Astronomy and Computing 16 (2016) 109–130

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

The moving mesh code SHADOWFAX

B. Vandenbroucke *, S. De Rijcke

Department of Physics & Astronomy, Ghent University, Krijgslaan 281, S9, 9000 Gent, Belgium

ARTICLE INFO

Article history:
Received 18 December 2015
Accepted 6 May 2016
Available online 18 May 2016

Keywords:
Methods: numerical
Hydrodynamics

ABSTRACT

We introduce the moving mesh code SHADOWFAX, which can be used to evolve a mixture of gas, subject to the laws of hydrodynamics and gravity, and any collisionless fluid only subject to gravity, such as cold dark matter or stars. The code is written in C++ and its source code is made available to the scientific community under the GNU Affero General Public Licence. We outline the algorithm and the design of our implementation, and demonstrate its validity through the results of a set of basic test problems, which are also part of the public version. We also compare SHADOWFAX with a number of other publicly available codes using different hydrodynamical integration schemes, illustrating the advantages and disadvantages of the moving mesh technique.

1. Introduction

Modern simulations of galaxy formation and evolution crucially depend on an accurate treatment of the hydrodynamics of the interstellar medium (ISM) (Vogelsberger et al., 2014; Schaye et al., 2015). The ISM fuels star formation and is disrupted by stellar feedback, and it is this complex interplay that at least partly governs the observable content of galaxies (Verbeke et al., 2015). If we want to be able to compare simulated galaxies with observations, we need to properly resolve these effects.

Hydrodynamics is also important on smaller scales, when simulating star-forming clouds (Greif et al., 2011; Dobbs, 2015), feedback from a single star (Geen et al., 2015), or even planet formation in a circumstellar disc (Duffell and MacFadyen, 2012). A robust hydrodynamical integration scheme, optionally extended with magnetic fields, self-gravity or radiation transport, is hence an indispensable tool for many astrophysical simulators.

Historically, two major classes of hydrodynamical solvers have been developed: grid based Eulerian techniques (Teyssier, 2002; Keppens et al., 2012), and particle-based Lagrangian techniques (Springel, 2005; Price, 2012). Both discretize the fluid as a finite set of fluid elements. In the former, the fluid elements are cells, usually defined through a (hierarchical) Cartesian grid, which have a fixed position in space, but can be allowed to refine or derefine according to the quality of the integration. In the latter, the fluid elements are particles, which move along with the flow, with the hydrodynamics being expressed as inter-particle forces. It is generally acknowledged that grid based Eulerian techniques are more accurate at solving the equations of hydrodynamics, especially since many particle-based implementations have fundamental difficulties in resolving hydrodynamical instabilities (Agertz et al., 2007). Nonetheless, Lagrangian techniques are widely used to simulate systems with a high dynamic range, like cosmological simulations and simulations of galaxies, since they more naturally concentrate computational resources on regions of interest, and provide a Galilean invariant reference frame.

Recently, a new class of hydrodynamical solvers has been developed, mainly through the work of Springel (2010), which aims to combine the advantages of Eulerian and Lagrangian techniques (see also Duffell and MacFadyen, 2011; Yalinewich et al., 2015). This new technique uses a moving grid to discretize the fluid, and combines an unstructured grid based finite volume integration scheme with the Lagrangian nature of a particle method. We will refer to this method as a *moving mesh technique*.

A number of moving mesh codes are presented in the literature (Springel, 2010; Duffell and MacFadyen, 2011), but only two of them are publicly available: RICH ascl:1410.005 (Yalinewich et al., 2015), written in C++, and FVMHD3D,[1] written in the parallel object-oriented language CHARM++ (Gaburov et al., 2012).

In this paper, we introduce the new, publicly available moving mesh code SHADOWFAX (the logo of the code is shown in Fig. 1). SHADOWFAX is written in C++, and makes ample use of the object-oriented capabilities of the language to provide an easy to extend

* Corresponding author.
E-mail addresses: bert.vandenbroucke@ugent.be (B. Vandenbroucke), sven.derijcke@ugent.be (S. De Rijcke).

http://dx.doi.org/10.1016/j.ascom.2016.05.001
2213-1337/

[1] https://github.com/egaburov/fvmhd3d.

Fig. 1. The SHADOWFAX logo.

framework. The code is parallelized for use on distributed memory systems using the Message Passing Interface (MPI),[2] and makes use of the open source BOOST C++ libraries[3] to extend basic C++ language features. The code supports input and output using the HDF5 library[4] in a format compatible with the output of GADGET2 ascl:0003.001 (Springel, 2005), GIZMO ascl:1410.003 (Hopkins, 2015) and SWIFT[5] (Gonnet et al., 2013). A user friendly compilation process is guaranteed through the use of CMAKE.[6]

The hydrodynamical algorithm implemented in SHADOWFAX is the same as described by Springel (2010), but with an additional per-face slope limiter and flux limiter, and optional alternative approximate Riemann solvers. The gravitational calculation is the same as the tree force calculation in GADGET2 (Springel, 2005), and uses the same relative tree opening criterion and Ewald summation technique for periodic boundary conditions. We have ported this algorithm to an object-oriented version, which makes use of compile-time polymorphism using C++ templates. This ensures a clear separation of the algorithmic details underlying the tree walk from the actual physics involved with the gravitational calculation. This way, it is much easier to focus on one particular aspect of the code, e.g. scalability, precision etc., without needing to worry about other aspects.

Likewise, we have separated the geometrical details contained in the moving mesh from the hydrodynamical integration as mush as possible, to make it easier to replace parts of the algorithm (e.g. the Riemann solver, the grid etc.) by simply implementing an alternative class.

Our code is predominantly meant to be used in astrophysical simulations of galaxy formation and evolution, but could have applications in other areas of science as well, as it is not difficult to replace the Euler equations of hydrodynamics by e.g. the shallow water equations by implementing a different Riemann solver. Furthermore, the Voronoi grid used to discretize the fluid can also be used for other purposes, e.g. for the suppression of Poisson noise in randomly sampled distributions through Lloyd's algorithm (Lloyd, 1982), or as density estimator in N-body simulations (Cloet-Osselaer et al., 2014).

In this paper, we outline the basic working of SHADOWFAX. We mainly focus on the C++ implementation and the object-oriented design of our code, and compare our code with other hydrodynamical solvers on a number of test problems. Although the current version of SHADOWFAX focuses more on design and accuracy than on performance, we also highlight some basic strong and weak scaling tests. Performance optimizations and extra physical ingredients (e.g. gas cooling, star formation and stellar feedback etc.) will be added in future versions of the code. The source code of SHADOWFAX is publicly available from https://github.com/AstroUGent/shadowfax, and is distributed under the GNU Affero General Public Licence.[7]

2. Algorithm

Many of the algorithms implemented in SHADOWFAX were already discussed in Springel (2005, 2010). For completeness, we summarize them below and point out the differences where necessary.

SHADOWFAX is based on a finite volume method, which subdivides the computational box into a (large) number of small cells. The hydrodynamical integration is governed by the exchange of fluxes between these cells.

These fluxes involve the *conserved variables*: mass (m), momentum ($\boldsymbol{p}$) and total energy (E). The Euler equations of hydrodynamics however are usually formulated in terms of *primitive variables*: density (ρ), flow velocity ($\boldsymbol{v}$) and pressure (p). The pressure is sometimes replaced by the thermal energy (u) or some form of entropic function of the fluid, by using the *equation of state* of the fluid. In this work, we will always assume an ideal gas, with an equation of state of the form

$$p = (\gamma - 1)\rho u, \tag{1}$$

where γ is the *adiabatic index* of the gas, for which we will adopt the value $\gamma = 5/3$, unless otherwise stated. The conserved variables and primitive variables can be converted into one another whenever a volume (V) is available, since

$$m = \rho V \tag{2}$$

$$\boldsymbol{p} = m\boldsymbol{v} \tag{3}$$

$$E = mu + \frac{1}{2}m\boldsymbol{v}^2. \tag{4}$$

It is common practise to combine the conserved and primitive variables into two *state vectors*,

$$\boldsymbol{Q} = \begin{pmatrix} m \\ \boldsymbol{p} \\ E \end{pmatrix} \quad \text{and} \quad \boldsymbol{W} = \begin{pmatrix} \rho \\ \boldsymbol{v} \\ p \end{pmatrix}. \tag{5}$$

The change in conserved variables $\boldsymbol{Q}_i$ for a cell i, during an integration time step of length Δt, is then given by

$$\Delta\boldsymbol{Q}_i = -\Delta t \sum_j A_{ij}\boldsymbol{F}_{ij}\left(\boldsymbol{W}_i, \boldsymbol{W}_j, \nabla\boldsymbol{W}_i, \nabla\boldsymbol{W}_j, \boldsymbol{v}_{ij}, \Delta t\right), \tag{6}$$

where A_{ij} is the surface area of the interface between cell i and cell j. $\boldsymbol{F}_{ij}$ is the flux between cell i and cell j, which in general depends on the primitive variables of both cells and their gradients, the velocity $\boldsymbol{v}_{ij}$ of the face with respect to a frame of reference fixed to the simulation box, and the integration time step.

When formulated in this way, the finite volume method can be applied to any discretization of the fluid, as long as this discretization yields volumes to convert conserved variables to primitive variables, and defines a concept of neighbour relations between cells, and an associated surface area and velocity for the neighbour interface. It can even be applied to mesh-free, particle-based methods (Hopkins, 2015).

In the case of a moving mesh method, the discretization is given by an unstructured Voronoi mesh, a 2D example of which is shown in Fig. 2. The mesh is defined by means of a set of mesh generating points (*generators*), with the cell associated with a specific generator containing the region of space closest to that generator. A Voronoi mesh can be defined in D dimensions, but

[2] http://www.mpi-forum.org.
[3] http://www.boost.org.
[4] https://www.hdfgroup.org/HDF5.
[5] http://icc.dur.ac.uk/swift/.
[6] https://cmake.org.
[7] http://www.gnu.org/licenses.