TECHNISCHE UNIVERSITÄT WIEN

INSTITUTE FOR THEORETICAL PHYSICS

DIPLOMARBEIT

A framework for a priori refined adaptive meshes applied to real-time lattice field theory in 1+1D

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

UE 066 461 Masterstudium Technische Physik

eingereicht von

Kayran Schmidt, BSc BSc
Matrikelnummer 01604789

ausgeführt am Institut für Theoretische Physik
der Fakultät für Physik der Technischen Universität Wien

Betreuung:
Betreuer: Priv.Doz. Dipl.-Ing. Dr.techn. Andreas Ipp
Mitwirkung: Dipl.-Ing. Dr.techn. David Müller

Wien, 06.09.2022

(Unterschrift Verfasser) (Unterschrift Betreuer)

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst, andere als die angegebenen Quellen sowie Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Datum, Ort

Unterschrift

Statutory declaration

Hereby I declare in lieu of oath, that I wrote this thesis myself, using only literature cited in this volume. If text passages from sources are used literally, they are marked as such.

Date, Place

Signature

Abstract

A straightforward computational method for solving real-time lattice field theories is to discretize spacetime on lattices with constant spacings. The numerical accuracy can be systematically increased by shrinking the lattice spacings for the entire lattice. However, this leads to unnecessary high resource demand if the solution does not benefit from higher accuracy on the entire domain. In this thesis, a new method is introduced that is designed to work with adaptive discretizations in 1+1D. Spacetime is discretized such that it has a smaller lattice spacing only for regions where the solution requires better accuracy. The regions with different lattice spacings are combined into a connected mesh. These lattices are used by a solver to calculate the time evolution of the free, massless scalar field. At the borders between the different regions, special refinement schemes are needed to update the values for the next time step. A collection of schemes for various border types is studied. The refinement equations embrace the symmetries of the equations of motion and a conserved quantity. Extensive numerical analysis demonstrates that these refinement schemes are successful. Their error scales with the lattice spacing at a rate that matches the discretization error of the equations of motion.

Zusammenfassung

Ein einfaches Rechenverfahren zum Lösen von Echtzeit-Gitterfeldtheorien besteht darin, die Raumzeit auf einem Gitter zu diskretisieren, das einen konstanten Gitterabstand hat. Die numerische Genauigkeit kann dabei systematisch erhöht werden, indem der Gitterabstand auf dem gesamten Gitter verkleinert wird. Dies führt zu unnötig hohem Rechenaufwand, wenn die Lösung nicht von einer höheren Genauigkeit auf dem gesamten Gitter profitiert. In dieser Arbeit wird eine neue Methode vorgestellt, die mit adaptiven Diskretisierungen in 1+1D arbeitet. Die Raumzeit wird so diskretisiert, dass das Gitter nur in den Bereichen, in denen die Lösung eine höhere Genauigkeit erfordert, einen kleineren Gitterabstand aufweist. Die Bereiche mit unterschiedlichen Gitterabständen werden zu einem zusammenhängenden Gitter zusammengefügt. Diese Gitter werden von einem Solver verwendet, um die Zeitentwicklung des freien, masselosen Skalarfeldes zu berechnen. Die Grenzen zwischen den unterschiedlichen Bereichen benötigen spezielle Verfeinerungsprozesse, um die Zeitentwicklung berechnen zu können. Untersucht werden verschiedene Schemata für diverse Arten von Grenzen. Die für die Verfeinerung verwendeten Gleichungen basieren auf Symmetrien der Bewegungsgleichungen und einer Erhaltungsgröße. Umfangreiche numerische Analysen zeigen, dass die Verfeinerungsprozesse erfolgreich sind. Ihr Fehler skaliert mit dem Gitterabstand mit derselben Rate, wie auch der Fehler der diskretisierten Bewegungsgleichungen.

Acknowledgements

I want to thank my supervisors Andreas Ipp and David Müller for trusting me with this project and guiding me on the way. Many fruitful discussion sessions with them and with my colleague Markus Leuthner enriched my thoughts and allowed me to stay focused.

I also owe eternal gratitude to my family who supported me ever since and especially during the years the current thesis was in the making.

Contents

Chapter 1

Introduction

Computational methods have become an important aspect of dealing with the complex theories behind the standard model of particle physics. For example, quantum chromodynamics (QCD) describes the strong interaction force based on quarks and gluons. Experimentally, the peculiarities of QCD make it necessary to probe at very high energy scales (currently in the TeV range) to be able to trace the fundamental particles [1]. This can be done using heavy-ion collisions, where different regimes are modeled by different approximations of full QCD. In their earliest stages, called the Glasma, the 3+1D numerical methods developed by [2, 3, 4, 5, 6, 7, 8] prove to be successful. However, they are currently limited by the available computational capabilities.

A strategy to reduce the computational requirements for these kinds of numerical simulations that work with lattice discretizations of spacetime is to employ adaptive meshes. The basic idea is to work on a non-uniform lattice that changes based on certain criteria during runtime. A common procedure is so-called adaptive mesh refinement, where the resolution of the lattice is increased in areas where the current solution is no longer accurately resolved or produces undesirable error estimates. This method is especially successful for hydrodynamics, where shock fronts change the overall shape of the solution on a comparatively small scale. The earliest applications are presented in [9, 10]. Recent improvements make use of multiresolution methods [11, 12] to analyze the structure of the numerical solution before refinement is triggered.

Integrating the full adaptive mesh refinement procedure into numerical simulations is very involving. Therefore, in this thesis, a simpler approach is developed. Instead of allowing for adaptive refinement during runtime, the refinement strategy is set a priori for the entire duration of the simulation. The success of this method will depend on the type of problem being solved and requires some knowledge of the behavior of the solution in advance.

This procedure must not be confused with multigrid methods. There the idea is to work with multiple, overlapping meshes that differ in resolution. It is therefore possible to capture the dynamics on different scales and exploit numerically advantageous properties of coarser grids to correct the finer simulation lattices. The main use cases are iterative solvers [13, 14].

In a recent effort to develop specialized refinement schemes that connect the differently resolved regions of a refined lattice, a series of discretized field equations are presented in [15]. This thesis is meant as an analysis of these equations and also provides a modular implementation using Python. In chapter 2 scalar field theory in 1+1D is introduced. The tools to encode the preset mesh refinement are discussed in chapter 3. Then, in chapter 4 the design of a solver that works on these lattices is presented and all components are combined into a framework. The refinement equations from [15] are revisited in chapter 5 and extended to cover more lattice configurations. Important properties of the resulting refinement schemes are identified in chapter 6. The concluding discussion of numerical results on a variety of adaptive meshes is given in chapter 7.

Chapter 2

Scalar field theory on the lattice

The field theory for a massless, real scalar field $\phi(x^\mu)$ in 1+1D is given by the action

$$S[\phi] = \frac{1}{2}\int_{\mathcal{M}} \mathrm{d}x^0 \mathrm{d}x^1 \sum_\mu \partial_\mu \phi(x^\mu)\partial^\mu \phi(x^\mu). \tag{2.1}$$

x^μ denotes a point in the two-dimensional Minkowski space $\mathcal{M}$ where the index $\mu \in \{0, 1\}$ and $x^0 \equiv t$, $x^1 \equiv x$. By setting the speed of light $c = 1$ the partial derivatives are defined as $\partial_\mu = \partial/\partial x^\mu$. The time evolution of ϕ is determined by the equations of motion (EOM)

$$\partial_0^2 \phi(x^\mu) - \partial_1^2 \phi(x^\mu) = 0, \tag{2.2}$$

which are derived by demanding that the variation of the action w.r.t. the field ϕ vanishes. We obtain the dispersion relation via Fourier analysis. Inserting the plane-wave ansatz $\phi(x^\mu) = \phi^0 \exp(ikx - i\omega t)$ with the constant ϕ^0 into eq. (2.2) and solving for k and ω yields

$$\omega^2 = k^2. \tag{2.3}$$

2.1 Discretized equations of motion

A common scheme for solving the time evolution of eq. (2.2) numerically is to apply a finite difference method [16, 17, 18]. First, we need to discretize all the objects and operators. The Minkowski space $\mathcal{M}$ is mapped to a two-dimensional lattice

$$\mathcal{M} \mapsto \Lambda^2 = \left\{ x \,|\, x = \sum_\mu n_\mu \hat{a}^\mu, \quad n_\mu \in \mathbb{Z} \right\}, \tag{2.4}$$

where $\hat{a}^\mu = a^\mu \hat{e}_\mu$ (no sum implied) are the lattice spacings along the the unit vectors $\hat{e}_\mu$. Similarly, the scalar field ϕ is only defined at the lattice sites x and is interpreted as the value of the continuum field at x

$$\phi_x \equiv \phi(x^\mu), \qquad \phi_{x+\mu} \equiv \phi(x^\mu + \hat{a}^\mu), \tag{2.5}$$

using the shorthand notation that moves the lattice site x^μ to a subscript. We approximate the partial derivatives by the forward and backward finite differences identified on the lattice as

$$\partial_\mu^F \phi_x \equiv \frac{\phi_{x+\mu} - \phi_x}{a^\mu}, \qquad \partial_\mu^B \phi_x \equiv \frac{\phi_x - \phi_{x-\mu}}{a^\mu}. \tag{2.6}$$

Equations (2.6) are accurate up to $\mathcal{O}(a)$ [19, 20]. We approximate second order derivatives up to $\mathcal{O}(a^2)$ [19, 20] by combinations of eqs. (2.6) as

$$\partial_\mu^2 \phi_x \equiv \partial_\mu^F \partial_\mu^B \phi_x = \frac{1}{(a^\mu)^2}(\phi_{x+\mu} + \phi_{x-\mu} - 2\phi_x). \tag{2.7}$$

We can obtain the discretized EOM for the theory defined in eq. (2.1) by either replacing the operators in eq. (2.2) directly using eq. (2.7) [16, 17, 18] or starting from a discretized action

$$S[\phi] = \frac{1}{2} \prod_{\mu} a^{\mu} \sum_{x}^{\Lambda^2} \left((\partial_0^F \phi_x)^2 - (\partial_1^F \phi_x)^2 \right) \tag{2.8}$$

and deriving the EOM based on a discrete variational principle [15, 5]. The second approach leads to "variational integrators" [21, 22, 23] that can have superior properties in terms of long-term stability and error propagation. Notably, if the discrete action has the same symmetries as its continuum counterpart, a discrete Noether's theorem can be applied to obtain conserved quantities for the given theory.

For the theory defined by eq. (2.1), both methods result in

$$\partial_0^2 \phi_x - \partial_1^2 \phi_x = 0. \tag{2.9}$$

We can algebraically solve these discrete EOM for the value of ϕ_{x+0} and get

$$\phi_{x+0} = \left(\frac{a^0}{a^1} \right)^2 (\phi_{x+1} + \phi_{x-1} - 2\phi_x) - \phi_{x-0} + 2\phi_x. \tag{2.10}$$

The resulting update equation requires the values ϕ_x and ϕ_{x-0} of the previous two time steps and is accurate up to $\mathcal{O}(a^2)$ [16, 17, 18].

2.2 Numerical dispersion and stability

Again, we use Fourier analysis to determine the numerical dispersion. Inserting the same plane wave ansatz as before into eq. (2.9) leads to the numerical dispersion relation

$$\sin^2 \left(\frac{\omega a^0}{2} \right) = \left(\frac{a^0}{a^1} \right)^2 \sin^2 \left(\frac{k a^1}{2} \right). \tag{2.11}$$

The stability of the numerical scheme eq. (2.10) is determined by the Courant number $C = a^0/a^1$. If $C > 1$, eq. (2.11) can only be fulfilled by complex valued frequencies ω that lead to exponentially amplified modes for the discrete solution ϕ_x. Hence, the numerical scheme of eq. (2.10) will not be stable.

The Courant number is also linked to the Courant-Friedrichs-Lewy (CFL) condition [24] which states that the numerical domain of dependence[1] has to contain the true domain of dependence of the corresponding partial difference equation. This is a necessary (but not sufficient) condition for the discrete solution to converge to the exact analytical solution in the limit $a^{\mu} \to 0$. In the case of eq. (2.10), the CFL condition simplifies to $C \leq 1$ and is equivalent to the stability condition for real-valued frequencies. In the special case that $a^0 = a^1$, C will be 1 and the numerical dispersion eq. (2.11) will reduce to the true non-dispersive eq. (2.3). However, this is only the case for the free scalar field in 1+1D. More elaborate stability and convergence analysis is done in [16, 17, 18].

[1] The domain of dependence is the part of $\mathcal{M}$ that influences the solution $\phi(x^{\mu})$ at x^{μ}.

Chapter 3

A priori refined collision meshes

Building a solver for the theory defined in the previous chapter is straightforward (see for example [17]). The so-called stencil defined in eq. (2.10) is applied once for each value of the next time slice ϕ_{x+0} that needs to be calculated. A time slice is the collection of all ϕ_x with the same temporal coordinate and represents the snapshot of the time evolution at that time. The values at the spatial boundaries need special treatment because there the stencil would refer to values outside of the domain. In the following, we adopt the simple Dirichlet boundary conditions where the boundary values will be fixed to 0. Other common boundary conditions, e.g. periodic boundaries, require more involved treatment and are discussed for example in [17, 18]. The result of a simulation based on this solver is depicted in fig. 3.1. With the choice of $a^0 = a^1 = a$, the solver becomes non-dispersive and the shape of the Gaussian pulse is perfectly preserved.

When we inspect fig. 3.1, it becomes clear that for every time slice about half of the values are just 0. Still, these 0 are recomputed at every time step and occupy memory because the lattice is homogeneous. To reduce this unnecessary computational and memory load, the idea is to coarsen the lattice in regions where little to no change is happening so that these regions are covered by fewer, bigger cells. This leads to "adaptive meshes", which are non-homogeneous lattices that adapt to the calculated solution. The mesh is resolved higher (= finer lattice spacing) where the values are changing with a larger gradient and resolved lower (= coarser lattice spacing) elsewhere. Of course, the shape of such an adaptive mesh depends on the solution of the discretized problem.

The proper design of an adaptive mesh requires knowledge about how the solution is expected to behave during its time evolution. Fortunately, for the theory introduced in chapter 2, this is not hard to achieve. We can write the plane wave ansatz for $\phi(x^\mu)$ as

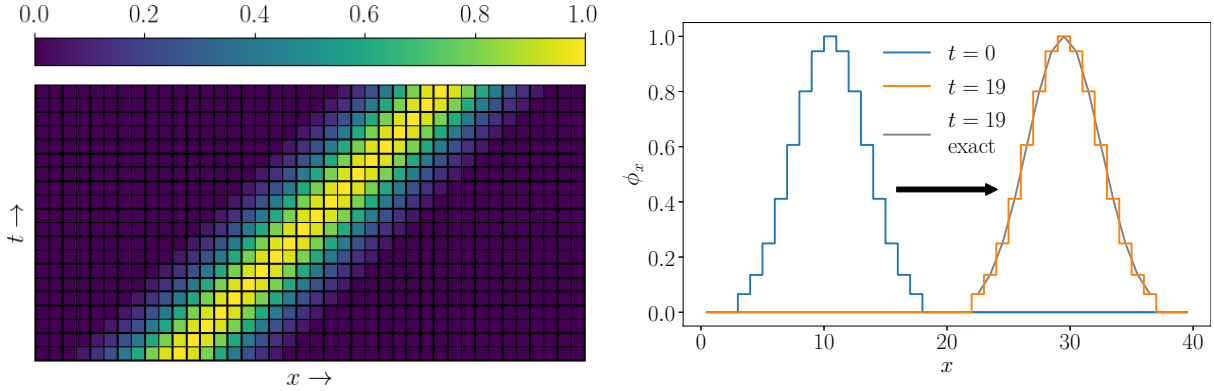
$$\phi(x^\mu) = \phi^0 \exp(ikx - i\omega t) = \phi^0 \exp\left[ik(x - vt)\right] = f(x - vt), \tag{3.1}$$

with $v = \omega/k = 1$ if $c = 1$. As long as $f(x-vt)$ can be differentiated twice, it is a valid solution of the EOM eq. (2.2) regardless of its shape. We can introduce the co-moving coordinate $\xi = x-vt$ so that the solution $f(x^\mu) \equiv f(\xi)$ only depends on one scalar parameter. Therefore, along a certain trajectory in Minkowski space that is given by the characteristic

$$\xi(x, t) = x - vt = \text{const.}, \tag{3.2}$$

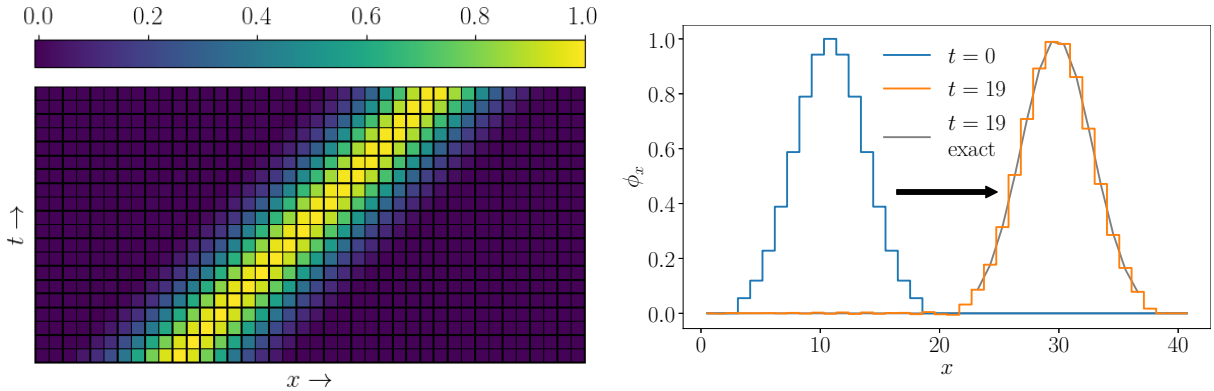
the solution f retains its value. This is the reason why in fig. 3.1 the pulse travels in a straight line with slope $v = 1$ and its shape is preserved.

The full solution spectrum of the EOM eq. (2.2) is given by $f(x \pm vt)$. The sign determines the direction in which the information is propagated. As before, for $-$ the value of ξ stays constant for growing t and x (i.e. right moving). For $+$ the slope is -1 and ξ stays constant for growing t



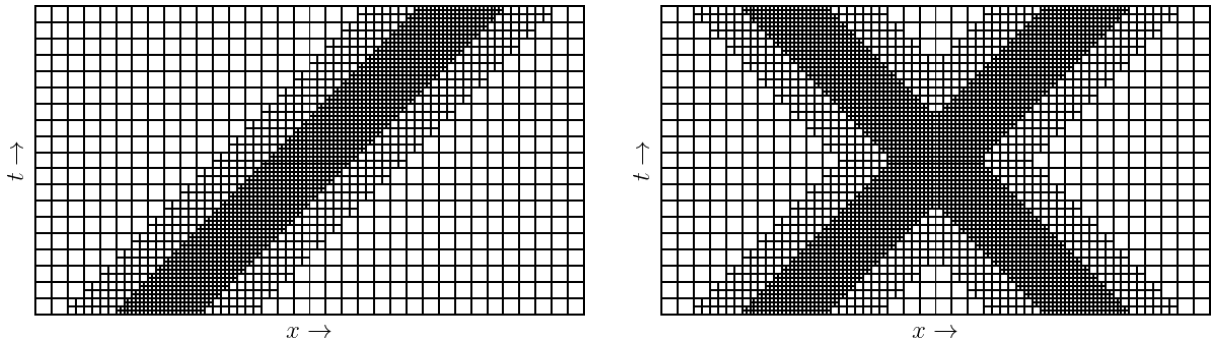
(a) Lattice representation. (b) Time slice plots.

Figure 3.1: Time evolution results for a solver based on eq. (2.10) with $a^0 = a^1$. 19 update steps are calculated. The profile is sampled from a Gaussian centered at $\bar{x} = 10$ with standard deviation $\sigma = 3$ and cutoff at 0.05 below which all values are set to 0.



(a) Lattice representation. (b) Time slice plots.

Figure 3.2: The same solver from fig. 3.1 is configured with the Courant number $C = a^0/a^1 = 32/33 = 0.\overline{96}$. The pulse profile is sampled from a Gaussian with $\bar{x} = \frac{10}{C} + \frac{1}{2}(\frac{1}{C} - 1) \approx 10.33$ and $\sigma = 3$ and cutoff at 0.05. (b): Dispersive effects modify the profile at $t = 19$ (orange).



(a) (b)

Figure 3.3: Mesh layouts for a single pulse (a) and double, colliding pulses (b).

and shrinking x (i.e. left moving). The example of fig. 3.1 employs initial conditions that single out the right moving characteristic. The values for the first two time slices are given in advance and are sampled from a Gaussian. Pulse profiles that are localized and fall off sufficiently fast can be placed on the simulation domain with enough distance to the spatial boundaries. This is important to avoid the influence of the boundary conditions that typically lead to reflections. Sampling the initial conditions from the previous plane wave ansatz instead of the Gaussian would fill the entire x-range of the initial time slices with values so that interactions with the boundary conditions become unavoidable.

The next example in fig. 3.2 demonstrates how the exact shape of the profile is not conserved when $C < 1$. The profile is sampled from the same Gaussian as before with slightly shifted $\bar{x}$. Additionally, some ripples are visible in the region that lies behind the pulse. The resulting profile deceptively seems to approximate the exact, undisturbed Gaussian when sampled at different positions relative to its peak. This is the result of the characteristics no longer coinciding with the lattice diagonals because the cells are no longer squares. Therefore, along the lattice diagonals, the value of the solution changes in contrast to the case with $C = 1$. In fig. 3.2a the cells are falsely depicted as squares so that the right moving characteristic appears with a steeper slope than $+1$. If the Courant number is chosen even smaller, the ripple amplifies and dispersive effects, like broadening, become more obvious.

Based on these preliminary examinations of the problems in concern, it is possible to design appropriate adaptive meshes. The most obvious setup is to put the finest mesh ("mesh kernel") along the expected propagation trajectory of an initial pulse and coarsen the remaining parts of the rectangular domain. Multiple coarsening steps allow to reduce the number of cells even more. The initial pulse can be contained by the mesh kernel in its entirety, or it can spill over to the neighboring coarser regions where less resolution is required. Such a lattice is given in fig. 3.3a. For a setting with two colliding pulses, the crossing of two opposing characteristics cannot be captured by this single "pulse shaped" mesh. Instead, extending the mesh kernel to cover both characteristics in a cross or "collision-shaped" mesh seems appropriate. Figure 3.3b illustrates this kind of lattice. The triangular regions in fig. 3.3b that are enclosed by two opposing characteristics starting at the same spacetime coordinate in the center of the domain are known as the light cones of the collision. When two pulses are placed at the kernels on both sides of the lattice each, the area of the forward light cone will capture their causally connected interaction. Until the pulses get close enough to influence each other in the forward light cone, they will propagate along their respective characteristics. Hence, the first half of the time evolution that is possible on the lattice given in fig. 3.3b is perfectly adequate. Only the forward light cone after the collision could be better shaped to capture the effects of the interaction on higher resolved regions. These kinds of optimizations are not covered in the work at hand but provide good opportunities to optimize the results of the presented adaptive mesh procedures. Further discussion of interacting pulses in this setting can be found in [15].

3.1 Refinement border types

Figure 3.4 shows again a collision-shaped lattice with only two different lattice resolutions. The mesh kernel shares eight different types of borders with the next coarser lattice cells. For their identification, we apply a multiword naming scheme. All identifiers together give the template as "Type-Resolution-To-Resolution-Direction". The type can be one of "Diagonal", "Straight" or "Cusp". For a border to be of type "Straight", at least four cells of the fine side (or two cells of the coarse side) need to stack up at the same x-position in the temporal direction. Otherwise, the resulting border is just a transition between two "Diagonal" borders. The following list contains all border types and their acronyms that are present on collision lattices.

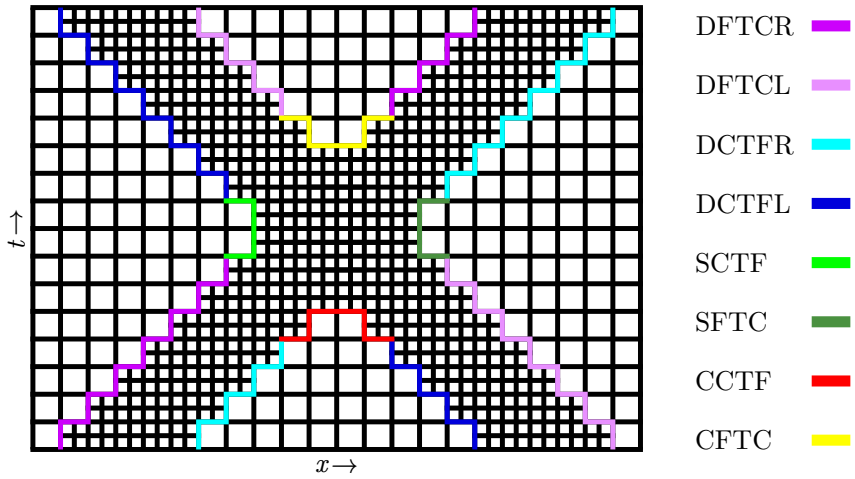


Figure 3.4: All border types and their acronyms for collision lattices.

- DCTFR: Diagonal Coarse To Fine Right
- DFTCL: Diagonal Fine To Coarse Left
- DFTCR: Diagonal Fine To Coarse Right
- DCTFL: Diagonal Coarse To Fine Left
- SCTF: Straight Coarse To Fine
- SFTC: Straight Fine To Coarse
- CCTF: Cusp Coarse To Fine
- CFTC: Cusp Fine To Coarse

3.2 Encoding the lattice configuration

3.2.1 The lattice array

The next task is to develop an efficient scheme for encoding the "shape" of the adaptive meshes. Saving attributes like position, size and neighbors for each cell in a data container leads to a lot of unnecessarily duplicated information. Instead, a single structured data element can represent simply connected regions consisting of cells with a particular size. Consider the first time slice of the previous fig. 3.4 for example. There are five different simply connected regions, two belonging to the mesh kernel and three containing the coarser cells. To encode this time slice, five data elements will be sufficient instead of one for each cell. Each of these elements is structured in the sense that it contains more than one primitive element (i.e. numbers). This concrete example gives rise to the following data array: `[(2, 1), (1, 10), (2, 10), (1, 10), (2, 1)]`. We build the structured data elements out of two numbers. The first number stands for the resolution of the cells in units of the resolution of the mesh kernel. Therefore, the mesh kernel always has `(1, N)` and because the coarser regions in fig. 3.4 are twice as large they have `(2, N)`. Specifying only one number for the resolution is sufficient because the Courant number defines the ratio of temporal to spatial resolution globally on the entire lattice. The last number gives the count `N` of the cells with the specified resolution in the spatial direction. Finally, each time slice gets its own data array. As mentioned earlier, there is a time slice for each time step. But regions that are coarser than the mesh kernel can only change after as many time steps as their temporal resolution dictates. For each of these time steps, these regions' data will be duplicated. The reason not

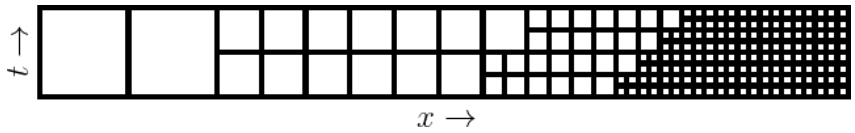


Figure 3.5: Eight time slices of a basic diagonal borders lattice with resolutions 8, 4, 2, and 1.

to also include the temporal extent in the structured data and eliminate this duplication is to keep the time slices independent from each other. This way, all border types can be naturally deduced from neighbor relations within the lattice array by comparing subsequent time slices. Putting all together, the first six time slices of fig. 3.4 become:

```
1: [(2, 1), (1, 10), (2, 10), (1, 10), (2, 1)]
2: [(2, 1), (1, 10), (2, 10), (1, 10), (2, 1)]
3: [(2, 2), (1, 10), (2, 8), (1, 10), (2, 2)]
4: [(2, 2), (1, 10), (2, 8), (1, 10), (2, 2)]
5: [(2, 3), (1, 10), (2, 6), (1, 10), (2, 3)]
6: [(2, 3), (1, 10), (2, 6), (1, 10), (2, 3)]
```

3.2.2 The *mrlattice* package

Having decided on the lattice encoding, the need for tools that allow the easy creation of these lattice arrays comes up. Therefore, the *mrlattice*[2] (multi-resolution lattice) Python package is being developed. The goal is to reduce complex meshes, like collision shapes, to as few tunable parameters as possible.

At first, the only rule that governs the creation of the lattices is the coarsening factor. The previous examples already respected that neighboring regions must only differ by one factor of 2 in both temporal and spatial direction. This splits a coarser cell into four cells of the next finer resolution. Picking a global factor makes sure that every type of border is self-similar regardless of the actual resolutions of the cells. The benefit is that the same refinement algorithms will be applicable. Additionally, one parameter is already fixed and simplifies the mesh generation process. Instead of 2, any other integer factor would be possible too but would produce more complicated coarsening schemes.

A good way to start is to identify the fundamental building blocks of the complex lattice configurations and build up the full lattices step by step. The most basic configuration is a homogeneous lattice and because it has no borders, it will not be of concern anymore. The first, real building blocks are diagonal borders. In fact, diagonal borders are all that is required to build the pulse-shaped lattice of fig. 3.3a. Figure 3.5 is one example of a complete lattice with diagonal borders and four different resolutions. Each region that does not share a domain boundary has one border with a coarser region and one with a finer region. Towards the finer region, after as many time steps as are required for a full cell (= temporal resolution), the region is extended by one cell in the spatial direction. However, toward the coarser region, only after twice as many time steps the spatial size of the region is corrected and this time by two cells. For one region that extends over multiple time slices (i.e. the region with resolution 2 in fig. 3.5), the continuation in the temporal direction results in alternating slices with a spatial size difference of one cell. We can identify this pattern best at the border between the regions with resolutions 2 and 1 on the right of fig. 3.5.

[2]The source code is hosted on https://gitlab.com/openpixi/mrlattice (visited 09/2022) and can be installed as a PYPI package.

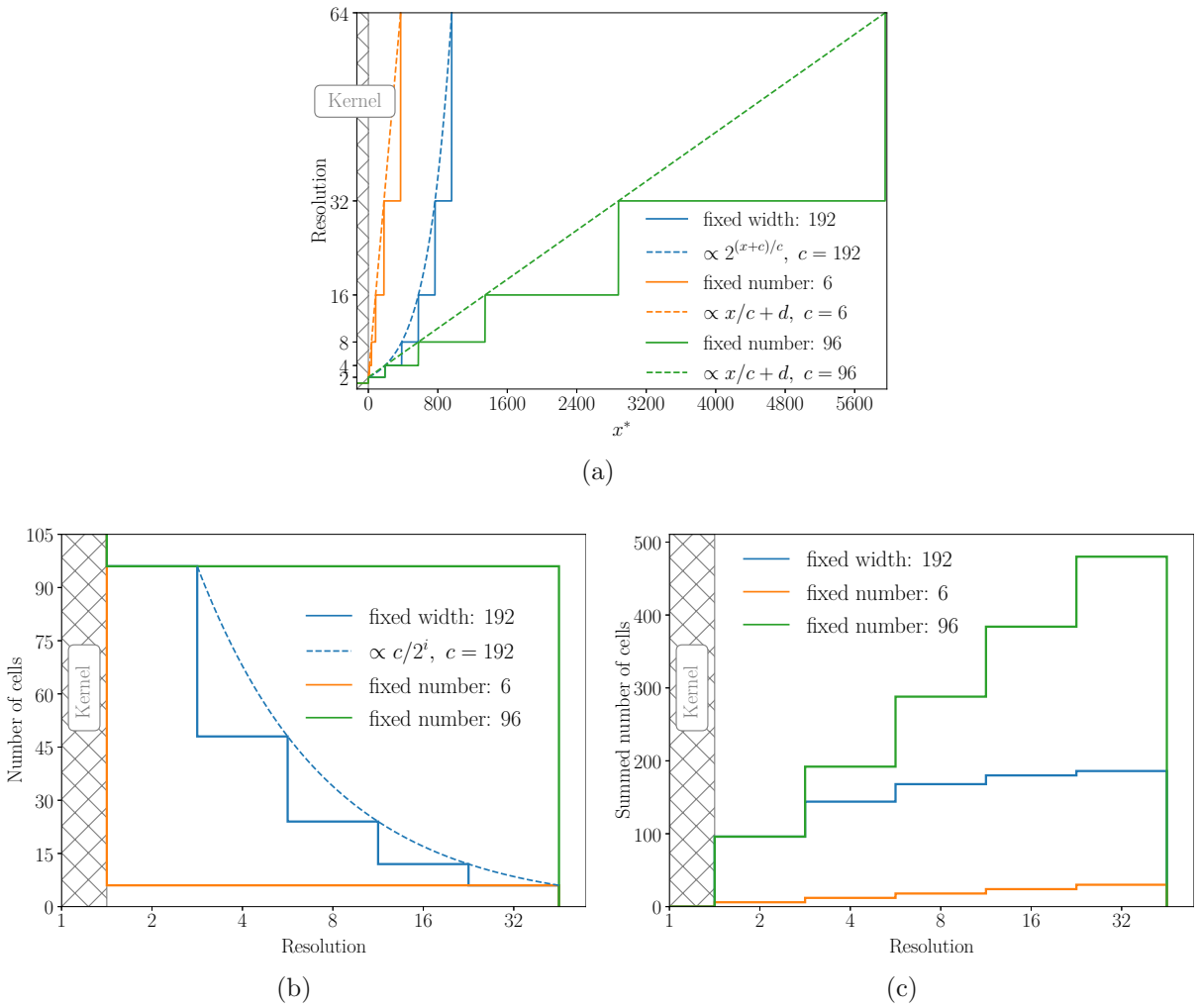


Figure 3.6: Comparison of the fixed number and fixed width coarsening schemes for the width parameters given in the legends. (a) shows the increase in resolution over the spatial distance to the mesh kernel x^*. (b) and (c) plot the number and cumulative sum of the number of cells per resolution 2^i.

A lattice segment with diagonal borders is characterized by how many coarsening steps it contains and by the spatial widths of each region. For simplicity, any given resolution shall only appear in one, simply connected region per time slice. The result is a monotonous increase in resolution for growing x. In order to specify the spatial widths, another simplification is appropriate: apart from the first and last region, we can use a simple rule to determine all the enclosed coarsening regions' widths. Excluding the first and last regions and awarding them with more flexibility comes in handy when considering that these regions play an important role when joining lattice segments together. We consider two schemes in the following:

Fixed number scheme. This approach sets a fixed number of cells for each coarsening region.

Fixed width scheme. This approach sets a fixed spatial width for each coarsening region. Because only integer numbers are possible for the cell count, the regions will only contain as many cells as will fit into the specified width without overstepping the limit. Thus, it can happen that the actual width of the coarsening regions on the resulting lattice is different from the set width if that value is not an integer multiple of all involved resolutions.

A comparison of the properties of both of these schemes is translated to plots in fig. 3.6. The values represent one time slice of the given configurations where the mesh kernel (grey hatching) and the coarsest region are not included. The values of the tunable parameter (= width) are given in the legend. They are picked such that the remaining finest and coarsest regions of the fixed width scheme share the same size as either one of the fixed number schemes. We can see this in fig. 3.6b, where the blue lines interpolate from the green to the orange lines. Setting a fixed width of 192 means that for the first region after the mesh kernel 96 cells of resolution 2 will fit. Into the same space, six cells with resolution $2^5 = 32$ will also fit.

Figure 3.6a compares the resolutions of the cells for each scheme at a distance x^* from the mesh kernel. The derivations of the formulas that are given in the legend are found in the documentation of *mrlattice*. The fixed width scheme starts with a flat slope like the fixed number scheme with 96 cells per region (green) and quickly steepens to match the fixed number scheme with 8 cells per region (orange). Together with the cumulative sum of cells per region in fig. 3.6c, the interpretation is that the fixed width scheme allows having regions with a large number of cells next to the mesh kernel while still converging within a short distance to the lowest resolution. This allows for a high-resolution lattice next to the mesh kernel for a longer distance. With either fixed number scheme, either the coarser regions have more cells than necessary, or the regions next to the kernel have fewer cells than necessary. Figure 3.6c also directly translates to the memory requirement of simulations on the given lattices and underlines the efficiency of the fixed width scheme. However, if for a particular simulation problem the mesh kernel completely covers the relevant spacetime region, picking a fixed number scheme with a low number of cells per region will be the most efficient. In cases where this would lead to unreasonably large mesh kernels, the fixed width scheme is an adequate solution.

Lattice modifiers

With the building block for diagonal borders prepared, the pulse-shaped mesh of fig. 3.3a can be assembled by making use of four lattice modifiers:

- Temporal mirror: All time slices of the lattice are reordered in reverse.
- Spatial mirror: All time slices' regions are reordered in reverse.
- Temporal extension: Two lattices are joined in temporal direction.
- Spatial extension: Two lattices are joined in spatial direction.

Obviously, joining two lattices in one direction requires that they are of the exact same size in the other direction. The segment with diagonal borders of fig. 3.5 can be transformed into a pulse segment by spatially extending it with its spatially and temporally mirrored copy. The result is a lattice with the mesh kernel as the region in the center and the coarsest regions at the domain boundaries. We refer to these outermost regions as the "buffer".

All regions up to the buffer already exhibit the desired borders, but the pulse segment still needs to be propagated in the temporal direction to extend the lattice. This can be accomplished by taking a copy of the pulse segment, shifting it by the width of one buffer cell (to the right in this concrete example) and appending it in the temporal direction. The diagonal borders will continue correctly. The empty space missing for a complete rectangular lattice is then filled with buffer cells, hence the name "buffer". By repeating this process we obtain a pulse lattice like fig. 3.3a. The direction of the diagonal borders can be flipped when mirroring the entire lattice in temporal or spatial direction.

Row #	Row # (binary)	Cell # per region
7	111	[7, 7, 7]
6	110	[7, 7, 6]
5	101	[7, 6, 7]
4	100	[7, 6, 6]
3	011	[6, 7, 7]
2	010	[6, 7, 6]
1	001	[6, 6, 7]
0	000	[6, 6, 6]

Table 3.1: Binary encoding for cell numbers.

In order to get to the collision lattices (fig. 3.3b), an additional modifier is necessary. The idea is to overlay two temporally mirrored pulse lattices with diagonal borders in opposite directions and let the higher-resolved, finer cells prevail over lower-resolution, coarser cells. The logic that is involved in implementing this modifier on the level of lattice arrays is quite extensive. Its description is also included in the documentation of *mrlattice*. Note that already the process of superimposing two building blocks with diagonal borders in this way leads to cusp-type borders between the different regions. These lattices only contain one cusp for each resolution. A sister modifier where the coarser cells prevail is also conceivable, but this effect is not needed.

Implementation details

In this section, we discuss peculiarities regarding the implementation of the previous procedures. The first challenge is specifying the building blocks for diagonal borders. As already mentioned, for a region that extends over multiple time slices an alternating pattern for the number of cells emerges (fig. 3.5). The correct pattern can be obtained from just the number of the time slice alone. Consider fig. 3.5, for example and ignore the buffer (coarsest region). The alternating pattern consists of 6 and 7 cells for all regions, where the mesh kernel is also included in the following. This differs from fig. 3.5 and would result in a ragged right boundary. In table 3.1 the eight time slices are listed in reverse (as they appear in fig. 3.5) and the last column contains the list of cell numbers for each time slice's regions. These lists are sorted like in the image such that the kernel is the last number and the resolution decreases to the left. The idea is to convert the row numbers of the time slices starting with 0 to their binary representations as given in the second column. Hereby, it is important to keep the same ordering for the bits as the lists containing the cell numbers. The bit that is representing 2^i must be in the same position as the cell number for the region with resolution 2^i in the list. In this case, the most significant bit (digit) is the first (most left) and corresponds to the coarsest region. Now, if the bit is 0 the cell number for that region is 6 and if it is 1 the number is 7. The same algorithm can also be used for the fixed width scheme where unlike before, the alternating number of cells is different for each region.

The next problem that deserves special attention arises from superimposing lattices with diagonal borders. When superimposing two spatially mirrored diagonal borders that have not yet been extended to a pulse segment, a spacelike cusp will form (figs. 3.7c and 3.7d in contrast to figs. 3.7a and 3.7b, which are parts of collision lattices). If there is more than one additional resolution other than the mesh kernel present on the lattice, the question comes up if the cusps should have single or double-cell tips. The simple answer is that double-cell tips are needed for all resolutions except for the coarsest one in order to be able to introduce the next coarser cusp properly. This can be seen in figs. 3.7a and 3.7b, for example. Here, the second coarsest region

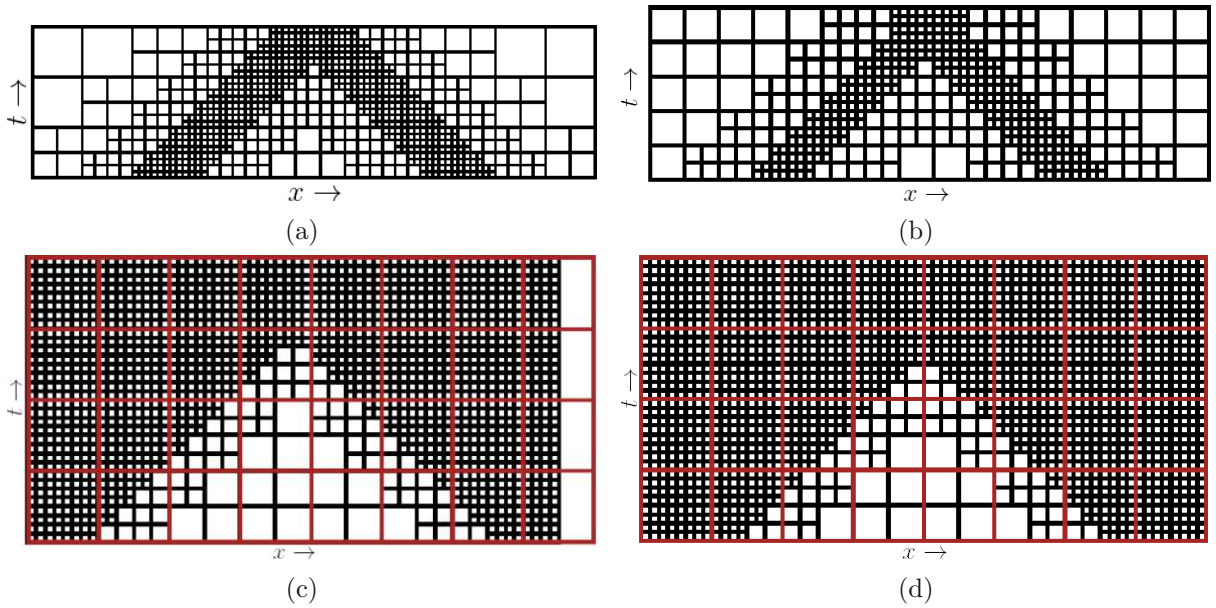


Figure 3.7: The double-cell tip problem for cusps is illustrated by (a), (b) and (c). The solution in (d) is to match the spatial extent of the superimposed lattice with a one step coarser, homogeneous mesh (red).

shows a single-cell tip in both images. It is unclear how the cusp for the coarsest region should look like. In both examples, the spatial mirror symmetry of the lattice is broken and gives suboptimal[3] results. Another argument against single-cell cusps, even for the coarsest region, is the fact that the refinement procedure for the cusp border will be different depending on the cell count of the tip. Therefore, narrowing down the possible border types reduces later work for refinement.

A simple solution for the double-cell tip problem is depicted in fig. 3.7d. By ensuring that the spatial extent of the superimposed lattice is an integer multiple of the next coarser resolution than the coarsest one on the lattice, all spacelike cells, including the coarsest region, will have double-cell tips. Another approach is to make sure the superimposed lattice can be obtained by repeatedly bisecting an imaginary, homogeneous lattice (red in fig. 3.7d) that is one step coarser than the coarsest resolution on the lattice. If the same strategy is applied with a homogeneous mesh that is of the same resolution as the coarsest one on the lattice, this coarsest region will then have a single-cell tip. This is shown in fig. 3.7c, where the lattice does not fill the red mesh by half a red cell. Enforcing these restrictions in the routines for generating the lattices imposes strict relations between the different tunable parameters. We look at the explanations of all parameters and their relations to each other in the next section.

Lastly, note that this double-cell tip problem is not an issue for timelike cusps, because when propagating lattice segments in the temporal direction the smallest multiplication already takes place in multiples of the coarsest resolution. This automatically results in double-cell tips up to (not including) the coarsest region as there are always two time slices of the next finer resolution per coarse time slice.

[3] Additionally, the logic behind this type of superposition is far more complicated than necessary. See the documentation of *mrlattice* for an example implementation.

3.2.3 How to use *mrlattice*

This section is a walkthrough of the tools provided by *mrlattice*. The toolkit is organized into classes or types. The most fundamental lattice type that represents a simple lattice with diagonal borders is the *LatticeArray*. It is composed of a *LatticeArrayScheme* and all the *LatticeArrayProperties* necessary to configure the diagonal borders (fig. 3.8). It also defines the four lattice modifiers that are used to construct more complicated layouts. For the sake of practicality, all of the previously discussed advanced configurations have their own types that inherit from *LatticeArray* and form the class hierarchy depicted in fig. 3.8. This allows to enforce the restrictions for the tunable parameters at the level of each subtype and avoids conflicts.

All discussed types[4] share the same set of tunable parameters:

- **Region width (rw):** Sets the width of the coarsening regions (regions in between the buffer and the kernel) depending on the configured *LatticeArrayScheme*. For the fixed width scheme, it is the spatial extent in units of the mesh kernel. For the fixed number scheme, it is just the fixed number of cells per region.

- **Buffer (b):** Sets the minimal number of cells for the coarsest region on the lattice. In the case of *PulseLatticeArray* and its subtypes in fig. 3.8, there is a buffer at both boundaries of the domain.

- **Coarse steps (cs):** Sets the number of diagonal borders for the building block segments. It is connected to the coarsest resolution r on the lattice: $\max|r| = 2^{\mathrm{cs}}$. In the case of *PulseLatticeArray* and its subtypes (fig. 3.8), this value is only counted for the diagonal border building blocks that are combined together.

- **Finest width (fw):** Sets the minimal number of cells for the finest region on the lattice.

- **Repeat (r):** Sets the number of repetitions for the building block segment.

Depending on the exact subtype, the effects of the parameters change and are illustrated in fig. 3.9. For example, when configuring a lattice with diagonal borders (*LatticeArray*) the "finest width" will determine the minimal width of the mesh kernel. This region shares one border with a coarser region and one boundary with the domain. Depending on the direction of the border, this minimal width will either occur at the first or last time slice, whereas all other time slices are filled up to restore a rectangular domain when propagating a segment. However, when working with *PulseLatticeArrays* the "finest width" will set the width of the mesh kernel that is fully enclosed by coarser regions and is the same for all time slices.

Table 3.2 lists the values and relations between all the parameters for both coarsening schemes for the lattice types. Enforcing the restrictions for double-cell tips on the minimal value and step size is sufficient to determine all possible values without explicitly providing them all. This is possible because when the condition for double-cell tips is met for the minimal values, it will also hold when growing the lattice with a certain step size. The derivations for the formulas in table 3.2 are included in the documentation of *mrlattice*. We borrow the notations for two operators from Python's syntax. We use the // operator for integer divisions where the remainder of the division is discarded and the % operator to extract the remainder of a division. The "$\geq$" shall be understood as having to increment the result of the formula by the step value of the parameter until it is larger than or equal to the given value.

[4] Except for *HomogeneousLatticeArray*.

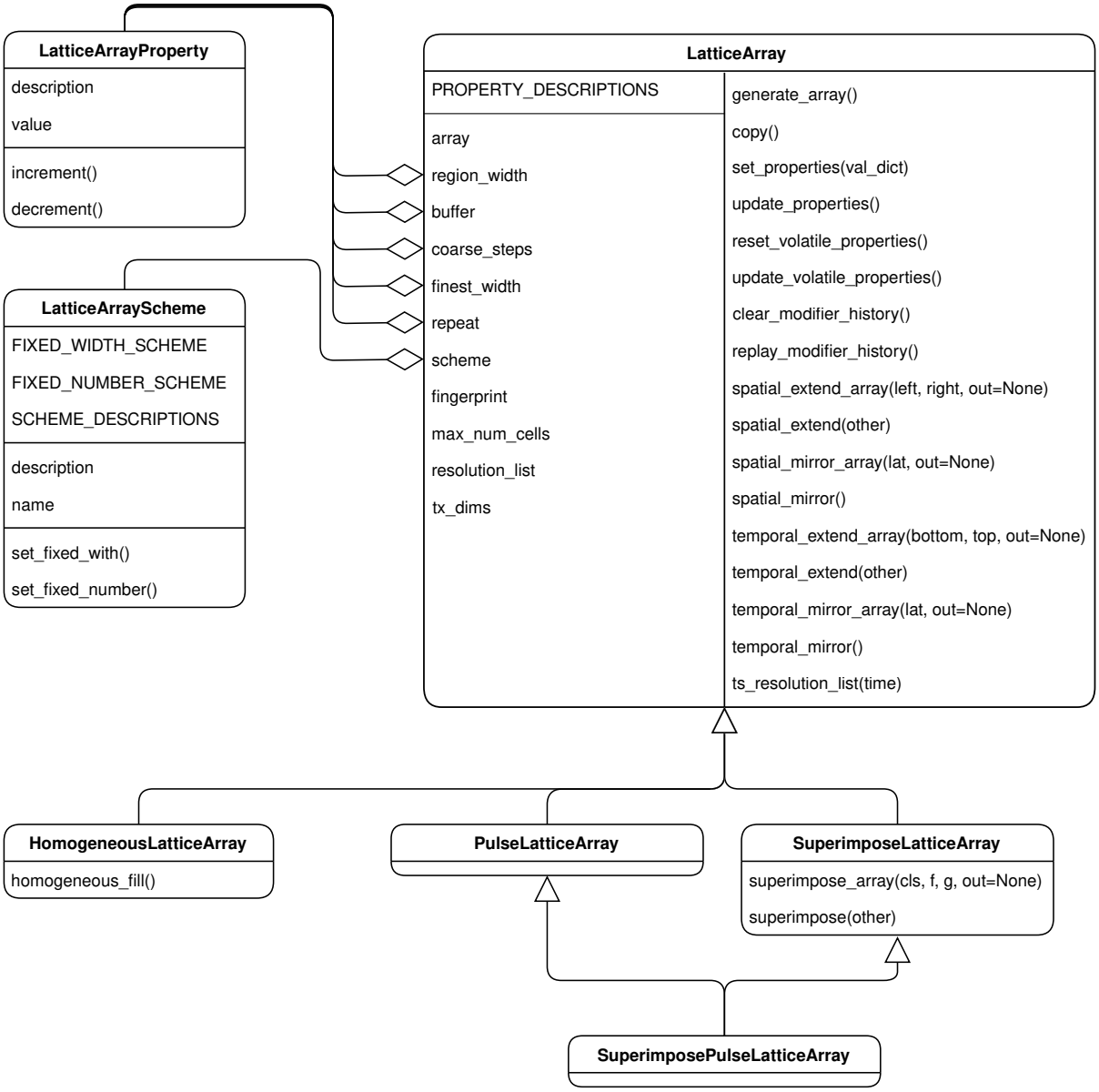


Figure 3.8: The class diagram for *mrlattice*. The base class *LatticeArray* is composed of *LatticeArrayProperties* and one *LatticeArrayScheme*. It generates the lattices with diagonal borders. More complicated types are created via inheritance.

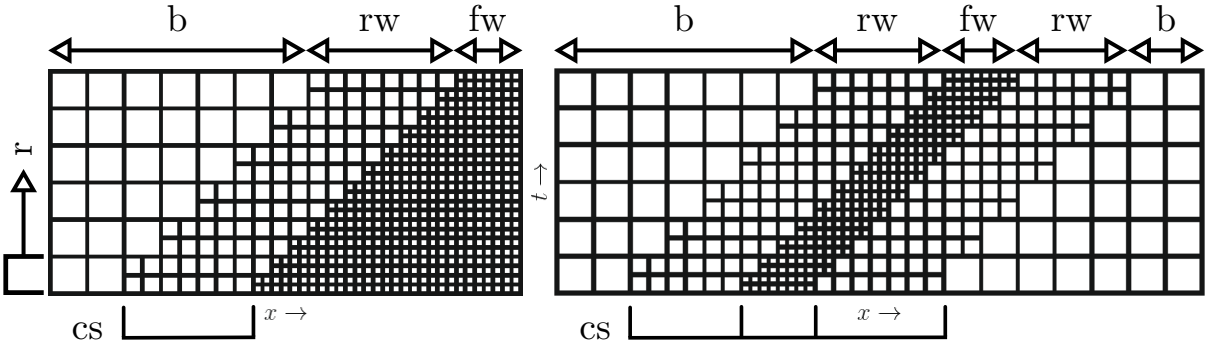


Figure 3.9: Tunable parameters (properties) of *LatticeArray* (left) and *PulseLatticeArray* (right): region width (rw), buffer (b), coarse steps (cs), finest width (fw), repeat(r).

			Step	Min
LatticeArray	Fixed width	rw	2	$3 * 2^{cs}$
		b	1	2
		cs	1	1
		fw	1	7
		r	1	2
	Fixed number	rw	1	6
		b	1	2
		cs	1	1
		fw	1	7
		r	1	2
Superimpose-LatticeArray	Fixed width	rw	2	$3 * 2^{cs}$
		b	1	1
		cs	1	1
		fw	2^{cs}	$2^{cs} - \left[\sum_{i=1}^{cs-1} 2^i(rw//2^i + 1)\right] \%2^{cs} \geq 7$
		r	2	$\left[\sum_{i=1}^{cs-1} 2^i(rw//2^i + 1) + fw\right] //2^{cs} + 3 + b\%2$
	Fixed number	rw	1	6
		b	1	1
		cs	1	1
		fw	2^{cs}	$2^{cs} - \left[(1 + rw)\sum_{i=1}^{cs-1} 2^i\right] \%2^{cs} \geq 7$
		r	2	$\left[(1 + rw)\sum_{i=1}^{cs-1} 2^i + fw\right] //2^{cs} + 3 + b\%2$
PulseLatticeArray	Fixed width	rw	2	$3 * 2^{cs}$
		b	1	2
		cs	1	1
		fw	1	6
		r	1	2
	Fixed number	rw	1	6
		b	1	2
		cs	1	1
		fw	1	6
		r	1	2
Superimpose-PulseLatticeArray	Fixed width	rw	2	$3 * 2^{cs}$
		b	1	2
		cs	1	1
		fw	2^{cs}	$(2^{cs} - \left[\sum_{i=1}^{cs-1} 2^i(rw//2^i + 1)\right] \%2^{cs}) * 2 + \sum_{i=1}^{cs-1} 2^i \geq 6$
		r	2	$\left[\sum_{i=1}^{cs-1} 2^i(2 * rw//2^i + 1) + fw\right] /2^{cs} + 5$
	Fixed number	rw	1	6
		b	1	2
		cs	1	1
		fw	2^{cs}	$(2^{cs} - \left[(1 + rw)\sum_{i=1}^{cs-1} 2^i\right] \%2^{cs}) * 2 + \sum_{i=1}^{cs-1} 2^i \geq 6$
		r	2	$\left[(1 + 2rw)\sum_{i=1}^{cs-1} 2^i + fw\right] /2^{cs} + 5$

Table 3.2: Restrictions for the parameters region width (rw), buffer (b), coarse steps (cs), finest width (fw) and repeat (r) of all lattice array types. The operator // stands for integer division and % stands for remainder division.

To motivate these equations without going through the full derivation, consider the following two reoccurring sums

$$\text{Fixed width:} \quad \sum_{i=1}^{\text{cs}-1} 2^i(\text{rw}//2^i + 1), \tag{3.3a}$$

$$\text{Fixed number:} \quad (1+\text{rw}) \sum_{i=1}^{\text{cs}-1} 2^i = (1+\text{rw})(2^{\text{cs}} - 2). \tag{3.3b}$$

They give the spatial extent of the coarsening regions for the two coarsening schemes. Knowing this value is required when checking if the resulting lattice is consistent with the red mesh in fig. 3.7 and if it satisfies the restriction for cusps with double-cell tips. Note that the sums start with 1 and stop at cs − 1 and thus are excluding the mesh kernel and buffer. For the fixed width scheme, the value of "region width" has to be divided by the resolution of each region in order to get the number of cells that will fit. This number needs to be multiplied with the resolution again and yields the spatial extent in units of the mesh kernel. As mentioned before, the correct shape of the diagonal borders for the coarsening regions results in an alternating number of cells that differ by 1 in the spatial direction. The +1 in eqs. (3.3) selects the larger one of these numbers. Similarly, for the fixed number scheme, the sum over all cell resolutions of the coarsening region just needs to be multiplied by the number of cells ("region width") to get the width of the coarsening region.

Table 3.2 highlights a clear dependency tree for the tunable lattice parameters. If, for example, the value of "coarse steps" is changed, the minimum value and step for other parameters need to be updated. Then, the current value of these parameters needs to be adjusted as well so that they are consistent with the new minimum value and new step size. More complex lattice types have more dependencies. In general, it is safe to assume the following dependency chain where parameters depend on the values to their left:

$$\text{buffer} \prec \text{coarse steps} \prec \text{region width} \prec \text{finest width} \prec \text{repeat}.$$

The design of *mrlattice* and its lattice types tries to provide an intuitive way to work with the lattice objects in the spirit of "pythonic" code. The interwoven connections of the tunable parameters are achieved via class composition and realized by making use of Python *properties* as instance attributes with getter and setter methods. For configuring a lattice array instance, we first need to specify a coarsening scheme. Then, we can set the values for all the tunable parameters. Note that changing parameters that are dependencies for others will affect the values of the dependent parameters. As the last step, we can generate the lattice array. Different ways to visualize the resulting lattice as images or via ASCII characters in the terminal are included in *mrlattice*.

Particularly useful tools for working with lattice types from Jupyter notebooks are the custom *LatticeArrayWidgets*. Screenshots of the two provided widget types are given in figs. 3.10 and 3.11. The first type (fig. 3.10) combines all the necessary steps for configuring lattice arrays in a clean interface. We can generate lattice plots by using the associated buttons. The "Export Snippet" button prints out a valid code snippet that reproduces the configured lattice and that can be pasted into scripts for further use. There is one such widget for each lattice type where the parameters' descriptions are updated accordingly. The second widget type (fig. 3.11) provides easy access to all the previously introduced lattice array modifiers. When a modifier is applied to a *LatticeArray* instance, that instance is permanently affected by the modifier. Even if its parameters are changed afterward, the effect of the modifier (or the "modifier history") is replayed onto the new lattice. The modifier history is independent of any instances used as the partner for the modifications and saves all the information necessary to recreate the partners when replaying the history.

Using the modifiers directly is a great way to create new meshes without relying on preconfigured lattice types. However, not all possible modifications lead to valid lattices. Therefore, an extensive validation testing suite is included in *mrlattice*. There are different tiers of test suites that are designed for different lattice types. All provided lattice types pass the test suite. The validation testing suite and its usage is discussed in the documentation in more detail.

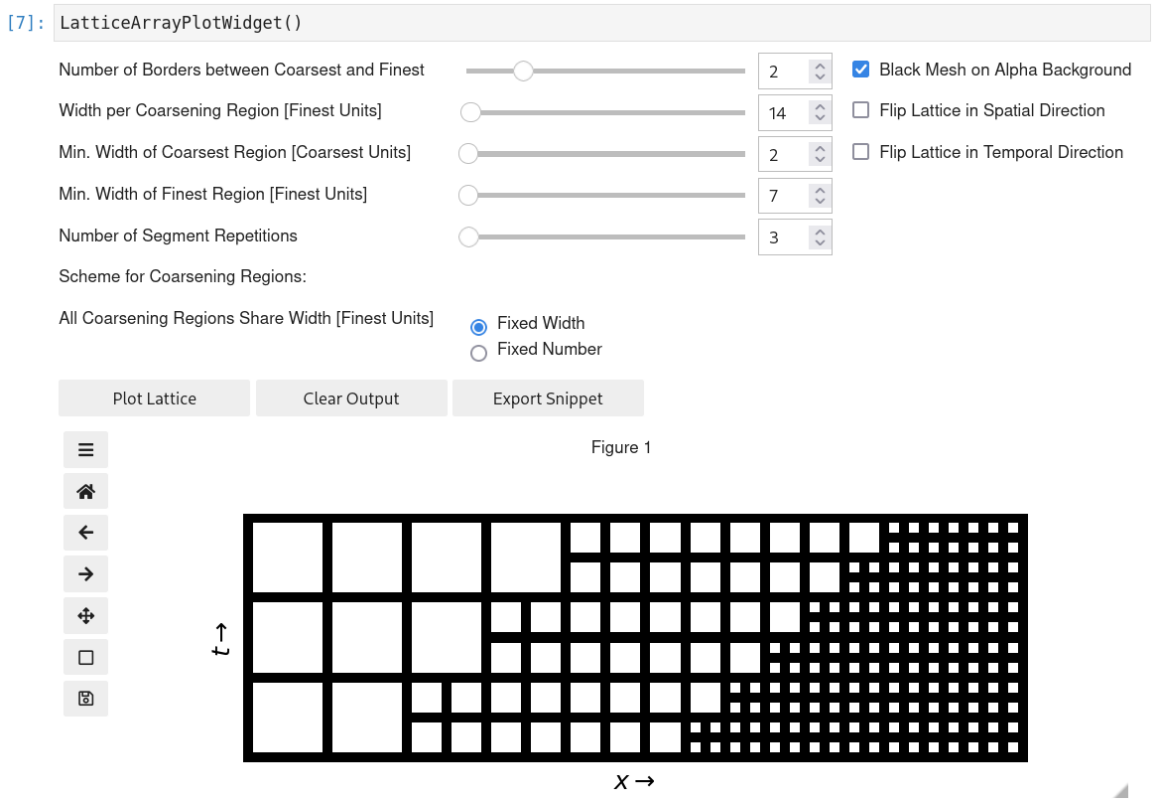


Figure 3.10: Jupyter widget for configuring lattice arrays.

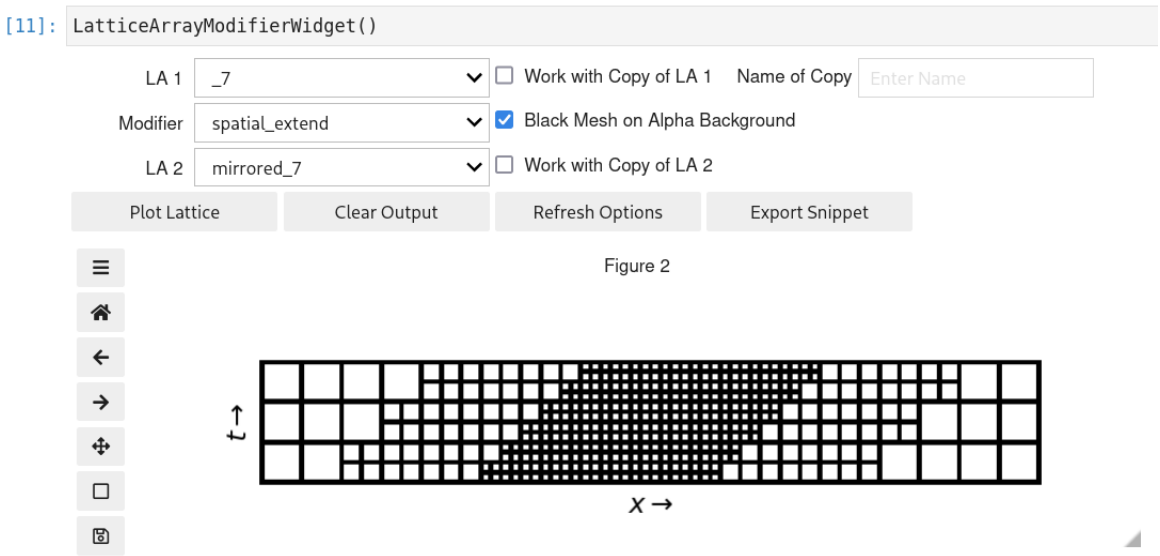


Figure 3.11: Jupyter widget for applying modifiers to lattice arrays.

Chapter 4

Scalar adaptive mesh (*sam*) solver

When working with the complex mesh configurations developed up until now, the solver for calculating the time evolution of a given problem needs to be more comprehensive. Typically, the core algorithm of a simple solver on homogeneous lattices can be reduced to two nested loops. The inner loop runs over all lattice sites of the next time slice whose values are unknown and applies the appropriate update stencil (eg. eq. (2.10)) to each of them. The outer loop just advances the current time step. A more detailed explanation with code samples is given in [17].

The most immediate change is the addition of the border refinement to the inner loop. It does not only require the implementation of the refinement procedures itself but also needs logic to determine the border types to select the correct update equations. The lattice array must be interpreted by the solver because it encodes the shape of the mesh.

Another aspect to cover is memory management and the underlying data structure that represents the discretized values the solver works with. No longer is it possible to simply use the same flat array that essentially shares the layout of a time slice for all time steps. Because now, lattice cells are added and removed throughout the time evolution, the neighbor relations between cells on the lattice do not translate to neighboring array elements anymore. For example, if a region in the middle of the time slice changes, it would be necessary to rearrange all the values in the array to maintain the correct neighbors. This makes it difficult to iterate over a time slice and access its values directly. An intelligent management scheme for fetching the values from a data container for the update step is important.

As will become evident in chapter 5, the border refinement process needs to store more data than there are cells on the lattice. These additional cells are called "ghost cells". In terms of their physical properties and contributions to the time evolution, there must not be any distinction between normal and ghost cells. Typically, ghost cells are redundant cells that overlap with other cells on the lattice but have a different resolution. Because ghost cells are specific to refinement, the solver is responsible of their placement and not the *mrlattice* tool that generates the mesh.

The *sam* (scalar adaptive mesh) Python package[5] is being developed to fulfill all of these tasks. The general architecture is inspired by the design patterns explored in [25]. The different concerns, which are data management, border refinement and interfacing the lattice, are abstracted to types with clean interfaces connecting them. The *ScalarField* class ties these pillars together and represents the complete solver.

In this chapter, we briefly go over the design and components of *sam*. After discussing the main loop's logic and data structure, we reiterate the concept of working with discretized values on differently resolved regions and present methods for evaluating analytical initial conditions. Then, we look at systems in place to debug and validate update and refinement equations. We

[5] The source code is hosted on https://gitlab.com/openpixi/sam_scalar_adaptive_mesh (visited 09/2022).

conclude with a demonstration of how to use *sam* and set up a simulation from start to finish. The refinement equations and their implementation as part of the interface in the form of border types are the topic of the next chapter.

4.1 Main loop

The main strategy of the solver that works on a lattice with multiple differently resolved regions simultaneously is to separate dealing with the border refinement from the time evolution that takes place within a single region. In this sense, each region can be updated independently from the others by treating the region borders like domain boundaries. The boundary conditions (the values of the cells at the region borders) are then taken as the result of the border refinement.

Starting from the outermost loop that runs over the time steps, it is no longer possible to update the entire next time slice for each iteration. The temporal resolutions of the cells of each region set the time interval for updating that region. This results in the basic rule that only those resolutions whose coarsening factors (as a power of 2) are integer divisors of the number of the next time step are considered. For each region that needs to be updated at any given time step, a series of steps is completed, only the last of which is the actual time evolution of the region. The other steps are responsible for interpreting the lattice and performing the border refinement. In the following, we discuss the ideas and responsibilities of each of these steps. Their implementation can be looked up in the source, together with documentation.

Step 1: Data container preparation

The first step is to prepare the data container for storing the values of the next time slice that are calculated below in step 4. Because of the simplicity of the update stencil, it turns out that it is only necessary to save two time slices concurrently. The values of the next time slice are completely determined by the values of the previous two slices. Furthermore, each newly calculated value can replace the value of the previous time slice that was used during its calculation. The only exceptions to this are the region borders. The possibility of changing cells at the borders requires to free or allocate new spots in the data container.

Step 2: Cusp and border detection

The next step is to interpret the lattice array and detect cusps and changes to border types. The detection of the border types is based on comparing their positions. The direction of diagonal borders is directly given by the movement of the border from one time slice to the next. If there is no movement, the border is straight. Distinguishing coarse-to-fine or fine-to-coarse borders is done by comparing the resolutions of the regions at the border. If a diagonal border changes its direction or changes to a straight border, the new straight cusp border type is identified. Note that a border can only change its type after the same time interval that also specifies the update interval of the coarser region attached to the border. This time interval will always be twice as large as the update interval of the finer region. From the point of view of this finer region, there are always two successive update steps where the border does not change. Therefore it makes sense to only consider the change of the coarser region.

The logic behind the detection of lattice cusps relies on a single condition. We recall any of the previous figures of lattices with cusps (i.e. fig. 3.7). Whenever there is a cusp emerging or disappearing, the number of regions of the last time slice containing the cusp and of the first time slice without the cusp differs by 2. This gives rise to a condition that is very straightforward to implement. However, there are some obvious limitations as a result of its simplicity. On the one hand, this condition limits the number of possible cusps per time slice to just one. If there

were more than one, the number of regions would change by a number different than 2. On the other hand, this condition fails to trigger for cusps that are located at the domain boundaries. In this case, the number of regions would only change by 1.

Once a cusp is detected, it still needs to be categorized as emerging (CFTC) or disappearing (CCTF). Because cusps are restricted to always being formed by two coarse cells embedded in a finer region, the number of cells in this finer region will change by 4. The number will be exactly 4 only if the entire lattice was homogeneous up until the cusp. If the cusp is somewhere in the middle of the time slice, the number can also change due to the borders of the embedding region with other regions. This will manifest as a shift in the starting or ending positions of the embedding region. Emerging or disappearing cusps are more complex than changing borders. Here, a border does not change, but two are either created or removed. Which type of borders are attached to either side of the cusp can be determined by the means described before.

Step 3: Cusp and border update

In this step, the actual refinement calculations are performed for all borders that share a region that needs to be updated at a given time step. Compared to applying the same stencil at all time steps, refinement consists of a multi-phase cycle where different calculations are performed for subsequent time steps. One reason for this is that, as previously mentioned, the finer resolved region at the border experiences a change of the lattice only after every second time step. This two-step cycle is typical for the considered lattice borders, whereas cusp types will have more steps per cycle. It is important to process the refinement before the standard time evolution of the regions in the next step because the refinement process typically depends on more values that will no longer be available after that time evolution.

Step 4: Region evolve step

The last step is to apply the standard update stencil to all regions' cells that need to be updated. There is a small subtlety with the boundary conditions, however. Treating all regions universally, regardless of their position on the lattice, is only possible if the boundary conditions for the domain boundaries are identical to the region borders. Namely, fixed boundary conditions are necessary that make use of the borders' cells calculated one time step earlier in step 3.

4.2 Border refinement interface

As previously mentioned, one pillar of *sam* consists of the implementation of the refinement procedure. To separate concerns and define a clear interface, borders and cusps are represented as subtypes of an abstract base class. The key elements of any class implementation are the management of any additional data (ghost cells) and access to the main data managed by the solver for writing out results. Both are standardized across all border subtypes, which makes them replaceable with each other. Each type of border provides its own implementation for the refinement equations as part of a multi-phase update cycle. This cycle reuses operations that are shared among all border types, including the base update stencil of the solver. Being able to use the same implementation of the update stencil, which is also used in step 4 above, for all types is one benefit of the standardized interface. Still, border types are not independent of each other. As borders can change over the course of the time evolution, instances of border types need to be replaceable and still be able to access the ghost cells of their predecessor.

Another responsibility of the border classes is to provide means for inspecting the values of their ghost cells because they are not accessible from the solver directly. Therefore, each border class can generate output that is especially useful for plotting these values.

4.3 Data structure

The data structure implemented in *ScalarField* turns out to be rather complex. It follows a similar strategy as before. Each region on the lattice is treated somewhat independently from all other regions. Through this, the goal is to embrace the workflow of running a simple update step on each region as naturally as possible. There should not be any complicated steps required to fetch a region's data.

Take for example, one of the green colored regions of fig. 4.1a and consider it isolated from all others. Initially, two arrays are filled contiguously with the data for all cells of the previous and current time slices in matching order. Over the course of the time evolution, existing cells will be removed from the front and new ones will be appended to the end of the region. Now, it is desirable to have the layout of the data arrays match the order of the region's cells for each time step so that the update stencil can be moved along the arrays of the previous and current time slices without issues. The stencil expects neighboring values in the array to match the cells that are neighbors on the lattice. In order to keep the data array contiguous and matching the ordering of the cells, it would be necessary to perform a lot of memory copying.

Preventing this unnecessary memory overhead is desirable. The method implemented in *ScalarField* introduces an access layer in between the data arrays and any operations working with data. This layer consists of arrays of pointers to elements of the data arrays. Any region modifications, like adding and removing cells, are carried out on this access layer only. The main reasoning behind this approach is that array modifications are much cheaper on arrays containing pointers. For the presented application, the difference between pointer and data arrays is only the size of the data type because both are one-dimensional. However, even in the case of two interacting fields, as discussed in [15], the data arrays contain two times more elements that are stored in two values per cell instead of one. Thus, two times more overhead for shifting these elements around is expected. The benefit of the access layer increases for higher-dimensional theories that store more data values per cell.

In practice, this translates to the update stencil naturally working with arrays that redirect memory access to the actual data. This data itself will no longer be contiguous with respect to the lattice cells. As depicted in fig. 4.1c and considering only one green region again, the numbers with the same color in the upper row of the Index container represent the cell indices when counting with increasing x (from left to right). Directly below them, the pointer array of that region is given. In this case, pointers are just integer indices for the data array in the Data container further down. The pointers point to vastly different places in the data array. However, the positioning of the pointers with respect to the colored numbers encodes exactly the relation of where each cell's data is stored.

The access layer also includes a bookkeeping mechanism that keeps track of which elements of the data arrays are in use and which are empty. This is necessary because the highly performant Numpy arrays in Python do not support in-place adjustments for array sizes. In fig. 4.1c, occupied spots are marked with three dots and free spots are empty. Bookkeeping is a crucial component, even for regions that do not change their total number of cells but just move them. For example, the region furthest to the left (yellow) in fig. 4.1a will require one additional spot each time step it is updated. But, also the green region requires additional cells at each update step. The only difference is that the green region also returns free spots, which in total account for no additional occupation of the data array. In fact, because of the fixed coarsening factor of 2 for lattice generation, each region border can only shift by a maximum of two cells. This can be exploited to calculate the memory requirement for each region in advance. For the special case of regions that do not grow or shrink monotonically in time (like the green region), it is therefore only necessary to reserve a predetermined number of extra spots in the data array. This will guarantee that at all times, there is enough free space in the data array.

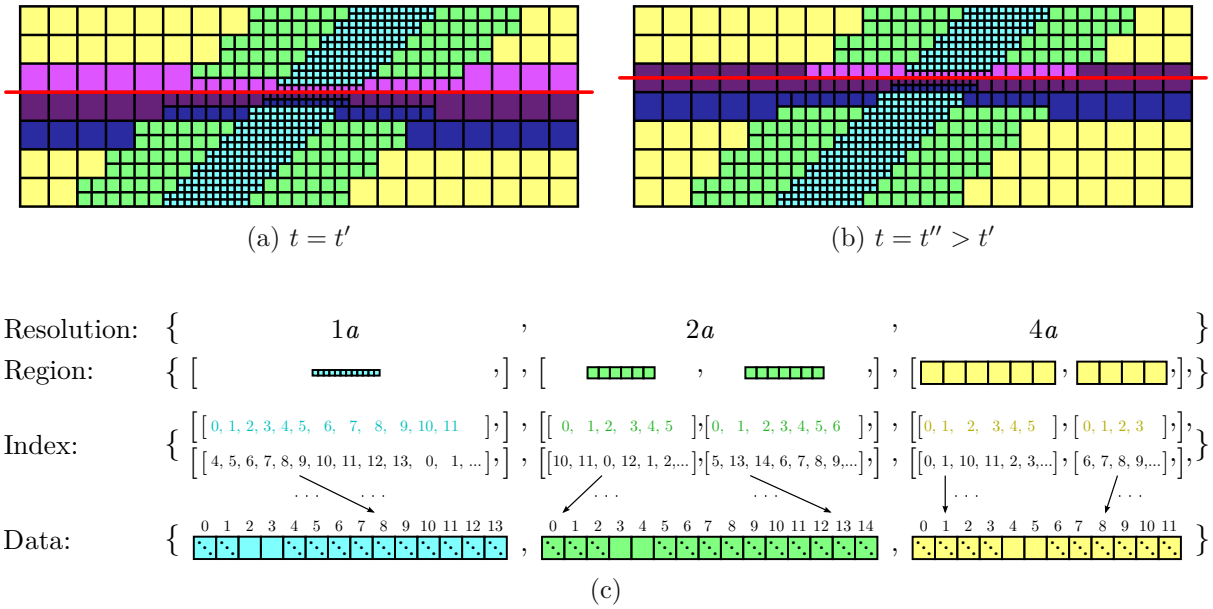


Figure 4.1: (a) and (b) highlight the previous (blue), current (purple) and next (pink) time slices for the time step t indicated by the horizontal red line. The resolutions of the regions are color coded: a for turquoise, $2a$ for green and $4a$ for yellow. (c) is a snapshot of the data alignment of *sam* for (a). Each region has a pointer array in the Index container and shares a data array in the Data container with all regions with the same resolution. Occupied spots in the data arrays are marked with dots.

The step to apply this data structure to arbitrary lattices consists of collecting one set of all the access layer and data arrays for each region in a list. Now, consider the differently resolved regions of the pink time slice of fig. 4.1a. Each region with resolution $1a$, $2a$ or $4a$ has one index array and one data array. This results in the data graph of fig. 4.1c. The colored blocks in the Region container illustrate how the cells of the pink slice are rearranged for the Index container. However, there is one obvious deviation from the strategy of keeping the regions independent. The data arrays for all regions with the same resolution are concatenated to just one. This reduced level of separation is a compromise between flexibility and memory requirement. If each region would be handled independently, in total more extra spots in the separate data arrays would be required, where each region has its own free spots. But, when we do the accounting it becomes clear that fewer extra spots would be sufficient, if the regions' data arrays are combined.

In order to see this from fig. 4.1, three time slices[6] are highlighted in pink, purple and blue in both figs. 4.1a and 4.1b. The horizontal red line indicates the current time step. The pink time slice consists of all cells that will be updated for the next time step. Recall that the update stencil writes the values of the pink time slice into the spots used by the previous time slice in blue. Therefore, cell shifting operations are not to be applied to subsequent time slices but time slices that morph into each other: the previous and next slices. In fig. 4.1b the red line is moved two time steps ahead and the yellow regions are no longer part of the pink slice to be updated. Their index arrays stay untouched and will only be accessed during the border refinement for the green regions.

[6] A time slice in figs. 4.1a and 4.1b would actually have the height $\Delta t = a$. Loosly referring to the highlighted slices with $\Delta t > a$ as time slices, makes it easier to write and where needing to be precise we use the explicit "update time slice" for the time step at which a region is updated.

After each time step, the references to the current and previous data arrays that have been modified are swapped so that they now reference the previous and current slices respectively. This makes sure that at the start of each update procedure the correct data is referenced for the previous and current time slices. Combining equally resolved regions' data arrays still leaves enough flexibility for the solver to work in this way because if at a time step a region with a certain resolution is updated, all regions with that same resolution are also updated. Therefore, the data array for all regions with a certain resolution only contains data that corresponds to the same time slice.

4.4 Initial value calculation

The EOM eq. (2.2) and their discretization eq. (2.9) are second-order (discrete) partial differential equations in both time and space. Hence, two initial conditions for each dimension are necessary. For the spatial direction, we use Dirichlet boundary conditions. As already introduced in chapter 3, we fix the first ($\min x$) and last ($\max x$) values ϕ_x of each time slice to the value 0. The simple update stencil of the solver can easily deal with these conditions by only updating all values in between and including the $\min x + 1st$ and $\max x - 1st$ positions.

In this chapter, we discuss the temporal initial conditions. In order to start a time evolution, the initial value boundary conditions are the most fitting. Here, all the values of the first two time slices are provided to the solver from the beginning by means of an analytical expression as a function of the single variable

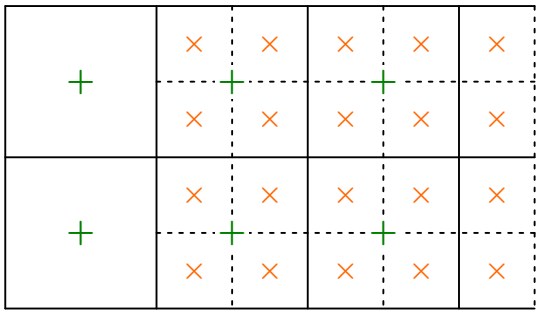
$$\xi = x - vt. \tag{4.1}$$

In the realm of finite difference methods, analytical expressions are approximated by discrete values on lattice sites. Each site with coordinates x and t belongs to a lattice cell. The examples already presented in chapter 3 sample a Gaussian, but no details are given for what exact values are taken for x and t that define the lattice sites. Indeed, for a homogeneous lattice, these details are trivial. As long as the values have a consistent offset to a fixed point (e.g. the center or lower-left corner) of the lattice cell they represent, they do not really matter. By exploiting the translational symmetry of the homogeneous lattice, the sites can always be shifted to a desired position within the cells. As a result, interpreting lattice sites on intersection points, cell centers, or any other way is analogous.

The translational symmetry is broken[7] for lattices with multiple, differently resolved regions. It is no longer possible to shift the lattice by a fraction of an arbitrary cell's dimension and have all sites now at the centers of all cells when they previously were taken at the intersection points. Therefore we need to clearly define the meaning of a lattice site and the relations between sites of different regions.

A natural way of sampling analytical expressions on a discrete grid is to take the centers of the lattice cells as the lattice sites. However, this leads to a problem that is illustrated in fig. 4.2. The lattice site sets are disjoint so that the green sites do not coincide with every n*th* orange site. How the relation of sites on coarser regions to sites on finer regions enters the refinement equations is the topic of chapter 5. But already when evaluating the initial conditions some ambiguity arises. We can think of two possible ways to calculate the values of the green sites that are part of the coarser region. On the one hand, we can directly insert the x and t values corresponding to the sites for ξ. In this case, each region only requires the evaluation of the analytical expression at its first two time slices. The results are exact values in the sense that for the first two time slices the error w.r.t. the analytical expression is 0.

[7] To be more precise: Some discrete symmetry may still be preserved. For example, for lattices with diagonal borders, shifting all sites parallel to the border by the size of the largest cells is still a symmetry operation. But, this translation does not change the positioning of the lattice sites within the cells.



$$\times \in \Lambda^a = \left\{ x \,|\, x = \sum_{\mu=0}^{1} n_\mu \hat{a}^\mu, \quad n_\mu \in \mathbb{Z} \right\}, \tag{4.2}$$

$$+ \in \Lambda^b = \left\{ x \,|\, x = \sum_{\mu=0}^{1} n_\mu \hat{b}^\mu, \quad n_\mu \in \mathbb{Z} \right\}, \tag{4.3}$$

with $b^\mu = 2a^\mu$.

Figure 4.2: Interpretation of cell centered lattice sites that belong to differently resolved regions.

On the other hand, it is conceivable that this exact way of putting the initial conditions on the lattice introduces some conflict with the refinement process. From fig. 4.2 it is visible that the upper time slice of the coarser region represents values for time steps that are not part of the initial conditions of the finer region. The upper two slices of the finer region that overlap with the second coarse slice will be calculated later during the time evolution. These calculations then include refinement steps that would also give results for the coarser cells at the region borders whose values are already fixed by the initial conditions. Therefore, the initial values of the coarser regions might be better calculated by using the same relations between differently resolved regions as are used later on for the refinement process.

In particular, the relation is the approximation of sites on coarser regions by overlapping finer regions' sites. First, the mesh kernel is extended to cover the entirety of the first two time slices. Note that because coarser cells have larger time steps, there are then more than two slices of the mesh kernel required to cover all resolutions' regions. In this context, we refer to time slices with the temporal width of each region. These additional mesh kernel cells are ghost cells that do not exist on the lattice but are introduced only as a means for calculations. The values for ξ are then taken as the positions of these ghost lattice sites and the values of the actual, coarser cells are approximated using these ghost cells.

From a geometrical point of view, the value $f(x,t)$ of a green site at (x,t) can be approximated based on the values of its four closest, orange sites at (x_i,t_i) using bilinear interpolation. The formula translates to a weighted average where the weights are calculated based on the distances between (x,t) and each of the (x_i,t_i). Written out explicitly this becomes

$$\begin{aligned} f(x,t) &\approx \frac{t-t_1}{t_2-t_1} f(x,t_2) + \frac{t_2-t}{t_2-t_1} f(x,t_1) \\ &\approx \frac{t-t_1}{t_2-t_1} \left(\frac{x-x_1}{x_2-x_1} f(x_2,t_2) + \frac{x_2-x}{x_2-x_1} f(x_1,t_2) \right) \\ &\quad + \frac{t_2-t}{t_2-t_1} \left(\frac{x-x_1}{x_2-x_1} f(x_2,t_1) + \frac{x_2-x}{x_2-x_1} f(x_1,t_1) \right). \end{aligned} \tag{4.4}$$

For the signs to appear as above $x_1 < x < x_2$ and $t_1 < t < t_2$. We can further simplify eq. (4.4) by using the constant coarsening factor for the lattice generation. Each $|x - x_i| = (x_2 - x_1)/2$, which is the same for t and also holds for non square cells (i.e. $C \neq 1$). The result is

$$f(x,t) = \frac{1}{4} \left(f(x_2,t_2) + f(x_1,t_2) + f(x_2,t_1) + f(x_1,t_1) \right) + \mathcal{O}\left((x_2-x_1)^2, (t_2-t_1)^2 \right), \tag{4.5}$$

a simple average over the four values of the higher resolved sites' values. We analyze the error term of eq. (4.5) in appendix A. Suffice to mention that it has the same quadratic dependence on the lattice spacing as the finite differences in eqs. (2.6) and therefore using eq. (4.5) should not introduce a lower order error for the solver. This averaging relation of fine to coarse cells

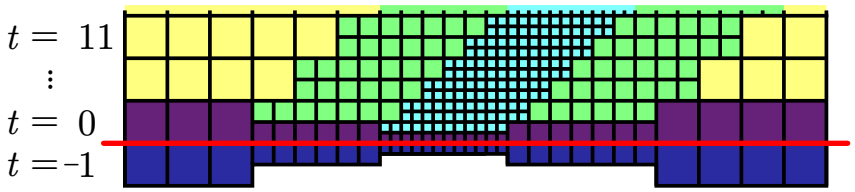


Figure 4.3: The pulse lattice with the three differently resolved regions in yellow ($4a$), green ($2a$) and turquoise ($1a$) only consists of all slices above the red line. For evaluating the initial conditions the blue ghost slice is appended at $t = -1$ by replicating $t = 0$. The results of the analytic expression for $t = 0$ are interpreted on the red line.

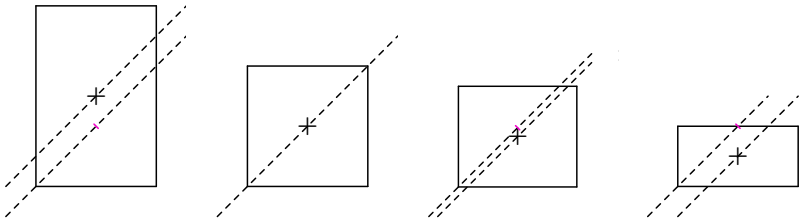
is universally applicable to any two resolutions. Calculating the value of a cell with a size of $4a$ as an average of four cells with size $2a$ is the same as using sixteen cells of size a. We can easily prove this by inserting averages over four size a cells for the values of the size $2a$ cells in the calculation that makes use of size $2a$ cells to calculate the size $4a$ cell's value.

4.4.1 Implementation for *sam*

Both the cell-centered and cell-averaged initial conditions are implemented in *sam*. The first amendment to the previous section is the clarification of how the first two time slices actually look like. In contrast to fig. 4.2 now fig. 4.3 is used as a real example for a pulse lattice with three differently resolved regions that are colored in yellow ($4a$), green ($2a$) and turquoise (a). The time steps t for each update time slice are annotated to the left of the mesh. In the following, we take care to distinguish these time step labels of the solver from the physical real-time τ. The first two time slices which take over the initial conditions are colored in purple and blue. The complete lattice, as it is generated from *mrlattice*, only starts from the horizontal red line onwards. The blue time slice for $t = -1$ is a pure ghost slice that is constructed by extending the first slice of the lattice to negative time. The reasoning behind evaluating the initial conditions on the two highlighted time slices as opposed to the $t = 0$ and $t = 1$ slices is to reduce the conflict of refinement equations and exact initial conditions. As previously mentioned, evaluating the initial conditions exactly at coarse lattice sites can cause problems because values for time steps that are going to be computed later are already fixed. By making use of the blue ghost slice, this is no longer the case. For the first update slice $t = 1$ of the time evolution, only the turquoise region is calculated. Then, for $t = 2$ the green and turquoise regions are updated where the refinement process will be used for values at the border without any conflict. The same applies to the border between the yellow and green regions at $t = 3$. If and what conflicts hide in the cell-centered evaluation method compared to cell averaging is the topic of section 7.4.

From fig. 4.3 it is also visible that the borders' types that appear in the slices for the initial conditions are different from the types for $t > 0$. This will only be the case for diagonal borders that will appear as straight borders for the initial conditions. It is therefore necessary to treat the first refinement cycle of diagonal borders differently. It has to be adapted to the changed geometry while still making use of diagonal border refinement equations. Any additional ghost cells that are required for the special first cycle can be calculated from the analytic expression during initialization.

The horizontal red line in fig. 4.3 also denotes the point in time at which the analytical expression for the initial conditions are interpreted. When evaluating $\xi(x, \tau)$ at $\tau = 0$, the results for varying x must correspond to the values at the red line. However, the red line and these values do not match any region's cell-centered lattice sites. Instead, a centering offset for ξ is required that propagates the initial conditions to the cells' centers. As a result, not



(a) $C > 1$ (b) $C = 1$ (c) $C \lesssim 1$ (d) $C < 1$

Figure 4.4: The centering offset is given by the intersection of the baseline with the characteristic (dashed line) that passes through the cell's center. If the Courant number $C = a^0/a^1 > 1$ the offset is negative, for $C < 1$ positive and exactly 0 if $C = 1$. When not considering C the initial conditions are falsely evaluated on the characteristic that passes through the lower left corner.

even the mesh kernel is actually evaluated at $\tau = 0$ but at later τ. This way of interpreting the analytical expression has the benefit that all regions, regardless of their resolutions, can be treated the same for determining the centering offset.

When calculating the centering offset, the value of the Courant number $C = a^0/a^1$ has to be considered. The physical scale is introduced by the free parameter $dt = \tau/(a^0 t) = \Delta\tau/a^0$ that is the fraction of real-time passed per time step $\Delta\tau$ and the temporal width of one update time step a^0. Then, the temporal axis of the lattice is counted in steps of $dt \cdot a^0$ and the spatial axis in $dt \cdot a^0/C$. The factor dt is crucial to connect the lattice scale, and therefore the units used by the solver to physical units. The analytical expression for the initial conditions has to incorporate these physical units because it is evaluated at $\xi(x, \tau)$. The resolution $r = 2^k$ with $k \in \mathbb{N}^0$ of each region also enters as a factor. All factors are combined so that the left edges of each region's cells are located at $x = dt \cdot r \cdot a^0 \cdot i_x/C$, where $i_x \in \mathbb{N}^0$ is the number of the cell counting from 0. This assumes that x is the offset to the beginning of the region at $x = 0$. Similarly, the value for τ at the bottom edge of the cells is given by $\tau = dt \cdot r \cdot a^0 \cdot t$. Finally, the centering offset ξ_0 results from evaluating $\xi(x, \tau)$ at half stepping with $|v| = 1$

$$\xi_0 = x_0 \pm \tau_0 = dt \cdot r \cdot a^0 \cdot \frac{1}{2C} \pm dt \cdot r \cdot a^0 \cdot \frac{1}{2}. \tag{4.6}$$

The values for ξ at the cell centers are then pieced together from the centering offset ξ_0 and the lower-left corners of the cells as

$$\begin{aligned} \xi &= \xi_0 + dt \cdot r \cdot a^0 \cdot i_x/C \pm dt \cdot r \cdot a^0 \cdot t \\ &= dt \cdot r \cdot a^0 \cdot \left(\frac{1}{2}\left(\frac{1}{C} \pm 1\right) + \frac{i_x}{C} \pm t \right). \end{aligned} \tag{4.7}$$

The plus sign is assumed for left moving pulses with $v = -1$ and the minus sign for right moving pulses, which will be the default if not stated otherwise.

Figure 4.4 illustrates the effect of the Courant number on the calculation of the cell center positions. The panels show individual cells that can be seen as part of the first time slice so that the horizontal red line of fig. 4.3 overlaps with their bottom edges. The dashed lines represent the right-moving characteristic. In this case, and if $C = 1$, the centering offset in eq. (4.6) becomes 0 and ξ takes the value of the lower-left corner. One half time step propagates the value of the initial condition for that ξ to the center of the cell. If $C \neq 1$ but dropped from eq. (4.6), the value of the lower-left corner would be taken. Now, because the characteristics do not pass through the cells' centers, a wrong value would result.

Concluding this section is a list of all the analytical expressions that are provided in *sam* and can be used for initialization.

Unit pulse. The unit pulse is a constant valued function whose amplitude, center position and width can be set as parameters.

$$f(\xi;\ \text{center:}\ c, \text{width:}\ w, \text{amplitude:}\ a) = \begin{cases} a, & c - \frac{w}{2} \leq \xi \leq c + \frac{w}{2} \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

Triangular pulse. The triangular pulse is pieced together from two linear functions that intersect for $\xi = 0$ at the value of the set amplitude. The function's values are restricted to positive numbers. Again, the center, width and amplitude are tunable parameters.

$$f(\xi;\ \text{center:}\ c, \text{width:}\ w, \text{amplitude:}\ a) = \begin{cases} -\frac{2}{w} \cdot a \cdot |c - \xi| + a, & c - \frac{w}{2} \leq \xi \leq c + \frac{w}{2} \\ 0, & \text{otherwise} \end{cases} \tag{4.9}$$

Gaussian pulse. The values are sampled from a normal distribution with location parameters μ and σ and that is scaled with the amplitude. Additionally, as the Gaussian has values $\neq 0$ for $\xi \to \pm\infty$, a cutoff for the function values can be set below which the values will be put to 0.

$$f(\xi;\ \mu, \sigma, \text{cutoff} : c, \text{amplitude} : a) = \begin{cases} a \cdot \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}, & \text{if } a \cdot \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}} \geq c \\ 0, & \text{otherwise} \end{cases} \tag{4.10}$$

4.5 How to use *sam*

As before with *mrlattice*, we include a short, manual-like section explaining the basic steps for using *sam*. For this purpose, a code example is included as listing 4.1. This example also makes use of the `sam.plotting` module for generating the lattice plots. The concept of how plotting hooks into the solver is the same for any observables that can be computed during the time evolution.

Listing 4.1 is a complete minimal working example that produces the plot in fig. 3.1a. There are four separate parts in the code. First, the necessary modules and classes are imported from *mrlattice* and *sam*. The coupling between these two packages is rather strong with the interface being defined by composition. The initialization takes place in the second part. The configured and generated lattice array instance is passed as an argument to the class initializer of the plot and the *ScalarField* solver. The solver takes many additional arguments. After the lattice, the Courant number and global scale *dt* are supplied. The next three arguments specify the implementation of the update stencil (that is not part of *ScalarField* but an injected dependency), a mapping of border subtypes and a mapping of cusp subtypes. The temporal initial conditions of the solver are then set by using the cell-centered evaluation method and a Gaussian pulse shape. Here, the cell-averaged method or other pulse shapes can be chosen as well. The plot needs to be populated with the initial values for the first time slice of the lattice.

The third part of the code listing carries out the time evolution. By making use of the iterator protocol of Python, instances of *ScalarField* are iterables. This allows for the "syntactic sugar" on display where iterating over the instance automatically advances the solver by one time step. The loop variable takes the value of the time step that is current for the body of the loop. In this body, any observables can be calculated. In the example listing 4.1 only the values of the plot are updated with the current time slice's values. After the loop finishes, all time slices of the configured lattice will have taken part in the time evolution and the resulting plot is shown.

```python
from mrlattice.lattice.generate import HomogeneousLatticeArray
from sam.plotting import LatticePlot
from sam.sim import ScalarField
from sam.solvers.leapfrog import leapfrog
from sam.solvers.refinement.cusp import cusp_refinement
from sam.solvers.refinement import BORDER_TYPES
from sam.init import single_gauss_pulse

# lattice array setup
lat = HomogeneousLatticeArray(2, 40, 20)
lat.generate_array()

# plot setup
plot = LatticePlot(lat.array)

# solver setup
sim = ScalarField(
    lat, (1,), 1 / 4, leapfrog, BORDER_TYPES, cusp_refinement
)

# initial conditions
sim.init_cell_centered(
    single_gauss_pulse(mu=10, sigma=3, cutoff=0.05, amplitude=1), direction=+1,
)

# init plot's first time slice
plot.update_field_vals(
    sim.t,
    sim.u_now,
    sim.array[sim.t],
    sim.xidx_now,
    sim.res_list,
    sim.res_list,
    sim.reg_by_res_now,
    sim.p_now,
)

# time evolution
for t in sim:
    # calculations of observables per time step go here

    # update plot
    plot.update_field_vals(
        sim.t,
        sim.u_now,
        sim.array[t],
        sim.xidx_now,
        sim.res_list,
        sim.res_list,
        sim.reg_by_res_now,
        sim.p_now,
    )

# show result
plot.show()
```

Listing 4.1: Minimum working example for creating the results shown in fig. 3.1.

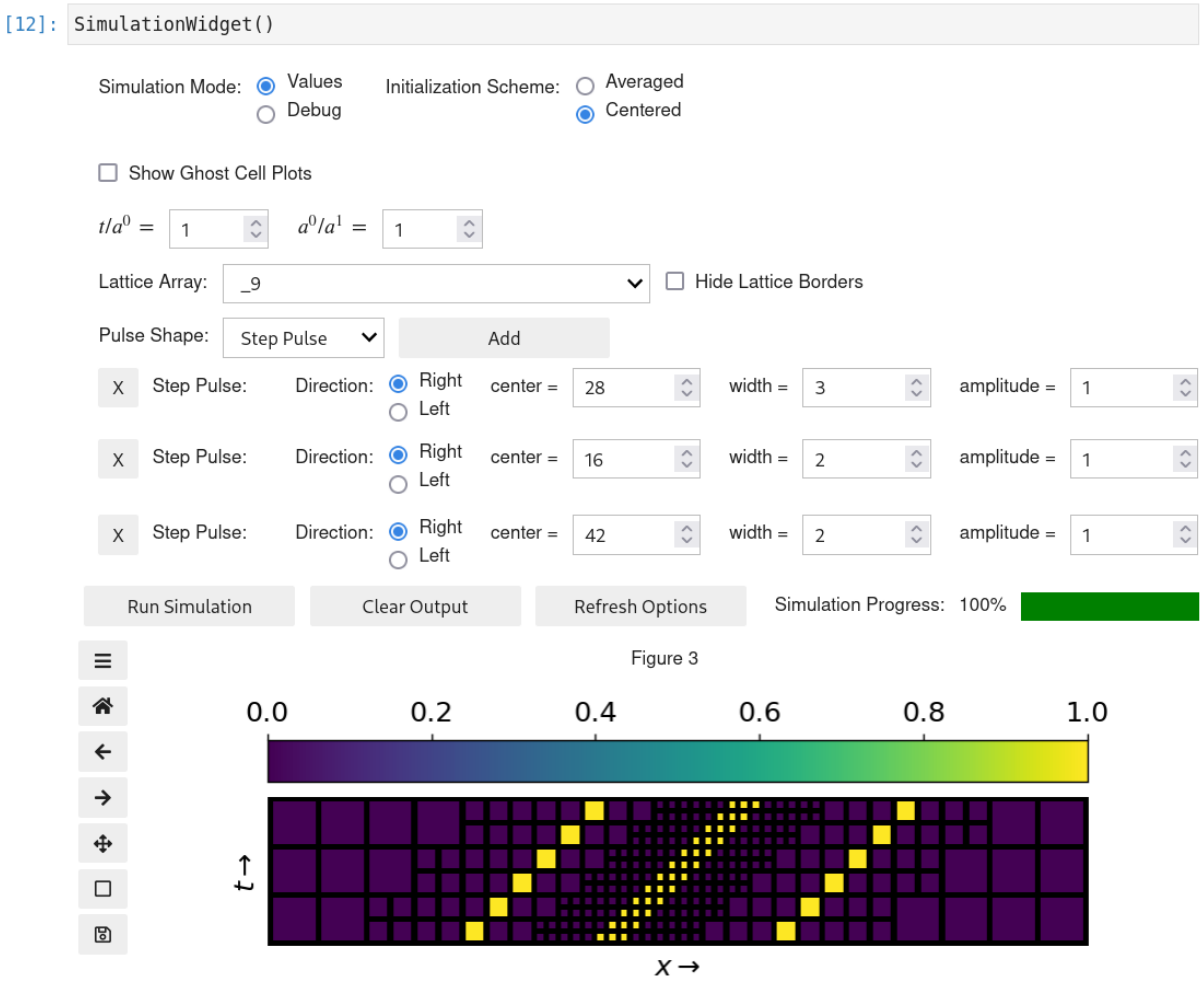


Figure 4.5: Jupyter widget for configuring and running simulations with *sam*.

In addition to the barebones front end using Python code, there is again a custom Jupyter widget provided for configuring and running the solver. An example screenshot for a different configuration than listing 4.1 is depicted in fig. 4.5. Just like the widgets for *mrlattice*, the user interface consists of buttons and input fields. The arguments of the initializer for the *ScalarField* from above are now clearly labeled and laid out. The initial conditions are set by selecting one of the provided pulse shapes, adding it and configuring its parameters. Multiple pulses can be stacked and overlapping values will be added together.

4.6 Debugging and validation

Good, reusable software should be easy to debug and test. The complicated nature of the data structure at the core of *sam* makes it inherently difficult to dissect the components of the solver in the case of erroneous results. Inspecting the buried, raw data, even when using modern debugging tools, is only possible when knowing the implementation details. The author's experience showed that the most common bugs are indexing mistakes for site data. Because the data is not stored contiguous w.r.t. the neighbor relations on the lattice, index errors lead to values popping up at unexpected spots in the time slices and are hard to track down.

Nevertheless, the design and modularity of *sam* allow for a very powerful debugging strategy explained in the following. This strategy can also be used for validation, i.e. proving that an implementation of the update stencil or refinement scheme performs the operations correctly without having to rely on numerical results that may be sensitive to numerical errors.

The underlying concept is using the `unittest.mock` module from the Python Standard Library to patch out logical units of the code. The injected changes to the execution flow will only affect the patched parts and therefore, the entire solver will still work undisturbed. In practice, this boils down to subtyping the evolve stencil or border types and patching single (refinement) equations. A collection of different debugging types is included in *sam*. Also, setting up and running these debugging setups is analogous to the setup presented in section 4.5. Additionally, example setups are provided in the source.

The main objective of the patched code is to communicate verbosely what happens during execution. Hence, we introduce debug values that replace the numerical values that are calculated and written by the solver. These debug values represent certain actions performed on or properties of the lattice cells they are written to. With this change, we naturally get strong, visual feedback with the already familiar lattice plots, like with fig. 4.6.

The debug values are determined by assigning the different "messages" to display to the bits of a bitmask. In the case of *sam*, a 5-bit wide mask proves to be sufficient. Consider the default update stencil for the case where the Courant number $C = 1$

$$\phi_{x+0} = \phi_{x+1} + \phi_{x-1} - \phi_{x-0}. \tag{4.11}$$

Four expressions enter this equation and are identified with four bits of the mask. The fifth bit is used for domain boundary or lattice region border cells that cannot be written by the update stencil. Using binary operations, debug values can accumulate different bits. If this bitmask is then interpreted as a 5-bit integer, each combination of "messages" results in a number between 0 and 31. This number is automatically written to the corresponding lattice site. Table 4.1 lists the interpretations of the different bits for the update stencil eq. (4.11). When feeding the lattice plots with the debug values, a discrete colorbar with colors for each of the integer values is generated.

The example debug value plot in fig. 4.6 shows a lattice with two diagonal borders and three regions with the resolutions $a, 2a$ and $4a$ from left to right. In total, there are 12 time slices on the lattice. At each time step, one of these slices is written to, except for the first slice that contains the initial conditions. Cells with coarser-than-finest resolution end up with multiple

2^4:	"B"	boundary cell	
2^3:	"R"	right neighbor cell	ϕ_{x+1}
2^2:	"L"	left neighbor cell	ϕ_{x-1}
2^1:	"P"	previous neighbor cell	ϕ_{x-0}
2^0:	"W"	cell was written	ϕ_{x+0}

Table 4.1: Debug values for the update stencil eq. (4.11).

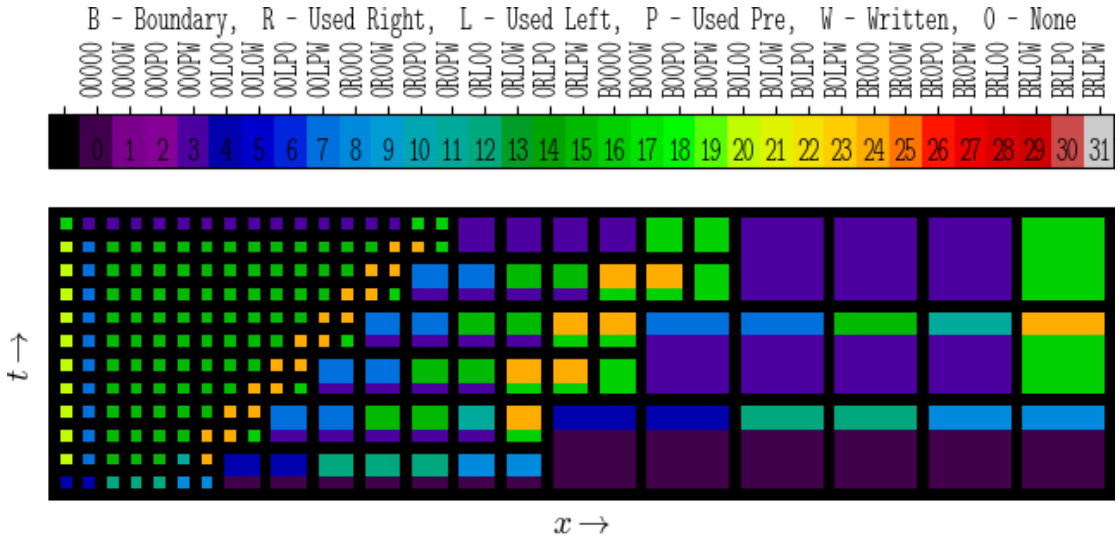


Figure 4.6: Lattice plot with debug values for each time step.

colors that change in the middle of the cells because their temporal width extends over multiple time steps. For steps where the cells are untouched, their previous value is displayed.

The values of the initial time slice are only read. They are either used left (4: `00L00` in dark blue), used right (8: `0R000` in light blue), or used for both (12: `0RL00` in teal). From the first slice that is written onward, boundary and border cells that cannot be calculated by eq. (4.11) additionally get the `B0000` bit (16: light green). This can be seen best in the center of fig. 4.6 for the border cells of the diagonal border between the $2a$ and $4a$ regions. Cells that can be calculated not only get the `0000W` bit (1: purple) for being written but also the `000P0` bit (2: purple). The reason is that the standard working data set of the solver only consists of two time slices, the current and the next. The latter will share memory with the previous slice. The `000P0` bit would have to be written to the previous time slice, but that is in the process of being overwritten as the next slice. Therefore, when adding the `000P0` bit to the previous slice, it will appear on the cell of the next slice that got the same memory location. In the case of eq. (4.11) this results in ϕ_{x+0} and ϕ_{x-0} being identified with each other and purple (3: `000PW`) values for written cells. In summary, all cells that were written to and read end up with `0RLPW` (15: dark green, not to be confused with `B0000` in light green). If they are close to either the left or right border of their region, they will loose the `0R000` or `00L00` bit respectively.

This example illustrates how debug values are used[8] to validate the implementation. We can ensure that boundary and region border cells are identified as such, that the correct lattice sites are used when indexing the data and that operations are performed at the correct time steps.

[8]It can be hard to identify the correct colors of the cells when just looking at saved image files. Instead, it is easier to inspect the debug value plots in an interactive plotting window where the values of the cells can be inspected interactively.

Chapter 5

Refinement equations

The main objective of this chapter is to present and iterate on the refinement equations developed in [15]. Starting point is the concepts behind the derivations that lead to a set of update functions for cells at region borders. The derivations themselves can be looked up in [15]. We also discuss new ways in which the update functions can be combined into refinement equations. For the most promising schemes, we investigate ways to tweak them for the purpose of implementing them in the solver. In subsequent chapters, we study the resulting schemes, identify their weaknesses and analyze the scaling of the discretization error numerically.

The underlying strategy for finding update functions pursued in [15] is to reduce the problem to the simplest case, the non-interacting scalar field on square lattice cells (ch. 2, "free" case from now on) and impose a set of local restrictions on the cells near region borders. These restrictions manifest the symmetries of the theory and EOM. Solving for the refinement equations then becomes a task of simple algebra. The important argument that supports this approach for more complex theories is that even in the interacting case, it shall be possible to approximate the exact time evolution with the same refinement schemes because the error that is introduced is confined to only the border region. This claim is validated in [15] for a selection of examples.

The first restriction introduced in [15] is the conservation of the sum of all field values per time slice, which we call the "norm". In the presence of multiple, differently resolved regions, however, calculating the norm is not straightforward. Because coarser cells also have larger spacings in the temporal direction, they contribute to multiple time slices. The resulting formula for one time slice is

$$a \sum_x 2^{i_x} \phi(x), \tag{5.1}$$

where each cell's value is weighted by the coarsening factor 2^{i_x} of the region it belongs to. The norm is scaled with the spacing of the mesh kernel to introduce physical scale. Figure 5.1 illustrates how the cells contribute and for this example eq. (5.1) evaluates to $a \cdot (a + b + c + d + 2A + 2B)$. For a homogeneous lattice, the norm is conserved by the EOM (2.2) for every time slice. This can be easily checked by inserting the EOM for each summand in eq. (5.1) and tracing the values back to previous time slices with known norm.

As soon as we introduce borders between differently resolved regions, the conservation of the norm depends on the refinement scheme. There are two ways to conserve the norm in this case. Either for each time slice the norm is conserved, just like for the homogeneous lattice. Or, less restrictively, the norm is only conserved when averaging the fine side to coarse cells and then calculating the norm.

The next restriction follows from the fundamental relation between fine and coarse cells that was derived in section 4.4, the averaging relation eq. (4.5). Consider the free time evolution on a fine, homogenous lattice with spacings $a^0 = a^1 = a$ and replace it with a coarse lattice with spacing $b = 2a$ by averaging the values of four fine cells for each new coarse cell. One can

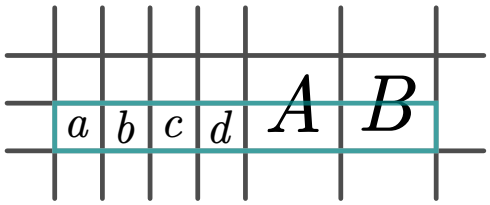


Figure 5.1: The highlighted area is used to calculate the norm across region borders.

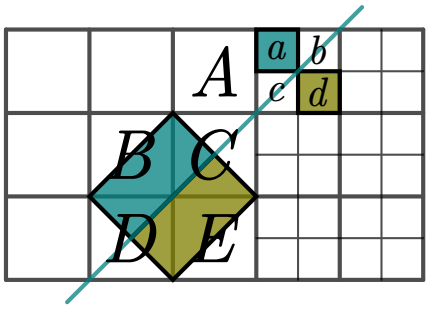


Figure 5.2: The difference of the teal and olive colored regions is preserved by free propagation along characteristics. From [15].

easily show that in this case, the coarse cells' values are consistent with the values obtained by evolving the coarse lattice directly in time, given the initial conditions are identical. The averaging process can be seen as a linear combination of solutions of the linear EOM (2.2) and therefore produces valid solutions. The same argument can be made for the norm eq. (5.1), which for a resulting coarse slice will be equal to the norm of two former fine slices and therefore is conserved by the averaging.

In the context of refinement equations, interesting results can be obtained by mixing coarse and fine cells from different regions in a single equation. When written down for a single coarse update step (i.e. eq. (2.9)) where depending on the setting different coarse cells $\phi_{x+0} = A$, $\phi_{x+1} = B$, $\phi_{x-1} = C$ and $\phi_{x-0} = D$ are replaced by averages of the corresponding fine cells x_1, x_2, x_3 and x_4 for $x = a, b, c, d$, this becomes

$$A = B + C - D, \tag{5.2}$$

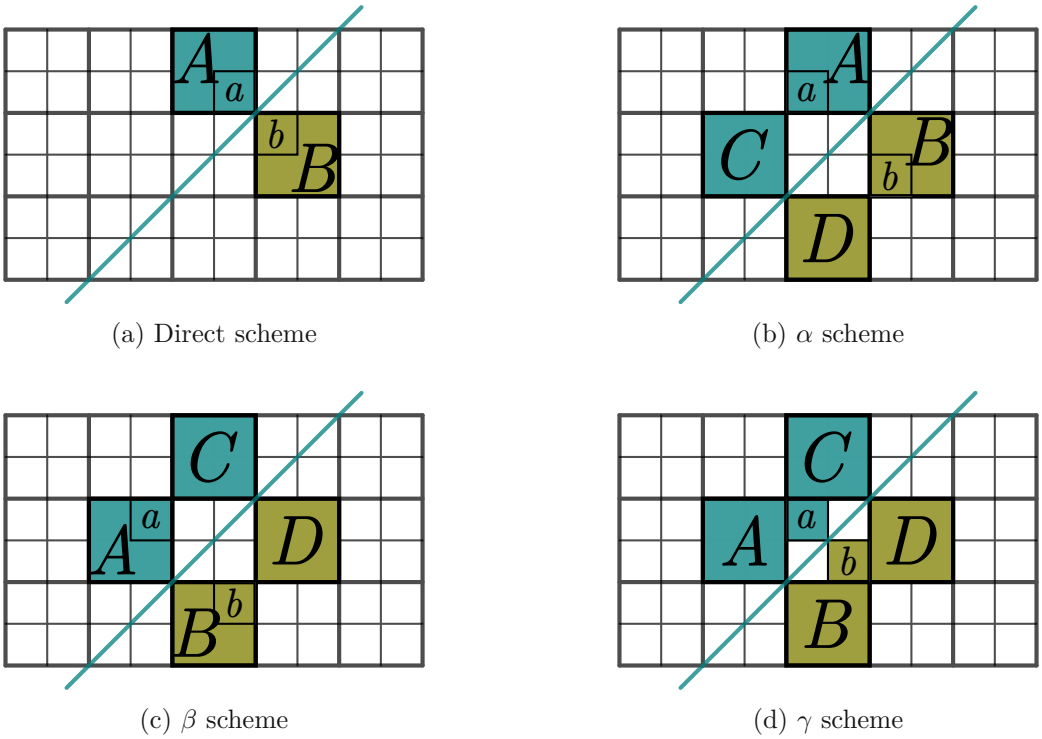
$$\overbrace{\frac{1}{4}(a_1 + a_2 + a_3 + a_4)}^{A} = \overbrace{\frac{1}{4}(b_1 + b_2 + b_3 + b_4)}^{B} + \overbrace{\frac{1}{4}(c_1 + c_2 + c_3 + c_4)}^{C} - \overbrace{\frac{1}{4}(d_1 + d_2 + d_3 + d_4)}^{D}. \tag{5.3}$$

Equation (5.3) can also be used to derive more symmetry patterns for the involved fine cells. For example, taking only every x_1 results in $a_1 = b_1 + c_1 - d_1$, which is also consistent with eq. (2.10), although these cells are not neighbors on the fine lattice.

A different symmetry, that is the basis of the next set of restrictions and studied thoroughly in [15] can be directly read off of the update eq. (2.10)

$$\phi_{x+0} - \phi_{x+1} = \phi_{x-1} - \phi_{x-0} \qquad \Longleftrightarrow \qquad \phi_{x+0} - \phi_{x-1} = \phi_{x+1} - \phi_{x-0}. \tag{5.4}$$

Diagonal differences of neighboring cells are preserved along characteristics. This is not only true for cells within a region but can also be extended to relate differently resolved cells to each other. As argued in [15], the difference $a - b$ in fig. 5.2 can be related to the difference of the equally colored regions on the coarse lattice. Furthermore, the colored region on the coarse side can be moved along the highlighted diagonal closer to the border so that parts of it overlap with the fine side. Then, instead of B and E, the cell A would enter and the missing coarse cell on the fine side can easily be added as a ghost cell. For the free theory, this would not be necessary because of the exact preservation of diagonal differences (eq. (5.4)). But in the more general settings, where this is not the case, it is important to keep the colored regions as close together as possible so that the error of using the equations derived for a simpler theory will be restricted "locally" to the border.



(a) Direct scheme (b) α scheme

(c) β scheme (d) γ scheme

Figure 5.3: The panels show the cell arrangements for each diagonal difference scheme.

Formalizing the diagonal differences leads to the set of eqs. (5.5) to (5.8). The corresponding cell arrangements are depicted in fig. 5.3. Each cell a of the four fine cells that overlap with a coarse cell has its own equation.

$$\text{Direct scheme (fig. 5.3a):} \quad a - b = \frac{1}{2}(A - B). \tag{5.5}$$

$$\alpha \text{ scheme (fig. 5.3b):} \quad a - b = \alpha(A - B) + (1 - \alpha)(C - D). \tag{5.6}$$

$$\beta \text{ scheme (fig. 5.3c):} \quad a - b = \beta(A - B) + (1 - \beta)(C - D). \tag{5.7}$$

$$\gamma \text{ scheme (fig. 5.3d):} \quad a - b = \frac{1}{2}\gamma(A - B) + \frac{1}{2}(1 - \gamma)(C - D). \tag{5.8}$$

Equations (5.6) to (5.8) introduce the weighting parameters α, β and γ, all $\in [0, 1]$, that balance the contributions of the two differences of coarse cells. In the free case, the diagonal differences of the coarse cells that are multiplied with these factors are identical such that the weights cancel out. Again, the weighting parameters come into play when considering more complex theories such that they no longer cancel. A manual process for determining the values of the weights based on interpreting the quality of the resulting time evolution is persued in [15].

An interesting aspect of eqs. (5.5) to (5.8) is that the contributions of the coarse cells carry different factors even before considering the weighting parameters. In fact, these factors are different from how coarser resolutions are handled in the norm eq. (5.1) and the averaging eq. (5.3), where the values of the coarse cells are weighted with the number of fine cells they replace. The systematic process for how one gets to the factors in eqs. (5.5) to (5.8) is presented in [15]. In short, one fixes the desired outcome of unit pulses with the width of a single cell based on the propagation without any borders. Then one tries to reproduce the same outcome in the presence of the borders by combining the previously established diagonal difference equations, including conservation of the norm eq. (5.1) and averaging, into refinement schemes.

5.1 Diagonal fine-to-coarse border (DFTC)

The scheme for DFTC is tabulated on page 45. Not all cells that carry labels in fig. 5.4 have their own equations. Typically, these are cells that are computed by the default update stencil of the solver, or whose values are known from the last cycle. In the listed equations, the default update stencil eq. (2.10) is not written out explicitly but shortened to

$$\phi_{x+0} = \Gamma(\phi_{x-1}, \phi_x, \phi_{x+1}, \phi_{x-0}), \tag{5.9}$$

with the cell dependencies in parentheses.

Each equation is annotated with a timestamp of the form $t = +\text{step}:\text{substep}$ that is used for sorting the order of evaluation. Working out the timestamps for each refinement equation is a prerequisite for their implementation in the solver. They follow from the order in which the cells need to be processed so that the standard working set, which consists of the values of the previous and current time slices that are currently saved in memory, stays a valid reference. The same treatment is also required for any additional ghost cells. All cells below the horizontal red line in fig. 5.4 are past cells, the ones above are future cells. The full refinement sequence is tabulated in fig. B.1a in appendix B, where the time annotations are put into context. Furthermore, appendix C tabulates the results for a single-cell unit pulse.

First, when focusing on the fine side, only the cell a cannot be calculated with the update stencil. Of its dependencies, the lower-left neighbor in fig. 5.4a is missing. The solution from [15] is given in eq. (5.12a) and depends on the exact preservation of diagonal differences in the free case. However, by using the ghost cell x, highlighted in fig. 5.4c, it is possible to compute all fine cells of this border by means of the update stencil in eqs. (5.10e) and (5.12b).

On the coarse side, there are two cells, A and B in fig. 5.4a, with missing dependencies. These can be supplemented as the four ghost cells E, F, G and H highlighted in fig. 5.4b. Their values can be obtained as the averages in eqs. (5.10a) to (5.10d) of the four fine cells they each overlap with. Note that G and H have to be saved from the last cycle, as otherwise, the values of the required fine cells would no longer be available.

5.2 Diagonal coarse-to-fine border (DCTF)

For the CTF transition on page 46, all coarse cells can be computed using the update stencil Γ. On the fine side all labeled, future cells have missing dependencies for the update stencil. However, they can all be computed using the diagonal difference eqs. (5.5) to (5.8). The coarse cells that are missing for the equations can be provided as the highlighted ghost cells in fig. 5.5b. All of these coarse cells can again be computed using the update stencil. In total, each time slice carries three ghost cells where only six of them are saved concurrently at any time. In the example fig. 5.5 only H, I and L still need to be computed. L can be saved onto P and is used for c, d and e, together with N. Then H and I can be saved onto M and N respectively and are used for the last two fine cells. The full refinement sequence is tabulated in fig. B.1b in appendix B. Although the refinement equations require access to cells from three, distinct time slices (i.e. slices containing H, K, O for a), this does not conflict with the standard working data set. The additional cells that are required from the next time slice are ghost cells and are managed separately.

Diagonal fine-to-coarse border (DFTC)

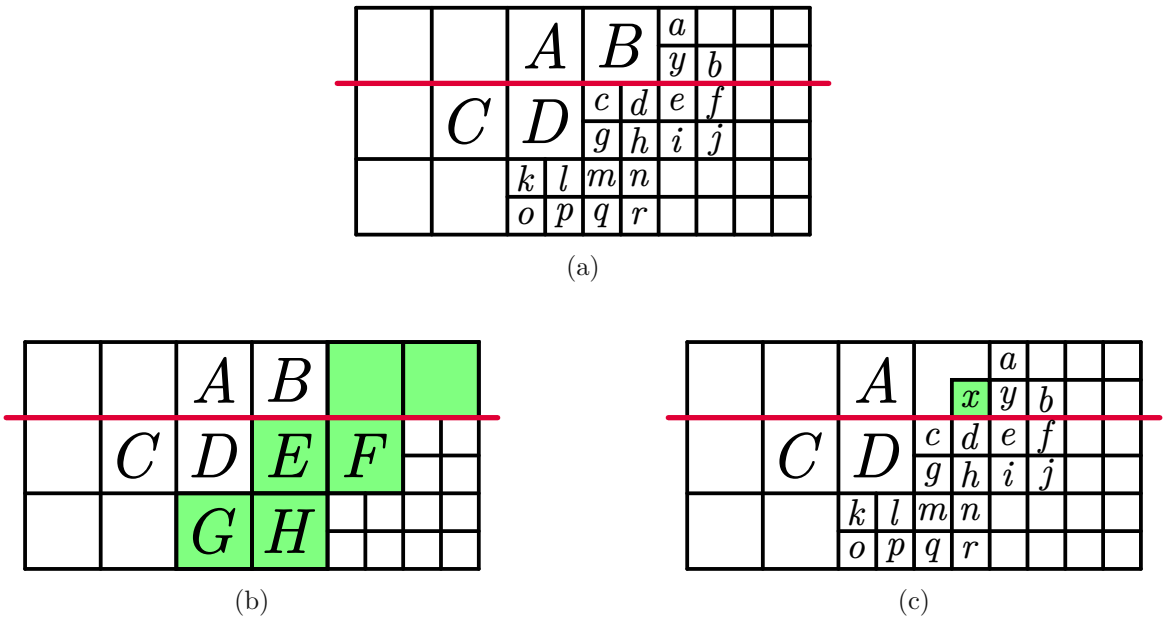


Figure 5.4: Diagonal fine-to-coarse border. Cells highlighted in green are ghost cells. Cells below the red line are past cells with existing values. Cells above the red line are future cells to be calculated.

Ghost cells:

$t = +0{:}0$

$$G = \frac{1}{4}(k + l + o + p) \tag{5.10a}$$

$$H = \frac{1}{4}(m + n + q + r) \tag{5.10b}$$

$t = +1{:}1$

$$E = \frac{1}{4}(c + d + g + h) \tag{5.10c}$$

$$F = \frac{1}{4}(e + f + i + j) \tag{5.10d}$$

$t = +1{:}2$

$$x = \Gamma(c, d, e, h) \tag{5.10e}$$

Coarse cells:

$t = +1{:}2$

$$A = \Gamma(C, D, E, G) \tag{5.11a}$$

$$B = \Gamma(D, E, F, H) \tag{5.11b}$$

Fine cells:

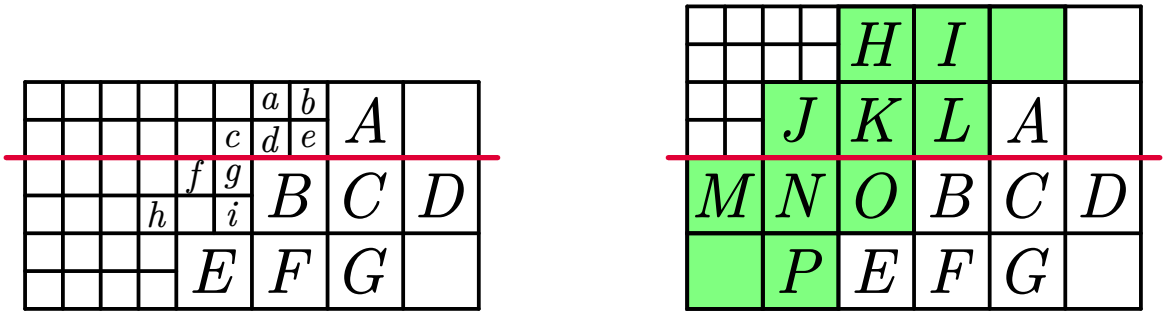
$t = +2{:}1$

$$a = b + c - h \tag{5.12a}$$

or

$$a = \Gamma(x, y, b, e) \tag{5.12b}$$

Diagonal coarse-to-fine border (DCTF)



(a) (b)

Figure 5.5: Diagonal coarse-to-fine border.

Ghost cells:

$t = +0{:}0$

$$J = \Gamma(M, N, O, P) \tag{5.13a}$$
$$K = \Gamma(N, O, B, E) \tag{5.13b}$$

$t = +1{:}1$

$$L = \Gamma(O, B, C, F) \tag{5.13c}$$

$t = +2{:}1$

$$H = \Gamma(J, K, L, O) \tag{5.13d}$$
$$I = \Gamma(K, L, A, B) \tag{5.13e}$$

Fine cells:

$t = +1{:}2$

$$c = h + \alpha(K - N) + (1 - \alpha)(B - E) \tag{5.14a}$$
$$d = g + \frac{1}{2}(L - O) \tag{5.14b}$$
$$e = i + \alpha(L - O) + (1 - \alpha)(C - F) \tag{5.14c}$$

$t = +2{:}2$

$$a = f + \beta(L - O) + (1 - \beta)(H - J) \tag{5.14d}$$
$$b = d + \frac{\gamma}{2}(A - B) + \frac{1 - \gamma}{2}(I - K) \tag{5.14e}$$

5.3 Straight border (SFTC & SCTF)

The situation is more complicated for straight borders because neither the coarse nor the fine cells at the border be calculated using the update stencil Γ. As illustrated in fig. 5.6a on page 48 there are now three cells, A, a and e with missing dependencies for the update stencil. As before, introducing coarse ghost cells provides the missing cell for A and the cells for the refinement eqs. (5.5) to (5.8) for a and e. Still, these ghost cells do not solve the problem for the coarse side because they effectively shift the border to the right as depicted in fig. 5.6b. Ghost cells that share the same x position as E (F, H, $J, \ldots$) can be computed using the update stencil, but not their right neighbors at the same x position as G ($I, \ldots$). Instead, one can fall back to averaging the fine cells that overlap with G and those ghost cells, as the fine cells in that area are given by the update stencil. The only problem is that G is a future cell and thus depends on future fine cells c, d, g and h. At the time G is needed, however, some of these cells (c, d) are not yet computed.

Rather than skipping refinement and fixing the border cells later in time when all required fine cells are available, the missing fine cells are calculated as ghost cells. This ensures that time evolution steps are consistently calculating all past values (i.e. below the red line). If it were different, depending on the time step some past cells might be still missing values. On the downside, these calculations are redundant in the sense that they will be repeated as part of the default solver and also add to the memory requirement with more ghost cells.

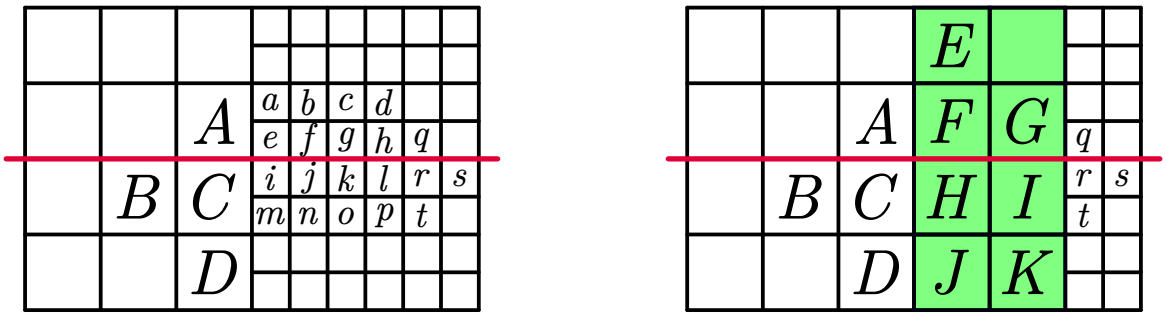
As it turns out, the refinement scheme in eqs. (5.15) to (5.17) for the straight border that only uses the refinement eqs. (5.5) to (5.8) produces unphysical reflections that we investigate later in section 6.2.1. The solution presented in [15] is a delicate combination of refinement equations and averaging. Only a is calculated using eq. (5.8). Then, the average $\frac{1}{4}(a + b + e + f) = F$ is solved for e. The future, unknown value of b is replaced by the diagonal difference relations that hold in the free case $b - e = g - j$ and f is computed using the update stencil.

The changes given in eqs. (5.18) are based on this solution. Comparing eq. (5.18c) and eq. (5.17b) allows to identify common factors used for a and e. Written differently, $e = 2F - \frac{1}{2}(a + f + g - j)$ where b is already substituted. Calculating b with the update stencil in the substep after e as part of the default solver could lead to inconsistencies. Depending on the EOM, the update stencil can differ from the diagonal difference relation used for b in the previous steps. Then, the average over a, b, e and f could differ from F. Still, using the update stencil can provide better values for b because the stencil solves the discretized EOM of the theory and the diagonal differences are specific to the free case.

Working out the dependencies of each equation and the order in which they need to be calculated exposes a problem. The scheme, as it is given in eqs. (5.18), depends on E and G for the $t = +1\,\mathrm{X}$ time steps already. At that time, none of the future, fine cells required for G are available so that the c, d, f, g, h and q cells in fig. 5.6a need to be redundantly introduced as ghost cells. In total, this results in ten ghost cells. Only four coarse ghost cells are saved at the same time and overwrite each other similar to the diagonal border refinement schemes. The remaining are all redundant, fine ghost cells.

The flow graph in fig. B.2b in appendix B follows a slightly adapted scheme. The number of ghost cells can be reduced to six by neglecting the term proportional to $1 - \gamma$ in eq. (5.18c), effectively setting $\gamma = 1$. This gets rid of the problematic dependencies on E and G and only two fine ghost cells for f and g, which are later reused for c and d are needed. Now, this value for e can be treated as temporary if it is necessary to read out the calculated values before $t = +2\,\mathrm{X}$. Still, e needs to be corrected to the full eq. (5.18c) at a later refinement step. If the correction takes place during the same refinement cycle at $t = +2\,\mathrm{X}$ and therefore before the cells $b, c, d, \ldots$ are calculated by the default solver, the temporary value for e never propagates. The optimal time for the correction is at the last step of the cycle where all cells required by

Straight border (SFTC & SCTF)



(a) (b)

Figure 5.6: Straight border.

Variant reusing eqs. (5.10) to (5.14) for diagonal borders:

Ghost cells:

$t = +0{:}0$

$$I = \frac{1}{4}(k + l + o + p) \tag{5.15a}$$

$$F = \Gamma(C, H, I, J) \tag{5.15b}$$

$t = +2{:}1$

$$c = \Gamma(f, g, h, k) \tag{5.15c}$$

$$d = \Gamma(g, h, q, l) \tag{5.15d}$$

$t = +2{:}2$

$$G = \frac{1}{4}(c + d + g + h) \tag{5.15e}$$

$$E = \Gamma(A, F, G, H) \tag{5.15f}$$

Coarse cells:

$t = +1{:}1$

$$A = \Gamma(B, C, H, D) \tag{5.16}$$

Fine cells:

$t = +1{:}1$

$$e = o + \alpha(F - I) + (1 - \alpha)(C - J) \tag{5.17a}$$

$t = +2{:}3$

$$a = f + \frac{\gamma}{2}(A - H) + \frac{1 - \gamma}{2}(E - G) \tag{5.17b}$$

Variant with dedicated, local equations (e from eq. (5.17a) corrected with F):

$t = +1{:}1$

$$f = \Gamma(i, j, k, n) \tag{5.18a}$$

$$g = \Gamma(j, k, l, o) \tag{5.18b}$$

$t = +1{:}2$

$$e = 2F - \frac{\gamma}{4}(A - H) - \frac{1 - \gamma}{4}(E - G) + \frac{1}{2}(j - g) - f \tag{5.18c}$$

Variant with dedicated, local equations (a from eq. (5.17b) corrected with F):

$t = +2{:}1$

$$b = \Gamma(e, f, h, j) \tag{5.19a}$$

$t = +2{:}2$

$$a = 4F - b - e - f \tag{5.19b}$$

Variant with dedicated, local equations (G average eq. (5.15e) substituted):

$t = +2{:}2$

$$G = H - K + \frac{1}{2}(r + s) + \frac{1}{4}(t - q) \tag{5.20}$$

eq. (5.18c) are available. It is also no longer necessary to substitute eq. (5.17b) for the value of a and instead, a can be used directly. As a result, the following changes to eqs. (5.15) to (5.18) arise

$$t = +1{:}2 \qquad e = 2F - \frac{1}{4}(A - H) + \frac{1}{2}(j - g) - f, \tag{5.21a}$$

$$t = +2{:}4 \qquad e = 2F - \frac{1}{2}(a + f + g - j). \tag{5.21b}$$

The next variant of eqs. (5.19) applies the same strategy as [15] but with the change that now e is still calculated via the refinement eq. (5.17a) and a is extracted from the average for F resulting in eq. (5.19b). The latter makes it obvious that F is the average of the four fine cells. The benefit of this variant is that there are much fewer redundant ghost cells required. In fact, only b remains, which now is also consistent with the average for F in any case. The number of coarse ghost cells is still the same, but they are used at different time steps.

The derivation of eqs. (5.18) in [15] uses the averaging relation eq. (5.3) of fine and coarse cells to calculate the coarse border cells as a starting point. Therefore, the correction terms appear in the formulas for the fine border cells. A new refinement scheme can be derived if instead, the eqs. (5.17a) and (5.17b) for diagonal borders are used as the starting point and the averaging relation for coarse border cells is corrected.

In analogy to [15], first, we tabulate the desired propagation of single-cell wide unit pulses at the straight border. The configurations in appendix D put the result of eqs. (5.10) to (5.14) at diagonal borders onto SCTF transitions and force clean, coarse cells without reflections. For pulses traveling from the fine to the coarse side, the averages over fine cells are reused. The panels in appendix D also contain values for the coarse ghost cells.

Next, we make the Ansatz for A

$$A = A_1 B + A_2 C + A_3 D + A_4 i + A_5 j + A_6 m + A_7 n + A_8 e, \tag{5.22}$$

that contains the coarse and fine cells that normally would affect the result and the additional cell e. The system of linear equations for the coefficients $A_1, \ldots, A_8$ that follows from appendix D by inserting the numerical values of the cells into eq. (5.22) is:

$$0 = 2A_3 + A_4 + 2A_5 + 2A_6 + A_7 \quad (5.23\text{: i})$$
$$0 = 2A_2 + A_4 + 2A_8 \quad (5.23\text{: ii})$$
$$2 = 2A_1 \quad (5.23\text{: iii})$$
$$1 = 4A_5 + 4A_8 \quad (5.23\text{: iv})$$
$$1 = 2A_4 + 2A_7 \quad (5.23\text{: v})$$
$$1 = A_2 + 4A_6 \quad (5.23\text{: vi})$$
$$0 = A_2 \quad (5.23\text{: vii})$$
$$0 = A_1 + A_2 + A_3 \quad (5.23\text{: viii})$$
$$0 = A_1 + A_3 \quad (5.23\text{: ix})$$

There are nine equations for only eight parameters, but eqs. vii) to ix) are not linearly independent so that only eight conditions can be derived from appendix D. This results in

$$A = B - D + \frac{1}{2}(i + j) + \frac{1}{4}(m - e). \tag{5.24}$$

Equation (5.24) strongly resembles the update stencil eq. (5.9) for the free case. The diagonal difference of $B - D$ is carried over to $A - H$, but H is no longer the average of i, j, m and n. Instead, H has to be identified as the contribution of the fine cells appearing in eq. (5.24). The relation of this correction to the average for H is explored in section 6.2.1.

Implementing eq. (5.24) as a new refinement scheme requires some thinking. Simply replacing eqs. (5.16) by eq. (5.24) does not work, because effectively, the FTC transition occurs at the x position of the ghost cells $G, I, K, \ldots$. There, the average of fine cells would still be used,

leading to inconsistencies of coarse non-ghost and ghost cells. The solution is to use eq. (5.24) only for G instead of A and keeping the default update stencil for all other coarse (ghost) cells. This leads to the variant in eqs. (5.20) that only changes eq. (5.15e) compared to the variant that reuses the equations for diagonal borders. The eqs. (5.17a) and (5.17b) for the fine border cells e and a are used without correction terms.

Inspecting eqs. (5.20) in the context of the flow graph fig. B.3b reveals additional dependencies on past cells. The coarse ghost cell K can be easily supplied by keeping the value of I one more refinement step, instead of saving G onto I. Similarly, the fine cell t is required at a time step where it can no longer be fetched from the current working set. It also has to be saved to a ghost cell in one of the $t = +1{:}\mathrm{X}$ steps, before it is deleted in favor of the time slice containing $q, h, g \ldots$. As a result, the refinement scheme based on eqs. (5.20) requires six ghost cells, the same as the previously discussed schemes for straight borders.

5.4 Fine-to-coarse cusp (CFTC)

The simplest cusp type is the fine-to-coarse cusp. The scheme on page 51 is only applicable to the next two fine update steps that include the cells A and B and the fine border cells a (a') and b (b') in fig. 5.7a. Afterwards, this cusp changes into FTC diagonal borders on each side.

All fine cells can be calculated using the update stencil Γ by introducing the two ghost cells z and z'. The dependencies of a and a' are then fulfilled. The situation changes for the coarse cells, though. One has to remember that only two past time slices are accessible, but because there are only fine cells in the past, G and H in fig. 5.7c are not available and cannot be calculated as averages of fine cells. This is the meaning of their brown-colored highlighting. With the available fine cells, it is only possible to calculate the values for the coarse ghost cells C, D, E and F. However, these ghost cells do not help when computing A and B without the other, missing, coarse cells. Therefore, it is necessary to continue the fine lattice two more time steps by introducing the fine ghost cells highlighted in fig. 5.7d. These ghost cells can then be averaged for A and B accordingly. Still, it is necessary to calculate and store the other coarse ghost cells highlighted in green as well. Even though they are not used for this refinement scheme, the diagonal borders that emerge from the cusp depend on these values being available.

5.5 Coarse-to-fine cusp (CCTF)

As before, going from coarse cells to a region with only fine cells is more difficult. The scheme on pages 52 and 53 includes two variants. The first variant recites the dedicated equations from [15]. For their derivations, the procedure is again to fix how narrow unit pulses should propagate and then reconstruct the matching refinement equations.

The second variant just extends the diagonal borders on each side of the cusp by using the ghost cells of the other border's refinement scheme as the cells on the coarse side. The situation is visualized in fig. 5.8c. Then, the same eqs. (5.13) to (5.14) can be reused. As it is for the coarse-to-fine diagonal border, three ghost cells per slice and side (H, I, L and H', I', L') are needed.

When comparing the dedicated eqs. (5.30) to (5.31) to the diagonal border eqs. (5.32) to (5.33), many differences catch attention. None of the equations match and only d and e (and d', e') share common terms. Equation (5.30b) for d modifies the direct scheme eq. (5.5) and adds another weight λ and the counter term with coarse cells from the past. Equation (5.30c) for e reuses the α scheme but with a different weight κ. Additionally, the contribution of f is halved and more cells enter. As a result, only the four ghost cells highlighted in fig. 5.8b instead of fig. 5.8c are required.

Fine-to-coarse cusp (CFTC)

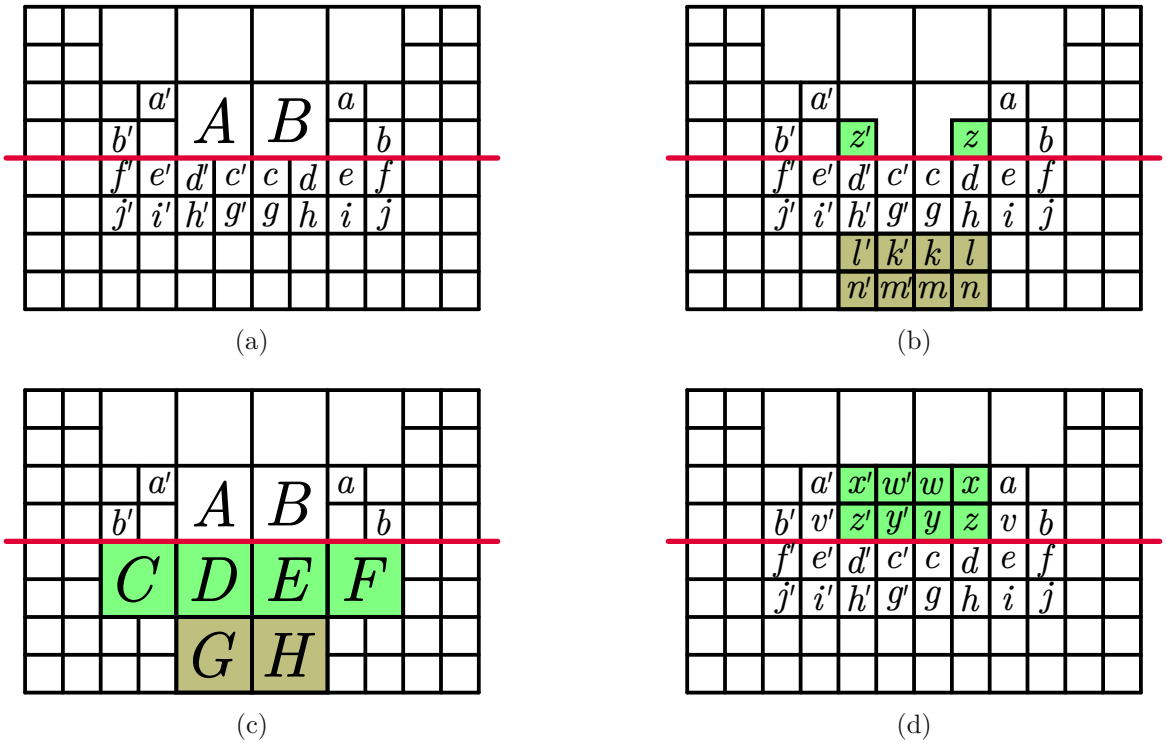


Figure 5.7: Fine-to-coarse cusp. Cells highlighted in brown are unavailable.

Ghost cells:

$t = +1{:}1$

$$C = \frac{1}{4}(e' + f' + i' + j') \tag{5.25a}$$
$$D = \frac{1}{4}(c' + d' + g' + h') \tag{5.25b}$$
$$E = \frac{1}{4}(c + d + g + h) \tag{5.25c}$$
$$F = \frac{1}{4}(e + f + i + j) \tag{5.25d}$$

$t = +1{:}2$

$$y = \Gamma(c', c, d, g) \tag{5.25e}$$
$$y' = \Gamma(c, c', d', g') \tag{5.25f}$$
$$z = \Gamma(c, d, e, h) \tag{5.25g}$$
$$z' = \Gamma(c', d', e', h') \tag{5.25h}$$

$t = +1{:}3$

$$w = \Gamma(y', y, z, c) \tag{5.25i}$$
$$w' = \Gamma(y, y', z', c') \tag{5.25j}$$
$$x = \Gamma(y, z, v, d) \tag{5.25k}$$
$$x' = \Gamma(y', z', v', d') \tag{5.25l}$$

Coarse cells:

$t = +1{:}4$

$$A = \Gamma(C, D, E, G) \tag{5.26a}$$
$$B = \Gamma(D, E, F, H) \tag{5.26b}$$

or

$$A = \frac{1}{4}(w' + x' + y' + z') \tag{5.26c}$$
$$B = \frac{1}{4}(w + x + y + z) \tag{5.26d}$$

Fine cells:

$t = +2{:}1$

$$a = b + c - h \tag{5.27a}$$
$$a' = b' + c' - h' \tag{5.27b}$$

or

$$a = \Gamma(z, v, b, e) \tag{5.27c}$$
$$a' = \Gamma(z', v', b', e') \tag{5.27d}$$

Coarse-to-fine cusp (CCTF)

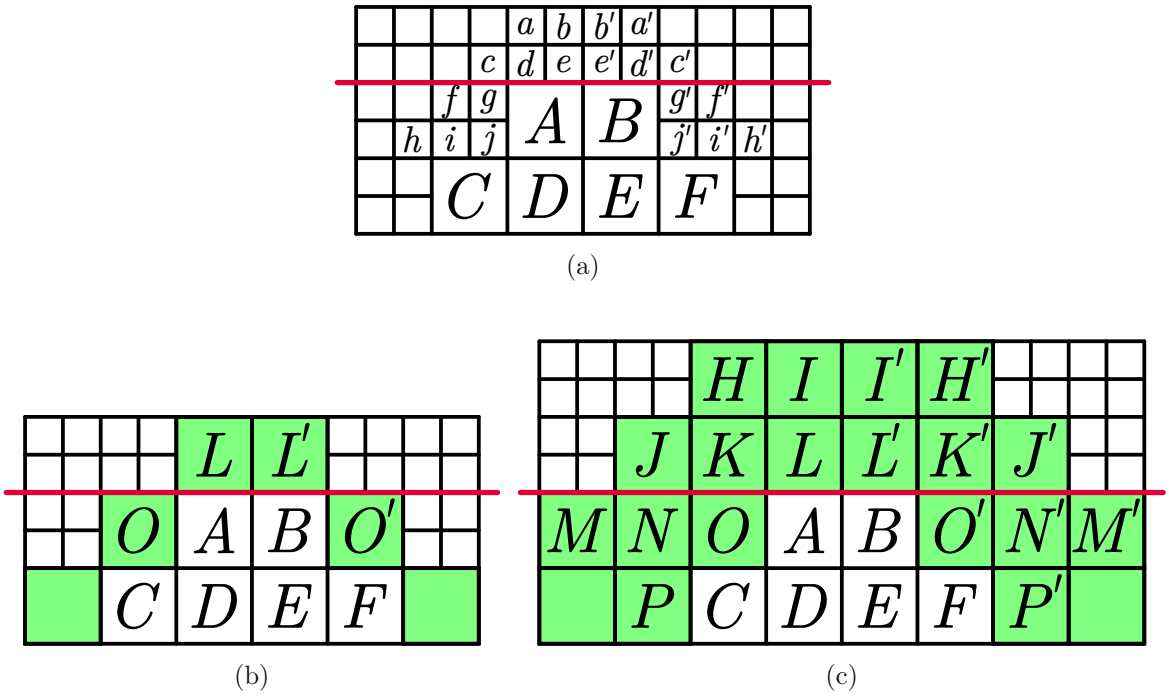


Figure 5.8: Coarse-to-fine cusp.

Ghost cells:

$t = +1{:}1$

$$L = \Gamma(O, A, B, D) \tag{5.28a}$$

$t = +2{:}1$

$$H = \Gamma(J, K, L, O) \tag{5.28b}$$

$$I = \Gamma(K, L, L', A) \tag{5.28c}$$

$t = +1{:}1$

$$L' = \Gamma(A, B, O', E) \tag{5.29a}$$

$t = +2{:}1$

$$H' = \Gamma(J', K', L', O') \tag{5.29b}$$

$$I' = \Gamma(K', L', L, B) \tag{5.29c}$$

Coarse-to-fine cusp (CCTF)

Variant using dedicated equations:

Fine cells

$t = +1{:}2$

$$c = h + 2(g - i) \tag{5.30a}$$
$$d = g + \frac{\lambda}{2}(B - D) + \frac{1-\lambda}{2}(L - O) \tag{5.30b}$$
$$e = \frac{1}{2}(f - h + j + C) + \kappa(B - D) + (1-\kappa)(L - O) \tag{5.30c}$$

$t = +2{:}1$

$$a = f + 2(d - g) \tag{5.30d}$$
$$b = e' + g + \frac{1}{2}(c - f) - A \tag{5.30e}$$

$t = +1{:}2$

$$c' = h' + 2(g' - i') \tag{5.31a}$$
$$d' = g' + \frac{\lambda}{2}(A - E) + \frac{1-\lambda}{2}(L' - O') \tag{5.31b}$$
$$e' = \frac{1}{2}(f' - h' + j' + F) + \kappa(A - E) + (1-\kappa)(L' - O') \tag{5.31c}$$

$t = +2{:}1$

$$a' = f' + 2(d' - g') \tag{5.31d}$$
$$b' = e + g' + \frac{1}{2}(c' - f') - B \tag{5.31e}$$

Variant reusing eqs. (5.13) to (5.14) for diagonal borders:

Fine cells

$t = +1{:}2$

$$c = h + \alpha(K - N) + (1-\alpha)(A - C) \tag{5.32a}$$
$$d = g + \frac{1}{2}(L - O) \tag{5.32b}$$
$$e = j + \alpha(L - O) + (1-\alpha)(B - D) \tag{5.32c}$$

$t = +2{:}2$

$$a = f + \beta(L - O) + (1-\beta)(H - J) \tag{5.32d}$$
$$b = d + \frac{\gamma}{2}(L' - A) + \frac{1-\gamma}{2}(I - K) \tag{5.32e}$$

$t = +1{:}2$

$$c' = h' + \alpha(K' - N') + (1-\alpha)(B - F) \tag{5.33a}$$
$$d' = g' + \frac{1}{2}(L' - O') \tag{5.33b}$$
$$e' = j' + \alpha(L' - O') + (1-\alpha)(A - E) \tag{5.33c}$$

$t = +2{:}2$

$$a' = f' + \beta(L' - O') + (1-\beta)(H' - J') \tag{5.33d}$$
$$b' = d' + \frac{\gamma}{2}(L - B) + \frac{1-\gamma}{2}(I' - K') \tag{5.33e}$$

5.6 Straight border cusps

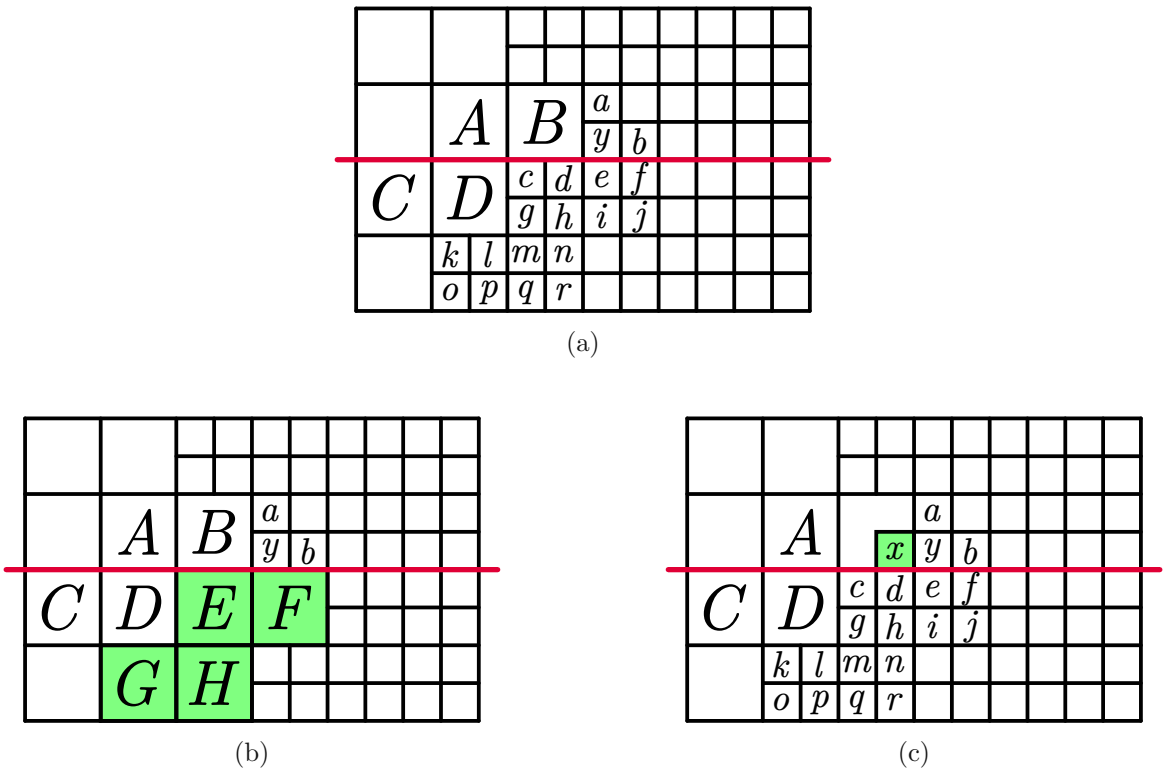
Previously, in section 3.1, where all border types on collision lattices are identified, there is no mention of straight border cusps. Only straight borders are listed in fig. 3.4, whose refinement equations are discussed in section 5.3. The reason is that straight border cusps are combinations of diagonal borders and straight borders. After having introduced each of these types separately, we discuss how they work together.

Straight border cusps require some additional steps to prepare ghost cells after the border changes. These steps depend on the exact lattice geometry. There are two possible ways to construct straight border cusps. On the one hand, a fine-to-coarse diagonal border can directly change into a coarse-to-fine diagonal border (or vice versa). This case produces a straight, single-cell cusp. On the other hand, the FTC border can first change into a straight border, which then changes to the coarse-to-fine diagonal border. If the last change happens at the earliest possible time step, the result is a straight, double-cell cusp. Otherwise, the cusp accumulates one cell for each additional coarse time step. In this case, the straight border is extended in time. This is not different from the straight double-cell cusp in terms of refinement equations, because it does not matter how long the straight border extends before changing and completing the cusp.

5.6.1 Straight single-cell cusp

The scheme on pages 55 and 56 is separated into the situations before and after the cusp. The step before the cusp (fig. 5.9) reuses the eqs. (5.10) to (5.12) from FTC diagonal borders unchanged. The intention for the step after the cusp (fig. 5.10) is to reuse eqs. (5.13) to (5.14) for CTF diagonal borders. However, these equations depend on the highlighted ghost cells in fig. 5.10b, which cannot all be calculated by these equations. In particular, P would be required, but can no longer be calculated because the fine cells necessary for averaging are too far in the past and discarded already. F and E were calculated in the last step as part of the previous border's refinement. The solution is to disregard the brown ghost cells and calculate the turquoise cells via averages of fine cells. For J it is necessary to introduce a couple of future fine ghost cells, similar to the straight border. The values for the remaining future coarse ghost cells are then obtained from the default update stencil. In total, the straight single-cell cusp refinement scheme requires eight ghost cells. The resulting eqs. (5.37) to (5.38) carry time steps that are reset after the step before the cusp and match the flow graph in fig. B.6 in appendix B.

Straight single-cell cusp



(a) (b) (c)

Figure 5.9: Straight single-cell cusp at the step before the cusp.

Step before the cusp shown in fig. 5.9 and same as eqs. (5.10) to (5.12):

Ghost cells:

$t = +0{:}0$

$$G = \frac{1}{4}(k + l + o + p) \tag{5.34a}$$

$$H = \frac{1}{4}(m + n + q + r) \tag{5.34b}$$

$t = +1{:}1$

$$E = \frac{1}{4}(c + d + g + h) \tag{5.34c}$$

$$F = \frac{1}{4}(e + f + i + j) \tag{5.34d}$$

$t = +1{:}2$

$$x = \Gamma(c, d, e, h) \tag{5.34e}$$

Coarse cells:

$t = +1{:}2$

$$A = \Gamma(C, D, E, G) \tag{5.35a}$$

$$B = \Gamma(D, E, F, H) \tag{5.35b}$$

Fine cells:

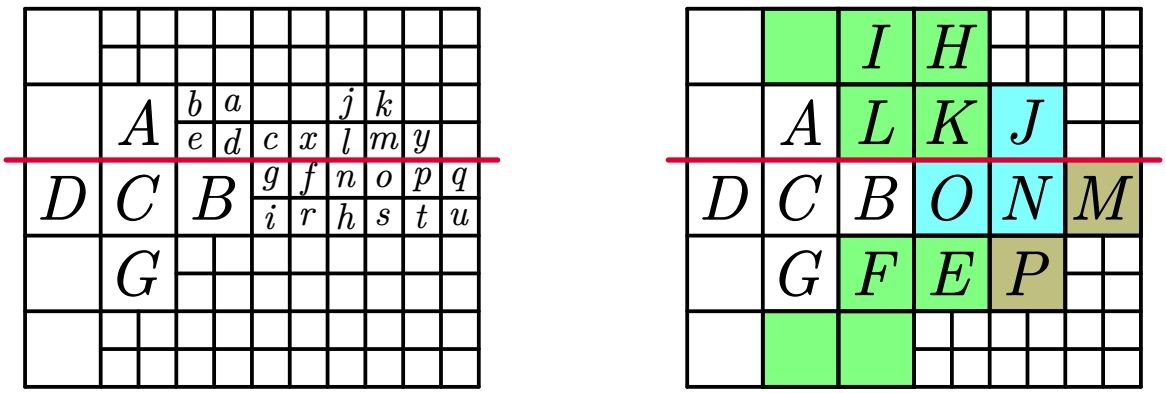
$t = +2{:}1$

$$a = b + c - h \tag{5.36a}$$

or

$$a = \Gamma(x, y, b, e) \tag{5.36b}$$

Straight single-cell cusp



(a) (b)

Figure 5.10: Straight single-cell cusp at the step after the cusp. The labels of the cells are changed compared to the previous step in fig. 5.9. Turquoise ghost cells require special treatment and brown ghost cells are unavailable.

Step after the cusp shown in fig. 5.10 and similar to eqs. (5.13) to (5.14):

Ghost cells:

$t = +1{:}1$

$$N = \frac{1}{4}(n + o + h + s) \tag{5.37a}$$
$$O = \frac{1}{4}(g + f + i + r) \tag{5.37b}$$

$t = +1{:}2$

$$K = \Gamma(N, O, B, E) \tag{5.37c}$$
$$L = \Gamma(O, B, C, F) \tag{5.37d}$$

$t = +2{:}2$

$$H = \Gamma(J, K, L, O) \tag{5.37e}$$
$$I = \Gamma(A, L, K, B) \tag{5.37f}$$
$$J = \frac{1}{4}(j + k + l + m) \tag{5.37g}$$

Fine cells:

$t = +1{:}3$

$$c = h + \alpha(K - N) + (1 - \alpha)(B - E) \tag{5.38a}$$
$$d = g + \frac{1}{2}(L - O) \tag{5.38b}$$
$$e = i + \alpha(L - O) + (1 - \alpha)(C - F) \tag{5.38c}$$

$t = +2{:}1$

$$j = \Gamma(x, l, m, n) \tag{5.38d}$$
$$k = \Gamma(l, m, y, o) \tag{5.38e}$$

$t = +2{:}3$

$$a = f + \beta(L - O) + (1 - \beta)(H - J) \tag{5.38f}$$
$$b = d + \frac{\gamma}{2}(A - B) + \frac{1 - \gamma}{2}(I - K) \tag{5.38g}$$

5.6.2 Straight double-cell cusp

Again, the scheme on pages 58 to 60 is separated into three steps for each border type. The step before the cusp (fig. 5.11) reuses eqs. (5.10) to (5.12). The step at the cusp (fig. 5.12) is based on eqs. (5.15) to (5.20). The variants are mostly the same as in section 5.3. Equations (5.45) contain all the steps for the multiple calculations of e for the variant where e is corrected with F and the equivalent of eqs. (5.19) is no longer listed.

However, there is a complication for the variant that uses the average correction for G in eqs. (5.46). The ghost cell K, that enters these equations, is not available and cannot be calculated at this step. We can solve this problem by studying the lattice configuration for this refinement step. Note that all the fine and coarse cells that influence G are either calculated via the default update stencil Γ for fine cells or via standard averaging of these fine cells. This means that at this step of the cusp refinement there cannot be CTF transitions. Therefore, we do not expect any issues with reflections at the border when we use the default average eq. (5.42f) instead. In section 6.2.1 we determine that correcting the average is necessary for pulses after CTF transitions because the default averaging leads to inconsistencies in the ghost cells. As a result, we end up with only two refinement schemes for the step at the cusp. The third scheme eqs. (5.20) only enter again when the straight border of the cusp is continued for the next time slices instead of changing to a diagonal border and forming a double-cell cusp.

For the step after the cusp (fig. 5.13) it is necessary to manage additional ghost cells so that it is possible to apply the scheme eqs. (5.13) to (5.14) for CTF diagonal borders. The comparison with fig. 5.5 reveals that the turquoise ghost cells J and M are missing. The latter can be calculated by averaging. Additionally, E and P (H and I from fig. 5.12) need to be kept available. P is used when calculating the missing cell J with the default update stencil and can be overwritten in this step and E is required for eqs. (5.48). All other ghost cells follow from the update stencil in accordance with CTF diagonal border refinement. In total, the straight double-cell cusp refinement scheme requires nine ghost cells.

The flow graph in fig. B.7 in appendix B matches the given time steps that start from the step at the cusp and depicts the variant with eqs. (5.42) to (5.44) and eqs. (5.46) for the step at the cusp. The flow graph for the other eqs. (5.45) is not included.

Straight double-cell cusp

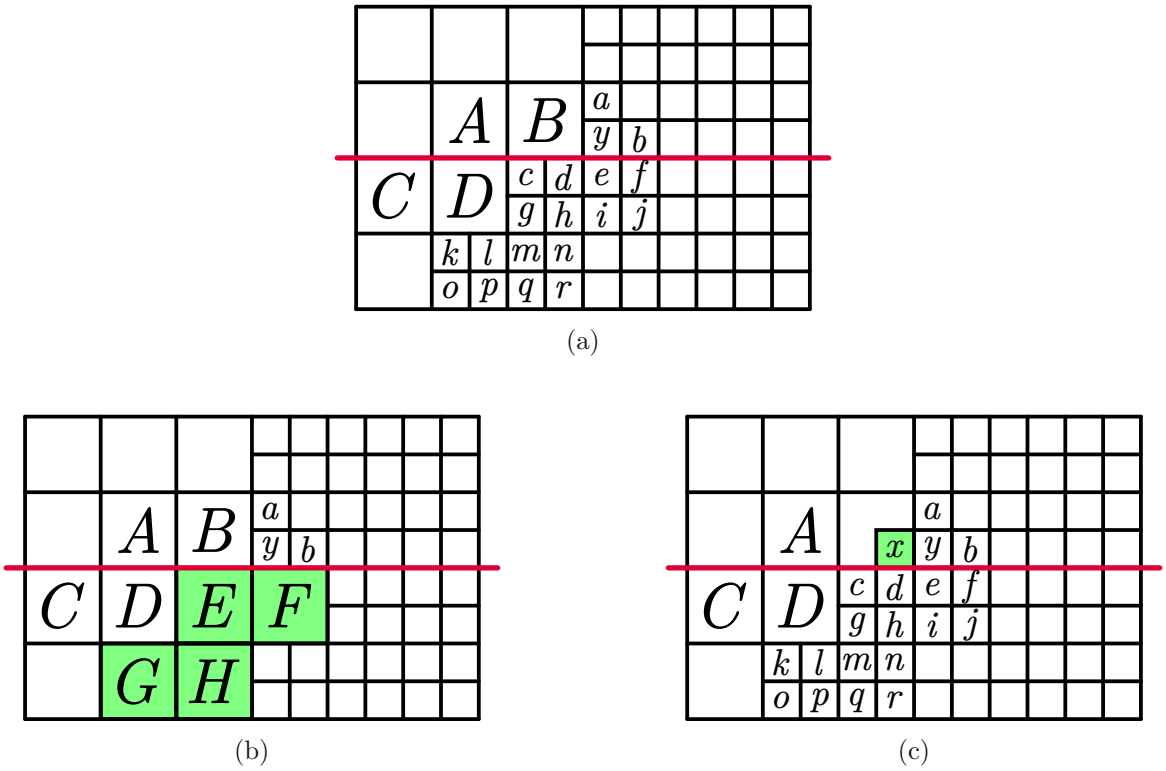


Figure 5.11: Straight double-cell cusp at the step before the cusp.

Step before the cusp shown in fig. 5.11 and same as eqs. (5.10) to (5.12):

Ghost cells:

$t = +0{:}0$

$$G = \frac{1}{4}(k + l + o + p) \tag{5.39a}$$

$$H = \frac{1}{4}(m + n + q + r) \tag{5.39b}$$

$t = +1{:}1$

$$E = \frac{1}{4}(c + d + g + h) \tag{5.39c}$$

$$F = \frac{1}{4}(e + f + i + j) \tag{5.39d}$$

$t = +1{:}2$

$$x = \Gamma(c, d, e, h) \tag{5.39e}$$

Coarse cells:

$t = +1{:}2$

$$A = \Gamma(C, D, E, G) \tag{5.40a}$$

$$B = \Gamma(D, E, F, H) \tag{5.40b}$$

Fine cells:

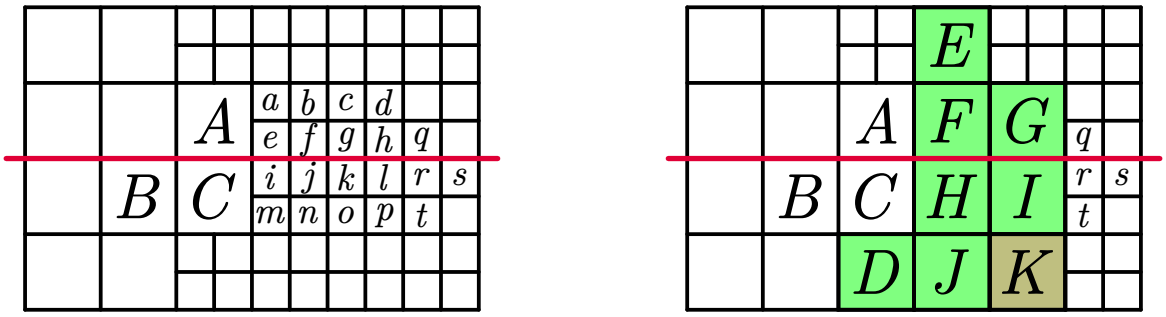
$t = +2{:}1$

$$a = b + c - h \tag{5.41a}$$

or

$$a = \Gamma(x, y, b, e) \tag{5.41b}$$

Straight double-cell cusp



(a) (b)

Figure 5.12: Straight double-cell cusp at the step at the cusp. The labels of the cells are changed compared to the previous step in fig. 5.11. The brown ghost cell is unavailable.

Step at the cusp shown in fig. 5.12 and same as eqs. (5.15) to (5.20) with renamed labels:

Variant reusing eqs. (5.10) to (5.14) for diagonal borders:

Ghost cells:

$t = +1{:}1$

$$H = \frac{1}{4}(i + j + m + n) \tag{5.42a}$$

$$I = \frac{1}{4}(k + l + o + p) \tag{5.42b}$$

$$F = \Gamma(C, H, I, J) \tag{5.42c}$$

$t = +2{:}1$

$$c = \Gamma(f, g, h, k) \tag{5.42d}$$

$$d = \Gamma(g, h, q, l) \tag{5.42e}$$

$t = +2{:}2$

$$G = \frac{1}{4}(c + d + g + h) \tag{5.42f}$$

$$E = \Gamma(A, F, G, H) \tag{5.42g}$$

Coarse cells:

$t = +1{:}2$

$$A = \Gamma(B, C, H, D) \tag{5.43}$$

Fine cells:

$t = +1{:}2$

$$e = o + \alpha(F - I) + (1 - \alpha)(C - J) \tag{5.44a}$$

$t = +2{:}3$

$$a = f + \frac{\gamma}{2}(A - H) + \frac{1 - \gamma}{2}(E - G) \tag{5.44b}$$

Variant with dedicated, local equations (e from eq. (5.44a) corrected with F):

$t = +1{:}1$

$$f = \Gamma(i, j, k, n) \tag{5.45a}$$

$$g = \Gamma(j, k, l, o) \tag{5.45b}$$

$t = +1{:}2$

$$e = 2F - \frac{1}{4}(A - H) + \frac{1}{2}(j - g) - f \tag{5.45c}$$

$t = +2{:}4$

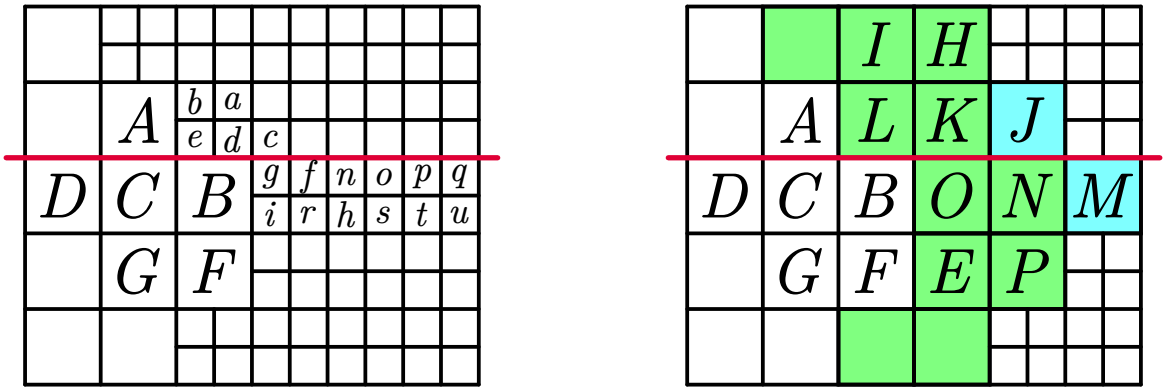
$$e = 2F - \frac{1}{2}(a + f + g - j) \tag{5.45d}$$

Variant with dedicated, local equations and average correction for G is equal to eq. (5.42f):

$t = +2{:}2$

$$G \neq H - K + \frac{1}{2}(r + s) + \frac{1}{4}(t - q) = \frac{1}{4}(c + d + g + h) \tag{5.46}$$

Straight double-cell cusp



(a) (b)

Figure 5.13: Straight double-cell cusp at the step after the cusp. The labels of the cells are changed compared to the previous step in fig. 5.12. The turquoise ghost cells require special treatment.

Step after the cusp shown in fig. 5.13 and similar to eqs. (5.13) to (5.14):

Ghost cells:

$t = +3{:}1$

$$J = \Gamma(O, N, M, P) \tag{5.47a}$$
$$M = \frac{1}{4}(p + q + t + u) \tag{5.47b}$$
$$L = \Gamma(O, B, C, F) \tag{5.47c}$$

$t = +4{:}1$

$$H = \Gamma(J, K, L, O) \tag{5.47d}$$
$$I = \Gamma(A, L, K, B) \tag{5.47e}$$

Fine cells:

$t = +3{:}2$

$$c = h + \alpha(K - N) + (1 - \alpha)(B - E) \tag{5.48a}$$
$$d = g + \frac{1}{2}(L - O) \tag{5.48b}$$
$$e = i + \alpha(L - O) + (1 - \alpha)(C - F) \tag{5.48c}$$

$t = +4{:}2$

$$a = f + \beta(L - O) + (1 - \beta)(H - J) \tag{5.48d}$$
$$b = d + \frac{\gamma}{2}(A - B) + \frac{1 - \gamma}{2}(I - K) \tag{5.48e}$$

Chapter 6

Properties of the refinement schemes

In the following, we study two important properties of the refinement schemes that are introduced in the previous chapter 5. The discussion is only applicable to these specific equations and lattices with square cells $a^0/a^1 = 1$. On the one hand, we examine the symmetry of the coarse-to-fine and fine-to-coarse schemes for each border type. Therefore, we start with the results of CTF refinement and use them as the starting values for a FTC transition at the flipped border. Of particular interest is to determine if it is possible to recover the initial conditions. Failing to do so is an indication that the modifications of CTF refinement, that deal with missing information, persist when the pulse transitions back to a coarse lattice. In this case, the new values supplied by CTF equations change the time evolution so that it is no longer consistent with the coarse lattice.

On the other hand, we verify the conservation of the norm that is calculated with eq. (5.1). As stated in [15], the norm of a single-cell wide unit pulse cannot be preserved during border refinement. We investigate this statement for general pulse profiles. Still, the conserved value of the norm is recovered when calculated for time slices after border refinement, where the values of all border cells are zero. This is always the case because the refinement schemes are designed to conserve the norm in this way.

6.1 Diagonal borders

6.1.1 Refinement direction symmetry at diagonal borders

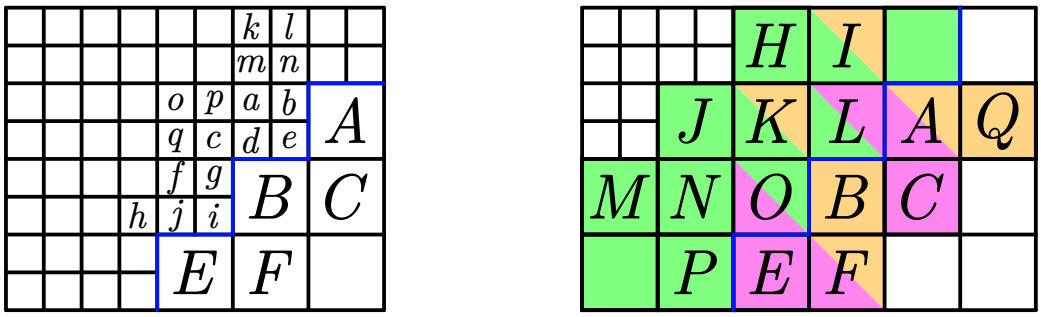
If we start with a DCTF transition in fig. 6.1 for a pulse traveling from right to left and time running upwards, we are able to calculate all labeled fine cells using eqs. (5.13) to (5.14) and the EOM (2.10). Now, we take the previously calculated values for the fine cells as given and use them for a DFTC transition, effectively reverting the border. The final result does not reproduce the initial pulse on the coarse side (fig. 6.2).

We can convince ourselves of this result by explicitly looking at the eqs. (5.10) to (5.12) and eqs. (5.13) to (5.14). For the free EOM, the equations for the DCTF transition reduce to

$$\begin{aligned} a &= f + (L - O), & d &= g + \frac{1}{2}(L - O), \\ b &= d + \frac{1}{2}(A - B), & e &= i + (L - O). \end{aligned} \tag{6.1}$$

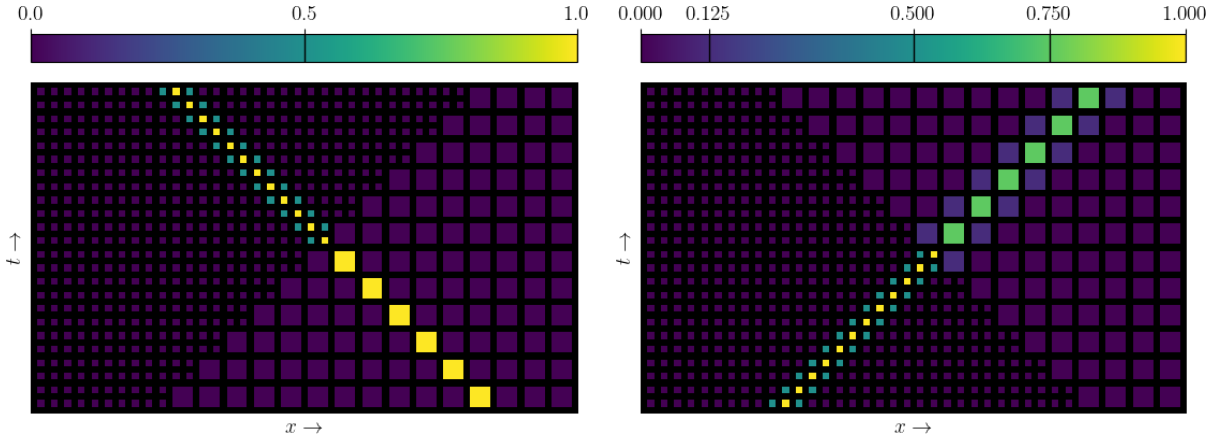
The final pulse on the coarse side will be consistent if, after changing the direction of refinement, the DFTC transition restores the values of coarse cells that depend on these fine cells. We can check for consistency by evaluating

$$C - F = L - O \stackrel{?}{=} \frac{1}{4}(a + b + d + e) - \frac{1}{4}(f + g + j + i), \tag{6.2}$$



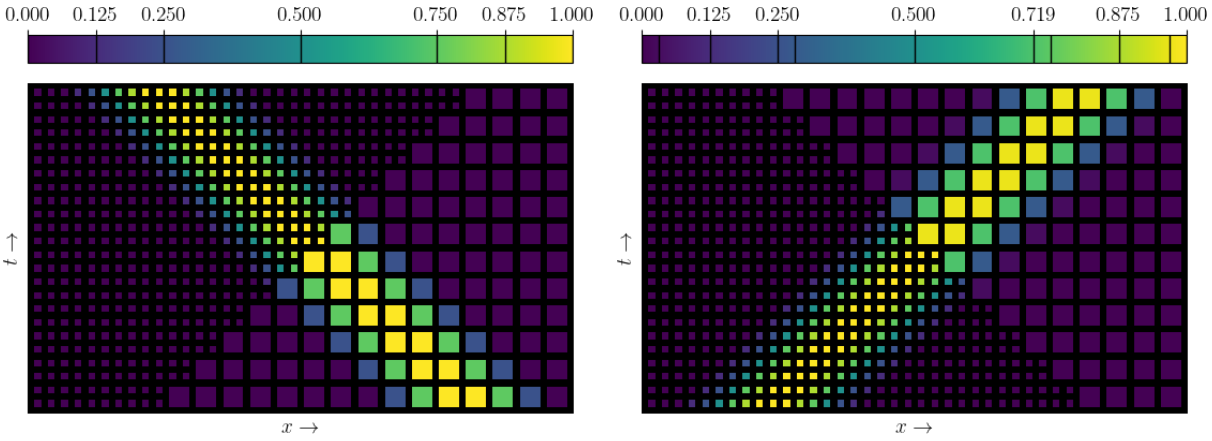
(a) (b)

Figure 6.1: Refinement direction symmetry at the diagonal border. (b): Green cells are ghost cells. For A and B to regain their values when changing the direction of refinement at the border (blue), $I - K$ has to be consistent with averages over fine cells. This condition links together all orange cells. For C and F to regain their values, $L - O$ has to be consistent with averages over fine cells. This links together all pink cells.



(a) (b)

Figure 6.2: Using the last two time slices of (a) as initial conditions after reverting time does not reproduce the initial pulse in (b).



(a) (b)

Figure 6.3: Same as fig. 6.2, but this special pulse shape is advantageous for recovering the inital conditions.

where we equate the coarse ghost cells L and O to averages of fine cells the same way as they enter eqs. (5.10) to (5.12) for DFTC. Inserting eqs. (6.1) and using $g - j = \frac{1}{2}(B - E)$ results in

$$C - F = \frac{1}{2}(A - E). \tag{6.3}$$

If the value of F is to be reproduced with eq. (6.2), this condition needs to be fulfilled. However, eq. (6.3) is not compatible with the EOM. The coarse cells that enter this relation never couple through the EOM. Thus, eq. (6.3) appears as an additional constraint for the theory that cannot be enforced in general.

Consider the single-cell unit pulse of fig. 6.2a that is placed such that the pulse crosses the cell C in fig. 6.1. Other than C, the values of the cells in eq. (6.3) are 0 and the equation does not hold. Similarly, in fig. 6.2b only C and A have values other than 0 but require one of $F \neq 0$ or $E \neq 0$ for the equation to hold. Neither C nor B or A, the only non-zero coarse cells after DFTC, are recovered.

This raises the question if there can exist a specific pulse profile that is restored when changing the direction of refinement with the schemes for diagonal borders. Equation (6.3), however, is not enough to determine the shape of such a pulse. It overlooks that I and K also need to be consistent with averages over fine cells because they enter the same way in eqs. (5.10) to (5.12) for DFTC when calculating B. With eq. (6.3) we can enforce that the cells on the same diagonal as P, O and L are pairwise consistent and reproduce F, C and Q. To do the same for the diagonal of N, K and I and thus covering the border's region without gaps, an analogous condition is needed.

We can repeat the same exercise as before, writing

$$A - B = I - K \overset{?}{=} \frac{1}{4}(k + l + m + n) - \frac{1}{4}(o + p + q + c). \tag{6.4}$$

Then, we use the default update stencil to relate the fine cells to the cells in eqs. (6.1) and also make use of the last, unused equation of eqs. (5.13) to (5.14): $n = c + (A - B)$. The result is

$$A - B = \frac{1}{2}(Q - F). \tag{6.5}$$

Again, eq. (6.5) appears as an additional constraint to the EOM. It is interesting to see that the single-cell unit pulse of fig. 6.2 is consistent with eq. (6.5). These additional constraints of eqs. (6.3) and (6.5) are formulated as diagonal differences and are therefore "weaker" than demanding consistency on a per cell basis. If the values of the coarse cells themselves are not reproduced when changing the direction of refinement, but instead their diagonal differences, the scheme stays consistent still. However, as stated above, it is not enough that only some coarse cells agree with the conditions. This is further enforced by eq. (6.3) and eq. (6.5) coupling together. Figure 6.1b is helpful for visualizing how the different conditions are located on the lattice relative to the border (blue). We can see that there are cells that enter both eq. (6.3) (pink) and eq. (6.5) (orange).

Based on these results, we can argue that it is not possible to construct a pulse that is reproduced when changing the direction of refinement. The cells that enter each of the constraints are not part of the same causal region. This means that for consistency, cells that cannot influence each other causally are suddenly linked together. For example, the cell I (orange), that is the furthest in the future in fig. 6.1b, is linked to the value of Q (also orange). But, Q cannot causally be related to I because only the cells within its domain of dependence (a triangle with 45° edges and I at its tip) are. Additionally, the equations are coupled so that considerations cannot be restricted to only a certain part of the border. Each time one of the conditions is enforced, the values of more cells need to be made consistent until finally, the values on the entire lattice are pre-determined.

Still, it is possible to construct pulses that are reproduced with a certain error when changing the direction of refinement. This error can be controlled. One example is given in fig. 6.3. If B and C from fig. 6.1 are placed on the yellow cells with value 1, both eq. (6.3) and eq. (6.5) are satisfied with $A = F$ and $Q = E$. Consistency breaks down at the edges of the pulse. When comparing fig. 6.3a and fig. 6.3b the yellow cells become 0.969, the green cells 0.719 and the blue cells 0.281. Moreover, the pulse is one cell wider with values of 0.031 on each side. All of these values have an error of 0.031, which is smaller than 0.125 in the best case of fig. 6.2. We also note that in comparison, the pulse in fig. 6.3 in still fairly well resolved on the coarse lattice. This is in contrast to fig. 6.2 where the pulse is at the limit of what can be resolved on the coarse side. The interplay of lattice resolution and the scale of the pulse's details strongly affects the error. A systematic approach to control this error is not included in the current scope.

The problem with refinement direction symmetry boils down to that the fundamental averaging relation eq. (5.3) is not valid across lattice borders for the considered refinement schemes. This makes it complicated to relate values of differently resolved regions. The effect of broken refinement direction symmetry is studied numerically in chapter 7.

6.1.2 Conservation of the norm at diagonal borders

Connected with averaging inconsistencies is the conservation of the norm eq. (5.1) across lattice borders. It is not clear how the norm behaves at border transitions if the fundamental averaging relation eq. (5.3) does not hold. Still, all of the examples shown in the previous section do have the same norm when it is calculated for time slices with no non-zero border cells.

We can verify the last statement with an explicit calculation. Given fig. 6.4, we can calculate the norm $N(t)$ for each time slice. We use $\gamma(t)$ for the weighted sum on the fine side and $\Gamma(T)$ for the coarse side as

$$N(t,T) = \gamma(t) + \Gamma(T) = \sum_l \varphi_l(t) + 2\sum_L \Phi_L(T), \tag{6.6}$$

where we set the scale a^0 of the fine region to 1 and sum together the individual norms of the fine (φ_l) and coarse (Φ_L) cells. The sums run over all the cells for one time slice of a given region. In the following, we assume that $N(t,T) = N$ is conserved for $t, T < 0$ and the valid combinations of t and T are

$$N = \gamma(-1) + \Gamma(-1) = \gamma(-2) + \Gamma(-1) = \gamma(-3) + \Gamma(-2) = \gamma(-4) + \Gamma(-2). \tag{6.7}$$

We take care not to assume that the norm is conserved for $t, T > 0$. Instead, we want to calculate $\gamma(t=1) + \Gamma(T=1)$ using the eqs. (5.13) to (5.14) for DCTF and check if the norm is conserved. First, the labeled fine cells in fig. 6.4 need to be pulled out of the sum in eq. (6.6). Then, by inserting the EOM (eq. (2.9)) for the remaining cells in the fine sum as well as the

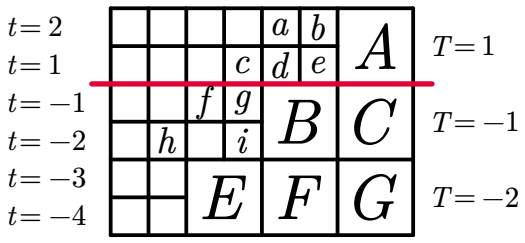


Figure 6.4: Fine (t) and coarse (T) time slices enter the norm N in eq. (6.6). Each coarse time slice contributes to two fine slices. The red line separates $t, T < 0$ from $t, T > 0$.

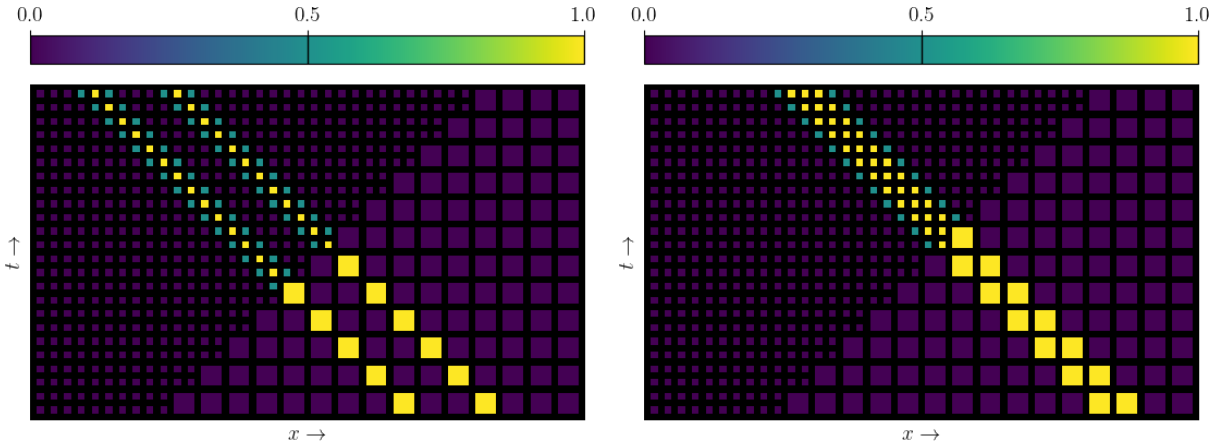


Figure 6.5: Norm conservation at DCTF. (a): The norm for single-cell wide unit pulses is not conserved for the first refinement step. (b): The combination of the two pulses from (a) conserves the two-slice averaged norm.

coarse sum and eqs. (5.13) to (5.14) for the fine border cells we can relate $N(t = 1, T = 1)$ to the known, conserved value N.

Demanding that the norm N is conserved for the step $(t = 1, T = 1)$ gives rise to the condition

$$\Gamma(-2) - \Gamma(-1) = (E - B) - \frac{1}{2}(C - F) + 2i + (h - f). \tag{6.8}$$

We can simplify this expression by demanding that there must not be a border crossing pulse for $t, T < 0$ in addition to the conservation of the norm. This means that the norms of the fine side $\gamma(t < 0)$ and coarse side $\Gamma(T < 0)$ are conserved independently and the left side of eq. (6.8) is 0. Furthermore, $E - B = 0$ and $h - f = 0$ because otherwise, pulses would develop that travel along $G \to B$ and $i \to f$ that can only come from past border-crossings. What remains is $\frac{1}{2}(C - F) = 2i$. If there is a pulse coming from the right and meeting the border at the cell C, then $C - F \neq 0$ and i cannot be 0. But, i can only be non-zero if there was a previous crossing at the border, which we excluded, or $\gamma(-2) \neq \gamma(-1)$. The latter follows from prohibiting pulses along $i \to f$, in which case the value of i cannot propagate to the $t = -1$ slice and the norm must be different for these two time slices. As a result, the norm N cannot be conserved for $N(t = 1, T = 1)$ using the diagonal border refinement scheme from chapter 5.

If we now look at the $t = 2$ time slice and demand that $N = \gamma(2) + \Gamma(1)$, we arrive at

$$\gamma(-2) - \gamma(-1) + \Gamma(-2) - \Gamma(-1) = \frac{1}{2}(A - B) - (B - E) + 2i + (h - f). \tag{6.9}$$

Reusing the same arguments as before leads to $\frac{1}{2}(A - B) = -2i$. This time i can be 0 if A and B are equal and we see that the norm can be conserved at the second refinement step if the pulse hitting the border at C is only one cell wide. Even if the norm changes for $t = 1$, it is restored to its conserved value at $t = 2$.

The two pulses in fig. 6.5a show the two possible ways to place a single-cell unit pulse at a DCTF transition. The pulse further to the right is consistent with the current discussion. The pulse to the left is shifted by one coarse cell relative to the border's geometry and overlaps with the cell B from fig. 6.4 instead of C. This makes it more difficult to repeat the above calculations because the norm changes already at $t = -1$. Still, with similar arguments as before, eq. (6.8) simplifies to $C = F$ and eq. (6.9) simplifies to $A = E$. Both are satisfied for a

single-cell unit pulse crossing the cell B. However, the norm changes at $t = -1$, which is the first time step at which values cross the border.

Instead of investigating the norm for each fine time slice individually, we can also look at the averaged norm $\overline{N} = \frac{1}{2}(\gamma(1) + \gamma(2)) + \Gamma(1)$. Repeating the same exercise as before leads to

$$4\Big(\gamma(-2)-\gamma(-1)\Big)+2\Big(\Gamma(-2)-\Gamma(-1)\Big) = \frac{1}{2}(A-B)-2(B-E)-\frac{1}{2}(C-F)+2(h-f)+4i, \quad (6.10)$$

which can be simplified to $\frac{1}{2}(C-F)-\frac{1}{2}(A-B) = 4i$. With $i = 0$, the condition can be satisfied for example with $F = B = 0$ and $A = C$. This can be achieved with the very specific placement of the pulse in fig. 6.5b. If instead $B = C$, effectively shifting the pulse by one coarse cell to the left, i would be required non-zero. Then, $\gamma(-2) \neq \gamma(-1)$ which we exclude.

6.2 Straight borders

6.2.1 Reflections at straight borders

Before repeating the previous discussions for straight borders, we will first compare all the different refinement schemes for the straight border from section 5.3. The results of SCTF transitions for each of the introduced schemes for a single-cell unit pulse are compared on page 67. Each row corresponds to a different scheme. The images on the LHS give the original pictures and on the RHS, the fine cells are hidden and instead, the coarse ghost cells are visible on the fine side of the lattice.

The refinement equations for diagonal borders were derived by demanding a certain, favorable outcome for the fine pulse after the CTF transition that can be reproduced by the diagonal difference eqs. (5.5) to (5.8) or for the coarse pulse after FTC using the averaging relation eq. (5.3). As such, they are constructed to work for diagonal borders and unsurprisingly fail when being naively applied to straight borders. In particular, the pulse in fig. 6.6 has the expected shape on the fine side, but a reflection appears on the coarse side.

This reflection can be removed by changing the diagonal difference equations. The solution presented in [15] is a delicate combination of modified diagonal differences and averaging. Only one fine border cell is calculated using eq. (5.8). The other fine border cell's value is then determined by demanding that the average relation eq. (5.3) holds across the border. The resulting scheme eqs. (5.18) produce fig. 6.7. However, the side effect of this scheme is that the shape of the pulse on the fine side also changes compared to fig. 6.6. Notably, the value of the pulse center is overshooting and the resulting error for the norm is corrected by negative contributions on the outer flanks of the pulse (purple).

By comparing fig. 6.6 with eqs. (5.15) to (5.17) and fig. 6.7 with eqs. (5.18), the reflection in the former scheme seems to be removed by enforcing the average relation over the diagonal difference equations. As such, we expect that using the diagonal difference eq. (5.6) on the previously average-corrected fine border cell and correcting the other by the average relation should also lead to acceptable results. This change is part of eqs. (5.19) and results in fig. 6.8. The alternating stripes on the fine side, that appear after the transition, demonstrate the failure of this scheme. Still, there is no reflection and the presumed connection of violating the average relation and reflections at the straight border is further strengthened.

The culprit of this situation is that at timelike borders, both CTF and FTC transitions need to be handled together by a single set of refinement equations. In other words, whatever equations make a SCTF transition possible also must not interfere with the chosen SFTC refinement strategy. The refinement schemes for diagonal borders, however, are two independent sets of equations that never mix and by design produce correct results for their respective transitions. What is more, as discussed in section 6.1.1, the DCTF and DFTC transitions are

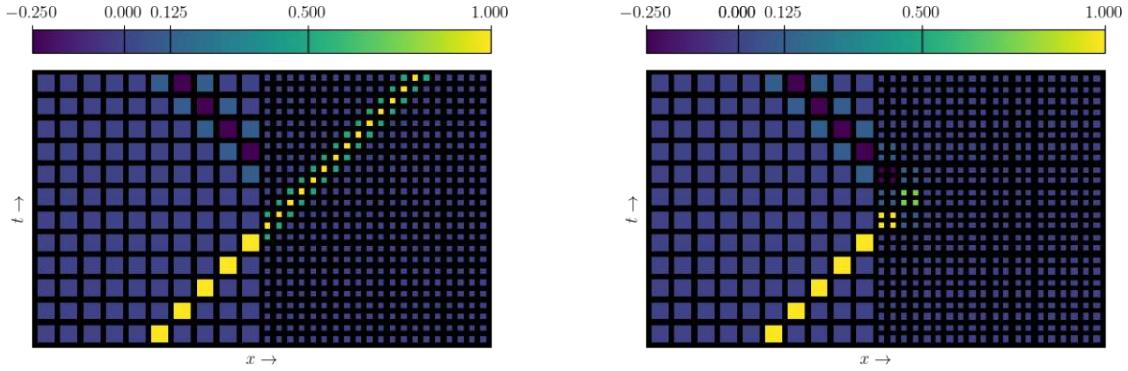


Figure 6.6: SCTF with eqs. (5.15) to (5.17). Results (left) and coarse ghost cells (right).

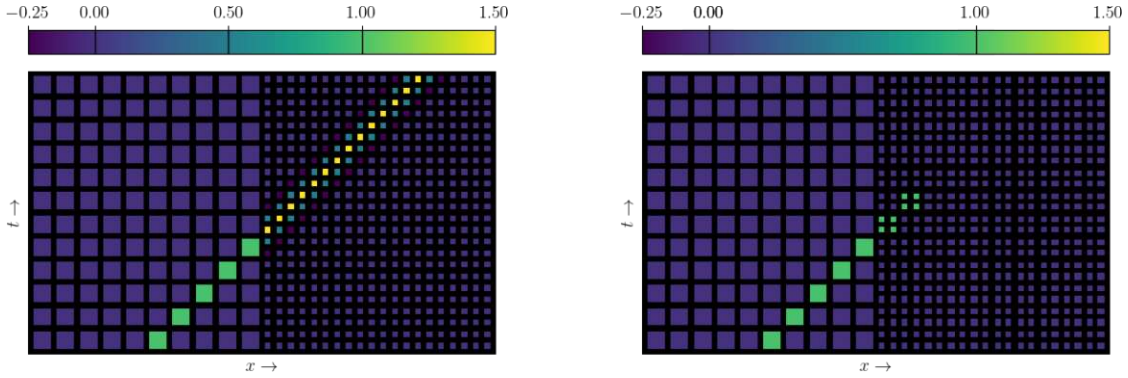


Figure 6.7: SCTF with eqs. (5.18). Results (left) and coarse ghost cells (right).

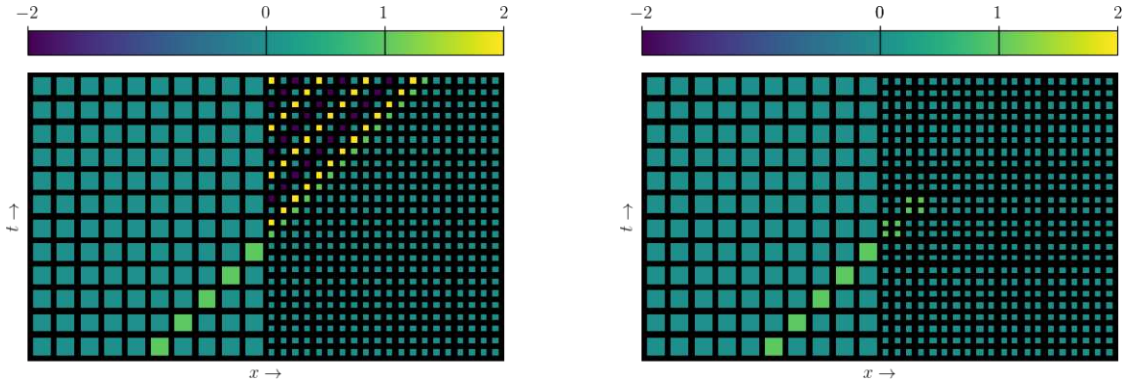


Figure 6.8: SCTF with eqs. (5.19). Results (left) and coarse ghost cells (right).

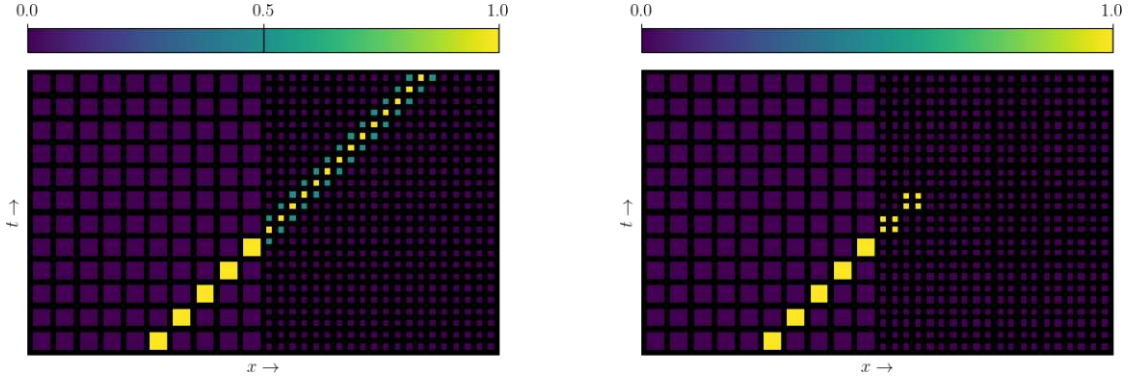


Figure 6.9: SCTF with eqs. (5.20). Results (left) and coarse ghost cells (right).

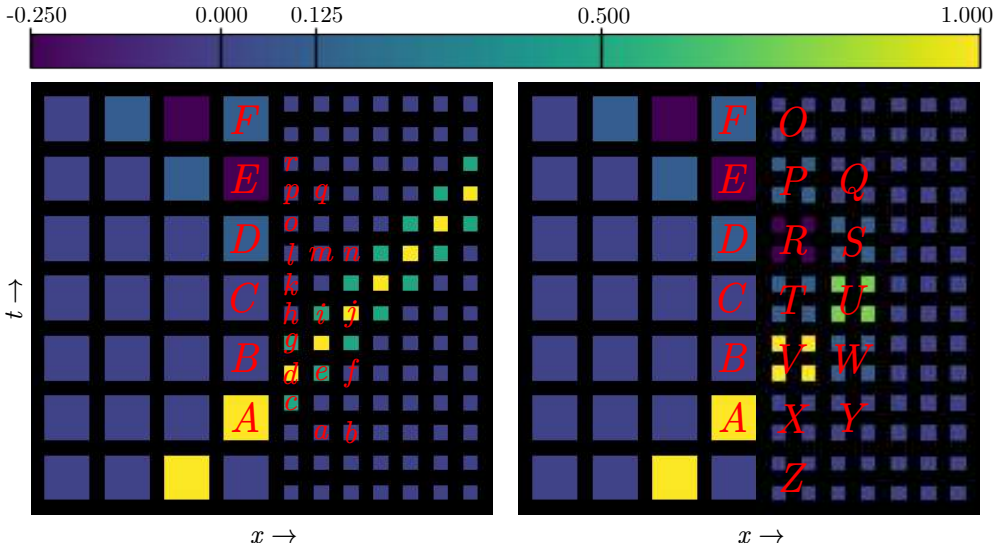


Figure 6.10: Detailed look at fig. 6.6 with labeled cells.

not symmetric w.r.t. changing the direction of refinement. This manifests directly at straight borders, where both refinement directions are combined.

The last scheme eqs. (5.20) produce the desired pulse shape on the fine side (fig. 6.9) by using the unmodified diagonal difference equations. They also manage to get rid of the reflection of fig. 6.6. Their downside is that the averaging relation is violated at the border and a modified average is used to calculate coarse border cells based on fine cells.

To understand how the reflection is connected to the pulse on the fine side and the averaging relation and how it can be avoided, we have to look at the individual steps of the refinement scheme eqs. (5.15) to (5.17) in fig. 6.6 in more detail. For now, we focus on fig. 6.10 and label all cells that enter the refinement equations. Again, the panel on the RHS draws the values of the coarse ghost cells on top of the four, overlapping, fine cells instead of the actual values of the fine cells. In the case of straight borders, there are only two coarse ghost cells for each time slice and any cells with larger x values stay empty.

The first fine cell calculated to have a non-zero value is c and is the result of the γ-scheme eq. (5.17b). It gets its value only from the coarse cells A and V because all the other cells (Y, Z, a) are 0. The ghost cell V inherits its value from A via the default update stencil. Norm conservation demands that the norm of the pulse on the fine side is $2A$ so that for SCTF transitions, off-centered cells (like c) have to obtain only half of the value of the coarse cells carrying the pulse to the fine side. Off-centered in this context shall mean not centered w.r.t. the characteristic of the incoming coarse pulse. On the other hand, d is centered and calculated with the α-scheme eq. (5.17a). The same coarse cells contribute (Y, Z, b are 0) and d is equal to A. With c and d determined, we can propagate their values along the incoming characteristic using the default update stencil because these future cells are no longer affected by the treatment of border cells. We end up with a pulse on the fine side that has a value of A $(= V)$ centered on the incoming pulse and $A/2$ at its leading edge toward larger values of x. From this alone, we can already construct the trailing edge of the pulse on the fine side. Because of symmetry along the pulse center and for conserving the norm, there can only be one cell with value $A/2$ left of the pulse center. To check if the refinement scheme delivers this result, we have to look at g, which is again calculated using the γ-scheme. The only contributing cell is e ($B, X = 0$ as well as $T - W$), which is consistent with the symmetry argument. W is calculated as $A/8$ by averaging the overlapping fine cells and T is then the result of using the default update stencil with B, W and X.

We now know all the values of the cells that are not part of the forward light cones of the cells C and h and the value of the ghost cell T. The shape of the pulse on the fine side is the result of $A - Z = V - Y = A$ and $B - X = T - W = 0$. The choice of the values for the parameters α and γ, that enter the diagonal difference eqs. (5.5) to (5.8), does not affect the results at straight borders for the simple, free theory, which is what we have to demand to be consistent with the refinement schemes of diagonal borders. We also identified the first component of the reflection that originates in W. However, the reflection does not influence the values of the fine cells carrying the pulse because only the difference of the coarse cells along the propagation characteristic of the reflection enter. This value is constant 0.

Continuing to walk the steps of the refinement scheme leads to $h = 0$ because the diagonal differences of the coarse ghost cells $B - X = T - W = 0$ as well as $f = 0$. For $k = 0$, we first obtain $U = 3A/4$ by averaging the pulse on the fine side to a value that is smaller than necessary to properly propagate the coarse pulse. Because of this, the default update stencil yields a non-zero R and effectively results in a reflection that propagates from R to E and back to non-ghost cells. The only non-zero contributions from coarse cells to the diagonal difference equations for k come from $C - V$ and $R - U$, which again are unaffected by the reflection. In spite of the reflection, both of these differences have the same value $-A$. All the other future fine cells at the border become 0.

Therefore, the light blue flanks of the reflection play no role in this refinement scheme. If they were not here ($= 0$), nothing would change, except that the conservation of the norm would be violated on the coarse side. The situation is different for the purple part of the reflection. Because $U \neq V$ the negative value of R is required to give $C - V = R - U$. The latter equality is again a manifestation of the simple theory not being susceptible to the values of the parameters in the diagonal difference equations. Any approach to remove the reflection at straight borders has to respect this equality while also providing a means for the coarse ghost cells' values to be influenced by the overlapping fine cells' values for SFTC transitions.

We already know that eqs. (5.18) (fig. 6.7) are a solution to this problem. They manage to cure the reflection by making $U = V$ while simultaneously obtaining U from averaging fine cells. For this to happen, the pulse on the fine side is changed. Keeping the labels from fig. 6.10, d, which is centered at the incoming characteristic, is now overshooting by a factor of 1.5. This cell is calculated by the modified diagonal difference eq. (5.18c) which simplifies to $2V - e = 1.5A$ in this case. The off-centered cells c, e, g turn out with the same values as before ($A/2$) because their calculation remains unchanged. The weight factor 2 in front of V is necessary to reproduce the value of $V = A$ when averaging the overlapping fine cells, two of which are d and e. This way, the incoming coarse pulse propagates into the coarse ghost cells, that are partly obtained by averaging and partly by the default update stencil, as if there were no border. Previously, in fig. 6.6 the value of the coarse ghost cells along the incoming pulse's characteristic changed suddenly. Equations (5.18) also get rid of the light blue reflection flanks as a result of enforcing norm conservation on the pulse on the fine side. As the centered fine cells now overshoot by 1.5, the norm and average can only be conserved by adding more cells to the flanks of the pulse (ie. f and h) that compensate for this value by carrying a value of $-0.25A$ each. The resulting pulse is still symmetric along the incoming characteristic. Then, calculating W (or T, S) by averaging results in exactly 0.

In conclusion, we can now complete the picture and understand where the reflection for eqs. (5.15) to (5.17) comes from. The reflection in fig. 6.10 is three coarse cells wide. The outer flanks (light blue) are symmetric and have positive values, the center (purple) is negative. When we examine the reflection's center, it is clear that the issue stems from the ghost cell U with the largest x value. When tracking the incoming pulse from the coarse side, the first coarse ghost cell on its path (V) retains the value of the pulse value but the second (U) no longer does and is now green instead of yellow. The difference in amplitude is then reflected.

Another way of thinking is to say that for the green ghost cell U to change to its value, there has to be another pulse coming from the right, which, when superimposed with the original pulse from the left, results in the change of value. This pulse then manifests on the coarse side. Having now pinpointed the origin of the reflection to the ghost cells with the largest x value, their calculation seems to be problematic. Indeed, those coarse ghost cells are the only coarse cells in the scheme eqs. (5.15) to (5.17) that are calculated by averaging fine cells. They are not calculated using the default update stencil. As previously discussed, averaging is not consistent with diagonal difference equations and this problem is reappearing here.

With this in mind we also understand the origin of the flanks of the reflection. Averaging already produces a non-zero ghost cell (W, light blue) one coarse time step before the green coarse ghost cell U. Because of how the scheme is executed, the cells of this "column" on the lattice only contribute to the left half of their forward light cone leading to a new pulse that is travelling to the left. It should not be surprising that the reflected pulse on the coarse side has 0 norm because the light blue cells are $-1/2$ times the negative valued purple cells. The diagonal borders equations are overall norm conserving (see section 6.1.2) so that the pulse on the fine side has the same norm as the incoming coarse pulse. When there is a reflection, the norm across the entire time slice can only be conserved if the reflection's norm cancelles out.

Fixing the average problem

Having obtained a deeper understanding of the situation at straight borders, we are now left with fig. 6.9 using eqs. (5.20). At a first glance, all problems seem to be resolved. In order to keep the pulse shape on the fine side consistent with the diagonal borders, the diagonal difference eqs. (5.5) to (5.8) have to be used unmodified. Additionally, to get rid of the reflection, the averaging procedure of fine cells needs to be adapted. In the following, we will compare the fundamental averaging relation eq. (5.3) to the modified eqs. (5.20). Figure 5.6 is repeated for convenience.

We want to compare the update equations for the cell A (eq. (6.11)) of the different refinement schemes. For this, we pick out eqs. (5.16), where for the cell H averaging is used and compare it with eq. (5.24) that uses the modified average for H

$$A = B - D + H, \tag{6.11}$$

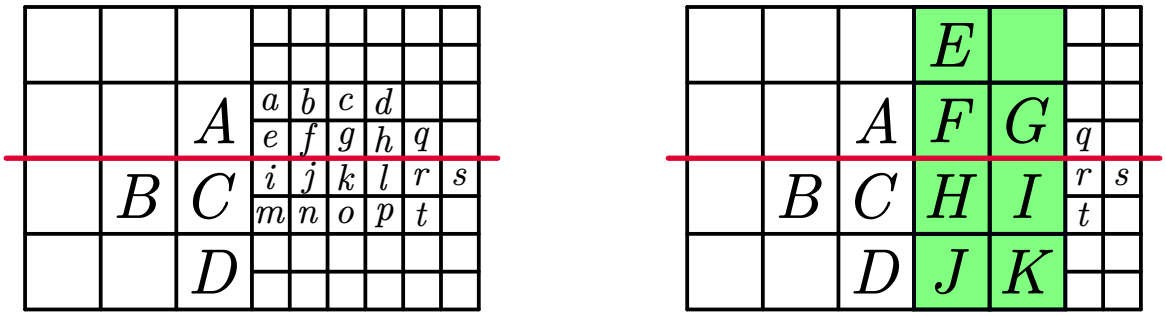
$$A' = B - D + \frac{1}{4}(i + j + m + n), \tag{5.16}$$

$$A'' = B - D + \frac{1}{2}(i + j) - \frac{1}{4}(m - e). \tag{5.24}$$

Subtracting eq. (5.16) from eq. (5.24) yields the modification of the average

$$A'' - A' = \frac{1}{4}\Big((i - n) - (e - j)\Big). \tag{6.12}$$

The suggestive placement of parenthesis groups the cells along diagonals. It is easy to see that for pulses propagating from the fine to the coarse side at the straight border eq. (6.12) will be 0. Such pulses have conserved values along the propagation characteristics, ie. $n \to i$ and $j \to e$. Therefore, SFTC transitions as part of eqs. (5.20) are calculated using the unmodified averaging relation. The average correction only takes effect for pulses propagating away from the border, which are the result of past SCTF transitions. This is exactly the behavior we are looking for to avoid reflections during SCTF. However, eq. (6.12) can also be 0 for SCTF, at least for some time slices. A straightforward example is given by a coarse unit pulse that is multiple cells wide so that all the cells that contribute to the average correction have the same value. This example is similar to fig. 6.3 and its discussion in section 6.1.1.



(a) (b)

Figure 5.6: Straight border.

Equation (6.12) contains fine border cells that can be substituted by their respective refinement equations from eqs. (5.15) to (5.17). By doing so, the weighting factors of the diagonal difference eqs. (5.5) to (5.8) are introduced into the average correction

$$A'' - A' = \frac{1}{4}\left(j - o + \left[\frac{1}{2}(F - I) - (C - J)\right] - (\frac{\gamma}{2} + \alpha)\Big[(F - I) - (C - J)\Big]\right). \tag{6.13}$$

As always, the second, bracketed expression with the coarse cells reduces to 0 for the free theory and the effect of the weights is removed. For more general theories, the fixed contribution from the coarse cells in the first bracket is extended by the second, bracketed expression that is the only one being affected by the weights. Note that while the weights for the diagonal difference equations were introduced independently from each other, they are coupled together in eq. (6.13). This makes it more involved to tune the values of the weights.

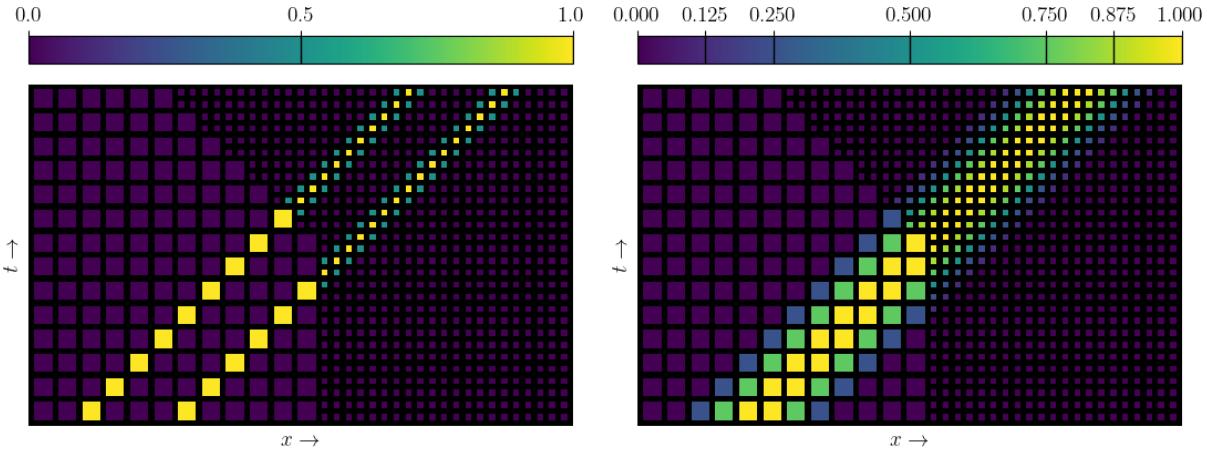
Universality of diagonal and straight borders

The importance of having universal results at CTF borders for both diagonal and straight borders is highlighted in fig. 6.11. The lattice is built from a straight CTF border that morphs into a diagonal border. This change is categorized as the step after the cusp for straight double-cell cusps and eqs. (5.42) to (5.48) apply. In particular, fig. 6.11 employs eqs. (5.46) and is therefore consistent with eqs. (5.20) for straight borders.

If we compare the values of the first and last time slices in fig. 6.11a, we recognize the results of CTF refinement of single-cell coarse pulses. However, without knowing the entire time evolution, we are unable to identify if the refinement took place at a diagonal or straight border. Therefore, we can argue that SCTF and DCTF and thus, any CTF transition are equivalent in the current refinement schemes. This result is important and, in fact, necessary if we want to minimize the effect of inexact refinement procedures. Only then can we derive reliable error estimations.

In comparison, if instead of eqs. (5.20), the straight border refinement scheme with modified diagonal difference equations for the fine border cells' eqs. (5.18) was used, the pulse to the right in fig. 6.11a would not match the left pulse in the final time slice. It would have the profile shown in fig. 6.7. Without knowing the history of the lattice, we would not be able to deal with two separate errors for a CTF transition on a single lattice.

The second example in fig. 6.11b provides another perspective. The pulse has the same special shape that was derived in section 6.2.2 and is favorable for refinement direction symmetry. In the current context, the pulse is placed in such a way that it directly passes through the cusp. Unaffected by the lattice cusp, the results on the fine side are identical to fig. 6.14. Again, if instead eqs. (5.18) were used, this result would not be possible.



(a) (b)

Figure 6.11: Lattice with a SCTF border changing to a DCTF border. (a): Refinement at either border with eqs. (5.20) and eqs. (5.13) to (5.14) respectively yields the same result for coarse single-cell unit pulses. (b): The special pulse from fig. 6.14 hits the cusp and still produces results identical with fig. 6.14.

6.2.2 Refinement direction symmetry at straight borders

We now repeat the discussions of the earlier sections for the different straight border refinement schemes. On page 73 the same approach as in fig. 6.2 is used to tabulate the results for changing the direction of refinement for the different schemes. It is important to reiterate that for lattice configurations with straight borders, only a single set of refinement equations handles both CTF and FTC transitions simultaneously. This is in contrast to diagonal borders where each transition has its separate set of equations.

The first scheme eqs. (5.15) to (5.17) in fig. 6.12 are using the unchanged diagonal difference eqs. (5.5) to (5.8) for the fine border cells and the fundamental averaging relation eq. (5.3) for coarse cells. Except for the reflection discussed in the previous section 6.2.1, this scheme produces identical results as for diagonal borders. For refinement direction symmetry to hold, F (in fig. 5.6) needs to be consistent with the average over the overlapping fine cells a, b, e and f. Because the scheme's equations are identical and this starting point is identical to section 6.1.1 we can obtain the same results. To conserve space, these results for the straight border are given pictorially in fig. 6.16. The same colors are used to mark coarse ghost cells and cells entering the two conditions

$$F - I = C - J = \frac{1}{2}(A - K) \qquad \text{(pink)}, \tag{6.14}$$

$$E - G = A - H = \frac{1}{2}(M - J) \qquad \text{(orange)}. \tag{6.15}$$

In the case of straight borders, both conditions are identical because they describe the same cell configuration that is translated on the lattice. Previously, for diagonal borders, the two conditions resulted from the different positioning of cells w.r.t. the lattice border. Still, repeating the condition and marking the corresponding cells in two colors highlights again that the equations are coupled. Hence, the same results for controlling the error when changing the direction of refinement at diagonal borders apply, except for the reflection in the this scheme.

In the context of refinement direction symmetry, the reflection can be interpreted further. The resulting coarse pulse on the RHS of fig. 6.12 differs by the original coarse pulse on the LHS by exactly the negative of the reflection. Hence, we can argue that the reflection contains the information required to fully restore the initial pulse when changing the direction of refinement.

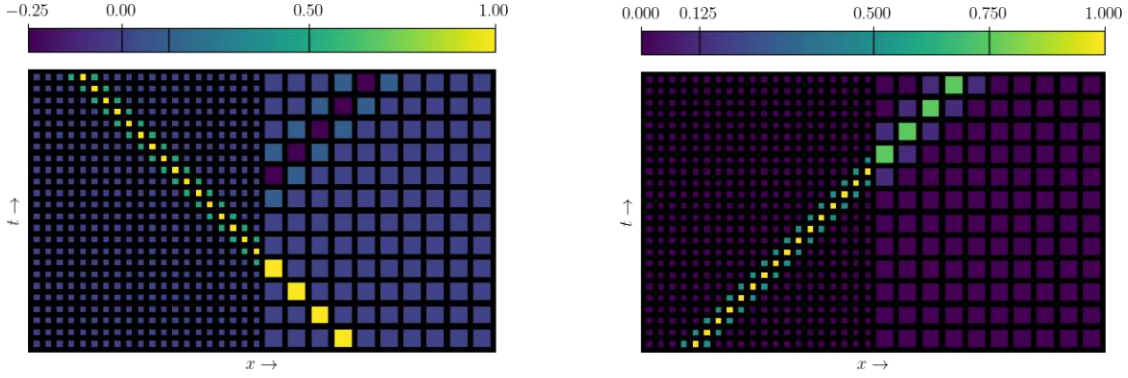


Figure 6.12: Reverting the refinement direction for eqs. (5.15) to (5.17).

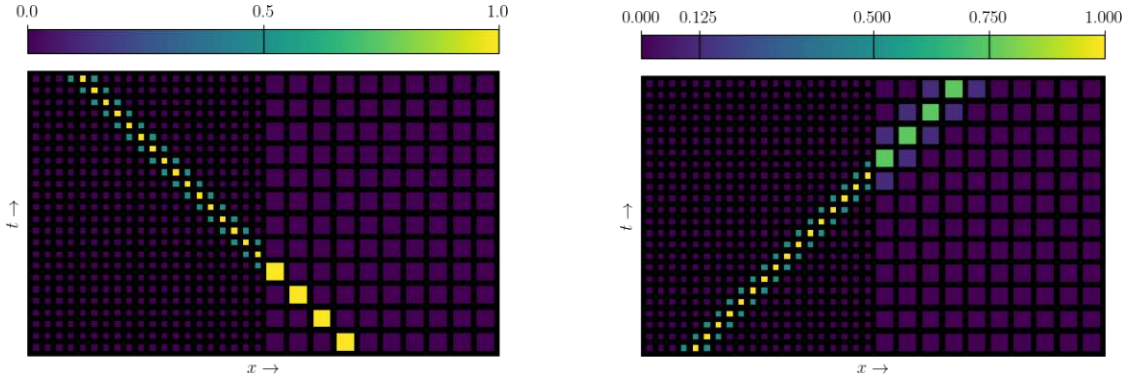


Figure 6.13: Reverting the refinement direction for eqs. (5.20).

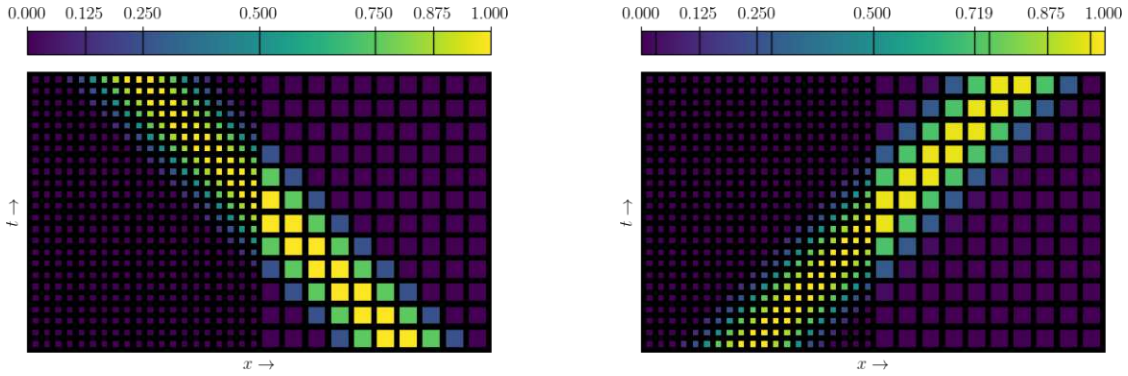


Figure 6.14: Reverting the refinement direction for eqs. (5.20) using the same pulse as fig. 6.3.

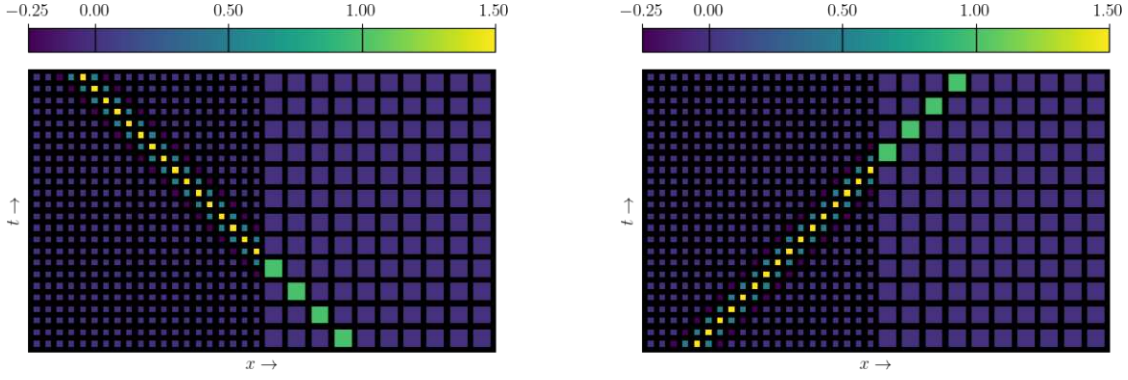
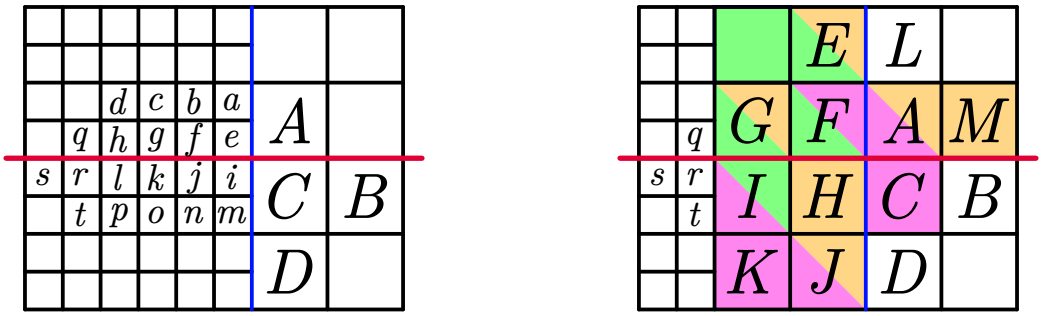
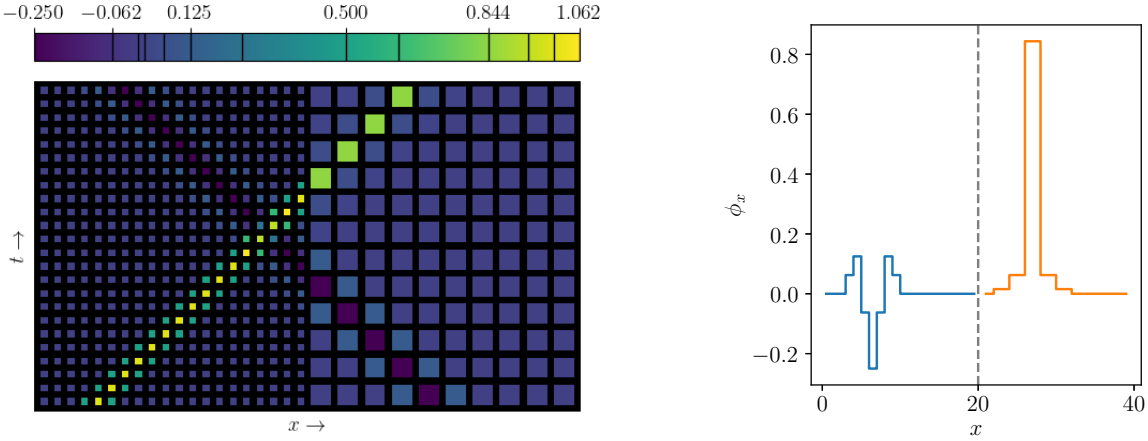


Figure 6.15: Reverting the refinement direction for eqs. (5.18).



(a) (b)

Figure 6.16: Refinement direction symmetry at the straight border (blue). (b): For F to be consistent when changing the direction of refinement in the schemes of eqs. (5.15) to (5.17) or eqs. (5.20), the values of the pink cells are linked together. By shifting this result on the lattice the orange cells are linked together. This result is identical to fig. 6.1b.



(a) (b)

Figure 6.17: Changing the direction of refinement for eqs. (5.15) to (5.17) with the reflection of a CTF transition from fig. 6.12 as part of the initial time slice (b): The profile plot corresponds to the last time slice of the lattice plot (a).

The results for the average corrected scheme eqs. (5.20) are given in fig. 6.13. In principle, the same discussion can be repeated. The obvious difference to fig. 6.12 is that now the reflection is removed. Still, it is not straightforward to get to the same eqs. (6.14) and (6.15) in the average corrected scheme because the modified average relation enters the derivation. As proven in appendix E, the modified average can be consistently used instead of the standard average in this scheme. Furthermore, we should not be surprised to recover the same conditions as for diagonal borders. The underlying equations are reused between the schemes and the average correction only takes effect for CTF transitions and reduces to the standard average when changing the direction of refinement for FTC (see section 6.2.1). To further underline this relation, the special pulse for the case for diagonal borders in fig. 6.3 is also used at the straight border with eqs. (5.20) in fig. 6.14. identical results with identical errors are produced.

The last fig. 6.15 on page 73 uses eqs. (5.18) with a modified diagonal difference scheme for the fine border cells. This straight border scheme manages to preserve refinement direction symmetry. The reason is that the value of F (fig. 6.16) is recovered by using the default average on the overlapping fine cells, which is made possible by the special profile of the fine pulse after the SCTF transition.

Finally, we want to investigate the results of changing the direction of refinement while also including the reflection generated by the scheme eqs. (5.15) to (5.17) in fig. 6.12. We can test the previous statement that the reflection carries the necessary information to restore the initial pulse. The results are given in fig. 6.17. For better identification of the values, the profile plot of the last time slice is also given. The final profile of the coarse pulse comes closer to the initial pulse in fig. 6.12 than without including the reflection. However, refinement direction symmetry cannot be recovered with the inclusion of the reflection because the initial configuration of fig. 6.17 does not lead to the same coarse ghost cells as in the case of the SCTF transition. Additionally, the incoming coarse reflection transitions to the fine side and manifests as a pulse, which is also visible in fig. 6.17b.

6.2.3 Conservation of the norm at straight borders

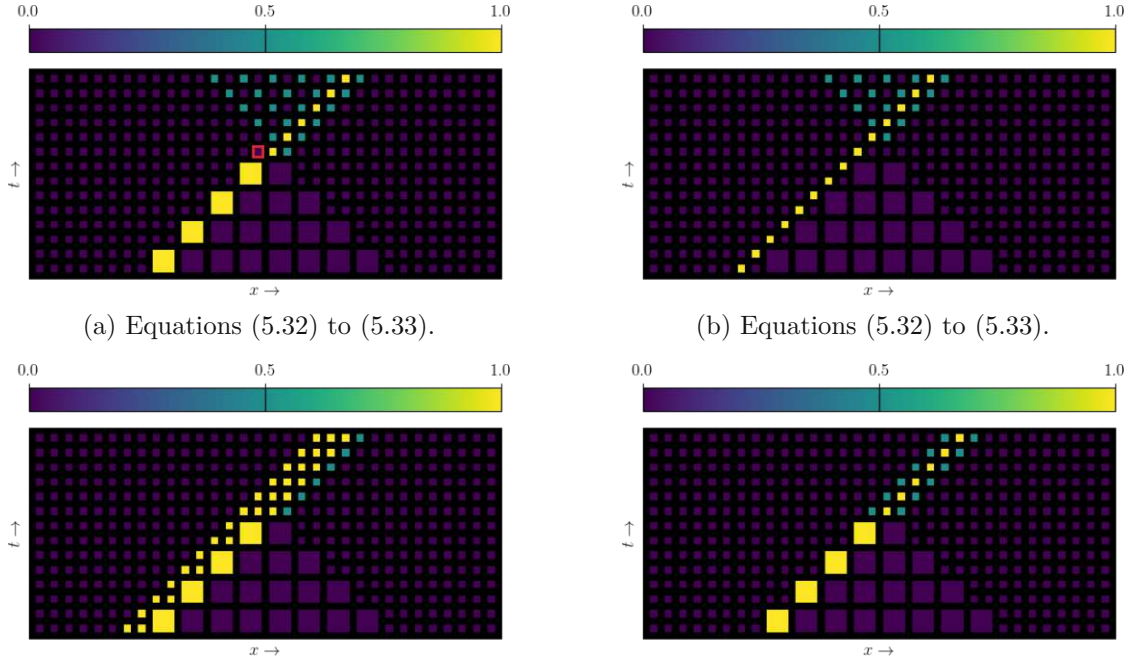
Again, the procedure is analogous to section 6.1.2 and we want to explicitly derive conditions for norm preserving pulses at the straight border for the different refinement schemes. However, there is a slight complication when working with eqs. (5.15) to (5.17) and eqs. (5.20), the two schemes that reuse the unmodified diagonal differences. We can only demand norm conservation for coarse time slices $T < 0$ and fine time slices $t < -1$ (fig. 6.4). In comparison to diagonal borders, the norm on the fine side can be violated for $t = -1$. The reason for this change is that if we wanted to prohibit border crossing values for $T < 0$ and $t < 0$, we would have to demand $C = 0$ (fig. 6.4). Otherwise, cell i would obtain a contribution from C via the refinement equations. But this would restrict the possible placements of pulses at the border. It turns out that SCTF refinement with these two schemes requires an additional pulse from the fine side to conserve the norm for single time slices. The two-slice averaged norm is conserved for any SCTF transition. In the case of single SFTC transitions, all norms are conserved for single-cell unit pulses that align with the centers of the coarse cells.

The situation is more involved for eqs. (5.18) because the value of C also affects m. Additionally, the CTF transition in this scheme takes five fine time slices to complete (ie. a coarse unit pulse becomes five cells wide on the fine side) and therefore requires to back-track the norm for as many time slices. Still, we can argue some results. From section 6.2.1 we know that this scheme is preserving the fundamental averaging relation eq. (5.3) across the border. Therefore, it immediately follows that the two-slice averaged norm is conserved for all pulse configurations.

6.3 Diagonal borders at CCTF

Two refinement schemes for CCTF are introduced in section 5.5. Now, we study their results in fig. 6.18. When reusing the DCTF eqs. (5.13) to (5.14) as part of the cusp refinement eqs. (5.32) to (5.33) for single-cell unit pulses on the fine side (fig. 6.18b) or coarse side (fig. 6.18a), checkerboard artifacts are generated. In the latter case, the incoming coarse pulse transitions correctly to the fine region. However, the cell to the left of the cusp's center (red border in fig. 6.18a) is missing its value so that the propagation of the left flank of the pulse on the fine side is wrong. That cell is calculated as part of the left border's refinement and is a continuation of the previous time slices' results. After the cusp, no more refinement takes place and the cell at the same x and $t + 2$ as the marked one can be calculated via the default update stencil that is then missing the red cell's value for correct results.

In the configuration of fig. 6.18b, again, the DCTF transition as part of the right border's refinement is correct. Although the incoming pulse is a single-cell fine pulse, the coarse ghost cells of the left border are used for refinement instead. These coarse ghost cells contain the representation of the fine pulse as a single-cell unit pulse. This time, the problem is the additional values at the flanks of the fine pulse.



(a) Equations (5.32) to (5.33).

(b) Equations (5.32) to (5.33).

(c) Equations (5.32) to (5.33).

(d) Dedicated eqs. (5.30) to (5.31).

Figure 6.18: Refinement schemes at CCTF. (a), (b) and (c) reuse DCTF refinement. (d) uses dedicated equations. In comparison, checkerboard artifacts appear in (a) and (b) because of missing (red) or additional values.

The observations lead to the conclusion that the artifacts for CCTF transitions arise because the incoming pulse's representation on fine and coarse cells does not match. The single-cell unit pulse is the extreme case with the most discrepancy between two resolutions. Therefore, when placing a pulse at the cusp that has a consistent representation on the fine and (overlapping) coarse cells, the problems can be circumvented. The details of such a pulse profile are then well resolved on both the fine and coarse side. This is demonstrated in fig. 6.18c with a unit pulse that is multiple cells wide placed on the left border.

The peculiarities of the dedicated eqs. (5.30) to (5.31) of the second CCTF refinement scheme are highlighted in section 5.5. Compared to the previous scheme, the changes cure the artifacts for single-cell coarse unit pulses and result in fig. 6.18d. The additional terms are necessary to allow for both fine and coarse (ghost) cells of either border to enter the equations and eliminate inconsistencies. The changed geometry of the coarse-to-fine cusp leads to missing information when reusing the equations for diagonal border refinement. The fine cells of the respective other border do not exist for the DCTF scheme and the overlapping coarse ghost cells are not resolved high enough to supplement this information.

Chapter 7

Numerical results

In this chapter, we study the behavior of Gaussian and triangular pulses under the refinement schemes from chapter 5. In general, the theory defined in chapter 2 ("free case") is discretized on square lattice cells. The possible combinations of the pulse shapes, the different lattice resolutions and the refinement border types lead to an extensive collection of results. Only a select subset is included in this chapter. More results are available in the documentation supplement of *sam*.

In the following, we focus on two aspects. On the one hand, we investigate the deformation of the pulse profiles after refinement. We characterize the error with its time slice profile for the last simulation time step. Verifying the conservation of the norm eq. (5.1) for each case is also important. Therefore, we compare the norms of the discretized pulses on the first and final time slices. Appendix F tabulates the errors of the norm for all discussed setups.

On the other hand, we determine the order p of the scaling behavior $\sim \mathcal{O}(a^p)$ of the numerical error with the lattice constant a. The order of the error introduced by the refinement equations should match or be higher than the order of the discretization scheme for the theory ($p = 2$, see chapter 2). This is important because it ensures that the convergence behavior of the continuum limit of the theory does not change. Except for the averaging eq. (5.3), which is of $\mathcal{O}(a^2)$ (see appendix A), we did not derive any analytical values previously. Instead, the error scaling of the refinement schemes is estimated based on numerical results and compared to $p = 2$.

The error is given by the difference of the exact and simulated results at each discrete site on the final time slice of the lattice. As discussed in chapter 3, the solutions for the free theory can be easily constructed from the initial conditions. The same functional expression that is evaluated for the initial pulse is also evaluated at appropriately shifted arguments which results in the exact values $\phi^E(x, t_N)$. The numerical values $\phi_x(t_N)$ are obtained from the last time step t_N. N is the number of time steps and depends on the temporal size of the lattice and the chosen scale of the cells set by t/a. This ratio gives the amount of time passed per calculated time step. The numerical results are always interpreted as an approximation of the exact results at the centers of the lattice sites. Therefore, the exact values are always calculated as the cell-centered values of the analytic expressions.

To quantify the error, we use the two norms

$$\|e^i\|_{l^2} = \sqrt{a_i \sum_{j=1}^{N_x^i} \left(\phi^E(x_j, t_{N_i}) - \phi^i_{x_j}(t_{N_i})\right)^2}, \tag{7.1}$$

$$\|e^i\|_{l^\infty} = \max_{1 \le j \le N_x^i} \left|\phi^E(x_j, t_{N_i}) - \phi^i_{x+j}(t_{N_i})\right|. \tag{7.2}$$

The index i is used to label different simulation runs and $j = 1, \ldots, N_x^i$ runs over all lattice sites of the last time slice for the ith run.

The scaling rate of the error is estimated following a standard procedure (see eg. [17]). A series of simulation runs with identical initial conditions but increasingly finer lattices (scale) are collected. Let $\|e^i\|$ be the error measured in any norm for the ith run. We model $\|e\| \sim A_0 \cdot (a^0)^p + A_1 \cdot (a^1)^p$ and by including the Courant number C, the error term only depends on one lattice spacing $a := a^0$

$$\|e^i\| \sim Dh_i^p := (A_0 + A_1/C^p) \cdot (a_i^0)^p. \tag{7.3}$$

By absorbing all prefactors and dimensions in the constant D, the only scaling parameter becomes h_i. Now, consider a series of runs where $h_i = 2^{-i}$ with i starting from 0. We can estimate the value of p by taking the $\log_2$ of eq. (7.3) for two subsequent runs and subtracting the equations as

$$\begin{aligned} \log_2\left(\frac{\|e^{i+1}\|}{\|e^i\|}\right) &= p \log_2\left(\frac{h_{i+1}}{h_i}\right) \\ \Rightarrow p = \frac{\log_2(\|e^{i+1}\|/\|e^i\|)}{\log_2(2^{-i-1}/2^{-i})} &= -\log_2(\|e^{i+1}\|/\|e^i\|). \end{aligned} \tag{7.4}$$

Instead of calculating p for each pair of $\{i, i+1\}$, we plot $\log_2(\|e^i\|)$ over $\log_2(h^i)$ so that the slope of a linear regression through the data points will approximate p. We use a logarithmic axis ratio of $\log_2 y / \log_2 x = 2$. The x-axis with the values for h^i is flipped with decreasing values (finer scale) to the right. The result for $p = 2$ on such a plot is a line with a $-45°$ slope. This reference line is always included in the scaling plots where its position is set by the first data point. In comparison, worse scaling will result in a flatter slope.

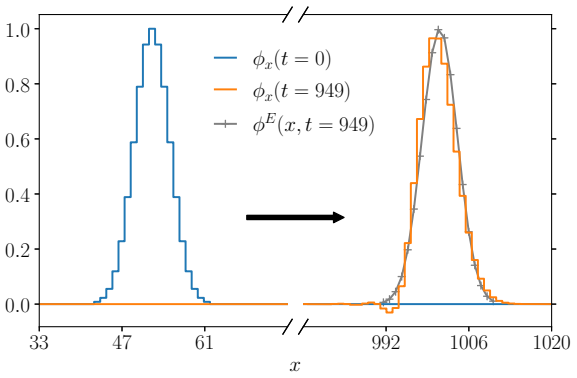
7.1 Homogeneous lattices

In order to get familiar with the testing methodology, we revisit the configuration from fig. 3.2 with dispersive effects. An identical Gaussian with its cutoff lowered to 0.005 is again placed on a homogeneous lattice with $C = 32/33$. In contrast to fig. 3.2, the time evolution is calculated over a much longer timespan until $t_N = 949$. The results are shown in fig. 7.1.

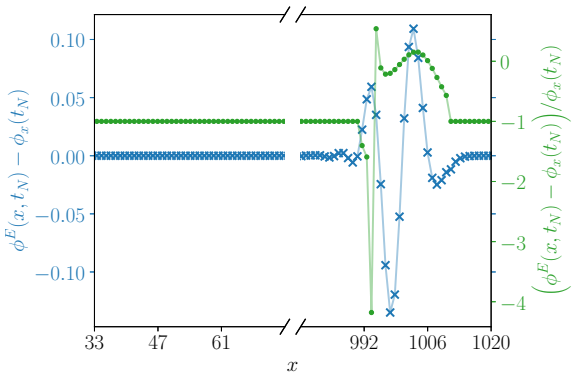
There are three different plot types. The first, fig. 7.1a, shows the values of the lattice sites of a time slice for the given time step t, i.e. the pulse profile. $\phi^E(x, t_N)$ is also sampled at the final time step for comparison. The x-axis is the spatial coordinate on the lattice in units of a divided by the scale. When reducing the scale, the lattice accumulates more cells with smaller a so that more values are sampled per unit of space. The second, fig. 7.1b, contains the error and relative error profiles. The error profile (blue) is the result of subtracting the calculated values from $\phi^E(x, t_N)$. For the relative error (green) these error values are normalized to the values of $\phi_x(t_N)$ for each site. The relative error is only plotted if $\phi_x(t_N)$ is not zero[9]. The third, fig. 7.1c, allows estimating the scaling rates of the two error norms following the previous discussion. The targeted scaling with $\mathcal{O}(a^2)$ is marked with the dotted gray line.

We start our analysis by inspecting fig. 7.1a. The final profile for $t_N = 949$ (orange) is deformed. Its base is broadened and its peak falls behind the exact values. At the trailing edge, the ripples that were already present for $N = 19$ (fig. 3.2) are amplified. They lead to oscillating values in fig. 7.1b that stabilize to values above -1 for the relative error. The relative error has values at -1 across the entire time slice, which means that erroneous, non-zero values appear where $\phi^E(x, t_N)$ is exactly zero. However, the magnitude of these errors is too small to spot in the plots.

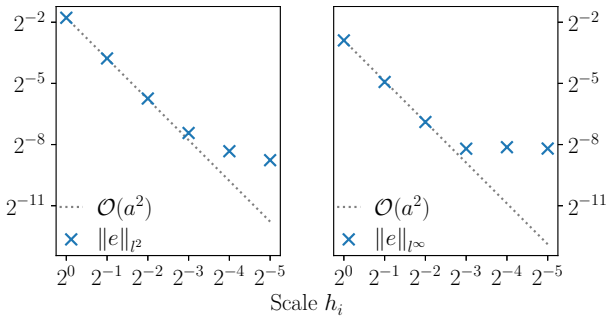
[9] Values numerically close to zero, as measured by `numpy.isclose()` with default arguments, are discarded.



(a) Time slice plots for scale $h_i = 2^0$.

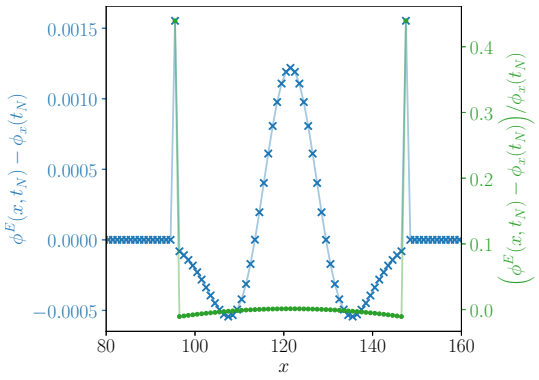


(b) Error profile at $t_N = 949$ and scale $h_i = 2^0$.

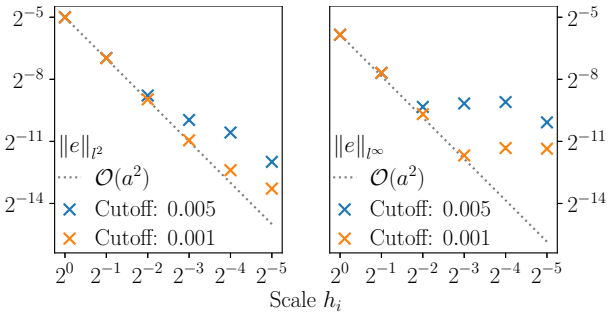


(c) Error scaling.

Figure 7.1: Results for a homogeneous lattice with $C = 32/33$. The pulse is sampled cell-centered from a Gaussian with $\bar{x} = \frac{50}{C} + \frac{1}{2}(\frac{1}{C} - 1)$, $\sigma = 3$ and cutoff at 0.005. (a): The final pulse at $t = 949$ (orange) is no longer identical to the starting pulse (blue). (b): Constant relative error values with -1 (green) indicate that $\phi^E(x, t_N) = 0$ but $\phi_x(t_N) \neq 0$. (c): Both errors from eqs. (7.1) and (7.2) scale with the rate $p = 2$ (gray line) up to small h_i.

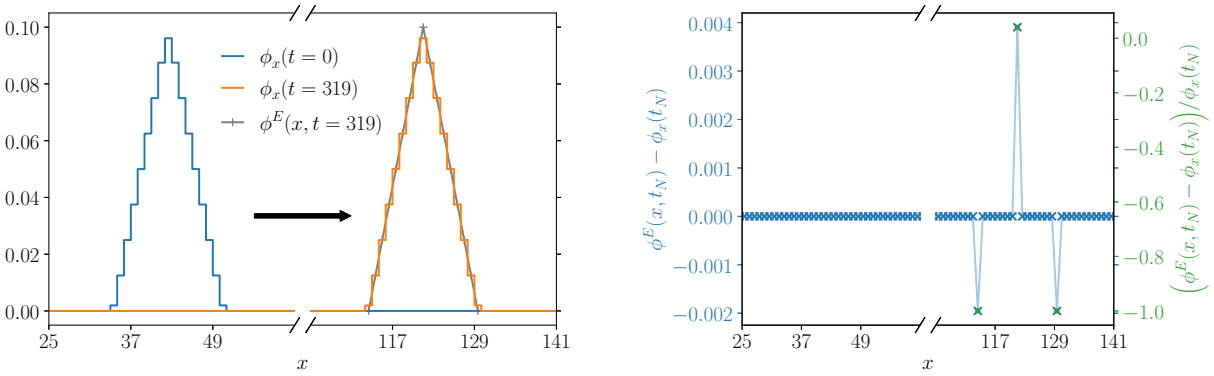


(a) Error profile at $t_N = 319$ and $h_i = 2^{-2}$.



(b) Error scaling.

Figure 7.2: Results for a homogeneous lattice. The pulse is sampled cell-averaged from a Gaussian with $\sigma = 8$ and cutoff at 0.005.



(a) Time slice plots for scale $h_i = 2^{-2}$.

(b) Error profile at $t_N = 319$ and $h_i = 2^{-2}$.

Figure 7.3: Results for a homogeneous lattice. The pulse is sampled cell-averaged from a triangular shape centered at $x = 42$ that is 16 x-units wide with a peak value of 0.1. (b): Three distinct error values appear at the center and edges.

Still, the error scaling in fig. 7.1c follows the reference line (gray). Because there is no refinement performed, the scaling rate has to match the theoretical value obtained by discretizing the EOM (2.10). The deviation from this rate at finer scales is a systematic discretization error. This problem is that at those small scales, the magnitude of the error drops below the cutoff used when sampling from the analytic pulse.

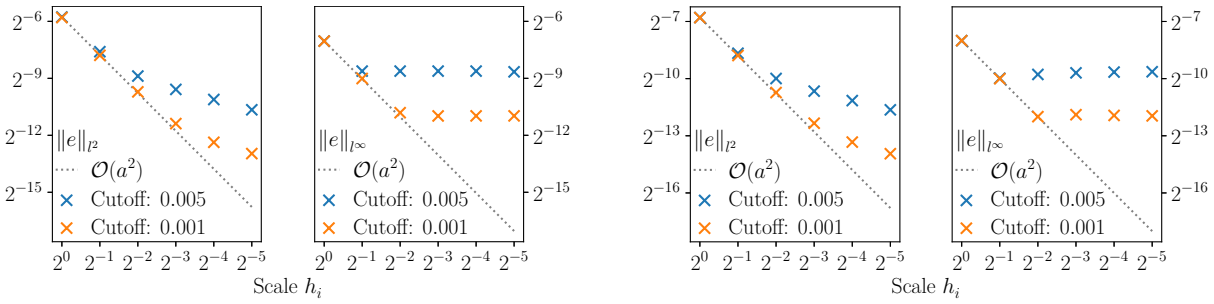
This systematic error can be controlled by adjusting the cutoff, as visible in fig. 7.2. Here, $C = 1$ again and the Gaussian pulse is sampled cell-averaged. In this configuration we expect that the pulse profile is exactly preserved. However, because we are comparing to analytic values that are sampled cell-centered, errors are generated that scale with $\mathcal{O}(a^2)$ (see appendix A). Directly comparing the results for two different cutoffs demonstrates that for the lower cutoff, the errors follow this rate up to smaller scales. The deviations at small scales are caused by two values at the edges of the pulse that do not follow the error profile (fig. 7.2a) and jump to higher values.

For a Gaussian, cell-averaged initialization produces errors that behave as expected. Unsurprisingly, the triangular pulse in fig. 7.3 is only affected slightly by the averaging error. Indeed, if we average a linear function's values from different x-positions, it will equal the value of the function at the averaged x. There are only three distinct errors for triangular pulses in fig. 7.3b, regardless of the scale. Two lie at the edge with their relative errors of -1 indicating that the cell-centered value for these cells is already zero. The third one corresponds to the value at the peak. The scaling rates for both error norms are found to be worse than the target rate, but a reason could not be determined.

7.2 Diagonal borders

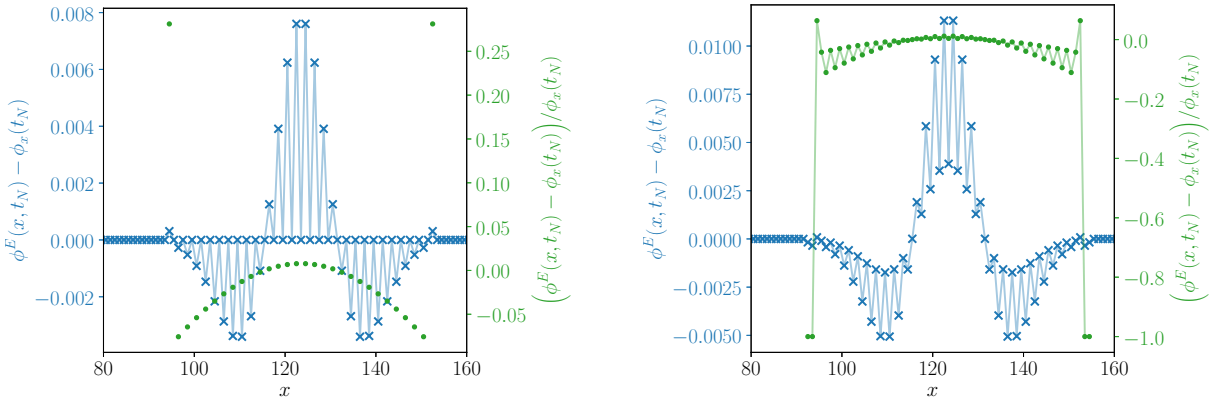
The numerical data in fig. 7.4 reveal that errors for both DCTF eqs. (5.13) to (5.14) and DFTC eqs. (5.10) to (5.12), scale with the targeted rate (gray line). Reducing the cutoff improves scaling for smaller h_i. However, deviations from the target rate are already visible at $h_i = 2^{-1}$ with the higher value of the cutoff. For these diagonal border refinement schemes, it is therefore more beneficial to reduce the cutoff rather than the lattice scale. The norms of all tested pulse profiles are conserved within machine precision limits.

The error profiles for Gaussian pulses at DFTC are similar to fig. 7.2a. This is expected because eqs. (5.10) to (5.12) are based on averaging. However, for DCTF the error profiles differ and are given in fig. 7.5 for both of the initialization methods. First, consider the cell-centered



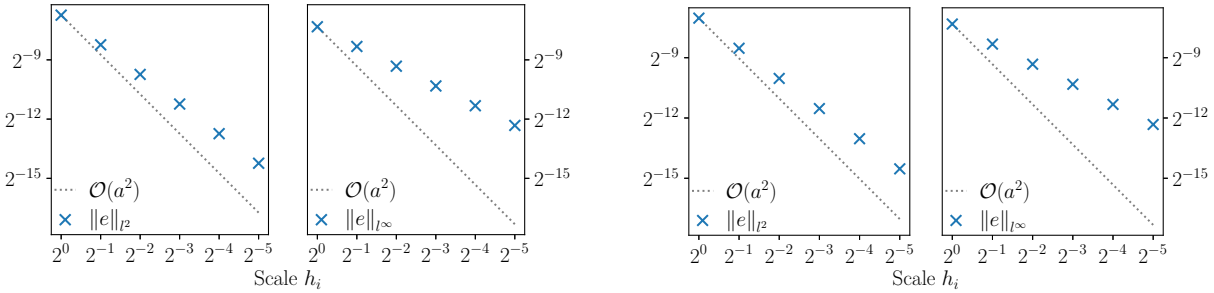
(a) DCTF. (b) DFTC.

Figure 7.4: Error scaling for diagonal borders. The pulse is sampled cell-centered from a Gaussian with $\sigma = 8$. (a): Using eqs. (5.13) to (5.14). (b): Using eqs. (5.10) to (5.12).



(a) Cell-centered initialization. (b) Cell-averaged initialization.

Figure 7.5: Comparison of the error profiles of the initialization schemes. Results for a single DCTF transition using eqs. (5.13) to (5.14). The pulse is sampled from a Gaussian with $\sigma = 8$ and cutoff at 0.001. The simulation scale $h_i = 2^0$. (b): Cell-averaging adds errors to (a).



(a) DCTF. (b) DFTC.

Figure 7.6: Same as fig. 7.4, but the pulse has a triangular shape and is sampled identical to fig. 7.3. Both error norms scale worse than $p = 2$. (a): Using eqs. (5.13) to (5.14). (b): Using eqs. (5.10) to (5.12).

results (fig. 7.5a). Every other error value on the fine side turns out to be exact so that the values of the profile of the relative error are disconnected. If we use cell-averaged sampling instead (fig. 7.5b), these exact values obtain an error. The data points can be visually separated into two interwoven distributions. The shape of the profile from fig. 7.5a reappears. Additionally, a second curve with identical features and smaller magnitude takes the place of the previously exact values. Adding the averaging error of fig. 7.2a to fig. 7.5a falls short of the observed error magnitude for the second profile in fig. 7.5b. Therefore, more errors are introduced by cell-averaged initialization and DCTF refinement.

The errors for triangular pulses at DCTF borders vanish for cell-centered sampling and for cell-averaging appear as single values at the center and edges of the pulse. For DFTC and cell-centering, the three values produced by averaging return. Changing to cell-averaging adds more error values around the center and edges. Because the majority of the values are still exact for both transition types at diagonal borders, the error scaling rates (fig. 7.6) are determined by the single error values and are worse than the target rate.

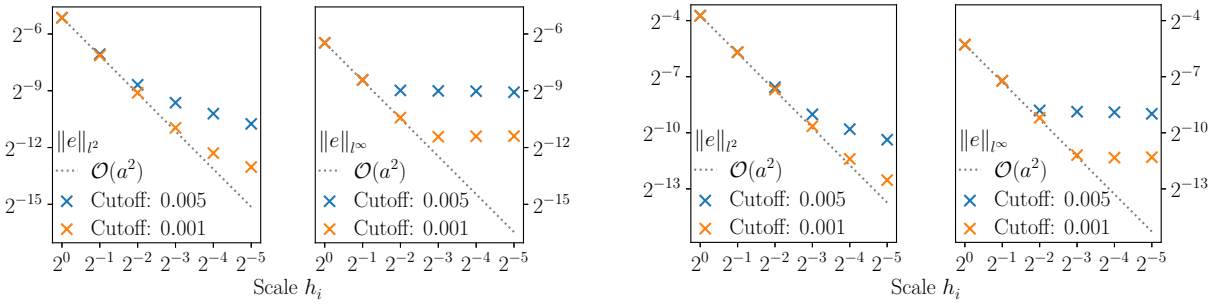
7.2.1 Repeated diagonal borders

Setups with repeated borders are designed to complete the discussion about changing the refinement direction at a single border from chapter 6. Here, a series of CTF and FTC transitions are chained together. Strictly speaking, this setup is not the same as using the refinement results of a scheme as the initial conditions of the flipped transition. Still, when alternating fine and coarse regions, the consistency of the refinement scheme can be analyzed. Two different configurations are possible. A coarse region can be embedded in finer regions leading to DFTC + DCTF or the order can be swapped for fine region inserts.

The error scaling plots for Gaussians and one or three inserts are given in figs. 7.7 and 7.8. All the results agree with the scaling rate $\mathcal{O}(a^2)$. Still, increasing the number of inserts leads to larger error magnitudes. For three inserted regions, the errors are roughly multiplied by a factor of 3 when compared to only one insert. Of the two border combinations, coarse inserts are expected to produce an error. Unsurprisingly, the previously exact values for cell-centered sampling at DCTF transitions (fig. 7.5a) are no longer obtained. The error profile in this case resembles cell averaging and can be explained by remembering that at the first DFTC border, the refinement scheme eqs. (5.10) to (5.12) use averaging.

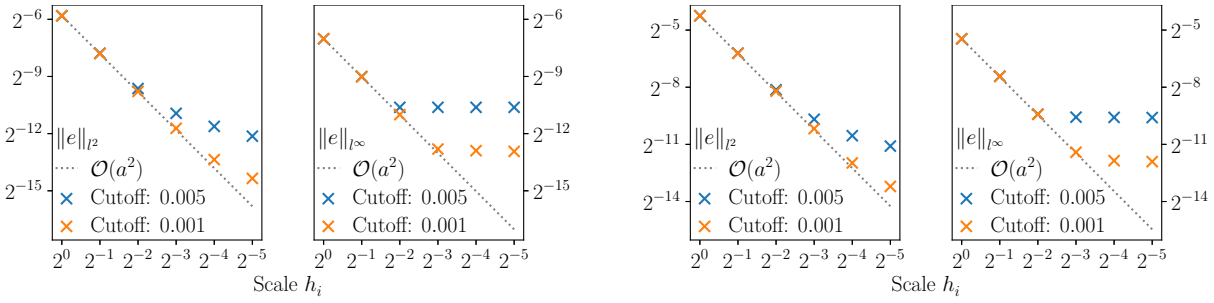
Fine inserts, on the other hand, are crucial for arguing consistency. It is in those configurations that missing values are supplied by DCTF refinement. If processing the result of a coarse-to-fine transition at the flipped DFTC border does not return to the initial pulse, the refinement scheme introduces modifications that cannot be reverted within the scheme. In that case, DCTF will not be consistent with DFTC and the new values generated by DCTF also propagate to the coarse side. However, because the observed scaling rate follows the target value, consistency is upheld within controllable error limits.

The error scaling rates for triangular pulses are again dominated by single error values and are worse than the target rates similar to fig. 7.6. These values are visible in fig. 7.9, but still outnumbered by the exact values. Increasing the number of inserted regions adds errors to values around the three problematic points for triangular pulses, the center and edges. Otherwise, the behavior is similar to Gaussians.



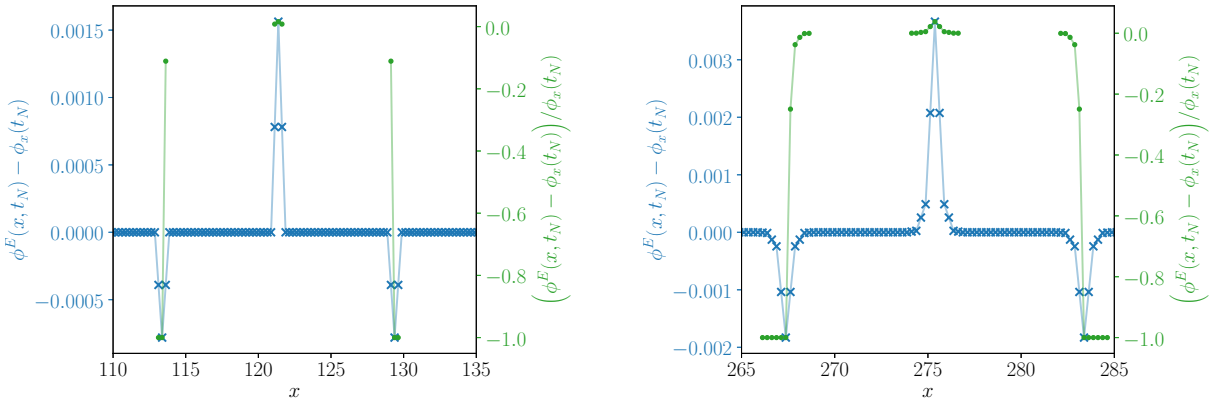
(a) DFTC + DCTF. (b) DFTC + 2×(DCTF + DFTC) + DCTF.

Figure 7.7: Error scaling for repeated diagonal border transitions. The lattice has stripes of coarse regions embedded in finer regions. The pulse is sampled cell-centered from a Gaussian with $\sigma = 8$. (a): One inserted coarse region stripe. (b): Three inserted coarse region stripes.



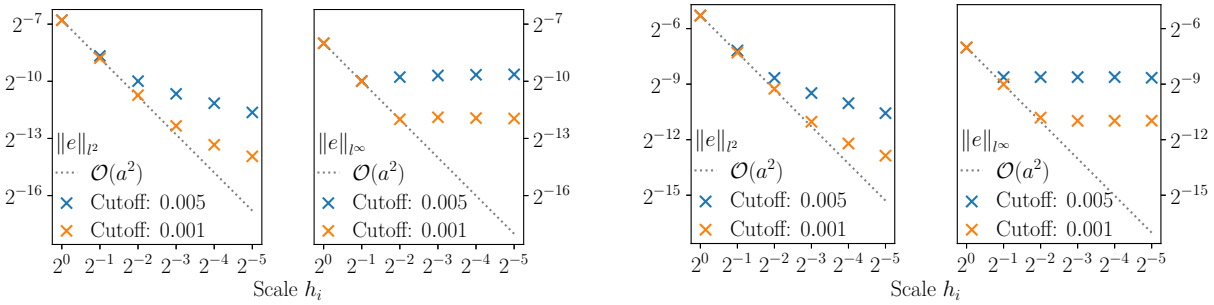
(a) DCTF + DFTC. (b) DCTF + 2×(DFTC + DCTF) + DFTC.

Figure 7.8: Same as fig. 7.7, but for fine inserted regions. (a): One inserted fine region stripe. (b): Three inserted fine region stripes.



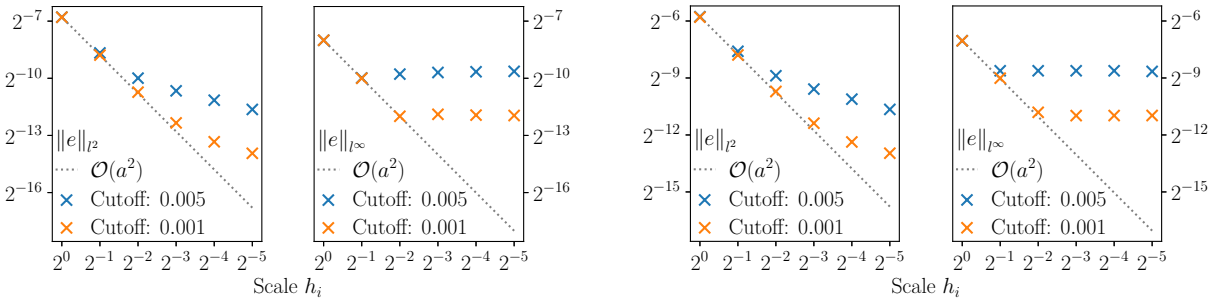
(a) DFTC + DCTF. (b) DFTC + 2×(DCTF + DFTC) + DCTF.

Figure 7.9: Error profiles for repeated diagonal border transitions. The lattice has stripes of coarse regions embedded in finer regions. The pulse is sampled cell-centered from a triangular pulse that is 16 x-units wide with its peak at 0.1. The simulation scale $h_i = 2^{-2}$. (b): For each added stripe more of the previously exact values become erroneous.



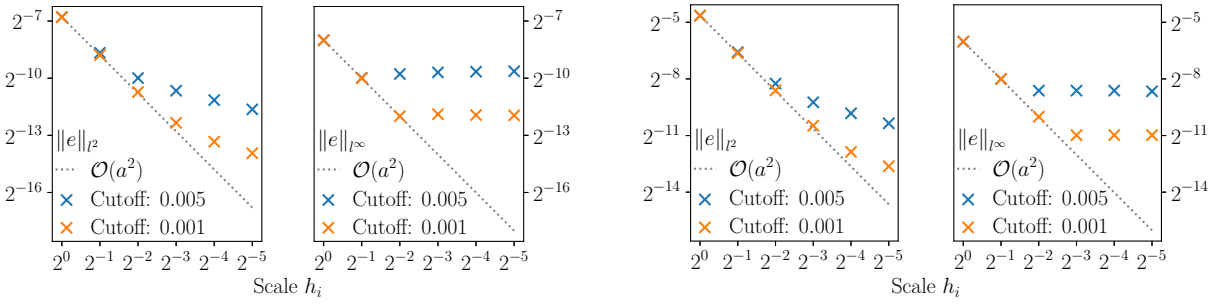
(a) SFTC. (b) SCTF.

Figure 7.10: Error scaling for straight borders using eqs. (5.15) to (5.17). The pulse is sampled cell-centered from a Gaussian with $\sigma = 8$.



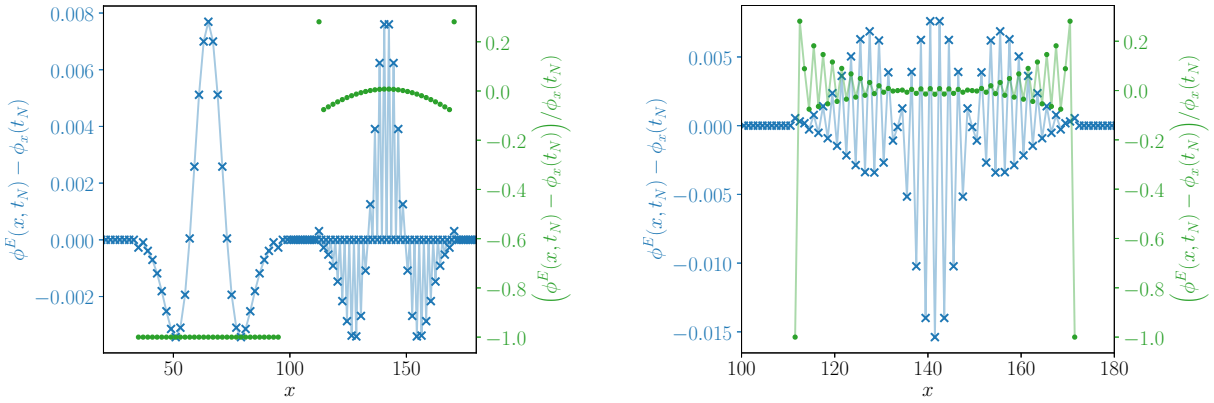
(a) SFTC. (b) SCTF.

Figure 7.11: Same as fig. 7.10, but using eqs. (5.20).



(a) SFTC. (b) SCTF.

Figure 7.12: Same as fig. 7.10, but using eqs. (5.18).



(a) SCTF eqs. (5.15) to (5.17).

(b) SCTF eqs. (5.18).

Figure 7.13: Error profiles for SCTF at $x = 100$. The pulse is sampled cell-centered from a Gaussian with $\sigma = 8$ and cutoff at 0.001. The simulation scale $h_i = 2^0$. (a): Equations (5.15) to (5.17) are similar to fig. 7.5a on the fine side but have a reflection on the coarse side. (b): Equations (5.18) differ by an interwoven curve with opposite sign from fig. 7.5a.

7.3 Straight borders

The most promising schemes from section 5.3 are tested. These are eqs. (5.15) to (5.17) using the unmodified diagonal differences, eqs. (5.20) that correct the averaging of fine cells for CTF transitions and eqs. (5.18) that correct the update equation of one fine border cell. All of these schemes' errors scale with the target rate for Gaussians (figs. 7.10 to 7.12) and all tested pulse profiles' norms are conserved. Also, at the SFTC border, all schemes perform identically to the DFTC refinement for both Gaussian and triangular pulses respectively.

The situation for the SCTF border is depicted in fig. 7.13. By design, the scheme using eqs. (5.15) to (5.17) (fig. 7.13a) reproduces the same error profile on the fine side as DCTF in fig. 7.5a. It is based on the same refinement equations. As discussed in section 6.2, however, it also generates a reflection on the coarse side. The magnitude of the reflected values can be directly read off of fig. 7.13a and is comparable to the error of the transitioned pulse. Similarly, because the combined error scales with the target rate, the reflection vanishes with $\mathcal{O}(a^2)$. In the case of eqs. (5.20), there is no reflection and the results are identical to DCTF. Both of these straight border schemes are affected by cell-averaged sampling the same way as diagonal borders so that the previously exact values on the fine side obtain an error together with overall larger error magnitudes.

Figure 7.13b is focused only on the coarse side of the SCTF border because there is no reflection to plot for this scheme. Two interwoven curves can be identified. In addition to a profile that has similar features to the other schemes, another profile that is mirrored at the x-axis appears. This new error profile has a negative, center peak with positive values on the sides. It is also larger in magnitude than the other profile. Interestingly, this discrepancy in magnitude is removed when changing to cell-averaged sampling.

For cell-centered triangular pulses at SCTF, eqs. (5.20) produce exact results. Equations (5.15) to (5.17) add a reflection with three discrete values that correspond to the center and edges of the triangular pulse where the center has negative and the edges positive values. This shape is mirrored at the x-axis compared to previous errors of transitioned triangular pulses. Conversely, eqs. (5.18) result in errors for the transitioned pulse and the error profile follows the reflection error of the last scheme. All schemes acquire errors at the center and edges when changing to cell-averaged sampling, similar to the case for diagonal borders. Hence, for

eqs. (5.18) that already had errors with opposite signs around these positions, oscillations of positive and negative errors appear over a very small range. The scaling rates for the errors of all schemes in line with previous results for triangular pulses.

7.3.1 Repeated straight borders

In analogy to diagonal borders, the results for three inserted regions with straight borders for both possible configurations and Gaussians are presented in figs. 7.14 to 7.16. The scaling rates for all schemes agree with the targeted value and the norms of the pulses are conserved. It has to be pointed out, though, that the error magnitudes for the scheme eqs. (5.18) differ from the other schemes. The data for coarse region inserts has smaller error values that are identical to the case with only one coarse insert. Additionally, for fine inserts the values are exact. The reason for these results was already discussed in section 6.2.2 and can be traced back to the fact that this scheme is preserves refinement direction symmetry for combinations of SCTF + SFTC in that particular order. If the order is reversed, errors are only generated at the first SFTC and last SCTF borders and all transition pairs in between are exact.

7.4 Initialization of ghost cells at borders

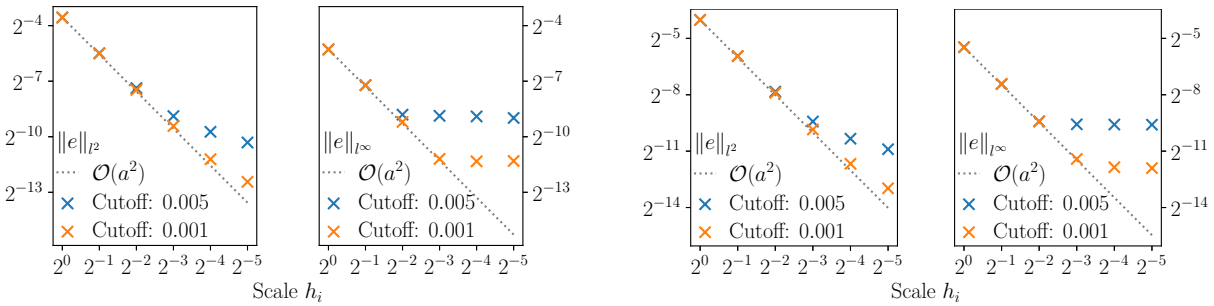
The data presented until now was generated from setups where the initial pulse is spaced out far enough from any lattice borders so that interactions with the borders only happen at later times. This approach allows to separate the influence of the refinement schemes from complications with initial conditions. In this section, we will focus only on setups where the initial pulses are deliberately placed in such a way that they cross the border. Then, the ghost cells for the refinement scheme of the border need to be populated with initial values. These values are also sampled according to the same method that is chosen for the standard lattice sites.

In the case of diagonal borders, we distinguish two configurations. The border can either be parallel or normal to the propagation of the pulse. DCTF parallel borders prove to be harmless. However, DFTC parallel borders generate oscillating error values on the coarse side behind a Gaussian that is sampled cell-centered (fig. 7.17a). Changing to cell-averaging in fig. 7.17b removes the oscillations but adds the averaging error to the previously exact pulse. The reason behind the oscillations is that because the coarse ghost cells are also initialized with cell-centered values, they are inconsistent with later ghost cells that are calculated via averaging in the scheme eqs. (5.10) to (5.12). Hence, cell-averaged sampling restores consistency.

The situation for normal borders is worse. Here, DCTF produces oscillating errors for both sampling methods. The initial conditions for the standard lattice sites are not compatible with results from CTF refinement. Still, DFTC is harmless because the ghost cells never propagate back to the coarse lattice sites. Nonetheless, all initialization errors scale with the target rate.

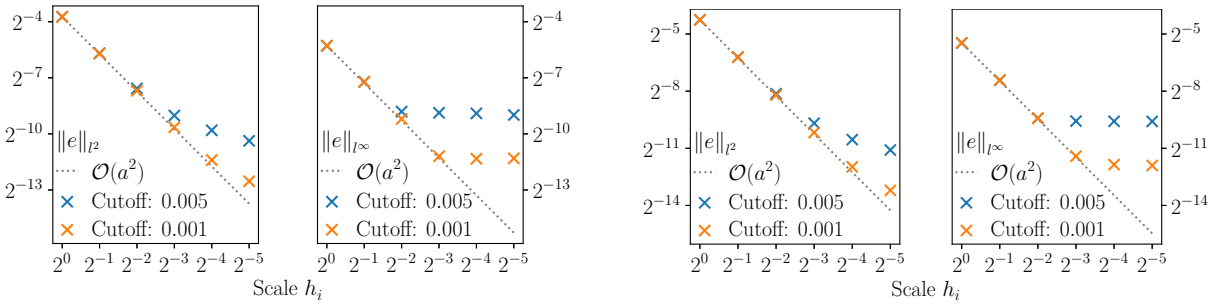
In the case of straight borders, the pulse is always moving away from the border. Thus, for Gaussians, the situation is similar to diagonal borders with normal propagation. SFTC is harmless for eqs. (5.15) to (5.17) and eqs. (5.20), but eqs. (5.18) produce oscillating errors behind the pulse for cell-centered sampling. Problems arise at SCTF where all schemes generate oscillations behind the pulse and even cell-averaged sampling does no longer cure them.

The results for triangular pulses reveal that no scheme produces oscillating errors for cell-centered initialization. In the case of cell-averaged sampling, however, all schemes obtain such an error for the SCTF border, contrary to all previous findings. It seems that the erroneous initial values at the edges and center for cell-averaging are enough to trigger the error because these values are the only ones that change for triangular pulses between the sampling methods.



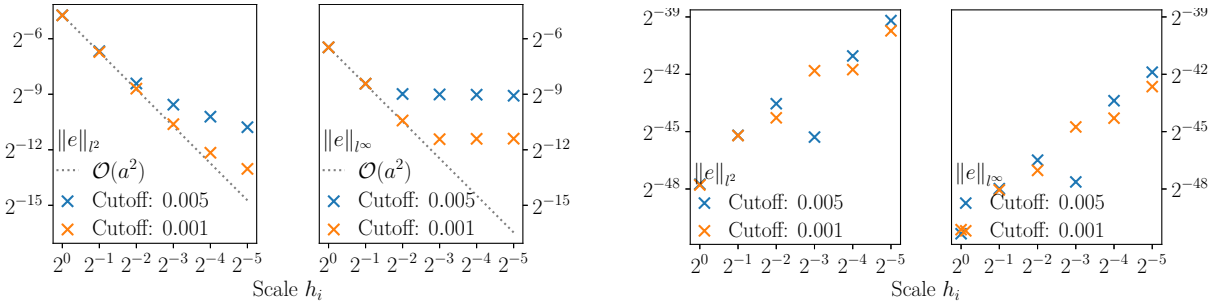
(a) SFTC + 2×(SCTF + SFTC) + SCTF. (b) SCTF + 2×(SFTC + SCTF) + SFTC.

Figure 7.14: Error scaling for inserted region stripes with straight borders using eqs. (5.15) to (5.17). The pulse is sampled cell-centered from a Gaussian with $\sigma = 8$. (a): Three inserted coarse region stripes. (b): Three inserted fine region stripes.



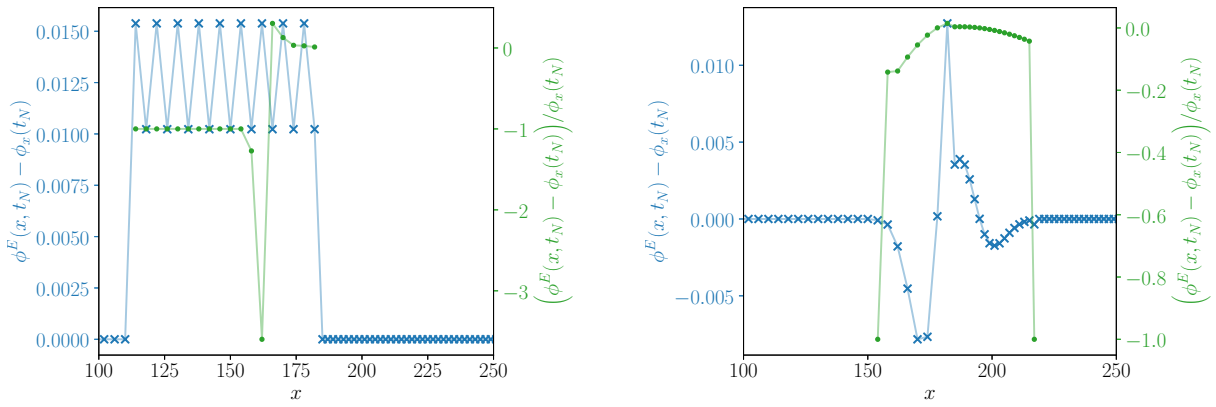
(a) SFTC + 2×(SCTF + SFTC) + SCTF. (b) SCTF + 2×(SFTC + SCTF) + SFTC.

Figure 7.15: Same as fig. 7.14, but using eqs. (5.20).



(a) SFTC + 2×(SCTF + SFTC) + SCTF. (b) SCTF + 2×(SFTC + SCTF) + SFTC.

Figure 7.16: Same as fig. 7.14, but using eqs. (5.18). (b): The error values are zero within machine precision limits.



(a) Cell-centered sampling.

(b) Cell-averaged sampling.

Figure 7.17: Error profiles for a DFTC border. The pulse is sampled from a Gaussian with $\sigma = 8$ and cutoff at 0.001 that is centered at the border. The pulse propagates parallel to the border. For the plotted time step the border is at $x = 187$. The simulation scale $h_i = 2^0$.

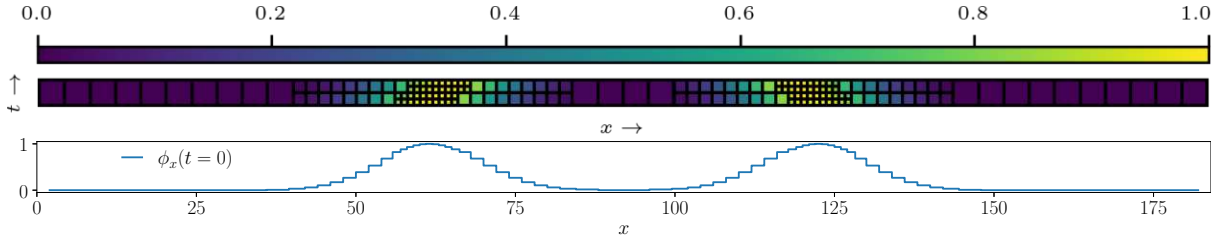
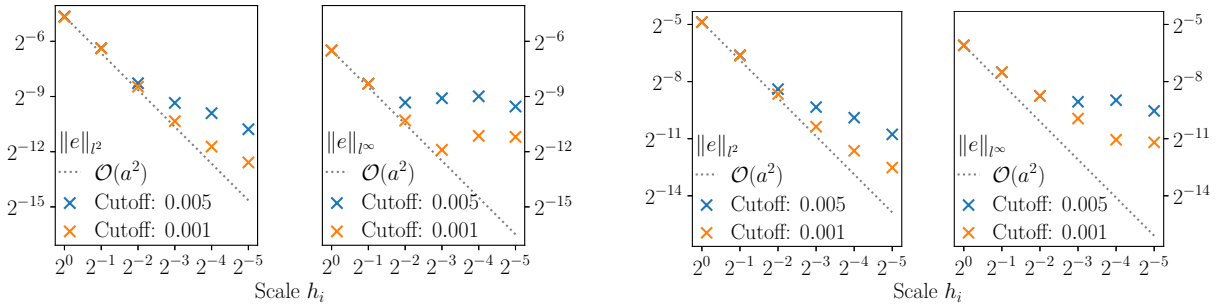


Figure 7.18: Initialization of a collision lattice setup with Gaussians. Two Gaussians with with $\sigma = 8$ are sampled cell-averaged and placed like shown.



(a) Equations (5.32) to (5.33).

(b) Equations (5.30) to (5.31).

Figure 7.19: Error scaling for the collision lattice setup of fig. 7.18.

7.5 Collision lattices

A similar collision lattice as in fig. 3.3b is used that combines regions with three different lattice spacings. The starting time slices with values and the initial pulse profile are visible in fig. 7.18. Two Gaussians are placed centered in the mesh kernel regions with a σ-value large enough so that the values spill over to the neighboring coarser regions. The lattice only has straight double-cell cusps and for the step at the cusp, eqs. (5.42) to (5.44) are used that are based on diagonal border CTF refinement. For the coarse-to-fine cusps, the two schemes compared in section 6.3 are tested. All other borders make use of their respective schemes from chapter 5.

The error scaling rates are separated into the two CCTF schemes and are depicted in fig. 7.19. The variant with eqs. (5.32) to (5.33) in fig. 7.19a meets the target scaling rate. For fig. 7.19b with dedicated eqs. (5.30) to (5.31), the l^2 norm also agrees with the target rate but the l^∞ norm shows worse scaling. The error values responsible for this seem to be isolated and do not affect the l^2 norm. The previous discussion comparing the two CCTF schemes in section 6.3 concludes that the dedicated equations are superior because they eliminate artifacts at the cusp and yield the same pulse profile on the fine side. From the current results, we see that in this particular case, the artifacts do not hinder the error scaling rate.

The pulse profile norm (eq. (5.1)) on the final time slice does not conserve its initial value. The relative error is $\sim 0.1\%$. As discussed in chapter 6, the reason is that the conservation of the norm is violated in general for time steps where values are actively refined at borders. In the current case, because the initial pulses are spread out across multiple differently resolved regions on the lattice, there is refinement being done for all time slices. The norm will only return to its conserved value after refinement transitions are completed, as illustrated by the results of the previous sections.

The collision lattice setup contains all border types that are analyzed in the previous sections. We expect that the combined results for these schemes, as discussed in this section, can be deduced and will follow the individual schemes' behaviors. Indeed, we are able to demonstrate that fig. 7.19 is not worse than the scheme with the worst scaling.

Chapter 8

Conclusion

In this thesis, computational methods for solving real-time lattice field theories in 1+1D are studied. Calculating the time evolution numerically requires to discretize spacetime on a lattice. Increasing the numerical accuracy of the results is typically achieved by reducing the lattice spacings and working with a larger number of sites. A common problem with existing methods is that the entire spacetime domain is discretized with the same lattice spacing, although the higher accuracy is only beneficial for a small region. This limits the performance and domain size of these methods because more memory is needed and more calculations are performed than would be necessary.

To solve this issue, the two software packages *mrlattice* and *sam* are developed. With *mrlattice*, it is possible to generate adaptive 1+1D lattices. In contrast to existing discretization methods, *mrlattice* allows the combination of regions with different lattice spacings into one connected mesh. The idea is to reduce the lattice spacing only in those regions of the domain where more accuracy is desired. As a special case, the lattice constants of neighboring regions are limited to only differ by a factor of 2. Complex meshes are built using the two fundamental region border types: straight borders that stay at the same spatial position for all times and diagonal borders that move one lattice cell in either spatial direction for each time step.

sam is a solver for the time evolution of discrete field theories in 1+1D that is designed to work with the meshes generated by *mrlattice*. In particular, the free, massless scalar field is studied. The discrete lattice is configured such that it adapts to the solutions of the theory, and the regions with the smallest lattice spacing are placed where the solution is expected to vary the most. In this sense, the adaptive mesh has to be supplied a priori and does not change during the calculations of the solver.

The time evolution is based on an update stencil that is derived from the discretized equations of motion (EOM). However, at the borders between lattice regions with different resolutions, the stencil can no longer be applied. For each border type, it is necessary to develop specialized refinement schemes that combine the discrete values from the regions into new update equations. The method used to derive these equations is based on two aspects: first, the optimal refinement outcome for a special solution of the EOM is determined and then this result is reconstructed using symmetries of the EOM and a conserved quantity of the theory. In the case of the free, massless scalar field, the chosen conserved quantity is the norm calculated as the sum of all values for one time slice. The symmetries of the EOM are exploited to relate diagonal differences of cells from multiple time slices within a region to analogous differences of cells from the neighboring region. Also, cells from 2×2 blocks in temporal and spatial directions are averaged to obtain a value for an overlapping coarser cell. The choice to fix the results for initially single-cell wide unit pulses is motivated by the observation that a certain number of these pulses can be used in a linear combination to reconstruct any desired discrete shape.

The numerical performance of all the resulting refinement schemes for a variety of border and cusp lattice configurations is evaluated. The initial conditions are sampled cell-centered or cell-averaged from Gaussian distributions and triangular pulses. Most importantly, the error that is introduced by the refinement procedure is proven to scale with the lattice spacing at the same rate that is obtained by discretizing the EOM alone. Additionally, persistent artifacts like reflections, checkerboard patterns or divergencies are effectively suppressed.

Furthermore, the equations based on the diagonal difference symmetry of the EOM are shown to be applicable for every coarse-to-fine refinement scheme. Although relying solely on these equations leads to minor artifacts, the scaling rate of the errors matches the rate of the discretized EOM. In contrast, equations that are specialized for a certain border type and do not produce artifacts deviate from the original diagonal differences.

The in-depth analysis of the individual refinement equations concludes that the averaging relation between coarse and fine cells can lead to inconsistencies. Straight borders require simultaneous coarse-to-fine and fine-to-coarse refinement and are only successful if either the average or the diagonal difference relations are modified. Averaging also affects the sampling of initial conditions for cells at borders.

Similarly, starting with the results of a coarse-to-fine transition and then applying fine-to-coarse refinement at the same border only leads back to the exact initial conditions in certain cases. The generated errors are an indication that refinement schemes introduce modifications to the time evolution that cannot be reverted when changing the direction of refinement.

The conservation of the special norm is violated for time steps where the solution is actively being refined at borders. Its value is only restored after the refinement process is complete. For complex lattice configurations, where the solution always stays dominant next to borders, this leads to the problem that the norm is not conserved exactly. Following up on the results of this thesis, it needs to be studied whether new refinement schemes can be found that conserve the norm at each time step.

Overall, the magnitude of the error depends strongly on the exact shape of the initial conditions. Profiles with small gradients across the refinement borders are beneficial. In those cases, the resolution of either region at the border is good enough to represent the details of the solution. Hence, the refinement process is not limited by features below the lattice resolution.

On the contrary, deriving the refinement equations based on single-cell unit pulses focuses on solutions that are insufficiently well represented by the discretized values. By design, those solutions have a large gradient. If the refinement schemes are able to deal with this limiting case, they are expected to be well-behaved in general. This statement is supported by the numerical results presented in this thesis. The connection of the generated errors with the gradients of the solutions could be formalized in a continuation of this discussion.

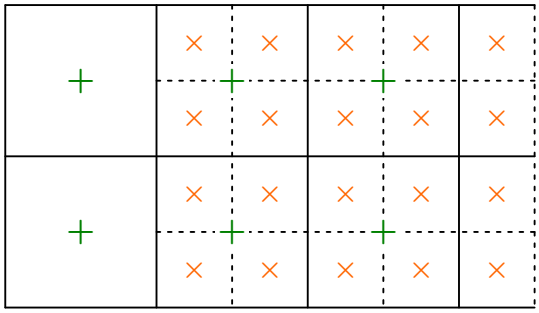
Other possible approaches to developing refinement schemes are to look at the polynomial expansion of general solutions. Instead of only one special solution, the ideal refinement outcome for a family of functions like the polynomials could be the starting point. Indeed, the results discussed for triangular pulses show that first-order polynomials behave differently under the same refinement schemes. A more physically motivated expansion in the modes of plane waves is also conceivable. The natural regulation of the lattice for high frequencies would determine the highest mode that presumably will behave similarly to single-cell pulses because it is again at the limit of the lattice resolution.

The work in this thesis can be used for many different problems. First, the developed refinement schemes can be applied to more complex theories in 1+1D. Of special interest are theories that can no longer be solved analytically. Also, extending the solver to 3+1D presents new technical challenges and will reveal the potential computational performance improvements obtainable from adaptive meshes. The same 1+1D refinement process can be reused in higher dimensional spacetime for refinement along one chosen dimension.

Appendix A

Error term of fine cell averages

A proof for the error term of eq. (4.5) is presented in the following. Given a scalar function $f(x^\mu)$ that depends on an N dimensional vector x^μ, the discretizations on the two differently resolved lattices



$$\times \in \Lambda^a = \left\{ x \,|\, x = \sum_\mu^N n_\mu \hat{a}^\mu, \quad n_\mu \in \mathbb{Z} \right\}, \tag{A.1}$$

$$+ \in \Lambda^b = \left\{ x \,|\, x = \sum_\mu^N n_\mu \hat{b}^\mu, \quad n_\mu \in \mathbb{Z} \right\} \tag{A.2}$$

with lattice spacings a^μ and b^μ are given by

$$f^a(x) : x \mapsto f(x) \quad \text{for } x \in \Lambda^a, \tag{A.3}$$

$$f^b(x) : x \mapsto f(x) \quad \text{for } x \in \Lambda^b. \tag{A.4}$$

The two lattices are disjoint $\Lambda^a \cap \Lambda^b = \{\}$ and $b = 2a$. The situation for $N = 2$ is depicted above. In order to approximate the values of f^b based on values of the finer resolved f^a, an average of the n next neighbors $\langle x \rangle \in \Lambda^a$ of a point x on Λ^b is used

$$f^b(x \in \Lambda^b) \approx \frac{1}{n} \sum_{y \in \langle x \rangle} f^a(y). \tag{A.5}$$

The next neighbors $\langle x \rangle$ can be written as

$$\langle x \rangle : \left\{ x^\mu + i_{k\mu} \frac{\hat{a}^\mu}{2}, \quad k \in \{1, 2\}, \mu \in \{0, \ldots, N-1\} \right\}, \tag{A.6}$$

where no sum is implied and $i_{k\mu}$ is $(1, -1)_k$ for each dimension μ and gives the direction to each neighbor. Recall the Taylor series for a scalar function of a vector valued argument around the point x

$$f(y) = f(x) + \sum_\mu^N (y-x)^\mu \partial_\mu f \Big|_x + \frac{1}{2} \sum_\mu^N \sum_\nu^N (y-x)^\mu (y-x)^\nu \partial_\mu \partial_\nu f \Big|_x + \mathcal{O}(|(y-x)|^3). \tag{A.7}$$

Here, the indices μ and ν for the arguments are moved outside the parentheses denoting the vector difference from the evaluation point y. The error term ε is identified as the difference

between the value of f^b and its approximation via values from f^a. By expanding each of the terms in the sum over the next neighbors in eq. (A.5) into Taylor series around $x \in \Lambda^b$ the error term becomes

$$\begin{aligned}\varepsilon &= f(x) - \frac{1}{n}\sum_{y\in\langle x\rangle} f^a(y) = f(x) - \frac{1}{n}\sum_{y\in\langle x\rangle} f(y) \\ &= f(x) - \frac{1}{n}\sum_{y\in\langle x\rangle}\left[f(x) + \sum_{\mu}^{N}(y-x)^\mu \partial_\mu f\Big|_x \right. \\ &\qquad \left. + \frac{1}{2}\sum_{\mu}^{N}\sum_{\nu}^{N}(y-x)^\mu (y-x)^\nu \partial_\mu\partial_\nu f\Big|_x + \mathcal{O}(a^3)\right].\end{aligned}$$

For the lattice of eqs. (A.1) and (A.2), the difference $(y-x)^\mu$ is either $+\hat{a}^\mu/2$ or $-\hat{a}^\mu/2$ so that for each neighbor k per dimension τ the expression $(y-x)^\mu_{k\tau} = i_{k\tau}(\hat{a}^\tau)^\mu/2$. Inserted into ε

$$\begin{aligned}\varepsilon = f(x) - f(x)\frac{1}{n}\sum_{y\in\langle x\rangle} 1 &- \frac{1}{n}\sum_{\mu}^{N}\sum_{k=1}^{2}\sum_{\tau}^{N} i_{k\tau}\frac{(\hat{a}^\tau)^\mu}{2}\partial_\mu f\Big|_x \\ &- \frac{1}{2n}\sum_{\mu}^{N}\sum_{\nu}^{N}\sum_{k=1}^{2}\sum_{\tau}^{N}(i_{k\tau}\frac{(\hat{a}^\tau)^\mu}{2})(i_{k\tau}\frac{(\hat{a}^\tau)^\nu}{2})\partial_\mu\partial_\nu f\Big|_x + \mathcal{O}(a^3).\end{aligned}$$

Using that there are n neighbors in $\langle x\rangle$ and $\sum_{k=1}^{2} i_{k\tau}\hat{a}^\tau = 0$ for each dimension, only terms of $\mathcal{O}(a^2)$ and higher orders in a remain

$$\begin{aligned}\varepsilon &= -\frac{1}{2n}\sum_{\mu}^{N}\sum_{\nu}^{N}\sum_{k=1}^{2}\sum_{\tau}^{N}\underbrace{i_{k\tau}i_{k\tau}}_{+1}\frac{(\hat{a}^\tau)^\mu}{2}\frac{(\hat{a}^\tau)^\nu}{2}\partial_\mu\partial_\nu f\Big|_x + \mathcal{O}(a^3) \\ &= -\frac{1}{n}\sum_{\mu}^{N}\sum_{\nu}^{N}\sum_{\tau}^{N}\frac{1}{4}\underbrace{(\hat{a}^\tau)^\mu(\hat{a}^\tau)^\nu}_{\delta_{\mu\nu}(a^\tau)^2}\partial_\mu\partial_\nu f\Big|_x + \mathcal{O}(a^3) \\ &= -\frac{1}{n}\sum_{\mu}^{N}\sum_{\tau}^{N}\left(\frac{a^\tau}{2}\right)^2\partial_\mu^2 f\Big|_x + \mathcal{O}(a^3) \\ &= -\frac{N}{n}\sum_{\mu}^{N}\left(\frac{a}{2}\right)^2\partial_\mu^2 f\Big|_x + \mathcal{O}(a^3).\end{aligned}$$

In the last step we used that the lattice spacings are equal for all dimensions τ. This proves that eq. (A.5) (and eq. (4.5)) is accurate up to $\mathcal{O}(a^2)$.

Appendix B

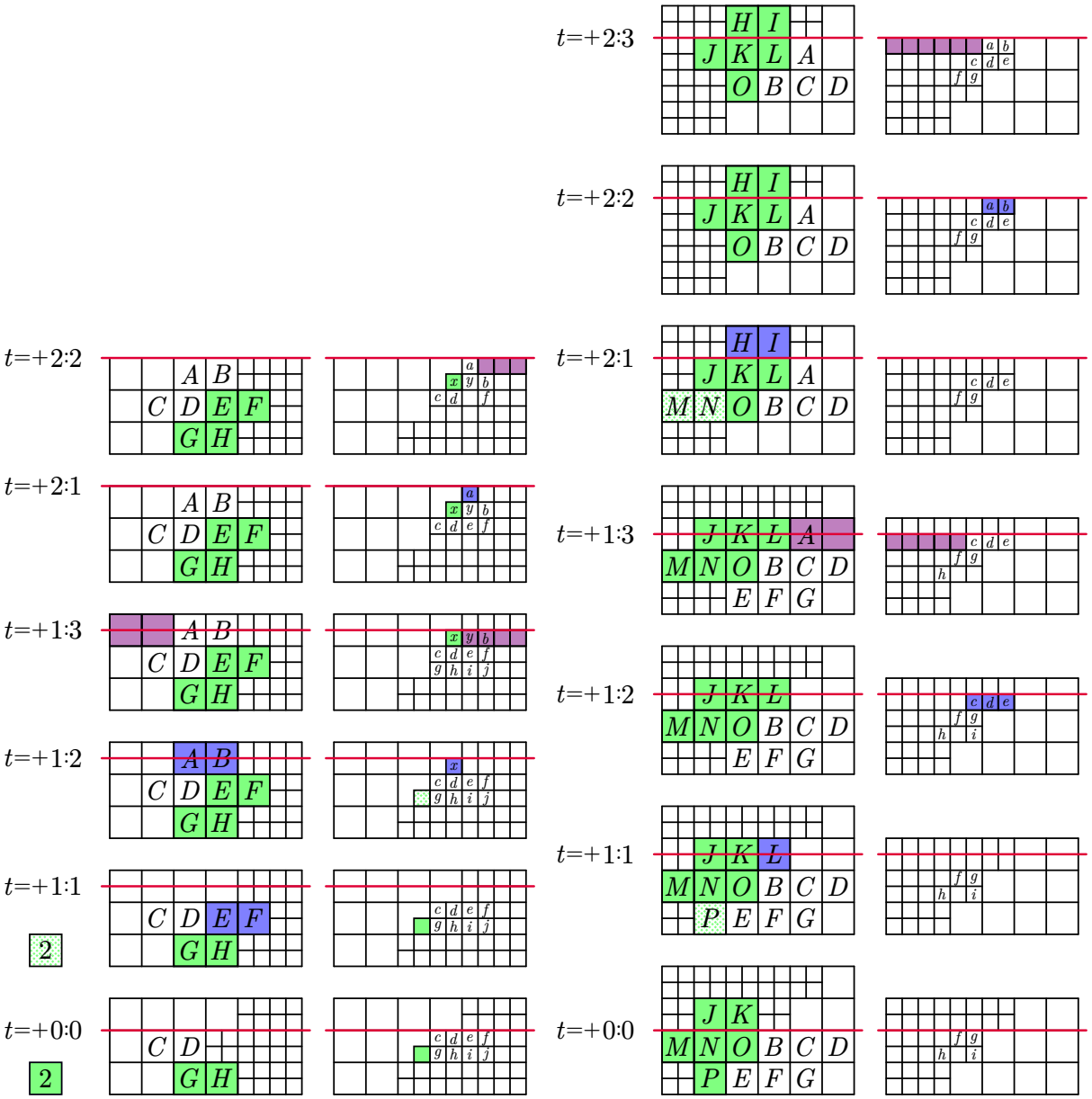
Flow graphs for refinement equations

The following graphs are visual representations of the flow of calculations discussed in chapter 5. Some border refinement types have multiple possibilities to perform the refinement. The panels are layed out from bottom to top with increasing time t. The time steps are split into ticks that advance the solver time and substeps that are required to order calculations within the same refinement step. The resulting annotations take the form $t = +\text{step} : \text{substep}$ and their numerical values are meant as an offset to the first panel and are not the global solver time. The horizontal red line indicates the current time step as a separator between the past and future. Each flow graph starts with a panel at $t = +0:0$ and ends with a panel that is identical to $t = +0:0$ in terms of the known and unknown values. Therefore, each figure represents a complete cycle that can be repeated for the duration of the time evolution.

In the figures below, a certain color coding is used to highlight what operations are performed at each step. Their meanings are as follows:

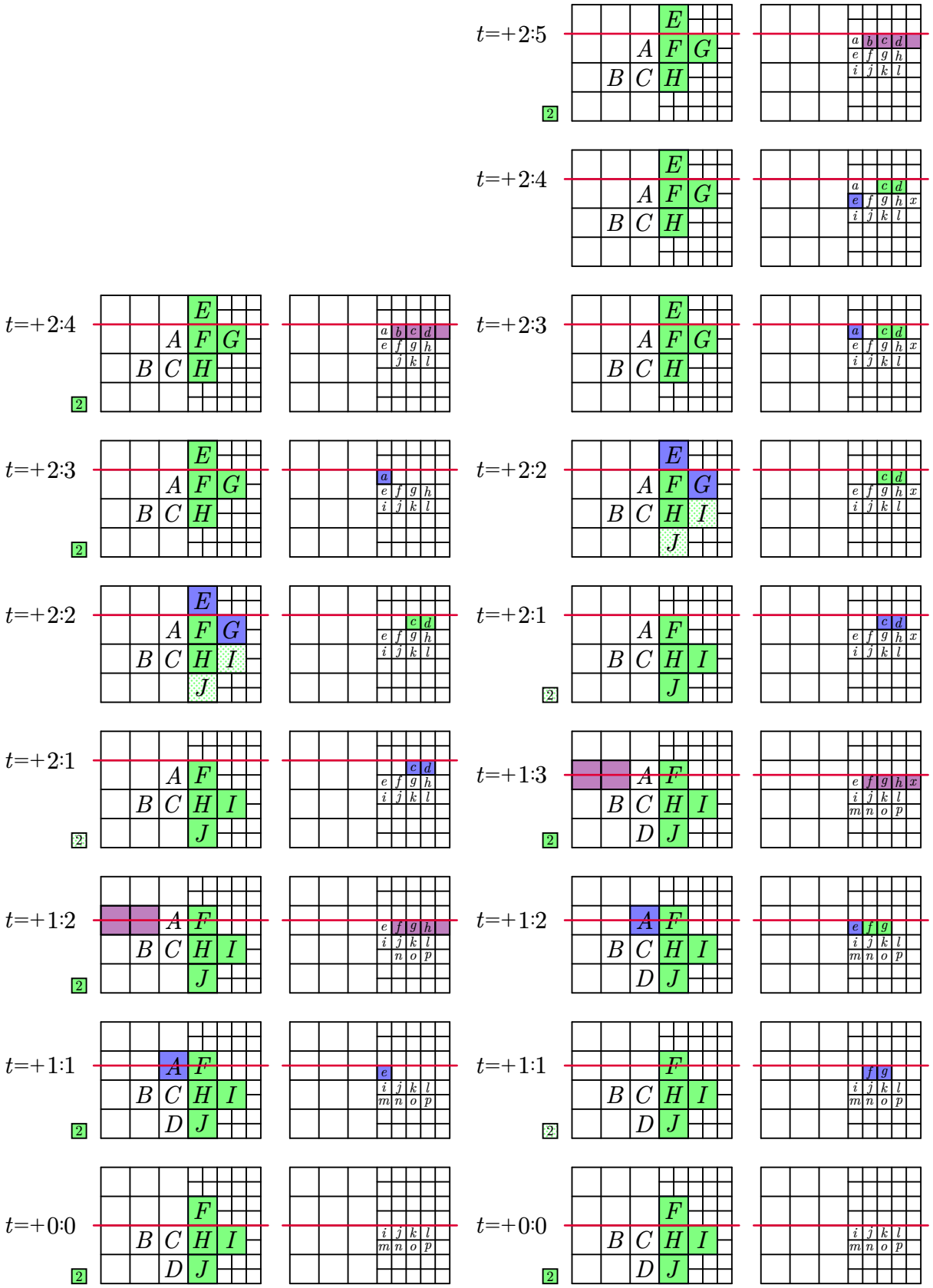
- Default, accessible cells stored in memory.
- Cells that are computed using the default update stencil of the solver.
- Cells that are computed by making use of special refinement equations.
- Ghost cells that were calculated previously and are now accessible.
- Ghost cells whose memory location is overwritten at this step.

Additionally, at each panel there may be extra ghost cells which are currently not in use but will be in future steps. These are positioned to the left of the actual panel underneath the time step indicator with the number of extra cells inside them. If this number is red, as it often is for straight cusps, it means that there are not enough extra cells available for that step and extra ghost cells need to be provided.



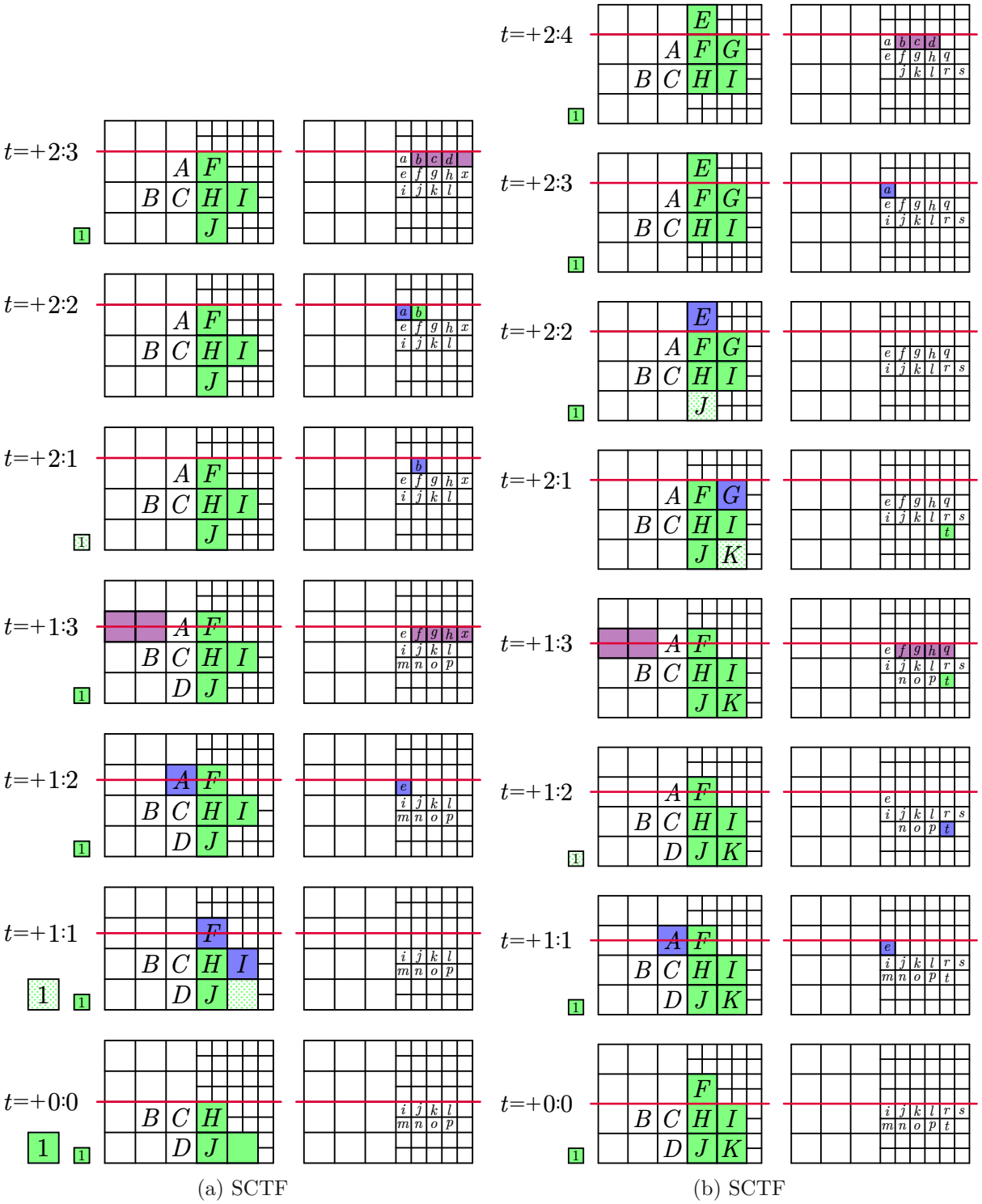
(a) DFTC (b) DCTF

Figure B.1: Diagonal border flow graphs for fine-to-coarse (a) with eqs. (5.10) to (5.12) and coarse-to-fine (b) with eqs. (5.13) to (5.14) from sections 5.1 and 5.2.



(a) SCTF (b) SCTF

Figure B.2: Straight border flow graphs for section 5.3. (a) for the variant that reuses diagonal border eqs. (5.15) to (5.17). (b) for dedicated eqs. (5.18) with e corrected.



(a) SCTF (b) SCTF

Figure B.3: Straight border flow graphs for section 5.3. (a) for the variant with dedicated eqs. (5.19) that correct a. (b) for dedicated eqs. (5.20) with G corrected.

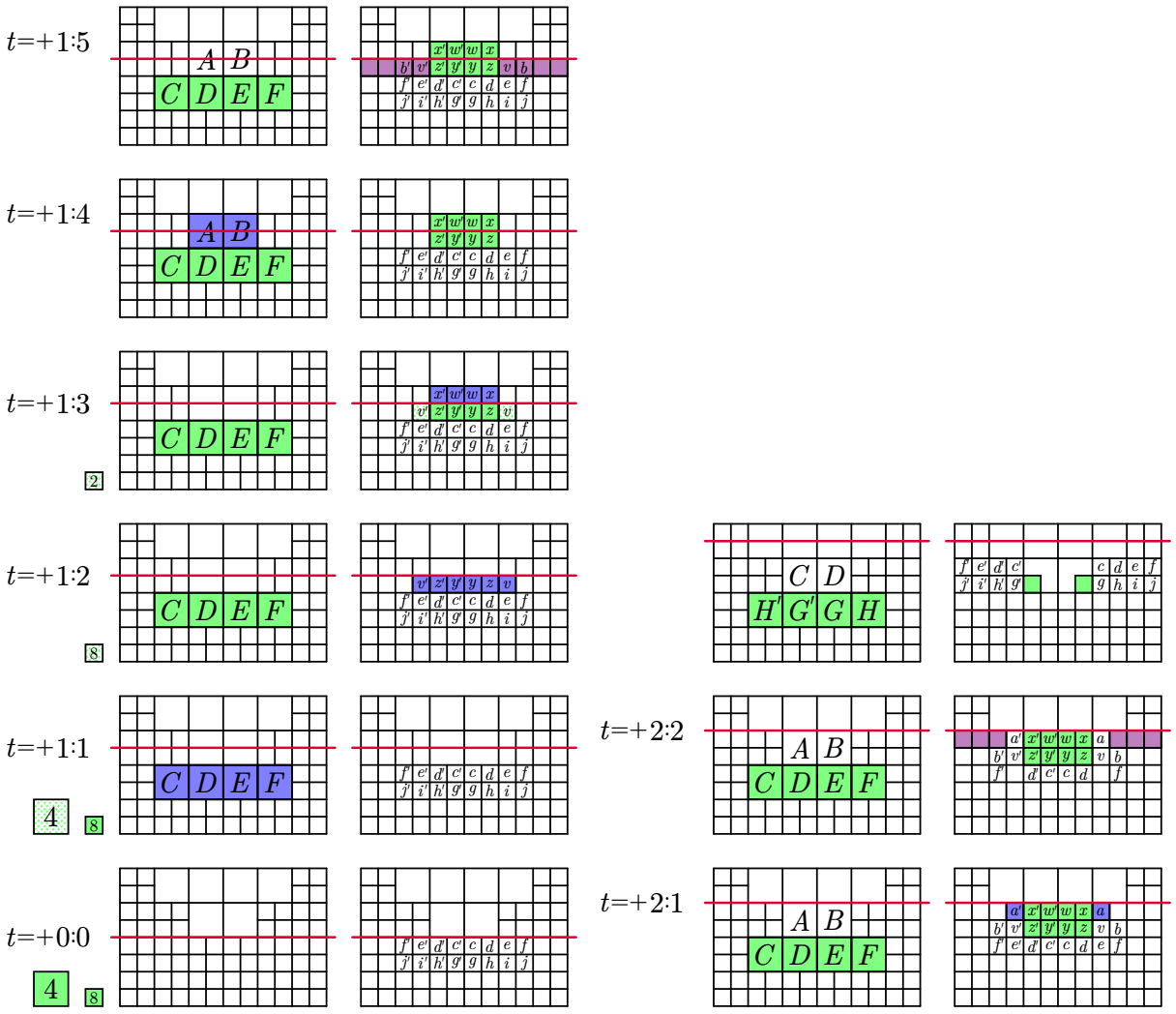
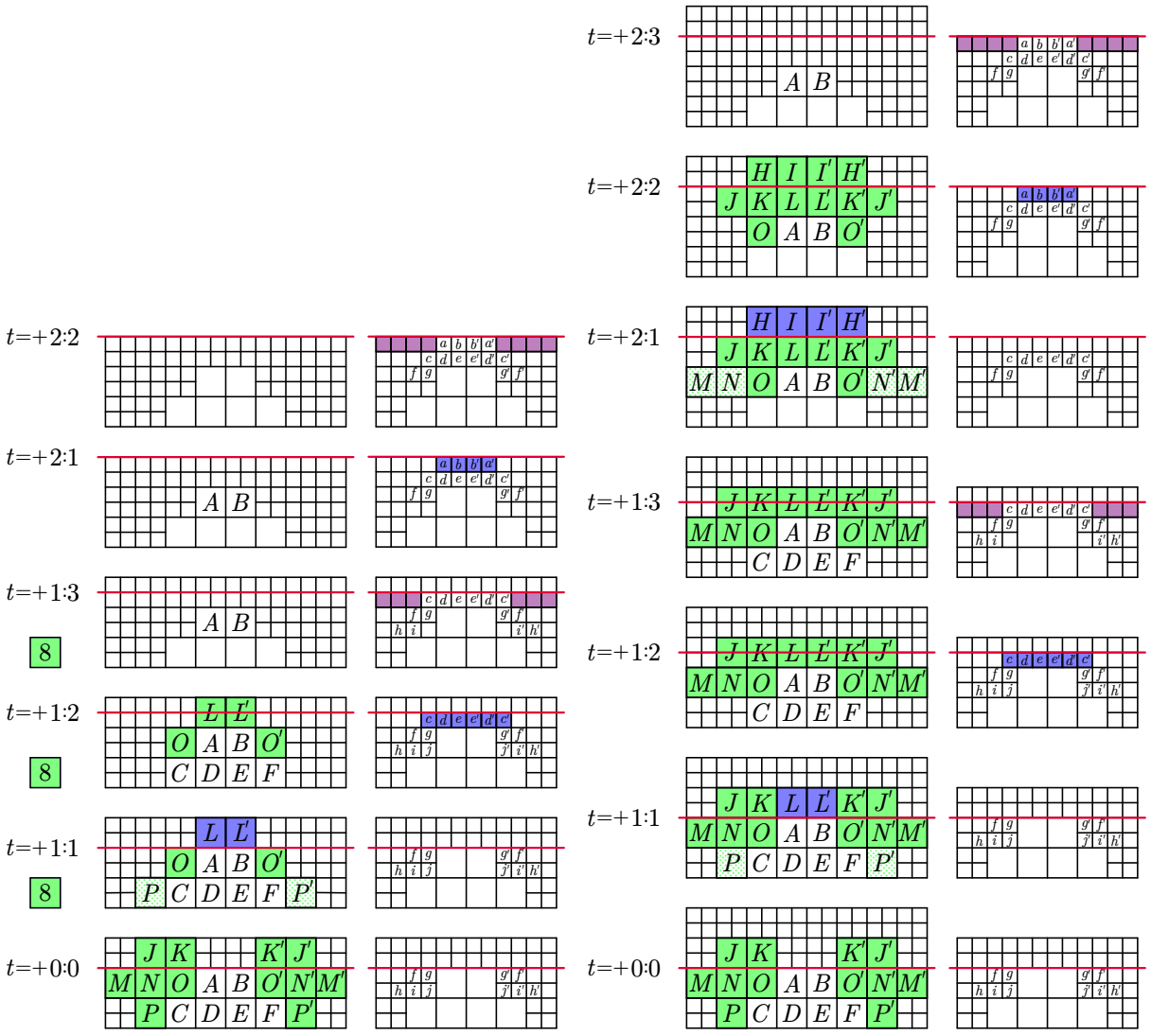


Figure B.4: Fine-to-coarse cusp (CFTC) flow graph for eqs. (5.25) to (5.27) from section 5.4.



(a) CCTF (b) CCTF

Figure B.5: Coarse-to-fine cusp flow graphs for section 5.5. (a) for dedicated equations and (b) for reusing diagonal border equations.

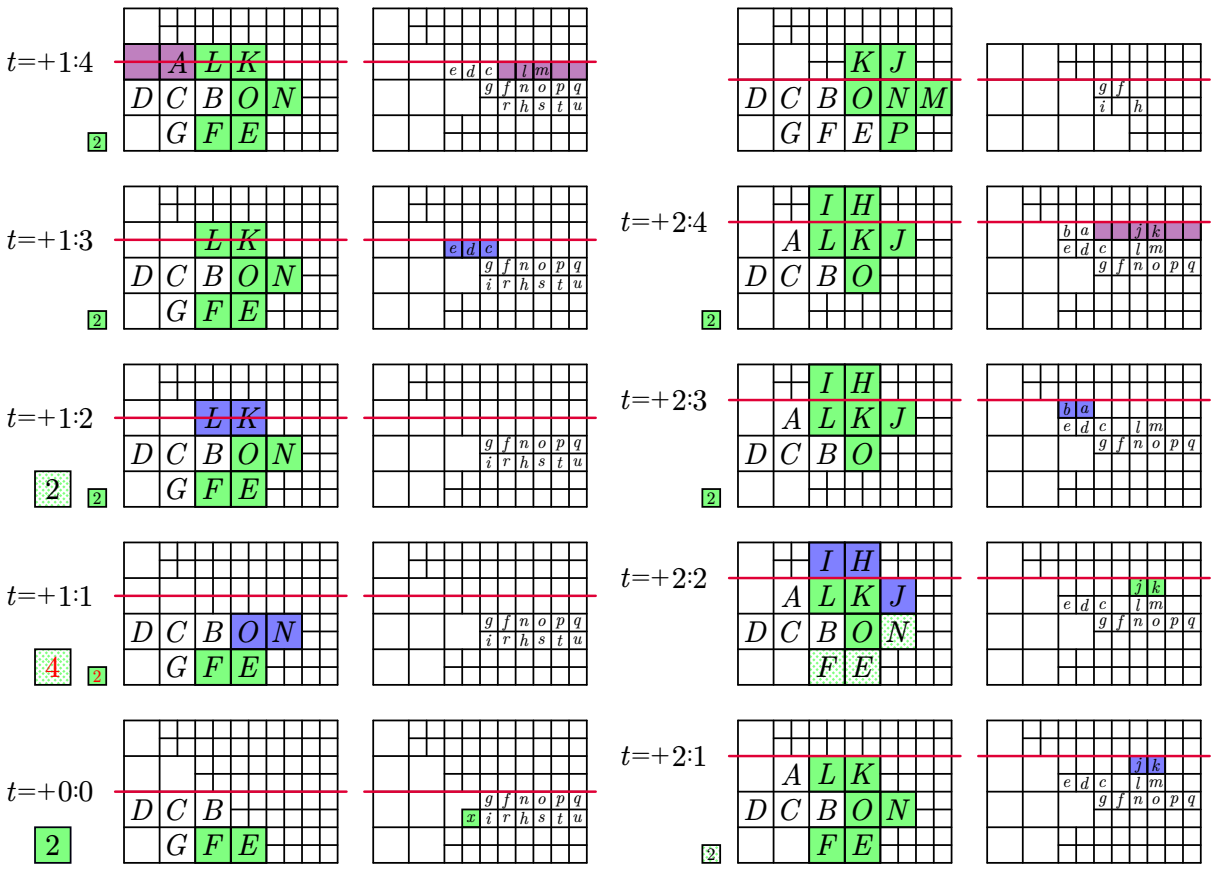


Figure B.6: Straight single-cell cusp flow graph for section 5.6.1.

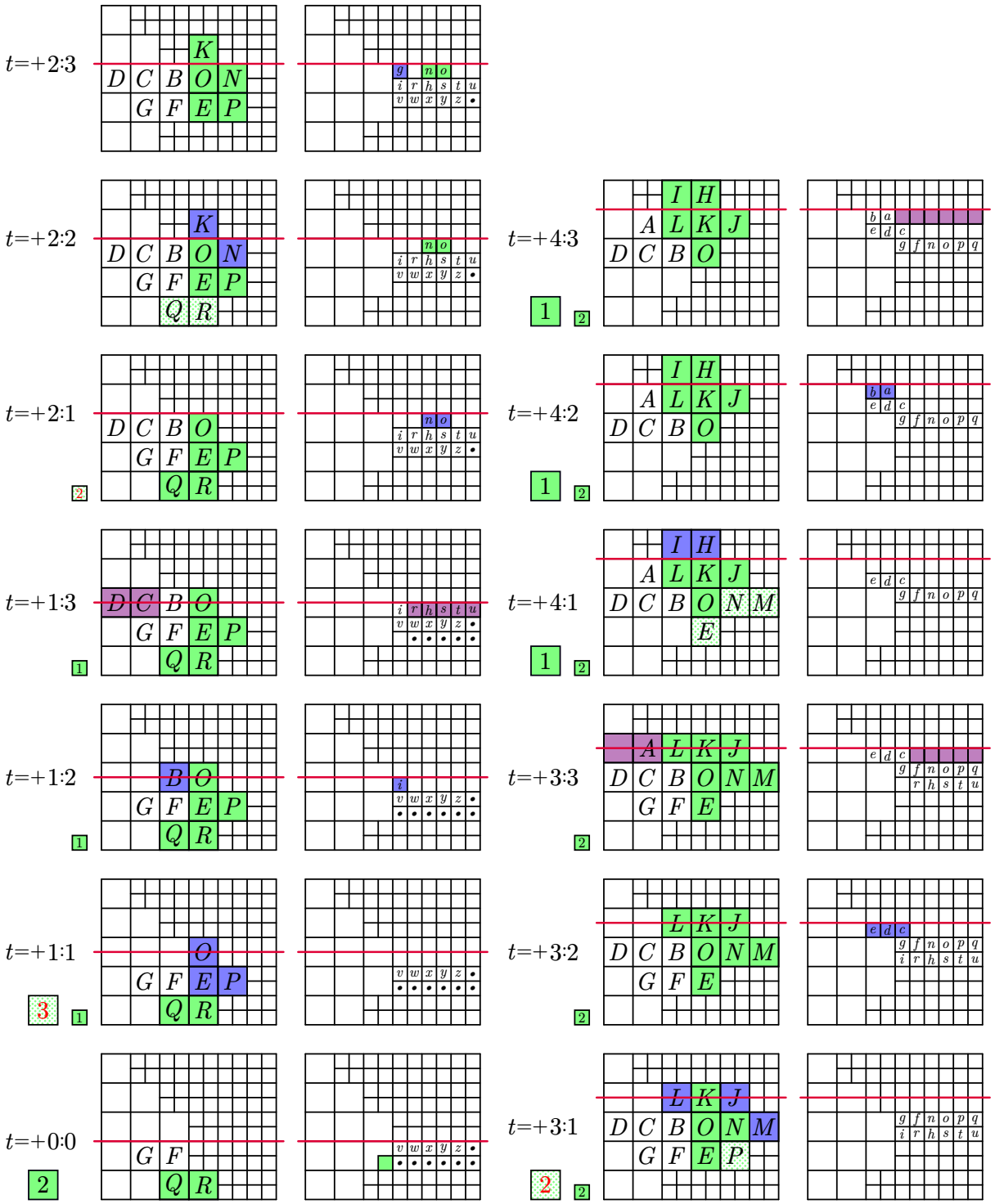


Figure B.7: Straight double-cell cusp flow graph for section 5.6.2.

Appendix C

Single-cell unit pulse results

Refinement results for the schemes discussed in chapter 5 for different single-cell unit pulses at straight and diagonal borders for fine-to-coarse, coarse-to-fine and parallel configurations.

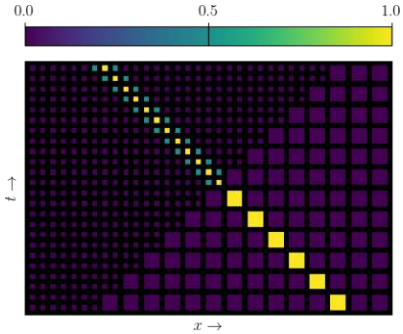


Fig. C.1: Eqs. (5.10) to (5.12).

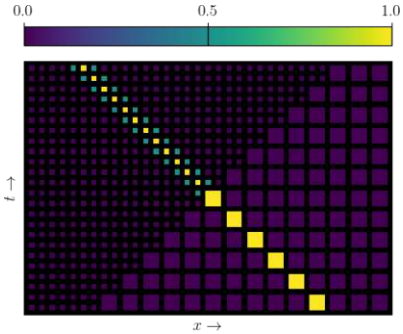


Fig. C.2: Eqs. (5.10) to (5.12).

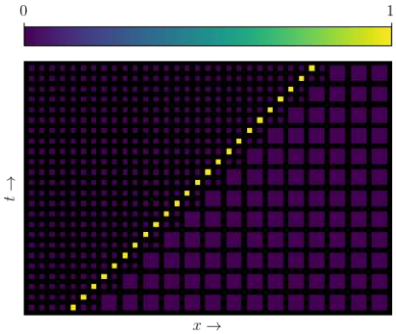


Fig. C.3: Eqs. (5.10) to (5.12).

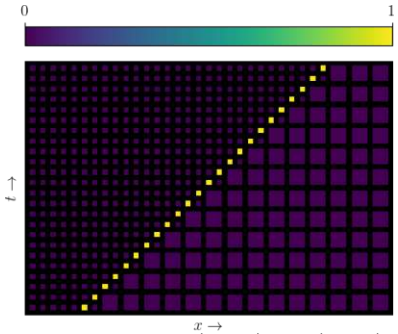


Fig. C.4: Eqs. (5.10) to (5.12).

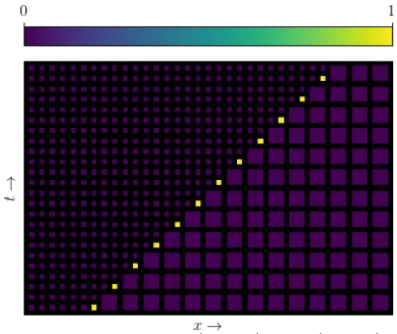


Fig. C.5: Eqs. (5.10) to (5.12).

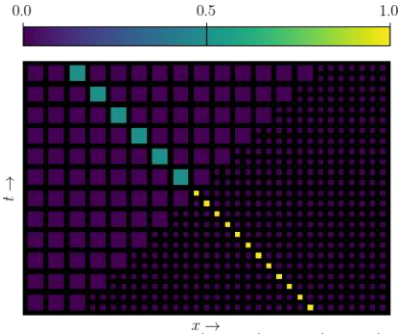


Fig. C.6: Eqs. (5.13) to (5.14).

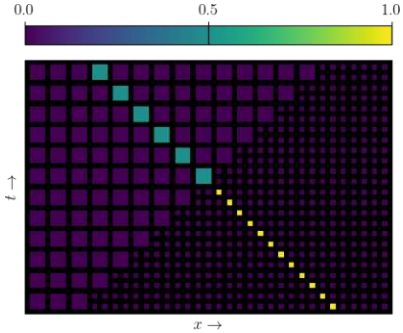


Fig. C.7: Eqs. (5.13) to (5.14).

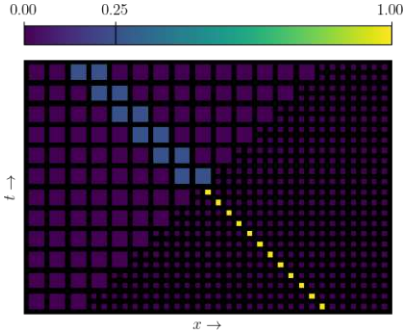


Fig. C.8: Eqs. (5.13) to (5.14).

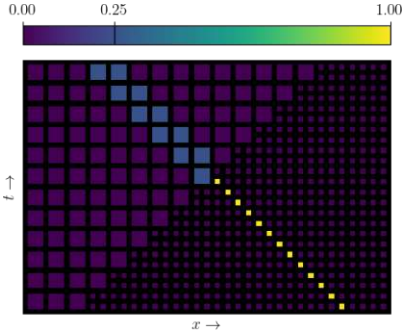


Fig. C.9: Eqs. (5.13) to (5.14).

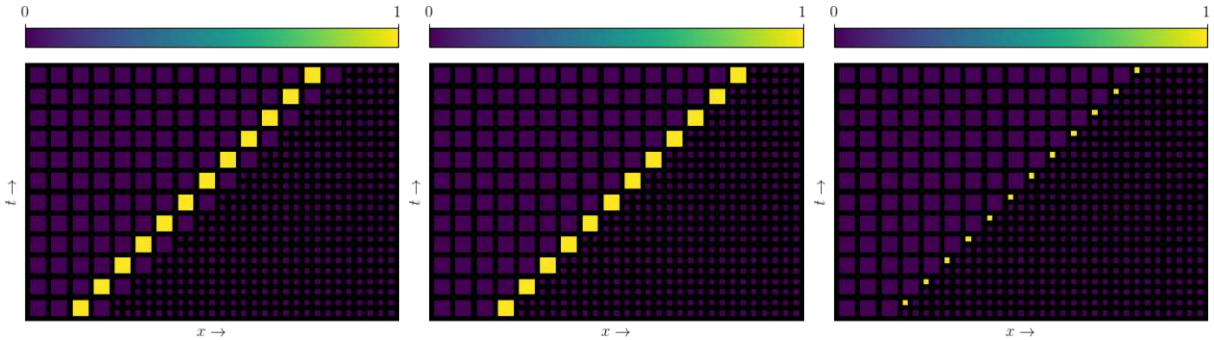


Fig. C.10: Eqs. (5.13) to (5.14). Fig. C.11: Eqs. (5.13) to (5.14). Fig. C.12: Eqs. (5.13) to (5.14).

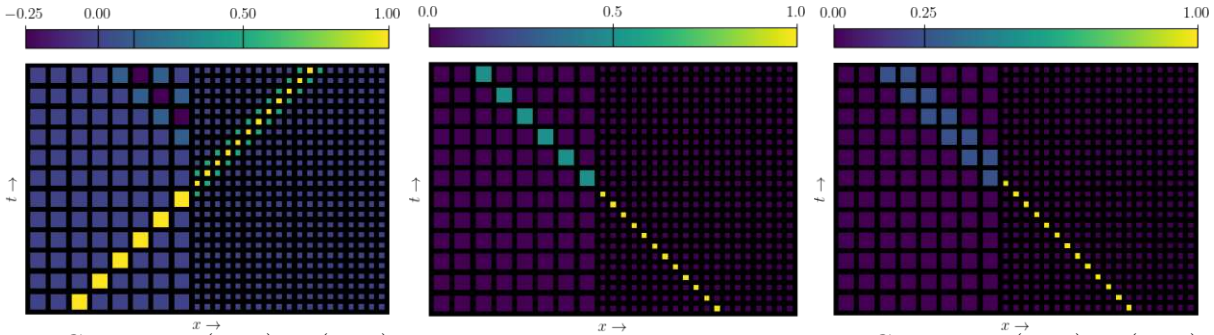


Fig. C.13: Eqs. (5.15) to (5.17). Fig. C.14: Eqs. (5.15) to (5.17). Fig. C.15: Eqs. (5.15) to (5.17).

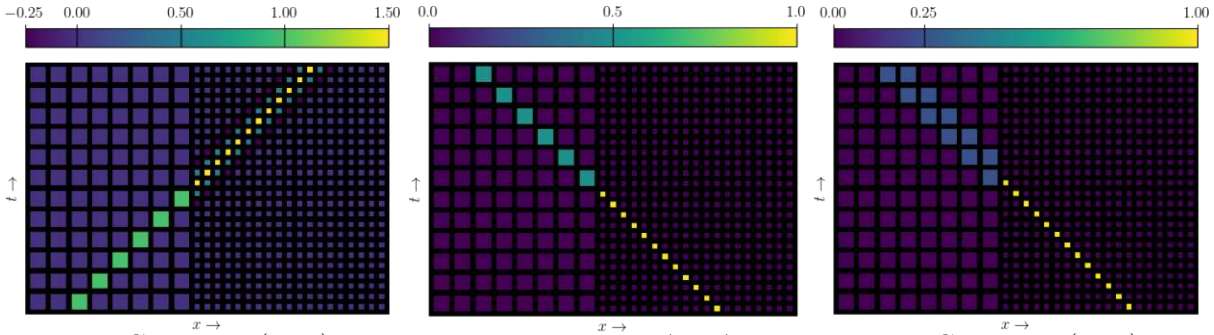


Fig. C.16: Eq. (5.18). Fig. C.17: Eq. (5.18). Fig. C.18: Eq. (5.18).

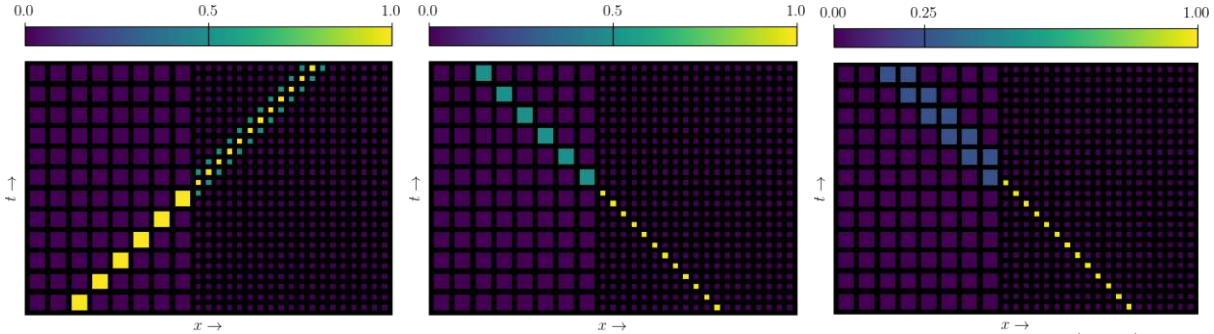


Fig. C.19: Eq. (5.20) Fig. C.20: Eq. (5.20) Fig. C.21: Eq. (5.20)

Appendix D

Ideal single-cell unit pulse propagation at the straight border

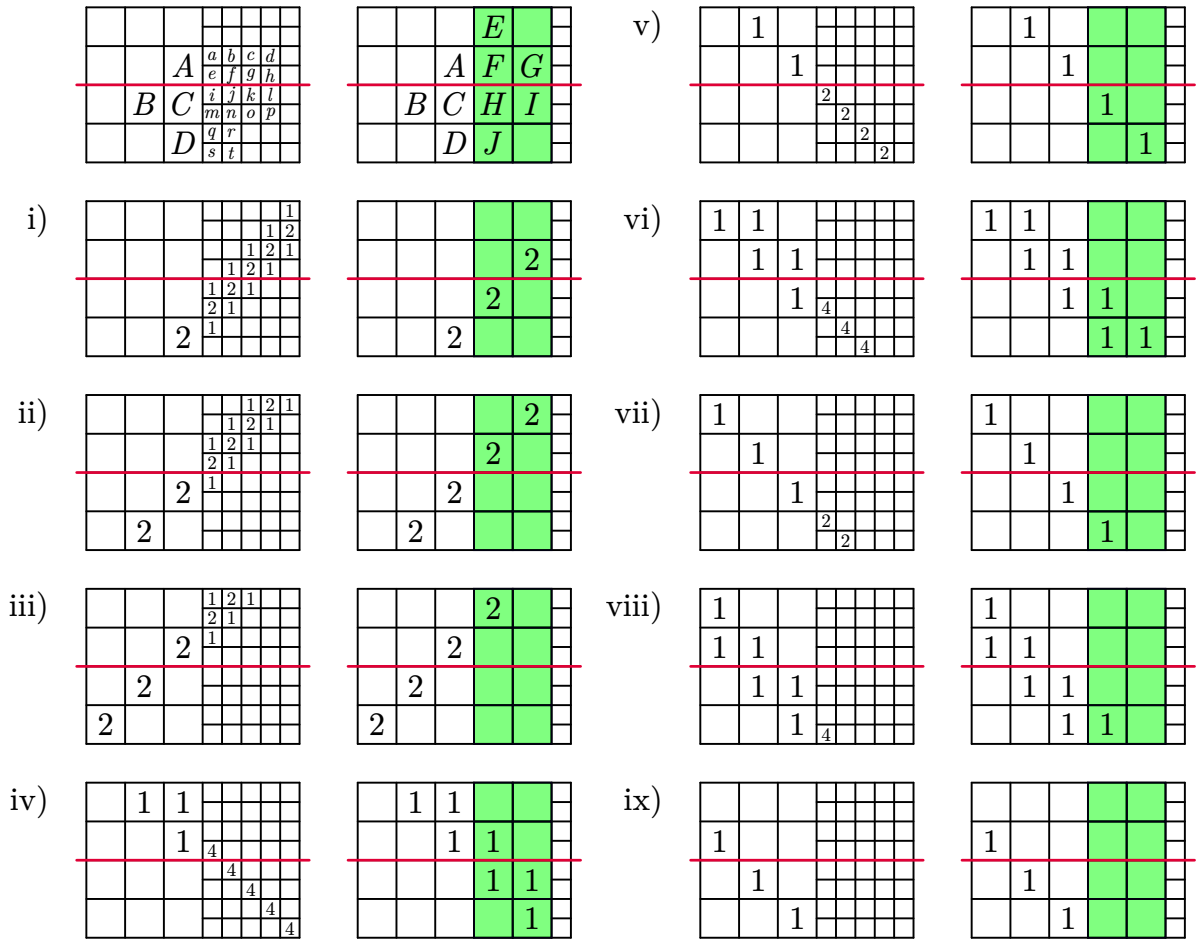


Figure D.1: Ideal propagation of single-cell wide unit pulses across the straight border. The numbers represent the values of the cells (white) and ghost cells (green). The SCTF pulse is the result of eqs. (5.10) to (5.14) for diagonal borders. The SFTC pulse is the result of averaging fine cells. Panels i) to ix) lead to a system of linear equations that is solved for the coarse border cell A of the next time slice after the red line. See section 5.3.

Appendix E

Modified averaging at the straight border

In the straight border refinement scheme given by eqs. (5.20), the cell G is calculated using

$$G = H - K + \underbrace{\frac{1}{2}(r+s) + \frac{1}{4}(t-q)}_{L}. \tag{E.1}$$

We can now identify the additional ghost cell L in fig. E.1 as the contribution of the fine cells entering eq. (E.1) and even go as far as to declare

$$L = \frac{1}{2}(r+s) + \frac{1}{4}(t-q) \tag{E.2}$$

as the substitute equation for calculating coarse cells bases on "averaging" fine cells. The consistency of this approach is proven in the following. Equation (E.2) is asymmetric in time, where following the time axis (up in fig. E.1) one contributing fine cell is in a future time slice that does not overlap with the coarse cell to calculate.

We start by using eq. (E.2) on F. Then, we insert the refinement eqs. (5.20) for the fine border cells and the default update stencil eq. (4.11) for fine non-border cells until we only have contributions from fine and coarse cells below the red line, except for F

$$4F = 4\left[\frac{1}{2}(a+b) + \frac{1}{4}(e-z)\right], \tag{E.3}$$

$$= 4(F-I) + 2k + 2o + l - j. \tag{E.4}$$

Next, we use eq. (E.2) again, this time for

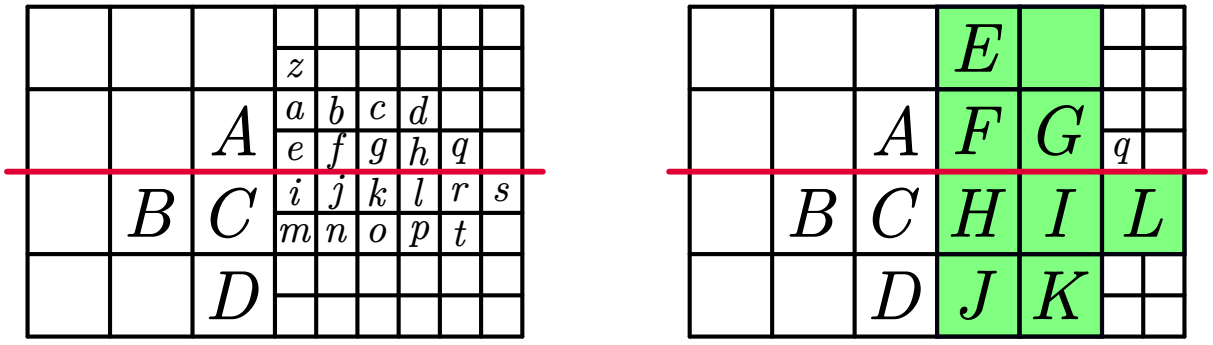
$$4I = 2(k+l) + o - g = 2k + 2o + l - j. \tag{E.5}$$

By inserting eq. (E.5) for I in eq. (E.4), we recover a true statement.

If we change the time direction and thus start instead with

$$F = \frac{1}{2}(e+f) + \frac{1}{4}(a-i), \tag{E.6}$$

we can no longer recover a true statement.



(a) (b)

Figure E.1: L can be identified as the modified average in eqs. (5.20). This average can be used consistently for any coarse ghost cell.

Appendix F

Norm conservation for included setups

Setup	Page	Scale h^i	Δ	Relative Δ
Figure 7.1b †	79	2^0	-7.96×10^{-3}	-5.46×10^{-4}
Figure 7.2a	79	2^{-2}	-1.14×10^{-13}	-1.42×10^{-15}
Figure 7.3b	80	2^{-2}	3.55×10^{-15}	1.11×10^{-15}
Figure 7.5a	81	2^0	3.55×10^{-15}	1.77×10^{-16}
Figure 7.5b	81	2^0	$0.00\times10^{+0}$	$0.00\times10^{+0}$
Figure 7.9a	83	2^{-2}	4.88×10^{-14}	1.53×10^{-14}
Figure 7.9b	83	2^{-2}	1.29×10^{-13}	4.04×10^{-14}
Figure 7.13a	85	2^0	1.07×10^{-14}	5.23×10^{-16}
Figure 7.13b	85	2^0	3.55×10^{-15}	1.77×10^{-16}
Figure 7.17a ‡	88	2^0	$-1.02\times10^{+0}$	-5.34×10^{-2}
Figure 7.17b ‡	88	2^0	$-1.93\times10^{+0}$	-1.01×10^{-1}

Table F.1: Errors of the norms of the referenced setups from chapter 7 when comparing the values of the first and last time slices. The norm is given in units of a^1 by eq. (5.1): $a^1 \sum_x 2^i \phi_x$.
† For this large number of time steps there are already reflections happening at the boundaries that violate norm conservation.
‡ Time slices with values actively being refined at borders are not conserving the norm (section 6.1.2).

Figures

Listings

Tables

Abbreviations

1+1D	1+1 dimensions / dimensional
3+1D	3+1 dimensions / dimensional
CCTF	Cusp Coarse To Fine
CFL	Courant-Friedrichs-Lewy
CFTC	Cusp Fine To Coarse
CTF	coarse-to-fine
DCTF	Diagonal Coarse To Fine
DFTC	Diagonal Fine To Coarse
EOM	equations of motion
FTC	fine-to-coarse
LHS	left-hand side
QCD	quantum chromodynamics
RHS	right-hand side
SCTF	Straight Coarse To Fine
SFTC	Straight Fine To Coarse
w.r.t.	with respect to

Bibliography

[1] N. Brambilla, S. Eidelman, P. Foka et al. *QCD and strongly coupled gauge theories: challenges and perspectives*. In: *The European Physical Journal C* 74.10 (Oct. 2014). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-014-2981-5.

[2] D. Gelfand, A. Ipp and D. Müller. *Simulating collisions of thick nuclei in the color glass condensate framework*. In: *Physical Review D* 94.1 (July 2016). ISSN: 2470-0029. DOI: 10.1103/physrevd.94.014020.

[3] A. Ipp and D. Müller. *Broken boost invariance in the Glasma via finite nuclei thickness*. In: *Physics Letters B* 771 (2017), pp. 74–79. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2017.05.032.

[4] D. Müller and A. Ipp. *Rapidity profiles from 3+1D Glasma simulations with finite longitudinal thickness*. In: *PoS* EPS-HEP2017 (2017), p. 176. DOI: 10.22323/1.314.0176.

[5] A. Ipp and D. Müller. *Implicit schemes for real-time lattice gauge theory*. In: *Eur. Phys. J. C* 78.884 (Oct. 31, 2018). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-018-6323-x.

[6] D. Müller. *Simulations of the Glasma in 3+1D*. PhD thesis. TU Vienna, Mar. 2019. URL: https://resolver.obvsg.at/urn:nbn:at:at-ubtuw:1-122958.

[7] A. Ipp, D. I. Müller and D. Schuh. *Anisotropic momentum broadening in the 2 + 1D glasma: Analytic weak field approximation and lattice simulations*. Oct. 2020. DOI: 10.1103/PhysRevD.102.074001.

[8] S. Schlichting and P. Singh. *3-D structure of the Glasma initial state: Breaking boost-invariance by collisions of extended shock waves in classical Yang-Mills theory*. In: *Phys. Rev. D* 103 (1 Jan. 2021), p. 014003. DOI: 10.1103/PhysRevD.103.014003.

[9] M. J. Berger and J. Oliger. *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*. In: *Journal of Computational Physics* 53.3 (Mar. 1984), pp. 484–512. DOI: 10.1016/0021-9991(84)90073-1.

[10] M. J. Berger and P. Colella. *Local Adaptive Mesh Refinement for Shock Hydrodynamics*. In: *Journal of Computational Physics* 82.1 (May 1989), pp. 64–84. DOI: 10.1016/0021-9991(89)90035-1.

[11] A. Harten. *Multiresolution algorithms for the numerical solution of hyperbolic conservation laws*. In: *Communications on Pure and Applied Mathematics* 48.12 (1995), pp. 1305–1342. DOI: https://doi.org/10.1002/cpa.3160481201.

[12] G. Chiavassa, R. Donat and S. Müller. *Multiresolution-based adaptive schemes for Hyperbolic Conservation Laws*. In: *Adaptive Mesh Refinement - Theory and Applications*. Ed. by T. Plewa, T. Linde and V. Gregory Weirs. Springer Berlin Heidelberg, 2005, pp. 137–159. ISBN: 978-3-540-27039-3.

[13] U. Trottenberg, C. W. Oosterlee and A. Schüller. *Multigrid*. Academic Press, 2001. ISBN: 012701070X.

[14] P. Wesseling. *An introduction to multigrid methods*. Pure and applied mathematics. Wiley, 1992. ISBN: 0471930830.

[15] M. Leuthner. *Adaptive mesh refinement for real-time lattice scalar field theory*. TU Wien. Project thesis. Mar. 3, 2021.

[16] L. Lapidus and G. F. Pinder. *Numerical Solution of Partial Differential Equations in Science and Engineering*. John Wiley & Sons, Ltd, 1999. ISBN: 9781118032961. DOI: 10.1002/9781118032961.

[17] H. P. Langtangen and S. Linge. *Finite Difference Computing with PDEs*. Vol. 16. Texts in Computational Science and Engineering. Springer International Publishing, 2017. ISBN: 978-3-319-55456-3. DOI: 10.1007/978-3-319-55456-3.

[18] R. J. LeVeque. *Finite difference methods for ordinary and partial differential equations : steady-state and time-dependent problems*. SIAM, 2007. ISBN: 0898716292.

[19] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Applied mathematics series. U.S. Government Printing Office, 1972. ISBN: 0486612724.

[20] F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier et al. *NIST Digital Library of Mathematical Functions*. Release 1.1.2 of 2021-06-15. URL: http://dlmf.nist.gov/ (visited on 09/2021).

[21] A. J. Lew and P. Mata A. *A Brief Introduction to Variational Integrators*. In: *Structure-preserving Integrators in Nonlinear Structural Dynamics and Flexible Multibody Dynamics*. Ed. by P. Betsch. Springer International Publishing, 2016, pp. 201–291. ISBN: 978-3-319-31879-0. DOI: 10.1007/978-3-319-31879-0_5.

[22] A. Lew, J. E. Marsden, M. Ortiz and M. West. *Variational time integrators*. In: *International Journal for Numerical Methods in Engineering* 60.1 (2004), pp. 153–212. DOI: https://doi.org/10.1002/nme.958.

[23] A. Lew, J. E. Marsden, M. Ortiz and M. West. *An Overview of Variational Integrators*. In: *Finite Element Methods: 1970's and Beyond*. Theory and engineering applications of computational methods. International Center for Numerical Methods in Engineering (CIMNE), 2004, pp. 1–18. ISBN: 9788495999498. URL: https://resolver.caltech.edu/CaltechAUTHORS:20101005-091206576.

[24] R. Courant, K. Friedrichs and H. Lewy. *Über die partiellen Differenzengleichungen der mathematischen Physik*. In: *Math. Ann.* 100 (1928), pp. 32–74. DOI: 10.1007/BF01448839.

[25] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. 37th ed. Addison-Wesley professional computing series. Addison-Wesley, Mar. 2009. ISBN: 0201633612.

[26] Python Software Foundation. *Python Language Reference*. Version 3.10.5. June 6, 2022. URL: https://docs.python.org/release/3.10.5/.

[27] C. R. Harris, K. J. Millman, S. J. van der Walt et al. *Array programming with NumPy*. In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[28] P. Virtanen, R. Gommers, T. E. Oliphant et al. *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.* In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[29] J. D. Hunter. *Matplotlib: A 2D Graphics Environment.* In: *Computing in Science Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[30] T. Kluyver, B. Ragan-Kelley, F. Pérez et al. *Jupyter Notebooks - a publishing format for reproducible computational workflows.* In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas.* Ed. by F. Loizides and B. Scmidt. Netherlands: IOS Press, 2016, pp. 87–90. URL: https://eprints.soton.ac.uk/403913/.