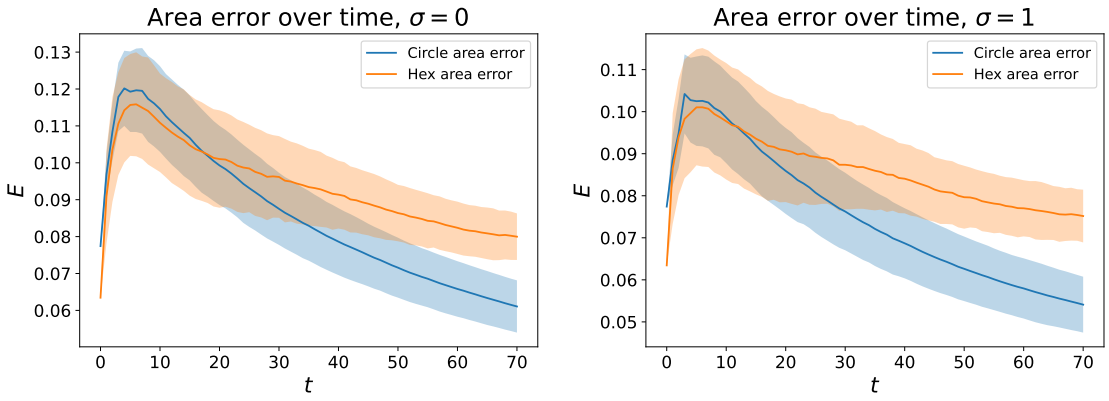


Figure 4.8: The area errors are smaller than for the small circle, but the variance is higher. There's a larger separation of the error terms for the higher σ.

In much the same way, the radial errors start at different points as the initial shape is a hexagon, however the hexagonal error quickly outpaces the circular one. Compared to the small circle initial condition it grows at roughly the same pace, with both errors being roughly equal at larger times for both initial conditions.

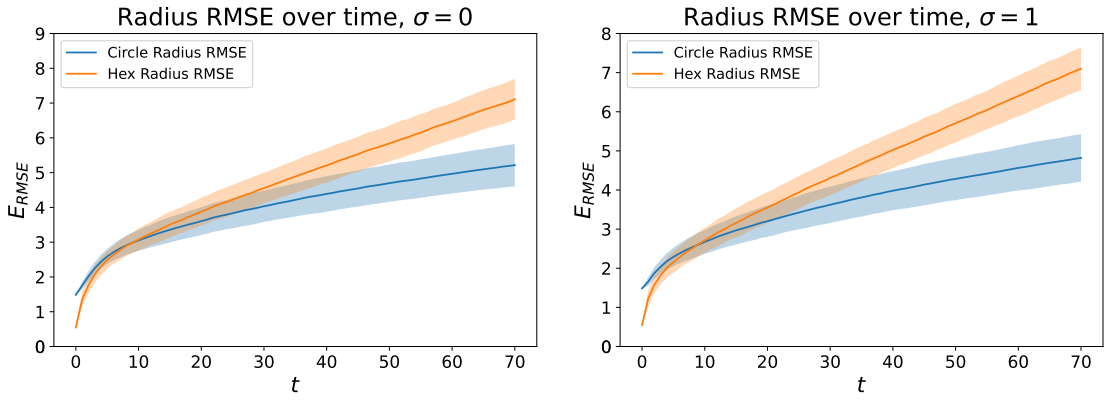


Figure 4.9: The hexagonal error starts off at 0 due to the shape of the initial condition, but quickly overtakes the circular error. Both grow less than linearly even past initial times.

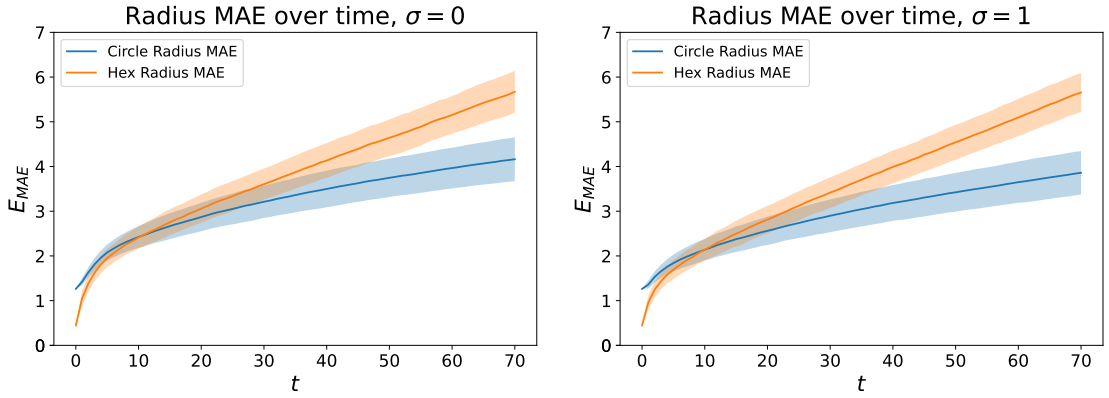


Figure 4.10: Like in Figure 4.6, the errors grow less than linearly. Similarly to Figure 4.9 the hexagonal error starts at 0 but quickly overtakes the circular error. Both grow less than linear over time.

The best-fitting angle is following the same pattern as for the circle. As it starts with a perfect 30-degree offset, the average angle is at 0 due to symmetry, and once it starts to rotate in both directions that angle is roughly kept. The variance is high here as well, just like for the small circle.

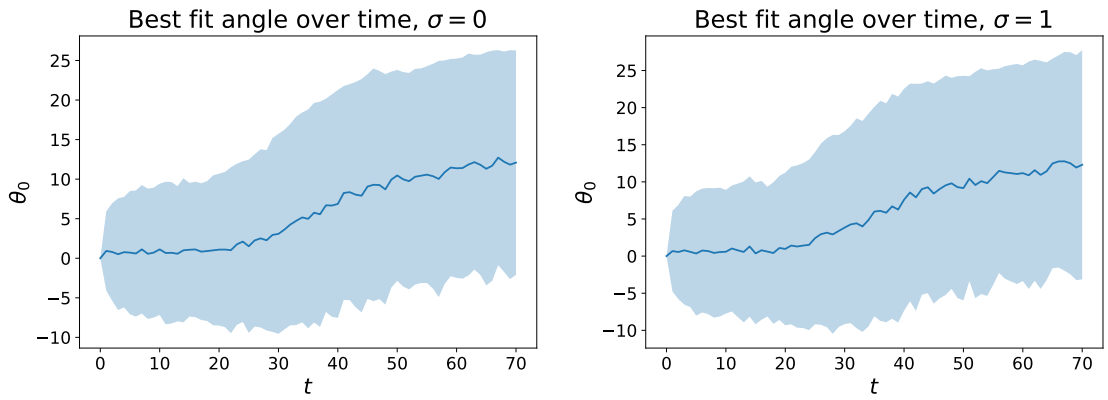


Figure 4.11: Even though the starting condition is rotated 30 degrees, the average is 0 due to rotational symmetry. Similar to the small circle initial condition, we quickly see the best-fit angle spread out.

4.3 Comparing errors by type

In the previous section we only compared the proximity of each shape against each other for each scale factor, but not the scale factors themselves. Thus, each error type has been added here and each σ has been plotted against each other.

For the area errors, the error drops as the relative reaction rate increases. This can be explained by the higher relative reaction rate favoring annihilation of more exposed atoms, which minimizes the total circumference for a given area, thus making it more circular in shape.

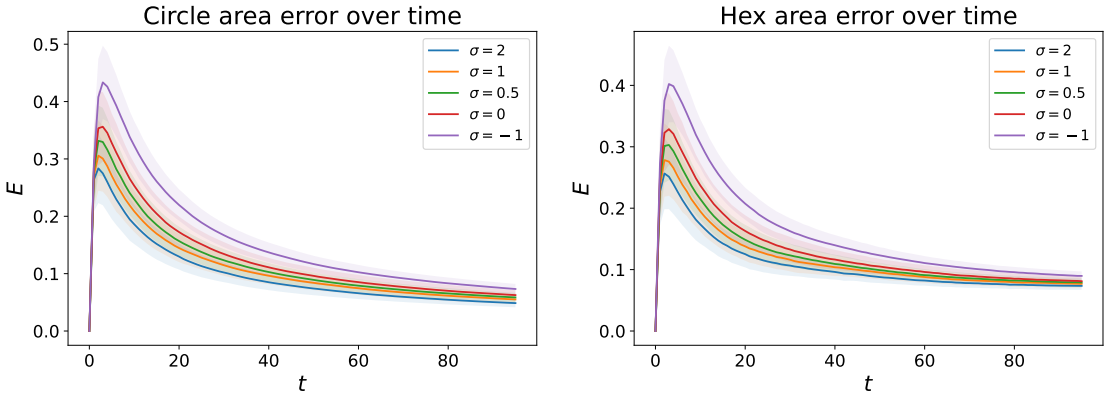


Figure 4.12: Area errors. As expected, higher values of σ results in smaller errors for both after the very initial instability.

The radial errors tell a similar story, a higher scaling factor σ leads to a slower error growth for the circle, and for all but late-time hexagonal errors the higher values of σ lead to smaller error terms.

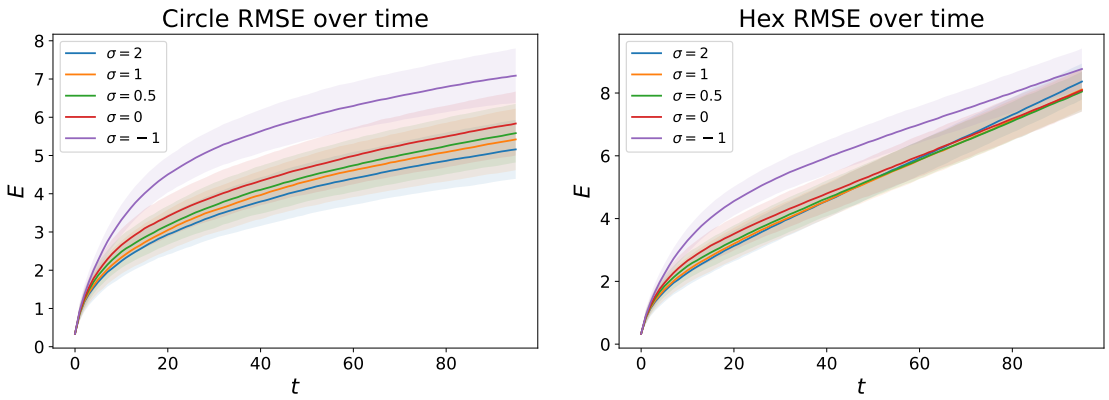


Figure 4.13: RMSE for both. In the short term, both increase quickly, before slowing down in the medium term. In the long term, the higher values of σ experience a faster increase of the hexagonal error. The same can be seen in Figure 4.14.

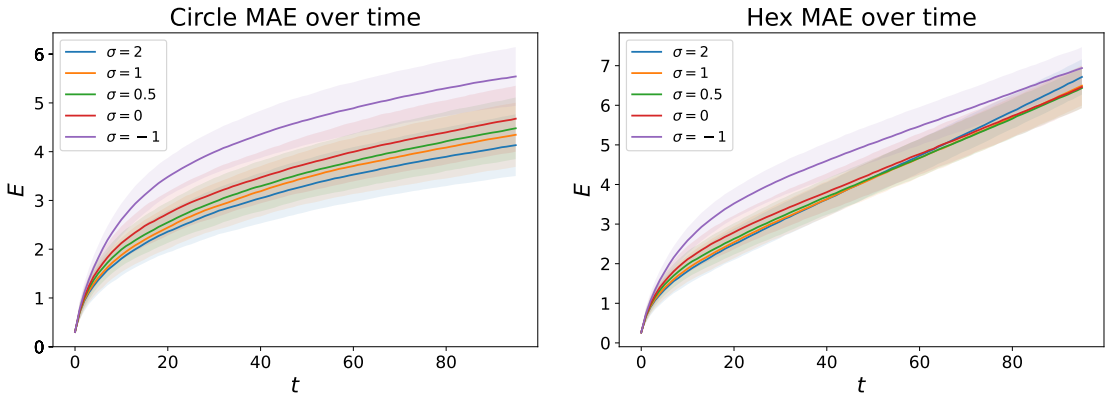


Figure 4.14: MAE for both. The same phenomenon as in Figure 4.13 can be seen that the hexagonal error starts growing faster at larger time scales for larger values of σ.

We saw that the errors were growing in a less-than-linear fashion $E_{RMSE} \sim E_{MAE} \sim t^{\beta}$, but from the plots we could not immediately see the growth factor β. Doing a log-log fit for all values of σ, the following growth factors β were obtained, shown in Table 4.2.

Table 4.2: Growth factors of the radial errors over time.

σ	-1	0	$1/2$	1	2
$\beta^{(c)}_{RMSE}$	0.2638	0.3342	0.3403	0.3603	0.3562
$\beta^{(c)}_{MAE}$	0.2777	0.3347	0.3382	0.3585	0.3534
$\beta^{(h)}_{RMSE}$	0.4619	0.6085	0.6377	0.6459	0.6764
$\beta^{(h)}_{MAE}$	0.4887	0.6151	0.6446	0.6527	0.6855

It is worth pointing out that by doing log-log slope calculations of the hex error

for $\sigma = 2$ at $t > 70$ we get a $\beta^{(h)}_{MAE} = 0.7866$ and $\beta^{(h)}_{RMSE} = 0.7713$, compared to the slope calculated at $t > 30$ (seen in Table 4.2), showing that the hexagonal error greatly increases as time goes on.

4.4 Growth

Looking at the growth patterns, we can see that the area grows roughly squared over time. The easiest way to see it is to plot the data compared to square time. This plot shows a straight line, with a higher slope for a higher σ, clearly indicating a quadratic relation between the two. We can also plot the interface size over time, showing that the rate increases linearly with respect to time, apart from early times. The fact that more exposed atoms are less likely to get annihilated for negative values of σ leads to more atoms getting exposed, resulting in a less circular shape, considering that a circle is the shape that maximizes the area for a given circumference.

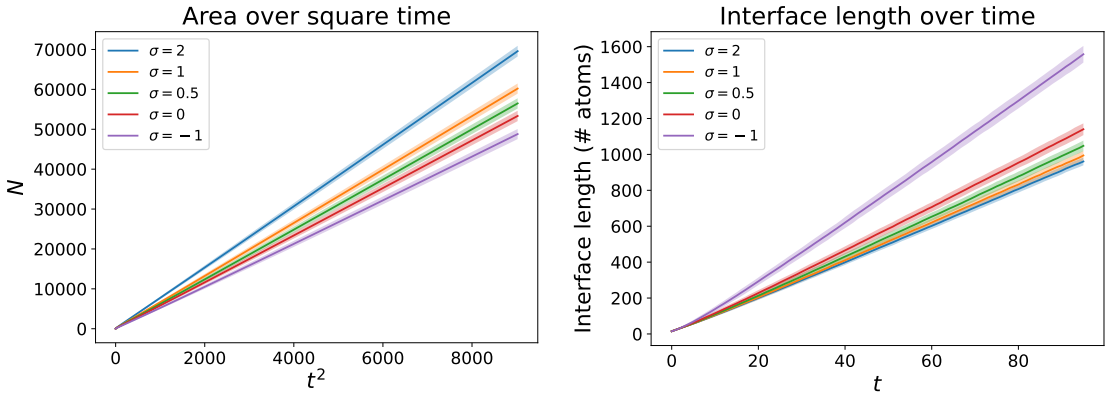


Figure 4.15: Growth with respect to time. Note that the growth rates are inverse for the area and the interface lengths for each scaling factor σ.

5

Conclusion

To summarize, this project used Monte Carlo simulations applying the Gillespie algorithm to simulate the time evolution of an etching process on a honeycomb lattice structure. In order to quantify the shapes we checked how much the area of the simulated shape differed from an ideal circle or hexagon, as well as how much the radius diverged from that shape. These analyses were made over time to study the behavior on different time scales. Different relative reaction rates and different initial conditions were used to examine their effect on the system.

What can be seen from the simulations is that the shape of the hole eventually becomes circular as more and more time passes. For a higher scaling factor σ, it happens earlier and earlier. This makes sense as solitary particles are preferentially selected for annihilation, thus leading to the atoms minimizing their number of free nearest neighbors, and by extension the circumference, too. As for the hexagonal error, it also becomes more and more similar to a hexagon up to a point where it plateaus. The higher the value of σ, the earlier it happens.

The results indicate that the lattice structure plays a significant role especially in the early times, and that the randomness of which atom decays drives the initial rise in the error terms. As such, when the lattice is presented with a larger hole as a starting condition, the relative errors don't grow as much or as quickly before starting to regress.

All of this points to a σ being very close to 0 would be the solution to the real-world scenarios found in the lab at Chalmers.

5.1 Further studies

As was seen in Figures 4.5, 4.6, 4.9, and 4.10, the model seems to act differently at late times. Since this was not allowed to play out fully, this can be further studied in the future. Moreover, the growth factor for the two-dimensional solution to the KPZ equation could also be studied, and compared to experimental results. Quickly exploring the initial condition of a non-rotated hexagon could also be of interest.

We could also look at different lattices, both those that occur in nature and those that don't, such as the simple square lattice or the Kagome lattice, to mention one from each category.

Finally, to make the reaction play out like in real life, a simulation of type $A+B \rightarrow \emptyset$ could be simulated, with refresh time to emulate the replenishing of the etchant.

References

[1] P. Walker and W. H. Tarn, *CRC handbook of metal etchants*. CRC press, 1990.

[2] P. L. Krapivsky, S. Redner, and E. Ben-Naim, *A Kinetic View of Statistical Physics*. Cambridge University Press, 2010, ISBN: 9780511780516. DOI: 10.1017/CBO9780511780516.

[3] R. Livi and P. Politi, *Nonequilibrium Statistical Physics: A Modern Perspective*. Cambridge University Press, 2017, ISBN: 9781107278974. DOI: 10.1017/9781107278974.

[4] P. Hofmann, *Solid State Physics: An Introduction*. Wiley, 2022, ISBN: 9783527-414109. [Online]. Available: https://www.wiley.com/en-se/Solid+State+Physics%3A+An+Introduction%2C+3rd+Edition-p-9783527837267.

[5] S. H. Simon, *The Oxford solid state basics*. Oxford, UK: Oxford Univ. Press, 2013, ISBN: 9780199680771.

[6] C. Nordling and J. Österman, *Physics Handbook for Science and Engineering*, 8th. Studentlitteratur, 2006, ch. Crystal Structure of Chemical Elements and Compounds, p. 138, ISBN: 9789144044538.

[7] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[8] P. Virtanen *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, Feb. 2020, ISSN: 1548-7105. DOI: 10.1038/s41592-019-0686-2. [Online]. Available: http://dx.doi.org/10.1038/s41592-019-0686-2.

[9] S. Darjani, J. Koplik, and V. Pauchard, "Extracting the equation of state of lattice gases from random sequential adsorption simulations by means of the gibbs adsorption isotherm," *Physical Review E*, vol. 96, Nov. 2017. DOI: 10.1103/PhysRevE.96.052803.

A

Example code

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve2d

dim = 100
atoms = np.ones((dim, dim))
nearest_neighbors = np.zeros_like(atoms)
end_time = 25
kernel = np.array([[0, 1, 0], [1, 0, 1], [0, 1, 0]])

def init():
    # start with 2x2 hole in center
    atoms[
        dim // 2 : dim // 2 + 2,
        dim // 2 : dim // 2 + 2,
    ] = 0

    # alternative starting condition: thin line
    # atoms[
    #     3 * dim // 10 : 7 * dim // 10,
    #     dim // 2 : dim // 2 + 2,
    # ] = 0

    # possible to do an imshow here
    # if showing starting condition is desired

    # this gives us the relative probabilities
    # of annihilation (emptiness only expands)
    nearest_neighbors[:] = convolve2d(
        1 - atoms, kernel, mode="same"
    )

def update():
    t = 0
    while t < end_time:
        # find index of atom to annihilate
        reaction_sites = nearest_neighbors * atoms
```

```
        # reaction rate is be proportional to
        # the number of available atoms
        tau = np.random.exponential(
            1 / np.sum(reaction_sites)
        )
        # Here an analysis step has been added in the full
        # code to calculate the errors
        t += tau

        # fastest way of finding the right atom to remove
        cs = np.cumsum(reaction_sites)
        x, y = np.unravel_index(
            cs.searchsorted(
                np.random.uniform() * cs[-1], "right"
            ),
            reaction_sites.shape,
        )
        atoms[x, y] = 0
        # this simulation can never reach the boundaries
        # nor should it realistically,
        # since we study the interface.
        # If it does, either extend grid
        # or lessen number of iterations
        min_x = max(0, x - 5)
        max_x = min(x + 5, dim - 1)
        min_y = max(0, y - 5)
        max_y = min(y + 5, dim - 1)
        nearest_neighbors[
            min_x + 1 : max_x - 1,
            min_y + 1 : max_y - 1
        ] = convolve2d(
                1 - atoms[min_x:max_x, min_y:max_y],
                kernel, mode="valid"
            )
    plt.imshow(atoms)
    plt.title("atoms")
    plt.figure()
    plt.imshow(atoms * nearest_neighbors > 0)
    plt.title("interface")
    plt.figure()
    plt.imshow(nearest_neighbors)
    plt.title("nearest neighbors")
    plt.show()

init()
update()
# This also counts the number of atoms removed
# as part of the initial condition
print(f"Removed atoms: {np.count_nonzero(1-atoms)}")
```

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY