Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

Unterschrift Betreuer

TECHNISCHE
UNIVERSITÄT
WIEN

DIPLOMARBEIT

Implementing a 3+1-dimensional simulation for a Color Glass Condensate model in C++

ausgeführt am Institut für theoretische Physik
der Technischen Universität Wien

unter der Anleitung von
Privatdoz. Dipl.-Ing. Dr.techn. Andreas Ipp

durch

Patrick Michael Kappl

Markt 49, 3365 Allhartsberg

October 30, 2018

Unterschrift Student

Zusammenfassung

Schwerionenkollisionen erlauben es, die Quantenchromodynamik im Bereich hoher Dichte und hoher Energien zu untersuchen. Das Modell des Farbglaskondensats (engl. color glass condensate, CGC) ist eine klassische, effektive Feldtheorie, welche die kollidierenden Kerne als flache, lorentzkontrahierte Scheiben beschreibt. Der vorliegende C++-Code kann die ersten Phasen von Schwerionenkollisionen im Rahmen dieses Farbglaskondensats simulieren und verwendet dabei Kerne mit endliche Dicke. Die gesamten Bewegungsgleichungen, Zwangsbedingungen und die Hamiltondichte sind abgeleitet von der CGC-Wirkung. Zusätzlich wird eine alternative Diskretisierung des Poynting-Vektors hergeleitet, welche eine bessere Erhaltung des Poynting-Theorems erlaubt. Im Vergleich zum Vorgänger, dem OpenPixi Simulator, sollte der vorliegende C++-Code eine leichter vorhersagbare Skalierung aufweisen, schneller sein und weniger Speicher verbrauchen. Um dies zu untermauern wurde eine Reihe von Tests und Benchmarks durchgeführt. So wurden wohlbekannte Effekte wie das Auftreten einer Druckanisotropy und Gaußscher Rapiditätsprofile im Vergleich mit OpenPixi überprüft.

Abstract

Heavy ion collisions allow the study of quantum chromodynamics in the dense, high energy regime. The color glass condensate (CGC) model is a classical, effective field theory that describes the colliding nuclei as flat, Lorentz contracted discs. The present C++ code can simulate the early stages of heavy ion collisions in the CGC framework with nuclei of finite thickness. All equations of motion and constraints as well as the Hamilton density are derived from the CGC action. Additionally, an alternative discretization of the Poynting vector, which allows better conservation of Poynting's theorem, is obtained. Compared to its predecessor, the OpenPixi simulator, the new C++ code should have more predictable scaling, run faster and use less memory. In order to confirm this, a set of tests and performance benchmarks were conducted. Furthermore, the emergence of well-known features like the pressure anisotropy and Gaussian rapidity profiles were studied in comparison with OpenPixi.

Contents

1. Introduction 1

2. Early Stages of Heavy Ion Collisions 3
- 2.1. Parton Model . . . 3
- 2.2. Gluon Saturation . . . 4
- 2.3. Color Glass Condensate . . . 6

3. Numerical Methods 9
- 3.1. Real-time Lattice Gauge Theory . . . 9
- 3.2. Colored Particle in Cell Method . . . 12
- 3.3. Variation of the Action . . . 14
- 3.4. Gauss Constraint . . . 15
- 3.5. Equations of Motion . . . 16
- 3.6. Hamiltonian Density . . . 17
- 3.7. Poynting Vector . . . 19
- 3.8. Initial Conditions . . . 21

4. Implementation 27
- 4.1. Simulation Core . . . 27
- 4.2. Initial Conditions . . . 33
- 4.3. Diagnostics . . . 34
- 4.4. Tests . . . 36
- 4.5. Parallelization . . . 36

5. Test Results 39
- 5.1. Pressure Anisotropy . . . 39
- 5.2. Rapidity Profiles . . . 40
- 5.3. Transverse Pressure . . . 43
- 5.4. Execution Time and Memory Usage . . . 44
- 5.5. OpenMP . . . 48

6. Conclusion 51

References i

A. Plaquettes and Field Strength Tensor v

B. Conservation of Gauss Constraint **vii**

1. Introduction

In the standard model of particle physics, quantum chromodynamics (QCD) is the fundamental theory of the strong interaction, which describes the forces between the color charged quarks and gluons. However, due to confinement they can not be directly observed under normal conditions. Only at very high temperatures and densities the hadrons will melt into their constituents, creating a quark–gluon plasma (QGP) [1]. It is believed that very shortly after the Big Bang the universe was in this state of matter [2]. Nowadays, a quark–gluon plasma can be produced artificially in heavy ion collisions presently performed at Brookhaven National Laboratory's Relativistic Heavy Ion Collider (RHIC) and CERN's Large Hadron Collider (LHC). These collision experiments help to gain a better understanding of the interactions between the fundamental particles of QCD. So far the theoretical description of heavy ion collisions relies on effective theories and numerical methods to solve them. The color glass condensate (CGC) is a classical model, which describes the colliding nuclei as very thin, Lorentz contracted discs of hard, color charged particles and soft gluon fields [3, 4]. Right after the collision of these discs, a highly coherent state of matter consisting of longitudinal color flux tubes, the so called glasma, is created. This glasma then transforms into the quark–gluon plasma [5].

OpenPixi – Open Colored Particle-in-Cell (CPIC) simulator – can simulate these early stages of heavy ion collisions within the CGC framework [6, 7]. This is achieved by using real-time lattice gauge theory and the CPIC method which allow the description of Yang-Mills fields coupled to color charged point particles. In contrast to other simulations, OpenPixi uses nuclei with finite longitudinal thickness in 3+1 dimensions [8, 9]. Therefore it can go beyond the boost-invariant case, which is the limit of infinite Lorentz contraction that leads to infinitesimally thin nuclei and effectively reduces the dimensions to 2+1 [10–12]. This limit also makes the whole theory invariant under Lorentz boosts along the longitudinal direction, hence the name. Using thick nucleons enables studying collisions at lower energies, because thicker ions correspond to lower Lorentz contraction. In principle it is also possible to find the limits of applicability in this regard of the CGC framework, which is only an accurate theory for QCD in the limit of infinite energies.

OpenPixi is written in Java which is an uncommon choice in high performance computing (HPC). This is because it usually is not very resource-efficient with rather unpredictable memory usage. Consequently, one can not reliably predict the memory consumption of the simulation and is at risk of running out of memory at some point. The goal of this master thesis is to continue the work, started in my preceeding project thesis [13], and port the existing Java code to C++. This language was designed with performance, efficiency and flexibility of use in mind [14]. It allows a high level of abstraction via features such as object-oriented and generic programming, while also providing low-level programming features. Therefore it should be possible to fix the issue mentioned above and reduce memory consumption as well as execution time.

Chapter 2 gives more details on the theoretical description of the early stages of heavy ion collisions. The numerical methods are discussed in chapter 3. It gives an introduction to real-time lattice gauge theory, as well as the colored particle in cell method. Furthermore, the equations of motion, the Hamiltonian density, the Poynting vector and the initial conditions are derived. Chapter 4 describes how the software is organized in general and how the numerical methods and other modules are implemented. The results of various tests that verify the validity and benchmark the C++ code are presented in Chapter 5. Finally, chapter 6 gives a conclusion.

2. Early Stages of Heavy Ion Collisions

Trying to describe relativistic heavy ion collisions in a fully quantum-field theoretical way from the fundamental QCD-Lagrangian is way out of reach with current calculation tools and computational power. Due to the strong coupling and the non-Abelian nature of QCD even calculating the energy spectra of hadrons, let alone nuclei, is not trivial. The kinematics of high energy collisions however help simplify things. The following overview of the theoretical description of heavy ion collisions is based on papers by François Gelis. For more details see [15–17].

A prominent feature of QCD is its running coupling which decreases at small distances or high energies and increases at large distances or low energies. This leads to the so called asymptotic freedom and to confinement respectively. If one squeezes many hadrons into a tiny volume, the small inter-quark distances will lead to weak coupling. Thus the quarks will no longer be confined into hadrons but will form a plasma of deconfined quarks and gluons. This suggests that instead of the color singlet hadrons the many quarks and gluons are the relevant degrees of freedom in such a situation. Experimentally, such conditions with high particle densities can be achieved in heavy ion collisions presently performed by the RHIC and the LHC. Figure 2.1 shows the different stages of such collisions, which have a duration in the order of 10 fm/c. It also shows the theories and tools used in each of the stages. The first one is described by the color glass condensate which will be discussed in more detail in section 2.3. The transition from the anisotropic fields of this stage to the thermalized quark–gluon plasma described by viscous hydrodynamics is an active field of research. A possible candidate for a theory that can explain the fast isotropization is the recently developed effective kinetic theory [18, 19]. For the stages of the isotropic quark–gluon plasma, viscous hydrodynamics is still the prevailing tool, before kinetic theory can be used to describe the dynamics of the emerged hadrons. The goal of the software, created as part of this thesis, is to simulate the very first stage of heavy ion collisions. At this early time, the transition from the color glass condensate to the quark–gluon plasma goes through a state of matter called glasma, a contraction of glass (from the color glass condensate) and plasma (from the quark–gluon plasma) [20].

2.1. Parton Model

The parton model describes hadrons as a set of point-like constituents called partons. For protons and neutrons at low energies, these constituents are their three valence quarks.

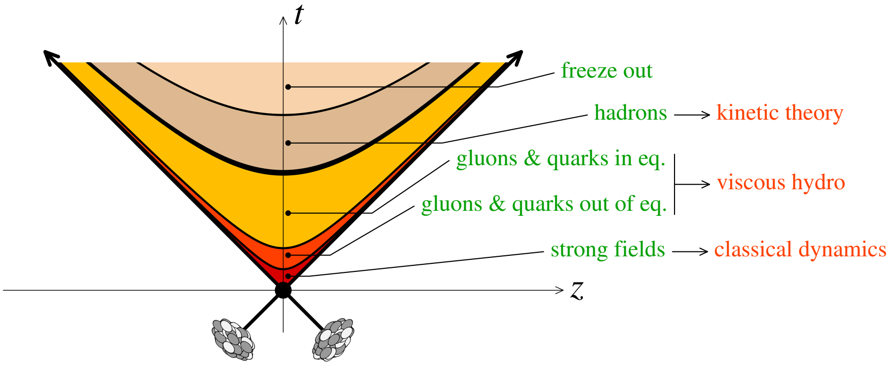
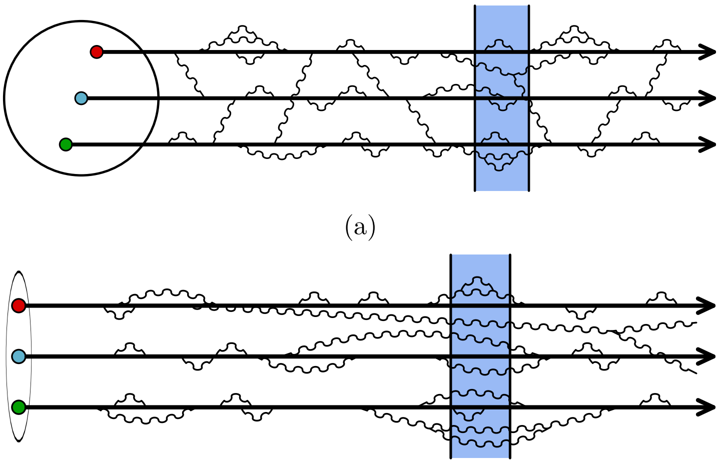


Figure 2.1.: Minkowski diagram of a collision of two heavy ions moving close to the speed of light along the light cone. The different stages of the collisions are separated by hyperbolas representing different proper times. The tools and theories used to describe the various stages are also mentioned [17, Fig. 5].

This changes however when going to higher energies. Figure 2.2a shows a simplified picture of the dynamics of the valence quarks and gluons inside a slow nucleon. The thick lines are the valence quarks, while the wiggly lines are virtual gluons providing the binding force. Sea-quarks – pairs of virtual quarks and antiquarks – are not shown. The blue strip highlights a typical timespan relevant for collisions. Even at these low energies, virtual quarks spring in and out of existence, but their lifetime, compared to the collision time, is too short for them to be relevant. If one collides such a slow hadron with a very fast one, as depicted in fig. 2.2b, the blue strip stays the same. However, the internal dynamics are slowed down by time dilation and the lifetime of some gluons is now longer than the collision time. Therefore these previously off-shell constituents become indistinguishable from on-shell particles. Another way to look at it is by considering that a lower collision time Δt at higher energies corresponds to a greater energy uncertainty $\Delta E \sim \frac{1}{\Delta t}$ of the gluons during the collision event. Again, this makes them appear to be on-shell although they were off-shell before. Increasing the energy uncovers more gluons because virtual particles and fluctuations exist at an arbitrarily small timescale. Thus the parton model describes high energy nucleons as a set of quasi-free particles, called partons, which have a density that increases with energy.

2.2. Gluon Saturation

At low energies, collisions mainly involve single parton interactions. This is no longer true at higher energies because of the increased parton density. Multiparton interactions and correlations between single partons become relevant and can change the behavior of collision processes. When the parton density increases up to a value of the order of the inverse coupling $1/g^2$, a strongly interacting regime – called gluon saturation – is reached. This saturation can be illustrated in the following way. At low energies



(a)

(b)

Figure 2.2.: Simplified dynamics of valence quarks and virtual gluons in nucleons. Sea quarks are not shown. The blue strip marks a typical timespan for collisions. (a) shows a low energy nucleus [17, Fig. 6], while (b) shows a highly boosted one [17, Fig. 7].

a hadron is described by only a few partons. However at increasing speed more and more gluons are emitted by bremsstrahlung and since they are confined in the volume of the hadron, the gluon density rises. At some point the gluon cascades cannot evolve independently any more and recombination processes become relevant, counteracting the density increase. In simplified terms, this happens when the number of gluons per unit area times the cross section for recombining two gluons into one is larger than one [17, eq. 4],

$$\underbrace{\alpha_s Q^{-2}}_{\sigma_{gg\to g}} \times \underbrace{A^{-2/3} xG(x, Q^2)}_{\text{surface density}} \geq 1. \tag{2.1}$$

Here α_s is the strong coupling, Q is the 4-momentum of the photon exchanged in the scattering, A is the mass number of the nucleon and $xG(x, Q^2)$ is the integrated gluon distribution. Rearranging this yields an inequality on Q [17, eq. 5],

$$Q^2 \leq \underbrace{Q_s^2 := \frac{\alpha_s xG(x, Q_s^2)}{A^{2/3}}}_{\text{saturation momentum}} \sim A^{1/3} x^{-0.3}. \tag{2.2}$$

Processes are governed by saturation physics if their typical momenta are below the saturation scale Q_s. Figure 2.3 plots Q_s^2 over A and x. Although the non-linear interaction of the gluons in this regime makes calculations complicated, there is also a positive side to it. Q_s replaces all softer scales for the typical momentum of a parton and thus controls the running of the coupling. Since Q_s increases when x decreases (i.e. when energy increases), this allows for a weak coupling treatment of the partons in high energy collisions.

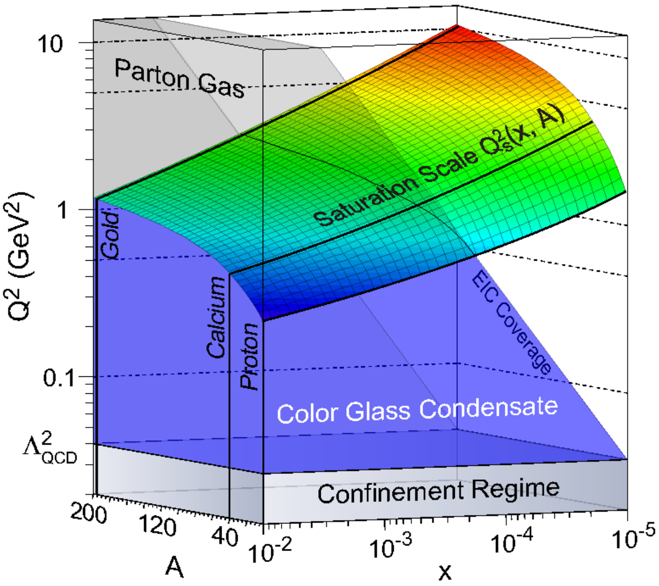


Figure 2.3.: Saturation momentum plotted over the fraction x of longitudinal momentum carried by a parton and the mass number A [21, Fig. 1]. The blue area illustrates the saturation regime that is kinematically accessible by an elctron-ion collider.

2.3. Color Glass Condensate

The color glass condensate is an effective theory which quantitatively describes the gluon saturation regime. As discussed in section 2.1, in high energy ion collisions the nucleons look like thin disks of partons that are essentially frozen in the transverse directions but carry a large longitudinal momentum. For an observer in the center of mass frame, the only relevant information of such partons is the color current j_a^μ, where μ is the Lorentz index running from 0 to 3 and a is the color index running from 1 to the number of generators of the gauge group, which is 8 for SU(3). In light-cone coordinates, the dominant, longitudinal part of a hadron moving in positive z-direction reads

$$j_a^\mu(x) = \rho_a(x^-, x_T)\delta^{\mu+}, \tag{2.3}$$

where $\rho_a(x)$ is the density of color charges, $x^\pm = \frac{1}{\sqrt{2}}(t \pm z)$ the standard light-cone coordinates, x_T the two transverse coordinates and $\delta^{\mu+}$ is the Kronecker delta which equals 1 if the index μ is in x^+-direction and vanishes otherwise. Due to time dilation and Lorentz contraction, ρ_a does not depend on x^+ and its x^- dependence is very peaked around $x^- = 0$. The important part is that not all partons can be treated as static longitudinal color currents. Some of them have a smaller longitudinal momentum than the hard partons. These soft modes need to be described as full quantum fields.

The CGC model introduces a momentum cutoff between the rapidity of the laboratory frame and the hadron. Partons below the cutoff are mostly gluons and described as Yang-Mills gauge fields, whereas those above it are treated as static longitudinal currents. With this, the CGC effective action reads

$$\mathcal{S}_{\mathrm{CGC}} = \int \mathrm{d}^4 x \left(-\frac{1}{4} F^a_{\mu\nu} F^{\mu\nu}_a - j^\mu_a A^a_\mu \right) , \tag{2.4}$$

where $F^a_{\mu\nu}$ is the gluon field strength tensor, j^μ_a is the current density caused by the hard partons and A^a_μ is the gauge field. One has to consider that the current still contains the charge density $\rho_a(x^-, x_T)$, which is very hard to determine for heavy ions. With current methods, it is not possible to calculate it from first principles. Instead one has to use models that are available with various degrees of sophistication. The McLerran-Venugopalan model is one of the simplest ones [22–24]. It describes the charge configuration as random charges with a Gaussian distribution. Consequently, when calculating observables, one has to average over many charge configurations to obtain the expectation value of the desired observable.

Although the action looks rather simple, one encounters problems when doing power counting. In the saturation regime, ρ_a is of the order of the inverse coupling $1/g$. However, for every new source there needs to be a vertex of order g to connect it to the rest of the Feynman diagram. Therefore one can add an arbitrary number of sources without changing the magnitude of the diagram. This means that an infinite number of graphs contributes at each order in g^2. Fortunately, it can be shown that at leading order the sum of all diagrams is just the solution of the classical field equations of motion with boundary conditions

$$\lim_{x^0 \to -\infty} A^a_\mu(x) = 0 \ . \tag{2.5}$$

In conclusion, the CGC model drastically simplifies the description of heavy ion collisions, requiring to leading order only the solution of the classical equations of motion of an effective action. However, this is still not an easy task and requires numerical methods like real-time lattice gauge theory which is explained in the following chapter.

3. Numerical Methods

The previous chapter showed that one only has to solve the equations of motion of a classical field theory to describe heavy ion collisions at leading order. However, to achieve that, the use of numerical simulations is necessary. First of all, a proper formulation of the action on the lattice has to be found. Then, one has to describe the color charged, point-like particles moving through the grid and interacting with the fields in a consistent way, which is done with the colored particle in cell (CPIC) method. The final step is to derive the equations of motion for both the fields and the particles, and to formulate them in a way which can easily be implemented in a simulation code.

3.1. Real-time Lattice Gauge Theory

Lattice gauge theory often refers to a non-perturbative approach to solving quantum field theories, especially QCD, with the help of numerical methods like Monte-Carlo simulations. However, this is not what will be described here and used later on. The goal of this section is to find a formulation of the classical CGC action on a cuboid lattice that is still gauge invariant at the discrete level. The equations of motion, derived from this action, can then be solved by a simple explicit solver. In this regard, it is more similar to common applications of finite difference methods.

First, a few definitions, sign conventions and notation have to be explained. According to section 2.3, the continuum CGC action in natural units ($\hbar = 1$, $c = 1$) and 3+1 dimensions reads

$$S = \int d^4x \left(-\frac{1}{4} F^a_{\mu\nu} F^{\mu\nu}_a - j^\mu_a A^a_\mu \right) = \int d^4x \left(-\frac{1}{2} \operatorname{Tr}(\mathcal{F}_{\mu\nu} \mathcal{F}^{\mu\nu}) - 2 \operatorname{Tr}(j^\mu \mathcal{A}_\mu) \right). \tag{3.1}$$

Throughout the whole thesis, the Minkowski metric with mostly minus signs is chosen

$$\eta_{\mu\nu} = \operatorname{diag}(+, -, -, -). \tag{3.2}$$

The gauge group SU(2) instead of SU(3) is used, since this is already a non-Abelian group and thus leads to qualitatively similar results, while being far less computationally intensive. The three generators of the group are denoted by t^a and have the following properties

$$(t^a)^\dagger = t^a, \tag{3.3}$$
$$[t^a, t^b] = if^{ab}{}_c t^c, \tag{3.4}$$
$$\mathrm{Tr}(t^a t^b) = \frac{1}{2}\delta^{ab}, \tag{3.5}$$

where f^{abc} are the antisymmetric structure constants. The covariant derivative acting on a field ψ is chosen to be

$$\mathcal{D}_\mu \psi := \partial_\mu \psi - \mathrm{i}g\mathcal{A}_\mu \psi, \tag{3.6}$$

where g is the coupling constant and i the imaginary unit. The gauge field, current density and field strength tensor therefore read

$$\mathcal{A}_{x,\mu} := A^a_{x,\mu} t_a = A^a_\mu(x) t_a, \tag{3.7}$$
$$j_{x,\mu} := j^a_{x,\mu} t_a = j^a_\mu(x) t_a, \tag{3.8}$$
$$\begin{aligned}\mathcal{F}_{x,\mu\nu} := F^a_{x,\mu\nu} t_a &= \partial_\mu \mathcal{A}_{x,\nu} - \partial_\nu \mathcal{A}_{x,\mu} - \mathrm{i}g[\mathcal{A}_{x,\mu}, \mathcal{A}_{x,\nu}] \\ &= (\partial_\mu A^a_{x,\nu} - \partial_\nu A^a_{x,\mu} + g f^a{}_{bc} A^b_{x,\mu} A^c_{x,\nu}) t_a\end{aligned} \tag{3.9}$$

and transform under gauge transformations with a group element Ω_x like

$$\mathcal{A}'_{x,\mu} = \Omega_x(\mathcal{A}_{x,\mu} + \frac{\mathrm{i}}{g}\partial_\mu)\Omega^\dagger_x, \tag{3.10}$$
$$j'_{x,\mu} = \Omega_x j_{x,\mu} \Omega^\dagger_x, \tag{3.11}$$
$$\mathcal{F}'_{x,\mu\nu} = \Omega_x \mathcal{F}_{x,\mu\nu} \Omega^\dagger_x. \tag{3.12}$$

The external current density j also needs to satisfy the continuity equation

$$\mathcal{D}_\mu j^\mu = 0, \tag{3.13}$$

in order for the action to be gauge invariant up to a total derivative. The following correspondence between 3- and 4-vectors is used

$$V^\mu = (V^0, V^i) := (V^0, \mathbf{V}). \tag{3.14}$$

Latin indexes always denote spacial components of a 4-vector and never those of the 3-vector $(\mathbf{V})_i$. The electric and magnetic fields are chosen with the following signs

$$\mathcal{E}^i := E^{a,i} t_a := +\mathcal{F}^{i0}, \tag{3.15}$$
$$\epsilon_{ijk}\mathcal{B}^k := \epsilon_{ijk} B^{a,k} t_a := -\mathcal{F}_{ij}, \tag{3.16}$$

with $\epsilon_{123} := 1$.

To preserve the gauge symmetry of the action even in the discrete case, it is rewritten in terms of Wilson loops instead of gauge fields. These loops are gauge-invariant quantities defined as

$$W_C := \mathrm{Tr}\left(\mathcal{P}\exp\left(\mathrm{i}\oint_C \mathcal{A}_\mu \,\mathrm{d}x^\mu\right)\right), \tag{3.17}$$

where $\mathcal{P}$ is the path-ordering operator and C a closed curve in space. The path-ordered exponential transforms locally under gauge transformations

$$\mathcal{P}\exp\left(\mathrm{i}\oint_C \mathcal{A}_\mu \,\mathrm{d}x^\mu\right) \to \Omega(x)\mathcal{P}\exp\left(\mathrm{i}\oint_C \mathcal{A}_\mu \,\mathrm{d}x^\mu\right)\Omega^{-1}(x), \tag{3.18}$$

with x being the initial and end point of the curve C, because gauge transformations in between cancel each other. The invariance of the trace under cyclic permutations ensures that the Wilson loop is indeed gauge invariant. In order to obtain the discrete version of them on a 3+1-dimensional, cuboid lattice with lattice spacings a_μ, it is convenient to introduce so called gauge links $U_{x,\mu}$. They correspond to the parallel transport of the gauge field along the shortest possible path on the lattice. Since in the discrete case $\mathcal{A}_\mu$ is constant along the path from one grid point to the next, the path ordered integral turns into a multiplication of the gauge field at point x with the lattice spacing in direction μ,

$$U_{x,\mu} \coloneqq \exp(-\mathrm{i}g a_\mu \mathcal{A}_{x,\mu}). \tag{3.19}$$

Gauge links oriented in a negative direction are denoted with negative indexes and are given by the hermitian conjugate of the neighboring link,

$$U_{x,-\mu} = U^\dagger_{x-\mu,\mu}. \tag{3.20}$$

They transform under gauge transformations Ω_x like

$$U'_{x,\mu} = \Omega_x U_{x,\mu}\Omega^\dagger_{x+\mu}, \tag{3.21}$$

$$U'_{x,-\mu} = \Omega_x U_{x,-\mu}\Omega^\dagger_{x-\mu} \tag{3.22}$$

where the subscript $x+\mu$ denotes the point one gets when shifting x by one lattice spacing in the direction μ. Any Wilson loop on the lattice can now be built using these gauge links as building blocks. The smallest possible loops (without the trace) are called plaquettes and are defined as

$$\begin{aligned} U_{x,\mu\nu} &\coloneqq U_{x,\mu}U_{x+\mu,\nu}U_{x+\mu+\nu,-\mu}U_{x+\nu,-\nu} \\ &= U_{x,\mu}U_{x+\mu,\nu}U^\dagger_{x+\nu,\mu}U^\dagger_{x,\nu}, \end{aligned} \tag{3.23}$$

$$\begin{aligned} U^\dagger_{x,\mu\nu} &= U_{x,\nu}U_{x+\nu,\mu}U_{x+\mu+\nu,-nu}U_{x+\mu,-\mu} \\ &= U_{x,\nu}U_{x+\nu,\mu}U^\dagger_{x+\mu,\nu}U^\dagger_{x,\mu} \\ &= U_{x,\nu\mu}, \end{aligned} \tag{3.24}$$

with the following transformation law

$$U'_{x,\mu\nu} = \Omega_x U_{x,\mu\nu}\Omega^\dagger_x. \tag{3.25}$$

Using the Backer-Campbell-Hausdorff formula and approximating the gauge field at $x+\nu$ with

$$\mathcal{A}_{x+\nu,\mu} = \mathcal{A}_{x,\mu} + a_\nu \partial_\nu \mathcal{A}_{x,\mu} + \mathcal{O}(a^2), \tag{3.26}$$

one finds that the plaquette variables are, up to linear order, the exponential of the field strength tensors

$$U_{x,\mu\nu} = \exp(-\mathrm{i} g a_\mu a_\nu (\mathcal{F}_{x,\mu\nu} + \mathcal{O}(a))). \tag{3.27}$$

For a complete derivation see appendix A. To get the discretized action in terms of the plaquettes, $\mathrm{Tr}(U_{x,\mu\nu} + U^\dagger_{x,\mu\nu})$ needs to be expanded up to $-\frac{1}{2} g^2 a_\mu^2 a_\nu^2 (\mathcal{F}_{x,\mu\nu} + \mathcal{O}(a))^2$ and $\mathrm{Tr}(\mathcal{F}_{\mu\nu}\mathcal{F}^{\mu\nu})$ needs to be rewritten in the following way

$$\begin{aligned}\mathrm{Tr}(\mathcal{F}_{\mu\nu}\mathcal{F}^{\mu\nu}) &= \mathrm{Tr}(\mathcal{F}_{0i}\mathcal{F}^{0i} + \mathcal{F}_{i0}\mathcal{F}^{i0}) + \mathrm{Tr}(\mathcal{F}_{ij}\mathcal{F}^{ij}) \\ &= -2\sum_{i=1}^{3} \mathrm{Tr}(\mathcal{F}_{0i}^2) + \sum_{i=1}^{3}\sum_{j\neq i} \mathrm{Tr}(\mathcal{F}_{ij}^2). \end{aligned} \tag{3.28}$$

The linear terms from the expansion of the plaquettes cancel and the constant term does not matter because it does not contribute to the equations of motion. Using the trapezoidal rule to discretize the integral, one finally finds the gauge-invariant lattice action

$$\boxed{\begin{aligned} S_\Box = &-\frac{V}{g^2}\sum_x \sum_{i=1}^{3}\left(\frac{1}{a_0^2 a_i^2}\mathrm{Tr}(U_{x,0i} + U^\dagger_{x,0i}) - \frac{1}{2}\sum_{j\neq i}\frac{1}{a_i^2 a_j^2}\mathrm{Tr}(U_{x,ij} + U^\dagger_{x,ij})\right) \\ &- V\sum_x \left(j^\mu_{x,a} A^a_{x,\mu} + C + \mathcal{O}(a^2)\right), \end{aligned}} \tag{3.29}$$

where $V = a_0 a_1 a_2 a_3$ is the spacetime volume of the unit cell and C is a constant. Note that the error term is kept in the lattice action, which means that $S_\Box \equiv S$. This unusual notation is chosen because it is convenient for finding the error terms in all following calculations. Furthermore, instead of the expected $\mathcal{O}(a)$, the error term in the action is $\mathcal{O}(a^2)$. Without doing a detailed calculation, this can easily be seen with symmetry arguments. Since the action is invariant under the transformations $x^\mu \to -x^\mu$, there can only appear even error terms.

3.2. Colored Particle in Cell Method

The colored particle in cell (CPIC) method is used to simulate the dynamics of point-like, color charged particles moving through a grid. The non-Abelian gauge fields that influence them are discretized to the grid points while the particles' positions are tracked in continuous space-coordinates. Therefore, this kind of simulations requires two solvers, namely the so called particle mover or pusher and the field solver. Since the back reaction of the fields onto the particles' velocities is not taken into account in the present simulation, the particle mover is trivial to implement

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}\Delta t, \quad \mathbf{v} = (0,0,\pm c) = const., \tag{3.30}$$

where $\mathbf{x}_i$ is the position of a particle at time step i, $\mathbf{v}$ is the velocity and Δt is the time step. For the fields, an explicit solver, the leapfrog method, is used. Generally it evaluates positions and velocities at different times shifted by half a time step, which can be written as

$$\mathbf{v}_{i+1/2} = \mathbf{v}_{i-1/2} + \mathbf{a}(\mathbf{x}_i)\Delta t, \tag{3.31}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_{i+1/2}\Delta t, \tag{3.32}$$

with the same notation as above and $\mathbf{a}(\mathbf{x}_i)$ being the acceleration.

Applying this to the gauge fields of a CGC simulation, the positions and velocities are replaced by the electric field and gauge links. The equations of motion that will be derived in section 3.5, show that the acceleration or force in eq. (3.31) comes from two sources, the first being the self-interacting nature of non-Abelian gauge theories. This means that not only do the gauge links determine the evolution of the electric field but the links themselves are also influenced by the E-field. The second source is the charge and current density of the colliding nuclei. The important thing to note is that these densities must be defined on the grid points, although the particles carrying the charges and causing the current have continuous coordinates. Therefore an interpolation from the particles' positions to the grid, which preserves charge, energy etc. is necessary. The nearest-grid-point (NGP) interpolation is chosen for this purpose. It maps the whole charge of a particle to its nearest grid point, which is easy to implement and fast. Since the particle's charge is influenced by the surrounding gauge fields, it is necessary to also interpolate the fields from the grid to the particle's position. However, if the particles are placed in the simulation volume in such a way that their transverse coordinates coincide with those of the grid points, the particles will always move along longitudinal grid lines, jumping from one cell to the next. Since the gauge links necessary for the parallel transport of the charges, are defined along the lines between two grid points, their interpolation becomes trivial. One just has to use the link that connects the two grid points between which the particle moves. Although it would be sufficient to have a single particle per cell to probe the charge density, it is better to use more of them, because that leads to smoother current densities. In the present simulation, the number of particles N_p is given by

$$N_p = \frac{a_l}{a_t}, \tag{3.33}$$

where a_l is the lattice spacing in longitudinal direction and a_t is the time step. This ensures that at every time step, a single particle per cell moves far enough to be interpolated to the next grid point.

3.3. Variation of the Action

There are multiple ways to get from the action of a theory to its equations of motion. One option is to use Hamilton's principle. Given the action functional

$$S[\mathbf{q}] := \int_{t_1}^{t_2} L(\mathbf{q}(t), \dot{\mathbf{q}}(t), t)\,\mathrm{d}t, \tag{3.34}$$

with the Lagrangian $L(\mathbf{q}(t), \dot{\mathbf{q}}(t), t)$, it states that the evolution $\mathbf{q}(t)$ of a system, described by n generalized coordinates $\mathbf{q}$, is a stationary point of that action, i.e.

$$\frac{\delta S}{\delta \mathbf{q}(t)} = 0. \tag{3.35}$$

In the case of the CGC action, the generalized coordinates correspond to the gauge fields $\mathcal{A}_{x,\mu}$. Therefore, to find this point, the variation of the action w.r.t. the gauge fields has to be calculated. For this, the following symbol is introduced

$$\delta_x^{a,\mu} := \frac{\partial}{\partial A_{x,\mu,a}}, \tag{3.36}$$

as well as these useful formulas

$$\begin{aligned}
\delta_x^{a,\mu} U_{y,\nu} &= -\,\mathrm{i} g a_\mu t^a U_{x,\mu} \delta_{x,y} \delta_\nu^\mu, && (3.37)\\
\delta_x^{a,\mu} U_{y,\nu}^\dagger &= +\,\mathrm{i} g a_\mu U_{x,\mu}^\dagger t^a \delta_{x,y} \delta_\nu^\mu, && (3.38)\\
\delta_x^{a,\mu} \operatorname{Tr}(U_{y,\alpha\beta}) &= -\,\mathrm{i} g a_\mu \Big(\delta_\alpha^\mu \delta_{x,y} \operatorname{Tr}(t^a U_{x,\alpha\beta}) - \delta_\beta^\mu \delta_{x,y} \operatorname{Tr}(t^a U_{x,\alpha\beta}) \\
&\quad - \delta_\alpha^\mu \delta_{x-\beta,y} \operatorname{Tr}(t^a U_{x,-\beta\alpha}) + \delta_\beta^\mu \delta_{x-\alpha,y} \operatorname{Tr}(t^a U_{x,\beta-\alpha}) \Big). && (3.39)
\end{aligned}$$

Variation w.r.t. $A_{x,0}^a$ yields

$$\delta_x^{a,0} S_\Box = \frac{V}{g^2} \sum_{i=1}^{3} \frac{\mathrm{i} g a_0}{a_0^2 a_i^2} \left(\operatorname{Tr}(t^a U_{x,0i} - \text{h.c.}) - \operatorname{Tr}(t^a U_{x,-i0} - \text{h.c.}) \right) - V \rho_x^a + V \mathcal{O}(a^2) = 0, \tag{3.40}$$

where h.c. is the hermitian conjugate of the previous term. Using the following identities

$$\begin{aligned}
2\mathrm{i} \operatorname{Im} \operatorname{Tr}(A) &= \operatorname{Tr}(A - A^\dagger), && (3.41)\\
2\mathrm{i} \operatorname{Im} \operatorname{Tr}(A^\dagger) &= \operatorname{Tr}(A^\dagger - A) = -\operatorname{Tr}(A - A^\dagger), && (3.42)
\end{aligned}$$

eq. (3.40) can be rewritten as

$$\sum_{i=1}^{3} \frac{1}{a_i^2} \operatorname{Im} \operatorname{Tr}(t^a U_{x,0i} + t^a U_{x,0-i}) + \mathcal{O}(a^3) = -\frac{g a_0}{2} \rho_x^a, \tag{3.43}$$

which is the discrete version of the Gauss constraint, as shown in section 3.4. Variation w.r.t. $A^a_{x,i}$ yields

$$\begin{aligned}\delta^{a,i}_x S_\Box =& \frac{V}{g^2}\left(\frac{\mathrm{i}ga_i}{a_0^2 a_i^2}\left(-\operatorname{Tr}(t^a U_{x,0i} - \text{h.c.}) + \operatorname{Tr}(t^a U_{x,i-0} - \text{h.c.})\right)\right.\\ &\left.- \sum_{j\neq i}\frac{\mathrm{i}ga_i}{a_i^2 a_j^2}\left(\operatorname{Tr}(t^a U_{x,ij} - \text{h.c.}) - \operatorname{Tr}(t^a U_{x,-ji} - \text{h.c.})\right)\right)\\ &- Vj^{a,i}_x + V\mathcal{O}(a^2) = 0. \end{aligned} \tag{3.44}$$

The factor $\frac{1}{2}$ in front of the sum $\sum_{j\neq i}$ gets canceled because the four terms in eq. (3.39) sum up pairwise, using $\operatorname{Tr}(t^a U_{x,ij} - \text{h.c.}) = -\operatorname{Tr}(t^a U_{x,ji} - \text{h.c.})$. Again, with eqs. (3.41) and (3.42), one finds

$$\operatorname{Im}\operatorname{Tr}(t^a U_{x,i0} + t^a U_{x,i-0}) - \sum_{j\neq i}\frac{a_0^2}{a_j^2}\operatorname{Im}\operatorname{Tr}(t^a U_{x,ij} + t^a U_{x,i-j}) + \frac{ga_0^2 a_i}{2} j^{a,i}_x + \mathcal{O}(a^5) = 0, \tag{3.45}$$

which is a true equation of motion and not a constraint.

3.4. Gauss Constraint

The Gauss constraint in the case of a Yang-Mills theory is very similar to the one from electrodynamics. The only differences are that the usual derivative is replaced by the covariant one and instead of a single electric charge, one has multiple color charges:

$$(\mathcal{D}_i \mathcal{E}^i)^a = \rho^a. \tag{3.46}$$

The interpretation of the Gauss law also remains. Each of the color charges ρ_a is the source of its corresponding electric color field E_a.

To show that Equation (3.43) is in fact the Gauss constraint, some rewriting and rearranging is necessary. Using

$$U_{x,\mu-\nu} = U^\dagger_{x-\nu,\nu} U_{x-\nu,\nu\mu} U_{x-\nu,\nu}, \tag{3.47}$$

it reads

$$\sum_{i=1}^{3}\frac{1}{a_i^2}\operatorname{Im}\operatorname{Tr}(t^a U_{x,0i} + t^a U^\dagger_{x-i,i} U_{x-i,i0} U_{x-i,i}) + \mathcal{O}(a^3) = -\frac{ga_0}{2}\rho^a_x. \tag{3.48}$$

To recover the usual form in terms of derivatives of the electric field, one has to expand the plaquette to linear order

$$\begin{aligned}U_{x,0i} &= \exp(-\mathrm{i}ga_0 a_i(\mathcal{F}_{x,0i} + \mathcal{O}(a))) = \exp(-\mathrm{i}ga_0 a_i(E^{a,i}_x t_a + \mathcal{O}(a)))\\ &= \mathbb{1} - \mathrm{i}ga_0 a_i E^{a,i}_x t_a + \mathcal{O}(a^3). \end{aligned} \tag{3.49}$$

Since the components of the electric field $E^a_{x,i}$, the coupling constant and the lattice spacings are real, the following identity holds

$$\operatorname{Im}\operatorname{Tr}(t^a \mathrm{i} g a_0 a_i E^i_{x,b} t^b) = g a_0 a_i E^i_{x,b} \operatorname{Re}\operatorname{Tr}(t^a t^b) = g a_0 a_i E^i_{x,b} \frac{1}{2}\delta^{ab} = \frac{1}{2} g a_0 a_i E^{a,i}_x. \tag{3.50}$$

With the definition

$$\widetilde{E}^{a,i}_{x-i} := (U^\dagger_{x-i,i} E^{b,i}_{x-i} t_b U_{x-i,i})^a \tag{3.51}$$

the Gauss constraint finally reads

$$\boxed{\sum_{i=1}^{3} \frac{E^{a,i}_x - \widetilde{E}^{a,i}_{x-i}}{a_i} + \mathcal{O}(a) = \rho^a_x,} \tag{3.52}$$

which is the discrete version of eq. (3.46).[1] As shown in appendix B, the equations of motion derived in the following section exactly conserve the discrete Gauss constraint.

3.5. Equations of Motion

The equations of motion for the fields have already been derived in section 3.3. The goal now is to rewrite eq. (3.45) in terms of the degrees of freedom used in the simulation. These are the electric field and gauge links. The equation will also be rearranged into a shape better suited for an explicit solver, i.e. the new fields will be expressed only in terms of old fields and densities. Fortunately, this shape will evolve naturally during the calculation.

Similarly to section 3.4 using eqs. (3.47), (3.49) and (3.50), the equations of motion (3.45) can be rewritten in the following way

$$\frac{1}{2} g a_0 a_i (E^{a,i}_x - \widetilde{E}^{a,i}_{x-0}) + \mathcal{O}(a^4) - \sum_{j \neq i} \frac{a_0^2}{a_j^2} \operatorname{Im}\operatorname{Tr}(t^a U_{x,ij} + t^a U_{x,i-j}) + \frac{g a_0^2 a_i}{2} j^{a,i}_x = 0. \tag{3.53}$$

Using the temporal gauge $A^a_{x,0} = 0$, which implies $U_{x,0} = \mathbb{1}$, the shifted electric field $\widetilde{E}^{a,i}_{x-0}$ can be written as

$$\widetilde{E}^{a,i}_{x-0} := U^\dagger_{x-0,0} E^{a,i}_{x-0} U_{x-0,0} = E^{a,i}_{x-0}. \tag{3.54}$$

---

[1] Naively, one would expect a constant error term, because $(E_x - \widetilde{E}_{x-i} + \mathcal{O}(a))/a = \Delta E/a + \mathcal{O}(1)$. However one has to be more careful with differences of error terms. It is important to know that derivation with respect to x does not change a term regarding its order in a, as long as $x \neq a$. Therefore, writing the error term with $R_x = \mathcal{O}(a)$ and disregarding the uninteresting terms with the E-field, yields $(R_x - \widetilde{R}_{x-i})/a = (R_x - R_x - a\mathcal{D}_\mu R_x + \mathcal{O}(a^2))/a = -\mathcal{D}_\mu R_x + \mathcal{O}(a) = \mathcal{O}(a)$.

With this and the rearrangement of some terms, the equation of motion for the electric field finally reads

$$\boxed{E_x^{a,i} = E_{x-0}^{a,i} + \frac{2a_0}{ga_i}\sum_{j\neq i}\frac{1}{a_j^2}\,\mathrm{Im}\,\mathrm{Tr}(t^a U_{x,ij} + t^a U_{x,i-j}) - a_0 j_x^{a,i} + \mathcal{O}(a^2).} \tag{3.55}$$

The temporal gauge also implies that

$$E_x^{a,i} := F_x^{a,i0} = -\partial_0 A_x^{a,i}, \tag{3.56}$$

which leads to

$$\dot{U}_{x,i} = -\mathrm{i}ga_i\partial_0 A^a_{x,i} t_a U_{x,i} = -\mathrm{i}ga_i E_x^{a,i} t_a U_{x,i}. \tag{3.57}$$

Solving this differential equation yields the following equations of motion for the gauge links

$$\boxed{U_{x+0,i} = \exp(-\mathrm{i}ga_0 a_i E_x^{a,i} t_a) U_{x,i}.} \tag{3.58}$$

The point-like particles that represent the external current appearing in the action move along the longitudinal direction of the grid with the speed of light,

$$x^3(t+a_0) = x^3(t) + va_0, \quad v = \pm 1, \tag{3.59}$$

rendering the problem effectively one-dimensional. They carry a charge $\mathcal{Q} = Q^a t_a$, which needs to be interpolated to the grid to get the charge and current density appearing in eqs. (3.52) and (3.55). The nearest-grid-point (NGP) interpolation scheme is used for this, which maps the whole charge of a particle to its nearest grid point. The contribution of a single particle to the charge density is

$$\rho_x^a = \frac{Q_x^a}{a_1 a_2 a_3}, \tag{3.60}$$

where the position index x of the charge Q_x^a does not denote its actual position but the nearest grid point which it gets interpolated to. A current is only induced if a particle crosses the boundary in the middle of the cell. If that happens between the cells at x and $x + a_3$, the only induced current is $j_x^{a,3}$. Using that and evaluating the one-dimensional continuity equation at x and $x + a_3$ as well as $t - a_0$ and t, one finds the following current and charge update for the right moving particle:

$$\boxed{\begin{aligned} j_x^{a,3}(t - \frac{a_0}{2}) &= \frac{a_3}{a_0}\frac{Q_x^a(t-a_0)}{a_1 a_2 a_3}, && (3.61)\\ \mathcal{Q}_{x+3}(t) &= U^\dagger_{x,3}(t - \frac{a_0}{2})\mathcal{Q}_x(t-a_0)U_{x,3}(t - \frac{a_0}{2}). && (3.62) \end{aligned}}$$

For the left moving particle from x to $x - a_3$ the update reads

$$\boxed{\begin{aligned} j_{x-3}^{a,3}(t - \frac{a_0}{2}) &= -\frac{a_3}{a_0}\frac{Q_{x-3}^a(t)}{a_1 a_2 a_3}, && (3.63)\\ \mathcal{Q}_{x-3}(t) &= U_{x-3,3}(t - \frac{a_0}{2})\mathcal{Q}_x(t-a_0)U^\dagger_{x-3,3}(t - \frac{a_0}{2}). && (3.64) \end{aligned}}$$

3.6. Hamiltonian Density

Since the Lagrangian of the CGC model does not explicitly depend on time, its Legendre transform, the Hamiltonian density, equals the energy density of the whole system. However, in the following only the Hamiltonian density of the fields will be calculated. In general, the Legendre transform of the Lagrangian is given by

$$H(\mathbf{p},\mathbf{q}) = \dot{\mathbf{q}}\mathbf{p} - L(\mathbf{q},\dot{\mathbf{q}}), \tag{3.65}$$

with the conjugate momenta $\mathbf{p}$ defined as

$$\mathbf{p} = \frac{\partial L}{\partial \dot{\mathbf{q}}}. \tag{3.66}$$

In case of the CGC action, the conjugate momentum to the gauge field $A_x^{a,i}$ is the negative electric field $-E_x^{a,i}$:

$$\begin{aligned}
p(A_x^{a,i}) &:= \frac{\partial \mathcal{L}}{\partial \partial_0 A_x^{a,i}} \\
&= -\frac{1}{2} F^b_{x,\mu\nu} \frac{\partial}{\partial \partial^0 A_x^{a,i}} F^{\mu\nu}_{x,b} \\
&= -\frac{1}{2} F^b_{x,\mu\nu} \delta^a_b (\delta^\mu_0 \delta^\nu_i - \delta^\nu_0 \delta^\mu_i) \\
&= -\frac{1}{2} (F^a_{x,0i} - F^a_{x,i0}) \\
&= -F^a_{x,0i} = F_x^{a,0i} \\
&= -E_x^{a,i}.
\end{aligned} \tag{3.67}$$

Using eq. (3.56) as well as the definitions of the electric and magnetic fields (eqs. (3.15) and (3.16)), the Hamiltonian density of the Yang-Mills fields in the temporal gauge reads

$$\begin{aligned}
\mathcal{H}_{\mathrm{YM}} &= p\dot{q} - \mathcal{L}_{\mathrm{YM}} \\
&= -E_x^{a,i} \dot{A}^i_{x,a} + \frac{1}{4} F^a_{x,\mu\nu} F^{\mu\nu}_{x,a} + \mathcal{O}(a^2) \\
&= E_x^{a,i} E^i_{x,a} + \frac{1}{4}\left(F^a_{x,0i} F^{0i}_{x,a} + F^a_{x,i0} F^{i0}_{x,a} + F^a_{x,ij} F^{ij}_{x,a} \right) + \mathcal{O}(a^2) \\
&= E_x^{a,i} E^i_{x,a} - \frac{1}{2} E_x^{a,i} E^i_{x,a} + \frac{1}{4} \epsilon_{ijk}\epsilon_{ijm} B_x^{a,k} B^m_{x,a} + \mathcal{O}(a^2).
\end{aligned} \tag{3.68}$$

The contraction of the two epsilon symbols is given by

$$\epsilon_{ijk}\epsilon_{ijm} = 2\delta_{km}, \tag{3.69}$$

and yields

$$\mathcal{H}_{\mathrm{YM}} = \frac{1}{2} E_x^{a,i} E^i_{x,a} + \frac{1}{2} B_x^{a,m} B^m_{x,a} + \mathcal{O}(a^2). \tag{3.70}$$

With the standard single plaquette definition for the magnetic contribution [25, 26]

$$\frac{1}{4}F^a_{x,ij}F_{x,a,ij} = \frac{2}{g^2}\sum_{1\leq i<j\leq 3}\frac{1}{a_i^2 a_j^2}\operatorname{Re}\operatorname{Tr}(\mathbb{1} - U_{x,ij}) + \mathcal{O}(a), \tag{3.71}$$

the discrete Hamiltonian density for the fields reads

$$\boxed{\mathcal{H}_{\mathrm{YM},x} = \frac{1}{2}\sum_{i=1}^{3}E_x^{a,i}E^i_{x,a} + \frac{1}{g^2}\sum_{i=1}^{3}\sum_{j\neq i}\frac{1}{a_i^2 a_j^2}\operatorname{Re}\operatorname{Tr}(\mathbb{1} - U_{x,ij}) + \mathcal{O}(a).} \tag{3.72}$$

3.7. Poynting Vector

The classical continuity equation of electrodynamics $\dot{\rho} + \operatorname{div}\mathbf{j} = 0$ with charge density ρ and current $\mathbf{j}$ can also be applied to conserved quantities other than the electric charge. In particular the conservation of energy is interesting for most systems. Its corresponding continuity equation in electrodynamics is called Poynting's Theorem and reads

$$\dot{\mathcal{H}} + \operatorname{div}\mathbf{S} = 0, \tag{3.73}$$

where $\mathcal{H}$ is the continuum Hamiltonian density of the fields and $\mathbf{S}$ is the Poynting vector. Since the latter has taken the place of the current density $\mathbf{j}$, it corresponds to the flow of the energy density $\mathcal{H}$. The Poynting vector is usually given by

$$\mathbf{S} = \mathbf{E}\times\mathbf{B}. \tag{3.74}$$

In the presence of an external current density $\mathbf{j}$, an additional term must be added to Poynting's Theorem, which accounts for the work that is done when charges are moved through an electric field $\mathbf{E}$. In this case, Poynting's theorem is given by

$$\dot{\mathcal{H}} + \operatorname{div}\mathbf{S} + \mathbf{j}\cdot\mathbf{E} = 0 \tag{3.75}$$

with the Poynting vector still given by eq. (3.74). The obvious approach to get the discrete version of $\mathbf{S}$ on the lattice is to use eq. (3.74) and just substitute the continuum variables with their lattice counterparts. However, one can also plug the discrete expressions for the Hamiltonian density, the current and the electric field into eq. (3.75) and then try to extract a formula for $\mathbf{S}$. This should give a discretization of the Poynting vector that best fits Poynting's theorem. For the case without particles and external currents, this has already been studied in more detail in [13].

First, the time derivative of the discrete Hamiltonian density has to be calculated:

$$\dot{\mathcal{H}}_{\mathrm{YM},x} = \sum_{i=1}^{3}E_x^{a,i}\dot{E}^i_{x,a} + \frac{1}{g^2}\sum_{i=1}^{3}\sum_{j\neq i}\frac{1}{a_i^2 a_j^2}\operatorname{Re}\operatorname{Tr}(-\dot{U}_{x,ij}) + \mathcal{O}(a). \tag{3.76}$$

The equations of motion for the electric field (3.55) can be rearranged to get the discrete time derivative of $E^a_{x,i}$ in the temporal gauge

$$\dot{E}_x^{a,i} + \mathcal{O}(a) = \frac{E_x^{a,i} - E_{x-0}^{a,i}}{a_0} = \frac{2}{g a_i} \sum_{j \neq i} \frac{1}{a_j^2} \operatorname{Im}\operatorname{Tr}(t^a U_{x,ij} + t^a U_{x,i-j}) - j_x^{a,i} + \mathcal{O}(a). \tag{3.77}$$

Using eqs. (3.23) and (3.57), the real trace of the time derivative of the plaquette reads

$$\begin{aligned} \operatorname{Re}\operatorname{Tr}(\dot{U}_{x,ij}) &= -g \operatorname{Re}\operatorname{Tr}\big(\mathrm{i}(a_i E_x^{a,i} t_a U_{x,ij} + a_j E_{x+i}^{a,j} t_a U_{x+i,j-i} \\ &\qquad -a_i E_{x+j}^{a,i} t_a U_{x+j,-ji} - a_j E_x^{a,j} t_a U_{x,ij})\big) \\ &= \ g \operatorname{Im}\operatorname{Tr}(a_i E_x^{a,i} t_a U_{x,ij} + a_j E_{x+i}^{a,j} t^a U_{x+i,j-i} \\ &\qquad -a_i E_{x+j}^{a,i} t_a U_{x+j,-ji} - a_j E_x^{a,j} t_a U_{x,ij}). \end{aligned} \tag{3.78}$$

Putting everything together yields

$$\begin{aligned} \dot{\mathcal{H}}_{\mathrm{YM},x} &= \frac{1}{g} \sum_{i=1}^{3} \sum_{j \neq i} \frac{2}{a_i^2 a_j^2} \operatorname{Im}\operatorname{Tr}(a_i E_x^{a,i} t_a U_{x,ij} + a_i E_x^{a,i} t_a U_{x,i-j}) \\ &\quad - \frac{1}{g} \sum_{i=1}^{3} \sum_{j \neq i} \frac{1}{a_i^2 a_j^2} \operatorname{Im}\operatorname{Tr}(a_i E_x^{a,i} t_a U_{x,ij} + a_j E_{x+i}^{a,j} t_a U_{x+i,j-i} \\ &\quad - a_i E_{x+j}^{a,i} t_a U_{x+j,-ji} - a_j E_x^{a,j} t_a U_{x,ij}) - \sum_{i=1}^{3} E_x^{a,i} j_{x,a}^{i} + \mathcal{O}(a) \\ &= \frac{1}{g} \sum_{i=1}^{3} \sum_{j \neq i} \frac{1}{a_i^2 a_j^2} \operatorname{Im}\operatorname{Tr}(2 a_i E_x^{a,i} t_a U_{x,ij} + 2 a_i E_x^{a,i} t_a U_{x,i-j} - a_i E_x^{a,i} t_a U_{x,ij} \\ &\quad - a_j E_{x+i}^{a,j} t_a U_{x+i,j-i} + a_i E_{x+j}^{a,i} t_a U_{x+j,-ji} + a_j E_x^{a,j} t_a U_{x,ij}) \\ &\quad - \sum_{i=1}^{3} E_x^{a,i} j_{x,a}^{i} + \mathcal{O}(a). \end{aligned} \tag{3.79}$$

Writing out the sums explicitly and collecting similar terms gives

$$\begin{aligned} \dot{\mathcal{H}}_{\mathrm{YM},x} &= \frac{1}{g} \operatorname{Im}\operatorname{Tr}\Bigg(\frac{1}{a_1^2 a_2^2} \Big(a_1 E_x^{a,1} t_a U_{x,12} + 2 a_1 E_x^{a,1} t_a U_{x,1-2} - a_2 E_{x+1}^{a,2} t_a U_{x+1,2-1} \\ &\quad + a_1 E_{x+2}^{a,1} t_a U_{x+2,-21} + a_2 E_x^{a,2} t_a U_{x,12} + (1 \leftrightarrow 2) \Big) + (2 \to 3) + (1 \to 2) \Bigg) \\ &\quad - \sum_{i=1}^{3} E_x^{a,i} j_{x,a}^{i} + \mathcal{O}(a) \\ &= \frac{1}{g} \operatorname{Im}\operatorname{Tr}\Bigg(\frac{1}{a_1 a_2} \Bigg(\frac{E_x^{a,1} t_a U_{x,12} + 2 E_x^{a,1} t_a U_{x,1-2} + E_{x+2}^{a,1} t_a U_{x+2,-21}}{a_2} \\ &\quad + \frac{-E_{x+2}^{a,1} t_a U_{x+2,1-2} + E_x^{a,1} t_a U_{x,21}}{a_2} + \frac{-E_{x+1}^{a,2} t_a U_{x+1,2-1} + E_x^{a,2} t_a U_{x,12}}{a_1} \end{aligned}$$

$$+\frac{E_x^{a,2}t_aU_{x,21}+2E_x^{a,2}t_aU_{x,2-1}+E_{x+1}^{a,2}t_aU_{x+1,-12}}{a_1}\Bigg)+(2\to 3)+(1\to 2)\Bigg)$$
$$-\sum_{i=1}^{3}E_x^{a,i}j_{x,a}^{i}+\mathcal{O}(a). \tag{3.80}$$

Using eq. (3.41), $\operatorname{Im}\operatorname{Tr}(U_{x,ij})=-\operatorname{Im}\operatorname{Tr}(U_{x,ji})$, some terms cancel and some can be combined yielding

$$\begin{aligned}\dot{\mathcal{H}}_{\mathrm{YM},x}=\frac{2}{g}\operatorname{Im}\operatorname{Tr}\Bigg(\frac{1}{a_1a_2}\Bigg(&\frac{E_{x+2}^{a,1}t_aU_{x+2,1-2}-E_x^{a,1}t_aU_{x,1-2}}{a_2}\\ &+\frac{E_{x+1}^{a,2}t_aU_{x+1,2-1}-E_x^{a,2}t_aU_{x,2-1}}{a_1}\Bigg)+(2\to 3)+(1\to 2)\Bigg)\\ &-\sum_{i=1}^{3}E_x^{a,i}j_{x,a}^{i}+\mathcal{O}(a).\end{aligned} \tag{3.81}$$

Apart from the term $\sum_{i=1}^{3}E_x^{a,i}j_{x,a}^{i}$, this result can easily be rewritten as the discrete divergence of a vector which by comparison with eq. (3.75) is the Poynting vector $\mathbf{S}$,

$$\boxed{S_x^i=\frac{2}{g}\sum_{j\neq i}\frac{1}{a_ia_j}\operatorname{Im}\operatorname{Tr}(E_x^{a,j}t_aU_{x,j-i}).} \tag{3.82}$$

In the limit of small a, the discrete Poynting vector becomes

$$\begin{aligned}S_x^i&=\frac{2}{g}\sum_{j\neq i}\frac{1}{a_ia_j}\operatorname{Im}\operatorname{Tr}(E_x^{a,j}t_a(\mathbb{1}-\mathrm{i}ga_ia_jF_{x,ij}^{b}t_b+\mathcal{O}(a^3)))\\ &=-2\sum_{j\neq i}\operatorname{Re}\operatorname{Tr}(E_x^{a,j}t_aF_{x,ij}^{b}t_b)+\mathcal{O}(a)\\ &=2\sum_{j\neq i}\sum_{k=1}^{3}\operatorname{Re}\operatorname{Tr}(E_x^{a,j}\epsilon_{ijk}B_x^{b,k}t_at_b)+\mathcal{O}(a)\\ &=\sum_{j\neq i}\sum_{k=1}^{3}\epsilon_{ijk}E_x^{a,j}B_x^{a,k}+\mathcal{O}(a)=(\mathbf{E}_x^a\times\mathbf{B}_x^a)_i+\mathcal{O}(a),\end{aligned} \tag{3.83}$$

which is the expected result of the continuum.

3.8. Initial Conditions

The initial conditions of the simulation need to describe the nuclei before they collide. For this, a model of the nuclei's charge configuration has to be found and the electric fields and gauge links have to be determined in a consistent manner. The external color current density for infinitely Lorentz contracted nuclei was already established in section 2.3. In Cartesian coordinates it reads

$$j_a^\mu(x) = \delta(t-z)\hat{\rho}_a(x_T)s^\mu, \tag{3.84}$$

where $s^\mu = (1,0,0,1)^\mu$ for a charge configuration $\hat{\rho}_a(x_T)$, traveling in the positive z-direction, at the speed of light. Note, that $\hat{\rho}_a(x_T)$ is now the 2-dimensional charge density in the transverse plane, not to be confused with the charge density in 3 dimensions, ρ_x. For the initial conditions of the present simulation, the delta function in eq. (3.84) is changed to an envelope function $f(t-z)$ to be able to describe nuclei with finite longitudinal thickness. As a model for the charge configuration of colliding heavy ions, the McLerran-Venugopalan (MV) model is chosen [22–24]. It uses a gauge-invariant, Gaussian probability functional $W[\hat{\rho}]$ with zero mean and the two-point correlation function

$$\langle \hat{\rho}_a(x_T)\hat{\rho}_b(y_T)\rangle = g^2\mu^2\delta_{ab}\delta^{(2)}(x_T-y_T), \tag{3.85}$$

where g is the coupling and μ is the MV model parameter controlling average color charge density. The subsequent derivation of the color current and the corresponding fields follows the one in [8, section 2.3].

In order to find a field configuration that is consistent with the color current, the following ansatz is used

$$j_a^\mu = f(t-z)\hat{\rho}_a(x_T)s^\mu, \tag{3.86}$$

$$A_a^\mu = f(t-z)\hat{\varphi}_a(x_T)s^\mu, \tag{3.87}$$

where A_a^μ is the gauge field and $\hat{\varphi}_a$ needs to be determined. For the arbitrary envelope function a Gaussian profile

$$f(t-z) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(t-z)^2}{2\sigma^2}} \tag{3.88}$$

is chosen, where the given width σ is proportional to the thickness of the nucleus in the laboratory frame. Using the Lorenz gauge $\partial_\mu A_a^\mu = 0$ and putting the ansatz into the Yang-Mills equations of motion

$$\mathcal{D}_\mu^{ab}F_b^{\mu\nu} = j^{a,\nu}, \tag{3.89}$$

yields a Poisson equation for the gauge fields

$$-\Delta_T A^{a,\nu} := -\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)A^{a,\nu} = j^{a,\nu}. \tag{3.90}$$

This is because nonlinear terms vanish due to $s^\mu s_\mu = 0$ and the time dependence drops out since $s^\mu\partial_\mu f(t-z) = 0$. With the ansatz (3.86) and (3.87), as well as using the formal notion of an inverse Laplace operator, the only part left of eq. (3.90) is

$$\hat{\varphi}_a(x_T) = -\frac{\hat{\rho}_a(x_T)}{\nabla_T^2}. \tag{3.91}$$

When solving this equation in momentum space, an infrared (IR) regulator m and an ultraviolet (UV) cutoff Λ can be introduced in the following way:

$$\hat{\varphi}_a(k_T) = \begin{cases} \frac{\hat{\rho}_a(k_T)}{|k_T|^2+m^2} & , \quad |k_T| \leq \Lambda, \\ 0 & , \quad |k_T| > \Lambda, \end{cases} \tag{3.92}$$

where $\hat{\varphi}_a(k_T)$ and $\hat{\rho}_a(k_T)$ are the Fourier transformations of $\hat{\varphi}_a(x_T)$ and $\hat{\rho}_a(x_T)$ respectively. The addition of the infrared regulator m causes a finite correlation in the transverse directions with a length of order m^{-1}. Neither of the two changes violates any equation of motion or constraint, because they can also be included via a redefined charge density

$$\hat{\rho}'_a(k_T) = \frac{|k_T|^2}{|k_T|^2 + m^2}\Theta(\Lambda - |k_T|)\hat{\rho}_a(k_T), \tag{3.93}$$

which satisfies the unmodified Poisson equation

$$\hat{\varphi}_a(k_T) = \frac{\hat{\rho}'_a(k_T)}{|k_T|^2}. \tag{3.94}$$

An interesting note is that an IR regulator of $m^2 > 0$ leads to global color neutrality, i.e.

$$\hat{\rho}'(k_T = 0) = 0. \tag{3.95}$$

Using the obtained solution for $\hat{\varphi}_a(x_T)$ and the definitions (3.9) and (3.15), the electric field reads

$$\begin{aligned} E_a^{i=1,2} &= \partial^i A_a^0 \\ &= f(t-z)\partial^i \hat{\varphi}_a(x_T), \end{aligned} \tag{3.96}$$

$$\begin{aligned} E_a^3 &= \partial^3 A_a^0 - \partial^0 A_a^3 \\ &= \left(-\partial_z f(t-z) - \partial_t f(t-z)\right) \hat{\varphi}_a(x_T) \\ &= 0, \end{aligned} \tag{3.97}$$

because $A_a^{i=1,2} = 0$ and $A_a^0 = A_a^3$. This means that gauge fields with only longitudinal and temporal components in the Lorenz gauge lead to an E-field with only transverse components. Since the lattice equations of motion were solved in the temporal gauge, the solutions for the initial fields need to be transformed to this gauge as well. Therefore the gauge transformation

$$\mathcal{A}'^{\mu}(x) = V(x)(\mathcal{A}^{\mu}(x) + \frac{\mathrm{i}}{g}\partial^{\mu})V^{\dagger}(x) \tag{3.98}$$

with $\Omega_x = V(x)$ must ensure that $\mathcal{A}'^0(x) \equiv 0$. This yields the following equation for $V(x)$,

$$\begin{aligned} \partial^0 V^{\dagger}(x) &= \partial_t V^{\dagger}(x) \\ &= \mathrm{i}g\mathcal{A}^0(x)V^{\dagger}(x). \end{aligned} \tag{3.99}$$

Since the gauge fields eq. (3.87) commute at different times, this is solved by

$$V^\dagger(t,x,y,z) = \exp(ig\hat{\varphi}_a t^a F(t,z)), \tag{3.100}$$

with $F(t,z) \coloneqq \int_{-\infty}^{t} f(t'-z)\,\mathrm{d}t'$, but without the need of a time ordered exponential. When the integral is rewritten in a more symmetric form

$$\int_{-\infty}^{t} f(t'-z)\,\mathrm{d}t' = \int_{-\infty}^{t-z} f(t'-z)\,\mathrm{d}(t'-z) \eqqcolon F(t-z), \tag{3.101}$$

it is easy to see, that

$$\partial^0 F(t-z) = \partial_t F(t-z) = -\partial_z F(t-z) = \partial^3 F(t-z), \tag{3.102}$$

and therefore

$$\partial^0 V^\dagger(x) = \partial^3 V^\dagger(x). \tag{3.103}$$

Again, using $A_a^{i=1,2} = 0$ and $A_a^0 = A_a^3$, the gauge fields in the temporal gauge read

$$\mathcal{A}'^{\mu=0,3}(x) = 0, \tag{3.104}$$

$$\mathcal{A}'^{\mu=1,2}(x) = \frac{\mathrm{i}}{g} V(x)\partial^\mu V^\dagger(x). \tag{3.105}$$

The color current must also be transformed $j'^\mu = V j^\mu V^\dagger$, but the electric field is easier calculated using eq. (3.56). It is important to note that in the temporal gauge one ends up with purely transverse gauge fields. Additionally, because of the integral over the envelope function $f(t-z)$, these fields do not vanish behind the nucleus, but remain at a constant value. Since these field configurations are pure gauge and can be transformed to vacuum, they do not carry any energy. However, they require the use of fixed boundary conditions in the longitudinal direction.

For a given simulation, the concrete parameters of the initial conditions are calculated in the following way. The contracted thickness of a nucleus is chosen to be 4σ where σ is the width of the longitudinal envelope function eq. (3.88). Therefore the relation between the width σ, the nuclear radius R and the Lorentz factor γ is given by

$$\gamma = \frac{R}{2\sigma}. \tag{3.106}$$

With a known center-of-mass energy per nucleon pair $\sqrt{s_{\mathrm{NN}}}$ the gamma factor also reads

$$\gamma = \frac{\sqrt{s_{\mathrm{NN}}}}{2m_N}, \tag{3.107}$$

where $m_N \approx 1\,\mathrm{GeV}$ is the nucleon mass. Putting everything together yields a formula for the width of the envelope function:

$$\sigma = R\frac{m_N}{\sqrt{s_{\mathrm{NN}}}}. \tag{3.108}$$

The gauge coupling is chosen to be $g = 2$, which is common in CGC literature [11, 27–29]. For early simulations, the parameter μ was determined according to an estimate of McLerran and Venugopalan,

$$\mu^2 = 1.1 A^{1/3} \mathrm{fm}^{-2}, \tag{3.109}$$

where A is the mass number of the nuclei and the gauge group is SU(3). Using gold with $A = 197$ yields

$$\mu = 0.505\,\mathrm{GeV}. \tag{3.110}$$

Results obtained using this value should be taken cautiously, because the simulation uses for simplicity the gauge group SU(2). For later simulations more accurate estimations are used like the following one for the saturation momentum

$$Q_s^2 \approx (\sqrt{s_{\mathrm{NN}}})^{0.25}\,\mathrm{GeV}^2, \tag{3.111}$$

with $\sqrt{s_{\mathrm{NN}}}$ given in GeV [27, 30, 31]. As suggested in [32], a simple relation between Q_s and the MV model parameter μ is chosen as

$$0.75 g^2 \mu \approx Q_s. \tag{3.112}$$

With a collision energy of $\sqrt{s_{\mathrm{NN}}} = 200\,\mathrm{GeV}$, this results in an MV model parameter of

$$\mu \approx 0.6464\,\mathrm{GeV}. \tag{3.113}$$

4. Implementation

The original Java version used the object-oriented programming paradigm quite well to create clear, modular, easily expendable code. This made it very simple to exchange or add e.g. new diagnostic modules or try new variants of existing solvers. The abstraction of the physical quantities was also done in a straightforward way, making it easy to translate equations to code. However, as the code base grew over time, there were several modules and parts of the code that were not used anymore and the effort to couple everything very loosely sometimes led to high class hierarchies. The C++ code tries to inherit the clarity and modularity of the Java version, but reduces and simplifies the hierarchy levels. As shown in fig. 4.1, it is segmented into four categories: simulation core, initial conditions, diagnostics and tests. Each of these will be explained in more detail in the following sections. Since C++ allows operator overloading, implementing and reading mathematical expressions in code is even more straightforward and easier than in Java. The standard template library (STL) is mainly used for handling resources, like memory. If the size of a data field is not known at compile time or it is too large for the stack, it is implemented as a `std::vector`, which uses dynamically allocated memory and can change its size during runtime. Otherwise a `std::array` the size of which must be determined at compile time is used. In general, pointers are avoided where possible. However, if necessary, smart pointers like `shared_ptr` and `unique_ptr` are used instead of naked pointers. All this allows consistent and convenient resource handling without noticeable loss in execution speed, while also preventing memory leaks, dangling pointers and the like. Compared to the garbage collector of Java, this should also be much more resource efficient. The whole source code can be found on GitLab [33]. The documentation is generated with doxygen and published online with the help of GitLab Pages [34]. A final, important note is that in contrast to the descriptions and calculations in previous sections, within the code the longitudinal direction is chosen to be x^1 instead of x^3.

4.1. Simulation Core

The simulation core contains the bare necessities like the grid settings, the data fields for the degrees of freedom, the equation of motion solver, a class for output data files, etc.

The file *CustomTypes.h* defines two new types called `rational` and `index`. They are used throughout the whole code for any non-integer data and for integer data representing

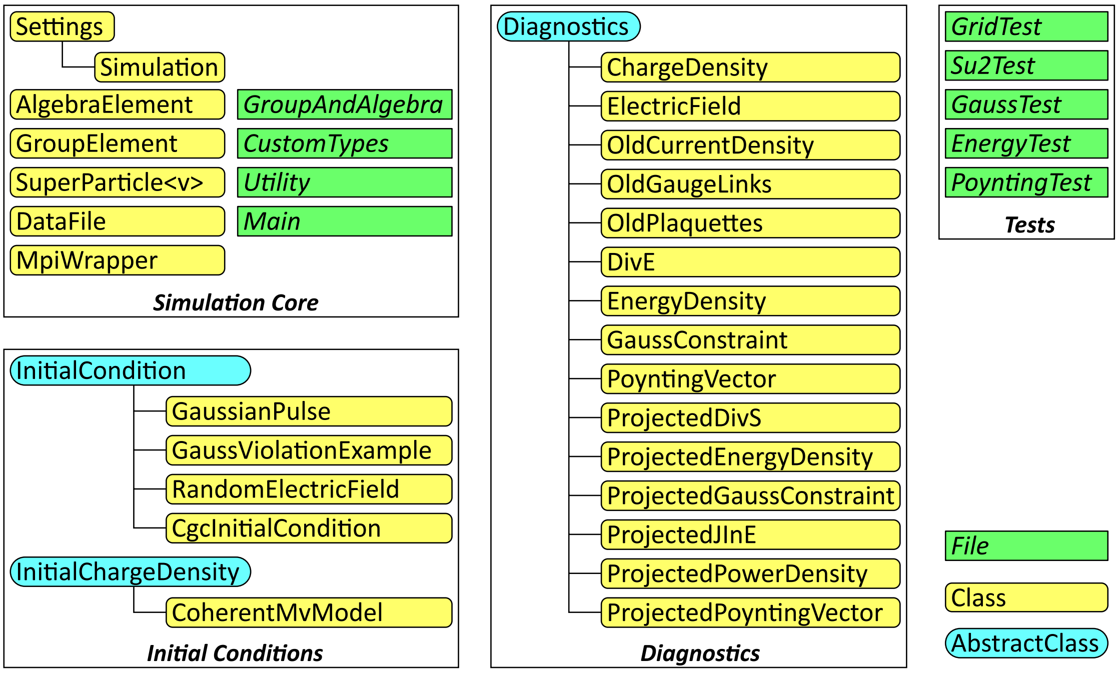


Figure 4.1.: Overview of all classes (yellow) and files (green) of the C++ code. Abstract base classes are shown in blue. Lines connect the derived classes with their parent class.

an index of some container, respectively. The definition of `rational` depends on the preprocessor constant `USE_FLOAT` and is either `float` or `double`. This allows for a convenient change of the floating point precision of the whole simulation.

The classes `AlgebraElement` and `GroupElement` are defined in *Su2.h* and implement the necessary $\mathfrak{su}(2)$ and SU(2) elements. An `AlgebraElement` consists of three member variables v_a defined by

$$v = \frac{1}{2} v_a \sigma_a, \tag{4.1}$$

where v is an algebra element in the fundamental representation and σ_a are the Pauli matrices. A `GroupElement` consists of four member variables, g_a and g_0, which are defined by

$$g = g_0 \mathbb{1} + \mathrm{i} g_a \sigma_a, \tag{4.2}$$

where g is a group element in the fundamental representation, $\mathbb{1}$ is the unit matrix and i the imaginary unit. With this representation a proper group element is only defined if the following identity holds:

$$g_0^2 + g_1^2 + g_2^2 + g_3^2 = 1. \tag{4.3}$$

The classes for group and algebra elements also contain overloaded operators for addition, multiplication, etc. as well as functions for operations like exp, log, hermitian conjugation, setting and getting single member variables and so on. Furthermore the group dimension and the number of generators are defined.

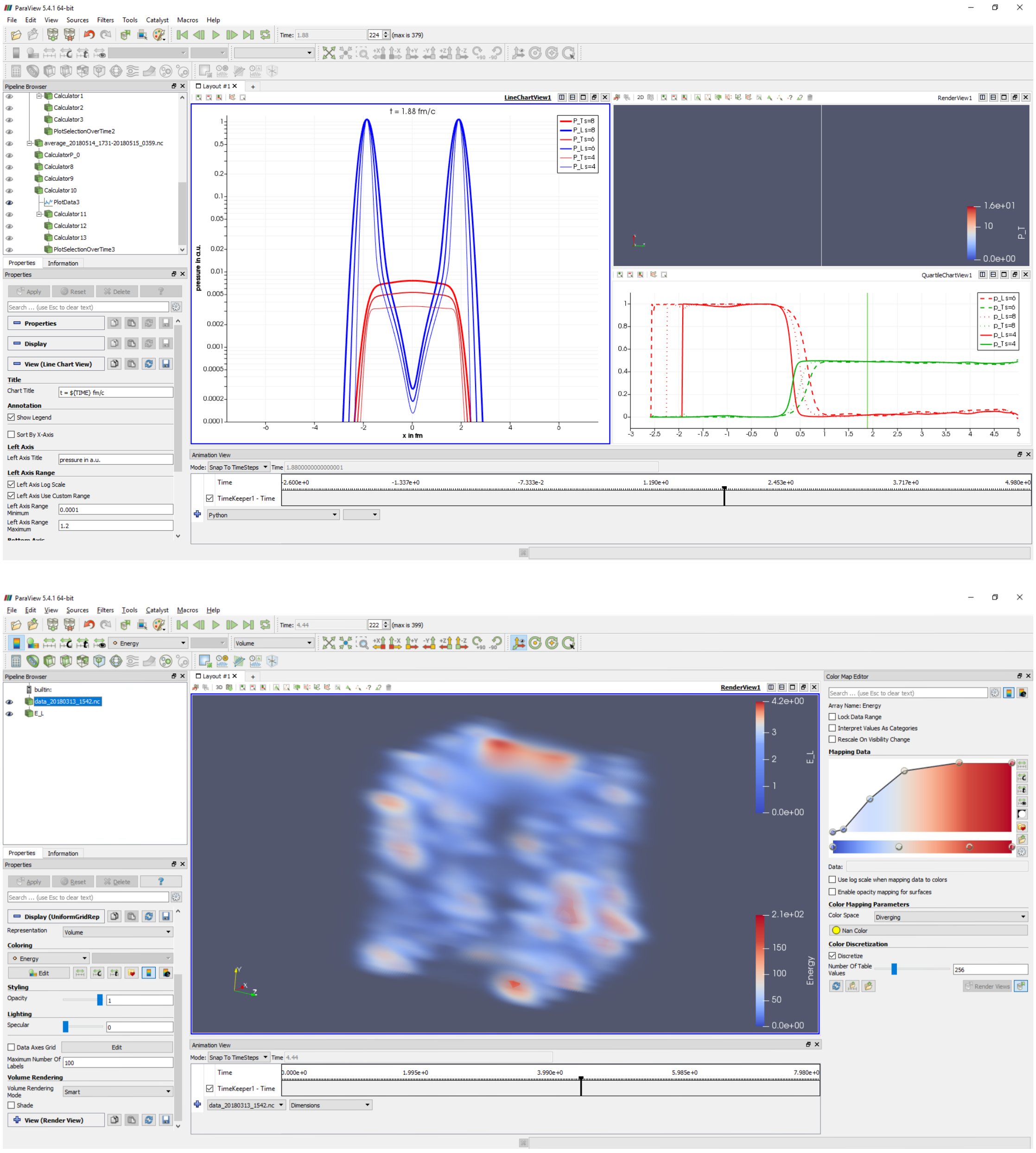
It is possible to implement similar classes for other groups like SU(3). For convenient switching between different implementations of `GroupElement` and `AlgebraElement`, the file *GroupAndAlgebra.h* is introduced and acts as an abstraction layer. Every file that uses group and algebra elements only includes this header without caring about the concrete implementation of the elements. The latter is chosen by the file that is included in *GroupAndAlgebra.h*. In the present case, that is *Su2.h*. If a different implementation should be used, *Su2.h* has to be replaced by some other file, e.g. *Su3.h* (if available).

`SuperParticle<v>` is a class template describing all color charged, point-like particles in the same transverse plane. This is possible because – as discussed in sections 3.2 and 3.5 – all particles move forever at the speed of light, in the longitudinal direction. The template parameter `v` is the velocity of the super particle and can either be `Velocity::positive` or `Velocity::negative`, corresponding to right and left moving particles respectively. The class template contains a `vector` of `AlgebraElement`s that describes the dimensionless charges of the particles and a member called `unitFactorQ` to transform the charges to physical quantities, which is given by

$$\texttt{unitFactorQ} = \frac{a_2 a_3}{g a_0 a_1}, \tag{4.4}$$

where g is the coupling constant, a_0 the time step and a_i are the lattice spacings. It also has member variables for the previous, current and next longitudinal position as well as an inter-cell position. The latter is necessary because it is possible to put more than one particle into a single cell. The provided `update()` function moves the super particle by changing the inter-cell position. If the particles jumped to the next cell, the previous, current and next position are updated as well. In case the particle hits the longitudinal boundary of the simulation grid, it will no longer be used in the simulation. Otherwise, the charges are parallel transported according to eqs. (3.62) and (3.64) and the current density is calculated as in eqs. (3.61) and (3.63). Finally the charges are interpolated to the grid points using nearest-grid-point interpolation.

The `DataFile` class is a handler for a NetCDF file to save diagnostic data. The network common data form (NetCDF) is a set of software libraries and self-describing, machine-independent data formats for array-oriented scientific data [35]. Since it is a binary data format it uses less memory compared to a text file and does not suffer form precision loss if the file uses the same types as the code. However, the main reason for choosing this file format is that it can easily be accessed in parallel when using MPI. This is important, because when doing highly parallelized computations with many processes, serial data access will become the main performance bottleneck. The disadvantage of not being human-readable is compensated by the large number of programs available for manipulating and displaying NetCDF files [36]. For the analysis of the results of the present C++ simulator a common and powerful tool called ParaView was used. It allows to display data in up to three dimensions and can also create animations as well as plots over time. The data can be further processed by using common operations and functions to calculate new quantities derived from the available ones. Features like the computation of isosurfaces are also present. Figure 4.2 shows two screenshots of the program illustrating some of its capabilities for two- and three-dimensional plots. The self-describing nature of the NetCDF file format comes in the form of so called *dimensions*, *attributes* and *variables*. Dimensions define the size that data arrays can have in a particular direction. It is possible to define one unlimited dimension, which – as the name suggests – can grow arbitrarily in size. Attributes can be tied to variables or the whole file and are used for further description. Variables are arrays of simple data types like `int`, `double`, `char`, etc. Each one has a distinct name and shape, given by a list of dimensions. Variables that have the same name as a dimension are called coordinate variable and usually define a physical coordinate, corresponding to the dimension. There exist conventions for naming them and some programs treat them in a special way. The `DataFile` class deals with all the dimensions, attributes and conventions automatically. Its constructor creates a NetCDF file with dimensions for the x-, y- and z-coordinate with the correct lengths, namely the grid sizes in the corresponding direction. Time is added as an unlimited dimension. Coordinate variables with the correct names and unit attributes are added for all four dimensions as well. All settings and initial conditions are added as global attributes, which allows to exactly reproduce a simulation from the data file. Finally, variables for all chosen diagnostics are added and the spacial coordinate variables are filled with data in physical units. The `DataFile` class also provides an easy to use function for writing to the file that hides all C-pointers and NetCDF functions behind nice STL containers.



Time: 1.88 224 (max is 379)

Chart Title: t = ${TIME} fm/c

☑ Show Legend

☐ Sort By X-Axis

Left Axis Title: pressure in a.u.

☑ Left Axis Log Scale

☑ Left Axis Use Custom Range

Left Axis Range Minimum: 0.0001

Left Axis Range Maximum: 1.2

Mode: Snap To TimeSteps Time: 1.88000000000001

Time: 4.44 222 (max is 399)

Representation: Volume

Coloring: Energy

Opacity: 1

Specular: 0

☐ Data Axes Grid

Maximum Number Of Labels: 100

Volume Rendering Mode: Smart

☐ Shade

Array Name: Energy

☐ Lock Data Range

☐ Interpret Values As Categories

☐ Rescale On Visibility Change

☐ Use log scale when mapping data to colors

☐ Enable opacity mapping for surfaces

Color Space: Diverging

☑ Discretize

Number Of Table Values: 256

Mode: Snap To TimeSteps Time: 4.44

Figure 4.2.: Two screenshots of ParaView. A tool which is used to analyze and study the C++ simulations. The top panel shows multiple two-dimensional plots of for analyzing pressure anisotropy, while the bottom panel depicts the three-dimensional distribution of the energy density shortly after a collision.

The `Settings` class has members for the grid size, lattice spacings, time step, number of time steps, initial time, coupling constant and number of particles per cell. It also contains `vectors` that hold the initial conditions and diagnostics of the simulation. Furthermore, it is possible to choose between periodic or fixed boundary conditions in the longitudinal direction. Fixed boundary conditions are simply implemented by not updating the left- and rightmost transverse plane of the simulation grid. Functions are provided that return the first and one past the last position index of the active part of the grid, which will get updates. This makes it easy to implement `for`-loops over this active grid.

The `Simulation` class is derived from `Settings` to allow a less cumbersome access to its members. The main purpose of this class is to provide the `vector`s containing the fields and particles as well as the equation of motion solver. Since a leapfrog scheme is used, the electric field and the charge density are stored at integer time steps, while the gauge links and current density are stored at half a time step in the past and the future. All of them are stored dimensionlessly, but member variables are provided to convert them to physical units. They are defined as

$$
\begin{aligned}
\texttt{unitFactorE}_i &= (ga_0a_i)^{-1}, && (4.5)\\
\texttt{unitFactorB}_i &= (ga_ja_k)^{-1}, \quad j \neq i \neq k \neq j && (4.6)\\
\texttt{unitFactorRho} &= (ga_0a_1^2)^{-1}, && (4.7)\\
\texttt{unitFactorJ} &= (ga_0^2a_1)^{-1}. && (4.8)
\end{aligned}
$$

The length of the `vector`s containing the fields equals the number of cells in the simulation grid. Since the electric field and gauge links have multiple components, each of their `vector` elements itself is an array of `AlgebraElements` or `GroupElements`. The equation of motion solver as well as many diagnostics use plaquettes in their calculation. It is therefore possible to store some of the plaquettes in addition to the gauge links, to prevent them from being computed every time they appear in a calculation. Although there are 24 different plaquettes, only 12 are used in the simulation and only three are stored if the preprocessor constant `STORE_PLAQUETTES` is set to 1. These are the ones where the indexes are positive and the first index is smaller than the second one. The rest of the plaquettes can either be calculated by hermitian conjugation or by using

$$U_{x,i-j} = (U_{x-j,ij}U_{x-j,j})^\dagger U_{x-j,j}. \tag{4.9}$$

This gives a good trade-off between memory usage and execution speed [13].

The `evolve()` function implements the equation of motion solver. First it swaps old and new gauge links and plaquettes, then it updates the fields and links according to eqs. (3.55) and (3.58). After that, the new plaquettes are calculated. Finally, the particles are updated and the charge and current density are interpolated to the grid. In addition to the solver, the `Simulation` class also provides functions that are frequently used in calculations, like the ones for shifting position indexes, getting specific gauge links, calculating plaquettes and the magnetic field, etc. It also contains functions that apply all initial conditions and execute all diagnostics stored in the corresponding `vector`s of `Settings` as well as writing all the diagnostic data to the output data file.

The class `MpiWrapper` is a very simple and lightweight C++ wrapper for MPI. It is implemented as a singleton, which guarantees that only a single MPI instance exists at a time. It is also ensured that initialization and finalization of the MPI environment is always done at the right time. Apart from that, only a few of the most common attributes of the MPI environment are provided.

The file *Utility.h* defines a new namespace called `utility` which contains a set of convenient functions that do not fit in any other class or file. This includes a function for solving the 2-dimensional Poisson equation, writing OpenMP info to the screen, a thread-safe function to get the local time and a parallel version of `std::fill()`.

4.2. Initial Conditions

In the following, only the newly added initial conditions are discussed. For a description of `RandomElectricField`, `GaussianPulse` and `GaussViolationExample`, see the preceeding project thesis [13].

`InitialChargeDensity` is an abstract base class from which all charge densities, required for the description of high energy nuclei in the CGC framework, derive. Here the term charge density refers to the 3-dimensional ρ_x. All derived classes must implement the function `initialize()`, which returns the charge density and `getDescription()`, which returns a `string` containing all useful information about themselves. This `string` will be saved in a data file.

`CoherentMvModel` is the only child of `InitialChargeDensity` so far. It implements the charge distribution described in section 3.8. The reason for it being called *coherent* MV model is that the charges do not change in color along the longitudinal direction. It would be possible to release this restriction and implement a changing longitudinal structure. This is however out of scope for the current thesis. When creating an Object of the `CoherentMvModel` class, the user has to specify the center point and width σ of the Gauss profile, which is the longitudinal envelope function, as well as the parameter μ, controlling the average charge density. It is also possible to define the seed of the 64-bit Mersenne Twister which is used as a pseudorandom number generator in this class. The longitudinal width of the volume, in which the charge density is defined, is 16σ. This ensures that ρ_x is sufficiently small at the boundaries and can be neglected outside of them.

All initial conditions derive from the abstract base class `InitialCondition` and need to implement the function `apply()`, which sets all E-fields and gauge links and spawns `SuperParticle`s where necessary. In order to use a particular initial condition for a simulation, one has to add a `unique_ptr` or `shared_ptr` to the required class to the `initialConditions` `vector` of the `Settings` class. Calling `applyInitialConditions()` sets all initial conditions as specified in that `vector`.

For simulating heavy ion collisions the class `CgcInitialCondition` is used. It implements the initial conditions derived in section 3.8. The user has to specify the initial charge density ρ_x, the velocity v of the nucleus (which can only be plus or minus the speed of light), the infrared regulator m and the momentum cutoff Λ. Upon calling the `apply()` function, the Poisson equation (3.92) is solved for every transverse plane of the given charge density. Either the FFTW library [37] or the Intel® MKL [38] is used for the Fourier transformations required for this. The obtained $\varphi_a(x) = f(t - x^1)\hat{\varphi}_a(x_T)$ is then used to calculate the gauge transformation V_x via eq. (3.100). However, the code already performs a path ordered exponential, which would be necessary for possible future charge densities with incoherent longitudinal structure. Calculating the electric field and gauge link in the longitudinal direction is simple because $\mathcal{A}^1_x = 0$ and therefore $E^{a,1}_x = 0$ and $U_{x,1} = \mathbb{1}$. Instead of also exponentiating the other gauge fields, it is easier to use the transformation law for the links (3.21) and the fact that the transverse gauge fields in Lorenz gauge vanish, yielding

$$U_{x,i} = V_x V^\dagger_{x+i}, \quad i = 2, 3. \tag{4.10}$$

Another trick is used to calculate the transverse electric field. First of all, $\varphi_a(x + a_0)$ is calculated by shifting $\varphi_a(x)$ one time step in the longitudinal direction. With this, the gauge transformation V_{x+0} as well as the gauge links $U_{x+0,i=2,3}$, which are one step in the future, can be calculated. The electric field is then obtained by rearranging the equations of motion for the links (3.58),

$$E^{a,i}_x t_a = \frac{\mathrm{i}}{g a_0 a_i} \ln\left(U_{x+0,i} U^\dagger_{x,i}\right), \quad i = 2, 3. \tag{4.11}$$

This ensures that the E-field matches the gauge links in the best numerically possible way. In a similar manner, the charge density is calculated from the E-field with the help of the Gauss constraint eq. (3.52). `SuperParticle`s are then spawned with their charge equal to

$$Q^a_x = \frac{\rho^a_x}{N_p}, \tag{4.12}$$

where N_p is the number of particles per cell and both Q^a_x and ρ^a_x are dimensionless.

4.3. Diagnostics

The diagnostics are conceptually designed similar to the initial conditions. `Diagnostics` is the abstract base class all diagnostic modules derive from. Each of them needs to implement the `execute()` function, which calculates the observable the class was made for, as well as the `createDataFileVariables()` and `writeToDataFile()` functions which are used for storing the observables in data files. The diagnostics chosen for a particular simulation must be added to the corresponding `vector` of the `Settings` class. Calling `executeDiagnostics()` and `writeDiagnosticsToDataFile()` then calculates all observables and writes them to the given files respectively. In general, the diagnostic classes compute data at every grid point in physical units. The classes with the prefix `Projected` additionally integrate the observables over transverse planes. Table 4.1 lists all available diagnostic modules and the formulas used to calculate the observables. Most of them are very simple and straightforward, but some of them need a little more explanation. First of all, the `EnergyDensity` and `PoyntingVector` classes are special, because they allow to choose between two ways of calculating their respective observable. The choice is made by correctly setting a constant member variable. Another thing to note about `EnergyDensity` is that it calculates and stores the electric and magnetic as well as the longitudinal and transverse parts separately. The reason why `ProjectedDivS` only calculates the longitudinal derivative is because the periodic boundary conditions in the transverse directions cause the other terms to vanish. In general, diagnostics also try to reuse already calculated results. This means e.g. that if possible `ProjectedEnergyDensity` only sums up the values already computed by `EnergyDensity` instead of calculating the energy again. Similar optimizations hold for `GaussConstraint` and every `Projected` diagnostic except for `ProjectedJInE`. The latter does not only store the dot product of the current density and the electric field integrated over transverse planes, but also integrates it over the whole simulation volume and additionally over time.

4.4. Tests

The tests in this section are to find logical errors in the program code and not for benchmarking. They are only described briefly, because they were already implemented before, and only little additions were made. For more details see [13].

- *GridTest.cpp* tests basic grid functions, like shifting indexes, mapping a position index to a vector, and so on.
- *Su2Test.cpp* tests all operations on `AlgebraElements` and `GroupElements`, like addition, multiplication, exp, ln, etc. It uses the Eigen 3 library for reference calculations.
- *GaussTest.cpp* tests the conservation of the Gauss constraint. For this, a small simulation is performed and the Gauss constraint at the beginning and the end is compared. A new test case with particles was added.
- *EnergyTest.cpp* tests the conservation of total energy, similar to the test of the Gauss constraint, by comparing initial and final energy of a small simulation. A new test case with particles was added.

`ChargeDensity`	$2\,\mathrm{Tr}(\rho_x)^2$
`ElectricField`	$2\,\mathrm{Tr}(\mathcal{E}^i_x)^2$
`OldCurrentDensity`	$2\,\mathrm{Tr}(j_{x-0/2})^2$
`OldGaugeLinks`	$2\,\mathrm{Tr}\left(2\,\mathrm{Im}\,\mathrm{Tr}(t^a U_{x-0/2,i})t_a\right)^2$
`OldPlaquettes`	$2\,\mathrm{Tr}\left(2\,\mathrm{Im}\,\mathrm{Tr}(t^a U_{x-0/2,ij})t_a\right)^2$
`DivE`	$\sum_{i=1}^3 (E^{a,i}_x - \widetilde{E}^{a,i}_{x-i})/a_i$
`EnergyDensity`	$\mathcal{H}_x = \begin{cases} \frac{1}{2}\sum_{i=1}^3 \left(E^{a,i}_x E^i_{x,a} + B^{a,i}_x B^i_{x,a}\right) \\ \frac{1}{2}\sum_{i=1}^3 \left(E^{a,i}_x E^i_{x,a} + \frac{1}{g^2}\sum_{j\neq i}\frac{1}{a_i^2 a_j^2}\,\mathrm{Re}\,\mathrm{Tr}(\mathbb{1} - U_{x,ij})\right) \end{cases}$
`GaussConstraint`	$\sum_{i=1}^3 (E^{a,i}_x - \widetilde{E}^{a,i}_{x-i})/a_i - \rho^a_x$
`PoyntingVector`	$S_{x,i} = \begin{cases} \frac{2}{g}\sum_{j\neq i}\frac{1}{a_i a_j}\,\mathrm{Im}\,\mathrm{Tr}\,E^{a,j}_x t_a U_{x,j-i}) \\ S_{x,i} = \epsilon_{ijk}\mathcal{E}^j_x \mathcal{B}^k_x \end{cases}$
`ProjectedDivS`	$\sum_{x_T} a_2 a_3 (S_{x+1} - S_x)/a_1$
`ProjectedEnergyDensity`	$\sum_{x_T} a_2 a_3 \mathcal{H}_x$
`ProjectedGaussConstraint`	$\sum_{x_T} a_2 a_3 \left(\sum_{i=1}^3 (E^{a,i}_x - \widetilde{E}^{a,i}_{x-i})/a_i - \rho^a_x\right)$
`ProjectedJInE`	$\sum_{x_T} a_2 a_3 \left(2\,\mathrm{Tr}\sum_{i=1}^3 j^i_x \mathcal{E}^i_x\right)$
`ProjectedPowerDensity`	$\sum_{x_T} a_2 a_3 (\mathcal{H}_x - \mathcal{H}_{x-0})/a_0$
`ProjectedPoyntingVector`	$\sum_{x_T} a_2 a_3 S_{x,i}$

Table 4.1.: List of all available diagnostics and the observables they calculate. Integration over transverse planes is denoted by $\sum_{x_T} a_2 a_3$. The frequently appearing factor of 2 is to cancel the 1/2 in eq. (3.5). The inner expression for `OldGaugeLinks` and `OldPlaquettes` calculates the linearized algebra element of $U_{x-0/2,i}$ and $U_{x-0/2,ij}$ respectively, which is then squared the same way as the other algebra elements. Some diagnostics have the possibility to choose between two formulas for calculating their observable. For the definition of the physical quantities used in the equations above, see sections 3.1, 3.4, 3.6 and 3.7.

- *PoyntingTest.cpp* tests the conservation of Poynting's theorem eq. (3.75). Again, this is done by comparing the violation of the theorem before and after a small simulation. A new test case with particles was added.

4.5. Parallelization

The already implemented shared-memory parallelization with OpenMP was extended and now covers all computationally expensive parts, namely the equation of motion solver, the particle pusher and the diagnostic modules. Only the initialization of the simulation and writing to the data file is not parallelized. For the latter one, this is, because MPI is needed for parallel access to NetCDF files. The main advantage of parallelizing code with OpenMP is, that it is easy to apply and requires minimal changes to the code. Most of the time it is sufficient to add a few `#pragma omp parallel` and `#pragma omp for` directives, which spread the work of the following `for`-loop over all available threads. A disadvantage is that OpenMP only allows shared-memory parallelization. This means that the work can only be distributed between CPU cores that can access the same main memory. In a modern multi-core CPU, the number of threads is therefore equal to the number of physical cores (twice the number if Intel®'s Hyper-Threading is used). Computer clusters like the Vienna Scientific Cluster 3 (VSC-3) contain many CPUs and usually only those on the same computation node can access the same memory via NUMA (non-uniform memory access) [39]. For the VSC-3, where most of the simulations of this thesis were performed, a maximum number of 16(32) threads can be used. This is because each node has two 8-core Intel® CPUs [40].

The Message Passing Interface (MPI) allows to lift the restriction of shared-memory parallelization and makes it possible to distribute work between CPUs on different computation nodes. The main disadvantage is that this distributed-memory parallelization works on a very low level, namely – as the name suggests – the passing of messages between CPUs. The data these messages contain, when and how they are sent and received, all of that has to be determined by the programmer. This results in a big implementation overhead. Therefore it is simpler to use parallelization with MPI at a very high level. As explained in section 2.3, to get meaningful physical data, the results of many simulations need to be averaged, because the initial conditions are based on random charge configurations. The loop over these simulations is parallelized with MPI. For example, if the user wants to run 16 simulations, he or she can divide them among as many as 16 computation nodes.

So far, there is no real need to run a single simulation on multiple nodes, because the 256 GB of memory each node holds are enough to handle the grid sizes used by now. However, larger grids may be interesting in the future, which would require parallelization with MPI on the level of a single Simulation.

5. Test Results

The results presented in this chapter are for validating and benchmarking the C++ code in comparison with the original Java version. All simulations were run on the VSC-3, with a coupling constant of $g = 2$ and a momentum cutoff of $\Lambda = 10\,\mathrm{GeV}$. The rest of the settings are detailed in the specific sections. Thread pinning was also used for all tests, meaning that each thread is pinned to a single physical core. If the threads had been allowed to move to other cores, the cache would not have been moved with them, resulting in more cache misses, leading to longer memory access time and therefore reduced performance. This effect has already been studied in [13, section 4.3]

5.1. Pressure Anisotropy

A well known feature of the early glasma is its pressure anisotropy. The longitudinal and transverse pressure $\langle p_L \rangle$ and $\langle p_T \rangle$ are calculated in the following way:

$$\langle p_T \rangle = \langle \epsilon_L \rangle, \tag{5.1}$$

$$\langle p_L \rangle = \langle \epsilon_T \rangle - \langle \epsilon_L \rangle, \tag{5.2}$$

where $\langle \epsilon_L \rangle$ and $\langle \epsilon_T \rangle$ are the longitudinal and transverse energy densities respectively. Right after the collision so called color flux tubes are created and their longitudinal fields give rise to a large transverse and negative longitudinal pressure. Observations show that within a few fm/c the system then transitions into a quark–gluon plasma, with roughly equal pressure in all directions [41, 42]. A more detailed physical discussion as well as the reference results for the simulations in this section can be found in [8].

For the simulations, a grid size of $N_L \times N_T^2 = 380 \times 256^2$ cells, $N_t = 380$ time steps, a lattice spacing of $a_i = 0.04\,\mathrm{fm}$ in all directions and a time step of $a_t = a_z/2 = 0.02\,\mathrm{fm/c}$ was chosen. The infrared regulator was set to $m = 2\,\mathrm{GeV}$ and the MV parameter to $\mu = 0.5\,\mathrm{GeV}$. The longitudinal width σ, was set to $4a_z$, $6a_z$ and $8a_z$. For each of the three widths, the results of 32 simulations were averaged. Figure 5.1 shows the projected, longitudinal and transverse pressures at different times with $\sigma = 4a_z$, which corresponds to a gamma factor of $\gamma \approx 23$. The plots are in good agreement with [8, Figure 6]. They show the same flat shape of the arising transverse pressure, as well as the exponential decrease of the longitudinal pressure between the nuclei. The reason for the right plot to be at $t = 4.6\,\mathrm{fm/c}$ instead of $t = 5\,\mathrm{fm/c}$ – as in the reference result – is, that the nuclei

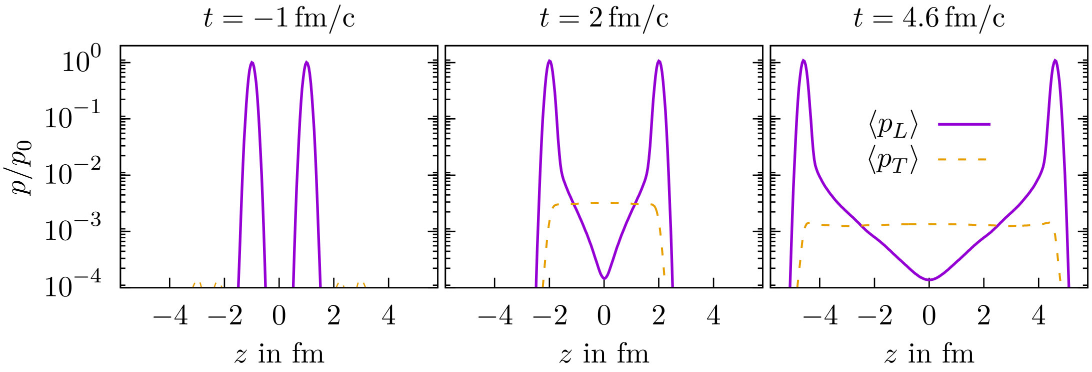


Figure 5.1.: The longitudinal and transverse pressure, $\langle p_L \rangle$ and $\langle p_T \rangle$, are plotted over the longitudinal axis z at different times. The pressures are scaled relative to the maximum, longitudinal pressure of the initial nuclei $\langle p_0 \rangle$.

are initially further apart in the C++ code. Therefore, the same number of time steps leads to a lower maximum time after the collision.

Additional simulations were run with smaller lattice spacings $a_i = 0.004\,\mathrm{fm}$ and time step $a_t = a_z/2 = 0.002\,\mathrm{fm/c}$. The rest of the settings stayed the same. Figure 5.2 shows the time evolution of the pressure components at the center point of the collision, $z = 0$. The left figure plots the results of the coarse grid, the right one those of the fine grid. At the very beginning, the longitudinal pressure of the nuclei dominates. However, as soon as they move away far enough, the transverse pressure rises to $\langle p_T \rangle / \langle \epsilon \rangle \approx 1/2$, where $\langle \epsilon \rangle$ is the energy density, and the longitudinal pressure vanishes. The latter implies that the transverse energy density equals the longitudinal one. The results of the fine grid even show negative longitudinal pressure at the beginning. Both these phenomena are also observed in boost-invariant simulations [43]. There however, one starts right away with $\langle p_T \rangle = \langle \epsilon_L \rangle = -\langle p_L \rangle$, because the nuclei are infinitesimally thin and their fields are not taken into account. Contrary to the boost-invariant case, a steady, albeit too slow, trend towards the isotropic values of the pressure components, $\langle p_T \rangle / \langle \epsilon \rangle = \langle p_L \rangle / \langle \epsilon \rangle \approx 1/3$, can be seen. When comparing fig. 5.2 with [8, Figure 8], again one finds a good agreement of the plots. However, one has to be careful with the comparison, because for convenience the C++ version used a thickness of exactly one tenth of the coarse results while one of the Java simulations used even thinner nuclei, resulting in a greater negative pressure. In particular the range of thicknesses was $\sigma \in \{0.016\,\mathrm{fm}, 0.024\,\mathrm{fm}, 0.032\,\mathrm{fm}\}$ for the C++ code and $\sigma \in \{0.008\,\mathrm{fm}, 0.016\,\mathrm{fm}, 0.032\,\mathrm{fm}\}$ for the Java simulations.

5.2. Rapidity Profiles

The rapidity profiles discussed in this section are plots of the local rest frame energy density $\langle \epsilon_{\mathrm{loc}} \rangle$ over the rapidity η at constant proper time τ. The latter two are defined

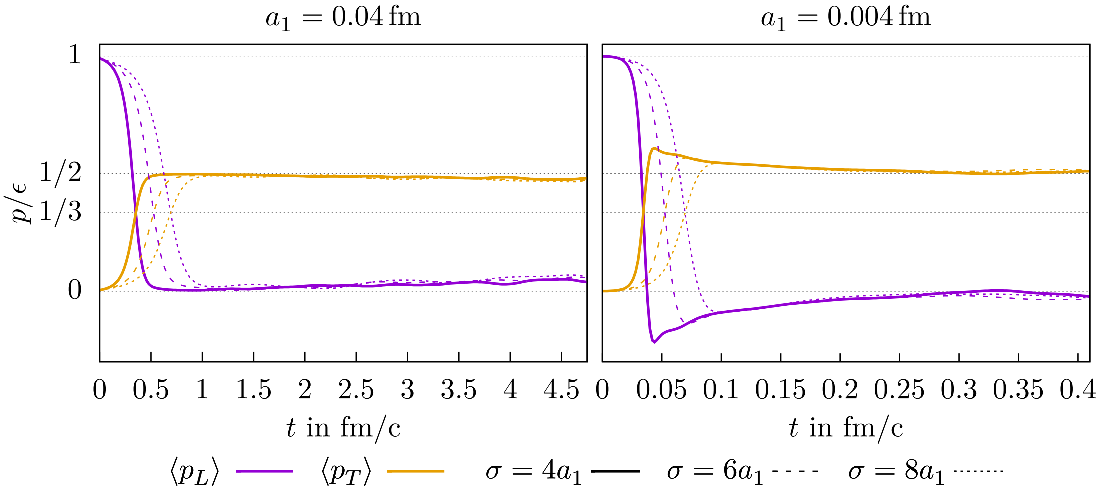


Figure 5.2.: The longitudinal and transverse pressure $\langle p_L \rangle$ and $\langle p_T \rangle$ are plotted over time at the center point of the collision $z = 0$. The pressures are scaled relative to the energy density $\langle \epsilon \rangle$.

in the following way:

$$\tau(t, z) = \sqrt{t^2 - z^2}, \tag{5.3}$$

$$\eta(t, z) = \frac{1}{2} \ln \frac{t - z}{t + z}, \tag{5.4}$$

with the coordinate origin chosen to be at the point of maximum transverse pressure. Since there shall be no energy flow in the local rest frame, $\langle \epsilon_{\mathrm{loc}} \rangle$ can be calculated by diagonalizing the energy-momentum tensor $\langle T^{\mu}{}_{\nu} \rangle$. In the laboratory frame of the simulations, it can be written as

$$\langle T^{\mu}{}_{\nu} \rangle = \begin{pmatrix} \langle \epsilon \rangle & 0 & 0 & -\langle S_L \rangle \\ 0 & -\langle p_T \rangle & 0 & 0 \\ 0 & 0 & -\langle p_T \rangle & 0 \\ \langle S_L \rangle & 0 & 0 & -\langle p_L \rangle \end{pmatrix}, \tag{5.5}$$

with $\langle \epsilon \rangle$ being the energy density and $\langle S_L \rangle$ the longitudinal component of the Poynting vector. When solving the eigenvalue problem, the local rest frame energy density is the eigenvalue corresponding to the timelike eigenvector and reads

$$\langle \epsilon_{\mathrm{loc}} \rangle = \frac{1}{2} \left(\langle \epsilon \rangle - \langle p_L \rangle + \sqrt{\left(\langle \epsilon \rangle + \langle p_L \rangle \right)^2 - 4 \langle S_L \rangle^2} \right). \tag{5.6}$$

It is also the largest eigenvalue. The square root in eq. (5.6) turned out to be very sensitive to small discretization errors, because the terms underneath have to cancel each other for most of the simulation volume. If this cancellation is off by just a small

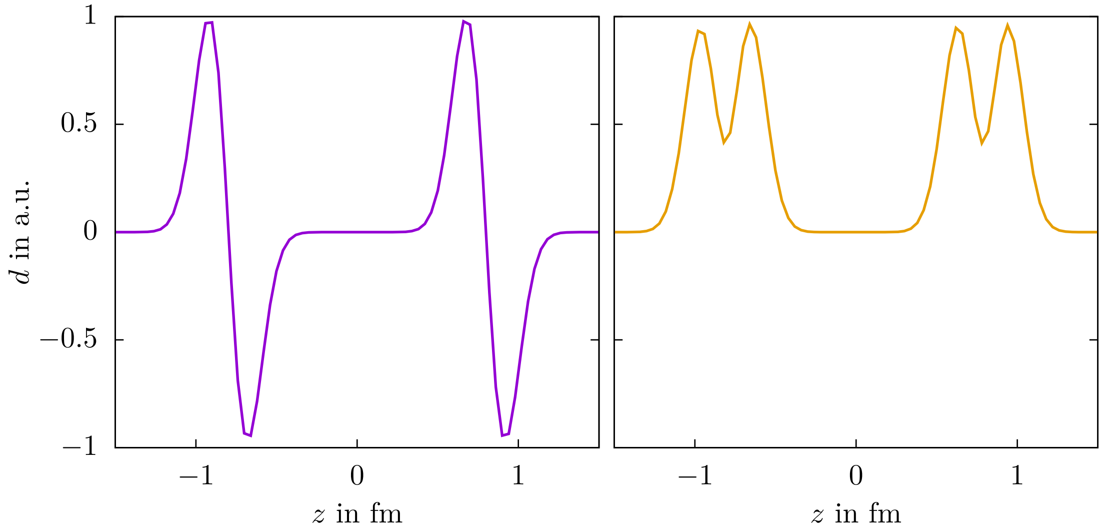


Figure 5.3.: Plots of the discriminant $d = (\epsilon + p_L)^2 - 4{S_L}^2$ of the local rest frame energy density (eq. (5.6)) along the longitudinal axis shortly before the collision for a single event. The left panel calculates the Poynting vector according to eq. (3.82), while the right panel uses $S_{x,i} = \epsilon_{ijk}\mathcal{E}^j_x\mathcal{B}^k_x$.

amount, it can lead to negative values under the root and to imaginary and therefore unphysical local energy densities. Figure 5.3 shows the value under the square root some time before the collision for a single event. The left panel uses eq. (3.82) and clearly shows negative values, while the plot on the right side uses a naive discretization for the Poynting vector and is well behaved along the whole longitudinal axis. This is the reason why in this section the Poynting vector is not calculated with the equation derived from Poynting's theorem, but with $S_{x,i} = \epsilon_{ijk}\mathcal{E}^j_x\mathcal{B}^k_x$.

The simulations are expected to show rapidity profiles with a Gaussian shape, as this has been found in experiments at LHC [44] and RHIC [45, 46], as well as in holographic calculations of colliding shock waves [47–49]. An explanation for this shape is given by the Landau model [50]. For a more detailed physical discussion as well as the reference results for this section, see [9].

In the longitudinal direction, a very fine grid is necessary to get smooth interpolations when changing the coordinates from (t, z) to (τ, η). Therefore, the grid size was set to $N_L \times N_T^2 = 2048 \times 192^2$ cells, the number of time steps to $N_t = 2048$, the lattice spacings to $a_x = a_y = 0.031\,25\,\mathrm{fm}$, $a_z = 0.002\,93\,\mathrm{fm}$ and the time step to $a_t = a_z/2 = 0.001\,465\,\mathrm{fm/c}$. This results in a physical simulation volume of $(6\,\mathrm{fm})^3$. The infrared regulator is set to $m = 0.2\,\mathrm{GeV}$, the MV parameter to $\mu = 0.6464\,\mathrm{GeV}$ and the thickness parameter to $\sigma = 12.459 a_z = 0.0365\,\mathrm{fm}$, which corresponds to a center-of-mass energy per nucleon pair of $\sqrt{s_{\mathrm{NN}}} = 200\,\mathrm{GeV}$. All observables are integrated over transverse planes and the results are averaged over 16 simulations. The left panel of fig. 5.4 shows

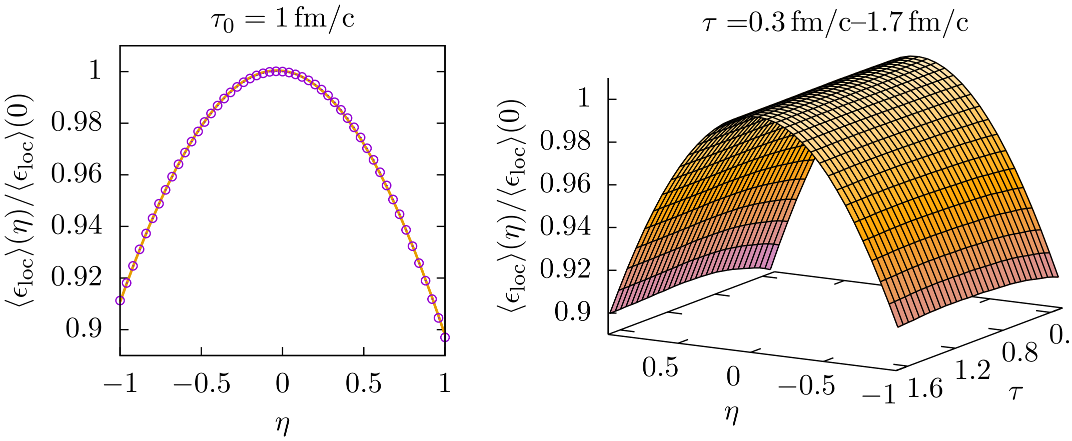


Figure 5.4.: Rapidity profiles, i.e. plots of the local rest frame energy density $\langle\epsilon_{\mathrm{loc}}\rangle(\tau,\eta)$ over the rapidity η, at collision energies of $\sqrt{s_{\mathrm{NN}}} = 200\,\mathrm{GeV}$. The left figure includes a Gaussian fit according to eq. (5.7).

the obtained rapidity profile at $\tau_0 = 1\,\mathrm{fm/c}$ in the range $\eta \in [-1, 1]$. As expected, the profile has a Gaussian shape. A fit of the form

$$\langle\epsilon_{\mathrm{loc}}\rangle(\tau_0,\eta) \approx \langle\epsilon_{\mathrm{loc}}\rangle(\tau_0,0)\exp\left(-\frac{(\eta-\mu_\eta)^2}{2\sigma_\eta^2}\right), \tag{5.7}$$

is also plotted in the figure and yields a width of $\sigma_\eta = 2.23$. The results agree reasonably well with the ones presented in [9, Figure 2], which are $\sigma_\eta = 2.34$ for the Java Simulation and $\sigma_\eta = 2.25$ for experimental data from RHIC [45]. The reason why the profile is not symmetric, but has its center shifted to the left, is because of the random color charges used for the initial collisions. They lead to different energy densities of the two colliding nuclei and even after averaging over 16 simulations, there is still some asymmetry left. In the right panel of fig. 5.4 one can see that the Gaussian shape of the rapidity profile is already formed at $\tau = 0.3\,\mathrm{fm/c}$ and stays roughly the same afterwards. This behavior has also been observed in the Java simulations.

5.3. Transverse Pressure

The transverse pressure $\langle p_T\rangle$ is interesting to study, because it is equal to the energy density of the longitudinal fields which are characteristic for the glasma. It is zero before the collision and only builds up afterwards. Figure 5.5 plots the longitudinal energy density in the transverse plane at the center point of the collision, at the maximum overlap of the nuclei. For these simulations the lattice spacing was set to $a_i = 0.028\,\mathrm{fm}$, the time step to $a_0 = a_i/2$, the IR regulator to $m = 2\,\mathrm{GeV}$ and the MV model parameter to $\mu = 0.5\,\mathrm{GeV}$. For the left plot, a thickness parameter of $\sigma = 4a_1$ and a longitudinal

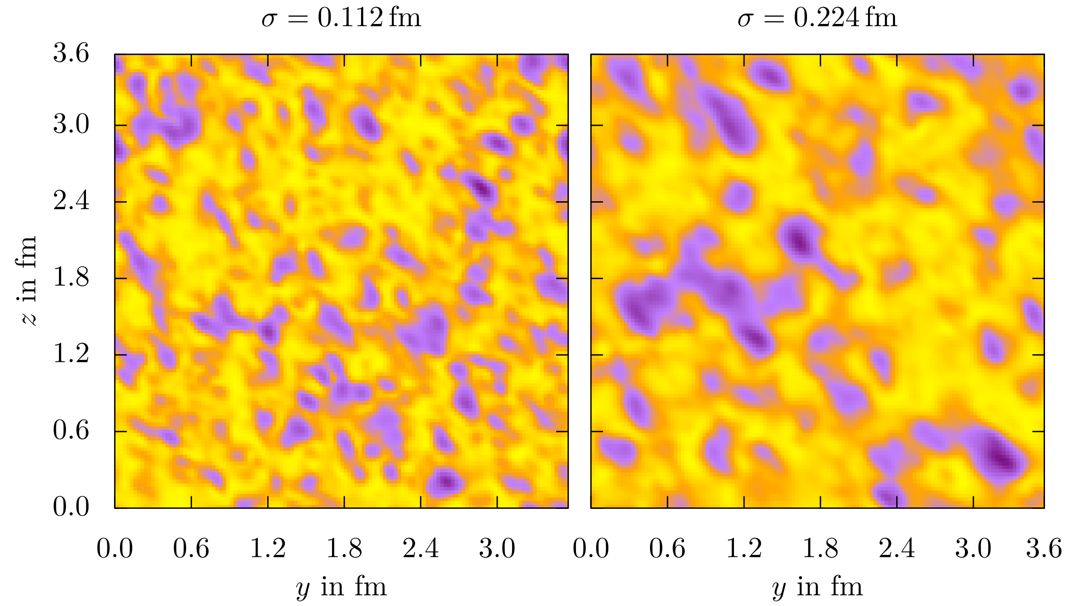


Figure 5.5.: Plots of the longitudinal energy density $\langle \epsilon_L \rangle$ in the transverse plane at the center point of the collision $z = 0$ at the time of maximum overlap. Two different values for the thickness parameter were used.

number of cell of $N_L = 128$ were used, while for the right plot the amount of longitudinal cells as well as the thickness parameter were doubled. In the transverse direction the grid size was set to $N_T^2 = 128^2$. One can see that longitudinal fields appear randomly throughout the whole transverse plane and that thicker nuclei causes the random structure to be more washed out. These observations agree with those of similar Java simulations (see [8, Figure 4]).

Figure 5.6 shows a plot of the transverse pressure in and around the forward light cone. For this simulation, the same settings as in section 5.2 are used. In contrast to the boost-invariant case, which would yield constant fields along the light cone, a rather steep drop of $\langle p_T \rangle$ is observed. This – and the plot as a whole – agrees well with the Java simulations (compare with [9, Figure 4]).

5.4. Execution Time and Memory Usage

A benchmark test, measuring start-up time, execution time of the main loop, as well as memory usage, was conducted to compare the performance of the Java and C++ code. Another member of the institute, David Müller, implemented a very similar simulator in Cython – a C-extension for Python – which was also included in the comparison. No diagnostic modules were active and no data file has been created in the tests. The lattice

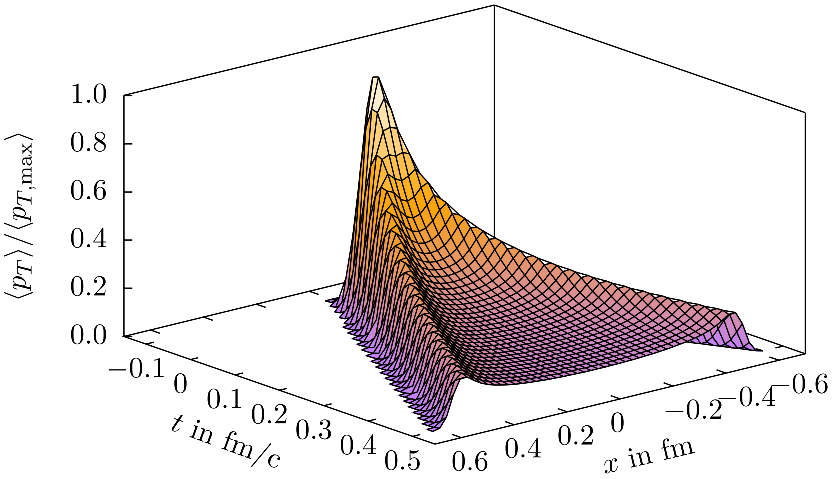


Figure 5.6.: Transverse pressure $\langle p_T\rangle$ of a collision with $\sqrt{s_{\mathrm{NN}}} = 200\,\mathrm{GeV}$, normalized to its maximum and plotted in and around the forward light cone.

		Java			C++			Cython		
size	n	t_{start} in s	t_{exe} in ms	memory in MiB	t_{start} in s	t_{exe} in ms	memory in MiB	t_{start} in s	t_{exe} in ms	memory in MiB
64^4	1	15	547	3490	0.5	153	158	3.0	134	133
64^4	8	15	109	3490	0.5	23	158	0.7	23	133
128^4	8	120	1070	8530	4.1	196	1166	5.6	172	1024
256^4	8	1646	9740	56890	31.7	1510	10240	45.7	1360	8325

Table 5.1.: Results of a benchmark, comparing the start-up time t_{start}, the execution time per time step t_{exe} and the memory usage of Java, C++ and Cython simulations. The problem size is given as the number of grid cells times the number of time steps and n is the number of threads.

spacings were set to $a_i = 1\,\mathrm{fm}\ \forall i$, the time step to $a_t = a_i/2$, the IR regulator to $m = 0.2\,\mathrm{GeV}$ and the MV parameter to $\mu = 0.6464\,\mathrm{GeV}$. Three different cubic grid sizes with an equal number of time steps were used, i.e. $N_L \times N_T^2 \times N_t \in \{64^4, 128^4, 256^4\}$. For each of these girds, a test with eight threads was performed. An additional, singlethreaded one was run on the smallest grid. The thickness parameter was set to $\sigma = N_L/32$. Since the width of a single initial charge density used in `CoherentMvModel` is 16σ (see section 4.2), the whole simulation volume was filled with particles.

Table 5.1 shows the results of the benchmark, which were averaged over four runs. They were all performed on the VSC-3 on nodes with an Intel Xeon IvyBridge-EP E5-2650v2 with 2.6 GHz and 256 GB of main memory [39, 40]. The execution time per time step and memory usage of the multithreaded tests are illustrated in fig. 5.7. It is easy to see that the C++ and Cython code are faster, use less memory and better utilize multiple

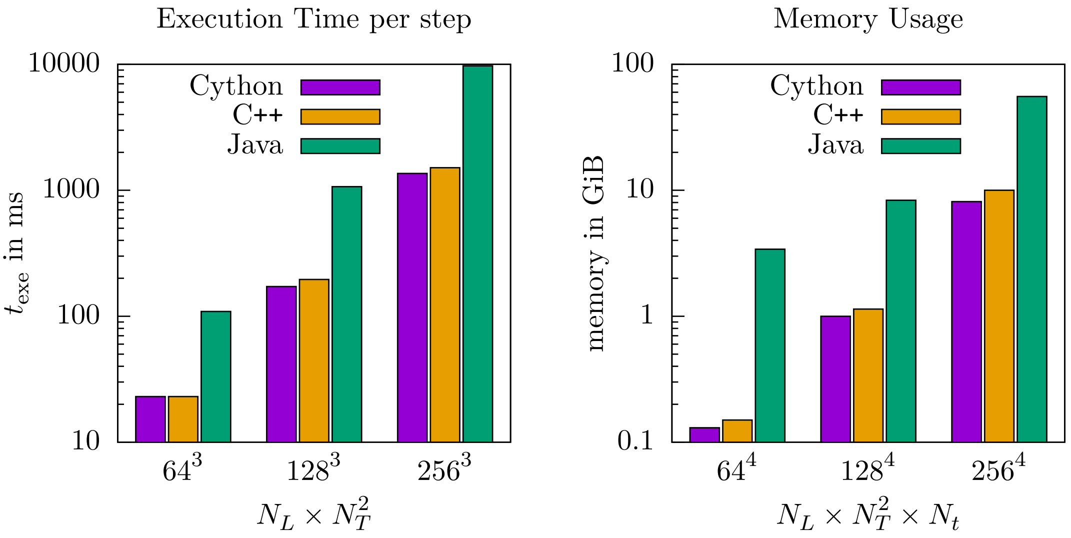


Figure 5.7.: Comparison of execution time per time step and memory usage of the Java, C++ and Cython code for different grid sizes. Data is taken form table 5.1.

threads than the Java version. The latter has an uneven and hard to predict scaling of all the measured quantities. Particularly the memory consumption on small grids is considerably larger than expected. The C++ and Cython code on the other hand show almost perfect scaling of start-up time, execution time and memory usage. For reasonably large grid sizes – 256^3 cells with 256 time steps – and when looking at the total simulation time the C++ code is almost 10 times faster than the Java version, while using less than a fifth of the memory. When comparing the C++ and Cython versions, table 5.1 shows that the latter is up to 13 % faster and uses up to 20 % less memory. The lower memory consumption is simply explained by the fact that the Cython version does not store plaquettes. The difference in execution speed however was unexpected and is harder to explain, since the C++ version was faster for simulations not involving particles, as shown in table 5.2 and fig. 5.8. It was concluded that the particle pusher was so much slower that the Cython code won the contest of overall execution speed. Further investigations traced the poor performance of the C++ code back to two reasons. The first and most important one is a different ordering of the update functions in the simulation cycle. The C++ version calculates the electric field at time t and the gauge links at $t + a_0/2$. Then the particles are updated, the charges parallel transported and the charge density at time t is interpolated. Finally the new current density at time $t + a_0/2$ is computed. The problem with this approach lies in eq. (3.63). In order to calculate the new current density of the left moving particles at $t + a_0/2$, the charges at the future time step $t + a_0$ are necessary. For this purpose, the charges are temporarily parallel transported. The important thing to note here is that the parallel transport of the charges is by far the most computationally expensive part of the particle pusher as

	Java	C++	Cython
grid size	t_{exe} in ms	t_{exe} in ms	t_{exe} in ms
64^3	217	94	122
128^3	2421	734	970
256^3	26270	5622	7820

Table 5.2.: Results of a singlethreaded benchmark comparing execution time per time step t_{exe} for a simulation without using particles. The lattice spacings were set to $a_i = 1\,\mathrm{fm}$, the time step to $a_0 = a_i/2$ and the coupling to $g = 1$. The number of time steps was $N_t = 1000$ for the first two grid sizes and $N_t = 100$ for the largest one.

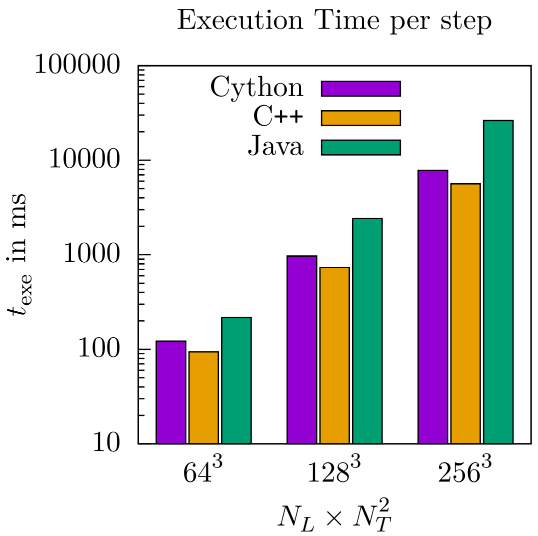


Figure 5.8.: Comparison of execution time per time step of the Java, C++ and Cython code for different grid sizes and without particles. Only a single thread was used. Data is taken from table 5.2.

#threads (#CPUs)	t_{start} in s	t_{exe} in s	s	p in %
1(1)	2.0	207.0	1.0	–
2(1)	1.9	108.0	1.9	95
4(1)	1.8	57.7	3.6	96
8(1)	1.8	31.7	6.5	97
16(1)*	1.8	25.1	8.2	94
16(2)	1.8	18.2	11.4	97
32(2)*	1.8	20.9	9.9	93

Table 5.3.: Results of the parallelization benchmark, measuring start-up time t_{start}, execution time of the main loop t_{exe}, speedup factor s, as well as level of parallelization p. In the rows marked with star *, Intel®'s Hyper-Threading was used.

it involves two group multiplications. Consequently, the additional parallel transport of the left moving particles increases the computational effort by almost 50 percent. The Cython version circumvents this problem by executing the particle pusher before the solver for the fields. As a result, the charge density is still calculated at time t, but the new current density has to be computed at time $t - a_0/2$. Therefore, charges at a future time step $t + a_0$ are no longer needed. By adopting the order of calculation regarding the charge and current density depending on the direction of movement of the particles, no additional parallel transports are necessary. The second reason for the better execution speed of the Cython code is that by combining the charge and current update functions in a single loop, a more efficient memory access pattern is achieved, which potentially increases the performance even further.

5.5. OpenMP

To find out how well the C++ code is parallelized with OpenMP, a set of tests with different numbers of threads was performed. The grid size times number of time steps was set to $N_L \times N_T^2 \times N_t = 128^4$, the lattice spacings to $a_i = 1\,\text{fm}\ \forall i$, the time step to $a_0 = a_i/2$, the thickness parameter to $\sigma = 4a_i$, the MV parameter to $\mu = 0.6464\,\text{GeV}$ and the IR regulator to $m = 0.2\,\text{GeV}$. The most common diagnostic modules were used, namely `ProjectedEnergyDensity`, `ProjectedGaussConstraint` and `ProjectedJInE`. All their diagnostic values were stored in a data file. The results, shown in table 5.3, were averaged over four runs. One of the first things to note is that the start-up time t_{start} stays almost constant. The reason for this is that only the calculation of the initial energy and Gauss constraint, which get printed on the screen, is parallelized. Setting up the simulation and applying the initial conditions is all done in serial. When looking at

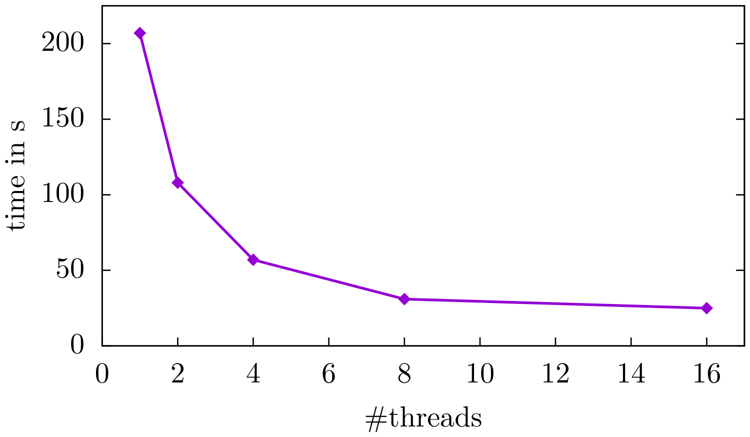


Figure 5.9.: Single CPU results of OpenMP test. The execution time t_{exe} of the main loop is plotted over the number of threads. Data is taken from table 5.3.

the speedup factor s in table 5.3, given by

$$s(n) = \frac{t_{\text{exe}}(1 \text{ thread})}{t_{\text{exe}}(n \text{ threads})}, \tag{5.8}$$

one can see that it grows much slower than the number of threads. This is because every parallelized program also contains serial code, which does not execute faster with more threads, and therefore sets a lower limit for the execution time and an upper limit for the speedup factor. Figure 5.9 shows how the execution time approaches this limit with an increasing number of threads. To better understand how well the code is parallelized, the so called level of parallelization p is introduced. Let a quantify the serial part of the code and b the amount of perfectly parallel code. The level of parallelization is then defined as

$$p := \frac{b}{a+b}, \tag{5.9}$$

which is the fraction of code that is perfectly parallelized. Given the speedup factor s and the number of threads n, one can also calculate p in the following way:

$$p = \frac{n(s-1)}{s(n-1)}. \tag{5.10}$$

As expected, the level of parallelization in table 5.3 stays roughly the same for all tests. Only those with Hyper-Threading have lower values. The reason for that is the lower performance of the virtual threads used with this technology [51]. Surprisingly, the best performance is achieved when using both CPUs of the computation node without Hyper-threading. This is unexpected, because the additional virtual threads increase performance on a single CPU. A possible reason for this behavior is that the so called first touch policy is not implemented in the present code. Consequently, all the variables and arrays are allocated in the main memory of a single CPU. Although the other CPU on the node can access this memory, the connection that allows this can become a bottleneck as this is not as fast as accessing the CPU's own memory. Therefore, a future improvement would be to implement an allocator that puts the elements of arrays and vectors closest to the thread that uses them.

6. Conclusion

The color glass condensate is a classical and effective theory, describing the early stages of heavy ion collisions, which allows the study of quantum chromodynamics in the dense, high energy regime. A formulation of this theory on a cuboid lattice that is gauge invariant even in the discrete case was presented. Based on the colored particle in cell method, which is utilized to describe the point-like particles representing the hard partons, all aspects of the CGC model are expressed on the grid. The derivation of the equations of motion was done in the temporal gauge and they were formulated in way allowing them to be used in an explicit numerical solver. For the Poynting vector, an alternative discretization was derived that better preserves the Poynting theorem. However, when calculating the local rest frame energy density, the naive discretization of the continuum Poynting vector must be used since imaginary energy densities arise due to discretization errors when using the alternative form.

Regarding the simulator itself, the available code was expanded by color charged particles and CGC initial conditions using the McLerran-Venugopalan model. In particular, the equation of motion solver as well as many diagnostic modules were adopted, rewritten or newly added. With this, all necessary parts of OpenPixi have been implemented in C++. The comparison tests and benchmarks show that the new simulator gives valid results in comparison with the original Java version. It generates prominent features such as pressure anisotropy and Gaussian rapidity profiles. The transverse pressure in the forward light cone also agrees very well with the results of the Java code. Furthermore, using thinner nuclei creates results more similar to boost-invariant simulations, which was expected since the boost-invariant case corresponds to infinitely thin nuclei. Because of their well behaved scaling, the execution time and memory usage can be predicted more reliably for the C++ code. On top of that it was proven, that the C++ simulator is about an order of magnitude faster than the Java version, while using only a fraction of the memory. However, a Cython code also implementing a CGC simulator, showed even higher performance. Thanks to a detailed analysis, the reasons for the difference in execution speed and memory usage were found. It turned out that only little changes are needed to make the C++ code at least as performant as the Cython code. The final tests show that the parallelization with OpenMP is quite effective, with an achieved level of parallelization of up to 97 %. In summary, porting OpenPixi from Java to C++ was a worthwhile effort, although there are still a few details that need some improvement.

The documentation that has been created for the whole code makes it easier to use and expand it in the future. Possible improvements would be to implement a new solver which does not suffer from lattice dispersion in the longitudinal direction or add distributed-memory parallelism to single simulations with the help of MPI to allow larger grid sizes. An incoherent version of the MV model for thick nuclei with nontrivial longitudinal structure would also be interesting.

References

[1] Helmut Satz. “The Quark-Gluon Plasma: A Short Introduction”. In: *Nucl. Phys.* A862-863 (2011), pp. 4–12. DOI: `10.1016/j.nuclphysa.2011.05.014`. arXiv: `1101.3937 [hep-ph]`.

[2] Johann Rafelski. “Connecting QGP-Heavy Ion Physics to the Early Universe”. In: *Nuclear Physics B - Proceedings Supplements* 243-244 (2013). Proceedings of the IV International Conference on Particle and Fundamental Physics in Space, pp. 155–162. ISSN: 0920-5632. DOI: `https://doi.org/10.1016/j.nuclphysbps.2013.09.017`. URL: `http://www.sciencedirect.com/science/article/pii/S0920563213005410`.

[3] Edmond Iancu. “Gluon saturation at small x”. In: *Multiparticle dynamics. Proceedings, 31st International Symposium, ISMD 2001, Datong, China, September 1-7, 2001.* 2001, pp. 184–191. DOI: `10.1142/9789812778048_0029`. arXiv: `hep-ph/0111400 [hep-ph]`. URL: `http://www.slac.stanford.edu/econf/C010901`.

[4] Edmond Iancu, Andrei Leonidov, and Larry McLerran. “The Color glass condensate: An Introduction”. In: *QCD perspectives on hot and dense matter. Proceedings, NATO Advanced Study Institute, Summer School, Cargese, France, August 6-18, 2001.* 2002, pp. 73–145. arXiv: `hep-ph/0202270 [hep-ph]`.

[5] Raju Venugopalan. “From Glasma to Quark Gluon Plasma in heavy ion collisions”. In: *J. Phys.* G35 (2008), p. 104003. DOI: `10.1088/0954-3899/35/10/104003`. arXiv: `0806.1356 [hep-ph]`.

[6] *OpenPixi. Open Particle-in-Cell (PIC) simulator.* URL: `http://www.openpixi.org/` (visited on 10/03/2017).

[7] *OpenPixi source code.* URL: `https://github.com/openpixi` (visited on 10/03/2017).

[8] Daniil Gelfand, Andreas Ipp, and David Müller. “Simulating collisions of thick nuclei in the color glass condensate framework”. In: *Phys. Rev.* D94.1 (2016), p. 014020. DOI: `10.1103/PhysRevD.94.014020`. arXiv: `1605.07184 [hep-ph]`.

[9] Andreas Ipp and David Müller. “Broken boost invariance in the Glasma via finite nuclei thickness”. In: *Phys. Lett.* B771 (2017), pp. 74–79. DOI: `10.1016/j.physletb.2017.05.032`. arXiv: `1703.00017 [hep-ph]`.

[10] Alex Krasnitz and Raju Venugopalan. “Nonperturbative computation of gluon minijet production in nuclear collisions at very high-energies”. In: *Nucl. Phys.* B557 (1999), p. 237. DOI: `10.1016/S0550-3213(99)00366-1`. arXiv: `hep-ph/9809433 [hep-ph]`.

[11] T. Lappi. “Production of gluons in the classical field model for heavy ion collisions”. In: *Phys. Rev.* C67 (2003), p. 054903. DOI: `10.1103/PhysRevC.67.054903`. arXiv: `hep-ph/0303076 [hep-ph]`.

[12] Alex Kovner, Larry D. McLerran, and Heribert Weigert. “Gluon production from nonAbelian Weizsacker-Williams fields in nucleus-nucleus collisions”. In: *Phys. Rev.* D52 (1995), pp. 6231–6237. DOI: `10.1103/PhysRevD.52.6231`. arXiv: `hep-ph/9502289 [hep-ph]`.

[13] Patrick Kappl. “Implementing a Simulation for Classical Yang-Mills Fields in C++”. project thesis. Institute für theoretische Physik, Technische Universität Wien, 2017.

[14] Bjarne Stroustrup. *Lecture: The essence of C++*. University of Edinburgh, May 6, 2014. URL: `https://www.youtube.com/watch?v=86xWVb4XIyE` (visited on 10/03/2017).

[15] F. Gelis. “Color Glass Condensate and Glasma”. In: *Int. J. Mod. Phys.* A28 (2013), p. 1330001. DOI: `10.1142/S0217751X13300019`. arXiv: `1211.3327 [hep-ph]`.

[16] F. Gelis. “The Initial Stages of Heavy Ion Collisions”. In: *Acta Phys. Polon.* B45.12 (2014), pp. 2257–2306. DOI: `10.5506/APhysPolB.45.2257`.

[17] F. Gelis. “Initial state and thermalization in the Color Glass Condensate framework”. In: *Int. J. Mod. Phys.* E24.10 (2015), p. 1530008. DOI: `10.1142/S0218301315300088`. arXiv: `1508.07974 [hep-ph]`.

[18] Aleksi Kurkela et al. “Effective kinetic description of event-by-event pre-equilibrium dynamics in high-energy heavy-ion collisions”. In: (2018). arXiv: `1805.00961 [hep-ph]`.

[19] Aleksi Kurkela et al. “Matching the non-equilibrium initial stage of heavy ion collisions to hydrodynamics with QCD kinetic theory”. In: (2018). arXiv: `1805.01604 [hep-ph]`.

[20] T. Lappi and L. McLerran. “Some features of the glasma”. In: *Nucl. Phys.* A772 (2006), pp. 200–212. DOI: `10.1016/j.nuclphysa.2006.04.001`. arXiv: `hep-ph/0602189 [hep-ph]`.

[21] Abhay Deshpande, Rolf Ent, and Richard Milner. “The EIC’s route to a new frontier in QCD”. In: *CERN Cour.* 49N9 (2009), pp. 13–15.

[22] Larry D. McLerran and Raju Venugopalan. “Computing quark and gluon distribution functions for very large nuclei”. In: *Phys. Rev.* D49 (1994), pp. 2233–2241. DOI: `10.1103/PhysRevD.49.2233`. arXiv: `hep-ph/9309289 [hep-ph]`.

[23] Larry D. McLerran and Raju Venugopalan. “Gluon distribution functions for very large nuclei at small transverse momentum”. In: *Phys. Rev.* D49 (1994), pp. 3352–3355. DOI: `10.1103/PhysRevD.49.3352`. arXiv: `hep-ph/9311205 [hep-ph]`.

[24] Larry D. McLerran and Raju Venugopalan. “Green’s functions in the color field of a large nucleus”. In: *Phys. Rev.* D50 (1994), pp. 2225–2233. DOI: `10.1103/PhysRevD.50.2225`. arXiv: `hep-ph/9402335 [hep-ph]`.

[25] Istvan Montvay and Gernot Münster. *Quantum Fields on a Lattice.* Cambridge Monographs on Mathematical Physics. Cambridge University Press, 1994. DOI: `10.1017/CBO9780511470783`.

[26] D. Bodeker, Guy D. Moore, and K. Rummukainen. "Chern-Simons number diffusion and hard thermal loops on the lattice". In: *Phys. Rev.* D61 (2000), p. 056003. DOI: `10.1103/PhysRevD.61.056003`. arXiv: `hep-ph/9907545 [hep-ph]`.

[27] T. Lappi. "Energy density of the glasma". In: *Phys. Lett.* B643 (2006), pp. 11–16. DOI: `10.1016/j.physletb.2006.10.017`. arXiv: `hep-ph/0606207 [hep-ph]`.

[28] Kenji Fukushima. "Randomness in infinitesimal extent in the McLerran-Venugopalan model". In: *Phys. Rev.* D77 (2008), p. 074005. DOI: `10.1103/PhysRevD.77.074005`. arXiv: `0711.2364 [hep-ph]`.

[29] Kenji Fukushima and Francois Gelis. "The evolving Glasma". In: *Nucl. Phys.* A874 (2012), pp. 108–129. DOI: `10.1016/j.nuclphysa.2011.11.003`. arXiv: `1106.1396 [hep-ph]`.

[30] T. Lappi. "Wilson line correlator in the MV model: Relating the glasma to deep inelastic scattering". In: *Eur. Phys. J.* C55 (2008), pp. 285–292. DOI: `10.1140/epjc/s10052-008-0588-4`. arXiv: `0711.3039 [hep-ph]`.

[31] Dmitri Kharzeev and Eugene Levin. "Manifestations of high density QCD in the first RHIC data". In: *Phys. Lett.* B523 (2001), pp. 79–87. DOI: `10.1016/S0370-2693(01)01309-0`. arXiv: `nucl-th/0108006 [nucl-th]`.

[32] Bjoern Schenke, Prithwish Tribedy, and Raju Venugopalan. "Event-by-event gluon multiplicity, energy density, and eccentricities in ultrarelativistic heavy-ion collisions". In: *Phys. Rev.* C86 (2012), p. 034908. DOI: `10.1103/PhysRevC.86.034908`. arXiv: `1206.6805 [hep-ph]`.

[33] *openpixi_c source code.* URL: `https://gitlab.com/openpixi/openpixi_c` (visited on 10/03/2017).

[34] *openpixi_c code documentation.* URL: `http://openpixi.gitlab.io/openpixi_c/` (visited on 09/29/2018).

[35] *NetCDF.* URL: `https://www.unidata.ucar.edu/software/netcdf/` (visited on 09/29/2018).

[36] *NetCDF Software.* URL: `https://www.unidata.ucar.edu/software/netcdf/software.html` (visited on 09/29/2018).

[37] *FFTW Homepage.* URL: `http://www.fftw.org/` (visited on 10/04/2018).

[38] *Intel® Math Kernel Library Homepage.* URL: `https://software.intel.com/en-us/mkl` (visited on 10/04/2018).

[39] *VSC-3.* URL: `http://vsc.ac.at/systems/vsc-3/` (visited on 10/26/2018).

[40] *VSC-3: compute nodes.* 2014. URL: `http://vsc.ac.at/fileadmin/user_upload/vsc/documents/vsc3/VSC-3-poster-node-web.pdf` (visited on 10/26/2018).

[41] Paul Romatschke and Ulrike Romatschke. "Viscosity Information from Relativistic Nuclear Collisions: How Perfect is the Fluid Observed at RHIC?" In: *Phys. Rev. Lett.* 99 (2007), p. 172301. DOI: `10.1103/PhysRevLett.99.172301`. arXiv: `0706.1522 [nucl-th]`.

[42] Radoslaw Ryblewski and Wojciech Florkowski. "Highly-anisotropic hydrodynamics in 3+1 space-time dimensions". In: *Phys. Rev.* C85 (2012), p. 064901. DOI: `10.1103/PhysRevC.85.064901`. arXiv: `1204.2624 [nucl-th]`.

[43] Hirotsugu Fujii and Kazunori Itakura. "Expanding color flux tubes and instabilities". In: *Nucl. Phys.* A809 (2008), pp. 88–109. DOI: `10.1016/j.nuclphysa.2008.05.016`. arXiv: `0803.0410 [hep-ph]`.

[44] Ehab Abbas et al. "Centrality dependence of the pseudorapidity density distribution for charged particles in Pb-Pb collisions at $\sqrt{s_{\mathrm{NN}}} = 2.76$ TeV". In: *Phys. Lett.* B726 (2013), pp. 610–622. DOI: `10.1016/j.physletb.2013.09.022`. arXiv: `1304.0347 [nucl-ex]`.

[45] I. G. Bearden et al. "Charged meson rapidity distributions in central Au+Au collisions at s(NN)**(1/2) = 200-GeV". In: *Phys. Rev. Lett.* 94 (2005), p. 162301. DOI: `10.1103/PhysRevLett.94.162301`. arXiv: `nucl-ex/0403050 [nucl-ex]`.

[46] Christopher E. Flores. ""The Rapidity Density Distributions and Longitudinal Expansion Dynamics of Identified Pions from the STAR Beam Energy Scan"". In: *Nuclear Physics A* 956 (2016). The XXV International Conference on Ultrarelativistic Nucleus-Nucleus Collisions: Quark Matter 2015, pp. 280–283. ISSN: 0375-9474. DOI: `https://doi.org/10.1016/j.nuclphysa.2016.05.020`. URL: `http://www.sciencedirect.com/science/article/pii/S0375947416301403`.

[47] Jorge Casalderrey-Solana et al. "From full stopping to transparency in a holographic model of heavy ion collisions". In: *Phys. Rev. Lett.* 111 (2013), p. 181601. DOI: `10.1103/PhysRevLett.111.181601`. arXiv: `1305.4919 [hep-th]`.

[48] Wilke van der Schee. "Gravitational collisions and the quark-gluon plasma". PhD thesis. Utrecht U., 2014. arXiv: `1407.1849 [hep-th]`. URL: `http://dspace.library.uu.nl/handle/1874/294809`.

[49] Wilke van der Schee and Bjoern Schenke. "Rapidity dependence in holographic heavy ion collisions". In: *Phys. Rev.* C92.6 (2015), p. 064907. DOI: `10.1103/PhysRevC.92.064907`. arXiv: `1507.08195 [nucl-th]`.

[50] L. D. Landau. "On the multiparticle production in high-energy collisions". In: *Izv. Akad. Nauk Ser. Fiz.* 17 (1953), pp. 51–64.

[51] Deborah T. Marr et al. "Hyper-Threading Technology Architecture and Microarchitecture". In: *Intel Technology Journal* 06 (01 Feb. 14, 2002), pp. 4–15. ISSN: 1535766X. URL: `https://www.intel.com/content/dam/www/public/us/en/documents/research/2002-vol06-iss-1-intel-technology-journal.pdf`.

A. Plaquettes and Field Strength Tensor

To show that the plaquettes are, at leading order, equal to the exponential of the field strength tensor, one has to combine the definition of the plaquettes (eq. (3.23)) and the gauge links (eq. (3.19)):

$$\begin{aligned} U_{x,\mu\nu} &= U_{x,\mu} U_{x+\mu,\nu} U^\dagger_{x+\nu,\mu} U^\dagger_{x,\nu} \\ &= \exp(-iga_\mu \mathcal{A}_{x,\mu}) \exp(-iga_\nu \mathcal{A}_{x+\mu,\nu}) \exp(iga_\mu \mathcal{A}_{x+\nu,\mu}) \exp(iga_\nu \mathcal{A}_{x,\nu}). \end{aligned} \tag{A.1}$$

Since the fields $\mathcal{A}_\mu$ are elements of a non-Abelian Lie algebra, the Baker-Campbell-Hausdorff formula,

$$e^{\epsilon A} e^{\epsilon B} = e^{\epsilon(A+B) + \frac{\epsilon^2}{2}[A,B] + \mathcal{O}(\epsilon^3)}, \tag{A.2}$$

has to be used to write all terms under a single exponential. Using the linearity of the commutator $[\cdot, \cdot]$ and only collecting terms up to quadratic order yields the following formula for combining three exponentials,

$$\begin{aligned} e^{\epsilon A} e^{\epsilon B} e^{\epsilon C} &= \exp\left(\epsilon(A+B+C) + \frac{\epsilon^2}{2}[A,B] + \frac{1}{2}[\epsilon A + \epsilon B + \frac{\epsilon^2}{2}[A,B], \epsilon C] + \mathcal{O}(\epsilon^3)\right) \\ &= \exp\left(\epsilon(A+B+C) + \frac{\epsilon^2}{2}([A,B] + [A,C] + [B,C]) + \mathcal{O}(\epsilon^3)\right). \end{aligned} \tag{A.3}$$

With that, the plaquettes can be written as

$$\begin{aligned} U_{x,\mu\nu} = \exp\Bigg(& -\mathrm{i}g\,(a_\mu \mathcal{A}_{x,\mu} + a_\nu \mathcal{A}_{x+\mu,\nu} - a_\mu \mathcal{A}_{x+\nu,\mu} - a_\nu \mathcal{A}_{x,\nu}) \\ & - \frac{g^2 a_\mu a_\nu}{2}\,([\mathcal{A}_{x,\mu}, \mathcal{A}_{x+\mu,\nu}] - [\mathcal{A}_{x,\mu}, \mathcal{A}_{x,\nu}] - [\mathcal{A}_{x+\mu,\nu}, \mathcal{A}_{x+\nu,\mu}] + [\mathcal{A}_{x+\nu,\mu}, \mathcal{A}_{x,\nu}]) \\ & - \frac{g^2 a_\mu^2}{2}[\mathcal{A}_{x,\mu}, \mathcal{A}_{x+\nu,\mu}] - \frac{g^2 a_\nu^2}{2}[\mathcal{A}_{x+\mu,\nu}, \mathcal{A}_{x,\nu}] + \mathcal{O}(a^3)\Bigg). \end{aligned} \tag{A.4}$$

Using the following approximation,

$$\mathcal{A}_{x+\nu,\mu} = \mathcal{A}_{x,\mu} + a_\nu \partial_\nu \mathcal{A}_{x,\mu} + \mathcal{O}(a^2), \tag{A.5}$$

the last two terms in eq. (A.4) vanish in quadratic order, yielding

$$\begin{aligned}
U_{x,\mu\nu} = {}& \exp\Bigg(- \mathrm{i}g \left(a_\mu \mathcal{A}_{x,\mu} + a_\nu \mathcal{A}_{x+\mu,\nu} - a_\mu \mathcal{A}_{x+\nu,\mu} - a_\nu \mathcal{A}_{x,\nu}\right) \\
& \quad - \frac{g^2 a_\mu a_\nu}{2} \left([\mathcal{A}_{x,\mu}, \mathcal{A}_{x,\nu}] - [\mathcal{A}_{x,\mu}, \mathcal{A}_{x,\nu}] - [\mathcal{A}_{x,\nu}, \mathcal{A}_{x,\mu}] + [\mathcal{A}_{x,\mu}, \mathcal{A}_{x,\nu}]\right) + \mathcal{O}(a^3)\Bigg) \\
= {}& \exp\Bigg(- \mathrm{i}g a_\mu a_\nu \left(\frac{\mathcal{A}_{x+\mu,\nu} - \mathcal{A}_{x,\nu}}{a_\mu} - \frac{\mathcal{A}_{x+\nu,\mu} - \mathcal{A}_{x,\mu}}{a_\nu} - \mathrm{i}g[\mathcal{A}_{x,\mu}, \mathcal{A}_{x,\nu}]\right) + \mathcal{O}(a^3)\Bigg) \\
= {}& \exp\Big(-iga_\mu a_\nu \left(\partial_\mu \mathcal{A}_{x,\nu} - \partial_\nu \mathcal{A}_{x,\mu} - \mathrm{i}g[\mathcal{A}_{x,\mu}, \mathcal{A}_{x,\nu}]\right) + \mathcal{O}(a^3)\Big) \\
= {}& \exp\left(-iga_\mu a_\nu (\mathcal{F}_{x,\mu\nu} + \mathcal{O}(a))\right),
\end{aligned} \tag{A.6}$$

which is the claimed result.

B. Conservation of Gauss Constraint

If the equation of motion solver preserves the Gauss constraint, calculating it at different time steps should not change the result,

$$\begin{aligned}&\sum_{i=1}^{3}\frac{\mathcal{E}_{x,i}-U^{\dagger}_{x-i-0/2,i}\mathcal{E}_{x-i,i}U_{x-i-0/2,i}}{a_i}-\rho^a_x t_a \overset{!}{=}\\&\sum_{i=1}^{3}\frac{\mathcal{E}_{x+0,i}-U^{\dagger}_{x-i+0/2,i}\mathcal{E}_{x+0-i,i}U_{x-i+0/2,i}}{a_i}-\rho^a_{x+0} t_a.\end{aligned}\tag{B.1}$$

Here $U_{x-i\pm 0/2,i}$ denotes the gauge link at time $t \pm \Delta t/2$. Expressing the electric field and gauge links at the later time step with eqs. (3.55) and (3.58) yields

$$\begin{aligned}\sum_{i=1}^{3}\frac{1}{a_i}\Big[&\mathcal{E}_{x,i}-\Big(\exp(-\mathrm{i}ga_0a_i\mathcal{E}_{x-i,i})U_{x-i-0/2,i}\Big)^{\dagger}\mathcal{E}_{x-i,i}\exp(-\mathrm{i}ga_0a_i\mathcal{E}_{x-i,i})U_{x-i-0/2,i}\\&+\sum_{j\neq i}{}'\operatorname{Im}\operatorname{Tr}(t^aU_{x+0/2,ij}+t^aU_{x+0/2,i-j})t_a-a_0\,j^i_{x+0/2}\\&-\sum_{j\neq i}{}'U^{\dagger}_{x-i+0/2,i}\operatorname{Im}\operatorname{Tr}(t^aU_{x-i+0/2,ij}+t^aU_{x-i+0/2,i-j})t_aU_{x-i+0/2,i}\\&+a_0U^{\dagger}_{x-i+0/2,i}\,j^i_{x-i+0/2}U_{x-i+0/2,i}\Big],\end{aligned}\tag{B.2}$$

where the primed sum is introduced as a short hand notation, defined in the following way

$$\sum_{j\neq i}{}' := \frac{2a_0}{ga_i}\sum_{j\neq i}\frac{1}{a_j^2}.\tag{B.3}$$

The terms involving either ρ or j are treated separately and will be dealt with later. For now, only the terms with E-fields and plaquettes are of interest. Since $\exp(-\mathrm{i}ga_0a_i\mathcal{E}_{x-i,i})$ commutes with $\mathcal{E}_{x-i,i}$, the two exponentials in the first line of eq. (B.2) cancel each other, leaving only the desired terms for the Gauss constraint of the original time step, plus the terms with the primed sum. Using the Fierz identity for the generators of SU(N)

$$(t^a)_{\alpha\beta}(t^a)_{\gamma\delta}=\frac{1}{2}\left(\delta_{\alpha\delta}\delta_{\beta\gamma}-\frac{1}{N}\delta_{\alpha\beta}\delta_{\gamma\delta}\right),\tag{B.4}$$

the imaginary trace of a group element X can be rewritten in the following way

$$\begin{aligned}
\operatorname{Im}\operatorname{Tr}(t^a X)t^a &= \frac{1}{2\mathrm{i}}\Big((t^a)_{\alpha\beta}X_{\beta\alpha} - ((t^a)_{\alpha\beta}X_{\beta\alpha})^\dagger\Big)(t^a)_{\gamma\delta} \\
&= \frac{1}{2\mathrm{i}}\Big((t^a)_{\alpha\beta}(t^a)_{\gamma\delta}X_{\beta\alpha} - (t^a)_{\beta\alpha}(t^a)_{\gamma\delta}X^\dagger_{\alpha\beta}\Big) \\
&= \frac{1}{4\mathrm{i}}\left(X_{\gamma\delta} - X^\dagger_{\gamma\delta} - \frac{1}{N}(X_{\alpha\alpha} - X^\dagger_{\alpha\alpha})\delta_{\gamma\delta}\right) \\
&=: \frac{1}{2}\left[X\right]_{\mathrm{ah}}\,,
\end{aligned} \tag{B.5}$$

where $[X]_{\mathrm{ah}}$ denotes the anti hermitian, traceless part of X. It is easy to see that $[\cdot]_{\mathrm{ah}}$ commutes with gauge transformations.

$$\begin{aligned}
U^\dagger 2\mathrm{i}\left[X\right]_{\mathrm{ah}} U &= U^\dagger X U - U^\dagger X^\dagger U - \frac{1}{N}\operatorname{Tr}(X - X^\dagger)U^\dagger \mathbb{1} U \\
&= U^\dagger X U - (U^\dagger X U)^\dagger - \frac{1}{N}\operatorname{Tr}(U^\dagger X U - (U^\dagger X U)^\dagger)\mathbb{1} \\
&= 2\mathrm{i}\left[U^\dagger X U\right]_{\mathrm{ah}}
\end{aligned} \tag{B.6}$$

Together with eq. (3.47) the primed sum now reads

$$\begin{aligned}
&\sum_{i=1}^{3}\sum_{j\neq i}{}' \frac{1}{2a_i}\Bigg(\left[U_{x+0/2,ij}\right]_{\mathrm{ah}} + \left[U_{x+0/2,i-j}\right]_{\mathrm{ah}} \\
&\qquad - \left[U^\dagger_{x-i+0/2,i}U_{x-i+0/2,ij}U_{x-i+0/2,i}\right]_{\mathrm{ah}} \\
&\qquad - \left[U^\dagger_{x-i+0/2,i}U_{x-i+0/2,i-j}U_{x-i+0/2,i}\right]_{\mathrm{ah}}\Bigg) = \\
&\frac{a_0}{g}\sum_{i=1}^{3}\sum_{j\neq i}\frac{1}{a_i^2 a_j^2}\Bigg(\left[U_{x+0/2,ij}\right]_{\mathrm{ah}} + \left[U_{x+0/2,i-j}\right]_{\mathrm{ah}} \\
&\qquad - \left[U_{x+0/2,j-i}\right]_{\mathrm{ah}} - \left[U_{x+0/2,-j-i}\right]_{\mathrm{ah}}\Bigg).
\end{aligned} \tag{B.7}$$

The second together with the third term is antisymmetric under the exchange of the indexes. Since the sum over i and j goes over every possible index pair with $i \neq j$, these two terms cancel. The first as well as the last term are antisymmetric on their own because

$$\left[U_{x,ij}\right]_{\mathrm{ah}} = \left[U^\dagger_{x,ji}\right]_{\mathrm{ah}} = -\left[U_{x,ji}\right]_{\mathrm{ah}}\,. \tag{B.8}$$

Therefore the whole primed sum vanishes.

Regarding the terms with ρ and j, one has to show that

$$-\rho^a_x t_a = -\rho^a_{x+0}t_a - \sum_{i=1}^{3}\frac{a_0}{a_i}\left(j^i_{x+0/2} - U^\dagger_{x-i+0/2,i}\,j^i_{x-i+0/2}U_{x-i+0/2,i}\right). \tag{B.9}$$

However, this is just the discrete continuity equation, which every particle fulfills by definition, as this was the starting point for the derivation of the charge and current update (see section 3.5). With that, it is finally proven that the equations of motion exactly preserve the Gauss constraint.