



SCALABLE PARALLEL ASTROPHYSICAL CODES FOR EXASCALE

OpenGADGET3

**A Practical Guide to Numerical Hydrodynamics
and Cosmological Simulations**

Based on the Hydrodynamics Tutorial Series by
Klaus Dolag (Universitäts-Sternwarte München / LMU Munich)

*A student handbook for graduate courses
in numerical astrophysics*

Editors: G. Perna [ENGINSOFT], E. Sciacca, G. Taffoni [INAF].

June, 2026

Repository: <https://gitlab.lrz.de/MAGNETICUM/Hydro-OpenGadget3>



Co-funded by
the European Union



EuroHPC
Joint Undertaking

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain under grant agreement No 101093441

Scalable Parallel Astrophysical Codes for Exascale

DISCLAIMER

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain. Neither the European Union nor the granting authority can be held responsible for them.



Contents

Preface	ix
I Theoretical Background	1
1 Numerical Hydrodynamics and SPH: Theory and Algorithms	3
1.1 Fluid Dynamics in Astrophysics	3
1.1.1 The fluid approximation and its limits	3
1.1.2 The Euler equations	4
1.1.3 Lagrangian form	5
1.1.4 Conservation laws and their physical content	5
1.2 Shock Waves and Discontinuities	5
1.2.1 Jump conditions (Rankine–Hugoniot relations)	5
1.2.2 The Riemann problem	6
1.2.3 The Sedov–Taylor blast wave	7
1.2.4 Virialization	7
1.2.5 Sound waves	7
1.3 Magnetohydrodynamics	7
1.3.1 MHD wave modes	8
1.3.2 The $\nabla \cdot \mathbf{B} = 0$ constraint	8
1.4 Transport Phenomena: Viscosity and Conduction	8
1.4.1 Thermal conduction	9
1.4.2 Physical viscosity	9
1.5 Gravitational Collapse and the N-body Problem	9
1.5.1 The free-fall time and structure formation	9
1.5.2 The collisionless Boltzmann equation	10
1.5.3 The N-body approximation	10
1.5.4 Two-body relaxation and the collisionless limit	10
1.6 Smoothed Particle Hydrodynamics	11
1.6.1 Kernel density estimation	11
1.6.2 The smoothing length and adaptive resolution	11
1.6.3 Smoothing kernels	12
1.6.4 Particle distributions and numerical noise	12
1.6.5 The SPH Lagrangian and equations of motion	13
1.6.6 Artificial viscosity	13
1.6.7 Artificial conductivity	14
1.6.8 The Meshless Finite Mass (MFM) method	14
1.6.9 SPH versus Eulerian methods: a brief comparison	15
1.7 Gravitational Algorithms	15
1.7.1 Direct summation	15

1.7.2	The Barnes–Hut tree algorithm	15
1.7.3	The Particle-Mesh (PM) method	16
1.7.4	The TreePM hybrid method	16
1.7.5	Gravitational softening	17
1.7.6	Time integration	17
1.7.7	MPI and OpenMP parallelism	17
1.8	Sub-grid Physics	18
1.8.1	Radiative cooling	18
1.8.2	Star formation	18
1.8.3	Stellar feedback: galactic winds	19
1.9	Cosmological Simulations	19
1.9.1	The expanding universe and co-moving coordinates	19
1.9.2	Initial conditions and perturbation theory	20
1.9.3	Large-scale structure observables	20
1.9.4	The GADGET family and OpenGADGET3	20
II	Installation and Workflow	21
2	Getting the Code Running	23
2.1	Prerequisites and Environment	23
2.2	Directory Structure	24
2.3	Obtaining the Code	24
2.4	The Configuration File <code>Config.sh</code>	25
2.4.1	How compile-time flags work	25
2.4.2	Standard configurations per tutorial	25
2.4.3	Flag reference table	26
2.5	Compiling the Code	28
2.6	The Parameter File in Detail	28
2.6.1	File paths	28
2.6.2	Time control	29
2.6.3	Unit system	29
2.6.4	Boundaries and gravity	29
2.6.5	Gravitational softening	30
2.6.6	SPH parameters	30
2.6.7	Sub-grid physics	31
2.7	The SLURM Job Script	32
2.8	Restarting an Interrupted Run	33
2.9	The Standard Tutorial Workflow	33
3	Snapshot Format and I/O	35
3.1	File Formats	35
3.2	Particle Types	35
3.3	Data Blocks	36
3.4	Reading Snapshots with Python	36
3.4.1	g3read.py (Format 1 and 2)	36
3.4.2	h5py (Format 3 / HDF5)	37
3.4.3	Computing temperature	37
3.5	Visualisation Tools	38
3.6	The Snapshot Header	38
3.7	Computing Physical Quantities	38
3.7.1	Temperature	38

3.7.2	Pressure and Mach number	39
3.8	Tracking Individual Particles	39
3.9	Visualisation Tools	39
4	Generating Initial Conditions	41
4.1	IC File Structure	41
4.2	Setting Up Simple ICs with Python	42
4.2.1	Uniform grid (cubic lattice)	42
4.2.2	Non-uniform density: the spherical cloud	43
4.3	Slab and Non-cubic Geometries	44
4.4	Cosmological Initial Conditions	44
4.4.1	N-GenIC	44
4.4.2	MUSIC	45
4.4.3	Cosmological parameter file settings	46
III	Tutorial Compendium	47
5	Tutorial T0: Environment Setup and First Simulation	51
5.1	Physical Setup	51
5.2	Connecting to the Cluster	51
5.3	Step 1: Directory Setup	52
5.4	Step 2: Get the Code and Configuration Files	52
5.5	Step 3: Config.sh	52
5.6	Step 4: Initial Conditions	52
5.7	Step 5: Parameter File	53
5.8	Step 6: Running the Simulation	54
5.9	Step 7: Visualising the Results	54
5.9.1	Single-frame plot	54
5.9.2	Animation	55
5.9.3	Other languages	56
6	Tutorial T1: Free-fall Collapse	59
6.1	Physical Background	59
6.1.1	Free-fall time	59
6.1.2	Hierarchical structure formation	59
6.2	Changes from T0	60
6.3	Config.sh	60
6.4	Parameter File	60
6.5	Initial Conditions	61
6.5.1	Using the provided IC	61
6.5.2	Generating the IC from scratch (bonus)	61
6.6	Running the Simulation	62
6.7	Visualisation and Analysis	62
6.7.1	Animating the collapse	62
6.7.2	Measuring the free-fall time	63
6.7.3	Effect of the force accuracy parameter	64
7	Tutorial T2: Stream Lines and Cosmological Flows	67
7.1	Physical Background	67
7.1.1	Stream lines vs flow lines	67
7.1.2	Cosmological context	67

7.1.3	Grid-based velocity interpolation and NGP/CIC	67
7.2	Config.sh	67
7.3	Parameter File	68
7.4	Running and Computing Stream Lines	68
7.5	T2b: Stream Lines in a Cosmological Snapshot	69
7.5.1	Stream lines: instantaneous velocity topology	69
7.5.2	Flow lines: particle trajectories	69
7.5.3	TSC velocity grid and bilinear interpolation	70
8	Tutorial T3: Density Estimates and SPH Kernels	77
8.1	Physical Background	77
8.2	Initial Particle Distributions	77
8.3	Running Multiple Simulations	78
8.4	Analysis: Density Bias and Scatter	79
9	Tutorial T4: The Riemann Problem	83
9.1	Physical Background	83
9.1.1	Solving for P^* graphically	83
9.2	Config.sh: Three Versions	83
9.3	Parameter File	84
9.4	Initial Conditions	85
9.5	Progressive Improvement Strategy	85
9.6	Analysis	85
9.6.1	IDL and Fortran tools	87
10	Tutorial T5: The Evrard Collapse	89
10.1	Physical Background	89
10.1.1	The Evrard setup	89
10.1.2	What to expect: three phases	89
10.2	Config.sh	89
10.3	Parameter File	90
10.4	Initial Conditions	91
10.4.1	Using the provided IC	91
10.4.2	Building the IC from scratch	91
10.5	Running	92
10.6	Analysis	92
10.6.1	Energy evolution and virial theorem	92
10.6.2	Comparison with the PPM Eulerian solution	93
10.6.3	Density histogram: verification of the $1/r$ profile	93
11	Tutorial T6: Sound Wave Propagation	97
11.1	Physical Background	97
11.2	Config.sh	97
11.3	Parameter File	98
11.4	Initial Conditions	99
11.5	Simulation Results	100
11.5.1	Wave profiles	100
11.5.2	Comparison at different times	100
11.6	MPI+OpenMP Parallelism	100
11.7	Analysis	100
12	Tutorial T7a: Supernova Blast Wave	105

12.1	Physical Background	105
12.2	Config.sh and Parameter File	105
12.3	Initial Conditions	106
12.4	Results	107
12.5	Analysis: Sedov Scaling	107
13	Tutorial T7b: Flow through a Nozzle	109
13.1	Physical Background	109
13.2	Boundary Particles	109
13.3	Config.sh	109
13.4	Initial Conditions	110
13.5	Parameter File	110
13.6	Results	110
14	Tutorial T8a: Thermal Conduction	113
14.1	Physical Background	113
14.2	Config.sh	113
14.3	Parameter File	114
14.4	Initial Conditions and Running	114
14.5	Analysis	114
15	Tutorial T8b: Heat Transport and the Ancient Air Conditioner	119
15.1	Physical Motivation	119
15.1.1	The refrigeration cycle	119
15.1.2	Ancient theatres and natural ventilation	119
15.1.3	The adiabatic temperature change	120
15.2	Geometry and Setup	120
15.3	Config.sh	120
15.4	Parameter File	121
15.5	Setting Up the Initial Conditions	122
15.6	Running the Simulations	123
15.7	Analysis: Temperature in the Chamber	123
15.7.1	Profiles along the x-axis	124
15.7.2	What to look for	124
15.8	Programming Goals	125
16	Tutorial T9a: Navier–Stokes Viscosity (Couette Flow)	129
16.1	Physical Background	129
16.2	Config.sh	129
16.3	Initial Conditions	130
16.4	Parameter File	130
16.5	Results	131
16.6	Analysis	131
17	Tutorial T9b: Viscous Jet and Kelvin–Helmholtz Instability	135
17.1	Physical Background	135
17.2	Initial Conditions	135
17.3	Four Simulations to Compare	135
17.4	Results	136
17.5	Analysis	136
18	Tutorial T10: MHD Shock Tube	141
18.1	Physical Background	141

18.1.1	MHD wave modes	141
18.1.2	Test 5A of Brio & Wu (1988)	141
18.1.3	The $\nabla \cdot \mathbf{B} = 0$ constraint	142
18.2	Config.sh	142
18.3	Parameter File	142
18.4	Initial Conditions	142
18.5	Results	143
18.6	Analysis	143
19	Tutorial T11: Disk Galaxy with Star Formation	147
19.1	Physical Background	147
19.1.1	The Kennicutt–Schmidt relation	147
19.1.2	Radiative cooling and the two-phase ISM	147
19.1.3	Galactic winds	147
19.2	Config.sh	147
19.3	Initial Conditions	148
19.4	Parameter File	148
19.5	Visualising the Galaxy	149
19.6	The Kennicutt–Schmidt Relation	150
19.7	Star Formation History	151
IV	Post-processing and Analysis	153
20	Post-processing, Visualisation, and Derived Quantities	155
20.1	The g3read.py Workflow	155
20.2	Radial Profiles	156
20.3	Loading Data with yt	156
20.4	Cosmological Unit Conversions	157
20.5	Power Spectrum and Two-Point Statistics	157
A	Complete Annotated Parameter File	159
B	Complete Config.sh Flag Reference	163
C	GADGET Binary File Format	165
C.1	Format 2 Block Structure	165
C.2	Reading Format 2 in Python	165
D	Quick Reference Card	167
D.1	Essential Commands	167
D.2	Common Pitfalls	168

List of Tables

1.1	SPH versus AMR: key properties for astrophysical applications.	15
2.1	Commonly used <code>Config.sh</code> flags.	27
3.1	Particle types in OpenGADGET3 snapshots.	35
3.2	Standard data blocks in OpenGADGET3 snapshots.	36
17.1	Four T9b simulations, each adding one physical ingredient.	136
18.1	Initial states for MHD shock tube test 5A (Brio and Wu, 1988). Code units, $\gamma = 2$	141
19.1	Available galaxy IC files for Tutorial T11.	148
B.1	Complete <code>Config.sh</code> flag reference.	163
D.1	Required <code>Config.sh</code> changes per tutorial.	168

Preface

This manual is a practical guide to OpenGADGET3 , an advanced smoothed particle hydrodynamics (SPH) code for astrophysical and cosmological simulations. It is aimed at graduate students in astrophysics who have solid backgrounds in classical mechanics and thermodynamics but are new to numerical hydrodynamics.

The book grew from the Hydrodynamics Tutorial Series developed by Klaus Dolag at LMU Munich, which guides students through a progression of experiments from a simple uniform gas box to a full disk galaxy simulation with star formation and feedback. This manual reorganises and substantially expands that material into a coherent reference, supplemented with material from the broader literature on GADGET-family codes, inspired by the tutorials of the SPACE CoE held in Sexten, Italy, December 2025 winter school.

Structure. Part I provides the essential theoretical background in a single, dense chapter. Part II covers the complete practical workflow: installation, the configuration system, parameter files, initial condition generation, and HPC job submission. Part III is a self-contained tutorial compendium, with one chapter per tutorial experiment. Part IV covers post-processing and visualisation. Appendices provide reference material: the complete annotated parameter file, all `Config.sh` flags, the binary file format specification, and a quick-reference command card.

Conventions. Physical equations are in CGS units unless noted. Code-unit quantities are written in italics with the unit system defined in Section 2.6.3. Shell commands are shown in monospace. Parameter names are shown as `ParameterName`. Compile-time flags are shown as `FLAG_NAME`.

Part I

Theoretical Background

Chapter 1

Numerical Hydrodynamics and SPH: Theory and Algorithms

This chapter provides the theoretical foundation for OpenGADGET3 and for numerical hydrodynamics in astrophysics more broadly. It is written for a reader with a standard graduate-level background in classical mechanics and thermodynamics, but without prior exposure to numerical fluid dynamics. The chapter is intentionally self-contained: full derivations are given for the most important results, with references to the primary literature for extensions and subtleties. The chapter is longer than strictly necessary for running the tutorial simulations, but a deeper understanding of the numerics is essential for interpreting results correctly and for diagnosing problems when they arise.

The chapter proceeds from the physics of continuous fluids (§1.1) to the N-body problem for collisionless systems (§1.5), to the SPH discretisation (§1.6), to the gravitational algorithms (§1.7), and finally to the sub-grid physics modules (§1.8) and cosmological extensions (§1.9).

1.1 Fluid Dynamics in Astrophysics

1.1.1 The fluid approximation and its limits

The fluid description replaces the discrete nature of matter (atoms, ions, electrons) with a continuous medium characterised by macroscopic fields: mass density $\rho(\mathbf{r}, t)$, velocity $\mathbf{v}(\mathbf{r}, t)$, pressure $P(\mathbf{r}, t)$, and temperature $T(\mathbf{r}, t)$. This approximation is valid when two conditions hold simultaneously: (i) the *mean free path* λ_{mfp} between particle collisions is much smaller than the macroscopic scale of interest L , ensuring that local thermodynamic equilibrium (LTE) is established; and (ii) the number of particles in a volume element ℓ^3 (where $\lambda_{\text{mfp}} \ll \ell \ll L$) is large enough that fluctuations are negligible.

The ratio $\text{Kn} = \lambda_{\text{mfp}}/L$ is the *Knudsen number*. The fluid approximation requires $\text{Kn} \ll 1$.

For a fully ionised plasma, Coulomb scattering between ions gives a mean free path (Spitzer, 1962)

$$\lambda_{\text{mfp}} \approx 23 \text{ kpc} \left(\frac{T}{10^8 \text{ K}} \right)^2 \left(\frac{n}{10^{-3} \text{ cm}^{-3}} \right)^{-1}, \quad (1.1)$$

where n is the number density of ions and T the temperature. For the intracluster medium (ICM) of a galaxy cluster ($T \sim 10^8 \text{ K}$, $n \sim 10^{-3} \text{ cm}^{-3}$) this gives $\lambda_{\text{mfp}} \sim 23 \text{ kpc}$. The ICM extends over $\sim 1 \text{ Mpc}$, so $\text{Kn} \sim 0.02$: the fluid approximation is marginally valid. However, the ICM is permeated by a tangled magnetic field ($B \sim 1\text{--}10 \mu\text{G}$) whose associated ion Larmor radius is $r_L = m_p v_\perp / (eB) \sim 10^{-10} \text{ kpc}$, ten orders of magnitude smaller than the mean free

path. Particle diffusion perpendicular to field lines is suppressed to the Larmor scale, making the ICM behave as a well-coupled fluid on all astrophysically relevant scales.

Denser astrophysical environments are more straightforwardly in the fluid regime. The ISM in molecular clouds ($n \sim 10^2\text{--}10^6 \text{ cm}^{-3}$, $T \sim 10\text{--}100 \text{ K}$) has $\lambda_{\text{mfp}} \lesssim 10^{-5} \text{ pc}$, far smaller than molecular cloud sizes of $\sim 10\text{--}100 \text{ pc}$. Stellar interiors are in the fluid regime by many orders of magnitude.

Dark matter, by contrast, is collision-free: its mean free path is essentially infinite. OpenGADGET3 treats collisionless components (dark matter, collisionless stellar particles) as an N-body system coupled to the gas only through gravity. The fluid approximation does not apply to them.

1.1.2 The Euler equations

The evolution of a compressible, inviscid fluid (Euler fluid) is governed by three conservation laws. In Eulerian (fixed-frame) form:

Conservation of mass (continuity equation):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0. \quad (1.2)$$

Conservation of momentum:

$$\frac{\partial(\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v}) + \nabla P = \rho \mathbf{f}, \quad (1.3)$$

where $\mathbf{f}$ is the specific body force (gravitational acceleration $-\nabla\Phi$ in our case) and $\mathbf{v} \otimes \mathbf{v}$ is the dyadic product (momentum flux tensor).

Conservation of energy:

$$\frac{\partial e}{\partial t} + \nabla \cdot [(e + P)\mathbf{v}] = \rho \mathbf{f} \cdot \mathbf{v} + \mathcal{L}, \quad (1.4)$$

where $e = \rho u + \frac{1}{2}\rho v^2$ is the total energy density, u the specific internal energy per unit mass, and $\mathcal{L}$ collects additional source/sink terms (radiative cooling, heating).

Equation of state. The system of equations (1.2)–(1.4) is not closed without a relation between P , ρ , and u . For an ideal gas:

$$P = (\gamma - 1)\rho u = \frac{\rho k_B T}{\mu m_p}, \quad (1.5)$$

where γ is the adiabatic index ($5/3$ for monatomic gas, $7/5$ for diatomic), μ the mean molecular weight, and m_p the proton mass. The adiabatic sound speed is

$$c_s = \sqrt{\frac{\gamma P}{\rho}} = \sqrt{\frac{\gamma k_B T}{\mu m_p}}. \quad (1.6)$$

Entropy. For adiabatic flow (no heating or cooling), entropy is conserved along streamlines. It is convenient to define the *entropic function*

$$A = \frac{P}{\rho^\gamma} = (\gamma - 1)u\rho^{1-\gamma}, \quad (1.7)$$

which is conserved along fluid trajectories: $DA/Dt = 0$ in the absence of shocks and dissipation. GADGET-2/OpenGADGET3 optionally evolves A rather than u as the thermodynamic variable, guaranteeing exact entropy conservation in adiabatic regions (Springel and Hernquist, 2002).

1.1.3 Lagrangian form

The Lagrangian (material) derivative following a fluid element, $D/Dt = \partial/\partial t + \mathbf{v} \cdot \nabla$, simplifies the equations considerably:

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v}, \quad (1.8)$$

$$\frac{D\mathbf{v}}{Dt} = -\frac{\nabla P}{\rho} + \mathbf{f}, \quad (1.9)$$

$$\frac{Du}{Dt} = -\frac{P}{\rho} \nabla \cdot \mathbf{v} + \frac{\mathcal{L}}{\rho}. \quad (1.10)$$

SPH solves these Lagrangian equations directly, representing fluid elements as discrete particles that move with the local flow velocity.

1.1.4 Conservation laws and their physical content

The Euler equations embody three fundamental conservation laws that carry deep physical content:

Momentum conservation. Equation (1.3) in integral form over a control volume V bounded by surface ∂V :

$$\frac{d}{dt} \int_V \rho \mathbf{v} dV = - \oint_{\partial V} (\rho \mathbf{v} \otimes \mathbf{v} + P \mathbf{I}) \cdot \hat{\mathbf{n}} dS + \int_V \rho \mathbf{f} dV. \quad (1.11)$$

The change in momentum equals the flux of momentum through the boundary plus the body force. In the absence of external forces and across a periodic box, total momentum is strictly conserved.

Energy conservation. Similarly, the total energy (kinetic + internal + potential) is conserved for an isolated system. This constraint provides a powerful diagnostic: if E_{tot} drifts over time in a simulation without any energy source or sink, it signals a timestep accuracy problem or a coding error.

Kelvin's circulation theorem. For an inviscid, barotropic fluid ($P = P(\rho)$ only), the vorticity $\boldsymbol{\omega} = \nabla \times \mathbf{v}$ is frozen into the fluid: fluid elements that start irrotational remain so. This theorem breaks down at shocks and in the presence of baroclinic effects ($\nabla P \times \nabla \rho \neq 0$), which are the primary sources of vorticity generation in astrophysical flows.

1.2 Shock Waves and Discontinuities

Shocks are ubiquitous in astrophysics. They form wherever a supersonic flow encounters an obstacle or another flow, converting kinetic energy into thermal energy irreversibly. Galaxy cluster mergers, supernova remnants, stellar winds colliding with the ISM, galactic outflows — all involve strong shocks. Understanding shock physics is essential for interpreting simulation output.

1.2.1 Jump conditions (Rankine–Hugoniot relations)

At a shock front, the Euler equations break down in their differential form (the fields are discontinuous). Their integral form, however, must still hold across the discontinuity. For

a shock front moving at speed v_s into a stationary medium (subscript 1 = upstream, 2 = downstream), the *Rankine–Hugoniot* (RH) jump conditions are:

$$\rho_1 v_1 = \rho_2 v_2 \quad (\text{mass flux}), \quad (1.12)$$

$$\rho_1 v_1^2 + P_1 = \rho_2 v_2^2 + P_2 \quad (\text{momentum flux}), \quad (1.13)$$

$$\frac{v_1^2}{2} + \frac{\gamma P_1}{(\gamma - 1)\rho_1} = \frac{v_2^2}{2} + \frac{\gamma P_2}{(\gamma - 1)\rho_2} \quad (\text{energy flux}), \quad (1.14)$$

where velocities are measured in the shock rest frame ($v_1 = v_s$). Defining the shock Mach number $\mathcal{M} = v_s/c_{s,1}$, the RH conditions give:

$$\frac{\rho_2}{\rho_1} = \frac{(\gamma + 1)\mathcal{M}^2}{(\gamma - 1)\mathcal{M}^2 + 2}, \quad (1.15)$$

$$\frac{P_2}{P_1} = \frac{2\gamma\mathcal{M}^2 - (\gamma - 1)}{\gamma + 1}, \quad (1.16)$$

$$\frac{T_2}{T_1} = \frac{[2\gamma\mathcal{M}^2 - (\gamma - 1)][(\gamma - 1)\mathcal{M}^2 + 2]}{(\gamma + 1)^2\mathcal{M}^2}. \quad (1.17)$$

In the *strong shock limit* $\mathcal{M} \gg 1$:

$$\frac{\rho_2}{\rho_1} \rightarrow \frac{\gamma + 1}{\gamma - 1} = 4 \quad (\gamma = 5/3), \quad T_2 \rightarrow \frac{3\mu m_p}{16k_B} v_s^2. \quad (1.18)$$

For a supernova blast wave with $v_s \sim 10^4$ km/s, this gives $T_2 \sim 10^{10}$ K — consistent with X-ray observations of young supernova remnants.

The density jump saturates at $(\gamma + 1)/(\gamma - 1) = 4$: no matter how strong the shock, a single shock cannot compress the gas by more than a factor of 4 (for $\gamma = 5/3$). This is a fundamental constraint with important consequences for structure formation.

1.2.2 The Riemann problem

The *Riemann problem* consists of two uniform half-spaces separated at $x = 0$ by a contact discontinuity, with different densities and pressures but zero initial velocities:

$$(\rho, P, v) = \begin{cases} (\rho_L, P_L, 0) & x < 0, \\ (\rho_R, P_R, 0) & x > 0. \end{cases} \quad (1.19)$$

The self-similar solution ($x/t = \text{const}$) breaks into five regions:

1. **Left undisturbed state:** $(\rho_L, P_L, 0)$.
2. **Left rarefaction fan:** a smooth continuous wave expanding to the left at $-c_{s,L}$. The fluid accelerates and the density drops.
3. **Left post-contact state:** uniform pressure P^* and velocity v^* , with density ρ_{L^*} .
4. **Right post-contact state:** same P^* and v^* , with density ρ_{R^*} . A contact discontinuity separates states 3 and 4 at $x = v^*t$.
5. **Right undisturbed state:** $(\rho_R, P_R, 0)$.

The contact discontinuity pressure P^* satisfies the implicit equation

$$f(P^*) = f_L(P^*) + f_R(P^*) + (v_R - v_L) = 0, \quad (1.20)$$

where for a shock ($P^* > P_K$, $K = L$ or R):

$$f_K(P^*) = (P^* - P_K) \left[\frac{A_K}{P^* + B_K} \right]^{1/2}, \quad A_K = \frac{2}{(\gamma + 1)\rho_K}, \quad B_K = \frac{(\gamma - 1)P_K}{\gamma + 1}, \quad (1.21)$$

and for a rarefaction ($P^* \leq P_K$):

$$f_K(P^*) = \frac{2c_{s,K}}{\gamma - 1} \left[\left(\frac{P^*}{P_K} \right)^{(\gamma-1)/(2\gamma)} - 1 \right]. \quad (1.22)$$

Equation (1.20) is solved iteratively (e.g. by Newton–Raphson or bisection), starting from an initial guess. Tutorial T4 (Chapter 9) builds the exact Riemann solver and overlays it on the OpenGADGET3 simulation.

1.2.3 The Sedov–Taylor blast wave

A point explosion of energy E_0 in a uniform medium of density ρ_0 produces a spherically symmetric strong shock (Sedov–Taylor solution). Self-similarity arguments require the shock radius to scale as

$$R_s(t) = \left(\frac{E_0}{\alpha \rho_0} \right)^{1/5} t^{2/5}, \quad (1.23)$$

where $\alpha \approx 0.490$ for $\gamma = 5/3$ (Sedov, 1959). The shock velocity decreases as $v_s \propto t^{-3/5}$ and the temperature at the shock front as $T \propto v_s^2 \propto t^{-6/5}$. Tutorial T7a reproduces this scaling directly.

1.2.4 Virialization

When a self-gravitating gas cloud collapses, kinetic, potential, and thermal energies adjust according to the *virial theorem*. For a system in equilibrium:

$$2E_{\text{kin}} + E_{\text{pot}} + 3(\gamma - 1)E_{\text{int}} = 0. \quad (1.24)$$

This is a powerful diagnostic: in the Evrard collapse (Tutorial T5), the simulation should approach this balance at late times. If it does not, either the timestep is too large (insufficient time resolution) or the artificial viscosity is too small (shocks are not properly captured).

1.2.5 Sound waves

Small-amplitude perturbations in a uniform medium propagate as sound waves. Writing $\rho = \rho_0 + \delta\rho$ and $\mathbf{v} = \delta\mathbf{v}$ with $|\delta\rho/\rho_0| \ll 1$ and $|\delta v/c_s| \ll 1$, and linearising equations (1.8)–(1.10):

$$\frac{\partial^2 \delta\rho}{\partial t^2} = c_s^2 \nabla^2 \delta\rho. \quad (1.25)$$

Plane-wave solutions $\delta\rho \propto e^{i(\mathbf{k}\cdot\mathbf{r}-\omega t)}$ satisfy the dispersion relation $\omega = c_s|\mathbf{k}|$: sound propagates isotropically at speed c_s , with no dispersion. Tutorial T6 measures c_s directly from the simulation and compares it to the analytical value.

The sound crossing time across a box of size L is $t_{\text{cross}} = L/c_s$. For a stable simulation, the timestep Δt must satisfy the *Courant–Friedrichs–Lewy* (CFL) condition: $\Delta t < C_{\text{CFL}} h/v_{\text{sig}}$, where h is the smoothing length and v_{sig} the signal velocity. In OpenGADGET3, the CFL factor is `CourantFac` (default 0.15).

1.3 Magnetohydrodynamics

When a conducting plasma carries an embedded magnetic field $\mathbf{B}$, the electromagnetic Lorentz force modifies the fluid equations. The *ideal* MHD equations (infinite conductivity, no

resistivity) are:

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v}, \quad (1.26)$$

$$\frac{D\mathbf{v}}{Dt} = -\frac{\nabla P_{\text{tot}}}{\rho} + \frac{(\mathbf{B} \cdot \nabla)\mathbf{B}}{4\pi\rho} - \nabla\Phi, \quad (1.27)$$

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{v} \times \mathbf{B}), \quad (1.28)$$

$$\nabla \cdot \mathbf{B} = 0, \quad (1.29)$$

where $P_{\text{tot}} = P + B^2/(8\pi)$ is the total (thermal + magnetic) pressure and the second term in equation (1.27) is the magnetic tension. The induction equation (1.28) expresses *flux freezing*: magnetic field lines move with the fluid.

1.3.1 MHD wave modes

Three types of wave propagate in a magnetised plasma, with speeds that depend on the direction of propagation relative to $\mathbf{B}$:

Alfvén waves. Transverse, incompressible oscillations along field lines. Speed: $v_A = B/\sqrt{4\pi\rho}$ (Alfvén velocity), independent of propagation direction along $\mathbf{B}$. They carry angular momentum but no density perturbation.

Fast magnetosonic waves. Compressional waves. Speed: $v_f = \sqrt{c_s^2 + v_A^2}$ propagating perpendicular to $\mathbf{B}$; this is the maximum propagation speed in any direction. In galaxy clusters, $v_A \sim 10^2$ km/s and $c_s \sim 10^3$ km/s, so the fast wave is close to purely acoustic.

Slow magnetosonic waves. Also compressional, but slower than the fast wave. Speed: $v_s^{\text{slow}} \leq \min(c_s, v_A)$ and zero propagating perpendicular to $\mathbf{B}$.

1.3.2 The $\nabla \cdot \mathbf{B} = 0$ constraint

Equation (1.29) is not a dynamical equation but a constraint. It is satisfied by the initial conditions (magnetic monopoles do not exist) and the induction equation preserves it analytically. Numerically, however, truncation errors inevitably generate non-zero $\nabla \cdot \mathbf{B}$. This produces non-physical forces and can cause the simulation to blow up.

OpenGADGET3 addresses this with Powell–Tricco *hyperbolic divergence cleaning* (DIVBCLEANING_TRICCO): a cleaning field ψ is evolved together with $\mathbf{B}$ to actively damp divergence errors, propagating them away at a finite speed (Tricco et al., 2016):

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{v} \times \mathbf{B}) - \nabla\psi, \quad (1.30)$$

$$\frac{\partial \psi}{\partial t} = -c_h^2 \nabla \cdot \mathbf{B} - \frac{\psi}{\tau}, \quad (1.31)$$

where c_h is the cleaning speed and τ a damping timescale.

1.4 Transport Phenomena: Viscosity and Conduction

The ideal Euler equations ignore two physical transport processes that can be important in astrophysical plasmas: thermal conduction (heat diffusion along temperature gradients) and viscosity (momentum diffusion along velocity gradients). OpenGADGET3 implements both as optional modules.

1.4.1 Thermal conduction

In a plasma, heat is carried primarily by electrons along temperature gradients. For a fully ionised plasma, the *Spitzer* (or Braginskii) thermal conductivity is (Spitzer, 1962):

$$\kappa_{\text{Sp}} = 1.84 \times 10^{-5} \frac{T^{5/2}}{\ln \Lambda} \text{ erg s}^{-1} \text{ cm}^{-1} \text{ K}^{-1}, \quad (1.32)$$

where $\ln \Lambda \approx 37$ is the Coulomb logarithm. The $T^{5/2}$ dependence makes conduction strongly temperature-dependent: hot gas conducts much more efficiently than cold gas.

The heat flux is $\mathbf{q} = -\kappa \nabla T$, and the internal energy equation gains a diffusion term:

$$\rho c_v \frac{DT}{Dt} = \nabla \cdot (\kappa \nabla T), \quad (1.33)$$

where $c_v = k_B / [(\gamma - 1) \mu m_p]$ is the specific heat at constant volume. For constant κ , this is the standard heat equation with diffusion coefficient $D = \kappa / (\rho c_v)$. Its analytical solution for a step-function initial condition is used in Tutorial T8a.

In practice, magnetic fields suppress conduction perpendicular to field lines, reducing the effective conductivity by factors of ~ 10 – 100 . OpenGADGET3 accounts for this via the `ConductionEfficiency` parameter, which sets the fraction of the Spitzer value to use.

1.4.2 Physical viscosity

The Navier–Stokes equations augment the Euler momentum equation with a viscous stress tensor:

$$\frac{D\mathbf{v}}{Dt} = -\frac{\nabla P}{\rho} + \frac{1}{\rho} \nabla \cdot \boldsymbol{\sigma} + \mathbf{g}, \quad (1.34)$$

where the viscous stress tensor for a Newtonian fluid is

$$\sigma_{ij} = \eta \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} - \frac{2}{3} \delta_{ij} \nabla \cdot \mathbf{v} \right) + \zeta \delta_{ij} \nabla \cdot \mathbf{v}, \quad (1.35)$$

with η the dynamic (shear) viscosity and ζ the bulk viscosity (negligible for monatomic ideal gases). The kinematic viscosity $\nu = \eta / \rho$ controls the diffusion of momentum.

For the Spitzer shear viscosity of a fully ionised plasma:

$$\eta_{\text{Sp}} = 2.21 \times 10^{-15} \frac{(k_B T)^{5/2}}{(\ln \Lambda) m_p^{1/2} e^4} \text{ g cm}^{-1} \text{ s}^{-1}, \quad (1.36)$$

again with a $T^{5/2}$ dependence. In galaxy cluster plasma at $T \sim 10^8$ K, $\eta_{\text{Sp}} \sim 1 \text{ g cm}^{-1} \text{ s}^{-1}$, corresponding to a Reynolds number $\text{Re} = \rho v L / \eta \sim 10^2$ – 10^3 for merger velocities $v \sim 10^3 \text{ km/s}$. At these Reynolds numbers, viscosity can significantly suppress the Kelvin–Helmholtz instability (KHI) at cluster interfaces (Roediger et al., 2013). Tutorials T9a and T9b explore these effects.

1.5 Gravitational Collapse and the N-body Problem

1.5.1 The free-fall time and structure formation

As derived in Chapter 6 (Tutorial T1), a uniform pressureless sphere of density ρ_0 collapses in the *free-fall time*:

$$t_{\text{ff}} = \sqrt{\frac{3\pi}{32G\rho_0}}. \quad (1.37)$$

This single formula encapsulates much of the physics of cosmic structure formation. Since $t_{\text{ff}} \propto \rho_0^{-1/2}$, denser regions collapse faster. In a hierarchical universe where initial density fluctuations grow by gravitational instability, small overdense regions (proto-galaxies) collapse first, then merge into larger structures (galaxy groups, clusters). The characteristic density of a virialised halo scales as $\rho_{\text{vir}} \propto (1+z)^3$, giving $t_{\text{ff}} \propto (1+z)^{-3/2}$ and confirming that high-redshift structures collapse more quickly.

1.5.2 The collisionless Boltzmann equation

The evolution of a collisionless fluid (dark matter, or stars in a galaxy) cannot be described by the Euler equations. Instead, the distribution function $f(\mathbf{r}, \mathbf{v}, t)$ evolves according to the *collisionless Boltzmann equation* (CBE):

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f - \nabla \Phi \cdot \frac{\partial f}{\partial \mathbf{v}} = 0, \quad (1.38)$$

where Φ is the gravitational potential determined self-consistently by $\nabla^2 \Phi = 4\pi G \int f d^3v$. The CBE is a 6-dimensional PDE; it is far too expensive to solve directly for large systems.

1.5.3 The N-body approximation

The N-body method represents the distribution function f as a sum of N point masses (“superparticles”, each representing $\sim 10^6$ to 10^{10} actual particles):

$$f(\mathbf{r}, \mathbf{v}, t) = \sum_{i=1}^N m_i \delta^{(3)}(\mathbf{r} - \mathbf{r}_i(t)) \delta^{(3)}(\mathbf{v} - \mathbf{v}_i(t)). \quad (1.39)$$

Each superparticle follows its trajectory under the mutual gravitational field of all others. The resulting equations of motion are:

$$\dot{\mathbf{r}}_i = \mathbf{v}_i, \quad (1.40)$$

$$\dot{\mathbf{v}}_i = - \sum_{j \neq i} \frac{G m_j (\mathbf{r}_i - \mathbf{r}_j)}{(|\mathbf{r}_i - \mathbf{r}_j|^2 + \epsilon^2)^{3/2}}, \quad (1.41)$$

where ϵ is the *gravitational softening length* that prevents the divergence of the force at short range (see §1.7.5).

Direct summation of equation (1.41) scales as $\mathcal{O}(N^2)$ and is prohibitive for $N \gtrsim 10^5$. The tree and TreePM algorithms (Section 1.7) reduce this to $\mathcal{O}(N \log N)$.

1.5.4 Two-body relaxation and the collisionless limit

A real dark matter halo contains $\sim 10^{65}$ particles; simulations use $\sim 10^6$ – 10^{10} superparticles. Each superparticle is much more massive than a real dark matter particle. The artificial *two-body relaxation* (gravitational scattering between superparticles) occurs on a timescale:

$$t_{\text{relax}} \approx \frac{N}{8 \ln N} t_{\text{cross}}, \quad (1.42)$$

where t_{cross} is the crossing time. For $N = 10^6$ and $t_{\text{cross}} \sim 1$ Gyr, $t_{\text{relax}} \sim 10^5$ Gyr — much longer than the Hubble time. Gravitational softening further suppresses two-body relaxation by preventing close encounters.

1.6 Smoothed Particle Hydrodynamics

SPH was introduced independently by Lucy (1977) and Gingold and Monaghan (1977) for the simulation of fission in rotating protostars. Since then it has become one of the two dominant numerical methods in computational astrophysics (the other being Adaptive Mesh Refinement, AMR). The key idea is to represent the fluid as a set of N particles that move with the local flow velocity, each carrying mass, position, velocity, and thermodynamic properties. Continuous fields are recovered by kernel-weighted interpolation over neighbouring particles.

1.6.1 Kernel density estimation

The fundamental operation in SPH is the *kernel interpolation* of a field $A(\mathbf{r})$. Given particle positions $\{\mathbf{r}_j\}$ and values $\{A_j\}$, the interpolant is:

$$\langle A(\mathbf{r}) \rangle = \int A(\mathbf{r}') W(|\mathbf{r} - \mathbf{r}'|, h) d^3 r' \approx \sum_j \frac{m_j}{\rho_j} A_j W(|\mathbf{r} - \mathbf{r}_j|, h), \quad (1.43)$$

where $W(r, h)$ is the *smoothing kernel* (a symmetric, compact function) and h the *smoothing length*. The kernel must satisfy the *partition of unity* condition:

$$\int W(|\mathbf{r}|, h) d^3 r = 1, \quad (1.44)$$

and must reduce to a delta function as $h \rightarrow 0$:

$$\lim_{h \rightarrow 0} W(|\mathbf{r}|, h) = \delta^{(3)}(\mathbf{r}). \quad (1.45)$$

Gradients are computed by differentiating the interpolant: $\nabla A(\mathbf{r}) \approx \sum_j (m_j/\rho_j) A_j \nabla W$.

The density at particle i is estimated as:

$$\rho_i = \sum_j m_j W(|\mathbf{r}_i - \mathbf{r}_j|, h_i). \quad (1.46)$$

This automatically gives higher density where particles cluster — a key advantage of SPH for problems with large density contrasts.

1.6.2 The smoothing length and adaptive resolution

The smoothing length h_i is chosen adaptively so that each particle has approximately N_{ngb} neighbours within its kernel support (radius $2h_i$ for most kernels). The condition:

$$\frac{4\pi}{3} (2h_i)^3 n_i = N_{\text{ngb}} \quad \Rightarrow \quad h_i = \left(\frac{3N_{\text{ngb}}}{32\pi n_i} \right)^{1/3}, \quad (1.47)$$

where $n_i = \rho_i/m_i$ is the local number density. In OpenGADGET3, this is satisfied iteratively: h_i is adjusted until the actual neighbour count is within `MaxNumNgbDeviation` of `DesNumNgb`. The smoothing length therefore decreases in dense regions and increases in voids, providing *automatic adaptive resolution*: denser regions are resolved more finely with no user intervention.

This is a fundamental advantage of SPH over fixed-grid methods: the resolution follows the mass, not the volume. In a cosmological simulation, the dense interiors of galaxy clusters are automatically resolved at ~ 100 times higher spatial resolution than the voids.

1.6.3 Smoothing kernels

The shape of the kernel determines the accuracy and stability of the density estimate. A good kernel should be:

- **Compact:** $W(r, h) = 0$ for $r > r_{\text{cut}}$, so that only nearby particles contribute (computational efficiency).
- **Normalised:** $\int W \, d^3r = 1$.
- **Smooth:** at least C^2 continuous for SPH equations of motion to be well-defined; higher smoothness gives more accurate gradient estimates.
- **Positive definite:** $W \geq 0$ everywhere to avoid negative density estimates.

Cubic spline (M4 kernel). The most widely used SPH kernel until recently, defined as

$$W(r, h) = \frac{\sigma}{h^3} \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & 0 \leq q \leq 1, \\ \frac{1}{4}(2 - q)^3 & 1 < q \leq 2, \\ 0 & q > 2, \end{cases} \quad q = r/h, \quad (1.48)$$

where $\sigma = 8/\pi$ in 3D. This is the `STANDARD_KERNEL` in OpenGADGET3 . It is C^2 continuous and compact with support radius $2h$.

Wendland kernels. The Wendland C^{2k} kernels (Dehnen and Aly, 2012) are strictly positive definite, guaranteeing that the SPH density estimate matrix is positive definite and avoiding the *pairing instability* (artificial clustering of particle pairs) that afflicts higher-neighbour-count cubic spline simulations. The Wendland C^4 kernel in 3D is:

$$W(r, h) = \frac{\sigma_4}{h^3} \left(1 - \frac{r}{2h}\right)_+^6 \left(1 + 3\frac{r}{h} + \frac{35}{12}\frac{r^2}{h^2}\right), \quad (1.49)$$

where $(x)_+ = \max(0, x)$ and $\sigma_4 = 495/(32\pi)$. This is the recommended choice for production runs. It achieves similar accuracy to the cubic spline but with more neighbours (typically $N_{\text{ngb}} \sim 200$ in 3D), substantially reducing density noise.

Tutorial T3 provides a systematic comparison of kernels across different neighbour numbers and initial particle distributions.

1.6.4 Particle distributions and numerical noise

The bias and variance of the SPH density estimate depend not only on the kernel but on the regularity of the particle distribution. Three canonical distributions are used in practice:

Cubic lattice. Particles placed on a regular 3D Cartesian grid. The simplest to generate, but the regularity introduces anisotropic density errors: the estimated density is systematically higher along grid axes and lower along diagonals. Suitable for simple tests but not for production simulations.

Hexagonal close-packed (HCP). More isotropic than the cubic lattice. Alternating layers are shifted by $(0.5, 0.5, 0)$ in units of the lattice spacing, giving the densest possible regular packing. Density estimates are more isotropic, though lattice-related artefacts remain.

Glass distribution. Obtained by evolving a random particle distribution under a reversed-sign gravitational force until all particles reach a quasi-equilibrium. The resulting configuration is disordered but highly isotropic, giving the minimum possible density bias for a given number of particles. This is the recommended initial particle distribution for production simulations. Pre-computed glass files are provided with the tutorials.

Dehnen and Aly (2012) showed quantitatively that the Wendland C^4 kernel with a glass particle distribution gives density estimates accurate to $< 0.1\%$ with $N_{\text{ngb}} = 100$, while the cubic spline with a cubic lattice can have errors of $\sim 5\%$ with the same neighbour count. Tutorial T3 reproduces these comparisons.

1.6.5 The SPH Lagrangian and equations of motion

A major strength of SPH is that the equations of motion can be derived *rigorously* from a discrete Lagrangian, guaranteeing that the discretised system conserves energy, linear momentum, angular momentum, and entropy simultaneously (Springel and Hernquist, 2002). This is unlike many finite-difference methods where conservation must be verified separately.

The SPH Lagrangian for an ideal gas is:

$$\mathcal{L} = \sum_i m_i \left(\frac{v_i^2}{2} - u_i(\rho_i, A_i) \right), \quad (1.50)$$

where u_i is the specific internal energy, and ρ_i is the SPH density estimate (1.46) which depends on all particle positions through h_i . Applying the Euler-Lagrange equations $d(\partial\mathcal{L}/\partial\dot{\mathbf{r}}_i)/dt = \partial\mathcal{L}/\partial\mathbf{r}_i$ yields the SPH equations of motion (Springel and Hernquist, 2002):

$$\frac{D\mathbf{v}_i}{Dt} = - \sum_j m_j \left[\frac{P_i}{\Omega_i \rho_i^2} \nabla_i W_{ij}(h_i) + \frac{P_j}{\Omega_j \rho_j^2} \nabla_i W_{ij}(h_j) \right] + \mathbf{g}_i + \left(\frac{D\mathbf{v}_i}{Dt} \right)_{\text{visc}}, \quad (1.51)$$

where $W_{ij}(h) \equiv W(|\mathbf{r}_i - \mathbf{r}_j|, h)$ and the *grad-h correction term* is:

$$\Omega_i = 1 - \frac{\partial h_i}{\partial \rho_i} \sum_j m_j \frac{\partial W_{ij}(h_i)}{\partial h_i}. \quad (1.52)$$

The Ω_i term, absent in older SPH formulations, corrects for the spatial variation of the smoothing length. Without it, entropy conservation is violated whenever h varies in space (Springel and Hernquist, 2002).

The internal energy equation:

$$\frac{Du_i}{Dt} = \frac{P_i}{\Omega_i \rho_i^2} \sum_j m_j (\mathbf{v}_i - \mathbf{v}_j) \cdot \nabla_i W_{ij}(h_i) + \left(\frac{Du_i}{Dt} \right)_{\text{visc}} + \left(\frac{Du_i}{Dt} \right)_{\text{cond}} + \Lambda_i / \rho_i, \quad (1.53)$$

where Λ_i is the net cooling/heating rate per unit volume.

The symmetry of equations (1.51) in particle indices ensures exact conservation of linear and angular momentum, even on a particle-by-particle basis (Newton's third law is satisfied exactly).

1.6.6 Artificial viscosity

The SPH equations of motion (1.51) describe an inviscid fluid. They cannot spontaneously capture shocks: two particle streams approaching one another would simply interpenetrate and pass through each other. A *dissipative term* must be added to produce the correct shock structure and entropy generation.

The standard Monaghan–Gingold artificial viscosity adds a viscous pressure Π_{ij} to particle pairs where $(\mathbf{v}_i - \mathbf{v}_j) \cdot (\mathbf{r}_i - \mathbf{r}_j) < 0$ (approaching particles):

$$\Pi_{ij} = \frac{-\alpha \bar{c}_{ij} \mu_{ij} + \beta \mu_{ij}^2}{\bar{\rho}_{ij}}, \quad \mu_{ij} = \frac{h_{ij} (\mathbf{v}_{ij} \cdot \mathbf{r}_{ij})}{|\mathbf{r}_{ij}|^2 + \epsilon h_{ij}^2}, \quad (1.54)$$

where $\bar{c}_{ij}$, $\bar{\rho}_{ij}$, h_{ij} are mean sound speed, density, and smoothing length between particles i and j ; the linear term (α) captures shocks and the quadratic term (β) prevents particle interpenetration in very strong shocks. In OpenGADGET3, α is set by `ArtBulkViscConst` (typically 0.8 for moderate shocks, up to 1.5 for strong shocks), and $\beta = 2\alpha$.

The Balsara switch. The artificial viscosity should be active in shocks ($\nabla \cdot \mathbf{v} < 0$, converging flow) but suppressed in shear flows where it would produce spurious angular momentum transfer. The Balsara limiter multiplies Π_{ij} by a factor $f_i \in [0, 1]$:

$$f_i = \frac{|\nabla \cdot \mathbf{v}|_i}{|\nabla \cdot \mathbf{v}|_i + |\nabla \times \mathbf{v}|_i + 10^{-4} c_s/h}, \quad (1.55)$$

which is close to 1 in converging flows and close to 0 in rotational flows. For MHD simulations, the Balsara switch can interfere with MHD dissipation; it is disabled with `NO_BALSARA_LIMITER`.

Time-dependent viscosity. Morris and Monaghan (1997) proposed allowing α to evolve in time for each particle according to a source-and-decay equation:

$$\frac{d\alpha_i}{dt} = -\frac{\alpha_i - \alpha_{\min}}{\tau_i} + S_i, \quad S_i = \max(0, -\nabla \cdot \mathbf{v}_i) (\alpha_{\max} - \alpha_i). \quad (1.56)$$

Here α grows in regions of convergence ($S_i > 0$, near shocks) and decays exponentially towards $\alpha_{\min}$ in quiet regions, with decay time $\tau_i = h_i/(c_{s,i}\ell)$ where $\ell \approx 0.2$. This minimises artificial viscosity in shear flows while maintaining adequate shock capture. Enabled with `TIME_DEP_ART_VISC=2`.

1.6.7 Artificial conductivity

At contact discontinuities (density jumps with no velocity jump), standard SPH is unable to transport thermal energy across the interface. This produces a spurious pressure discontinuity — visible as a “pressure blip” in Riemann problem tests. Price (2008) introduced an *artificial conductivity* term:

$$\left(\frac{Du_i}{Dt}\right)_{\text{cond}} = \alpha_u \sum_j m_j \frac{v_{\text{sig}}^{ij} (u_j - u_i)}{\rho_{ij}} \hat{\mathbf{r}}_{ij} \cdot \nabla_i W_{ij}, \quad (1.57)$$

where $v_{\text{sig}}^{ij} = \sqrt{|P_i - P_j|/\bar{\rho}_{ij}}$ is a signal velocity that activates the conductivity only at pressure discontinuities. Enable with `ARTIFICIAL_CONDUCTIVITY`. This is demonstrated in Tutorial T4.

1.6.8 The Meshless Finite Mass (MFM) method

A well-documented weakness of SPH is its poor representation of Kelvin–Helmholtz instabilities at contact discontinuities. The origin is a spurious “surface tension” arising from the zeroth-order error in the SPH density estimate at density jumps.

The *Meshless Finite Mass* (MFM) method (Hopkins, 2015) addresses this by computing inter-particle fluxes using a Godunov-type Riemann solver, similar in spirit to Eulerian finite-volume methods but with Lagrangian particles. The mass of each particle is strictly conserved

(unlike smoothed-particle versions of the moving-mesh scheme). The result eliminates artificial viscosity, captures KHI correctly, and converges at higher order than standard SPH for smooth flows.

The trade-off: MFM is more expensive than SPH (due to the Riemann solve), and certain conservation properties of SPH (exact angular momentum) are not guaranteed. Enable with `USE_MESHLESS_FINITE_MASS`. Tutorial T9b demonstrates the difference between SPH and MFM in a jet-mixing experiment.

1.6.9 SPH versus Eulerian methods: a brief comparison

The two main alternatives in computational astrophysics are SPH (Lagrangian) and Adaptive Mesh Refinement (AMR, Eulerian). Their relative strengths and weaknesses are:

Table 1.1: SPH versus AMR: key properties for astrophysical applications.

Property	SPH (OpenGadget3)	AMR (e.g. FLASH, RAMSES)
Resolution strategy	Follows mass (Lagrangian)	Follows density (adaptive grid)
Conservation	Exact (energy, momentum, angular momentum)	Volume-weighted
Galilean invariance	Yes (exact)	Approximate
Shock capturing	Artificial viscosity	Godunov solvers
Kelvin–Helmholtz instability	Suppressed (SPH); correct (MFM)	Correct
Vacuum regions	No particles $\rightarrow$ no force	Grid exists everywhere
Coupling to N-body gravity	Exact and efficient	Requires interpolation

OpenGADGET3 is particularly well suited for problems with large density contrasts (cosmological structure formation, galaxy formation) and for problems where self-gravity and gas dynamics must be coupled accurately and efficiently.

1.7 Gravitational Algorithms

1.7.1 Direct summation

The most straightforward approach to the N-body problem is to compute the force on every particle by summing over all other particles:

$$\mathbf{g}_i = -G \sum_{j \neq i} m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{(|\mathbf{r}_i - \mathbf{r}_j|^2 + \epsilon^2)^{3/2}}. \quad (1.58)$$

This is exact but scales as $\mathcal{O}(N^2)$: for $N = 10^6$ particles, it requires 10^{12} operations per timestep. Impractical for any modern astrophysical simulation.

1.7.2 The Barnes–Hut tree algorithm

The Barnes–Hut (BH) algorithm (Barnes and Hut, 1986) reduces direct summation to $\mathcal{O}(N \log N)$ by approximating the force from distant groups of particles using their multipole moments.

The simulation volume is recursively subdivided into an *oct-tree*: the root node is the full box, each node is split into 8 equal sub-cubes (octants) until each leaf node contains at most one

particle. Each internal node stores the total mass and centre of mass of all particles in its subtree (and optionally higher multipole moments).

The force on particle i is computed by *tree walking*: starting from the root, descend into a sub-tree only if the *opening criterion* is met:

$$\frac{l}{r} > \theta_c, \quad (1.59)$$

where l is the size of the node, r the distance from particle i to the node's centre of mass, and θ_c is the *opening angle* parameter. If the criterion is *not* met, the node is treated as a single mass at its centre of mass (monopole approximation) and the tree walk does not descend further. In OpenGADGET3, the opening criterion is controlled by `ErrTolForceAcc`: a node is opened when the monopole force error would exceed this tolerance.

The typical number of tree nodes evaluated per particle is $\mathcal{O}(\log N)$, giving an overall scaling $\mathcal{O}(N \log N)$. The force error decreases as θ_c^2 for the monopole approximation; with quadrupole corrections (included in OpenGADGET3) the error decreases as θ_c^4 .

1.7.3 The Particle-Mesh (PM) method

For long-range gravitational forces in periodic boxes (cosmological simulations), the PM method is more efficient than the tree. The algorithm:

1. Assign particle masses to a regular mesh of N_{mesh}^3 cells using NGP, CIC, or TSC assignment.
2. Fourier-transform the density field: $\hat{\rho}(\mathbf{k}) = \mathcal{F}[\rho(\mathbf{x})]$.
3. Divide by $-k^2$ (Green's function of the Poisson equation): $\hat{\Phi}(\mathbf{k}) = -4\pi G \hat{\rho}(\mathbf{k})/k^2$.
4. Inverse Fourier-transform to obtain $\Phi(\mathbf{x})$.
5. Compute $\mathbf{g} = -\nabla\Phi$ by finite differences.
6. Interpolate $\mathbf{g}$ back to particle positions.

This scales as $\mathcal{O}(N_{\text{mesh}}^3 \log N_{\text{mesh}})$ for the FFT and $\mathcal{O}(N)$ for the assignment/interpolation steps. For $N_{\text{mesh}} \sim N^{1/3}$, the PM method is effectively $\mathcal{O}(N)$. However, it has a force resolution limited to the mesh spacing $\sim L/N_{\text{mesh}}$.

1.7.4 The TreePM hybrid method

The TreePM method (Bagla, 2002; Springel, 2005) combines the accuracy of the tree at short range with the efficiency of PM at long range. The gravitational force is split into two components:

$$\mathbf{g} = \mathbf{g}_{\text{short}}(r < r_s) + \mathbf{g}_{\text{long}}(r > r_s), \quad (1.60)$$

where r_s is the force-split scale (typically a few PM mesh cells). The short-range force is computed with the tree, and the long-range force with PM. Since the PM force varies slowly in space and time, it can be updated less frequently, saving substantial computational cost.

For cosmological simulations, this combination:

- Automatically handles periodic boundary conditions (via PM).
- Provides high force resolution in dense regions (via tree).
- Is well-parallelisable for MPI (both tree and PM scale efficiently).

OpenGADGET3 uses TreePM for all cosmological runs.

1.7.5 Gravitational softening

The N-body force (1.58) diverges as $r \rightarrow 0$. This is unphysical because each “superparticle” represents $\sim 10^6$ – 10^{10} real particles, not a physical mass concentration. Gravitational softening replaces the $1/r^2$ force with a spline-softened version for $r < \epsilon$:

$$|\mathbf{g}| = \frac{Gm}{r^2 + \epsilon^2} \quad (\text{Plummer softening}), \quad (1.61)$$

or (in OpenGADGET3) a cubic spline kernel that is exactly Newtonian for $r > 2.8\epsilon$ and goes to a constant force at $r = 0$. The effect is to make the force equivalent to that of a smooth mass distribution of size $\sim \epsilon$ rather than a point mass.

Choosing the softening: Too small ϵ leads to very short timesteps near close particle pairs (since $\Delta t \propto \sqrt{\epsilon/|\mathbf{g}|}$), making the simulation prohibitively slow. Too large ϵ suppresses structure formation on scales $\lesssim \epsilon$ and artificially heats the system (two-body relaxation). The standard recommendation (Power et al., 2003) is:

$$\epsilon \approx \frac{4 r_{200c}}{N_{200c}^{1/2}}, \quad (1.62)$$

where r_{200c} is the virial radius and N_{200c} the particle count within it. For tutorial simulations, a practical rule is $\epsilon \approx L/(30 N^{1/3})$ where L is the box size.

1.7.6 Time integration

OpenGADGET3 uses the *kick-drift-kick* (KDK) leapfrog integrator, a symplectic second-order scheme:

$$\mathbf{v}_{n+1/2} = \mathbf{v}_n + \frac{\Delta t}{2} \mathbf{g}_n \quad (\text{kick}), \quad (1.63)$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \Delta t \mathbf{v}_{n+1/2} \quad (\text{drift}), \quad (1.64)$$

$$\mathbf{v}_{n+1} = \mathbf{v}_{n+1/2} + \frac{\Delta t}{2} \mathbf{g}_{n+1} \quad (\text{kick}). \quad (1.65)$$

The symplectic property means that the integrator exactly conserves a modified Hamiltonian close to the true one, preventing secular energy drift over long integration times. Figure ?? in the GADGET-2 paper shows that the leapfrog dramatically outperforms 4th-order Runge–Kutta for long-term orbital integrations, despite being only 2nd-order per step.

Each particle carries its own timestep selected from a discretised power-of-two hierarchy $\Delta t = \Delta t_{\max}/2^n$ ($n = 0, 1, 2, \dots$):

$$\Delta t_i = \min \left(C_{\text{CFL}} \frac{h_i}{v_{\text{sig},i}}, \sqrt{\frac{2\eta \epsilon_i}{|\mathbf{g}_i|}}, \Delta t_{\max} \right), \quad (1.66)$$

where $C_{\text{CFL}} = \text{CourantFac}$ (default 0.15), $v_{\text{sig},i} = c_{s,i} + \max_j |\mu_{ij}|$ is the maximum signal velocity with any neighbour, $\eta = \text{ErrTolIntAccuracy} \approx 0.025$, and MaxSizeTimestep provides a hard ceiling.

Particles with short timesteps (e.g. in dense cluster centres) are integrated on their individual short steps, while particles in voids take large steps. This *individual adaptive timestep* scheme can reduce computation by factors of 10–100 compared to a global timestep.

1.7.7 MPI and OpenMP parallelism

OpenGADGET3 is parallelised with a hybrid MPI+OpenMP model.

Domain decomposition (MPI). The simulation volume is divided among MPI ranks using a *space-filling Peano–Hilbert curve*. This curve maps 3D space to a 1D curve such that spatially close regions map to nearby curve positions. Dividing the curve into N_{MPI} equal segments (by particle number) keeps spatially close particles on the same MPI rank, minimising ghost-particle communication for tree walks and SPH neighbour searches.

Shared-memory threads (OpenMP). Within each MPI rank, OpenMP threads parallelise loops over particles (the tree walk, the SPH force computation, the time integration). Thread-level parallelism avoids MPI communication overhead for tasks that fit within a shared memory node.

Load balance. Gravitational tree walks are more expensive in dense regions than in voids. The domain decomposition updates regularly to rebalance the load, monitored in `balance.txt`. High imbalance ($> 20\%$) indicates that the domain decomposition is not keeping up with the evolving density field; reducing `TreeDomainUpdateFrequency` forces more frequent rebalancing.

1.8 Sub-grid Physics

Many astrophysically important processes occur below the resolution limit of cosmological or galaxy-scale simulations. The interstellar medium is multiphase, with cold molecular clouds ($T \sim 10\text{ K}$, $n \sim 10^4\text{ cm}^{-3}$) embedded in a hot diffuse medium ($T \sim 10^6\text{ K}$, $n \sim 10^{-3}\text{ cm}^{-3}$), with structures from AU to kpc scales. Resolving all of this in a simulation that also follows cosmological structure from Mpc scales is impossible. Instead, the *sub-grid* approach models unresolved processes as effective prescriptions at the resolution scale.

1.8.1 Radiative cooling

Gas in thermal contact with a radiation field cools by emitting photons and heats by absorbing them. For a fully ionised plasma, the dominant cooling processes are:

- **Bremsstrahlung** (free-free): $\Lambda_{\text{ff}} \propto n_e^2 T^{1/2}$. Dominant above $\sim 10^7\text{ K}$.
- **Line emission** (collisional excitation/ionisation): dominant at $10^4\text{--}10^7\text{ K}$.
- **Recombination radiation**: important at 10^4 K .

OpenGADGET3 uses the cooling function of [Springel and Hernquist \(2003\)](#), which tabulates the net cooling rate $\Lambda_{\text{net}}(\rho, T, Z)$ for a range of metallicities Z and includes the UV photoionisation background from the `TREECOOL` table. At temperatures below 10^4 K , the UV background suppresses cooling (photoionisation heating exceeds cooling), preventing gas from cooling to very low temperatures except in very dense regions.

The cooling timescale at temperature T and density ρ is:

$$t_{\text{cool}} = \frac{\rho u}{\Lambda_{\text{net}}} = \frac{(\gamma - 1)^{-1} n k_B T}{\Lambda_{\text{net}}(n, T)}, \quad (1.67)$$

where n is the number density. In cluster cores, t_{cool} can be shorter than the Hubble time, producing the “cool-core” phenomenon observed in X-rays.

1.8.2 Star formation

Gas that has cooled below a temperature threshold and exceeded a density threshold is eligible for star formation. OpenGADGET3 implements the [Springel and Hernquist \(2003\)](#) sub-grid

multiphase model: above the threshold density ρ_{th} , gas is modelled as a two-phase medium (cold clouds + hot medium in pressure equilibrium) with a star formation rate:

$$\dot{\rho}_* = \frac{(1 - \beta)\rho_c}{t_*}, \quad t_* = t_0^* \left(\frac{\rho}{\rho_{\text{th}}} \right)^{-1/2}, \quad (1.68)$$

where ρ_c is the cold cloud density, β the mass fraction of supernovae returned immediately to the hot phase, and t_0^* the star formation timescale normalised to reproduce the observed Kennicutt–Schmidt relation (set by `MaxSfrTimescale`).

Star particles are created stochastically: a gas particle spawns a stellar particle of mass $m_{\text{gas}}/N_{\text{gen}}$ (where $N_{\text{gen}} = \text{GENERATIONS}$) when a random number falls below $\dot{\rho}_*/\rho \times \Delta t$. This probabilistic approach reproduces the ensemble-averaged SFR while maintaining discrete particle conservation.

1.8.3 Stellar feedback: galactic winds

Supernovae from massive stars inject energy and momentum into the surrounding ISM. At cosmological simulation resolution, individual supernovae are not resolved; instead, `OpenGADGET3` implements a *kinetic wind model*: newly formed stars probabilistically launch neighbouring gas particles as wind particles at velocity $v_w \propto \sigma$ (local velocity dispersion) with a mass loading $\eta = \dot{M}_w/\dot{M}_* = \text{WindEfficiency}$. Wind particles decouple from the hydrodynamics for a free-travel time proportional to the local sound crossing time, then recouple and thermalise.

The energy budget:

$$\dot{E}_w = \frac{1}{2}\dot{M}_w v_w^2 = \frac{1}{2}\eta\dot{M}_* v_w^2 = \varepsilon_{\text{SN}} \dot{M}_* e_{\text{SN}}, \quad (1.69)$$

where $\varepsilon_{\text{SN}} = \text{WindEnergyFraction}$ and $e_{\text{SN}} \sim 10^{49} \text{ erg } M_{\odot}^{-1}$ is the SN energy per unit stellar mass. Galactic winds are essential for reproducing the observed stellar mass function and star formation history ([Springel and Hernquist, 2003](#)).

1.9 Cosmological Simulations

1.9.1 The expanding universe and co-moving coordinates

In cosmology, the universe expands with scale factor $a(t)$ according to the Friedmann equation:

$$H^2(a) = \left(\frac{\dot{a}}{a} \right)^2 = H_0^2 \left[\Omega_m a^{-3} + \Omega_r a^{-4} + \Omega_{\Lambda} \right], \quad (1.70)$$

where H_0 is the Hubble constant, Ω_m the matter density parameter, Ω_r the radiation density parameter, and Ω_{Λ} the cosmological constant density parameter. The redshift z is related to the scale factor by $a = 1/(1 + z)$.

Physical (proper) coordinates $\mathbf{r}$ are related to co-moving coordinates $\mathbf{x}$ by $\mathbf{r} = a\mathbf{x}$. The peculiar velocity (motion relative to the Hubble flow) is $\mathbf{v} = \dot{\mathbf{r}} - H\mathbf{r} = a\dot{\mathbf{x}}$.

`OpenGADGET3` integrates the equations of motion in co-moving coordinates. Snapshot files store positions in co-moving coordinates and velocities as $a\dot{\mathbf{x}}$ (divided by an additional factor $\sqrt{a}$ in some conventions). Converting to physical quantities requires careful attention to these conventions; see Section 20.4 of the post-processing chapter.

1.9.2 Initial conditions and perturbation theory

Cosmological simulations start at high redshift ($z_i \sim 50\text{--}100$) from initial conditions derived from perturbation theory. The linear density field at z_i is characterised by the *power spectrum* $P(k) = \langle |\delta_k|^2 \rangle$, where δ_k are the Fourier modes of the density contrast $\delta = (\rho - \bar{\rho})/\bar{\rho}$.

The *Zel'dovich approximation* (first-order Lagrangian perturbation theory, 1LPT) gives initial particle displacements from a regular grid:

$$\mathbf{x}_i = \mathbf{q}_i + \mathbf{S}^{(1)}(\mathbf{q}_i), \quad \mathbf{S}^{(1)} = -\nabla\phi, \quad \nabla^2\phi = \delta(\mathbf{q}), \quad (1.71)$$

where $\mathbf{q}_i$ is the grid position and ϕ the displacement potential. Second-order LPT (2LPT) adds a correction:

$$\mathbf{x}_i = \mathbf{q}_i + \mathbf{S}^{(1)} + \mathbf{S}^{(2)}, \quad (1.72)$$

which reduces transient artefacts and allows starting at lower z_i without loss of accuracy. MUSIC implements 2LPT by default.

1.9.3 Large-scale structure observables

The primary observable from cosmological N-body and SPH simulations is the matter *power spectrum*:

$$P(k) = \frac{1}{V} \langle |\tilde{\delta}(k)|^2 \rangle, \quad (1.73)$$

where $\tilde{\delta}(\mathbf{k}) = \int \delta(\mathbf{x}) e^{-i\mathbf{k}\cdot\mathbf{x}} d^3x$ is the Fourier transform of the density contrast. On linear scales ($k < k_{\text{nl}}$), the power spectrum evolves as $P(k, a) = D^2(a) P_0(k)$ where $D(a)$ is the linear growth factor. On non-linear scales, the power spectrum is amplified and its shape changes due to gravitational clustering.

The *two-point correlation function* $\xi(r) = \langle \delta(\mathbf{x})\delta(\mathbf{x} + \mathbf{r}) \rangle$ is the Fourier transform of $P(k)$:

$$\xi(r) = \frac{1}{2\pi^2} \int_0^\infty P(k) j_0(kr) k^2 dk, \quad (1.74)$$

where $j_0(x) = \sin(x)/x$. The baryon acoustic oscillation (BAO) peak at $r \approx 150 \text{ Mpc}/h$ provides a standard ruler for measuring the expansion history.

1.9.4 The GADGET family and OpenGadget3

The GADGET (GALaxies with Dark matter and Gas intErAcT) code family was developed by Volker Springel, with the public release of GADGET-2 in 2005 (Springel, 2005). It introduced the modern SPH formulation with grad-h terms, the TreePM gravity algorithm, and quasi-symplectic time integration. GADGET-3 (not publicly released) extended this with improved SPH, MHD, radiative cooling, and star formation.

OpenGADGET3 is based on GADGET-3, developed at USM/LMU Munich by Klaus Dolag and collaborators within the Magneticum project. Key extensions relative to the base GADGET-3 include:

- Wendland kernel family for improved SPH density estimates.
- Constrained hyperbolic divergence cleaning for MHD.
- Physical thermal conduction and Navier–Stokes viscosity.
- Time-dependent artificial viscosity and conductivity.
- Meshless Finite Mass (MFM) hydro solver.
- Boundary particle implementation for laboratory-style experiments.
- Extended sub-grid physics for the Magneticum cosmological simulations.

The public repository is at <https://gitlab.lrz.de/MAGNETICUM/Hydro-OpenGadget3>.

Part II

Installation and Workflow

Chapter 2

Getting the Code Running

This chapter walks through everything needed to go from a fresh login on an HPC cluster to a completed first simulation. The steps are the same for every tutorial experiment; only the `Config.sh` flags, the parameter file values, and the initial conditions change from one experiment to the next. Read this chapter once carefully before starting Tutorial T0; you will return to specific sections as reference throughout the course.

2.1 Prerequisites and Environment

OpenGADGET3 is a compiled C/C++ program that uses MPI for distributed-memory parallelism and OpenMP for shared-memory (thread-level) parallelism within each MPI rank. Before compiling, the following software must be available:

C/C++ compiler with MPI wrappers. The Intel compilers (`icx/icpx` with `mpicxx`) give the best performance on Intel-based clusters and are used throughout these tutorials. GNU (`g++` with `mpicxx`) also works. The MPI wrapper `mpicxx` must be in `$PATH`.

GNU Scientific Library (GSL), version ≥ 1.9 . Used for special functions, random number generation, and numerical integration. Typically installed system-wide on HPC clusters.

FFTW3. Required only for the PM gravity solver used in large cosmological simulations. Not needed for the non-cosmological tutorial runs.

HDF5 library. Required only if you use `SnapFormat 3` (HDF5 snapshot format). Recommended for production runs but not required for tutorials.

OpenMP. Usually bundled with the compiler; no separate installation needed.

On HPC clusters managed with the `module` system, load the required environment before *every* compilation and every run session. On the tutorial cluster at USM/LMU Munich:

```
module load compiler openmpi
export SYSTYPE=Cast
```

Important

If you close your terminal and log back in, you must run `module load` again. The module environment is *not* persistent across login sessions. A good practice is to add these two lines to `~/.bashrc` so they run automatically, but check with system

administrators first since module conflicts can cause problems on some clusters.

The SYSTYPE variable tells the OpenGADGET3 `Makefile` which block of compiler paths and library locations to use. Open `Makefile` in a text editor and find the section that begins:

```
ifeq ($(SYSTYPE),"Cast")
CC      = mpicxx
OPTIMIZE = -O3 -march=native
GSL_INCL = -I/path/to/gsl/include
GSL_LIBS = -L/path/to/gsl/lib
...
endif
```

If your cluster is not listed, copy an existing block and adapt the library paths to your system. The cluster administrators can tell you where GSL, FFTW, and HDF5 are installed.

To verify that the environment is correctly set up:

```
mpicxx --version      # should print the compiler version, not an
                     error
which mpicxx          # should print a path
echo $SYSTYPE         # should print Cast (or your system name)
```

2.2 Directory Structure

The tutorials follow a consistent directory layout. Create a top-level directory with your name to avoid conflicts with other users of the shared cluster, then one subdirectory per tutorial, and one per experiment within each tutorial:

```
mkdir MyName
mkdir MyName/Tutorial00
mkdir MyName/Tutorial00/Experiment1
cd MyName/Tutorial00/Experiment1
```

Within each experiment directory, you will have:

```
Experiment1/
  Hydro-OpenGadget3/  # the code repository
  Config.sh           # compile-time flags for this experiment
  box.param           # runtime parameter file
  runme.sh            # SLURM job script
  box.ic              # initial conditions file
  output/             # directory where snapshots are written (must
                     exist!)
  slurm-XXXXX.out      # SLURM log (created when job runs)
  energy.txt          # global energy log (written during run)
  cpu.txt             # CPU timing breakdown
  parameters-usedvalues # actual parameter values used (written at run
                     start)
```

2.3 Obtaining the Code

Clone the OpenGADGET3 repository from LRZ GitLab. This is done once; the same clone is reused for all experiments:

```
git clone https://gitlab.lrz.de/MAGNETICUM/Hydro-OpenGadget3
```

The repository structure:

```
Hydro-OpenGadget3/
  src/                # C/C++ source files
  Makefile            # build system; contains SYSTYPE-specific sections
  Config.sh           # compile-time physics flags (you edit this)
  TREECOOL           # UV background cooling table (needed for COOLING
                      module)
  README              # brief notes
```

Once cloned, get the tutorial configuration and runtime files:

```
cd Hydro-OpenGadget3
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Config.sh
cd ..
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/box.param
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/runme.sh
```

2.4 The Configuration File Config.sh

OpenGADGET3 is highly configurable. Physics modules are activated or deactivated at *compile time* by adding or removing flags in `Config.sh`. This file is a shell script; each active flag is a bare word on its own line, and commenting out a line with `#` disables the corresponding feature.

Important

Every time you change `Config.sh` you **must recompile** with `make -j`. The compiled binary encodes the physics at compile time. A mismatch between the binary and the parameter file will produce at best a crash and at worst silently wrong results.

2.4.1 How compile-time flags work

When you run `make -j`, each active flag in `Config.sh` is translated into a `-D` preprocessor definition (e.g., `PERIODIC` becomes `-DPERIODIC`) passed to the compiler. The source code uses `#ifdef PERIODIC ... #endif` blocks to include or exclude the corresponding code paths. This means:

- Disabled features add zero runtime overhead — the code is simply not compiled in.
- Enabling a feature always requires recompilation.
- Only one kernel flag may be active at a time.

2.4.2 Standard configurations per tutorial

Pure hydro, periodic, no gravity (T0, T2, T3, T6, T7a):

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
```

Gravitational collapse, isolated (T1, T5):

```
#PERIODIC          % # disables: isolated boundaries
#NOGRAVITY         % # disables: gravity is active
WENDLAND_C4_KERNEL
EVALPOTENTIAL      % write potential to snapshots
```

Non-cubic slab (T4, T6, T8a, T10):

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
LONG_X=40          % x-extent = 40 * BoxSize
LONG_Y=1
LONG_Z=1
```

When `LONG_X=40` is set, `BoxSize` in the parameter file is the *base* side length; the physical *x*-extent is $40 \times \text{BoxSize}$.

2D simulation (T2, T4, T7b, T9a, T9b):

```
PERIODIC
NOGRAVITY
WENDLAND_C2_KERNEL
TWO DIMS           % particles confined to z=0; forces computed in 2D
```

Boundary particles — walls and pistons (T7b, T8a, T9a):

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
SPH_BND_PARTICLES % particles with ID=0 are walls
SPH_BND_PARTICLES_ALLOW_DRIFT % moving walls keep their initial
                             velocity
```

MHD shock tube (T10):

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
LONG_X=100
LONG_Y=1
LONG_Z=1
MAGNETIC
MUO_UNITY
NO_BALSARA_LIMITER
```

Galaxy simulation with cooling and star formation (T11):

```
#PERIODIC
#NOGRAVITY
WENDLAND_C4_KERNEL
COOLING
SFR
GENERATIONS=4
```

2.4.3 Flag reference table

Table 2.1: Commonly used `Config.sh` flags.

Flag	Effect
<i>Geometry and gravity</i>	
PERIODIC	Periodic boundary conditions.
NOGRAVITY	Disable self-gravity completely.
LONG_X=N	Stretch box by factor N in x .
LONG_Y=N	Stretch box by factor N in y .
LONG_Z=N	Stretch box by factor N in z .
TWODIMS	2D mode: particles in $z = 0$ plane.
EVALPOTENTIAL	Compute and write gravitational potential Φ .
<i>SPH kernel (choose exactly one)</i>	
STANDARD_KERNEL	Cubic spline M4, C^2 smoothness.
WENDLAND_C2_KERNEL	Wendland C^2 — good for 2D tests.
WENDLAND_C4_KERNEL	Wendland C^4 — recommended general choice.
WENDLAND_C6_KERNEL	Wendland C^6 — most accurate, highest cost.
QUINTIC_KERNEL	Quintic spline M6.
<i>Artificial dissipation</i>	
ARTIFICIAL_CONDUCTIVITY	Reduces pressure blip at contact discontinuities.
TIME_DEP_ART_VISC=2	Viscosity grows near shocks, decays in quiet regions.
TIME_DEP_ART_COND=2	Time-dependent artificial conductivity.
ARTIFICIAL_MAGNETIC_DISSIPATION	Artificial resistivity for MHD.
<i>Boundary particles</i>	
SPH_BND_PARTICLES	Particles with ID=0 act as fixed walls.
SPH_BND_PARTICLES_ALLOW_DRIFT	Moving boundary particles maintain velocity.
SPH_BND_PARTICLES_ALLOW_CONDUCTION	Thermal conduction through boundary particles.
<i>MHD</i>	
MAGNETIC	Full MHD equations.
MU0_UNITY	Set $\mu_0 = 1$ (Gaussian-like unit system).
NO_BALSARA_LIMITER	Disable the Balsara viscosity switch (needed for MHD).
DIVBCLEANING_TRICCO	Divergence cleaning to maintain $\nabla \cdot \mathbf{B} = 0$.
<i>Transport physics</i>	
CONDUCTION	Thermal conduction.
CONDUCTION_CONSTANT	Constant conductivity (vs. Spitzer $\propto T^{5/2}$).
NAVIERSTOKES	Physical Navier–Stokes viscosity.
NAVIERSTOKES_CONSTANT	Constant viscosity coefficient.
<i>Sub-grid physics</i>	
COOLING	Radiative cooling (requires TREECOOL in run dir).
SFR	Star formation (requires COOLING).
WINDS	Kinetic stellar feedback (galactic winds).
GENERATIONS=N	Max stellar generations per gas particle ($N = 4$ typical).
<i>Alternative hydro solver</i>	
USE_MESHLESS_FINITE_MASS	Replace SPH with the MFM Riemann solver.
<i>Miscellaneous</i>	
CONSTANT_GRAVITY_Y=V	Add constant gravitational acceleration V in y .

2.5 Compiling the Code

With `Config.sh` prepared and the modules loaded:

```
cd Hydro-OpenGadget3
make -j                # parallel build using all available cores
cd ..
```

A successful build ends with `Linking Gadget3` and produces the executable `Hydro-OpenGadget3/Gadget3`.

Common build failures and fixes:

mpicxx: command not found Run `module load compiler openmpi` and retry.

lm_gsl.a: No such file or directory GSL not found. Verify that `SYSTYPE` is set and that the GSL paths in the corresponding `Makefile` section are correct.

only one kernel may be selected Remove all but one kernel flag from `Config.sh`.

make: * [Gadget3] Error 1** Scroll up to the first error line; this is just the abort message.

To do a clean rebuild from scratch (always do this when changing `SYSTYPE` or after a `git pull` that changes source files):

```
cd Hydro-OpenGadget3 && make clean && make -j && cd ..
```

2.6 The Parameter File in Detail

The parameter file (by convention `box.param`) controls all runtime behaviour. Every parameter has a built-in default; you only need to specify those that differ from the default. At startup, OpenGADGET3 writes `parameters-usedvalues` with all parameters and their actual values, including defaults. Always check this file.

Tip

Keep `parameters-usedvalues` together with your `Config.sh` and IC file as the complete simulation record. These three files are all you need to reproduce a run exactly.

2.6.1 File paths

<code>InitCondFile</code>	<code>box.ic</code>	<i>% IC filename WITHOUT extension. % For multi-file ICs the code adds .0, .1, ...</i>
<code>OutputDir</code>	<code>./output</code>	<i>% MUST exist before the run starts. % The code will not create it automatically.</i>
<code>SnapshotFileBase</code>	<code>snap</code>	<i>% snapshots named snap_000, snap_001, ...</i>
<code>ICFormat</code>	<code>2</code>	<i>% 1=legacy binary, 2=block binary, 3= HDF5</i>
<code>SnapFormat</code>	<code>2</code>	<i>% same options; use same format as IC</i>

Important

Create `output/` before submitting: `mkdir output`. If missing, the code aborts immediately with a file-open error. This is the single most common mistake in the tutorials.

2.6.2 Time control

```

TimeBegin          0.0      % start time [code time units]
                        % for cosmological runs: scale factor a =
                        % 1/(1+z)
TimeMax            5.0      % end time in the same units
TimeBetSnapshot    0.1      % time between consecutive snapshots.
                        % e.g. TimeMax=5, TimeBetSnapshot=0.1 ->
                        % 50 snapshots
TimeOfFirstSnapshot 0.0     % time of the very first snapshot output
TimeBetRestartFile 12.0     % wall-clock hours between restart file
                        % writes.
                        % Two alternating restart files are kept
                        % so a crash during writing corrupts at
                        % most one.

```

For a tutorial run of a few hours, setting `TimeBetSnapshot` to give 50–100 snapshots is usually appropriate. More snapshots make better movies but increase I/O time and disk usage.

2.6.3 Unit system

`OpenGADGET3` has no built-in unit system. You define three base units and the code derives everything else. All quantities in parameter files, IC files, and snapshots are in these units.

```

UnitLength_in_cm    3.085678e21  % 1.0 kpc/h in cm
UnitMass_in_g       1.989e43     % 1e10 solar masses in grams
UnitVelocity_in_cm_per_s 1e5      % 1 km/s in cm/s

```

Listing 2.1: Standard cosmological unit system: kpc/h , $10^{10} M_{\odot}$, km/s .

From these three, the code derives:

- Time unit: $[t] = [L]/[v] = 3.086 \times 10^{16} \text{ s} \approx 0.978 \text{ Gyr}$.
- Energy per unit mass: $[u] = [v]^2 = 10^{10} \text{ cm}^2 \text{ s}^{-2}$.
- Gravitational constant: G in code units ≈ 43.009 .
- Boltzmann constant: k_B in code units (used internally for cooling).

For numerical experiments without physical units:

```

UnitLength_in_cm    1.0
UnitMass_in_g       1.0
UnitVelocity_in_cm_per_s 1.0
% Then [t]=1, [u]=1, G=1, all in code units.

```

2.6.4 Boundaries and gravity

```

PeriodicBoundariesOn      1          % 1=periodic, 0=isolated (open)
                                % Must match the PERIODIC flag in
                                % Config.sh

ErrTolForceAcc            0.005      % tree opening criterion; smaller=more
                                % accurate gravity but longer tree walk

                                % Default 0.005 is fine for all
                                % tutorials.

ErrTolIntAccuracy        0.025      % gravity timestep accuracy parameter eta.
                                % Timestep ~ sqrt(2 * eta * epsilon / |a|)

                                % Smaller = smaller timesteps, higher
                                % accuracy.

MaxSizeTimestep 0.01      % hard upper limit on any particle's timestep
                                % [code time units]. MUST be set; default 0
                                % means no limit and can cause instability.

CourantFac                0.15      % CFL safety factor for the hydro timestep.
                                % Timestep ~ CourantFac * h / v_signal.

```

2.6.5 Gravitational softening

```

% Softening length for each particle type [code length units].
% Type: 0=gas, 1=dark matter (halo), 2=disk, 3=bulge, 4=stars, 5=
% boundary.
SofteningGas              0.2        % prevents 1/r^2 divergence for r <
% epsilon
SofteningHalo             0.2
SofteningDisk             0.02
SofteningBulge           0.02
SofteningStars            0.02

% For cosmological runs: above values are in comoving units.
% Once the physical softening (= comoving * a) reaches MaxPhys,
% the softening is held fixed in physical coordinates.
SofteningGasMaxPhys      0.2
SofteningHaloMaxPhys     0.2

```

Rule of thumb: $\epsilon \approx L/(30 N^{1/3})$ where L is the box size and N the particle count. Too small a softening leads to very short gravity timesteps $\Delta t \propto \sqrt{\epsilon/|a|}$, making the run extremely slow.

2.6.6 SPH parameters

```

DesNumNgb                64          % target SPH neighbour count.
                                % Kernel-dependent default is set
                                % automatically;
                                % see parameters-usedvalues after first
                                % run.
                                % Typical values: 32-64 for cubic spline
                                % ,
                                % 100-200 for Wendland
                                % C4/C6.

```

MaxNumNgbDeviation adjusted	1	% allowed deviation from DesNumNgb; h is adjusted % so actual Nngb is in [DesNumNgb-1, DesNumNgb+1].
ArtBulkViscConst	0.8	% artificial viscosity coefficient alpha. % Sets the strength of dissipation at shocks. % Smooth flows (sound waves, conduction) : 0.05. % Moderate shocks (Riemann problem): 0.8. % Strong shocks (supernova blast wave): 1.5.
PartAllocFactor a	1.5	% memory allocated per particle type, as multiple of the initial particle count . % For SFR runs where gas converts to stars, % use 5.5 to avoid memory reallocation errors.

2.6.7 Sub-grid physics

```
%--- Cooling and star formation (requires COOLING + SFR in Config.sh)
-----
CoolingOn          1          % enable radiative cooling
StarformationOn    1          % enable star formation
MaxSfrTimescale    1.5        % star formation timescale t_0* [code time]
                        % sets the normalisation of the SK relation

%--- Thermal conduction (requires CONDUCTION + CONDUCTION_CONSTANT)
-----
ConductionEfficiency 0.1      % kappa = Efficiency * kappa_Spitzer(T)
                        % 1.0 = full Spitzer value
MaxSizeConductionStep 0.005  % maximum conduction timestep [code time]

%--- Physical viscosity (requires NAVIERSTOKES + NAVIERSTOKES_CONSTANT)
) ----
ShearViscosityTemperature 5.0e7 % reference temperature for Spitzer
eta [K]
FractionSpitzerViscosity 1.0    % eta = Fraction * eta_Spitzer(T)

%--- Galactic winds (requires WINDS in Config.sh)
-----
WindEfficiency      2.0      % mass loading: M_wind/M_* = WindEfficiency
WindEnergyFraction  0.5      % fraction of SN energy going into wind
WindFreeTravelMaxTimeFactor 0.025
WindFreeTravelDensFac 0.5

%--- MHD dissipation (requires ARTIFICIAL_MAGNETIC DISSIPATION)
-----
ArtificialMagneticDissipationConstant 0.2
```

2.7 The SLURM Job Script

On clusters running SLURM, simulations are submitted as batch jobs. The job script requests resources and issues the run command.

```
#!/bin/bash
#SBATCH --job-name=TO_box_exp1      # UNIQUE name per run; shows in
#                                     queue
#SBATCH --ntasks=8                  # number of MPI ranks
#SBATCH --cpus-per-task=4            # OpenMP threads per MPI rank
#SBATCH --time=04:00:00              # wall-clock limit hh:mm:ss
#SBATCH --output=slurm-%j.out        # log file; %j replaced by SLURM
#                                     job ID

# Reload environment (must match compilation environment exactly)
module load compiler openmpi
export SYSTYPE=Cast
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Run: ntasks MPI ranks, each using cpus-per-task OpenMP threads
mpirun -n $SLURM_NTASKS ./Hydro-OpenGadget3/Gadget3 box.param
```

Listing 2.2: Fully annotated SLURM job script (runme.sh).

Tip

The product `ntasks × cpus-per-task` must not exceed the number of physical cores on the node(s). On a 32-core node, good choices are: `ntasks=1,cpus=1` (tiny tests); `ntasks=2,cpus=2` (quick 2D runs); `ntasks=8,cpus=4` (standard 3D tutorials).

At startup, OpenGADGET3 confirms the parallelism in the log:

```
Running on 8 MPI tasks
Using 4 OpenMP threads
```

Submitting and monitoring:

```
mkdir output          # create output directory BEFORE submitting
sbatch runme.sh        # submit; prints "Submitted batch job 12345"

squeue -u $USER        # list your jobs: JOBID, STATE, TIME, NODELIST
scancel 12345          # cancel if needed

tail -f slurm-12345.out # follow log in real time (Ctrl-C to stop)
```

OpenGADGET3 prints a one-line progress summary per timestep:

```
Begin Step 0, Time: 0, Nf = 1000, ...
Begin Step 1, Time: 0.0012, Nf = 1000, ...
```

Nf is the number of active particles. A completed run ends with:

```
done with simulation
```

Diagnostic files written during the run:

energy.txt Time, internal energy E_{int} , potential E_{pot} , kinetic E_{kin} — one row per TimeBetStatistics interval. Useful for monitoring virialization and energy conservation.

cpu.txt Cumulative CPU time by component (tree gravity, SPH hydro, domain decomposition, I/O, ...). Use to identify bottlenecks.

balance.txt Load balance between MPI ranks (fraction of time each rank spends waiting for others). High imbalance ($> 20\%$) suggests a poor domain decomposition.

parameters-usedvalues All parameter values as actually used, including defaults. Written at startup.

2.8 Restarting an Interrupted Run

OpenGADGET3 writes binary restart files at intervals of `TimeBetRestartFile` wall-clock hours. Two alternating sets (`restart.0.X`, `restart.1.X`) are kept so that a crash during writing cannot corrupt both.

To resume from the last checkpoint:

```
# Change the last line of runme.sh to add the flag "1":
mpirun -n $SLURM_NTASKS ./Hydro-OpenGadget3/Gadget3 box.param 1
```

To restart from a specific snapshot (e.g., after changing a parameter):

```
# In box.param:
InitCondFile      output/snap_050      % point to the desired snapshot

# Then run with flag "2":
mpirun -n $SLURM_NTASKS ./Hydro-OpenGadget3/Gadget3 box.param 2
```

Important

Restart files are MPI-rank-count-specific. If you change `ntasks` between runs, you cannot use restart files — restart from a snapshot instead.

2.9 The Standard Tutorial Workflow

Every experiment follows the same five steps:

1. **Edit `Config.sh`** with the flags for this experiment. If unchanged from the previous experiment, skip to step 3.
2. **Compile.**

```
cd Hydro-OpenGadget3 && make -j && cd ..
```

3. **Prepare the IC file.** Either download with `wget` or generate with a setup script (Chapter 4).
4. **Edit `box.param`.** Set at minimum: `InitCondFile`, `TimeMax`, `TimeBetSnapshot`, softening lengths. Create the output directory: `mkdir output`.
5. **Submit.**

```
sbatch runme.sh && tail -f slurm-*.out
```

When complete, check `parameters-usedvalues` and begin analysis.

Always work in a new directory for each experiment. This prevents output files from one run overwriting another.

Chapter 3

Snapshot Format and I/O

Understanding how OpenGADGET3 writes its output is as important as understanding how to run it. Every analysis step — visualisation, derived quantities, comparisons with observations — starts by reading snapshot files. This chapter describes the file formats in detail, explains which data blocks are available and what they contain, and shows how to read them efficiently with Python.

3.1 File Formats

Format 1 — legacy binary. The original GADGET-1/2 format. Fortran-style binary records, each framed by a 4-byte integer giving the block size. Portable but tedious to parse selectively. Set `SnapFormat 1` (and `ICFormat 1`).

Format 2 — labelled binary. As Format 1, but each block is preceded by a 4-byte ASCII label (`HEAD`, `POS`, `VEL`, `ID`, etc.). Selective reading is straightforward. Set `SnapFormat 2`. This is the format used in all tutorials.

Format 3 — HDF5. Self-documenting, portable, directly readable by Python/h5py, yt, and many analysis tools. Strongly recommended for new production runs. Set `SnapFormat 3`; files are named `snap_000.hdf5`.

3.2 Particle Types

GADGET/OpenGADGET3 snapshots distinguish six particle types:

Table 3.1: Particle types in OpenGADGET3 snapshots.

Type	Name	Description
0	Gas	SPH gas particles; carry all hydrodynamic fields
1	Halo	Collisionless dark matter
2	Disk	Disk N-body particles
3	Bulge	Bulge N-body particles
4	Stars	Stellar particles formed by SFR
5	Bndry	Boundary particles (<code>ID = 0</code>)

In non-cosmological hydro tests, only type 0 (gas) is usually present. In galaxy simulations with `SFR`, type 4 (stars) is created dynamically. Boundary particles (type 5) are used to impose fixed or moving walls (Tutorials T7b, T8a, T9a).

3.3 Data Blocks

Each snapshot contains the following standard blocks (presence depends on compile-time flags and particle types present):

Table 3.2: Standard data blocks in OpenGADGET3 snapshots.

Block	Quantity	Notes
HEAD	Header	Box size, time, N_{part} , cosmology
POS	Positions	$3 \times N$ floats, code length units
VEL	Velocities	$3 \times N$ floats; comoving for cosmo runs
ID	Particle IDs	N integers; unique, used for tracking
MASS	Individual masses	Only if not all same mass per type
U	Specific internal energy	Gas only; code velocity ²
RHO	SPH density	Gas only; code mass / code length ³
HSML	Smoothing length	Gas only; code length
NE	Electron fraction n_e/n_H	If COOLING
NH	Neutral H fraction	If COOLING
SFR	Star formation rate [$M_{\odot} \text{yr}^{-1}$]	If SFR
AGE	Stellar formation time	Star particles; SFR
Z	Metallicity	If SFR
BFLD	Magnetic field	$3 \times N$ floats; if MAGNETIC
POT	Gravitational potential	If EVALPOTENTIAL

3.4 Reading Snapshots with Python

3.4.1 g3read.py (Format 1 and 2)

The tutorials use `g3read.py`, a lightweight reader for GADGET binary snapshots:

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Python/g3read.py
```

```

1 import g3read as g3
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Open snapshot (auto-detects format 1 or 2)
6 f = g3.GadgetFile('snap_010')
7
8 # Read positions of gas particles (type 0)
9 pos = f.read_new('POS ', 0) # shape: (N_gas, 3)
10 rho = f.read_new('RHO ', 0) # shape: (N_gas,)
11 u = f.read_new('U ', 0) # specific internal energy
12
13 # Dark matter particles (type 1)
14 pos_dm = f.read_new('POS ', 1)
15
16 # Header information
17 print('BoxSize:', f.header.BoxSize)
18 print('Time: ', f.header.time)
19 print('N_gas: ', f.header.npart[0])
20
```

```

21 # Simple scatter plot
22 plt.scatter(pos[:,0], pos[:,1], s=0.1, c=np.log10(rho))
23 plt.colorbar(label=r'$\log_{10}(\rho)$')
24 plt.savefig('snapshot.png', dpi=150)

```

Listing 3.1: Basic usage of g3read.py.

3.4.2 h5py (Format 3 / HDF5)

```

1 import h5py
2 import numpy as np
3
4 with h5py.File('snap_010.hdf5', 'r') as f:
5     # Header
6     h = f['Header'].attrs
7     print('Time:', h['Time'], 'BoxSize:', h['BoxSize'])
8
9     # Gas particles (PartType0)
10    pos = f['PartType0/Coordinates'][:] # (N, 3)
11    vel = f['PartType0/Velocities'][:] # (N, 3)
12    rho = f['PartType0/Density'][:] # (N,)
13    u = f['PartType0/InternalEnergy'][:] # (N,)
14    ids = f['PartType0/ParticleIDs'][:]
15
16    # Dark matter (PartType1)
17    pos_dm = f['PartType1/Coordinates'][:]

```

Listing 3.2: Reading an HDF5 snapshot with h5py.

3.4.3 Computing temperature

The temperature of gas particles is not stored directly. It is computed from the specific internal energy u using the equation of state:

$$T = \frac{(\gamma - 1) \mu m_p u}{k_B}, \quad (3.1)$$

where $\mu \approx 0.588$ for a fully ionised solar-abundance plasma and m_p , k_B are the proton mass and Boltzmann constant. In code units (kpc/h , $10^{10} M_\odot$, km/s):

```

1 # Constants in CGS
2 k_B = 1.3806e-16 # erg/K
3 m_p = 1.6726e-24 # g
4 mu = 0.588 # mean molecular weight (fully ionised solar plasma)
5 gamma = 5.0/3.0
6
7 # Unit conversion: (km/s)^2 to erg/g
8 vel_unit = 1e5 # 1 km/s in cm/s
9 u_cgs = u * vel_unit**2 # u in erg/g
10
11 # Temperature in Kelvin
12 T = (gamma - 1.0) * mu * m_p * u_cgs / k_B
13 print('T range:', T.min(), '--', T.max(), 'K')

```

Listing 3.3: Temperature calculation from specific internal energy.

3.5 Visualisation Tools

matplotlib The standard Python plotting library. Scatter plots (`plt.scatter`), 2D histograms (`plt.hist2d`), and animations (`FuncAnimation`) are used throughout the tutorials.

FFmpeg Converts sequences of PNG frames to MP4 movies:

```
ffmpeg -framerate 10 -i frame_%03d.png -c:v libx264 movie.mp4
ffplay -loop 0 movie.mp4
```

yt A powerful framework for analysing and visualising astrophysical simulation data. Reads GADGET binary and HDF5 snapshots directly, provides projection plots, slices, profiles, and halo finding. See Chapter 20 for examples.

gnuplot Lightweight and fast; used in some tutorial examples for quick plots from text files.

3.6 The Snapshot Header

The header is always the first block in every snapshot. It contains all the metadata needed to interpret the rest of the file. In Python with `g3read.py`:

```
1 import g3read as g3
2
3 f = g3.GadgetFile('output/snap_010')
4 h = f.header
5
6 print('Snapshot time:      ', h.time)
7 print('Redshift:           ', h.redshift)
8 print('Box size:           ', h.BoxSize, '[code length]')
9 print('Npart per type:      ', h.npart) # e.g. [1000, 0, 0, 0, 0, 0]
10 print('Mass per type:       ', h.mass) # 0.0 if individual masses stored
11 print('Omega_0:            ', h.Omega0)
12 print('OmegaLambda:         ', h.OmegaLambda)
13 print('HubbleParam h:       ', h.HubbleParam)
14 print('Flag cooling:          ', h.flag_cooling)
15 print('Flag SFR:             ', h.flag_sfr)
```

Listing 3.4: Reading and printing the snapshot header.

3.7 Computing Physical Quantities

Several quantities of interest are not stored directly but derived from the primary fields.

3.7.1 Temperature

The temperature of a gas particle is derived from specific internal energy u :

$$T = \frac{(\gamma - 1) \mu m_p u}{k_B}, \quad (3.2)$$

where $\gamma = 5/3$, $\mu \approx 0.588$ for a fully ionised solar-abundance plasma, m_p the proton mass, k_B Boltzmann's constant.

```
1 import numpy as np
2
3 k_B = 1.3806e-16 # erg/K
```

```

4 m_p = 1.6726e-24 # g
5 gamma, mu = 5.0/3.0, 0.588
6 vel_unit_cgs = 1e5 # 1 km/s in cm/s
7
8 u_cgs = u * vel_unit_cgs**2 # convert code (km/s)^2 to erg/g
9 T = (gamma-1.0) * mu * m_p * u_cgs / k_B
10
11 print(f'T range: {T.min():.2e} -- {T.max():.2e} K')

```

Listing 3.5: Computing gas temperature from internal energy.

3.7.2 Pressure and Mach number

```

1 P = (gamma - 1.0) * rho * u # pressure [code units]
2 c_s = np.sqrt(gamma * P / rho) # sound speed
3 v = np.sqrt((vel**2).sum(axis=1)) # total speed
4 mach = v / c_s # Mach number

```

3.8 Tracking Individual Particles

Each particle carries a unique ID preserved across snapshots, enabling tracking of individual fluid elements (Tutorial T2):

```

1 import g3read as g3, numpy as np, glob
2
3 target_id = 500
4 snaps = sorted(glob.glob('output/snap_???'))
5 track = []
6
7 for snap_path in snaps:
8     f = g3.GadgetFile(snap_path)
9     ids = f.read_new('ID ', 0)
10    pos = f.read_new('POS ', 0)
11    idx = np.where(ids == target_id)[0]
12    if len(idx) == 1:
13        track.append([f.header.time] + list(pos[idx[0]]))
14
15 track = np.array(track) # columns: t, x, y, z

```

Listing 3.6: Tracking a particle trajectory by ID.

3.9 Visualisation Tools

matplotlib Scatter plots, 2D histograms, animations. Key functions: `scatter`, `hist2d`, `imshow`, `animation.FuncAnimation`.

FFmpeg Converts PNG frames to video:

```

ffmpeg -framerate 10 -i frame_%03d.png -c:v libx264 movie.mp4
ffplay -loop 0 movie.mp4

```

h5ls / h5dump HDF5 command-line inspection tools. Useful for quickly browsing HDF5 snapshot structure without writing code.

gnuplot Fast diagnostic plots from text files. Tutorial-provided `.plt` scripts handle most common cases.

yt Full-featured analysis framework for astrophysical simulations. Reads GADGET binary and HDF5 directly; produces projections, slices, profiles, and halos. See Chapter [20](#).

Chapter 4

Generating Initial Conditions

Initial conditions (ICs) specify the positions, velocities, masses, and thermodynamic state of all particles at the start of the simulation. OpenGADGET3 reads ICs in the same binary format as snapshots (Section 3.1). This chapter covers both the simple hand-crafted ICs used in tutorials and the cosmological ICs generated by dedicated codes.

4.1 IC File Structure

An IC file in Format 2 consists of:

1. A HEAD block: the 256-byte header (same as snapshots).
2. A POS block: positions of all particles.
3. A VEL block: velocities.
4. An ID block: integer IDs (must be unique).
5. A MASS block: masses (optional if uniform per type).
6. A U block: specific internal energy (gas particles only).

The header structure in C:

```
struct io_header {
    int    npart[6];      /* particle numbers per type in this file */
    double mass[6];       /* mass per type (0 if individual masses) */
    double time;          /* simulation time or scale factor */
    double redshift;      /* redshift */
    int    flag_sfr;      /* star formation flag */
    int    flag_feedback;
    int    npartTotal[6]; /* total particle numbers across all files */
    int    flag_cooling;
    int    num_files;     /* number of files per snapshot */
    double BoxSize;       /* box size in code length units */
    double Omega0;
    double OmegaLambda;
    double HubbleParam;
    char   fill[256 - 6*4 - 6*8 - 2*8 - 2*4 - 6*4 - 2*4 - 4*8];
} header;
```

Listing 4.1: GADGET header structure.

For non-cosmological simulations: set `time = 0`, `redshift = 0`, `Omega0 = OmegaLambda = HubbleParam = 0`.

4.2 Setting Up Simple ICs with Python

The tutorials provide `g3read.py` and `ic_header` (a binary template header file) to write ICs in Python without implementing the full binary format from scratch:

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Python/g3read.py
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Python/ic_header
```

4.2.1 Uniform grid (cubic lattice)

```
1 import numpy as np
2 import struct
3
4 N = 10    # particles per side
5 L = 1.0   # box size
6
7 # Generate positions: cubic lattice
8 x = np.linspace(0, L, N, endpoint=False) + L/(2*N)
9 X, Y, Z = np.meshgrid(x, x, x)
10 pos = np.column_stack([X.ravel(), Y.ravel(), Z.ravel()]).astype(np.float32)
11
12 N_tot = N**3
13 vel = np.zeros((N_tot, 3), dtype=np.float32)
14 ids = np.arange(1, N_tot+1, dtype=np.int32)
15 mass = np.ones(N_tot, dtype=np.float32) * (1.0 / N_tot) # total mass
16      = 1
17 u      = np.ones(N_tot, dtype=np.float32) * 0.05          # cold gas
18
19 # Read the template header and modify it
20 with open('ic_header', 'rb') as f:
21     header = bytearray(f.read(256))
22
23 # Write N_tot gas particles (type 0), mass 1.0/N_tot each
24 import struct
25 struct.pack_into('6i', header, 0, N_tot, 0, 0, 0, 0, 0) # npart
26 struct.pack_into('6d', header, 24, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) #
27     mass in header = 0
28 struct.pack_into('d', header, 72, 0.0) # time
29 struct.pack_into('d', header, 80, 0.0) # redshift
30 struct.pack_into('6i', header, 116, N_tot, 0, 0, 0, 0, 0) #
31     npartTotal
32 struct.pack_into('i', header, 140, 1) # num_files
33 struct.pack_into('d', header, 144, L) # BoxSize
34
35 def write_block(f, label, data):
36     """Write a labelled binary block (Format 2)."""
37     raw = data.tobytes()
38     size = len(raw)
39     # Label record
40     f.write(struct.pack('i', 8))
41     f.write(label.encode('ascii').ljust(4)[:4])
42     f.write(struct.pack('i', size + 8))
43     f.write(struct.pack('i', 8))
44     # Data record
45     f.write(struct.pack('i', size))
```

```

43     f.write(raw)
44     f.write(struct.pack('i', size))
45
46 with open('box.ic', 'wb') as f:
47     # HEAD block
48     f.write(struct.pack('i', 8))
49     f.write(b'HEAD')
50     f.write(struct.pack('i', 264))
51     f.write(struct.pack('i', 8))
52     f.write(struct.pack('i', 256))
53     f.write(bytes(header))
54     f.write(struct.pack('i', 256))
55     # Data blocks
56     write_block(f, 'POS ', pos)
57     write_block(f, 'VEL ', vel)
58     write_block(f, 'ID  ', ids)
59     write_block(f, 'MASS', mass)
60     write_block(f, 'U   ', u)
61
62 print('Written box.ic with', N_tot, 'gas particles')

```

Listing 4.2: Setting up a 10x10x10 uniform gas grid.

Tip

For production simulations, use glass-file initial conditions rather than a cubic lattice. Pre-computed glass files are available from the tutorial server:

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/
    glass_10x10x10
```

4.2.2 Non-uniform density: the spherical cloud

To represent a non-uniform density profile (e.g. $\rho \propto 1/r$ for the Evrard collapse, Tutorial T5), start from a large glass distribution and transform particle positions to follow the desired cumulative mass profile. For a profile $\rho(r) \propto r^{-\alpha}$:

```

1  # Assume 'pos' is the glass distribution in [0,1]^3
2  # Centre the distribution
3  pos -= 0.5
4  r = np.sqrt((pos**2).sum(axis=1))
5
6  # Keep only particles inside the unit sphere
7  mask = r < 0.5
8  pos = pos[mask]
9  r = r[mask]
10
11 # Normalise so r goes from 0 to 1
12 r_norm = r / r.max()
13
14 # For rho ~ 1/r: M(<r) ~ r^2, so r_new ~ r_uniform^(1/2)
15 # Apply radial transformation: stretch r to match the desired CDF
16 r_new = r_norm**(1.0/2.0)      # for alpha=1
17
18 # Scale positions to new radii
19 scale = np.where(r > 0, r_new / r_norm, 0.0)
20 pos = pos * scale[:, None]

```

```

21
22 # Scale to desired physical radius and mass
23 R_star = 1.0      # total radius
24 M_star = 1.5815   # total mass (Evrard normalisation)
25 pos *= R_star
26 mass = np.ones(len(pos), dtype=np.float32) * M_star / len(pos)

```

Listing 4.3: Transforming a uniform glass to a $1/r$ density profile.

4.3 Slab and Non-cubic Geometries

Many tutorial experiments use elongated domains (`LONG_X`, `LONG_Y`, `LONG_Z` in `Config.sh`). The IC file still uses the base `BoxSize` set in the parameter file; the `LONG_X` factor stretches the physical domain while the normalised coordinates remain in $[0, 1]$.

A slab of $200 \times 10 \times 10$ particles along x , with the base glass:

```

1  import g3read as g3
2
3  base = g3.GadgetFile('glass_10x10x10')
4  pos0 = base.read_new('POS ', 0)  # 1000 particles in [0,1]^3
5
6  # Stack 20 copies along x, 1 copy in y and z
7  copies = []
8  for ix in range(20):
9      p = pos0.copy()
10     p[:, 0] += ix  # shift in x
11     copies.append(p)
12
13  pos_slab = np.vstack(copies)
14  pos_slab[:, 0] /= 20.0  # renormalise to [0,1] (LONG_X=20 in Config.sh)
15
16  N = len(pos_slab)
17  vel = np.zeros((N,3), dtype=np.float32)
18  ids = np.arange(1, N+1, dtype=np.int32)
19  mass = np.ones(N, dtype=np.float32) / N
20  u = np.ones(N, dtype=np.float32) * 0.05
21
22  # Write box.ic (same as above)

```

Listing 4.4: Building a slab from a base glass file.

For the Riemann problem (Tutorial T4), the left and right halves of the slab are given different pressures. For the MHD shock tube (Tutorial T10), a magnetic field vector is added to the IC.

4.4 Cosmological Initial Conditions

Cosmological simulations require initial conditions seeded with the correct power spectrum of density perturbations at a given starting redshift z_i (typically $z_i = 50$ – 100). Two tools are commonly used with OpenGADGET3 :

4.4.1 N-GenIC

N-GenIC (by Volker Springel) generates Gaussian random field initial conditions using the Zel'dovich approximation (1LPT). It is parallel, straightforward to configure, and produces

output directly in GADGET format:

```
wget http://www.mpa-garching.mpg.de/gadget/n-genic.tar.gz
tar xzf n-genic.tar.gz
cd N-GenIC && make
```

A minimal N-GenIC parameter file for a Λ CDM box:

```
Nmesh          256          % PM mesh size (must be >= N_part^(1/3))
Nsample        256          % particle grid per side
Box            150000.0     % box size in kpc/h
FileBase       ics          % output file base name
OutputDir      ./

Omega          0.3          % Omega_matter
OmegaLambda    0.7
OmegaBaryon    0.05
HubbleParam    0.7
Sigma8         0.8          % normalisation

SphereMode     0
WhichSpectrum  2            % 2 = Eisenstein-Hu transfer function
FileWithInputSpectrum ""

Seed           181170       % random seed
NumFilesWrittenInParallel 4
```

Listing 4.5: Minimal N-GenIC parameter file.

Run with: `mpirun -n 4 ./N-GenIC ngenic.param`

4.4.2 MUSIC

MUSIC (Hahn and Abel, 2011) is the modern replacement for N-GenIC. It supports second-order LPT (2LPT, more accurate at $z_i \lesssim 50$), zoom-in (nested grid) initial conditions, and multiple transfer function codes including CAMB.

```
git clone https://github.com/cosmo-sims/MUSIC2
cd MUSIC2 && mkdir build && cd build
cmake .. && make -j
```

A minimal MUSIC configuration file for a GADGET-compatible uniform grid IC:

```
[setup]
boxlength      = 150          % Mpc/h
zstart         = 99
levelmin       = 8            % base grid: 2^8 = 256 per side
levelmax       = 8            % same: no refinement (uniform grid)
ref_center     = 0.5 0.5 0.5
ref_extent     = 0.0 0.0 0.0

[cosmology]
Omega_m        = 0.3
Omega_L        = 0.7
Omega_b        = 0.046
H0             = 70.0
sigma_8        = 0.8
nspec          = 0.96
transfer       = eisenstein_hu
```

```
[random]
seed[8]      = 12345

[output]
format       = gadget2
filename     = ics.dat
gadget_lunit = kpc           % length unit for gadget
gadget_munit = 1e10msun

[poisson]
fft_fine     = yes
```

Listing 4.6: Minimal MUSIC configuration file (ics.conf).

Run with: `./MUSIC ics.conf`

Tip

For zoom simulations (high-resolution re-simulation of a selected halo), MUSIC supports nested refinement levels. The zoom region is identified by tracing halo particles back to the initial conditions from a low-resolution run, then rerunning MUSIC with refined levels centred on that region.

4.4.3 Cosmological parameter file settings

For cosmological OpenGADGET3 runs, the parameter file must include:

```
ComovingIntegrationOn 1           % enable comoving coordinates
TimeBegin             0.01       % scale factor at start (z=99)
TimeMax               1.0        % scale factor at end (z=0)
Omega0                0.3
OmegaLambda           0.7
OmegaBaryon           0.046
HubbleParam           0.7
BoxSize               150000.0   % in kpc/h

% Output list (alternative to TimeBetSnapshot)
OutputListOn          1
OutputListFilename    output_times.txt
```

Listing 4.7: Cosmological parameters in box.param.

The file `output_times.txt` lists scale factors for snapshot output, one per line (e.g. 0.0625, 0.125, ..., 1.0).

Part III

Tutorial Compendium

The chapters in this part follow the tutorial series developed by Klaus Dolag at USM/LMU Munich. Each tutorial builds on the previous one, introducing new physics, new compile-time flags, and new analysis techniques. The experiments are ordered by increasing complexity.

Each chapter is structured as follows: a brief statement of the physical question, the required `Config.sh` flags, the parameter file settings, how to obtain or generate the initial conditions, how to run the simulation, and how to analyse the output. All Python examples can be used as starting points for your own scripts.

The standard directory structure for each experiment:

```
MyName/TutorialXX/ExperimentN/  
  Hydro-OpenGadget3/    # the compiled code  
  Config.sh  
  box.param  
  runme.sh  
  box.ic  
  output/                # must exist before running
```


Chapter 5

Tutorial T0: Environment Setup and First Simulation

Tutorial T0 has one goal: get OpenGADGET3 running and produce a first visualisation. By the end you will have compiled the code, run a small gas simulation, read the output with Python, and made an animation. No physics understanding is required at this stage; the focus is entirely on the mechanics of the workflow.

5.1 Physical Setup

The T0 simulation is the simplest possible: a $10 \times 10 \times 10$ cubic lattice of 1000 gas particles in a periodic unit box, with uniform density and pressure, given a small uniform velocity in the x -direction. Since gravity is disabled and the box is periodic, the gas simply drifts and the simulation is essentially trivial. This is intentional: the goal is to learn the tools, not to study physics.

5.2 Connecting to the Cluster

From a Linux or macOS terminal, connect with X11 forwarding enabled (required for displaying plots directly on the remote machine):

```
ssh -4 -Y username@mpusm06.usm.uni-muenchen.de
# -4 forces IPv4; -Y enables trusted X11 forwarding
```

On Windows, install either:

- **Cygwin** with the OpenSSH and X11 packages: start the X server with `X -multiwindow` in one terminal, then `export DISPLAY=:0` and `ssh -4 -Y username@...` in another.
- **WSL2** (Windows Subsystem for Linux) with an X server such as VcXsrv or the built-in WSL2 GUI support on Windows 11.

On macOS, install [XQuartz](#) to obtain an X11 environment, then proceed as on Linux.

Once logged in, verify the environment:

```
module load compiler openmpi
export SYSTYPE=Cast
mpicxx --version           # should print the Intel compiler version
```

5.3 Step 1: Directory Setup

```
mkdir MyName
mkdir MyName/Tutorial00
mkdir MyName/Tutorial00/Experiment1
cd MyName/Tutorial00/Experiment1
```

5.4 Step 2: Get the Code and Configuration Files

```
# Clone the code repository
git clone https://gitlab.lrz.de/MAGNETICUM/Hydro-OpenGadget3

# Get the default Config.sh for the tutorial
cd Hydro-OpenGadget3
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Config.sh
cd ..

# Get the parameter file and job script
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/box.param
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/runme.sh

# Get the Python reader
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Python/g3read.py
```

5.5 Step 3: Config.sh

The default Config.sh for T0 enables periodic boundaries, disables gravity, and selects the Wendland C2 kernel. No changes are needed for the first run:

```
PERIODIC
NOGRAVITY
WENDLAND_C2_KERNEL
```

Listing 5.1: Config.sh for Tutorial T0b — no changes needed from default.

Compile the code:

```
cd Hydro-OpenGadget3
make -j
cd ..
```

Wait for Linking Gadget3 to appear. The compile takes 1–2 minutes.

5.6 Step 4: Initial Conditions

Download the prepared IC file (a $10 \times 10 \times 10$ cubic lattice with a small x -velocity):

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/box.ic
```

This is a binary Format 2 file containing 1000 gas particles (type 0) in a unit box $[0, 1]^3$ with uniform density $\rho = 1$, internal energy $u = 0.05$, and velocity $v_x = 0.1$. The box size is 1 code length unit.

Figure 5.1 shows the initial particle layout: a perfect $10 \times 10 \times 10$ cubic lattice at $t = 0$.

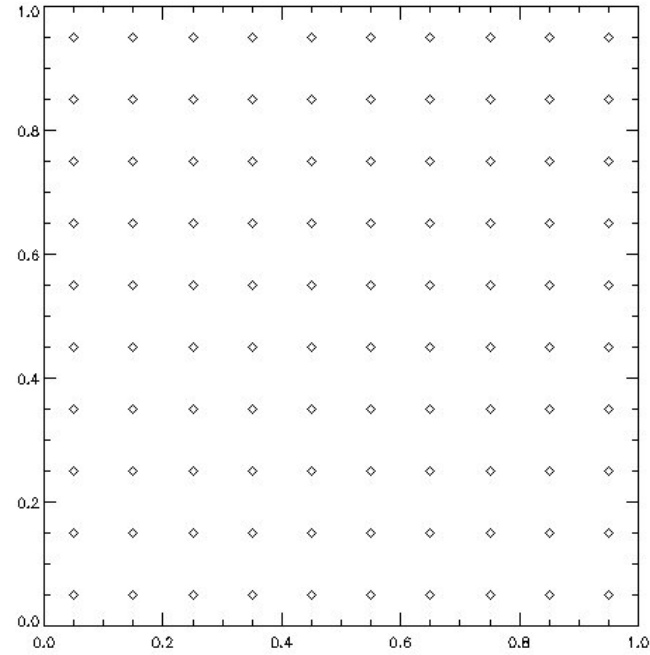


Figure 5.1: Initial conditions for Tutorial T0 (frame 000, $t = 0$), projected onto the x - y plane. The 1000 gas particles are arranged on a perfect $10 \times 10 \times 10$ cubic lattice with equal spacing $\Delta x = 0.1$ in a unit box. All particles have the same density, internal energy, and a uniform x -velocity $v_x = 0.1$. As the simulation runs, the particles advect to the right and the box boundary (with periodic boundaries) keeps them in the domain.

Alternatively, generate the IC yourself using Python. This is good practice: in later tutorials you will always write your own IC generator:

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Python/
ic_header
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Python/
setup_grid.py
python3 setup_grid.py
```

The script `setup_grid.py` places $10 \times 10 \times 10$ particles on a cubic lattice and writes `box.ic`. Inspect its source: it is the simplest example of an IC generator and the template for all future setups.

5.7 Step 5: Parameter File

The default `box.param` requires no changes for the first run. The key settings:

<code>InitCondFile</code>	<code>box.ic</code>	<i>% the IC file just downloaded</i>
<code>OutputDir</code>	<code>./output</code>	<i>% where snapshots will be written</i>
<code>SnapshotFileBase</code>	<code>snap</code>	
<code>TimeBegin</code>	<code>0.0</code>	
<code>TimeMax</code>	<code>1.0</code>	<i>% run for 1 code time unit</i>
<code>TimeBetSnapshot</code>	<code>0.01</code>	<i>% 100 snapshots total</i>
<code>PeriodicBoundariesOn</code>	<code>1</code>	<i>% matches PERIODIC in Config.sh</i>

```
UnitLength_in_cm      3.085678e21      % 1.0 kpc/h
UnitMass_in_g         1.989e43         % 1e10 Msun
UnitVelocity_in_cm_per_s 1e5          % 1 km/s

SofteningGas          0.2              % 20% of mean inter-particle spacing
SofteningGasMaxPhys   0.2
```

Listing 5.2: Key parameter file settings for Tutorial T0.

Create the output directory:

```
mkdir output
```

5.8 Step 6: Running the Simulation

Edit `runme.sh` to use a unique job name and minimal resources (1000 particles needs only 1 core):

```
# In runme.sh, change:
#SBATCH --job-name=MyName_T0_exp1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
```

Submit and watch:

```
sbatch runme.sh
tail -f slurm-*.out
```

The simulation should complete in under 30 seconds. You will see lines like:

```
Running on 1 MPI tasks
Using 1 OpenMP threads
...
Begin Step 0, Time: 0, Nf = 1000, ...
...
done with simulation
```

After completion, check the output directory:

```
ls output/
# snap_000 snap_001 snap_002 ... snap_099 snap_100
```

Also check `parameters-usedvalues` to see the full parameter set.

5.9 Step 7: Visualising the Results

5.9.1 Single-frame plot

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Python/
    show_single.py
python3 show_single.py
display frame_000.png      # or: eog frame_000.png
```

You should see 1000 particles arranged on a regular grid. The minimal Python script to do the same thing from scratch:

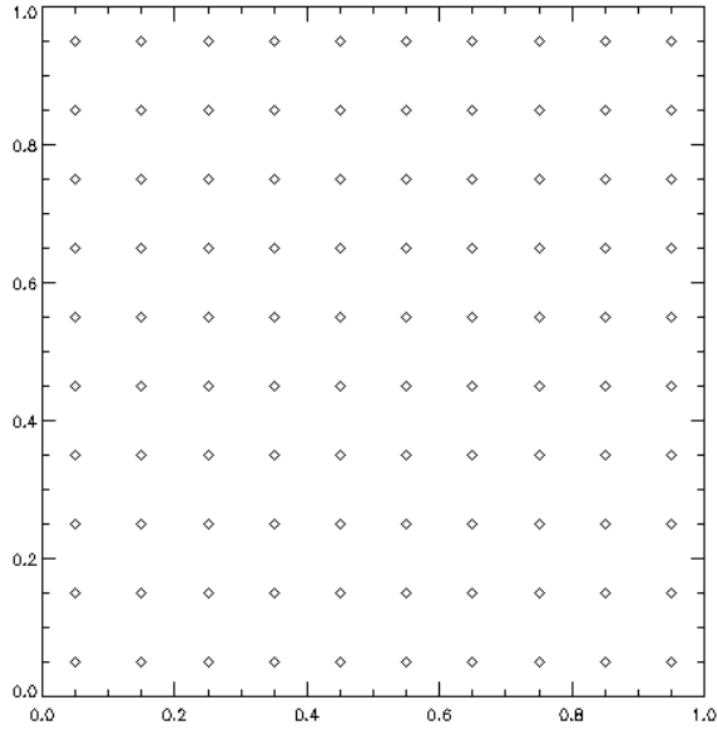


Figure 5.2: Sample result of the single-frame plot). black diamonds: the 1000 particles arranged on a regular grid.

```

1 import g3read as g3
2 import matplotlib.pyplot as plt
3
4 f = g3.GadgetFile('output/snap_000')
5 pos = f.read_new('POS ', 0) # type 0 = gas; shape (1000, 3)
6
7 fig, ax = plt.subplots(figsize=(6, 6))
8 ax.scatter(pos[:, 0], pos[:, 1], s=10, c='steelblue')
9 ax.set_xlim(0, 1)
10 ax.set_ylim(0, 1)
11 ax.set_xlabel('x [code units]')
12 ax.set_ylabel('y [code units]')
13 ax.set_title(f'T = {f.header.time:.3f}')
14 plt.tight_layout()
15 plt.savefig('frame_000.png', dpi=150)
16 print('Written frame_000.png')

```

Listing 5.3: Minimal single-frame visualisation.

5.9.2 Animation

```

wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Python/show.py
python3 show.py
ffplay -loop 0 basic_animation.mp4

```

You should see the particle grid drifting uniformly to the right and wrapping around (periodic boundaries). This confirms the simulation ran correctly.

To build the animation yourself:

```

1 import g3read as g3
2 import matplotlib.pyplot as plt
3 import matplotlib.animation as animation
4 import glob, numpy as np
5
6 snaps = sorted(glob.glob('output/snap_???'))
7 fig, ax = plt.subplots(figsize=(6, 6))
8
9 def update(frame):
10     ax.cla()
11     f = g3.GadgetFile(snaps[frame])
12     pos = f.read_new('POS ', 0)
13     ax.scatter(pos[:, 0], pos[:, 1], s=8, c='steelblue')
14     ax.set_xlim(0, 1); ax.set_ylim(0, 1)
15     ax.set_title(f'T = {f.header.time:.3f}')
16
17 ani = animation.FuncAnimation(fig, update, frames=len(snaps), interval
18                               =50)
19 ani.save('basic_animation.mp4', writer='ffmpeg', fps=20, dpi=100)
20 print('Saved basic_animation.mp4')

```

Listing 5.4: Creating an animation with matplotlib.

5.9.3 Other languages

The tutorial repository also provides equivalent examples in IDL, C, Fortran, and Julia. For C:

```

wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/C/gadget_io.h
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/C/show_single.
c
icx show_single.c
./a.out > data.txt
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/C/show_single.
plt
gnuplot show_single.plt

```

For Julia:

```

wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Julia/
show_single.jl
julia show_single.jl
display frame_000.png

```

Key commands for this tutorial

```

git clone https://gitlab.lrz.de/MAGNETICUM/Hydro-OpenGadget3
cd Hydro-OpenGadget3 && wget ../Config.sh && make -j && cd ..
wget ../box.param ../runme.sh ../g3read.py ../box.ic
mkdir output
# Edit runme.sh: --job-name=MyName_T0, --ntasks=1, --cpus-per-
task=1
sbatch runme.sh
python3 show.py && ffplay -loop 0 basic_animation.mp4

```

Exercise T0 — Exploration

1. Run the simulation with `ntasks=2`, `cpus-per-task=2`. Does the final result change? Compare the wall-clock time from the SLURM log (`squeue` shows elapsed time, or check the `cpu.txt` file).
2. Modify the visualisation script to colour particles by their x -velocity. Since all particles have the same velocity in this run, you should see a uniform colour. In T1 and later, the velocity field will be more interesting.
3. Open `parameters-usedvalues`. Find the parameter `DesNumNgb`. What value did the code use as the default for the Wendland C2 kernel?
4. (Programming) Generate the IC yourself. The Python example `setup_grid.py` places particles on a cubic lattice and gives them a uniform x -velocity. Modify it to give each particle a velocity $v_x = A \sin(2\pi x/L)$ where $A = 0.1$ and $L = 1$. Run the simulation and observe what happens to the sinusoidal perturbation. This is the beginning of sound wave physics (Tutorial T6).

Chapter 6

Tutorial T1: Free-fall Collapse

T1 introduces gravitational physics for the first time. A sphere of collisionless dark matter particles collapses under its own gravity. The simulation measures the free-fall time directly and compares it with the analytical prediction. It also introduces the key gravitational parameters: softening length, force accuracy, and the gravitational timestep.

6.1 Physical Background

6.1.1 Free-fall time

A pressureless, uniform sphere of density ρ_0 collapses under its own gravity in a time independent of its radius:

$$t_{\text{ff}} = \sqrt{\frac{3\pi}{32 G \rho_0}}. \quad (6.1)$$

This is the *free-fall time*. It depends only on the initial density. Denser systems collapse faster; the free-fall time scales as $t_{\text{ff}} \propto \rho_0^{-1/2}$.

Worked example — galaxy cluster. A cluster of mass $M \sim 10^{14} M_{\odot}$ forms from a region of the universe at mean cosmic density $\bar{\rho} = \Omega_m \rho_{\text{crit}} \simeq 1.2 \times 10^{-30} \text{ g cm}^{-3}$ at $z = 0$. In the standard code unit system (kpc/h, $10^{10} M_{\odot}$, km/s) with $G_{\text{code}} \approx 43.009$:

$$\rho_0 \approx 4.7 \times 10^{-8} \quad [\text{code}], \quad t_{\text{ff}} \approx 73 \quad [\text{code time}] \approx 71 \text{ Gyr}.$$

This is about five Hubble times, which is why the largest galaxy clusters are still assembling today.

Worked example — molecular cloud. A dense molecular cloud core with $n_{\text{H}_2} \sim 10^4 \text{ cm}^{-3}$, mean molecular weight $\mu = 2$, gives $\rho_0 \simeq 3.3 \times 10^{-20} \text{ g cm}^{-3}$ and $t_{\text{ff}} \approx 0.1 \text{ Myr}$. This is consistent with the observed timescale for star formation in dense molecular cloud cores.

Tip

Before running any simulation, compute t_{ff} analytically from the IC density and set `TimeMax` $\simeq 1.5\text{--}2 t_{\text{ff}}$. This ensures you capture the full collapse and the post-collapse bounce.

6.1.2 Hierarchical structure formation

Equation (6.1) directly explains the hierarchical assembly of cosmic structure. The largest structures (galaxy clusters, $\rho \sim \bar{\rho}$) have the longest t_{ff} : they form last. Smaller overdense

regions (galaxies, $\rho \gg \bar{\rho}$) collapse first. This *bottom-up* or *hierarchical* picture is a fundamental prediction of the Λ CDM model.

6.2 Changes from T0

T1 differs from T0 in three essential ways:

1. **Gravity is on:** remove NOGRAVITY from Config.sh. Self-gravity is now the dominant force.
2. **Boundaries are isolated:** remove PERIODIC. The simulation box is open; particles can fall inward without wrapping around.
3. **Particles are dark matter (type 1), not gas (type 0):** the IC has `npart[1] = N` with no gas particles, no U block, and no SPH forces. When reading snapshots, use `read_new('POS ', 1)`.

6.3 Config.sh

```
#PERIODIC           % disabled: isolated boundary conditions
#NOGRAVITY          % disabled: self-gravity is active
WENDLAND_C4_KERNEL
% Optional: write gravitational potential to snapshot
% EVALPOTENTIAL
```

Listing 6.1: Config.sh for Tutorial T1. Gravity is enabled; boundaries are isolated.

Always recompile after any change to Config.sh:

```
cd Hydro-OpenGadget3 && make -j && cd ..
```

6.4 Parameter File

```
InitCondFile          box.ic

% Isolated boundaries (matches #PERIODIC in Config.sh)
PeriodicBoundariesOn  0

% Set TimeMax to about 1.5 * t_ff (compute analytically first)
TimeMax               50.0
TimeBetSnapshot       0.5

% Standard cosmological units
UnitLength_in_cm      3.085678e21    % 1.0 kpc/h
UnitMass_in_g         1.989e43       % 1e10 Msun
UnitVelocity_in_cm_per_s 1e5         % 1 km/s
% => time unit = 0.978 Gyr, G_code = 43.009

% Dark matter softening [code length units = kpc/h]
% Rule of thumb: eps ~ mean inter-particle spacing / 30
SofteningHalo         20
SofteningHaloMaxPhys  20

% Tree force accuracy (default is adequate; see Experiment 2)
ErrTolForceAcc        0.005
```

Listing 6.2: Key parameter changes relative to T0.

Important

The gravitational timestep is $\Delta t \propto \sqrt{\epsilon/|a|}$. If `SofteningHalo` is too small, the timestep near the collapse centre becomes vanishingly small and the run stalls. If it is too large, the force is over-smoothed and the collapse is artificially slowed. For the tutorial IC with ~ 1000 particles in a sphere of radius ~ 50 kpc/ h , the value 20 kpc/ h is appropriate.

6.5 Initial Conditions

6.5.1 Using the provided IC

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T01/box.ic
```

This file contains dark matter particles (type 1). Verify the content:

```
1 import g3read as g3
2 f = g3.GadgetFile('box.ic')
3 print('N_gas :', f.header.npart[0])      # should be 0
4 print('N_DM  :', f.header.npart[1])      # should be N
5 print('BoxSize:', f.header.BoxSize)
```

6.5.2 Generating the IC from scratch (bonus)

Starting from the $10 \times 10 \times 10$ lattice of T0, select particles inside a sphere:

```
1 import numpy as np, struct
2
3 # Build cubic lattice
4 N_side = 10
5 L       = 1.0
6 coords = np.linspace(0, L, N_side, endpoint=False) + L/(2*N_side)
7 X, Y, Z = np.meshgrid(coords, coords, coords)
8 pos_all = np.column_stack([X.ravel(), Y.ravel(), Z.ravel()])
9
10 # Select sphere of radius R around box centre
11 centre = np.array([0.5, 0.5, 0.5])
12 r       = np.sqrt(((pos_all - centre)**2).sum(axis=1))
13 R       = 0.45
14 pos     = (pos_all[r < R]).astype(np.float32)
15 N       = len(pos)
16
17 vel     = np.zeros((N, 3), dtype=np.float32)      # zero initial
18         velocities
19 ids     = np.arange(1, N+1, dtype=np.int32)
20 mass    = np.ones(N, dtype=np.float32) * (1.0 / N)
21
22 # Header: set npart[1] = N, npart[0..0,2..5] = 0
23 # No U block (dark matter has no thermal energy)
24 # Write using the write_block() function from Chapter 4
25 print(f'Sphere: N = {N}, R = {R}, density = {N * (1.0/N) / (4*np.pi/3*
26         R**3):.4f}')
```

Listing 6.3: Creating a uniform-density sphere of dark matter.

6.6 Running the Simulation

Edit `runme.sh`:

```
#SBATCH --job-name=MyName_T1_collapse
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4 % 4 OpenMP threads accelerate the tree walk
```

```
mkdir output
sbatch runme.sh
tail -f slurm-*.out
```

The log shows the system collapsing; the time steps become smaller as the density increases near the centre. A typical run with 1000 particles takes 2–5 minutes.

You can start plotting intermediate results while the run is in progress — OpenGADGET3 writes each snapshot atomically when the output time is reached.

6.7 Visualisation and Analysis

6.7.1 Animating the collapse

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Python/
    show_cloud.py
python3 show_cloud.py
ffplay -loop 0 basic_animation.mp4
```

You should see 1000 dark matter particles converging to a central point, then dispersing slightly in a “bounce” as the particles pass through the centre (since there is no gas pressure, the collapse is collisionless and particles simply fly through each other, eventually virialising).

Figure 6.1 shows a representative frame during the collapse: the particles have fallen inward from their initial uniform sphere and are converging on the centre, forming the characteristic dense core visible as the tight red cluster at the origin.

A minimal scatter-plot animation:

```
1 import g3read as g3, numpy as np
2 import matplotlib.pyplot as plt, matplotlib.animation as animation,
  glob
3
4 snaps = sorted(glob.glob('output/snap_???'))
5 fig, ax = plt.subplots(figsize=(6, 6))
6
7 def update(i):
8     ax.cla()
9     f = g3.GadgetFile(snaps[i])
10    pos = f.read_new('POS ', 1) # type 1 = dark matter
11    ax.scatter(pos[:, 0], pos[:, 1], s=4, c='orangered', alpha=0.6)
12    ax.set_title(f'T = {f.header.time:.2f} [code time]')
13    # Auto-scale to current extent
14    margin = pos[:, :2].std() * 3
15    c = pos[:, :2].mean(axis=0)
16    ax.set_xlim(c[0]-margin, c[0]+margin)
17    ax.set_ylim(c[1]-margin, c[1]+margin)
18
19 ani = animation.FuncAnimation(fig, update, frames=len(snaps), interval
    =80)
```

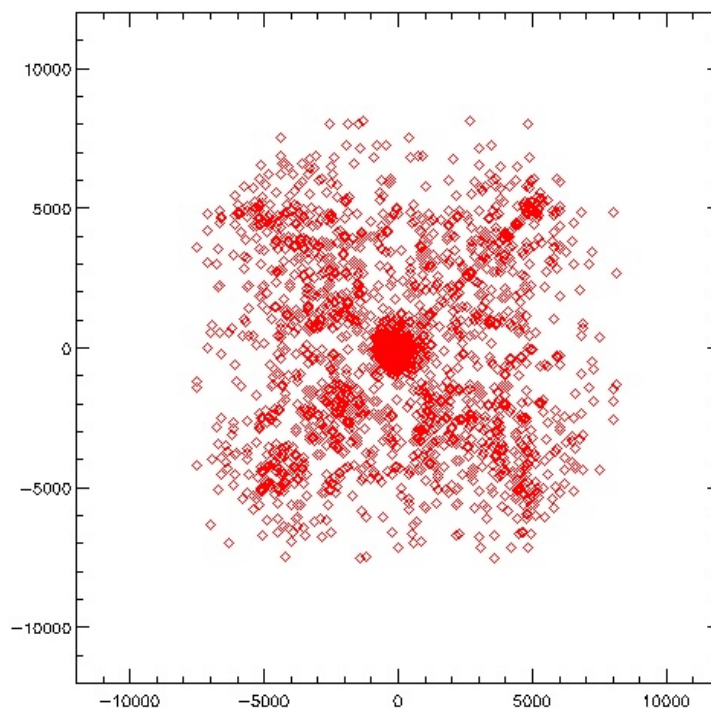


Figure 6.1: Snapshot from the Tutorial T1 dark matter collapse animation (projection onto the x - y plane). Red diamonds: the 1000 collisionless dark matter particles, shown during the infall phase. The dense core at the origin is the forming “halo”; particles at large radii are still falling in. The units on the axes are code length (kpc/ h in a typical galaxy simulation). After passing through the centre, the particles disperse and the system virialises. This is the direct N-body analogue of the analytical free-fall problem.

```
20 ani.save('collapse.mp4', writer='ffmpeg', fps=15, dpi=100)
```

Listing 6.4: Animating dark matter collapse.

6.7.2 Measuring the free-fall time

```
1 import g3read as g3, numpy as np, glob, matplotlib.pyplot as plt
2
3 snaps = sorted(glob.glob('output/snap_???'))
4 times = []
5 R_max = []
6 R_mean = []
7
8 centre = np.array([0.5, 0.5, 0.5]) % box centre; adjust if IC is
   different
9
10 for sp in snaps:
11     f = g3.GadgetFile(sp)
12     pos = f.read_new('POS ', 1)
13     r = np.sqrt(((pos - centre)**2).sum(axis=1))
14     times.append(f.header.time)
15     R_max.append(r.max())
16     R_mean.append(r.mean())
17
18 times = np.array(times)
```

```

19 R_max = np.array(R_max)
20 R_mean = np.array(R_mean)
21
22 # Free-fall time: time of minimum R_max
23 i_ff = np.argmin(R_max)
24 t_ff_sim = times[i_ff]
25 print(f'Simulated t_ff = {t_ff_sim:.2f} code time = {t_ff_sim*0.978:.2f} Gyr')
26
27 fig, ax = plt.subplots(figsize=(8, 4))
28 ax.plot(times, R_max, label=r'$R_{\mathrm{max}}$')
29 ax.plot(times, R_mean, label=r'$\langle r \rangle$', ls='--')
30 ax.axvline(t_ff_sim, color='r', ls=':', lw=2,
31            label=f'$t_{\mathrm{ff}} = {t_ff_sim:.1f}$')
32 ax.set_xlabel('Time [code units]')
33 ax.set_ylabel('Radius [code length]')
34 ax.legend()
35 plt.tight_layout()
36 plt.savefig('freefall.png', dpi=150)

```

Listing 6.5: Extracting t_{ff} from the simulation.

6.7.3 Effect of the force accuracy parameter

Add `ErrTolForceAcc 0.001` to `box.param` ($5\times$ more accurate than the default) and rerun. Compare:

- The measured t_{ff} : does it change significantly?
- The wall-clock time: check `cpu.txt`, column “treewalk”.
- The first lines of `balance.txt`: the more accurate run has longer per-step times.

Key commands for this tutorial

```

% Config.sh: comment out PERIODIC and NOGRAVITY, then recompile
cd Hydro-OpenGadget3 && make -j && cd ..
wget ../T01/box.ic
% box.param: PeriodicBoundariesOn 0, SofteningHalo 20, TimeMax
~2*t_ff
mkdir output && sbatch runme.sh
% Analysis: read_new('POS ', 1) --- note type 1, not 0
python3 show_cloud.py

```

Exercise T1

1. **Analytical prediction.** Before running: open the IC with `g3read.py`, compute the sphere volume from `header.BoxSize` and the particle positions, derive ρ_0 , and compute t_{ff} in code units. Set `TimeMax` to $1.5 t_{\text{ff}}$.
2. **Measurement.** After running: measure t_{ff} from the simulation. What is the percentage discrepancy from the analytical value? (Expect $\lesssim 5\%$ for 1000 particles with the default softening.)
3. **Cosmological context.** Use equation (6.1) to estimate t_{ff} for (a) a galaxy cluster at $z = 0$, (b) a typical L_* galaxy at $z = 3$, and (c) a molecular cloud. Compare with the corresponding age of the universe.
4. **Softening dependence.** Reduce `SofteningHalo` by a factor of 10. What

happens to the timestep distribution? Check `cpu.txt`. Does the measured t_{ff} change?

5. (**Advanced**) Create the IC yourself from a spherical selection of the T0 lattice. Verify that the density is uniform and that the measured t_{ff} matches the analytical prediction for that density.

Chapter 7

Tutorial T2: Stream Lines and Cosmological Flows

T2 introduces 2D simulations and the computation of stream lines and flow lines from a particle velocity field. The physical context is the large-scale velocity field of the universe and the T2b cosmological snapshot, which reveals the structure of voids, filaments, and cluster attractors.

7.1 Physical Background

7.1.1 Stream lines vs flow lines

A *stream line* is a curve everywhere tangent to the instantaneous velocity field: it shows where a massless tracer would move if the flow were frozen. A *flow line* (pathline) traces the actual trajectory of a fluid element over time. They coincide only for steady flows.

7.1.2 Cosmological context

The large-scale velocity field encodes the matter distribution. Converging regions correspond to forming structures (clusters, filaments); diverging regions to voids. The Laniakea supercluster was defined by [Tully et al. \(2014\)](#) as the basin of attraction of the peculiar velocity field. Tutorial T2b applies stream-line computation to a real cosmological snapshot, directly visualising the cosmic web.

7.1.3 Grid-based velocity interpolation and NGP/CIC

Given N particles with positions $\mathbf{r}_i$ and velocities $\mathbf{v}_i$, the velocity field on a regular grid is estimated by assigning particles to cells. Figure 7.1 illustrates the geometry.

7.2 Config.sh

```
PERIODIC
NOGRAVITY
WENDLAND_C2_KERNEL
TWO DIMS % 2D: forces and density estimates in the z=0 plane
```

Listing 7.1: Config.sh for Tutorial T2.

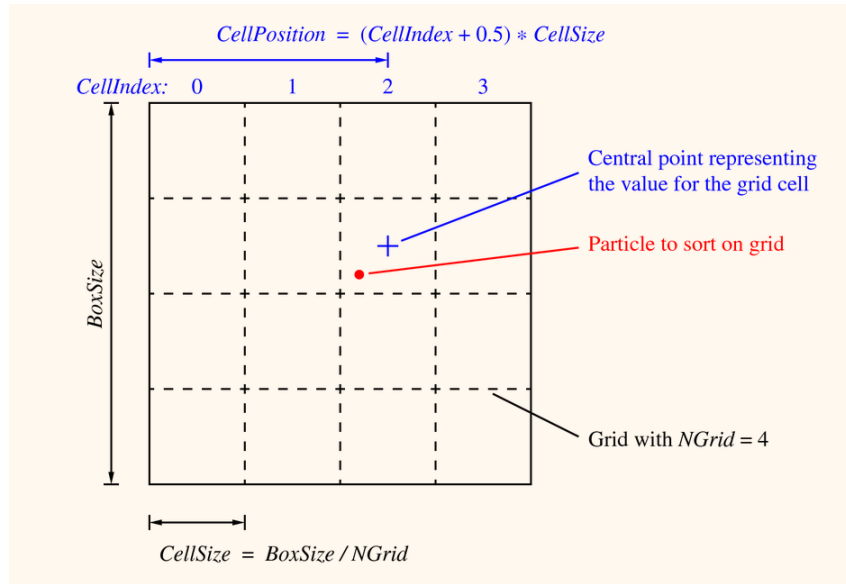


Figure 7.1: Grid-cell geometry for velocity assignment (NGP/CIC). The grid index runs $0, 1, 2, 3, \dots$ with cell centres at $x_{cell} = (CellIndex + 0.5) \times CellSize$. The blue cross marks the cell centre (“the value for the grid cell”) and the red dot the particle position. In NGP (Nearest-Grid-Point), the particle contributes entirely to the nearest cell. In CIC (Cloud-In-Cell), it contributes to the 8 (2D: 4) surrounding cells with weights proportional to the fractional overlap. TSC (Triangular-Shaped-Cloud) extends this to 27 (2D: 9) cells with a triangular kernel.

7.3 Parameter File

```
PeriodicBoundariesOn 1
TimeMax 5.0
TimeBetSnapshot 0.02
UnitLength_in_cm 3.085678e21
UnitMass_in_g 1.989e43
UnitVelocity_in_cm_per_s 1e5
SofteningGas 0.2
SofteningGasMaxPhys 0.2
```

7.4 Running and Computing Stream Lines

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T02/box.ic
mkdir output && sbatch runme.sh % ntasks=2, cpus-per-task=2
```

```
1 import g3read as g3, numpy as np, matplotlib.pyplot as plt
2
3 f = g3.GadgetFile('output/snap_050')
4 pos = f.read_new('POS ', 0)[: , :2]
5 vel = f.read_new('VEL ', 0)[: , :2]
6 L = f.header.BoxSize
7 Ngrid = 50; dx = L / Ngrid
8
9 % TSC velocity grid
10 vgrid = np.zeros((Ngrid, Ngrid, 2))
11 wgrid = np.zeros((Ngrid, Ngrid))
12 for i in range(len(pos)):
```

```

13     fc = pos[i] / dx
14     ic = (fc - 0.5).astype(int)
15     for ddx in range(-1, 2):
16         for ddy in range(-1, 2):
17             jx = (ic[0]+ddx) % Ngrid; jy = (ic[1]+ddy) % Ngrid
18             sx = fc[0]-(ic[0]+ddx+0.5); sy = fc[1]-(ic[1]+ddy+0.5)
19             wx = 0.75-sx**2 if abs(sx)<0.5 else 0.5*(1.5-abs(sx))**2
20             wy = 0.75-sy**2 if abs(sy)<0.5 else 0.5*(1.5-abs(sy))**2
21             vgrid[jx,jy] += vel[i]*wx*wy; wgrid[jx,jy] += wx*wy
22     nz = wgrid > 0; vgrid[nz] /= wgrid[nz, None]
23
24     % Trace stream lines
25     v_max = max(np.abs(vgrid).max(), 1e-10)
26     dt = 0.3*dx/v_max
27     fig, ax = plt.subplots(figsize=(8,8))
28     for x0 in np.linspace(dx/2, L-dx/2, 10):
29         for y0 in np.linspace(dx/2, L-dx/2, 10):
30             pts = [[x0, y0]]
31             for _ in range(150):
32                 px, py = pts[-1]
33                 ix = int(px/dx) % Ngrid; iy = int(py/dx) % Ngrid
34                 pts.append([(px+vgrid[ix,iy,0]*dt)%L, (py+vgrid[ix,iy,1]*
35                             dt)%L])
36             pts = np.array(pts)
37             ax.plot(pts[:,0], pts[:,1], 'k-', lw=0.6, alpha=0.7)
38     ax.set_xlim(0,L); ax.set_ylim(0,L)
39     plt.tight_layout(); plt.savefig('streamlines.png', dpi=150)

```

Listing 7.2: Stream line computation with TSC velocity grid.

7.5 T2b: Stream Lines in a Cosmological Snapshot

Tutorial T2b applies the stream-line method to a real cosmological simulation snapshot. The velocity field reveals the cosmic web structure: filaments, voids, and cluster attractors.

```

% Get and unpack the cosmological snapshot
cp ~/Hydro/snap_144.gz . && gunzip snap_144.gz

```

Figures 7.2–7.5 show four complementary visualisations of the velocity field computed from this snapshot.

Figure 7.6 shows the particle distribution at $t = 50$, with two shear layers (blue, moving in $+x$) separated by gas at rest (black). The Kelvin–Helmholtz instability is suppressed by SPH’s surface tension.

7.5.1 Stream lines: instantaneous velocity topology

Figures 7.7 show stream lines at two times. At $t = 50$ the topology is simple (parallel shear flow with small perturbations); by $t = 200$ the Kelvin–Helmholtz rolls have developed into large closed-orbit vortices.

7.5.2 Flow lines: particle trajectories

Flow lines track actual particle positions across snapshots. Figure 7.8 compares flow lines at the same two times. At early times flow lines and stream lines agree (the flow is nearly

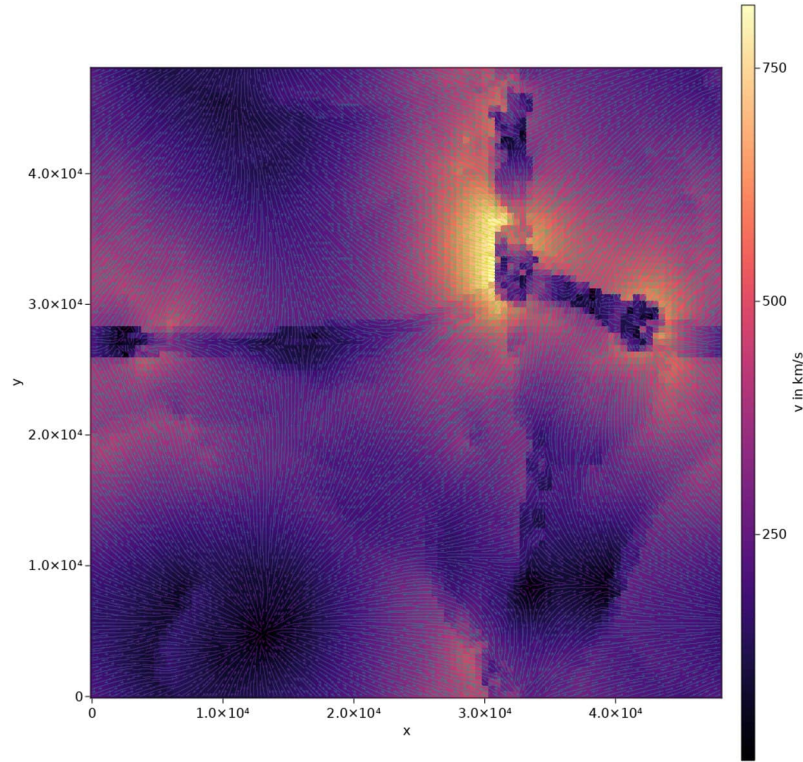


Figure 7.2: Line Integral Convolution (LIC) visualisation of the 2D projected velocity field from the cosmological snapshot (T2b). The LIC texture reveals the direction of flow at every point, while the colour scale encodes the velocity magnitude $|v|$ in km/s (yellow ≈ 750 km/s, black ≈ 0). The two bright convergence nodes (top-right quadrant and centre-right) are the potential wells of massive clusters; the dark filamentary structure passing through $y \approx 2.7 \times 10^4$ is a low-velocity coherent inflow region (a wall or filament). This plot is generated by binning particle velocities onto a grid and applying a standard LIC algorithm.

steady); at late times they diverge because vortices rotate the tracer particles away from the instantaneous stream-line direction.

7.5.3 TSC velocity grid and bilinear interpolation

Figure 7.9 shows the stream lines computed with TSC mass assignment and bilinear interpolation between grid nodes, overlaid on the SPH particle distribution.

Exercise T2

1. For the T2 simulation: compare stream lines and flow lines at snapshot 050. Do they agree? At which time do they begin to diverge?
2. For T2b: compute the 2D stream lines from the cosmological snapshot and reproduce a figure similar to Figure 7.3. Identify the cluster attractor(s) and count how many stream lines flow into each within a $5000 \text{ kpc}/h$ radius.
3. Compare NGP, CIC, and TSC assignment for the velocity grid. Which scheme gives smoother stream lines? Quantify by measuring the mean curvature of stream lines as a function of grid resolution.

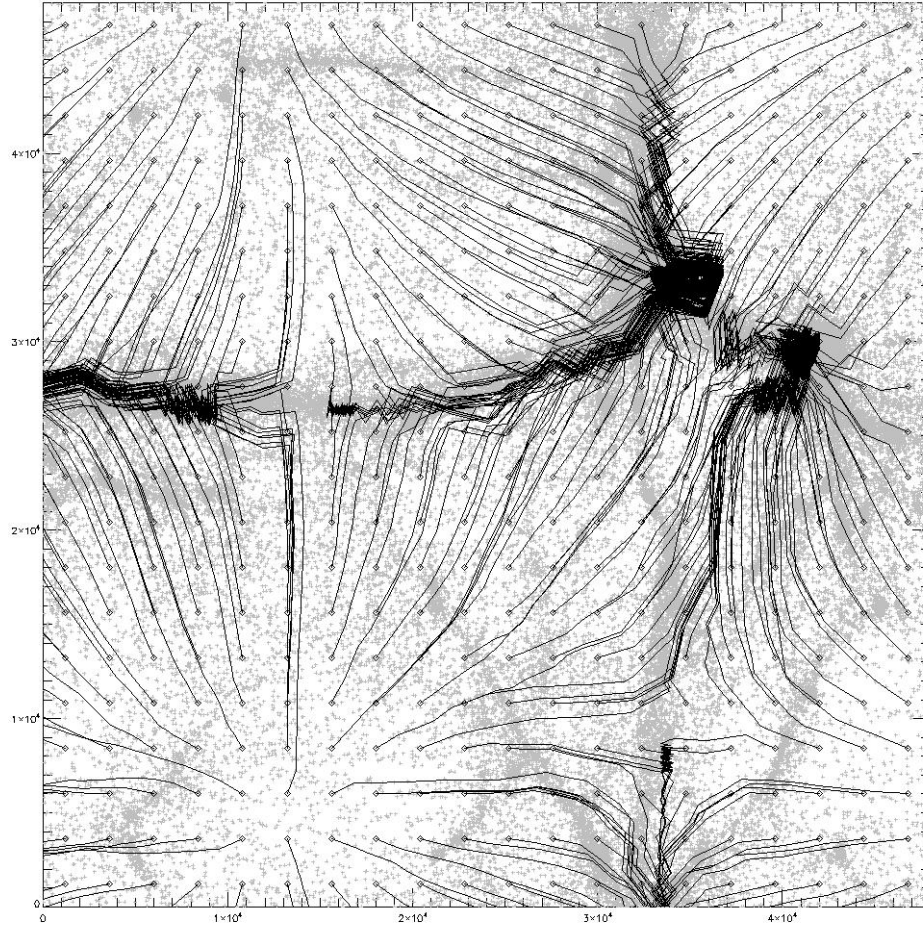


Figure 7.3: Stream lines computed from the 2D projected velocity field of the cosmological snapshot (T2b). Starting points are placed on a regular grid; each line is integrated forward using the grid-interpolated velocity until it exits the domain or the integration is terminated. The bright convergence nodes at $x \approx 2.8 \times 10^4$, $y \approx 3.5 \times 10^4$ (and nearby) are massive cluster attractors: all surrounding stream lines flow toward them. The complex topology of crossing and converging lines traces the filaments and walls of the cosmic web.

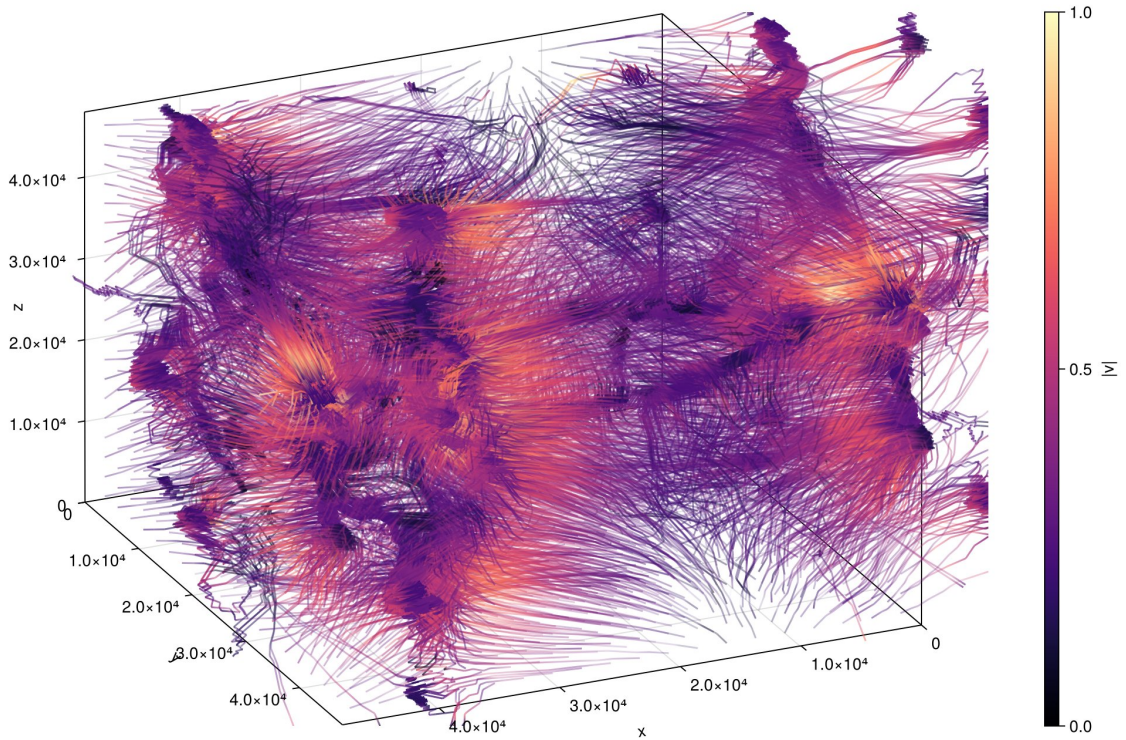


Figure 7.4: Three-dimensional stream lines through the cosmological snapshot, coloured by the local Mach number $\mathcal{M}$ (yellow = 1, black = 0). The 3D view reveals the full spatial extent of the velocity field and the tangled network of inflow streams feeding the cluster potential wells. Regions of high Mach number (yellow–orange) correspond to supersonic infall along filaments; low-Mach (dark blue–purple) regions correspond to the quiet interiors of voids. This visualisation uses Python’s `matplotlib` 3D streamplot functionality applied to a velocity field binned onto a $3 \times 3 \times 3$ Mpc/ h regular grid.

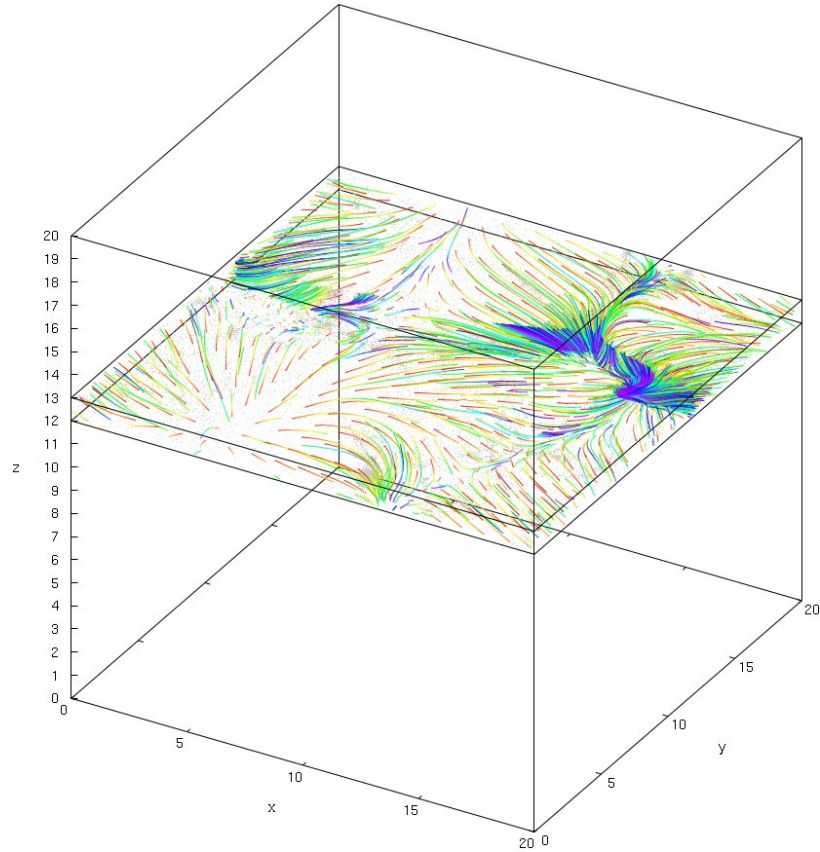


Figure 7.5: Stream lines projected onto a single z -slice through the cosmological volume (T2b), rendered in a 3D perspective view. The slice at $z \approx 13$ (in code units) shows the 2D velocity pattern in a horizontal plane. The rainbow colour coding indicates the local velocity magnitude. Two prominent attractors (cluster centres) are visible as convergence points where stream lines spiral inward. The peripheral lines show the large-scale coherent flows associated with the surrounding void regions.

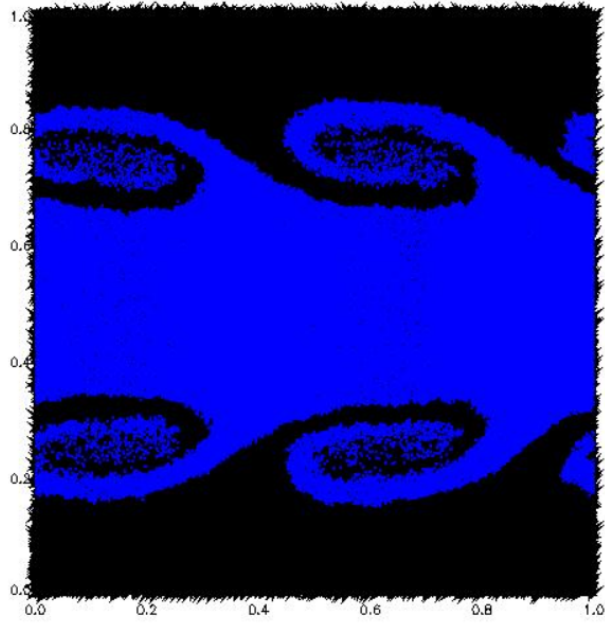
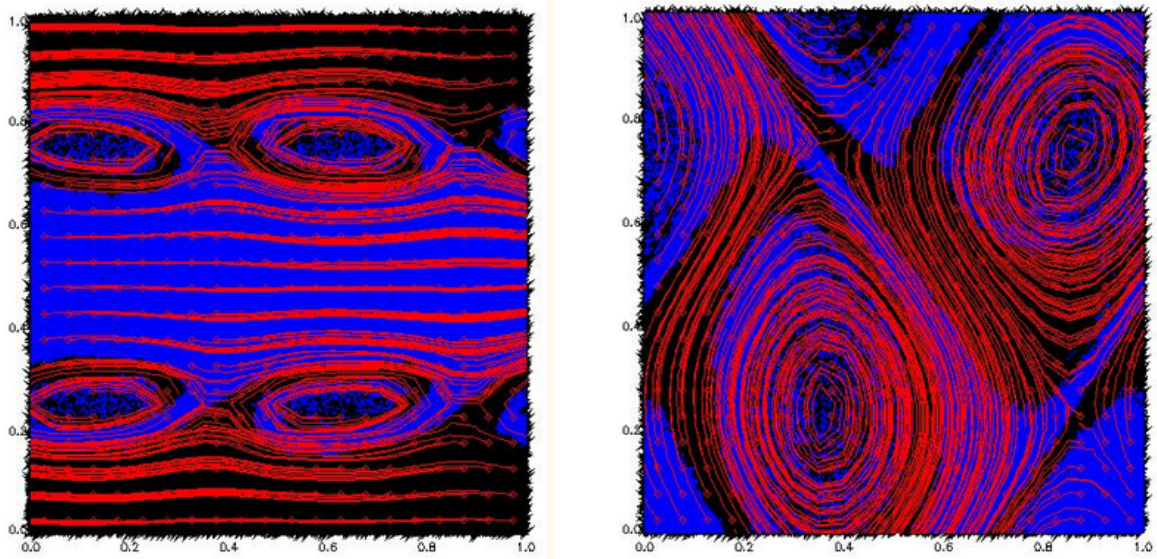


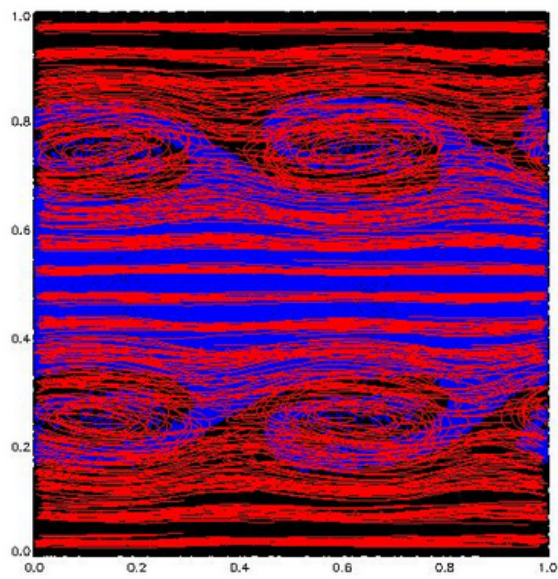
Figure 7.6: Particle positions in the T2 shear-flow simulation at $t = 50$. Blue: upper fluid layer (moving in $+x$). Black: lower fluid (moving in $-x$ or at rest). The shear interface at $y \approx 0.5$ remains sharp; the KHI is suppressed by SPH's effective surface tension at contact discontinuities.



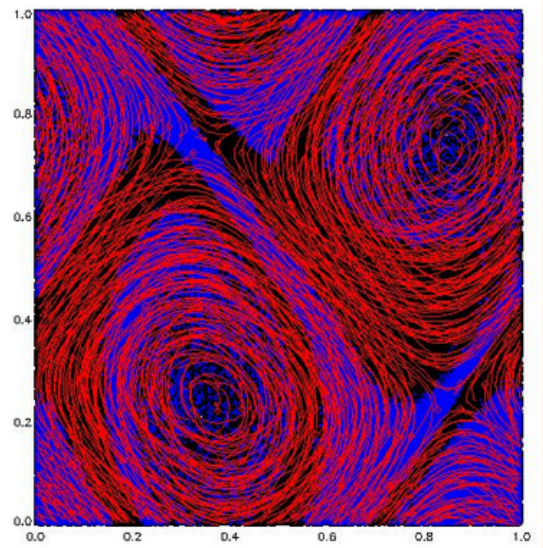
(a) Stream lines at $t = 50$. The flow is still nearly laminar: the upper (red) and lower (blue) streams move in opposite directions with small perturbations at the interface.

(b) Stream lines at $t = 200$. KHI vortices have grown into large closed-orbit structures (elliptical closed stream lines) separated by the surviving shear layers.

Figure 7.7: Stream lines (instantaneous velocity field) in the T2 shear-flow simulation at two times. Red: right-moving particles. Blue: left-moving. Note the transition from laminar to vortical topology between $t = 50$ and $t = 200$.



(a) Flow lines at $t = 50$. Nearly identical to the stream lines: the flow has not yet developed strong time dependence.



(b) Flow lines at $t = 200$. The trajectories now spiral around the vortex centres, mapping the history of particle motion rather than the instantaneous flow direction.

Figure 7.8: Flow lines (particle trajectories integrated over time) at $t = 50$ and $t = 200$ for the T2 shear-flow simulation.

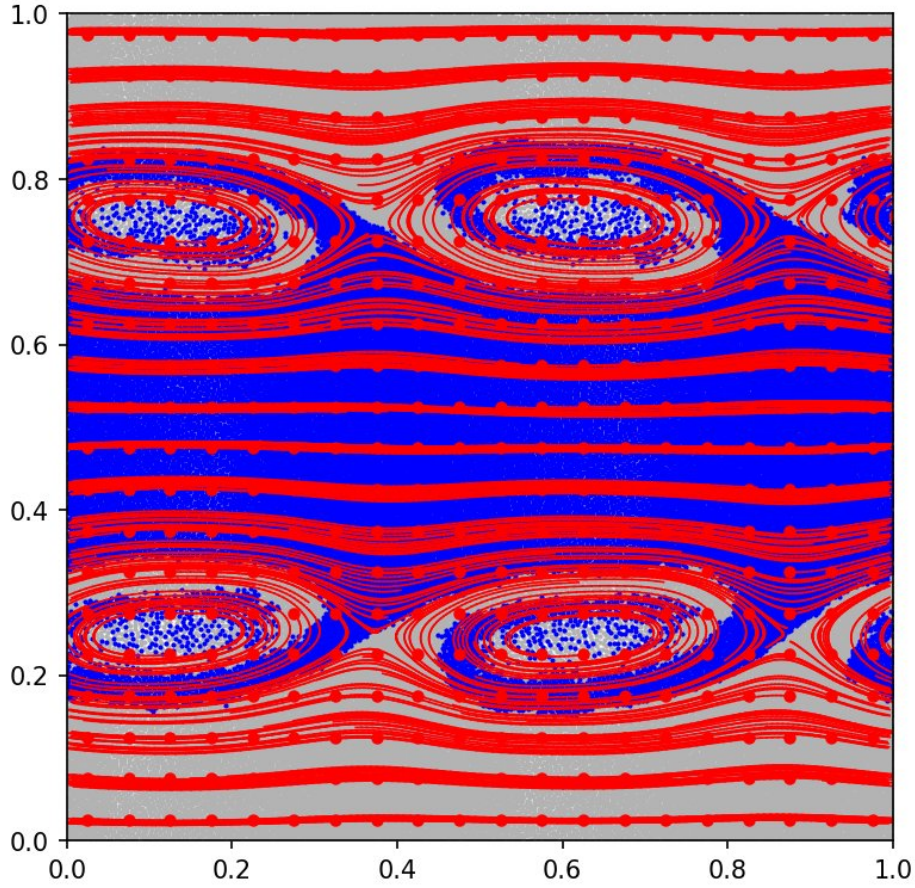


Figure 7.9: Stream lines at $t = 50$ computed using TSC velocity assignment and bilinear interpolation between grid nodes (Python implementation). Grey background: SPH particles colour-coded by v_x (red = right-moving, blue = left-moving). The stream lines reveal the same vortex topology as the IDL version in Figure 7.7, but with smoother curvature thanks to bilinear interpolation between grid nodes.

Chapter 8

Tutorial T3: Density Estimates and SPH Kernels

T3 is a systematic convergence study of the SPH density estimator. By running the same uniform-density box with different kernels, different neighbour numbers, and different initial particle distributions, you measure the bias and variance of the density estimate and learn how to design batch simulations. This tutorial reproduces the convergence study of [Dehnen and Aly \(2012\)](#).

8.1 Physical Background

The SPH density estimator (equation ??) introduces both a systematic bias and random scatter that depend on three factors:

- **Kernel order:** higher-order kernels (Wendland C4, C6) give smaller errors at the same neighbour count.
- **Neighbour count** N_{ngb} : more neighbours reduce scatter but increase the effective resolution length.
- **Particle distribution:** a glass reduces anisotropy errors compared to a cubic lattice.

8.2 Initial Particle Distributions

Figure 8.1 shows the three main distributions used in T3, projected onto the x - y plane. The cubic lattice (left) is perfectly regular but has strong preferred directions. The hexagonal close-packed (centre) is more isotropic. Both are uniform grids; only the glass (not shown here) achieves true isotropy through random relaxation.

Figure 8.2 shows a more comprehensive comparison of eight particle distributions rendered as 3D sphere-particle visualisations. The interior colour coding (rainbow scale) maps the kernel support volume or the local density estimate, revealing how accurately each distribution samples a uniform field.

```
% Cubic lattice 10x10x10 (1000 particles)
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/grid_10x10x10

% Hexagonal close-packed 12x12x12
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/
    packed_12x12x12
```

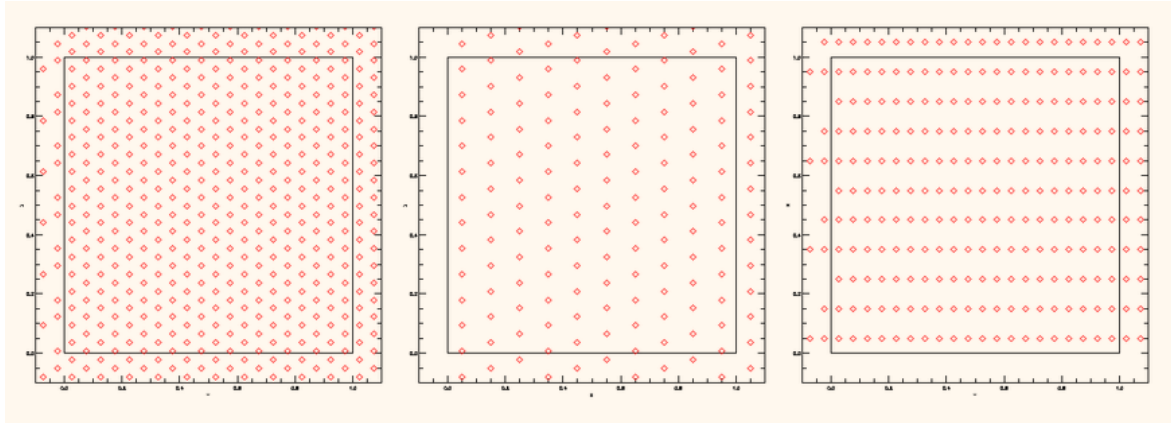


Figure 8.1: Three particle distributions used in Tutorial T3, projected onto the x - y plane (2D cross-sections of 3D distributions). *Left*: cubic lattice — uniform $10 \times 10 \times 10$ grid with equal spacing in all directions. Note the strong diagonal structure which introduces anisotropic errors in the SPH density estimate. *Centre*: hexagonal close-packed (HCP) lattice — denser packing with alternating row offsets, more isotropic than the cubic lattice but still structured. *Right*: sparser cubic lattice at lower resolution, showing how the particle spacing grows with fewer particles. All distributions have the same total mass and box volume.

```
% Glass 10x10x10 (1000 particles, relaxed)
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/glass_10x10x10
```

Listing 8.1: Download the three particle distribution files.

All have `BoxSize = 1` and uniform density $\rho_0 = 1$.

8.3 Running Multiple Simulations

Add `StopAfterNSteps 5` to the parameter file: the density is computed in the first step; there is no need to evolve the system further.

```
#!/bin/bash
KERNELS="STANDARD_KERNEL WENDLAND_C2_KERNEL WENDLAND_C4_KERNEL
WENDLAND_C6_KERNEL"
NNGBS="32 64 128 200"
DISTS="grid_10x10x10 glass_10x10x10"

for KERNEL in $KERNELS; do
  for NNGB in $NNGBS; do
    for DIST in $DISTS; do
      DIR="run_${KERNEL}_n${NNGB}_${DIST}"
      mkdir -p $DIR/output
      cp $DIST $DIR/box.ic

      cat > $DIR/Config.sh << EOF
PERIODIC
NOGRAVITY
${KERNEL}
EOF

      % Compile once per kernel
      if [ ! -f "${KERNEL}/Gadget3" ]; then
        mkdir -p ${KERNEL}
        cd Hydro-OpenGadget3
```

```

    cp ../$DIR/Config.sh . && make -j
    cp Gadget3 ../${KERNEL}/
    cd ..
fi

sed -e "s/NNGB_VALUE/$NNGB/" box_template.param > $DIR/box.param
echo "InitCondFile      box.ic"          >> $DIR/box.param
echo "StopAfterNSteps 5"                >> $DIR/box.param

cd $DIR
cp ../${KERNEL}/Gadget3 Hydro-OpenGadget3/Gadget3
sbatch ../runme_1cpu.sh
cd ..
done
done
done

```

Listing 8.2: Batch script for all kernel/Nngb/distribution combinations.

8.4 Analysis: Density Bias and Scatter

```

1 import g3read as g3, numpy as np, matplotlib.pyplot as plt
2 import itertools
3
4 true_rho = 1.0
5 kernels = ['STANDARD_KERNEL', 'WENDLAND_C2_KERNEL',
6           'WENDLAND_C4_KERNEL', 'WENDLAND_C6_KERNEL']
7 nngbs = [32, 64, 128, 200]
8 dists = ['grid_10x10x10', 'glass_10x10x10']
9
10 results = {}
11 for kernel, nngb, dist in itertools.product(kernels, nngbs, dists):
12     path = f'run_{kernel}_n{nngb}_{dist}/output/snap_000'
13     try:
14         f = g3.GadgetFile(path)
15         rho = f.read_new('RHO ', 0)
16         results[(kernel, nngb, dist)] = (
17             (rho.mean() - true_rho) / true_rho, % bias
18             rho.std() / true_rho % scatter
19         )
20     except:
21         pass
22
23 % Plot scatter vs Nngb for glass distribution
24 fig, ax = plt.subplots(figsize=(8, 5))
25 colors = {'STANDARD_KERNEL': 'C0', 'WENDLAND_C2_KERNEL': 'C1',
26          'WENDLAND_C4_KERNEL': 'C2', 'WENDLAND_C6_KERNEL': 'C3'}
27 for kernel in kernels:
28     y = [results.get((kernel, n, 'glass_10x10x10'), (np.nan, np.nan))
29          [1]
30          for n in nngbs]
31     label = kernel.replace('_', '').replace(' ', '')
32     ax.plot(nngbs, y, 'o-', label=label, color=colors[kernel])
33
34 ax.set_xlabel(r'$N_{\mathrm{nbg}}$'); ax.set_ylabel(r'$\sigma(\rho)/\rho_0$')
35 ax.set_title('Density scatter vs neighbours (glass distribution)')

```

```
35 ax.legend(fontsize=9); plt.tight_layout()  
36 plt.savefig('density_convergence.png', dpi=150)
```

Listing 8.3: Systematic comparison of density estimates.

Exercise T3

1. Run the simulation for cubic lattice and glass for three kernels and four neighbour numbers. Plot bias and scatter vs N_{ngb} . Which kernel reaches $\sigma(\rho)/\rho_0 < 0.01$ at the lowest N_{ngb} ?
2. From Figure 8.2, identify which distribution shows the most uniform interior colour. Does this match your quantitative result from the density scatter measurement?
3. Compare cubic lattice vs glass for the same kernel and N_{ngb} . By how much does the glass reduce the scatter? Is the bias also different?
4. (Advanced) Reproduce figure 3 of [Dehnen and Aly \(2012\)](#): plot $\sigma(\rho)/\rho_0$ vs N_{ngb} on a log-log scale for all kernels and compare the slopes.

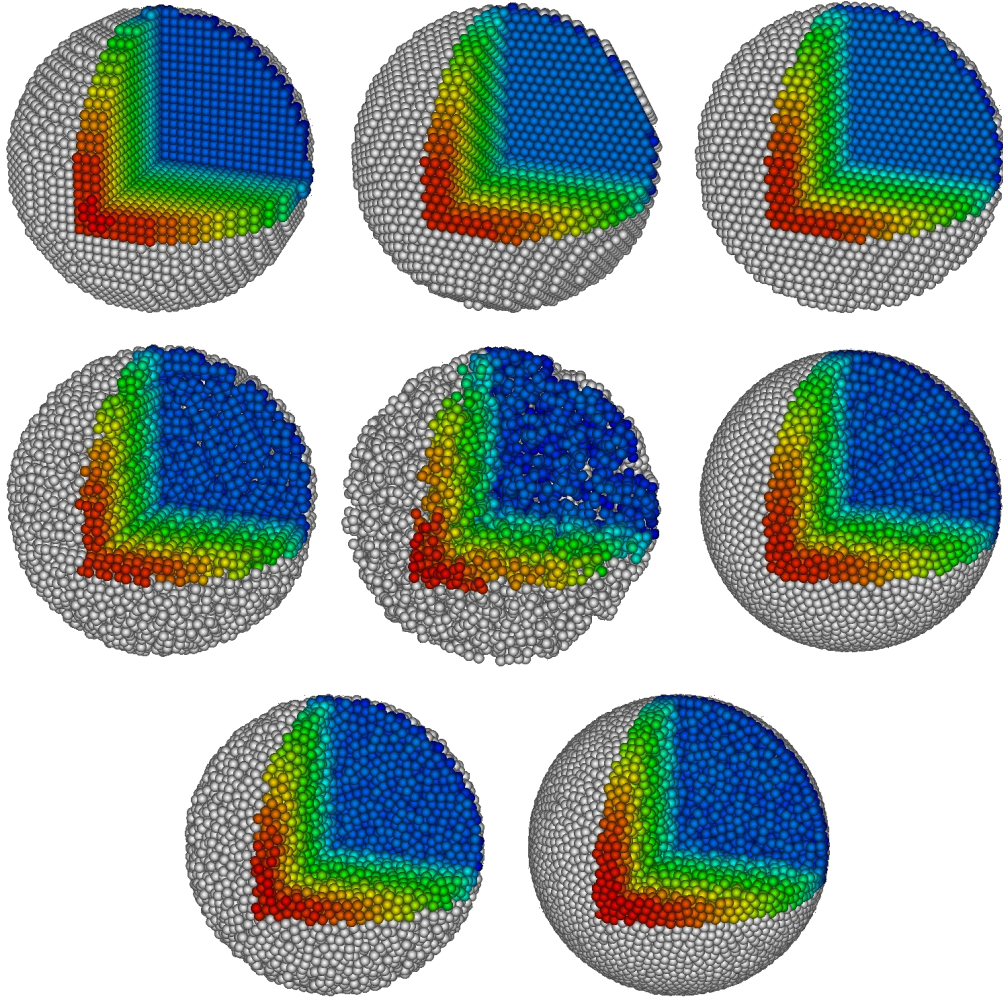


Figure 8.2: Eight particle distributions visualised as 3D sphere-particle renders (quarter sections removed to show interior structure). The colour scale (red = high, blue = low) shows the SPH density estimate normalised to the true uniform density ρ_0 . Ideal uniform sampling would show a single colour throughout. Deviations from uniform colour indicate density estimation errors. *Top row*: cubic lattice variants (regular grids); note the angular artefacts from the lattice symmetry. *Middle and bottom rows*: progressively more isotropic distributions (HCP, face-centred cubic, glass), which show more uniform colour inside the sphere, confirming lower density bias and scatter. The glass distribution (bottom right) gives the most uniform interior.

Chapter 9

Tutorial T4: The Riemann Problem

T4 is the key validation test for SPH shock capturing. Two half-spaces of gas at different densities and pressures are placed in contact; the numerical solution is compared with the exact analytical result. Three progressively refined setups reveal how kernel choice, particle distribution, and artificial conductivity each improve the quality of the solution.

9.1 Physical Background

The Riemann problem is the canonical one-dimensional shock tube. At $t = 0$, a membrane separating two uniform states is removed:

$$(\rho, P, v) = \begin{cases} (\rho_L, P_L, 0) & x < x_0, \\ (\rho_R, P_R, 0) & x > x_0. \end{cases} \quad (9.1)$$

For this tutorial: $(\rho_L, P_L) = (1, 1)$ and $(\rho_R, P_R) = (0.25, 0.1795)$ with $\gamma = 5/3$. The exact solution develops five distinct regions (see Section 1.2.2): a left rarefaction fan, a left contact state, a contact discontinuity, a right post-shock state, and a right shock.

9.1.1 Solving for P^* graphically

The contact-discontinuity pressure P^* satisfies the nonlinear equation $f(P^*) = f_L(P^*) + f_R(P^*) + (v_R - v_L) = 0$. A graphical approach plots $f_L(P)$ and $f_R(P)$ separately as functions of P ; the intersection gives P^* . Figure 9.1 shows this graphical solution for the tutorial initial conditions.

9.2 Config.sh: Three Versions

Version 1 — 2D with Wendland C2 (baseline):

```
PERIODIC
NOGRAVITY
WENDLAND_C2_KERNEL      % baseline: noisy but fast
TWO DIMS
LONG_X=40
LONG_Y=1
LONG_Z=1
```

Listing 9.1: Config.sh for T4, first run.

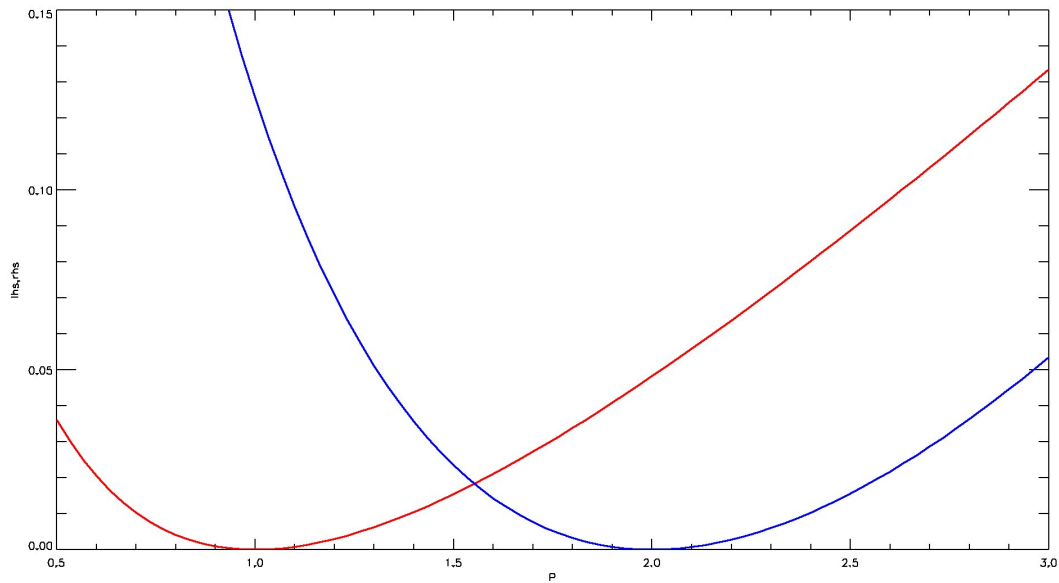


Figure 9.1: Graphical solution of the Riemann problem for Tutorial T4. The blue curve shows $f_L(P)$ (left rarefaction term) and the red curve shows $f_R(P)$ (right shock term) as functions of trial contact pressure P . The intersection gives $P^* \approx 0.303$. The tutorial asks you to reproduce this plot and solve for P^* numerically via Newton–Raphson.

Version 2 — Wendland C4 or C6, glass slab (improved):

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
TWO DIMS
LONG_X=40
LONG_Y=1
LONG_Z=1
```

Listing 9.2: Config.sh for T4, second run.

Version 3 — Add artificial conductivity (best):

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
TWO DIMS
LONG_X=40
LONG_Y=1
LONG_Z=1
ARTIFICIAL_CONDUCTIVITY % suppresses pressure blip at contact
discontinuity
```

Listing 9.3: Config.sh for T4, third run.

Each version requires a separate compilation. Run each in a separate directory.

9.3 Parameter File

```
PeriodicBoundariesOn 1
TimeMax 5.0
```

TimeBetSnapshot	0.1	
UnitLength_in_cm	1.0	% dimensionless units for this test
UnitMass_in_g	1.0	
UnitVelocity_in_cm_per_s	1.0	
SofteningGas	0.2	
SofteningGasMaxPhys	0.2	

9.4 Initial Conditions

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T04/box.ic
mkdir output && sbatch runme.sh
```

The IC places the contact discontinuity at $x_0 = L/2 = 20$ (in a 40×1 slab with `LONG_X=40`, `BoxSize=1`). The left half has $(\rho, P) = (1, 1)$ and the right half $(\rho, P) = (0.25, 0.1795)$. All particles have zero initial velocity and the entropy corresponding to their pressure and density via $A = P/\rho^\gamma$.

9.5 Progressive Improvement Strategy

Run all three versions and compare results at the same output time $t = 5$. This reveals the contributions of each ingredient to the final solution quality.

Step 1 — Wendland C2, cubic lattice. The solution is noisy: the cubic lattice has preferred directions that produce anisotropic density errors, making shock fronts jagged. The five regions are identifiable but with significant scatter.

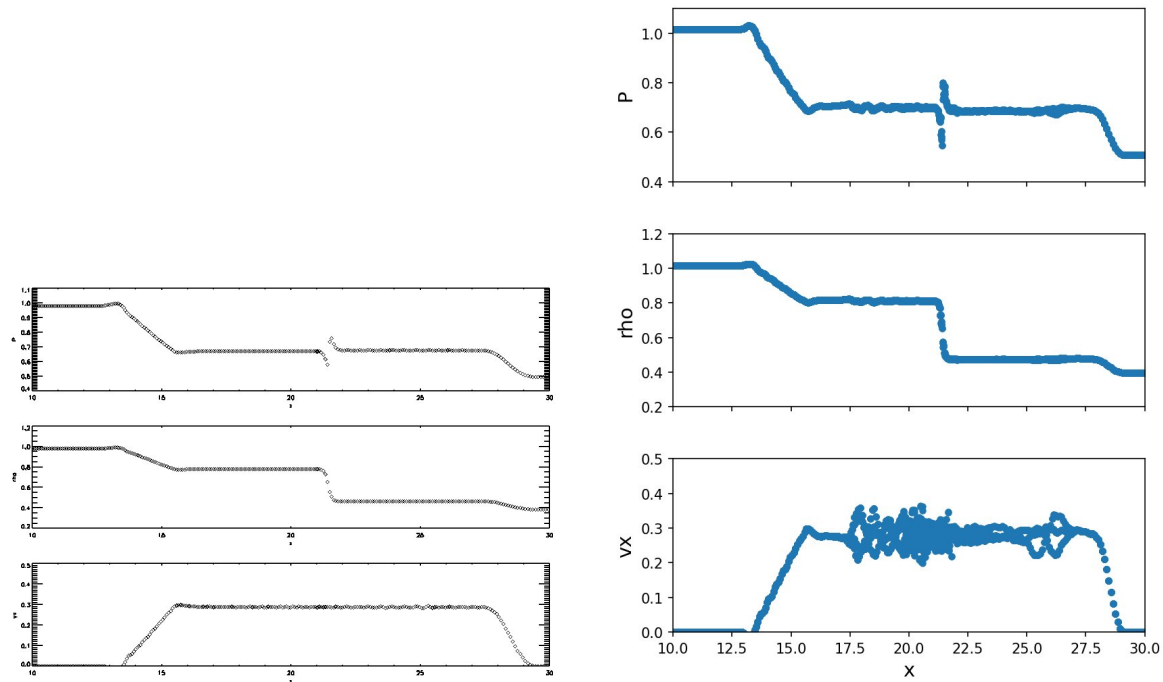
Step 2 — Wendland C4, glass slab. Substantially smoother. The rarefaction fan, contact discontinuity, and shock are all clearly resolved. A small *pressure blip* (a positive spike) persists exactly at the contact discontinuity: this is a known SPH artefact caused by the inability of standard SPH to diffuse entropy across a density jump (Section 1.6.7).

Step 3 — Wendland C4 + artificial conductivity. The pressure blip is strongly suppressed. This is the recommended configuration for production simulations involving strong contact discontinuities.

Figure 9.2 compares the same simulation output as rendered by the IDL solution script (light background) and by a Python script, showing that both approaches produce equivalent results.

9.6 Analysis

```
1 import g3read as g3, numpy as np, matplotlib.pyplot as plt
2
3 f = g3.GadgetFile('output/snap_030') % adjust snapshot number
4 pos = f.read_new('POS ', 0)
5 rho = f.read_new('RHO ', 0)
6 u = f.read_new('U ', 0)
7 vel = f.read_new('VEL ', 0)
8 x = pos[:, 0]
9 P = (5.0/3.0 - 1.0) * rho * u
10
11 idx = np.argsort(x)
12 x, rho, P, vx = x[idx], rho[idx], P[idx], vel[idx, 0]
13
14 fig, axes = plt.subplots(3, 1, figsize=(9, 9), sharex=True)
```



(a) IDL output: P , ρ , and v_x profiles at $t = 50$. (b) Python output: same simulation, same time.

Figure 9.2: Tutorial T4 results at $t = 50$. Three panels in each figure show (top) pressure P , (middle) density ρ , and (bottom) x -velocity v_x along the slab. The five regions of the Riemann solution are visible: rarefaction fan (left slope), left contact state (flat), contact discontinuity (density jump near $x \approx 21$), right post-shock state (flat), and right shock (right slope). The IDL and Python outputs are equivalent; both tools are provided with the tutorial.

```

15 for ax, data, ylabel in zip(axes, [P, rho, vx],
16                             [r'$P$', r'$\rho$', r'$v_x$']):
17     ax.plot(x, data, '.', ms=1.5, alpha=0.5, label='Simulation')
18     ax.set_ylabel(ylabel, fontsize=12)
19
20 % Analytical reference lines (for the standard IC, gamma=5/3, t=5):
21 % P* ~ 0.303, v* ~ 0.66 km/s, shock at x ~ 20 + 1.75*t
22 L = f.header.BoxSize * 40
23 t = f.header.time
24 x0 = L / 2 % initial contact position
25 x_shock = x0 + 1.75 * t
26 x_cd = x0 + 0.66 * t
27 x_rar = x0 - np.sqrt(5.0/3.0) * t
28
29 for ax in axes:
30     ax.axvline(x_shock, ls='--', c='r', lw=1.2, label='Shock')
31     ax.axvline(x_cd, ls='--', c='g', lw=1.2, label='Contact disc.')
32     ax.axvline(x_rar, ls=':', c='b', lw=1.2, label='Rarefaction
33         head')
34 axes[0].legend(fontsize=9, loc='upper right')
35 axes[-1].set_xlabel('x')
36 plt.suptitle(f'Riemann problem, t = {t:.1f}')
37 plt.tight_layout()
38 plt.savefig('riemann_solution.png', dpi=150)

```

Listing 9.4: Plotting the Riemann solution with analytical overplot.

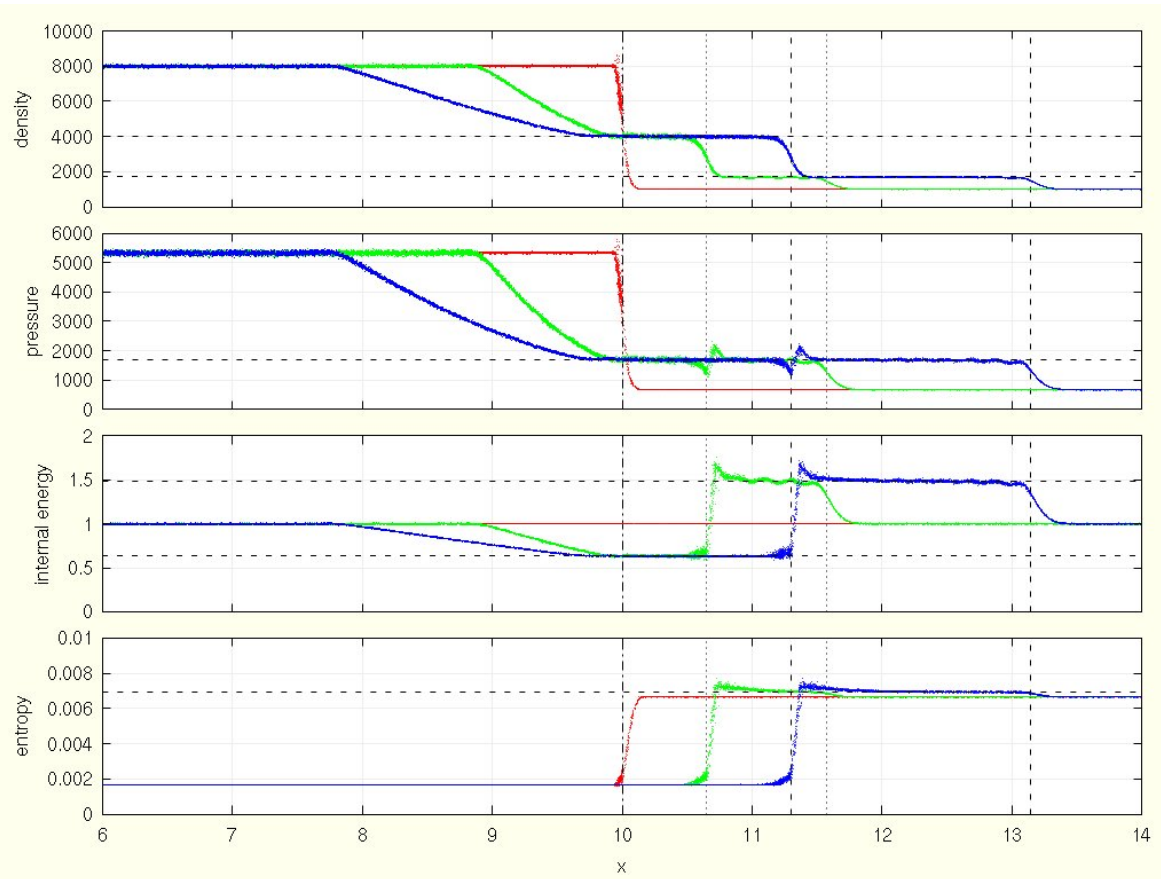


Figure 9.3: Time evolution of the Riemann solution: density, pressure, internal energy, and entropy as functions of x at several output times (blue, green, and red curves). The shock propagates to the right while the rarefaction fan expands to the left. Dashed lines indicate the five-region analytical structure. This multi-time plot is produced by the tutorial's IDL visualisation script `show_single.pro`.

9.6.1 IDL and Fortran tools

The tutorial also provides IDL and Fortran scripts for those who prefer them:

```
% IDL solution:
idl> .compile show_single.pro
idl> show_single

% Fortran reader + gnuplot:
ifx -o readsnap readsnap.f90
for file in snap_???; do ./readsnap $file > $file.txt; done
gnuplot pressure.plt % reproduces Figure~\ref{fig:t4_pressure}
```

Exercise T4

1. Reproduce Figure 9.1: write a short script to plot $f_L(P)$ and $f_R(P)$ as functions of P for the tutorial initial conditions. Find P^* by Newton–Raphson or bisection. Verify $P^* \approx 0.303$.
2. Run the three progressive steps (C2, C4, C4+conductivity) and make side-by-side plots at $t = 5$. Quantify the improvement: compute the RMS deviation of the simulated pressure from P^* in the post-shock flat region.

3. Overplot the exact five-region analytical solution (with correct ρ , v , P in each region) on your simulation plot. Which region is best captured by SPH? Which is worst?
4. Change the initial right-state pressure to $P_R = 0.5$ (a weaker shock). Does the pressure blip at the contact discontinuity become larger or smaller? Explain physically.
5. (Advanced) Repeat in 3D with `LONG_X=100` and a glass slab. Does the 3D result improve over 2D for the same particle count?

Chapter 10

Tutorial T5: The Evrard Collapse

T5 is the first simulation that couples gravity and hydrodynamics: the Evrard collapse (Evrard, 1988). A cold gas cloud with a $\rho \propto 1/r$ density profile collapses under its own gravity, forms a central shock, and virialises. It is the standard validation test for SPH codes and the first experiment in which OpenGADGET3 writes a physically meaningful `energy.txt`.

10.1 Physical Background

10.1.1 The Evrard setup

The initial conditions are a gas sphere with density profile $\rho(r) = M_*/(2\pi R_*^2) r^{-1}$ for $r < R_*$ (uniform in r^2), total mass $M_* = 1.5815$, radius $R_* = 1.0$, $G = 1.0$, and initial specific internal energy $u_0 = 0.05 u_*$. The characteristic scales:

$$t_* = \sqrt{R_*^3/(GM_*)} = 1.082, \quad (10.1)$$

$$u_* = GM_*/R_* = 1.5815, \quad (10.2)$$

$$\rho_* = 3M_*/(4\pi R_*^3) = 0.379. \quad (10.3)$$

The initial thermal energy $u_0 = 0.05 \ll u_*$ means the cloud is essentially cold: pressure is negligible and the collapse is nearly pressureless until the central density becomes high enough to generate a strong outward shock.

10.1.2 What to expect: three phases

1. **Collapse phase** ($0 < t \lesssim t_*$): the cloud free-falls inward. Kinetic energy grows at the expense of potential energy; internal energy rises slowly as shells cross and compress the central gas.
2. **Bounce** ($t \approx t_*$): a strong central shock forms and propagates outward. E_{int} spikes while E_{kin} drops sharply; E_{pot} rebounds from its minimum.
3. **Virialisation** ($t \gg t_*$): the system oscillates and settles to the virial equilibrium: $2E_{\text{kin}} + E_{\text{pot}} + 3(\gamma - 1)E_{\text{int}} = 0$.

Figure 10.1 shows the full energy evolution from the tutorial, clearly displaying all three phases.

10.2 Config.sh

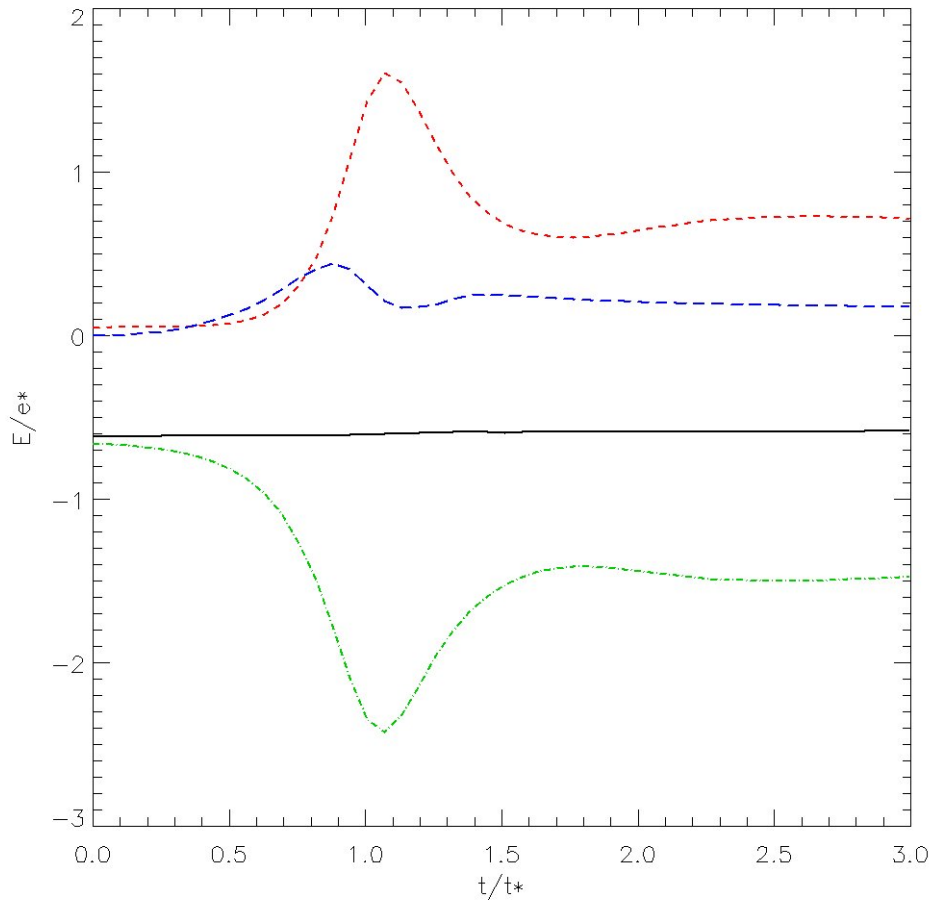


Figure 10.1: Energy evolution during the Evrard collapse (Tutorial T5). Time is in units of $t_* = 1.082$. Red dashed: internal energy E_{int} . Blue dashed: kinetic energy E_{kin} . Green dash-dot: potential energy E_{pot} . Black solid: total energy E_{tot} (conserved). The bounce at $t \approx t_*$ is clearly visible as the simultaneous spike in E_{int} and E_{kin} and the minimum of E_{pot} . At $t \gtrsim 2t_*$ the system approaches virial equilibrium.

```
#PERIODIC          % isolated boundary conditions: cloud not confined by
a box
#NOGRAVITY         % gravity ON: this is the whole point of T5
WENDLAND_C4_KERNEL
EVALPOTENTIAL      % write gravitational potential to snapshot (needed
for virial check)
```

Listing 10.1: Config.sh for Tutorial T5.

10.3 Parameter File

```
PeriodicBoundariesOn 0
TimeMax              3.0          % run for ~3 t_* = 3 code time
units
TimeBetSnapshot      0.05         % 60 snapshots over the run
TimeBetStatistics    0.01         % write energy.txt every 0.01 t (
smooth curve)
```

```

GravityConstantInternal  1.0      % G = 1.0 for this dimensionless
                             test
ArtBulkViscConst         0.8      % strong central shock: need
                             adequate viscosity
UnitLength_in_cm         1.0      % all units dimensionless
UnitMass_in_g            1.0
UnitVelocity_in_cm_per_s 1.0
SofteningGas             0.02     % ~1/50 of R_*; sets force
                             resolution
SofteningGasMaxPhys      0.02

```

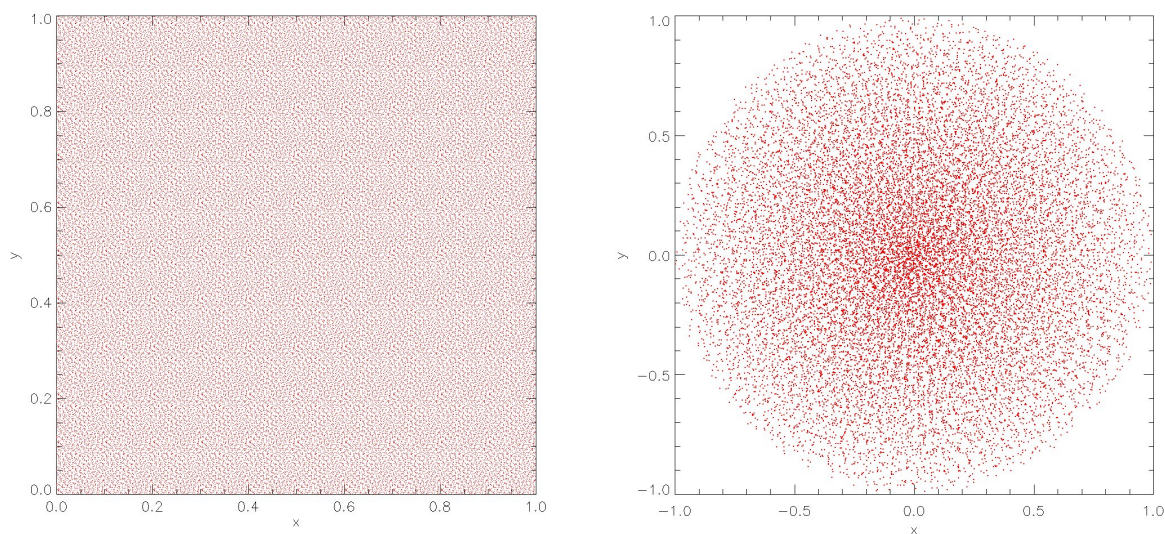
Listing 10.2: Parameter file for Tutorial T5.

10.4 Initial Conditions

10.4.1 Using the provided IC

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T05/box.ic
```

This file contains $\sim 10^5$ gas particles following a $\rho \propto 1/r$ profile, constructed from a glass distribution. Figure 10.2 shows the two stages of IC construction: the uniform glass cube and the transformed spherical distribution.



(a) Uniform glass distribution in the unit cube $[0, 1]^3$ (projected onto the x - y plane). This is the starting point: particles uniformly sample the cube with no preferred direction.

(b) After the $1/r$ density transformation and selection of a sphere. Particles are denser near the centre (where $\rho \propto 1/r$) and sparser toward the edge. The boundary is sharp at $r = R_* = 1$.

Figure 10.2: Construction of the Evrard collapse initial conditions. Left: the uniform glass template. Right: the transformed $\rho \propto 1/r$ spherical IC, shown projected onto the x - y plane.

10.4.2 Building the IC from scratch

```

# Step 1: get (or enlarge) a glass file
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/glass_10x10x10

```

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T05/enlarge_boxes.py
python3 enlarge_boxes.py      % produces a glass_100x100x100 (10^6
                             particles)
```

```
1 import g3read as g3, numpy as np
2
3 f = g3.GadgetFile('glass_100x100x100')
4 pos = f.read_new('POS ', 0) - 0.5 % centre at origin
5
6 r = np.sqrt((pos**2).sum(axis=1))
7 mask = r < 0.5
8 pos, r = pos[mask], r[mask]
9
10 % Transform: for rho ~ 1/r we need M(<r) ~ r^2
11 % If glass has M(<r_glass) = (r_glass/r_max)^3 (uniform),
12 % we want r_new such that (r_new/r_max)^2 = (r_glass/r_max)^3
13 % => r_new = r_max * (r_glass/r_max)^(3/2)
14 r_max = r.max()
15 r_new = r_max * (r / r_max)**(3.0/2.0)
16
17 scale = np.where(r > 0, r_new / r, 0.0)
18 pos = pos * scale[:, None]
19
20 % Evrard normalisation: M* = 1.5815, R* = 1.0
21 R_star, M_star = 1.0, 1.5815
22 pos *= R_star / r_max
23 N = len(pos)
24 mass = np.ones(N, dtype=np.float32) * M_star / N
25 u = np.ones(N, dtype=np.float32) * 0.05
26 vel = np.zeros((N, 3), dtype=np.float32)
27 ids = np.arange(1, N+1, dtype=np.int32)
28 % Write IC using write_block() from Chapter 4
```

Listing 10.3: Transforming glass positions to a $1/r$ density profile.

10.5 Running

```
mkdir output && sbatch runme.sh % recommended: ntasks=4, cpus-per-
task=2
```

The simulation takes 10–30 minutes depending on particle count and resources. Monitor the progress with `tail -f slurm-*.out`: you will see the timestep shrinking sharply near $t \approx 1$ as the central shock forms.

10.6 Analysis

10.6.1 Energy evolution and virial theorem

```
1 import numpy as np, matplotlib.pyplot as plt
2
3 data = np.loadtxt('energy.txt')
4 t = data[:, 0]
5 Eint = data[:, 1]
6 Epot = data[:, 2]
7 Ekin = data[:, 3]
```

```

8 Etot = Eint + Epot + Ekin
9
10 gamma = 5.0/3.0
11 virial = 2*Ekin + Epot + 3*(gamma-1)*Eint
12
13 fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 8), sharex=True)
14 ax1.plot(t, Ekin, 'b--', label=r'$E_{\mathrm{kin}}$')
15 ax1.plot(t, Epot, 'g-', label=r'$E_{\mathrm{pot}}$')
16 ax1.plot(t, Eint, 'r--', label=r'$E_{\mathrm{int}}$')
17 ax1.plot(t, Etot, 'k-', label=r'$E_{\mathrm{tot}}$', lw=2)
18 ax1.axhline(0, c='gray', ls=':')
19 ax1.set_ylabel(r'Energy [$G M_* / R_*]$')
20 ax1.legend(fontsize=9)
21
22 ax2.plot(t, virial/abs(Epot), c='purple')
23 ax2.axhline(0, c='gray', ls=':')
24 ax2.set_xlabel(r'Time [$t_*$]')
25 ax2.set_ylabel(r'Virial ratio $(2E_k+E_p+3(\gamma-1)E_i)/|E_p|$')
26 plt.tight_layout()
27 plt.savefig('energies.png', dpi=150)

```

Listing 10.4: Plotting energy evolution from energy.txt.

10.6.2 Comparison with the PPM Eulerian solution

The reference solution from the Piecewise Parabolic Method (PPM), an Eulerian code with excellent shock capturing, is provided with the tutorial:

```

wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/ppm.tar
tar xvf ppm.tar
% Each file snap_NNN: columns = radius, density, pressure

```

Figure 10.3 shows density, pressure, and radial velocity profiles at snapshot 006 (near maximum compression, $t \approx 0.8 t_*$) comparing the SPH result (red dots) with the PPM reference (blue line).

Figure 10.4 shows a similar comparison at early time $t/t_* = 0.13$, when the cloud is still in free fall and only the inner shells have begun to form the nascent shock.

10.6.3 Density histogram: verification of the $1/r$ profile

Figure 10.5 shows the radial density profile at a late snapshot in linear scale, compared with the analytical $1/r$ law. This confirms that the IC was set up correctly and that the particles sample the desired density profile.

```

1 import g3read as g3, numpy as np, matplotlib.pyplot as plt
2
3 % Load SPH snapshot
4 f = g3.GadgetFile('output/snap_006')
5 pos = f.read_new('POS ', 0)
6 rho = f.read_new('RHO ', 0)
7 u = f.read_new('U ', 0)
8 vel = f.read_new('VEL ', 0)
9
10 % Radial quantities (centre assumed at origin after centring the cloud)
11 r = np.sqrt((pos**2).sum(axis=1))
12 v_r = (pos * vel).sum(axis=1) / (r + 1e-10)

```

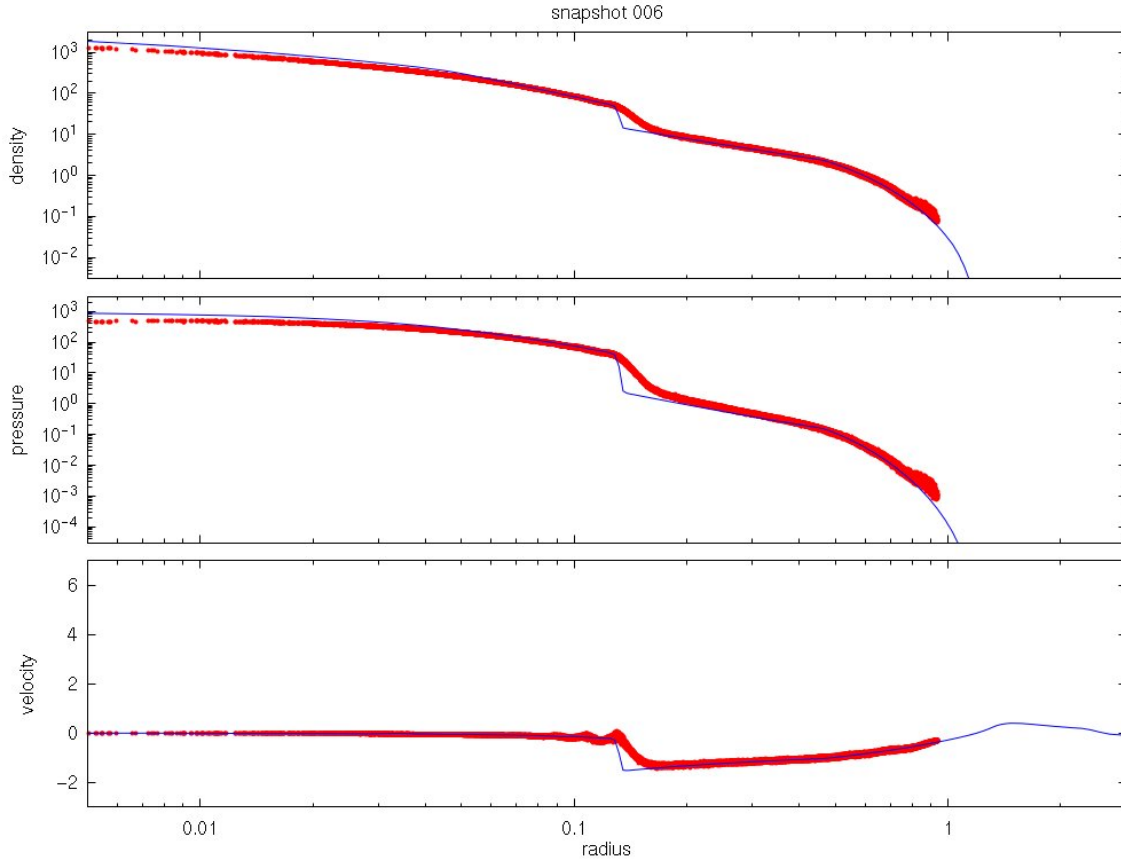


Figure 10.3: Radial profiles at snapshot 006 ($t \approx 0.8 t_*$) for the Evrard collapse. Three panels show density (top), pressure (middle), and radial velocity (bottom) as functions of radius, on logarithmic radial scale. Red filled circles: OpenGADGET3 SPH particles (binned into radial shells). Blue line: PPM Eulerian reference solution. The shock front is visible near $r \approx 0.15$ as the sharp density and pressure jump; the velocity reversal (infalling gas becomes outflowing post-shock gas) is clear in the bottom panel.

```

13 P = (5.0/3.0 - 1) * rho * u
14
15 % Bin into radial shells (logarithmic)
16 r_bins = np.logspace(np.log10(r[r>0].min()), np.log10(r.max()), 60)
17 r_mid = np.sqrt(r_bins[:-1] * r_bins[1:])
18 rho_bin = np.array([rho[(r >= r_bins[i]) & (r < r_bins[i+1])].mean()
19                     for i in range(len(r_bins)-1)])
20 P_bin = np.array([P[(r >= r_bins[i]) & (r < r_bins[i+1])].mean()
21                  for i in range(len(r_bins)-1)])
22 vr_bin = np.array([v_r[(r >= r_bins[i]) & (r < r_bins[i+1])].mean()
23                   for i in range(len(r_bins)-1)])
24
25 % Load PPM reference
26 ppm = np.loadtxt('ppm/snap_006') % radius, density, pressure
27 r_ppm, rho_ppm, P_ppm = ppm[:,0], ppm[:,1], ppm[:,2]
28
29 fig, axes = plt.subplots(3, 1, figsize=(8, 10), sharex=True)
30 axes[0].loglog(r_mid, rho_bin, 'r.', ms=3, label='SPH')
31 axes[0].loglog(r_ppm, rho_ppm, 'b-', lw=1.5, label='PPM')
32 axes[0].set_ylabel('Density')
33 axes[0].legend()

```

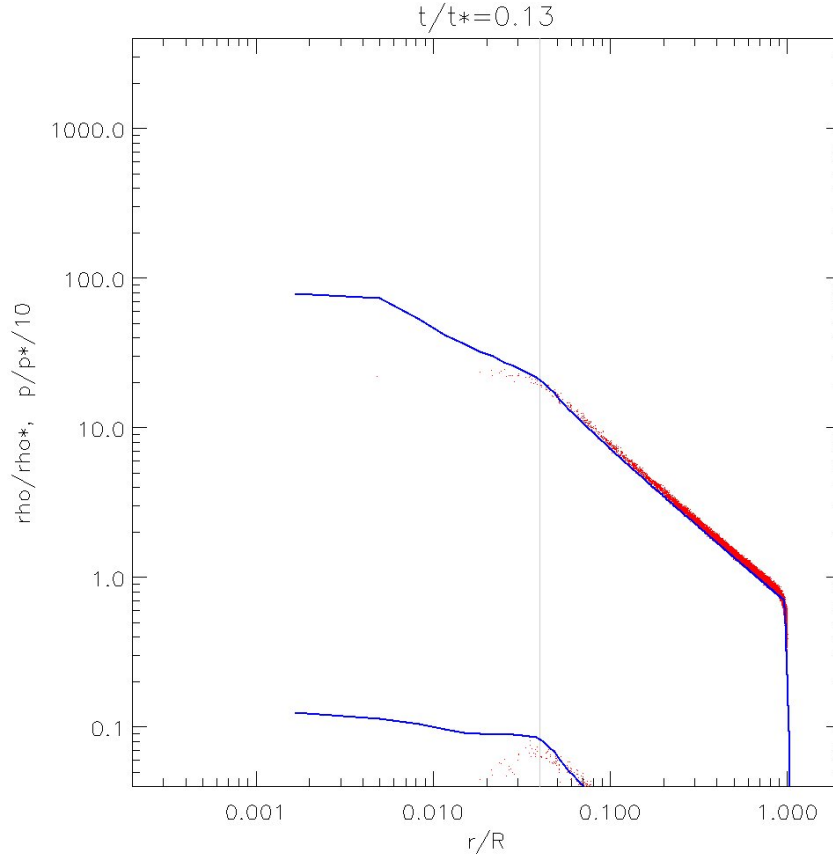


Figure 10.4: Density and pressure profiles at $t/t_* = 0.13$ (early collapse phase). Axes are normalised to the Evrard characteristic scales ρ_* and P_* . Red dots: SPH particles. Blue line: PPM reference. The vertical grey line marks the gravitational softening radius $\epsilon = 0.02 R_*$. SPH and PPM agree well in the outer regions where the density profile is still close to the initial $1/r$. Deviations appear at $r \lesssim \epsilon$ where the softened gravity affects the collapse rate.

```

34 axes[1].loglog(r_mid, P_bin, 'r.', ms=3)
35 axes[1].loglog(r_ppm, P_ppm, 'b-', lw=1.5)
36 axes[1].set_ylabel('Pressure')
37
38 axes[2].semilogx(r_mid, vr_bin, 'r.', ms=3)
39 axes[2].axhline(0, c='gray', ls=':')
40 axes[2].set_ylabel('Radial velocity'); axes[2].set_xlabel('Radius')
41
42 plt.tight_layout()
43 plt.savefig('evrard_profiles.png', dpi=150)
44

```

Listing 10.5: Plotting radial profiles and comparison with PPM.

Exercise T5

1. From Figure 10.1 (or your own simulation), identify the time of maximum compression (E_{pot} minimum) and the time at which the virial residual drops below 5% of $|E_{\text{pot}}|$. Convert both to physical time using $t_* = 1.082$ code time units.
2. Verify the virial theorem quantitatively at $t = 3t_*$. Read E_{kin} , E_{pot} , E_{int} from

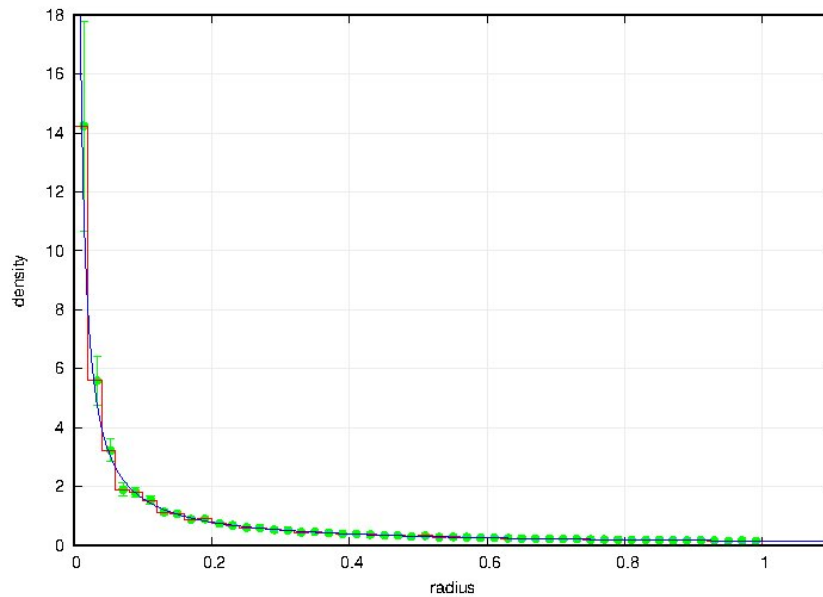


Figure 10.5: Radial density profile binned in linear radius at a late snapshot. Green circles: SPH data with Poisson error bars. Red/blue lines: two differently-binned representations of the same data. Black curve: analytical $\rho \propto 1/r$ reference. The agreement confirms the IC construction and the accuracy of the SPH density estimate over three decades in radius.

`energy.txt` and compute the virial ratio $(2E_k + E_p + 3(\gamma - 1)E_i)/|E_p|$. What percentage deviation from zero do you find?

3. Reproduce Figure 10.3: plot density, pressure, and velocity profiles at snapshot 006 in log-radius scale and overplot the PPM reference. Identify the shock position. Does the shock radius in SPH agree with PPM?
4. Reduce `ArtBulkViscConst` from 0.8 to 0.1 and rerun. Does the shock become sharper or does the simulation behave differently? Does the total energy remain conserved?
5. (Advanced) Change the softening from 0.02 to 0.001. What happens to the central density at the bounce? What happens to the timestep? Explain the trade-off between force resolution and computational cost.

Chapter 11

Tutorial T6: Sound Wave Propagation

T6 tests whether OpenGADGET3 propagates linear sound waves at the correct speed and with acceptable numerical damping. It introduces non-cubic slab geometries, demonstrates the sensitivity of wave propagation to kernel choice and artificial viscosity, and provides the first opportunity to explore MPI+OpenMP parallelism.

11.1 Physical Background

For a small perturbation $\delta\rho/\rho_0 \ll 1$ in a uniform adiabatic medium, the linearised Euler equations give a dispersion relation $\omega = c_s k$, where the adiabatic sound speed is:

$$c_s = \sqrt{\frac{\gamma P_0}{\rho_0}} = \sqrt{(\gamma - 1)\gamma u_0}. \quad (11.1)$$

The wave propagates without dispersion or damping for an ideal inviscid fluid. In OpenGADGET3, two numerical effects degrade the solution:

Amplitude damping from artificial viscosity, which acts on converging particle pairs even in the compression half-cycle of a smooth wave. Controlled by `ArtBulkViscConst`: reduce it to ~ 0.05 for sound wave tests.

Phase error from finite resolution: the effective SPH sound speed deviates from c_s at wavelengths $\lambda \lesssim 2h$. Use a high-order kernel (`WENDLAND_C6`) to minimise this.

In linear theory, the perturbations in density $\delta\rho$, internal energy δu , and velocity δv are all sinusoidal and in phase (for a compression wave) or 90° out of phase (for density vs velocity):

$$\frac{\delta\rho}{\rho_0} = \frac{\delta u}{u_0} = \frac{v_x}{c_s} \sin(kx - \omega t), \quad \omega = c_s k. \quad (11.2)$$

Figure 11.1 shows this directly: the fractional perturbations $\epsilon/\epsilon_0 - 1$, $\rho/\rho_0 - 1$, and v all oscillate sinusoidally across the box, with the same amplitude and wavelength.

11.2 Config.sh

```
PERIODIC
NOGRAVITY
WENDLAND_C6_KERNEL    % highest-order kernel: tiny amplitude requires
```

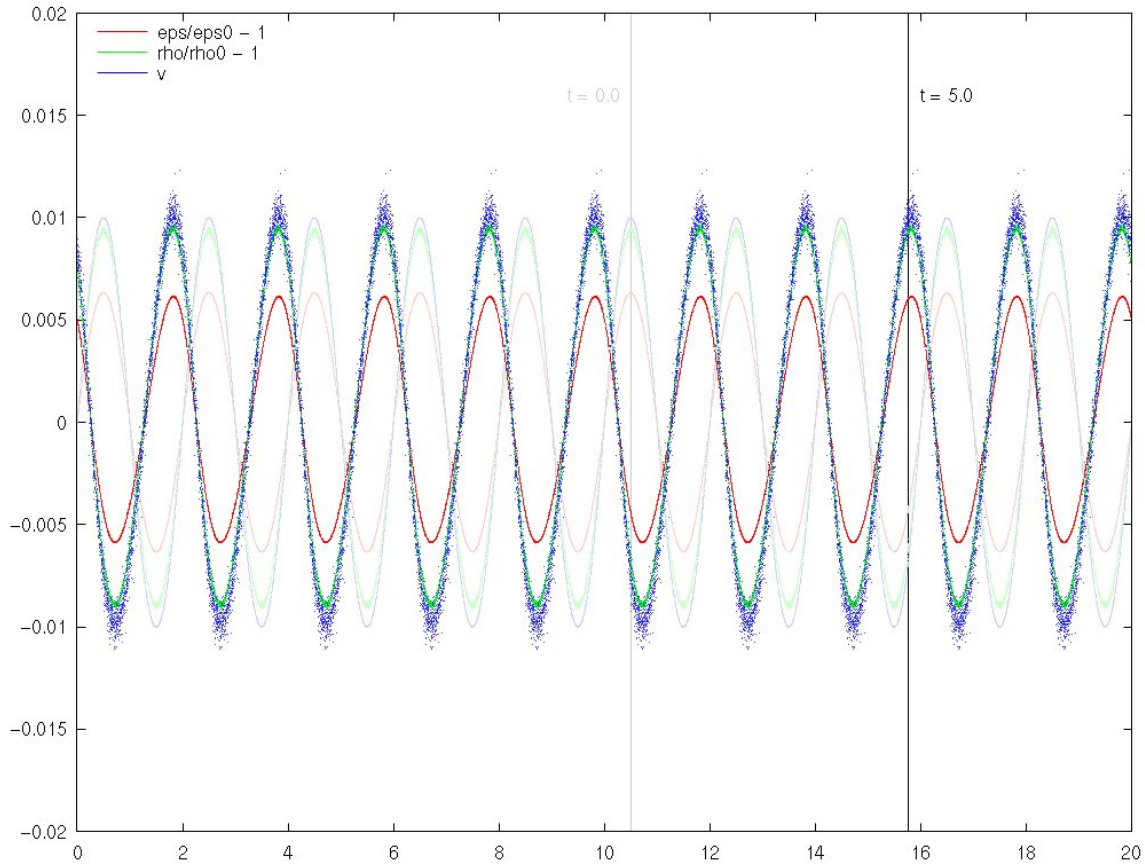


Figure 11.1: Sound wave perturbations as a function of position x at two output times ($t = 0$ and $t = 5$, marked by vertical lines). Red: relative perturbation in specific internal energy $\varepsilon/\varepsilon_0 - 1$. Green: relative density perturbation $\rho/\rho_0 - 1$. Blue: velocity v . The wave propagates to the right without significant damping. The amplitude ratio between energy and density perturbations is $(\gamma - 1)$, as predicted by linear theory.

```

LONG_X=20                                % best density recovery possible
LONG_Y=1                                % slab: 20x longer in x than in y or z
LONG_Z=1

```

Listing 11.1: Config.sh for Tutorial T6.

The choice of WENDLAND_C6_KERNEL is deliberate. With a wave amplitude $\delta\rho/\rho_0 \sim 10^{-3}$, the density must be recovered to one part in a thousand. Only the C6 kernel with $N_{\text{ngb}} \gtrsim 300$ achieves this accuracy; lower-order kernels drown the wave signal in density noise.

11.3 Parameter File

```

PeriodicBoundariesOn    1
TimeMax                  20.0
TimeBetSnapshot          0.1

UnitLength_in_cm         1.0
UnitMass_in_g             1.0
UnitVelocity_in_cm_per_s 1.0

SofteningGas              0.05

```

SofteningGasMaxPhys	0.05	
ArtBulkViscConst	0.05	% critical: default 0.8 damps the wave strongly

Listing 11.2: Parameter file for Tutorial T6.

11.4 Initial Conditions

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T06/box.ic
```

The IC is a $200 \times 10 \times 10$ glass slab (20000 particles) with a longitudinal velocity perturbation $v_x = A \sin(2\pi x/L_x)$ with amplitude $A = 10^{-3}$. The base state has $\rho_0 = 1$, $P_0 = 2/3$ so that $c_s \approx 1.054$.

Figure 11.2 shows the initial v_x profile: a smooth sinusoidal wave with one full period across the box length ($\lambda = L_x = 20$). The flat region in the centre is the zero crossing of the sine; note that the amplitude (± 0.1) is $\sim 10\times$ larger in this IDL example than in the standard IC (which uses $A = 10^{-3}$) to make the structure clearly visible.

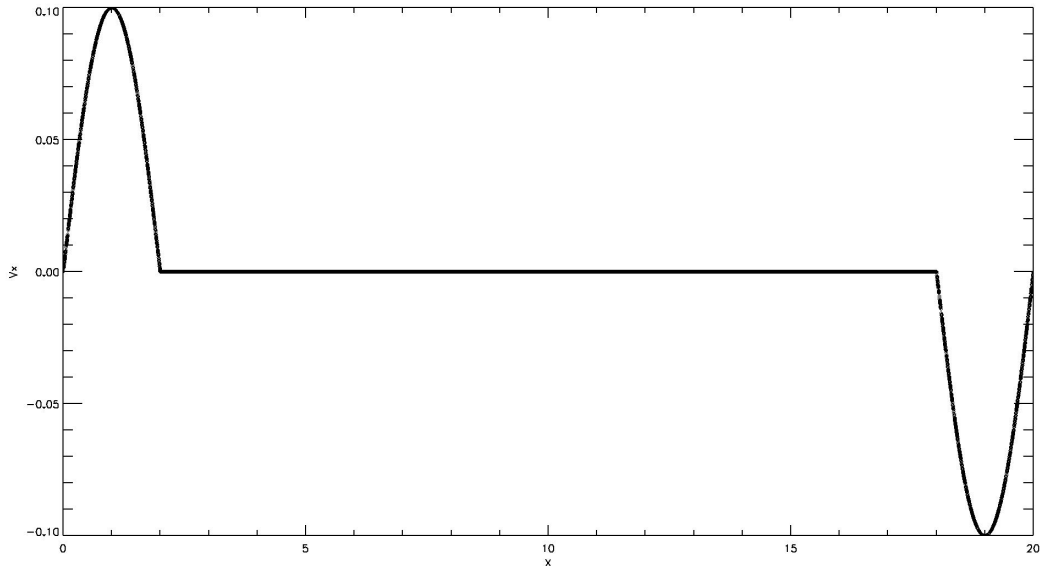


Figure 11.2: Initial condition for Tutorial T6: the x -velocity v_x as a function of position x at $t = 0$. A pure sinusoidal perturbation is applied along the slab ($v_y = 0$ initially). The amplitude here is exaggerated for visibility; the actual tutorial uses $A = 10^{-3} \ll c_s$ to remain in the linear regime. The flat region near $v_x = 0$ in the interior is the zero-crossing of the sine.

Building the IC yourself:

```
1 import g3read as g3, numpy as np
2
3 base = g3.GadgetFile('glass_10x10x10')
4 pos0 = base.read_new('POS ', 0) % (1000, 3) in [0,1]^3
5
6 copies = []
7 for ix in range(20):
8     p = pos0.copy(); p[:, 0] += ix
9     copies.append(p)
10 pos_slab = np.vstack(copies)
```

```

11 pos_slab[:, 0] /= 20.0    % normalise: x in [0,1], LONG_X=20 gives
    physical L=20
12
13 N    = len(pos_slab)
14 P0   = 2.0/3.0           % => c_s = sqrt(5/3 * P/rho) ~ 1.054
15 u0   = P0 / ((5.0/3.0 - 1.0) * 1.0)
16 A    = 1e-3              % amplitude << c_s: linear regime
17 x    = pos_slab[:, 0]
18 vx   = A * np.sin(2 * np.pi * x)    % one wavelength in [0,1]
19
20 vel  = np.zeros((N, 3), dtype=np.float32)
21 vel[:, 0] = vx.astype(np.float32)
22 u    = np.ones(N, dtype=np.float32) * u0
23 mass = np.ones(N, dtype=np.float32) / N
24 ids  = np.arange(1, N+1, dtype=np.int32)

```

Listing 11.3: IC setup for sound wave.

11.5 Simulation Results

11.5.1 Wave profiles

Figures 11.3 and 11.4 show the state of the slab at two times for longitudinal and transverse perturbations respectively.

11.5.2 Comparison at different times

Figure 11.5 shows the velocity perturbation δv_x at three successive times. The wave travels to the right at $c_s \approx 1.054$, and the amplitude decreases slightly due to the residual artificial viscosity. The wave maintains its sinusoidal shape over many crossing times, confirming that the SPH dispersion relation is correctly reproduced.

11.6 MPI+OpenMP Parallelism

T6 is an ideal benchmark for exploring parallelism: the slab geometry is efficiently decomposed by MPI (each rank handles one or more x -slices), and the uniform density field minimises load-balance problems.

```

% Try these configurations in runme.sh:
#SBATCH --ntasks=2 --cpus-per-task=2    % 4 cores total, 2 MPI ranks
#SBATCH --ntasks=2 --cpus-per-task=4    % 8 cores total, 2 MPI ranks
#SBATCH --ntasks=4 --cpus-per-task=2    % 8 cores total, 4 MPI ranks

```

The code reports the configuration at startup. For 20000 particles, MPI communication overhead may exceed the parallel speedup; this becomes less of an issue for larger production simulations.

11.7 Analysis

```

1 import g3read as g3, numpy as np, matplotlib.pyplot as plt, glob
2
3 snaps = sorted(glob.glob('output/snap_???'))
4 L_x   = 20.0    % physical x-extent (LONG_X=20, BoxSize=1)
5 Nbin  = 200

```

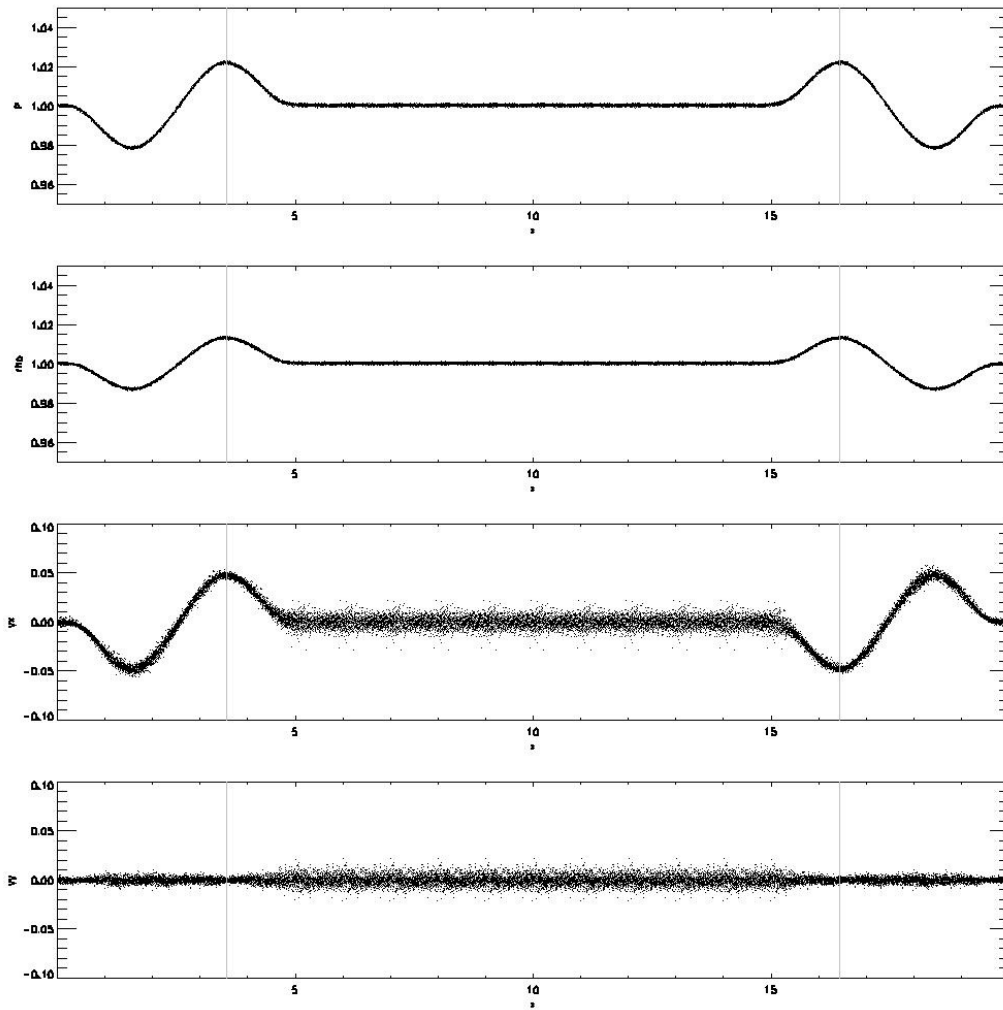


Figure 11.3: Sound wave with longitudinal perturbation at an intermediate time. Four panels (top to bottom): pressure P , density ρ , longitudinal velocity v_x , transverse velocity v_y . The sinusoidal wave is clearly visible in all fields. The density and pressure oscillations are in phase (compression wave). The transverse velocity v_y (bottom panel) remains near zero throughout, confirming that the longitudinal perturbation does not excite a transverse mode in a uniform medium.

```

6 times, amplitudes = [], []
7 prev_peak_x, prev_time, speeds = None, None, []
8
9 for s in snaps:
10     f = g3.GadgetFile(s)
11     pos = f.read_new('POS ', 0)
12     vel = f.read_new('VEL ', 0)
13     x = pos[:, 0] * L_x
14     vx = vel[:, 0]
15
16     xbins = np.linspace(0, L_x, Nbin)
17     counts = np.histogram(x, bins=xbins)[0]
18     vx_bin = np.histogram(x, bins=xbins, weights=vx)[0] / np.maximum(
19         counts, 1)
20     x_mid = 0.5*(xbins[:-1]+xbins[1:])
21     pk = np.argmax(vx_bin)

```

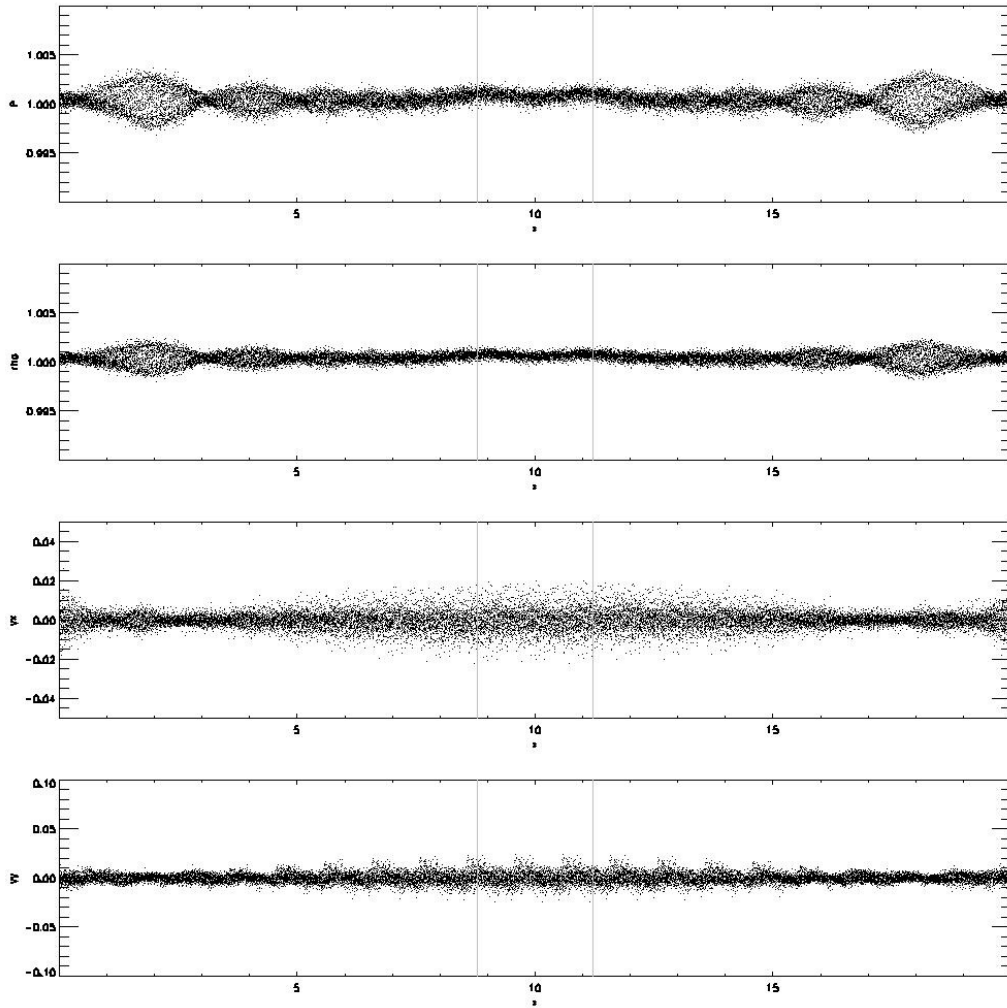


Figure 11.4: State of the slab with a purely transverse (v_y) initial perturbation. All four fields (P , ρ , v_x , v_y) are shown. Unlike the longitudinal case, a transverse perturbation in a non-magnetised ideal fluid does not produce any density or pressure variation: it is not a sound wave. This confirms that sound waves are strictly longitudinal in an isotropic ideal gas.

```

22     times.append(f.header.time)
23     amplitudes.append(vx_bin[pk])
24
25     if prev_peak_x is not None:
26         dx = x_mid[pk] - prev_peak_x
27         dt = f.header.time - prev_time
28         if 0 < dx < L_x/2:
29             speeds.append(dx/dt)
30     prev_peak_x = x_mid[pk]
31     prev_time = f.header.time
32
33 cs_theory = np.sqrt(5.0/3.0 * (2.0/3.0) / 1.0)
34 print(f'Theoretical c_s = {cs_theory:.4f}')
35 if speeds:
36     print(f'Measured c_s = {np.mean(speeds):.4f} ({abs(np.mean(
37         speeds)/cs_theory-1)*100:.2f}% error)')
38
39 fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 6), sharex=True)
40 ax1.plot(times, amplitudes, 'o-', ms=2)

```

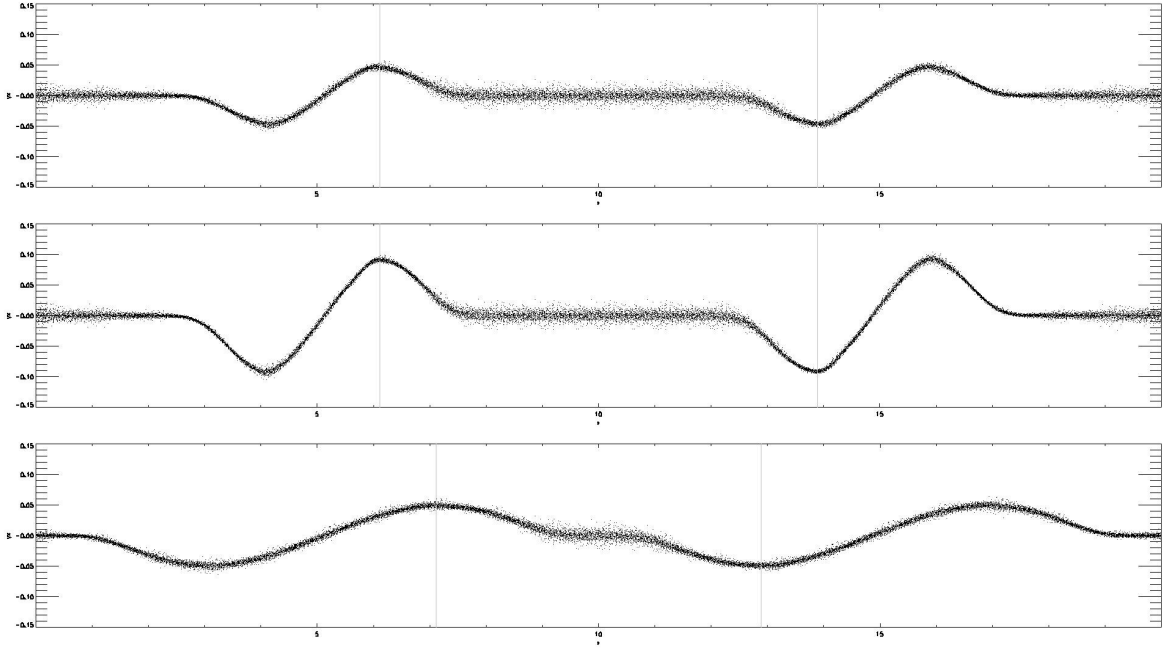


Figure 11.5: Velocity perturbation δv_x vs x at three successive output times (top to bottom panels). The wave moves to the right by approximately $c_s \Delta t$ between panels. The peak amplitude decreases slightly due to artificial viscosity damping, which is quantified in the analysis section. The wave shape remains sinusoidal, confirming linear propagation.

```

40 ax1.set_ylabel(r'Peak  $\delta v_x$  amplitude'); ax1.set_title('Sound wave
    propagation')
41 if speeds:
42     ax2.plot(times[1:], speeds, 'o-', ms=2, label='Measured')
43     ax2.axhline(cs_theory, ls='--', c='r', label=f'$c_s$ = {cs_theory
        :.4f}')
44     ax2.legend()
45 ax2.set_xlabel('Time [code units]'); ax2.set_ylabel('Propagation speed
    ')
46 plt.tight_layout(); plt.savefig('soundwave.png', dpi=150)
    
```

Listing 11.4: Measuring wave speed and amplitude damping.

Exercise T6

1. Measure the sound speed from your simulation and compare with the theoretical value $c_s = \sqrt{\gamma P_0 / \rho_0}$. What percentage error do you find?
2. Increase `ArtBulkViscConst` from 0.05 to 0.5. Plot the amplitude as a function of time for both values. Fit an exponential decay $A(t) = A_0 e^{-t/\tau}$ to each. How does the damping timescale τ scale with `ArtBulkViscConst`?
3. Switch from `WENDLAND_C6_KERNEL` to `STANDARD_KERNEL` (cubic spline). Does the wave shape remain sinusoidal? Does the sound speed change? Explain in terms of density estimation accuracy.
4. Apply a transverse perturbation $v_y = A \sin(2\pi x / L_x)$ (instead of v_x) as shown in Figure 11.4. Does it propagate as a wave? Does it damp? What does this tell you about transverse waves in a non-magnetised gas?
5. (Advanced) Introduce two wavelengths simultaneously: $v_x = A[\sin(2\pi x / L_x) + \sin(4\pi x / L_x)]$. Do the two modes propagate independently? What would happen

in a dispersive medium?

Chapter 12

Tutorial T7a: Supernova Blast Wave

T7a simulates a point explosion in a uniform medium — the Sedov–Taylor blast wave, the prototypical model for supernova remnants and superbubbles. It tests the ability of OpenGADGET3 to capture a strong, spherically symmetric shock expanding into a cold ambient medium.

12.1 Physical Background

A total energy E_0 is instantaneously deposited in a small central volume of a uniform medium at rest with density ρ_0 and negligible pressure. A spherical blast wave expands outward as a strong shock. The Sedov–Taylor self-similar solution, valid once the swept-up mass greatly exceeds the ejecta mass, gives a shock radius that grows as:

$$R_s(t) = \left(\frac{E_0}{\alpha \rho_0} \right)^{1/5} t^{2/5}, \quad (12.1)$$

where $\alpha \approx 0.49$ for $\gamma = 5/3$. The shock front compresses the ambient gas by a factor $(\gamma + 1)/(\gamma - 1) = 4$, concentrating most of the swept-up mass in a thin shell. The post-shock temperature from the Rankine–Hugoniot conditions is:

$$T_2 = \frac{3\mu m_p}{16k_B} \dot{R}_s^2, \quad (12.2)$$

which decreases as $t^{-6/5}$ as the shock decelerates. This is the standard model for young supernova remnants such as Cassiopeia A and Tycho.

12.2 Config.sh and Parameter File

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
% No LONG_X/Y/Z: cubic box, explosion is spherical
```

Listing 12.1: Config.sh for Tutorial T7a.

```
PeriodicBoundariesOn      1
TimeMax                   15.0      % run long enough to see Sedov
      scaling
TimeBetSnapshot           0.1
UnitLength_in_cm          1.0
```

```

UnitMass_in_g      1.0
UnitVelocity_in_cm_per_s  1.0
SofteningGas       0.005
SofteningGasMaxPhys 0.005
ArtBulkViscConst   1.5      % strong shock: increase from
                        default 0.8

```

Listing 12.2: Parameter file for Tutorial T7a.

Use `ntasks=4`, `cpus-per-task=2`. The run takes several minutes for 10^6 particles.

12.3 Initial Conditions

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T07a/box.ic
```

The IC is a $100^3 \approx 10^6$ particle glass slab at uniform density ρ_0 and low ambient pressure $P_{\text{amb}} \ll 1$. A small cluster of ~ 10 – 20 particles near the box centre carries a much higher pressure $P_{\text{blast}} \gg P_{\text{amb}}$, concentrating the explosion energy in a volume of radius $\sim 3h$.

Figure 12.1 shows the initial particle distribution at $t = 0$: the bulk of the gas (grey dots) is uniformly distributed in the unit cube, while the handful of energised central particles (red) are barely visible at the box centre.

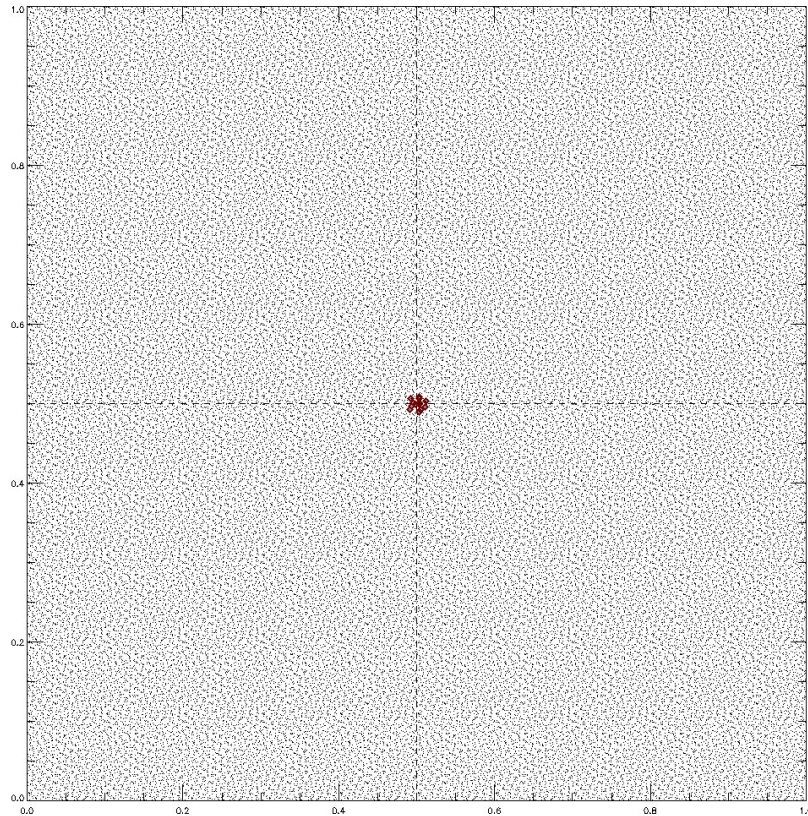


Figure 12.1: Initial conditions for Tutorial T7a projected onto the x - y plane. Grey: the $\sim 10^6$ uniform background particles at low pressure. Red (centre): the small cluster of ~ 10 – 20 particles with $P_{\text{blast}} \gg P_{\text{amb}}$, representing the instantaneous energy injection. The box is the unit cube $[0, 1]^3$ and the explosion is centred at $(0.5, 0.5, 0.5)$.

Tip

Compute the total injected energy before running: $E_0 = P_{\text{blast}} V_{\text{blast}} / (\gamma - 1)$, where V_{blast} is the volume of the energised region. This is what enters the Sedov scaling; an incorrect E_0 shifts the predicted $R_s(t)$ curve up or down.

12.4 Results

Figure 12.2 shows the simulation at $t = 10.1$. The left panels display radial profiles of density, velocity, internal energy, and pressure; the right panel shows a 2D colour map of the density field. The blast wave has reached $R_s \approx 0.55$ at this time, and the density peaks at the shock front at $\rho \approx 4\rho_0 = 0.5$ (strong shock compression factor).

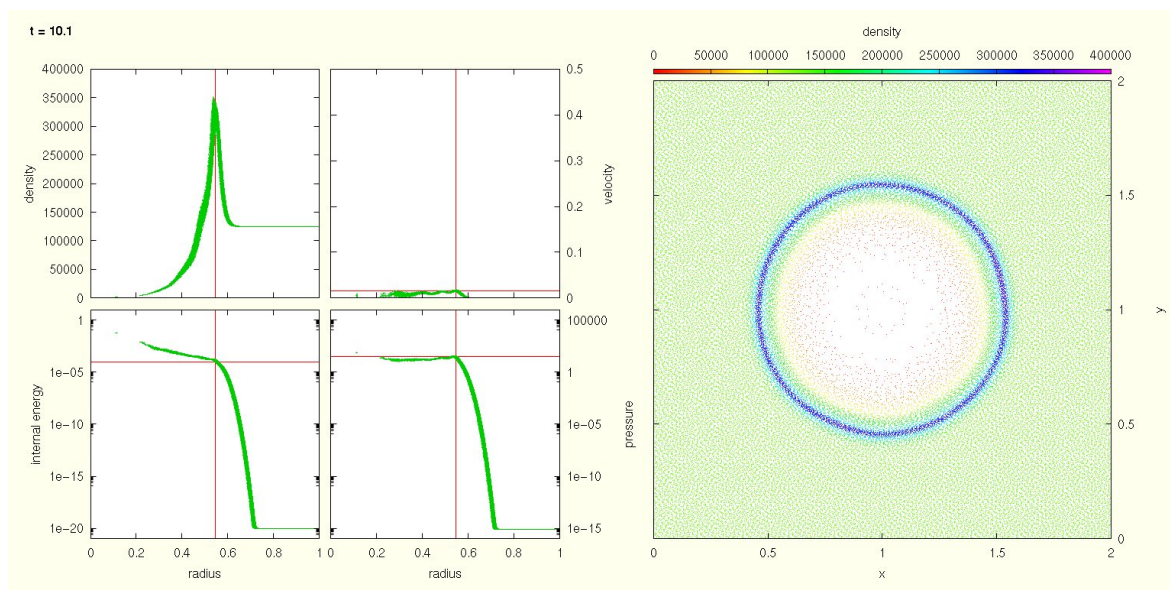


Figure 12.2: Tutorial T7a at $t = 10.1$. *Left panels*: radial profiles of density (top left), radial velocity (top right), specific internal energy (bottom left), and pressure (bottom right) as functions of radius. The red vertical line marks the shock position $R_s \approx 0.55$. The sharp density peak at the shock front reaches $\sim 4\rho_0$ (strong shock limit). *Right panel*: 2D projection of the SPH density field in the x - y plane, showing the expanding spherical shell. The colour scale (blue = dense shell, green = swept interior, white = hot low-density interior) reveals the characteristic structure of the Sedov blast wave.

12.5 Analysis: Sedov Scaling

```

1 import g3read as g3, numpy as np, glob, matplotlib.pyplot as plt
2
3 snaps = sorted(glob.glob('output/snap_???'))
4 times, R_shock = [], []
5
6 for s in snaps:
7     f = g3.GadgetFile(s)
8     pos = f.read_new('POS ', 0) - 0.5    % centre the box at origin
9     rho = f.read_new('RHO ', 0)
10    r = np.sqrt((pos**2).sum(axis=1))
11
12    % Radial density profile

```

```

13     Nbin    = 100
14     rbins   = np.linspace(0, 0.5, Nbin+1)
15     r_mid   = 0.5*(rbins[:-1]+rbins[1:])
16     rho_p, _ = np.histogram(r, bins=rbins, weights=rho)
17     cnt, _   = np.histogram(r, bins=rbins)
18     rho_p[cnt > 0] /= cnt[cnt > 0]
19
20     R_shock.append(r_mid[np.argmax(rho_p)])    % shock = density peak
21     times.append(f.header.time)
22
23 times    = np.array(times)
24 R_shock  = np.array(R_shock)
25
26 % Sedov prediction
27 E0, rho0, alpha = 1.0, 0.125, 0.49    % adjust E0 to your IC
28 xi       = (E0 / (alpha * rho0))**0.2
29 t_plot    = np.linspace(times[2], times[-1], 200)
30 R_sedov   = xi * t_plot**0.4
31
32 fig, ax = plt.subplots(figsize=(7, 5))
33 ax.plot(times, R_shock, 'o', ms=4, label='Simulation')
34 ax.plot(t_plot, R_sedov, '-', lw=2, label=r'Sedov: $R \propto t^{2/5}$')
35
36 ax.set_xlabel('Time'); ax.set_ylabel('Shock radius')
37 ax.set_title('Sedov--Taylor blast wave'); ax.legend()
38 plt.tight_layout(); plt.savefig('sedov_scaling.png', dpi=150)
39
40 print(f'Measured R(t=10): {np.interp(10.0, times, R_shock):.4f}')
41 print(f'Predicted R(t=10): {xi * 10.0**0.4:.4f}')

```

Listing 12.3: Measuring the shock radius and verifying Sedov scaling.

Exercise T7a

1. From Figure 12.2, estimate the shock radius at $t = 10.1$ directly from the radial density profile. Compare with the Sedov prediction $R_s = (E_0/\alpha\rho_0)^{1/5} t^{2/5}$.
2. Plot $\log R_s$ vs $\log t$ for all snapshots. Fit a power law and measure the exponent. How close is it to $2/5 = 0.4$?
3. Plot the temperature profile $T(r)$ at several times using equation (3.2). Does the post-shock temperature scale as $t^{-6/5}$?
4. Reduce `ArtBulkViscConst` from 1.5 to 0.5. Does the blast wave still expand correctly? What happens at the shock front in the density profile?
5. (Advanced) The interior of the blast wave (inside the shock) should be nearly isobaric in the Sedov solution. Verify this from the pressure profile in Figure 12.2: is $P(r)$ approximately constant for $r < R_s$?

Chapter 13

Tutorial T7b: Flow through a Nozzle

T7b introduces *boundary particles* — particles with $ID = 0$ that exert SPH forces on their neighbours but receive no hydrodynamic acceleration — to create fixed walls and a moving piston. This enables the simulation of driven flows through complex geometries, culminating in a supersonic nozzle with standing shocks.

13.1 Physical Background

In a converging–diverging (de Laval) nozzle, a subsonic flow entering from the left is accelerated as the cross-section narrows. If the throat reaches $\mathcal{M} = 1$, the flow becomes supersonic in the diverging section. A normal shock then forms downstream, decelerating the flow back to subsonic conditions. This is the operating principle of rocket nozzles and supersonic wind tunnels.

In the T7b experiment, a piston (moving boundary particle wall) compresses gas from the left, driving it through a symmetric converging–diverging nozzle. The resulting flow exhibits: subsonic acceleration, a $\mathcal{M} = 1$ throat, supersonic expansion, and one or more standing or propagating shocks downstream.

13.2 Boundary Particles

Boundary particles (type 5, $ID = 0$) contribute to the SPH density and pressure of their gas neighbours but receive no acceleration. Two sub-types are used:

- **Stationary walls** ($ID = 0$, $v = 0$): the nozzle walls and the right end wall. They form impenetrable boundaries.
- **Moving piston** ($ID = 0$, $v_x = -1$, enabled by `SPH_BND_PARTICLES_ALLOW_DRIFT`): compresses the gas from the left.

The pressure of boundary particles is set slightly above ambient to prevent gas from leaking through the wall.

13.3 Config.sh

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
TWO DIMS
LONG_X=30
```

```

LONG_Y=1
LONG_Z=1
SPH_BND_PARTICLES           % enable boundary particles
SPH_BND_PARTICLES_ALLOW_DRIFT % allow the piston to move

```

Listing 13.1: Config.sh for Tutorial T7b (2D).

13.4 Initial Conditions

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T07b/box.ic
```

Figure 13.1 shows the particle layout at $t = 0$. The domain extends from $x = 0$ to $x = 40$ (with `LONG_X=30` and `BoxSize = 1.33`). Three particle types are visible:

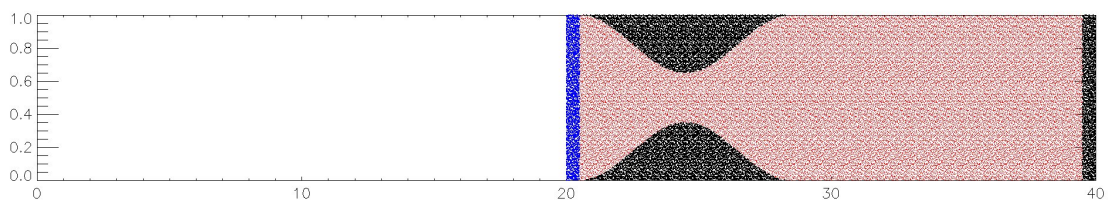


Figure 13.1: Initial conditions for Tutorial T7b. The domain is a 40×1 slab. *White region* ($0 < x < 20$): empty left chamber (gas has not yet filled it). *Blue vertical strip* ($x \approx 20$): the moving piston (boundary particles with $v_x = -1$). *Black regions*: the fixed nozzle walls forming the converging-diverging throat centred at $x \approx 23$. *Red particles*: free gas in the right half of the domain ($20 < x < 40$). Note that the piston moves right-to-left, compressing the gas through the nozzle.

13.5 Parameter File

```

PeriodicBoundariesOn      1
TimeMax                   20.0
TimeBetSnapshot           0.05
ArtBulkViscConst          0.8
UnitLength_in_cm          1.0
UnitMass_in_g             1.0
UnitVelocity_in_cm_per_s  1.0

```

```
mkdir output && sbatch runme.sh % ntasks=8, cpus-per-task=4
```

13.6 Results

Figures 13.2 and 13.3 show the simulation at two different times. The nozzle flow develops in stages: first the gas accelerates through the throat (subsonic then supersonic), then standing shocks form in the diverging section.

Exercise T7b

1. Compute the local Mach number $\mathcal{M} = v_x/c_s$ for each particle at $t = 20$. Produce a 2D colour map. Identify the subsonic ($\mathcal{M} < 1$) and supersonic ($\mathcal{M} > 1$) regions clearly.

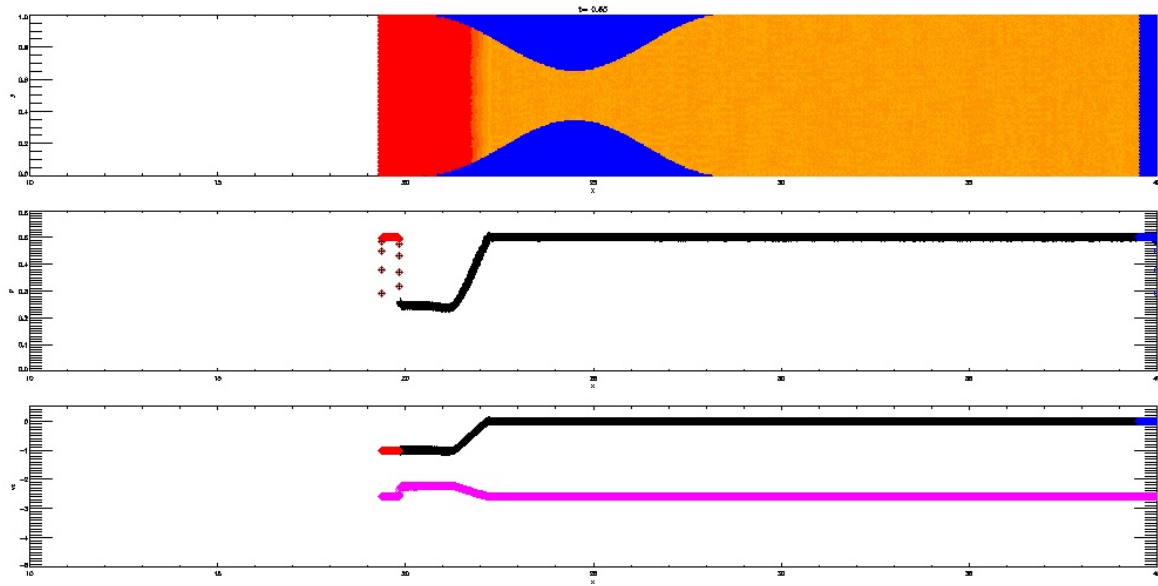


Figure 13.2: Tutorial T7b at $t \approx 0.65$ (early development). *Top panel:* 2D colour map of the velocity field (v_x), showing the converging–diverging flow structure. Red = high velocity (accelerated gas in throat), blue = low-velocity downstream region, orange = subsonic gas. *Middle panel:* pressure P vs x ; note the sharp pressure drop at the throat and the pressure recovery downstream. *Bottom panel:* x -velocity v_x (black) and y -velocity v_y (magenta) vs x ; the y -velocity remains near zero in the symmetric flow. The piston (blue marker on right) is visible at $x \approx 20$.

2. From Figure 13.3, identify the position of the normal shock (where v drops below c_s). Does it move between $t = 10$ and $t = 20$?
3. The gas cools adiabatically through the nozzle throat. From the internal energy profile, estimate the temperature drop. Is it consistent with the adiabatic relation $T \propto \rho^{\gamma-1}$?
4. What happens if you double the piston velocity? Does the shock move upstream or downstream?

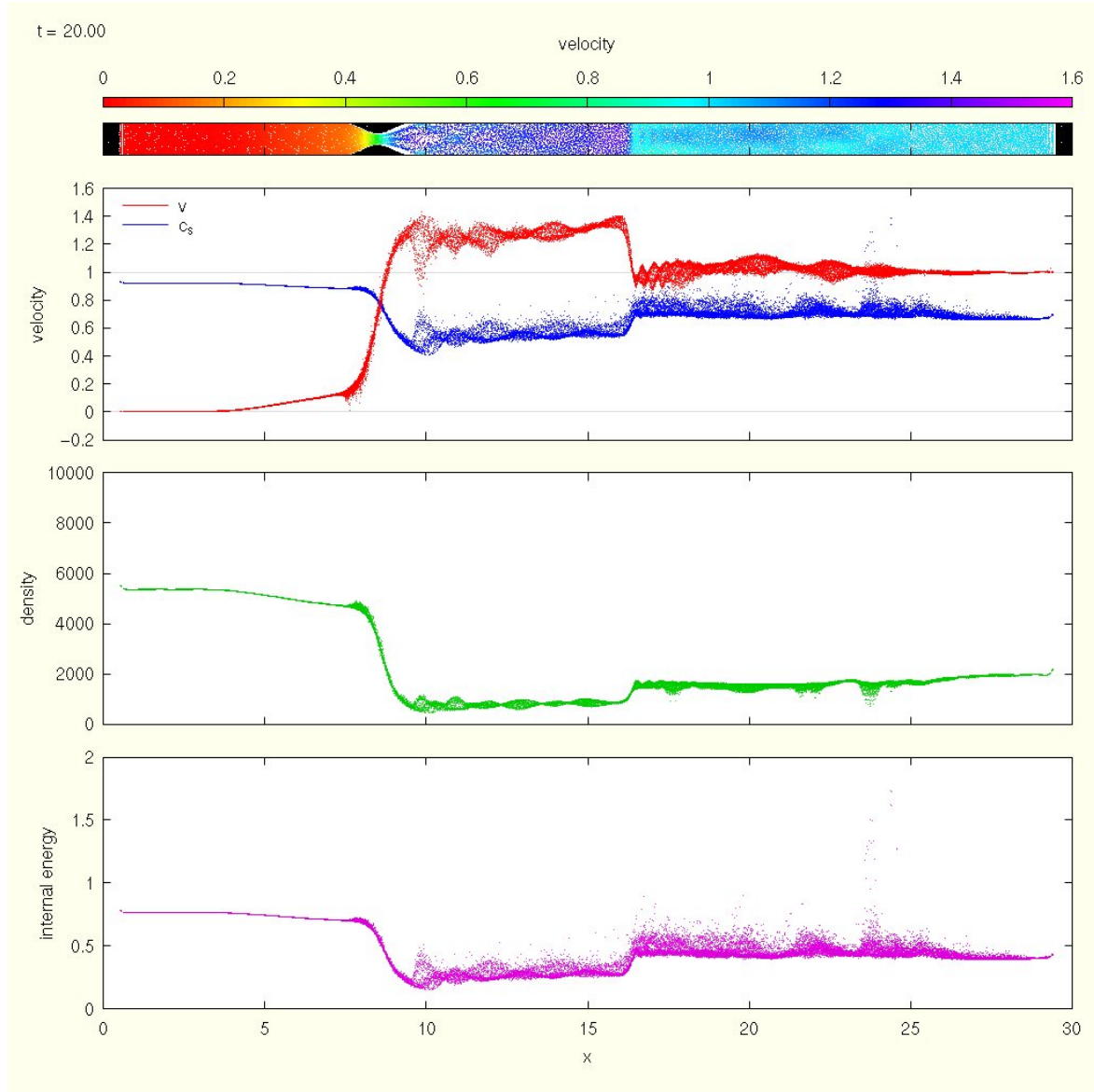


Figure 13.3: Tutorial T7b at $t = 20$ (fully developed flow). *Top strip:* 2D velocity colour map showing the complete nozzle structure from piston (left, slow red) through the supersonic throat (cyan/blue, $v > 1$) to the downstream region. *Second panel:* velocity v (red) and sound speed c_s (blue) vs x . In the throat and beyond, $v > c_s$ (supersonic); a normal shock at $x \approx 16$ decelerates the flow back to subsonic. *Third panel:* density vs x ; the density drops sharply at the throat (mass conservation with higher velocity) and rises again at the shock. *Bottom panel:* specific internal energy u vs x ; the gas cools adiabatically through the throat (energy converts to kinetic) and heats at the shock.

Chapter 14

Tutorial T8a: Thermal Conduction

T8a tests the implementation of thermal conduction in OpenGADGET3 by comparing the numerical equilibration of a temperature step with the exact analytical solution for 1D heat diffusion. All particles are fixed (boundary particles), so the process is purely diffusive with no advection.

14.1 Physical Background

For a gas with constant thermal conductivity κ , the 1D heat equation is:

$$\frac{\partial T}{\partial t} = D \frac{\partial^2 T}{\partial x^2}, \quad D = \frac{\kappa}{\rho c_v}. \quad (14.1)$$

For an initial step $T(x, 0) = T_H$ for $x < x_0$ and T_L for $x > x_0$, the solution is:

$$T(x, t) = \frac{T_H + T_L}{2} + \frac{T_H - T_L}{2} \operatorname{erf}\left(\frac{x - x_0}{\sqrt{4Dt}}\right). \quad (14.2)$$

The temperature profile evolves from a sharp step to a smooth S-curve whose width grows as $\sigma \propto \sqrt{t}$. This is the central prediction to verify. Conduction is irreversible: entropy increases monotonically even though the particles do not move.

Figures 14.1–14.3 show the three stages of this evolution.

14.2 Config.sh

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
LONG_X=20
LONG_Y=1
LONG_Z=1
SPH_BND_PARTICLES           % all particles are fixed walls
SPH_BND_PARTICLES_ALLOW_CONDUCTION % heat flows between them
CONDUCTION
CONDUCTION_CONSTANT         % constant kappa (not Spitzer T^{5/2})
```

Listing 14.1: Config.sh for Tutorial T8a.

Setting all particles as boundary particles (ID = 0) is the key design decision: it eliminates bulk fluid motion and isolates the conduction physics. If particles were free to move, pressure

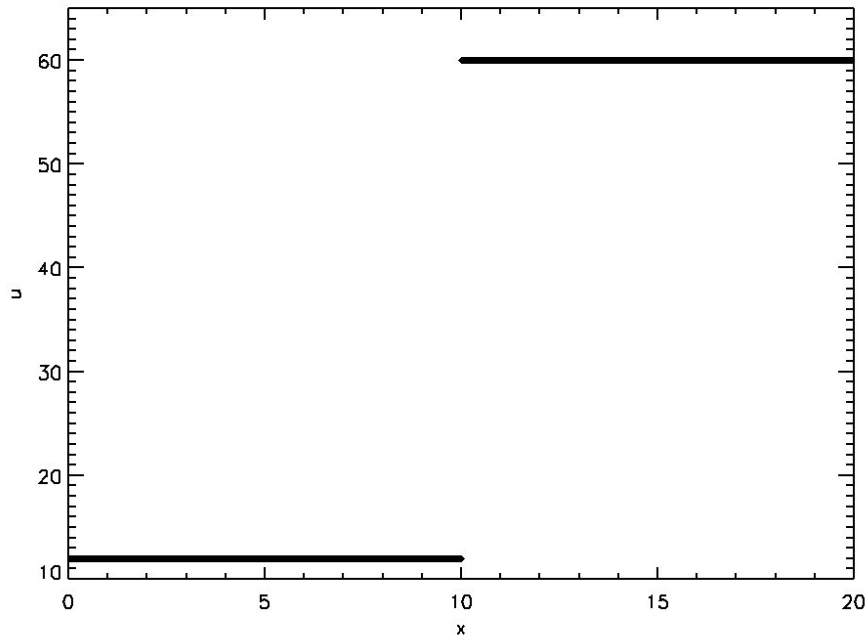


Figure 14.1: Initial conditions for Tutorial T8a: specific internal energy u vs x at $t = 0$. The slab has a sharp temperature discontinuity at $x = 10$: the left half is cold ($u \approx 10$, T_L) and the right half is hot ($u \approx 60$, $T_H \approx 6T_L$). All particles are boundary particles ($ID = 0$) and remain fixed throughout the simulation; only heat diffuses.

gradients from the temperature difference would drive bulk flow, mixing advection with diffusion and making comparison with the analytical solution impossible.

14.3 Parameter File

```

PeriodicBoundariesOn      1
TimeMax                   100.0      % conduction is slow: run for 100
                               t units
TimeBetSnapshot           1.0      % one snapshot per code time unit
UnitLength_in_cm          1.0
UnitMass_in_g             1.0
UnitVelocity_in_cm_per_s  1.0

ConductionEfficiency       0.1      % kappa = 0.1 * kappa_Spitzer(T)
MaxSizeConductionStep     0.005    % limit the conduction timestep
ArtBulkViscConst          0.01     % small: no shocks in this
                               experiment

```

Listing 14.2: Parameter file for Tutorial T8a.

14.4 Initial Conditions and Running

```

wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T08a/box.ic
mkdir output && sbatch runme.sh % ntasks=8, cpus-per-task=4

```

14.5 Analysis

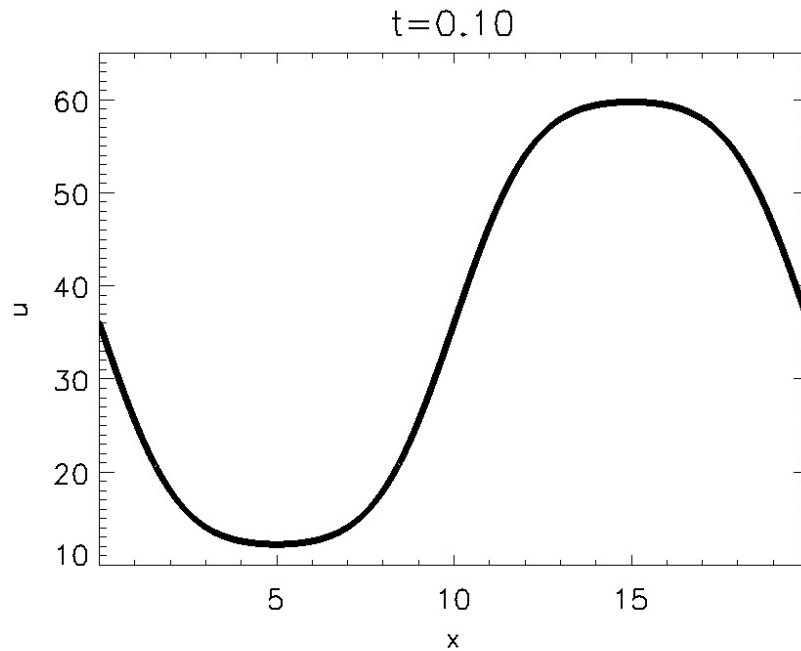


Figure 14.2: Specific internal energy $u(x)$ at $t = 0.10$. The sharp initial step has diffused into a smooth S-curve, consistent with the analytical error-function solution (equation 14.2). The profile is symmetric about $x_0 = 10$ and spans $u \in [10, 60]$. Comparison with the analytical prediction is the primary analysis task of this tutorial.

```

1 import g3read as g3, numpy as np, matplotlib.pyplot as plt
2 from scipy.special import erf
3
4 % --- Plot 4 snapshots and compare with analytical solution ---
5 fig, axes = plt.subplots(2, 2, figsize=(10, 8), sharex=True, sharey=
6     True)
7 axes = axes.ravel()
8 snap_nums = [5, 20, 50, 100]
9
10 for ax, sn in zip(axes, snap_nums):
11     f = g3.GadgetFile(f'output/snap_{sn:03d}')
12     pos = f.read_new('POS ', 0)[: , 0]
13     u = f.read_new('U ', 0)
14     t = f.header.time
15
16     % Sort by x and plot simulation
17     idx = np.argsort(pos)
18     ax.scatter(pos[idx]*20, u[idx], s=1, alpha=0.3, c='navy', label='
19         Simulation')
20
21     % Analytical solution
22     L = 20.0; x0 = L/2; D = 0.1
23     TH, TL = 60.0, 10.0
24     x_an = np.linspace(0, L, 400)
25     T_an = (TH+TL)/2 + (TH-TL)/2 * erf((x_an-x0)/np.sqrt(4*D*t+1e
26         -10))
27     ax.plot(x_an, T_an, 'r--', lw=2, label='Analytical' if sn==
28         snap_nums[0] else '')
29     ax.set_title(f't = {t:.0f}'); ax.set_ylabel('u')

```

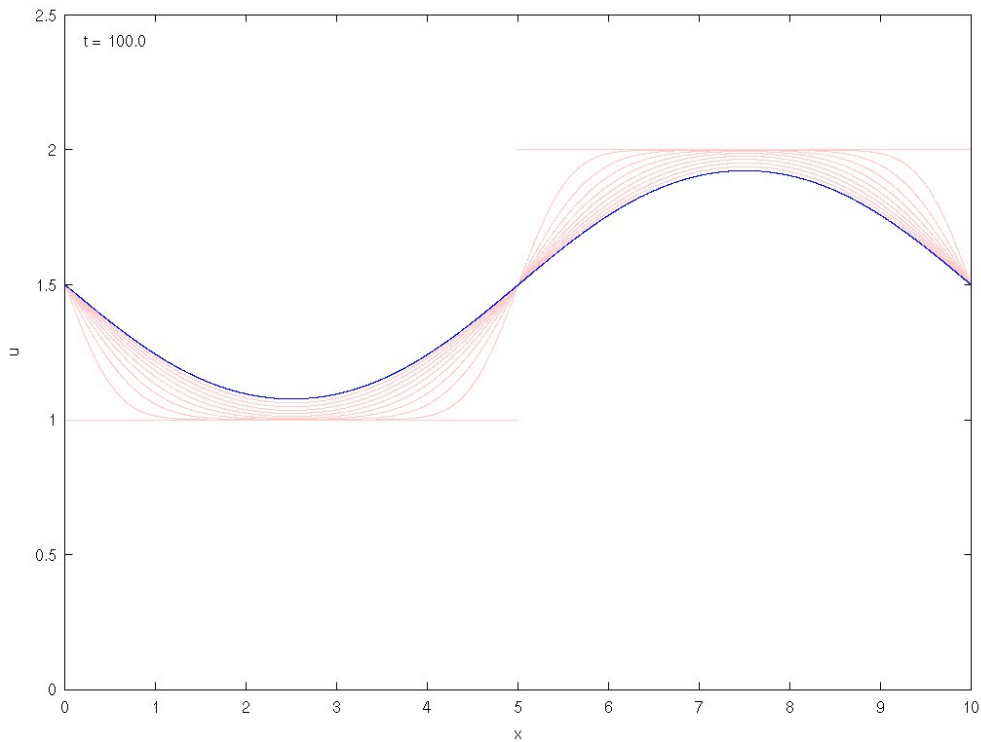


Figure 14.3: Full time evolution of the temperature profile in Tutorial T8a. Pink curves: snapshots at intermediate times, from $t \approx 0$ (outermost, close to the step function) to $t = 100$ (innermost, nearly flat). Blue curve: the final state at $t = 100$. The profile converges toward the uniform equilibrium value $u_\infty = (u_H + u_L)/2 \approx 35$ (energy conservation), but because the slab is periodic, the profile at $t = 100$ still retains a sinusoidal shape rather than being completely flat.

```

26
27 for ax in axes[-2:]: ax.set_xlabel('x')
28 axes[0].legend(fontsize=9)
29 plt.suptitle('T8a: thermal conduction'); plt.tight_layout()
30 plt.savefig('conduction_snapshots.png', dpi=150)
31
32 % --- Measure diffusion width vs time ---
33 snaps = sorted(__import__('glob').glob('output/snap_???'))
34 times, widths = [], []
35 for s in snaps:
36     f = g3.GadgetFile(s)
37     pos = f.read_new('POS ', 0)[: , 0] * 20
38     u = f.read_new('U ', 0)
39     % Profile width: fit erf to extract sigma
40     from scipy.optimize import curve_fit
41     def erf_model(x, x0, sigma, A, B):
42         return A + B*erf((x - x0)/sigma)
43     try:
44         popt, _ = curve_fit(erf_model, np.sort(pos), u[np.argsort(pos)]
45                             ],
46                             p0=[10, 1.0, 35, 25], maxfev=2000)
47         widths.append(popt[1])
48         times.append(f.header.time)
49     except:
50         pass

```

```

50 times = np.array(times); widths = np.array(widths)
51 fig2, ax2 = plt.subplots()
52 ax2.plot(times, widths, 'o', ms=3, label='Measured  $\sigma$ ')
53 D_fit = np.polyfit(np.log(times[times>1]), np.log(widths[times>1]), 1)
54 ax2.plot(times, np.exp(D_fit[1])*times**D_fit[0], '--r',
55         label=f'Fit:  $\sigma \propto t^{{D\_fit[0]:.2f}}$ ')
56 ax2.set_xlabel('Time'); ax2.set_ylabel('Profile width  $\sigma$ ')
57 ax2.legend(); ax2.set_xscale('log'); ax2.set_yscale('log')
58 plt.tight_layout(); plt.savefig('conduction_width.png', dpi=150)
59 print(f'Measured exponent: {D_fit[0]:.3f} (expected: 0.500)')
60

```

Listing 14.3: Comparing conduction simulation with analytical solution.

Exercise T8a

1. From Figure 14.3, at what time does the profile reach $u_\infty \approx (u_H + u_L)/2$ at the box centre? Compare with the diffusion timescale $\tau \sim L^2/(4D)$.
2. Verify the $\sqrt{t}$ scaling of the profile width: fit equation (14.2) to snapshots at $t = 1, 5, 10, 20, 50$ and plot $\sigma(t)$ on a log-log scale. Measure the power-law exponent.
3. Change ConductionEfficiency from 0.1 to 0.01 and 1.0. Does the diffusion timescale scale as $1/\kappa$ as expected?
4. Compute the total entropy $S \propto \int \ln(u/\rho^{\gamma-1}) dm$ as a function of time. Does it increase monotonically? What does this confirm about the second law of thermodynamics?

Chapter 15

Tutorial T8b: Heat Transport and the Ancient Air Conditioner

T8b is a creative extension of both T7b (flow through a nozzle) and T8a (thermal conduction). It combines boundary-particle fluid dynamics with heat transport to simulate a mechanism that the architects of ancient amphitheatres may have exploited to cool their interiors: a conical entrance that accelerates air through a constriction, lowering its temperature. The tutorial asks you to set up a custom geometry, run the simulation with and without conduction, and compare the temperature in the upstream chamber.

15.1 Physical Motivation

15.1.1 The refrigeration cycle

A refrigeration cycle (as in a fridge or air conditioner) works by exploiting the thermodynamics of a compressible fluid. In the simplest idealised picture: a gas is compressed adiabatically (temperature rises), heat is expelled to the environment, the gas then expands adiabatically through a constriction (temperature drops), and the cold gas absorbs heat from the interior to be cooled. The cycle then repeats.

The key element is the adiabatic expansion through the constriction. As gas is forced from a large volume through a narrow opening, it must accelerate (continuity equation: $\rho_1 v_1 A_1 = \rho_2 v_2 A_2$, so $v_2 > v_1$ when $A_2 < A_1$). In the adiabatic limit, kinetic energy is gained at the expense of internal energy: the gas cools.

15.1.2 Ancient theatres and natural ventilation

Roman and Greek theatres were built with conically shaped entrance corridors (vomitoria). Beyond the aesthetics and crowd-flow function, these conical passages may have produced a measurable cooling effect: warm outside air was forced through the narrowing, accelerated, partially cooled, and emerged into the seating area at a lower temperature. The effect is subtle (not a true refrigeration cycle since there is no compression stage) but the combination of adiabatic expansion and heat exchange with the cooler stone walls could provide meaningful cooling on hot summer days.

This tutorial simulates a simplified version of this geometry in 2D, comparing two cases: with and without thermal conduction switched on.

15.1.3 The adiabatic temperature change

For a compressible ideal gas undergoing adiabatic flow through a constriction, Bernoulli's equation (with the enthalpy term for compressible flow) gives:

$$\frac{v^2}{2} + \frac{\gamma}{\gamma-1} \frac{P}{\rho} = \text{const.} \quad (15.1)$$

Combining with the adiabatic relation $P \propto \rho^\gamma$ and the ideal gas law $P = \rho k_B T / (\mu m_p)$, the temperature in the accelerated jet is lower than in the ambient medium:

$$T_2 = T_1 \left(\frac{P_2}{P_1} \right)^{(\gamma-1)/\gamma} = T_1 \left[1 - \frac{(\gamma-1)}{2\gamma} \mathcal{M}_2^2 \left(1 - \left(\frac{A_2}{A_1} \right)^2 \right) \right], \quad (15.2)$$

where $\mathcal{M}_2$ is the Mach number in the throat and A_1 , A_2 are the cross-sectional areas of the upstream chamber and the throat respectively. For a subsonic flow ($\mathcal{M}_2 \sim 0.1\text{--}0.3$) the cooling is modest but measurable.

The role of conduction is to communicate this temperature drop back to the upstream chamber: without conduction, the cold gas in the throat does not affect the chamber temperature; with conduction, heat can diffuse upstream through the SPH particle distribution.

15.2 Geometry and Setup

The geometry is a modified version of the T7b nozzle, with three distinct regions:

1. **Left chamber** ($0 < x \lesssim 5$, full cross-section): the “theatre interior”, containing gas at rest at uniform temperature and pressure.
2. **Moving left wall** (at $x \approx 0$, moving in $+x$): the “piston” that drives gas through the nozzle. This is a set of boundary particles ($\text{ID} = 0$) with a fixed inward velocity.
3. **Fixed nozzle walls** (constriction at $x \approx 5\text{--}10$): boundary particles forming converging walls that reduce the cross-section to $\sim 1/3$ of the chamber size.
4. **Fixed right wall with a hole** ($x \approx 10\text{--}11$): the right part of the original T7b moving wall is now stationary, but with an opening at the centre that allows gas to pass through.
5. **Downstream region** ($x \gtrsim 11$): gas that has passed through the nozzle and exits to the right.

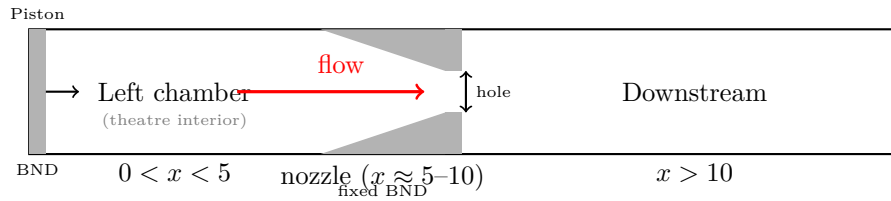


Figure 15.1: Schematic of the T8b geometry. The piston (left boundary particle wall) drives gas rightward through the converging nozzle. The right wall is fixed but has a central opening allowing gas to pass through into the downstream region. This mimics the entrance architecture of ancient amphitheatres.

Figure 15.2 shows the actual particle setup, with the gas particles (red), nozzle boundary particles (black), and the fixed right wall with its central opening (blue).

15.3 Config.sh

T8b uses the same flags as T7b (§13) with the addition of the conduction flags:

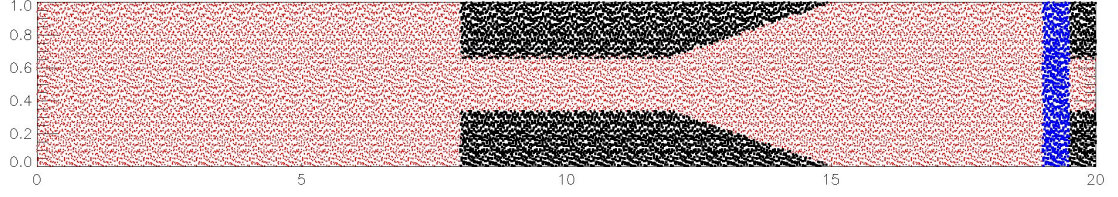


Figure 15.2: Initial conditions for Tutorial T8b at $t = 0$. Red particles: free gas in the left chamber and downstream region. Black particles: fixed boundary particles forming the converging nozzle walls. Blue particles: the stationary right wall with a central opening. The domain is a 30×1 slab (LONG_X=30).

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
LONG_X=30
LONG_Y=1
LONG_Z=1
SPH_BND_PARTICLES           % enable boundary particles
SPH_BND_PARTICLES_ALLOW_DRIFT % allow the piston to move
SPH_BND_PARTICLES_ALLOW_CONDUCTION % allow heat to flow through
    boundaries
CONDUCTION
CONDUCTION_CONSTANT         % use constant kappa for this test
```

Listing 15.1: Config.sh for Tutorial T8b.

You will run two simulations:

Run A: with CONDUCTION and CONDUCTION_CONSTANT (as above). This is the full physics case.

Run B: without CONDUCTION and CONDUCTION_CONSTANT (comment them out). This is the adiabatic control case.

Each run requires a separate compilation of Config.sh.

15.4 Parameter File

```
InitCondFile      nozzle.ic  % IC generated by the setup
    script
OutputDir         ./output

PeriodicBoundariesOn 1
TimeMax           100.0
TimeBetSnapshot    0.1

UnitLength_in_cm   1.0
UnitMass_in_g      1.0
UnitVelocity_in_cm_per_s 1.0
BoxSize            1.0      % LONG_X=30 gives physical size 30

% Conduction parameters (Run A only; comment out for Run B)
ConductionEfficiency 100    % full Spitzer value (as in tutorial)
MaxSizeConductionStep 0.01
```

ArtBulkViscConst	0.8
------------------	-----

Listing 15.2: Parameter file for Tutorial T8b.

Tip

The parameter `ConductionEfficiency` = 100 (not 0.1 as in T8a) is intentionally large in this tutorial: you want to see a clear effect. In a physically realistic simulation you would use a value between 0.1 and 1.

15.5 Setting Up the Initial Conditions

The IC setup script is an extension of the T7b nozzle setup. The key differences are:

1. The domain is $30 \times 1 \times 1$ (vs. $30 \times 1 \times 1$ for T7b, but with a shorter tube so that the left chamber occupies more volume).
2. The moving wall is split: the left part moves at a reduced velocity $v_{\text{piston}} \approx 0.1$ (slower than T7b); the right part is stationary with a central hole of height $\sim 1/3$ of the domain.
3. The nozzle converges from full height (1.0) to the hole width over a length of ~ 5 code units.

The Fortran setup script provided with the tutorial:

```
% Compile:
ifx -g -traceback -check all -fpe0 -o nozzlesetup nozzlesetup.f90
% Requires glass.txt from T4 (or T7b) as particle template

./nozzlesetup          % writes nozzle.ic

% Verify the geometry:
gnuplot grid2.plt      % should reproduce Figure~\ref{fig:t8b_ics}
```

The equivalent Python setup follows the same logic as T7b (Chapter 13) with the modifications described above. The essential parameters to change in the IC generation are:

```
1  # --- Geometry parameters for T8b ---
2  L_total = 30.0    # total domain length (with LONG_X=30, BoxSize=1)
3  L_chamber = 5.0   # length of the left chamber
4  H = 1.0          # domain height
5  nozzle_L = 5.0    # nozzle length (converging section)
6  hole_h = 0.33    # height of the hole in the right wall [fraction of H]
7  v_piston = -0.10  # piston velocity (negative = moving right in code
   coords)
8
9  # --- Particle identification ---
10 # Free gas:      ID != 0, in left chamber and downstream
11 # Piston wall:   ID = 0, v_x = v_piston (moving)
12 # Nozzle walls:  ID = 0, v_x = 0 (fixed)
13 # Right wall:    ID = 0, v_x = 0 (fixed), only top/bottom strips (hole
   in centre)
14
15 # --- Classify particles ---
16 def particle_type(x, y):
17     # Piston: leftmost slab
18     if x < 0.3:
19         return 'piston'
```

```

20  # Nozzle walls: converging region top and bottom
21  if L_chamber < x < L_chamber + nozzle_L:
22      frac = (x - L_chamber) / nozzle_L      # 0 at start, 1 at end
23      y_wall = 0.5 - (0.5 - hole_h/2) * frac  # wall profile
24      if y < y_wall or y > H - y_wall:
25          return 'nozzle_wall'
26  # Right wall with hole
27  if L_chamber + nozzle_L <= x <= L_chamber + nozzle_L + 0.5:
28      y_centre = H / 2
29      if abs(y - y_centre) > hole_h / 2:
30          return 'right_wall'
31  # Everything else is free gas
32  return 'gas'

```

Listing 15.3: Key modifications to the T7b IC script for T8b.

15.6 Running the Simulations

Run both configurations (A: with conduction, B: without) in separate directories, keeping the same IC file:

```

% Run A: with conduction
mkdir RunA && cp nozzle.ic RunA/ && cd RunA
% Config.sh has CONDUCTION and CONDUCTION_CONSTANT
cd ../Hydro-OpenGadget3 && make -j && cd ../RunA
mkdir output && sbatch runme.sh && cd ..

% Run B: no conduction
mkdir RunB && cp nozzle.ic RunB/ && cd RunB
% Edit Config.sh: comment out CONDUCTION and CONDUCTION_CONSTANT
% Then recompile
cd ../Hydro-OpenGadget3 && make -j && cd ../RunB
mkdir output && sbatch runme.sh

```

Both runs use:

```

#SBATCH --ntasks=4
#SBATCH --cpus-per-task=4

```

The runs take ~10–20 minutes each for ~100k particles.

15.7 Analysis: Temperature in the Chamber

The central question is: does the upstream chamber cool down, and does conduction enhance this effect?

```

1  import g3read as g3
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import glob
5
6  def mean_chamber_T(snap_path, L_chamber=5.0, gamma=5.0/3.0,
7                      vel_unit=1e5, k_B=1.3806e-16, m_p=1.6726e-24, mu
8                      =0.588):
9      """Return mean temperature in the left chamber."""
10     f = g3.GadgetFile(snap_path)

```

```

10     pos = f.read_new('POS ', 0)
11     u    = f.read_new('U   ', 0)
12     ids  = f.read_new('ID  ', 0)
13
14     # Select free gas particles (ID != 0) in the left chamber
15     mask = (ids != 0) & (pos[:, 0] * 30 < L_chamber)
16     if mask.sum() == 0:
17         return f.header.time, np.nan
18
19     u_cgs = u[mask] * vel_unit**2
20     T = (gamma - 1) * mu * m_p * u_cgs / k_B
21     return f.header.time, T.mean()
22
23 # Load time series for both runs
24 snaps_A = sorted(glob.glob('RunA/output/snap_???'))
25 snaps_B = sorted(glob.glob('RunB/output/snap_???'))
26
27 times_A, T_A = zip(*[mean_chamber_T(s) for s in snaps_A])
28 times_B, T_B = zip(*[mean_chamber_T(s) for s in snaps_B])
29
30 fig, ax = plt.subplots(figsize=(8, 4))
31 ax.plot(times_A, np.array(T_A)/T_A[0], label='With conduction', lw
32         =2)
33 ax.plot(times_B, np.array(T_B)/T_B[0], label='Without conduction', lw
34         =2, ls='--')
35 ax.set_xlabel('Time [code units]')
36 ax.set_ylabel(r'$T_{\mathrm{chamber}}(t)/T_{\mathrm{chamber}}(0)$')
37 ax.set_title('Chamber temperature: ancient air conditioning effect')
38 ax.legend()
39 ax.grid(True, alpha=0.3)
40 plt.tight_layout()
41 plt.savefig('chamber_temperature.png', dpi=150)

```

Listing 15.4: Comparing chamber temperature with and without conduction.

15.7.1 Profiles along the x-axis

Figure ?? shows the velocity, density, and internal energy profiles along the x -axis at $t = 50$ for both runs.

15.7.2 What to look for

Without conduction (Run B, Figure 15.4): The internal energy drops sharply at the nozzle throat, consistent with adiabatic cooling ($u \propto P/\rho \propto \rho^{\gamma-1}$). This cold region is isolated: upstream and downstream gas remain at the initial temperature. There is no net cooling of the left chamber.

With conduction (Run A, Figure 15.3): Heat diffuses from the warm surrounding gas into the cold throat, and from the throat back towards the left chamber. The temperature profile is smoother. Depending on `ConductionEfficiency`, the upstream chamber temperature can show a measurable decrease relative to the no-conduction case.

The physical message: the ancient air conditioner requires both the adiabatic cooling (the nozzle geometry) and a mechanism to transport that cold to the interior (equivalent to conduction, or in the real building, cold stone walls). Without both elements, the cooling effect does not reach the occupied space.

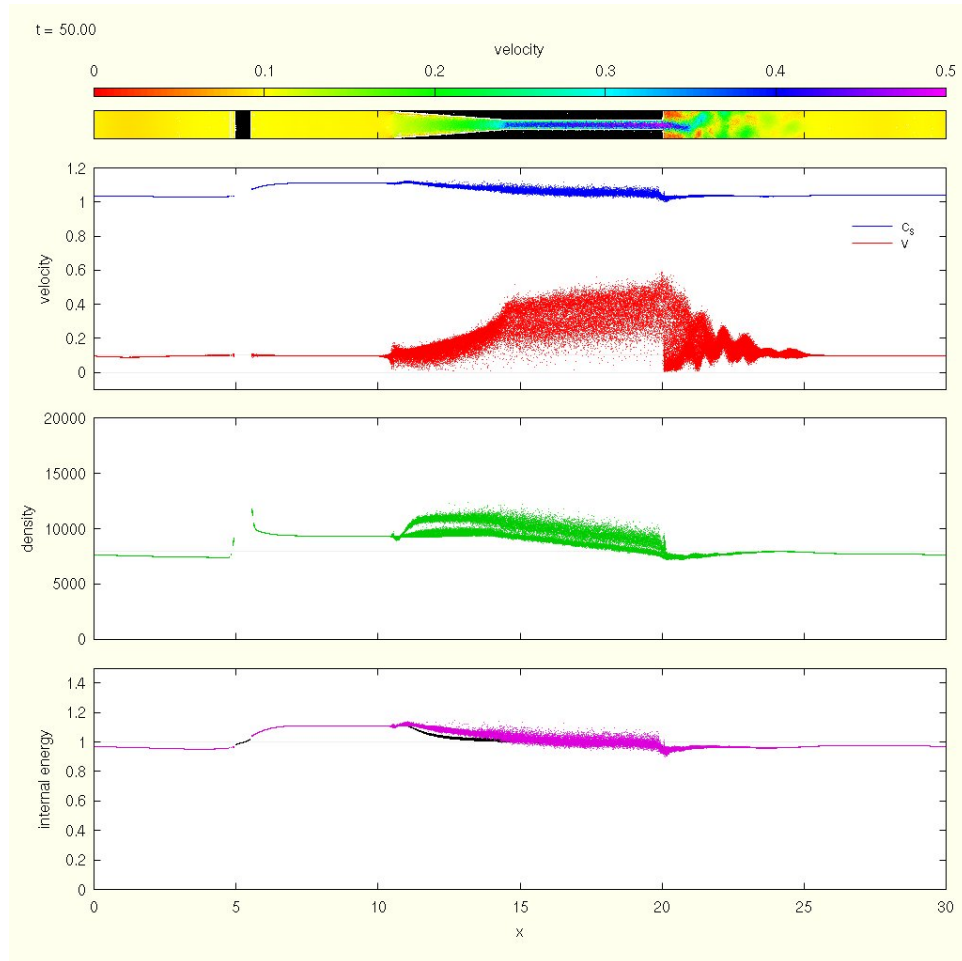


Figure 15.3: Tutorial T8b results at $t = 50$. Each panel shows (top) velocity and sound speed vs. x , (middle) SPH density, (bottom) specific internal energy. Red/magenta points are SPH particles coloured by v_x in the top strip. These plots presents the case including conduction (`ConductionEfficiency = 100`).

15.8 Programming Goals

The main skill developed in T8b is setting up complex, non-trivial particle geometries. Specifically:

- Combining multiple boundary-particle zones (moving piston, fixed nozzle walls, fixed wall with hole) in a single IC file.
- Implementing a split wall: some particles have $v_x \neq 0$ (moving) and others have $v_x = 0$ (stationary).
- Controlling the constriction geometry parametrically so that the nozzle angle and hole size can be varied systematically.

Key commands for this tutorial

```
% Two separate compilation passes:
% Run A: Config.sh with CONDUCTION + CONDUCTION_CONSTANT +
    SPH_BND_*
% Run B: Config.sh without CONDUCTION (same otherwise)
```

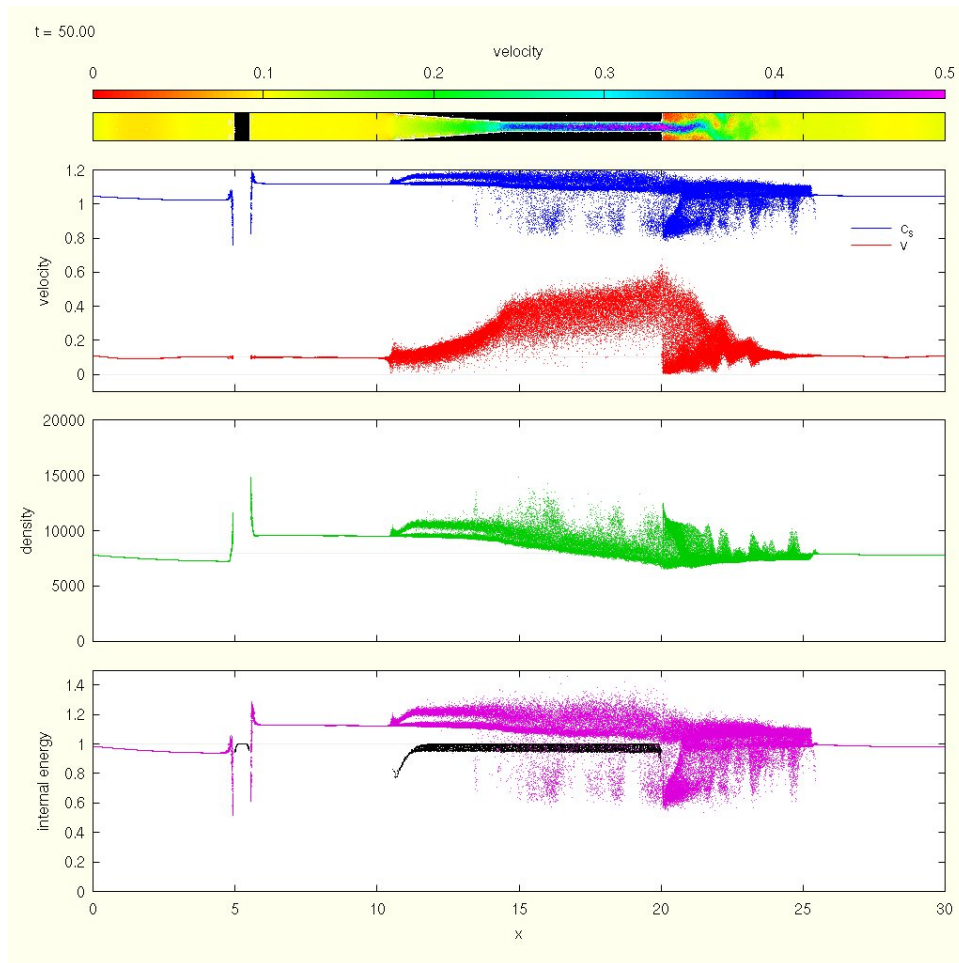


Figure 15.4: As figure 15.3 but without conduction (`ConductionEfficiency` = 0). Compare the internal energy profiles: conduction smooths out the temperature drop across the nozzle, while without conduction the cooling is confined to the throat.

```
% IC: extended from T7b with split wall and shorter tube
% Parameter file: BoxSize=1, LONG_X=30, ConductionEfficiency=100
(Run A)
% Key analysis: mean T in left chamber vs time, compare Run A vs
Run B
```

Exercise T8b

1. Run both configurations and plot the mean chamber temperature $T(t)/T(0)$ as a function of time. Is there a statistically significant difference between Run A and Run B at $t = 50$?
2. Vary `ConductionEfficiency` between 0, 1, 10, and 100. Does the chamber cooling scale linearly with `ConductionEfficiency`?
3. The conduction timescale across the nozzle length ℓ is $\tau \sim \ell^2/D$ where $D = \kappa/(\rho c_v)$. Estimate τ for each value of `ConductionEfficiency` and compare with the simulation time at which you first see a temperature difference in the chamber.
4. Reduce the piston velocity by a factor of 2. Does the chamber cool more slowly, more quickly, or at the same rate? Explain in terms of the Mach number in the

throat.

5. (Advanced) Vary the hole size (constriction ratio A_2/A_1) between 1/6, 1/3, and 1/2. How does the temperature drop at the throat scale with the constriction ratio? Compare with the prediction from equation (15.2).

Chapter 16

Tutorial T9a: Navier–Stokes Viscosity (Couette Flow)

T9a introduces physical Navier–Stokes viscosity by simulating Couette flow — the laminar flow of a viscous fluid between two parallel plates moving in opposite directions. The simulation is compared with the exact analytical steady-state profile.

16.1 Physical Background

For an incompressible viscous fluid between two plates at $x = 0$ and $x = d$ moving at velocities $v_y = -V$ and $v_y = +V$ respectively, the Navier–Stokes equations give a steady-state velocity profile:

$$v_y(x) = V \left(\frac{2x}{d} - 1 \right), \quad (16.1)$$

a linear profile that increases from $-V$ at the left wall to $+V$ at the right wall. The approach to this steady state is governed by the kinematic viscosity $\nu = \eta/\rho$ and is characterised by a timescale $\tau \approx d^2/(\pi^2\nu)$ for the fundamental diffusion mode.

The Spitzer dynamic viscosity of a hot fully-ionised plasma is:

$$\eta_{\text{Sp}} \propto T^{5/2}, \quad (16.2)$$

and is potentially important for suppressing Kelvin–Helmholtz instabilities at galaxy cluster merger shocks (Roediger et al., 2013).

16.2 Config.sh

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
TWO DIMS
SPH_BND_PARTICLES           % two plate walls at left and right
SPH_BND_PARTICLES_ALLOW_DRIFT % plates move at constant velocity
NAVIERSTOKES
NAVIERSTOKES_CONSTANT       % constant viscosity (not Spitzer T
                             ~{5/2})
```

Listing 16.1: Config.sh for Tutorial T9a.

16.3 Initial Conditions

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T09a/box.ic
```

Figure 16.1 shows the initial particle layout. The two side strips of blue diamonds are the boundary particles (the moving plates); the white diamond lattice between them is the gas initially at rest.

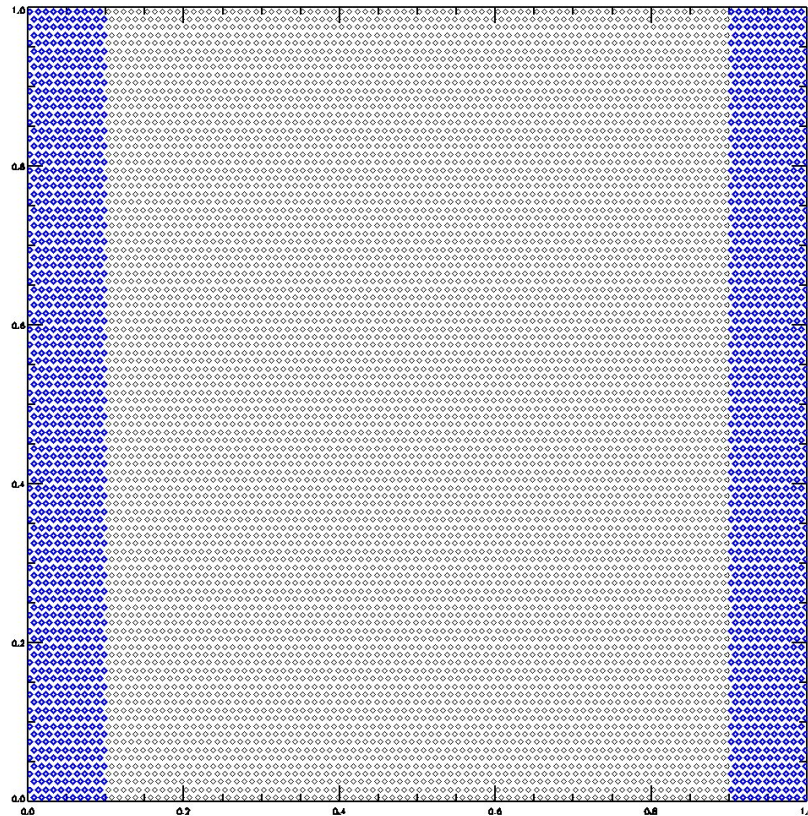


Figure 16.1: Initial conditions for Tutorial T9a. Blue filled diamonds (left and right strips): boundary particles with ID = 0 representing the two plates. The left plate moves at $v_y = -V$ and the right plate at $v_y = +V$. White open diamonds (interior): gas particles at rest ($v_y = 0$), uniformly distributed in the 2D domain.

16.4 Parameter File

```
PeriodicBoundariesOn      1
TimeMax                   200.0      % long run: viscous equilibration
                             is slow
TimeBetSnapshot           1.0
UnitLength_in_cm          3.085678e21
UnitMass_in_g              1.989e43
UnitVelocity_in_cm_per_s  1e5

ShearViscosityTemperature 5.0e7      % reference temperature for
                             Spitzer eta
FractionSpitzerViscosity   1.0       % 1.0 = full Spitzer value
```

Listing 16.2: Parameter file for Tutorial T9a.

```
mkdir output && sbatch runme.sh % ntasks=8, cpus-per-task=4
```

16.5 Results

Figures 16.2 and 16.3 show the evolution of the flow from the initial transient to the steady Couette state.

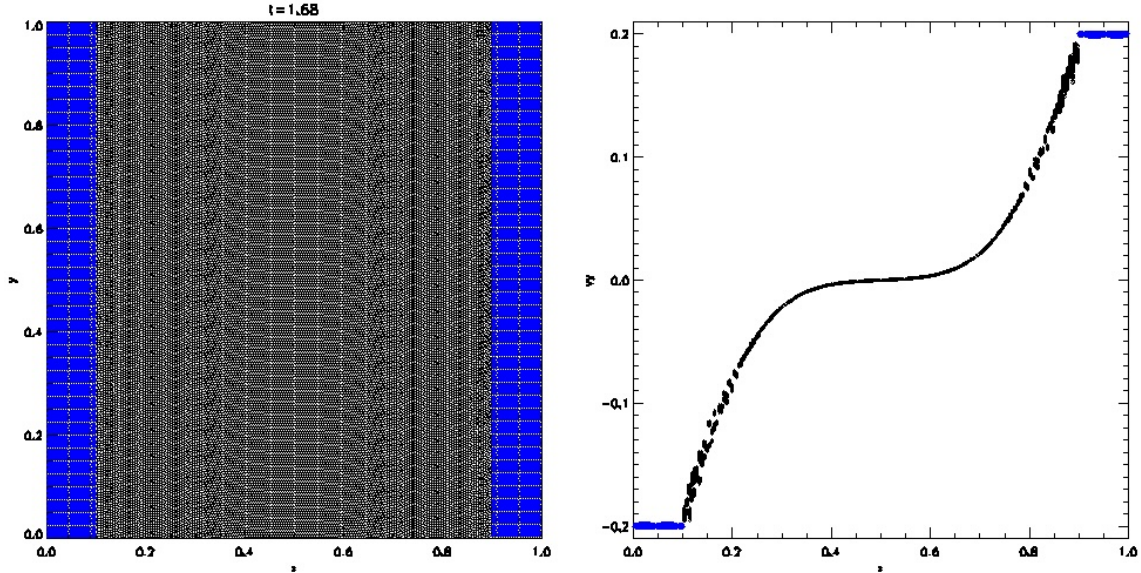


Figure 16.2: Tutorial T9a at $t = 1.68$ (developing flow). *Left panel:* 2D particle positions coloured by v_y . The blue walls are clearly visible at $x \approx 0.1$ and $x \approx 0.9$; the interior gas has begun to be dragged by the plates but the flow has not yet reached steady state. *Right panel:* v_y vs x for all particles. The profile is S-shaped rather than linear: the momentum has diffused inward from both walls but has not yet reached the box centre, leaving a near-zero-velocity core region.

16.6 Analysis

```
1 import g3read as g3, numpy as np, matplotlib.pyplot as plt
2
3 f = g3.GadgetFile('output/snap_100') # use a late snapshot
4 pos = f.read_new('POS ', 0)
5 vel = f.read_new('VEL ', 0)
6 ids = f.read_new('ID ', 0)
7
8 # Only fluid particles (ID != 0)
9 mask = ids != 0
10 x = pos[mask, 0]
11 vy = vel[mask, 1]
12
13 # Bin in x
14 Nbin = 40
15 L = f.header.BoxSize
16 xbins = np.linspace(0, L, Nbin+1)
17 x_mid = 0.5*(xbins[:-1]+xbins[1:])
18 vy_mean = np.array([vy[(x>=xbins[i])&(x<xbins[i+1])].mean()])
```

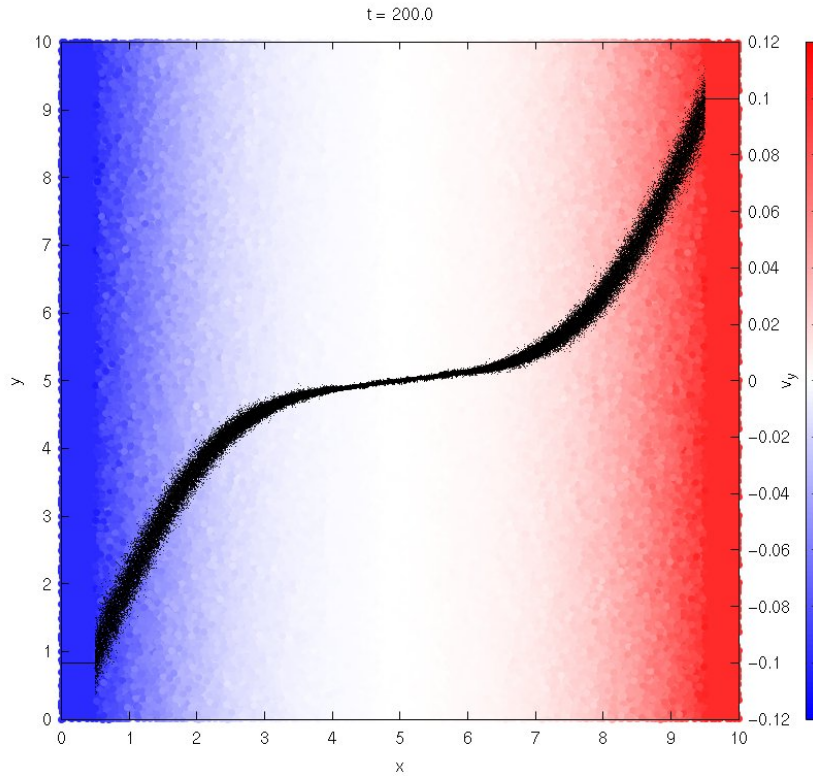


Figure 16.3: Tutorial T9a at $t = 200$ (steady-state Couette flow). The 2D colour map of v_y (red = $+0.12$, blue = -0.12 , white = 0) shows the characteristic linear gradient from the right plate ($v_y = +V$) to the left plate ($v_y = -V$). The colour gradient is uniform across x , confirming the linear Couette profile $v_y = V(2x/d - 1)$. The thin black diagonal strip is an artefact of the boundary particle geometry at the periodic y -boundaries. The domain size is 10×10 (note axis labels).

```

19         if ((x>=xbins[i])&(x<xbins[i+1])).sum() > 0 else
20             np.nan
21         for i in range(Nbin)])
22
23 V = 1.0
24 vy_theory = V * (2*x_mid/L - 1)
25
26 fig, ax = plt.subplots(figsize=(6, 5))
27 ax.plot(x_mid, vy_mean, 'o-', ms=4, label='Simulation')
28 ax.plot(x_mid, vy_theory, 'r--', lw=2, label=r'Couette: $V(2x/d-1)$')
29 ax.set_xlabel('x'); ax.set_ylabel(r'$v_y$')
30 ax.set_title(f't = {f.header.time:.0f}')
31 ax.legend(); plt.tight_layout()
32 plt.savefig('couette_profile.png', dpi=150)
33
34 # Residual: how close is steady state?
35 good = ~np.isnan(vy_mean)
36 rms = np.sqrt(np.mean((vy_mean[good] - vy_theory[good])**2))
37 print(f'RMS deviation from linear: {rms:.4f} (wall velocity V={V})')
38 print(f'Relative error: {rms/V*100:.1f}%')

```

Listing 16.3: Measuring and comparing the Couette velocity profile.

Exercise T9a

1. From Figure 16.2, estimate when the S-shaped profile transitions to the linear Couette profile. Compare with the theoretical timescale $\tau \approx d^2/(\pi^2\nu)$, where $\nu = \eta/\rho$ and η comes from `ShearViscosityTemperature` and `FractionSpitzerViscosity`.
2. At $t = 200$, compute the RMS deviation of the simulated v_y from the linear profile. Is it below 5%?
3. Increase `FractionSpitzerViscosity` to 5.0 and reduce to 0.1. How does the equilibration time scale with viscosity? Is it consistent with $\tau \propto 1/\nu$?
4. (Advanced) Plot the time evolution of v_y at the box centre $x = d/2$. Fit an exponential approach to zero and extract a timescale. Compare with the theoretical result.

Chapter 17

Tutorial T9b: Viscous Jet and Kelvin–Helmholtz Instability

T9b is the most physically demanding tutorial: a dense jet penetrates a lighter ambient medium under gravity, and mixing at the jet boundary is controlled by the Kelvin–Helmholtz instability (KHI). The tutorial compares four different hydrodynamic configurations to isolate numerical and physical effects on the instability.

17.1 Physical Background

The KHI develops at the interface between two fluids in relative motion. For a sharp velocity discontinuity $\Delta v = v_1 - v_2$, linear theory gives a growth rate for mode wavenumber k :

$$\sigma = k \frac{\rho_1 \rho_2}{(\rho_1 + \rho_2)} |\Delta v|. \quad (17.1)$$

Physical viscosity suppresses modes above $k_{\text{crit}} \sim |\Delta v|/\nu$.

Standard SPH artificially suppresses the KHI due to: (i) spurious surface tension at contact discontinuities from zeroth-order density errors, and (ii) artificial viscosity damping shear flows. The MFM solver eliminates both by using a Riemann solver at particle interfaces. Artificial conductivity partially mitigates the first effect within SPH.

17.2 Initial Conditions

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T09b/box.ic
```

Figure 17.1 shows the particle setup at $t = 0$. The domain is a 1×3 slab (`LONG_Y=3`): the upper two-thirds contain the jet structure (dense jet column + boundary-particle collimating walls), while the lower third contains lighter ambient gas at rest.

17.3 Four Simulations to Compare

```
PERIODIC
NOGRAVITY
WENDLAND_C2_KERNEL
TWO DIMS
SPH_BND_PARTICLES
SPH_BND_PARTICLES_ALLOW_DRIFT
```

Table 17.1: Four T9b simulations, each adding one physical ingredient.

Run	Config.sh additions	Physical?	KHI expected
1	Standard SPH	No	Strongly suppressed
2	+ ARTIFICIAL_CONDUCTIVITY	No	Partially recovered
3	+ NAVIERSTOKES	Yes	Suppressed at high k
4	USE_MESHLESS_FINITE_MASS	No	Correctly developed

```
LONG_X=1
LONG_Y=3
LONG_Z=1
CONSTANT_GRAVITY_Y=-0.1
```

Listing 17.1: Base Config.sh for T9b Run 1.

```
% Run 2: add
ARTIFICIAL_CONDUCTIVITY
% Run 3: add further
NAVIERSTOKES
NAVIERSTOKES_CONSTANT
% Run 4: replace SPH solver only
USE_MESHLESS_FINITE_MASS
```

Listing 17.2: Additional flags for Runs 2–4.

```
ArtBulkViscConst      0.05
ShearViscosityTemperature 1.0e6
FractionSpitzerViscosity 0.3
TimeMax                2.0
TimeBetSnapshot        0.01
```

Listing 17.3: Common parameter settings.

17.4 Results

Figure 17.2 shows the jet at $t = 50$ for Run 3 (SPH + Navier–Stokes viscosity). The jet has penetrated into the ambient medium forming a mushroom head. The H-shaped boundary particle structure (collimator) is clearly visible.

Expected behaviour by run:

Run 1 (pure SPH) Minimal mixing; smooth jet/ambient interface; KHI suppressed.

Run 2 (+ conductivity) Partial KHI recovery; small roll-ups at jet head.

Run 3 (+ viscosity) Physical suppression at small scales; large-scale KHI persists.

Run 4 (MFM) KHI develops at all resolved scales; most efficient mixing and breakup.

17.5 Analysis

```
1 import g3read as g3, numpy as np, glob, matplotlib.pyplot as plt
2
3 def jet_depth(snap_path):
4     f = g3.GadgetFile(snap_path)
5     pos = f.read_new('POS ', 0)
6     rho = f.read_new('RHO ', 0)
```

```

7     ids = f.read_new('ID ', 0)
8     % Free gas below the initial reservoir (y < 1.0)
9     mask = (ids != 0) & (pos[:, 1] < 1.0)
10    if mask.sum() == 0:
11        return f.header.time, np.nan
12    % Dense jet gas: top 50th percentile by density
13    thresh = np.percentile(rho[mask], 50)
14    jet = mask & (rho > thresh)
15    if jet.sum() == 0:
16        return f.header.time, np.nan
17    return f.header.time, pos[jet, 1].min()
18
19 fig, ax = plt.subplots(figsize=(7, 5))
20 for run in [1, 2, 3, 4]:
21     snaps = sorted(glob.glob(f'Run{run}/output/snap_???'))
22     results = [jet_depth(s) for s in snaps]
23     good = [(t, d) for t, d in results if not np.isnan(d)]
24     if good:
25         times, depths = zip(*good)
26         ax.plot(times, depths, label=f'Run {run}')
27
28 ax.set_xlabel('Time'); ax.set_ylabel('Jet head $$-position')
29 ax.set_title('Jet penetration depth'); ax.legend()
30 plt.tight_layout(); plt.savefig('jet_penetration.png', dpi=150)

```

Listing 17.4: Jet penetration depth and mixing rate comparison.

Exercise T9b

1. Run all four configurations. At $t = 1.5$, make side-by-side pressure maps. Which run shows the most mixing at the jet boundary?
2. Plot jet penetration depth vs time for all four runs. Do they give the same penetration speed? What controls the differences?
3. In Run 3, reduce `FractionSpitzerViscosity` from 0.3 to 0.03. Does the KHI recover? What does this imply about the critical Reynolds number for this geometry?
4. (Advanced) Measure the vortex ring radius at the jet head as a function of time. Compare with the KHI roll-up timescale $\tau \sim 1/\sigma$.

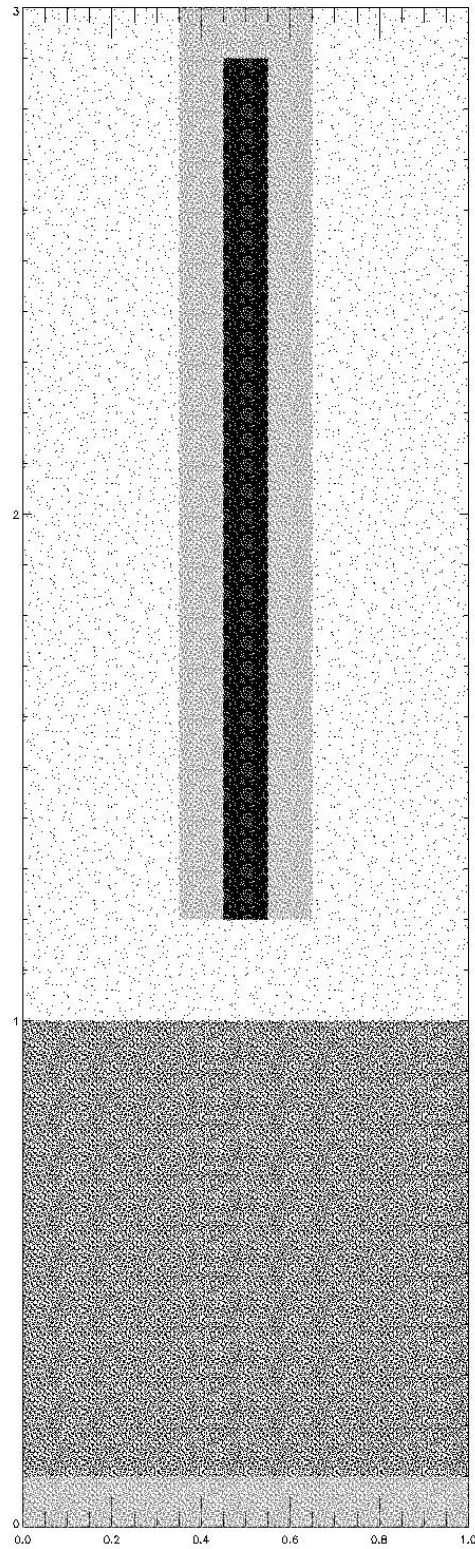


Figure 17.1: Initial conditions for Tutorial T9b (image inverted for readability). The domain is 1×3 (`LONG_Y=3`). *Upper two-thirds*: jet structure. White vertical strip (centre): dense jet gas. Medium grey surrounding it: boundary particles forming the H-shaped collimating walls visible in later snapshots. *Lower third*: lighter ambient gas at rest, into which the jet descends under the imposed downward gravity (`CONSTANT_GRAVITY_Y=-0.1`).

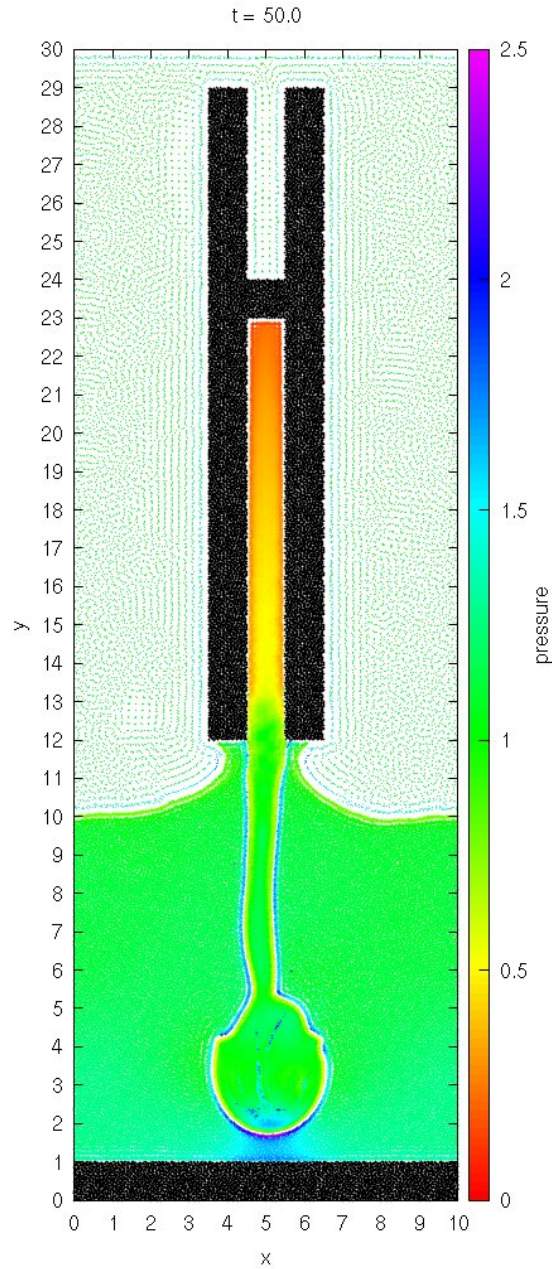


Figure 17.2: Tutorial T9b at $t = 50$ (Run 3: SPH + Navier–Stokes viscosity). Colour scale shows pressure (red = 2.5, green ≈ 1 , blue = 0). The H-shaped black structure is the boundary-particle collimator. The jet (orange–red high-pressure core) has descended to $y \approx 2$ –4 forming a mushroom head. The vortex ring at $y \approx 4$ is a KHI roll-up at the jet head; its size and coherence differ across the four runs.

Chapter 18

Tutorial T10: MHD Shock Tube

T10 extends the Riemann problem of T4 to full magnetohydrodynamics (MHD). The classical reference test is the shock tube 5A of [Brio and Wu \(1988\)](#), which exercises all five MHD wave modes in a single configuration. This tutorial requires the **MAGNETIC** compile-time flag and introduces the MHD-specific parameters for divergence cleaning and artificial dissipation.

18.1 Physical Background

18.1.1 MHD wave modes

In a magnetised plasma, three families of wave propagate:

Alfvén waves Transverse, incompressible; speed $v_A = B/\sqrt{4\pi\rho}$. Propagate along **B**; do not compress the plasma.

Fast magnetosonic waves Compressional; speed $v_f = \sqrt{c_s^2 + v_A^2}$ perpendicular to **B**. The MHD analogue of acoustic waves.

Slow magnetosonic waves Compressional, slower than fast; exist only when **B** is oblique to the propagation direction.

The MHD Riemann problem produces seven distinct regions (vs. five for HD): two fast shocks/rarefactions, two slow shocks/rarefactions, two rotational (Alfvénic) discontinuities, and a central contact discontinuity.

18.1.2 Test 5A of Brio & Wu (1988)

[Brio and Wu \(1988\)](#) provide five MHD shock tube tests. Test 5A (Table 18.1) is the most commonly used for SPH-MHD validation. It uses $\gamma = 2$ and an oblique field ($B_x \neq 0$, $B_y \neq 0$), exciting all wave modes simultaneously.

Table 18.1: Initial states for MHD shock tube test 5A ([Brio and Wu, 1988](#)). Code units, $\gamma = 2$.

Side	ρ	P	v_x	v_y	B_x	B_y
Left	1.000	1.000	0	0	0.75	+1.0
Right	0.125	0.100	0	0	0.75	−1.0

18.1.3 The $\nabla \cdot \mathbf{B} = 0$ constraint

Maintaining $\nabla \cdot \mathbf{B} = 0$ numerically is a fundamental challenge in MHD codes. Non-zero divergence generates unphysical forces. OpenGADGET3 implements Powell–Tricco hyperbolic divergence cleaning (DIVBCLEANING_TRICCO), which actively drives $\nabla \cdot \mathbf{B} \rightarrow 0$.

18.2 Config.sh

```
PERIODIC
NOGRAVITY
WENDLAND_C4_KERNEL
LONG_X=100
LONG_Y=1
LONG_Z=1
MAGNETIC
MUO_UNITY
NO_BALSARA_LIMITER
```

Listing 18.1: Config.sh for Tutorial T10 (baseline).

For the extended experiment with artificial dissipation:

```
ARTIFICIAL_CONDUCTIVITY
ARTIFICIAL_MAGNETIC DISSIPATION
TIME_DEP_ART_VISC=2
TIME_DEP_ART_COND=2
TIME_DEP_MAGN_DISP=2
DIVBCLEANING_TRICCO
```

18.3 Parameter File

```
PeriodicBoundariesOn      1
TimeMax                   6.0
TimeBetSnapshot           0.2
UnitLength_in_cm          1.0
UnitMass_in_g             1.0
UnitVelocity_in_cm_per_s  1.0
SofteningGas              0.02
SofteningGasMaxPhys       0.02
ArtificialMagneticDissipationConstant  0.2
```

Listing 18.2: Parameter file for Tutorial T10.

18.4 Initial Conditions

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T10/box.ic
mkdir output && sbatch runme.sh % ntasks=8, cpus-per-task=4
```

Figure 18.1 shows the initial state of the slab. The profiles of specific internal energy U (black), B_x (blue), and B_y (red) are uniform in each half, with the contact discontinuity at $x = 50$ (halfway along the $\text{LONG_X}=100$ domain). The step in B_y from $+1$ to -1 and the steps in U and B_x define the left and right states of Test 5A.

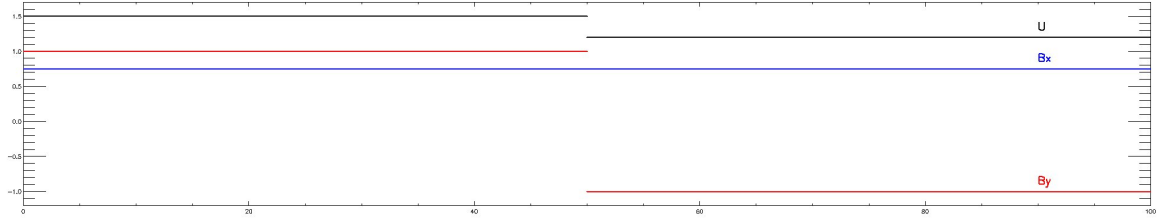


Figure 18.1: Initial conditions for Tutorial T10 (MHD shock tube 5A). Profiles of specific internal energy U (black, $U_L = 1.5$, $U_R = 0.8$), B_x (blue, constant = 0.75), and B_y (red, +1 left, -1 right) vs x at $t = 0$. The contact discontinuity is at $x = 50$. All velocities are zero. This IC triggers fast and slow MHD shocks, Alfvén waves, and a contact discontinuity upon release.

18.5 Results

Figures 18.2 and 18.3 show the simulation at two times, displaying all nine field components.

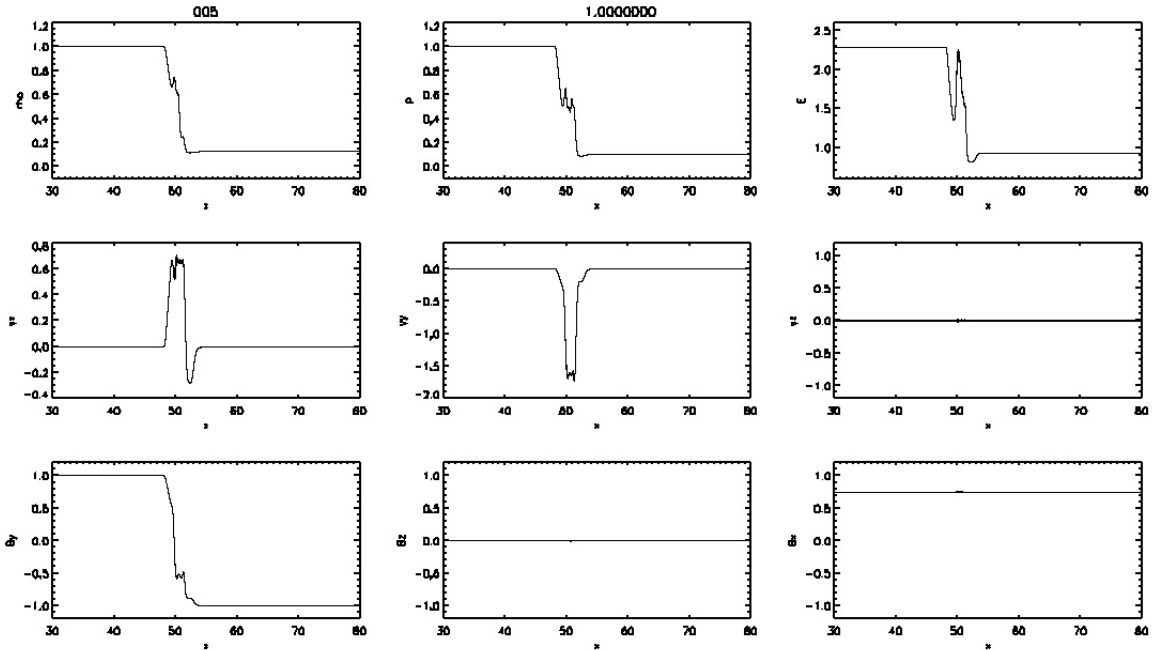


Figure 18.2: MHD shock tube 5A at an early time ($t \approx 1$). Nine panels show (left column) density ρ , v_x , B_y ; (middle column) pressure P , v_y , B_z ; (right column) specific internal energy ε , v_z , divergence of $\mathbf{B}$. The wave structure is already developing: the fast shock has propagated to the right, and the slow magnetosonic waves and rotational discontinuities are visible near the initial contact position.

18.6 Analysis

```
1 import g3read as g3, numpy as np, matplotlib.pyplot as plt
2
3 f = g3.GadgetFile('output/snap_030') # t=6 with TimeBetSnapshot
   =0.2
4 pos = f.read_new('POS ', 0)
5 rho = f.read_new('RHO ', 0)
6 u = f.read_new('U ', 0)
7 vel = f.read_new('VEL ', 0)
```

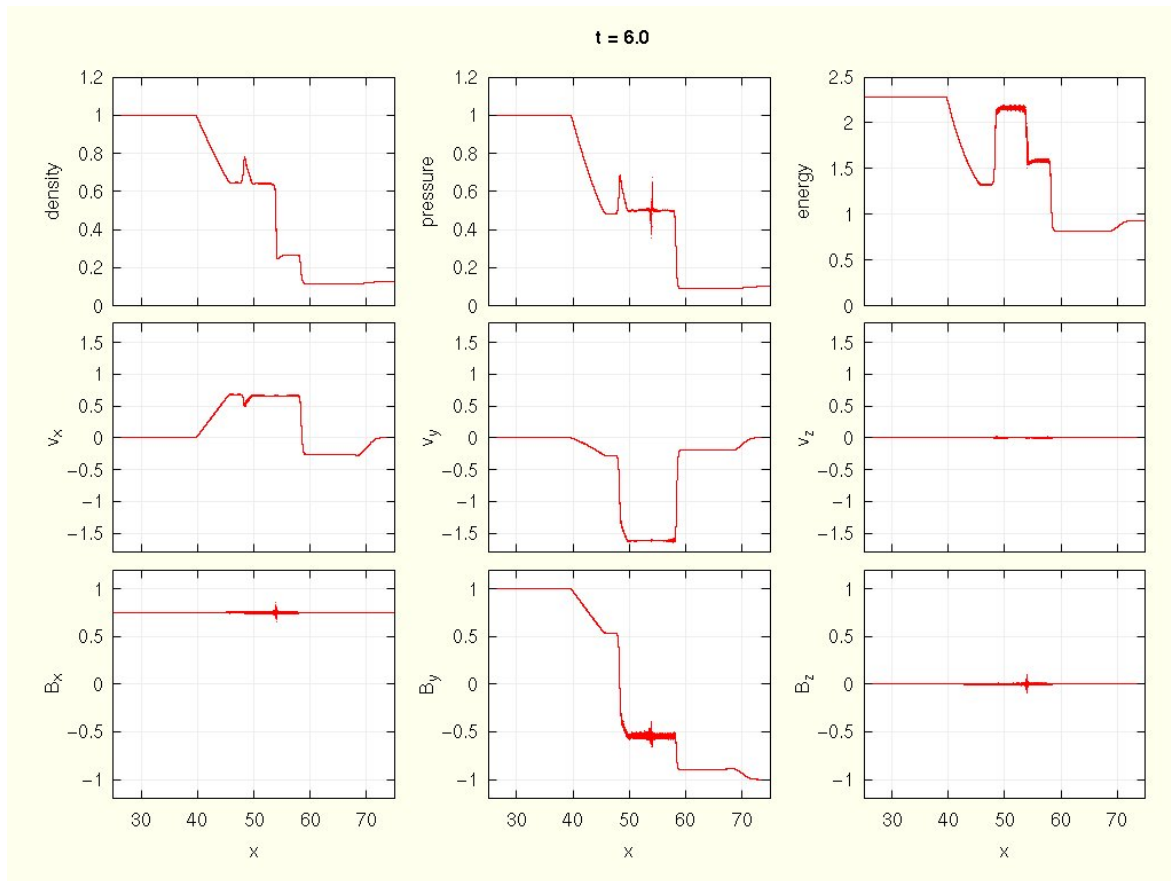


Figure 18.3: MHD shock tube 5A at $t = 6$ (fully developed solution). Nine panels: (top row) density, pressure, specific internal energy; (middle row) v_x , v_y , v_z ; (bottom row) B_x , B_y , B_z . The seven-region structure of the MHD Riemann solution is visible: fast rarefaction (left slope in density), slow shock, contact discontinuity, slow shock, fast shock (right side). The reversal of B_y from $+1$ to -1 is clearly resolved. Compare with the tabulated reference solution in [Brio and Wu \(1988\)](#).

```

8  B = f.read_new('BFLD', 0)
9
10 x = pos[:, 0]
11 P = (2.0 - 1.0) * rho * u # gamma = 2 for Test 5A
12 idx = np.argsort(x)
13
14 fig, axes = plt.subplots(3, 3, figsize=(12, 10), sharex=True)
15 fields = [rho, P, u,
16           vel[:,0], vel[:,1], vel[:,2],
17           B[:,0], B[:,1], B[:,2]]
18 labels = [r'$\rho$', r'$P$', r'$\varepsilon$',
19           r'$v_x$', r'$v_y$', r'$v_z$',
20           r'$B_x$', r'$B_y$', r'$B_z$']
21
22 for ax, fld, lbl in zip(axes.ravel(), fields, labels):
23     ax.plot(x[idx]*100, fld[idx], '.', ms=1, alpha=0.6)
24     ax.set_ylabel(lbl)
25     ax.grid(True, alpha=0.3)
26
27 for ax in axes[-1]:
28     ax.set_xlabel('x')

```

```

29 fig.suptitle('MHD Shock Tube 5A at t = 6')
30 plt.tight_layout()
31 plt.savefig('mhd_t6.png', dpi=150)

```

Listing 18.3: Plotting MHD profiles at t=6.

```

1 try:
2     divB = f.read_new('DIVB', 0)
3     B_mag = np.sqrt((B**2).sum(axis=1))
4     print(f'|divB|/|B| mean: {(np.abs(divB)/B_mag).mean():.3e}')
5     print(f'|divB|/|B| max: {(np.abs(divB)/B_mag).max():.3e}')
6 except:
7     print('DIVB block not present (compile with DIVBCLEANING_TRICCO)')

```

Listing 18.4: Checking divergence cleaning quality.

Exercise T10

1. From Figure 18.3, identify the seven regions of the MHD Riemann solution in the ρ and B_y profiles. Which features correspond to the fast shock, slow shock, and contact discontinuity?
2. Enable DIVBCLEANING_TRICCO and monitor $|\nabla \cdot \mathbf{B}|/|\mathbf{B}|$ over time. Does it decrease? What is the final value at $t = 6$?
3. Enable ARTIFICIAL_MAGNETIC DISSIPATION and compare the B_y profile at $t = 6$ with the baseline run. Are the wave fronts sharper or smoother?
4. (Optional) Run MHD test cases 1–4 from [Brio and Wu \(1988\)](#). Each tests different subsets of the wave modes.

Chapter 19

Tutorial T11: Disk Galaxy with Star Formation

T11 is the most physically complete tutorial: a simulation of an isolated disk galaxy with radiative cooling, star formation, and optional galactic winds. The primary goal is to reproduce the observed Kennicutt–Schmidt (SK) relation from first principles, using the sub-grid model of [Springel and Hernquist \(2003\)](#).

19.1 Physical Background

19.1.1 The Kennicutt–Schmidt relation

[Kennicutt \(1998\)](#) established an empirical power-law between gas surface density and SFR surface density, spanning six orders of magnitude:

$$\Sigma_{\text{SFR}} = A \Sigma_{\text{gas}}^n, \quad n \approx 1.4. \quad (19.1)$$

In OpenGADGET3, this emerges from the sub-grid model: gas above a density threshold forms stars at rate $\dot{\rho}_* = \rho/t_*(\rho)$, where t_* follows a power law calibrated to reproduce the observed slope.

19.1.2 Radiative cooling and the two-phase ISM

At $T > 10^4$ K gas cools by line emission and bremsstrahlung; below 10^4 K the UV background (TREECOOL) suppresses cooling. In equilibrium, gas settles into a two-phase ISM: cold dense clouds ($T \sim 10^4$ K) embedded in hot diffuse gas ($T \sim 10^6$ K). Star formation occurs in the cold phase.

19.1.3 Galactic winds

Supernovae inject energy into the ISM. The kinetic wind model (WINDS) launches gas particles from star-forming regions at velocity $v_w \propto \sigma$ with mass loading $\eta = \dot{M}_w/\dot{M}_* = \text{WindEfficiency}$.

19.2 Config.sh

```
#PERIODIC
#NOGRAVITY
WENDLAND_C4_KERNEL
```

```
COOLING
SFR
GENERATIONS=4
```

Listing 19.1: Config.sh for Tutorial T11 (no winds).

For the wind experiment: add WINDS and recompile.

19.3 Initial Conditions

Table 19.1: Available galaxy IC files for Tutorial T11.

File	Label	Particles
7320c.dat	Small	250,180
7318b.dat	Small+	420,822
7318a.dat	Medium	794,886
7319.dat	Large	2,337,900

```
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/T11/7320c.dat
```

These files are in Format 1 (ICFormat 1). They contain gas (type 0), dark matter (type 1), disk (type 2), and bulge (type 3) particles.

19.4 Parameter File

```
InitCondFile      7320c.dat
ICFormat          1
OutputDir         ./output
TimeMax           1.0           % ~ 1 Gyr
TimeBetSnapshot   0.01
PeriodicBoundariesOn 0
PartAllocFactor   5.5           % must be >= 5 for SFR runs
UnitLength_in_cm  3.085678e21
UnitMass_in_g     1.989e43
UnitVelocity_in_cm_per_s 1e5
SofteningGas      0.02
SofteningHalo     0.02
SofteningDisk     0.02
SofteningBulge    0.02
SofteningStars    0.02
SofteningGasMaxPhys 0.02
SofteningHaloMaxPhys 0.02
SofteningDiskMaxPhys 0.02
SofteningBulgeMaxPhys 0.02
SofteningStarsMaxPhys 0.02
CoolingOn         1
StarformationOn   1
MaxSfrTimescale   1.5
WindEfficiency     2.0
WindEnergyFraction 0.5
WindFreeTravelMaxTimeFactor 0.025
WindFreeTravelDensFac 0.5
```

Listing 19.2: Parameter file for Tutorial T11.

Important

Copy TREECOOL before submitting: `cp Hydro-OpenGadget3/TREECOOL .`
 Set `PartAllocFactor` ≥ 5.5 : the particle count grows as gas forms stars.

19.5 Visualising the Galaxy

Figure 19.1 shows the galaxy at $t = 2.16$ Gyr, clearly displaying the characteristic spiral arm morphology of a late-type disk galaxy. Gas particles (blue/violet) trace the spiral structure and the outer disk; stellar particles (yellow) are concentrated along the star-forming arms.



Figure 19.1: Tutorial T11 at $t = 2.16$ Gyr. Face-on particle map of the simulated disk galaxy. Blue/violet: gas particles (dark matter not shown). Yellow: stellar particles formed during the simulation. The spiral arm structure is clearly visible. The concentrated yellow regions along the arms mark the densest star-forming zones, where the gas density exceeds the threshold for the sub-grid SFR prescription.

Figure 19.2 shows face-on and edge-on projections at snapshot 007, with gas (orange) and stellar particles (blue) separately coloured.

```
1 import g3read as g3, numpy as np, matplotlib.pyplot as plt
2 from matplotlib.colors import LogNorm
3
4 f = g3.GadgetFile('output/snap_050')
5 pos_g = f.read_new('POS ', 0)
6 pos_s = f.read_new('POS ', 4)
7
8 fig, axes = plt.subplots(1, 2, figsize=(14, 6))
9 for ax, (ix, iy), title in zip(axes,
```

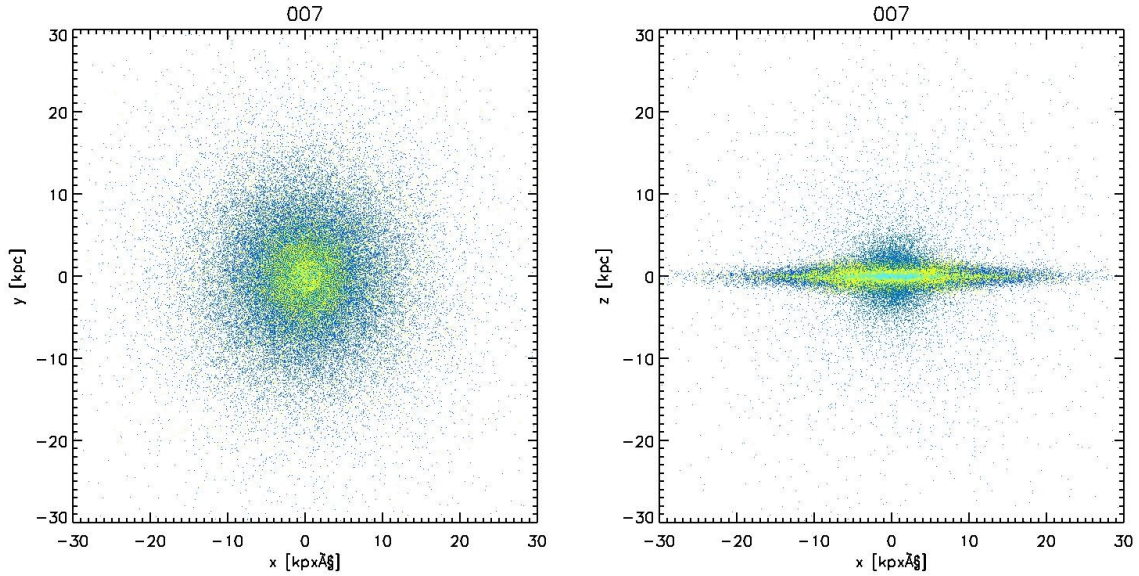


Figure 19.2: Tutorial T11, snapshot 007. *Left*: face-on view (x - y plane). Orange: gas particles. Blue: stellar particles concentrated in the central bulge region. The disk extends to ~ 20 kpc/h in radius. *Right*: edge-on view (x - z plane), showing the thin disk geometry with scale height ~ 1 – 2 kpc and the more extended gas envelope.

```

10         [(0,1,'Face-on'), (0,2,'Edge-on')],
11         ['x [kpc/h]', 'x [kpc/h]']):
12     h, xe, ye = np.histogram2d(pos_g[:,ix], pos_g[:,iy], bins=256)
13     ax.imshow(h.T, origin='lower', norm=LogNorm(vmin=1), cmap='inferno',
14             extent=[xe[0], xe[-1], ye[0], ye[-1]])
15     if len(pos_s):
16         ax.scatter(pos_s[:,ix], pos_s[:,iy], s=0.3, c='cyan', alpha=0.3)
17     ax.set_title(title)
18 plt.tight_layout(); plt.savefig('galaxy.png', dpi=150)

```

Listing 19.3: Face-on and edge-on visualisation.

19.6 The Kennicutt–Schmidt Relation

Figure 19.3 shows the SK relation measured from the simulation. The simulation reproduces the observed slope $n \approx 1.4$ across two decades in gas surface density, confirming that the sub-grid SFR model is correctly calibrated.

```

1 import g3read as g3, numpy as np, matplotlib.pyplot as plt
2
3 f = g3.GadgetFile('output/snap_050')
4 pos = f.read_new('POS ', 0)
5 mass = f.read_new('MASS', 0)
6 sfr = f.read_new('SFR ', 0)
7
8 R_kpc = np.sqrt(pos[:,0]**2 + pos[:,1]**2) / 0.7 # physical kpc
9 Rbins = np.linspace(0, 20, 25)
10 R_mid = 0.5*(Rbins[:-1]+Rbins[1:])
11 Sigma_gas = np.zeros(len(R_mid))
12 Sigma_sfr = np.zeros(len(R_mid))

```

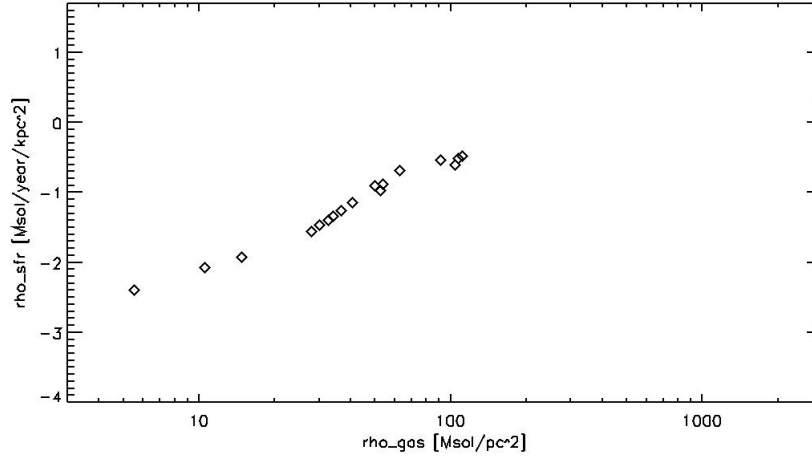


Figure 19.3: Kennicutt–Schmidt relation measured from Tutorial T11 (image inverted for readability; original axes are $\log_{10}(\Sigma_{\text{gas}} [M_{\odot} \text{pc}^{-2}])$ vs $\log_{10}(\dot{\Sigma}_{*} [M_{\odot} \text{yr}^{-1} \text{kpc}^{-2}])$). Each diamond is one radial annulus of the disk. The simulated points lie on a power law with slope ≈ 1.4 over $\Sigma_{\text{gas}} \sim 10\text{--}200 M_{\odot} \text{pc}^{-2}$, consistent with the observational Kennicutt–Schmidt relation (Kennicutt, 1998). This is the primary validation diagnostic for Tutorial T11.

```

13
14 for i in range(len(R_mid)):
15     mask = (R_kpc >= Rbins[i]) & (R_kpc < Rbins[i+1])
16     if mask.sum() == 0: continue
17     area = np.pi*(Rbins[i+1]**2 - Rbins[i]**2)
18     Sigma_gas[i] = mass[mask].sum() * 1e10 / area # Msun/kpc^2
19     Sigma_sfr[i] = sfr[mask].sum() / area # Msun/yr/kpc^2
20
21 good = Sigma_sfr > 0
22 fig, ax = plt.subplots(figsize=(6,5))
23 ax.scatter(np.log10(Sigma_gas[good]), np.log10(Sigma_sfr[good]),
24           c=R_mid[good], cmap='viridis', s=40)
25 Sg = np.linspace(4, 9, 50)
26 ax.plot(Sg, 1.4*Sg - 12.1, 'r--', lw=2, label='SK $n=1.4$')
27 ax.set_xlabel(r'$\log_{10}(\Sigma_{\mathrm{gas}} [M_{\odot}, \mathrm{kpc}^{-2}])$')
28 ax.set_ylabel(r'$\log_{10}(\dot{\Sigma}_{*} [M_{\odot}, \mathrm{yr}^{-1}, \mathrm{kpc}^{-2}])$')
29 ax.legend(); plt.tight_layout(); plt.savefig('sk_relation.png', dpi
=150)

```

Listing 19.4: Computing and plotting the SK relation.

19.7 Star Formation History

```

1 import g3read as g3, numpy as np, glob, matplotlib.pyplot as plt
2
3 snaps = sorted(glob.glob('output/snap_???'))
4 times, SFR_tot, M_star = [], [], []
5 for sp in snaps:
6     f = g3.GadgetFile(sp)
7     times.append(f.header.time * 0.978) # code time -> Gyr
8     SFR_tot.append(f.read_new('SFR ',0).sum())
9     N_star = f.header.npart[4]

```

```

10     M_star.append(f.read_new('MASS',4).sum()*1e10 if N_star>0 else
11         0.0)
12
13 times = np.array(times)
14 fig, (ax1,ax2) = plt.subplots(2,1,figsize=(8,6),sharex=True)
15 ax1.semilogy(times, SFR_tot, '-o', ms=3)
16 ax1.set_ylabel(r'SFR $[M_\odot\,\mathrm{yr}^{-1}]$')
17 ax2.plot(times, np.array(M_star)/1e9, '-o', ms=3)
18 ax2.set_ylabel(r'$M_*\ [10^9\,M_\odot]$'); ax2.set_xlabel('Time [Gyr]')
19 plt.tight_layout(); plt.savefig('sfh.png', dpi=150)

```

Listing 19.5: Global star formation history.

Exercise T11

1. From Figure 19.3, measure the slope n by fitting a power law to the data points. What is the uncertainty? Is it consistent with $n = 1.4$?
2. Enable WINDS and rerun. Compare the total stellar mass formed and the SK slope. How does the wind change the edge-on morphology in Figure 19.2?
3. Vary MaxSfrTimescale by factors of 0.5 and 2. Does the slope n change, or only the normalisation?
4. (Challenge) Run the medium IC (7318a.dat, 800k particles). Does the SK relation converge with resolution?

Part IV

Post-processing and Analysis

Chapter 20

Post-processing, Visualisation, and Derived Quantities

This chapter collects the most useful post-processing techniques for OpenGADGET3 output: reading and plotting snapshots, computing derived physical quantities, producing movies, and using the yt framework for more complex analysis.

20.1 The g3read.py Workflow

The workhorse for all tutorials is `g3read.py`. A complete analysis workflow for a sequence of snapshots:

```
1 import g3read as g3
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.animation as animation
5 import glob
6
7 snaps = sorted(glob.glob('output/snap_???'))
8 fig, ax = plt.subplots(figsize=(7,7))
9
10 def animate(i):
11     ax.cla()
12     f = g3.GadgetFile(snaps[i])
13     pos = f.read_new('POS ', 0)
14     rho = f.read_new('RHO ', 0)
15     # 2D histogram weighted by density
16     h, xe, ye = np.histogram2d(pos[:,0], pos[:,1], bins=128,
17                               weights=np.log10(rho+1e-10))
18     ax.imshow(h.T, origin='lower', cmap='inferno',
19              extent=[xe[0],xe[-1],ye[0],ye[-1]])
20     ax.set_title(f't = {f.header.time:.2f}')
21
22 ani = animation.FuncAnimation(fig, animate, frames=len(snaps),
23                               interval=100)
24 ani.save('simulation.mp4', writer='ffmpeg', fps=10, dpi=150)
25 print('Saved simulation.mp4')
```

Listing 20.1: Complete post-processing workflow.

20.2 Radial Profiles

Radial profiles (density, temperature, pressure) are a fundamental diagnostic for spherically symmetric systems (galaxy clusters, stellar collapse):

```

1 import g3read as g3, numpy as np, matplotlib.pyplot as plt
2
3 f = g3.GadgetFile('output/snap_050')
4 pos = f.read_new('POS ', 0) - 0.5 # centre at box centre
5 rho = f.read_new('RHO ', 0)
6 u = f.read_new('U ', 0)
7
8 r = np.sqrt((pos**2).sum(axis=1))
9
10 Nbin = 60
11 rbins = np.logspace(np.log10(r[r>0].min()), np.log10(r.max()), Nbin)
12 r_mid = np.sqrt(rbins[:-1]*rbins[1:]) # logarithmic midpoints
13
14 rho_prof = np.zeros(Nbin-1)
15 T_prof = np.zeros(Nbin-1)
16
17 # Physical constants for temperature conversion
18 k_B, m_p, mu, gamma = 1.3806e-16, 1.6726e-24, 0.588, 5.0/3.0
19 vel_unit = 1e5 # km/s to cm/s
20
21 for i in range(Nbin-1):
22     mask = (r >= rbins[i]) & (r < rbins[i+1])
23     if mask.sum() > 0:
24         rho_prof[i] = rho[mask].mean()
25         u_cgs = u[mask].mean() * vel_unit**2
26         T_prof[i] = (gamma-1)*mu*m_p*u_cgs/k_B
27
28 fig, (ax1, ax2) = plt.subplots(2,1,figsize=(7,8),sharex=True)
29 ax1.loglog(r_mid, rho_prof, '-o', ms=4)
30 ax1.set_ylabel(r'$\rho$ [code units]')
31 ax2.loglog(r_mid, T_prof, '-o', ms=4)
32 ax2.set_ylabel('T [K]'); ax2.set_xlabel('r [code length]')
33 plt.tight_layout(); plt.savefig('radial_profiles.png', dpi=150)

```

Listing 20.2: Computing and plotting radial profiles.

20.3 Loading Data with yt

The yt framework provides a high-level interface to GADGET data:

```

1 import yt
2
3 ds = yt.load('output/snap_050')
4
5 # Projection plot (column density)
6 p = yt.ProjectionPlot(ds, 'z', ('gas', 'density'))
7 p.set_cmap('density', 'inferno')
8 p.save('yt_projection.png')
9
10 # Radial profile
11 sp = ds.sphere('c', (500, 'kpc'))
12 profile = yt.create_profile(sp, 'radius', ['density', 'temperature'],
13                             n_bins=50, weight_field='cell_mass')

```

```
14 profile.save_as_dataset('radial_profile.h5')
```

Listing 20.3: Loading and visualising a snapshot with yt.

20.4 Cosmological Unit Conversions

For cosmological runs, positions and velocities are stored in co-moving coordinates. Converting to physical quantities:

```
1 a = f.header.time          # scale factor a = 1/(1+z)
2 z = 1.0/a - 1              # redshift
3 h = f.header.HubbleParam    # dimensionless Hubble parameter
4
5 # Physical positions [kpc] = comoving positions [kpc/h] * a / h
6 pos_phys = pos * a / h      # (assuming stored in kpc/h comoving)
7
8 # Physical velocities [km/s]
9 # GADGET stores v_stored = v_peculiar / sqrt(a)
10 vel_phys = vel * np.sqrt(a) # peculiar velocity in km/s
11
12 print(f'z = {z:.3f}, a = {a:.4f}')
```

Listing 20.4: Co-moving to physical coordinate conversion.

20.5 Power Spectrum and Two-Point Statistics

For cosmological boxes, the matter power spectrum quantifies the amplitude of density fluctuations on each scale:

```
1 import numpy as np
2 import Pk_library as PKL
3
4 # Load positions as (N,3) float32 array in Mpc/h
5 BoxSize = 150.0 # Mpc/h
6 grid = 512 # PM mesh size
7
8 # Compute power spectrum using CIC mass assignment
9 Pk = PKL.Pk(pos.astype(np.float32), BoxSize, grid, MAS='CIC', verbose=
    True)
10 k = Pk.k3D # wavenumbers [h/Mpc]
11 Pk_matter = Pk.Pk[:,0] # power spectrum [Mpc/h]^3
12
13 import matplotlib.pyplot as plt
14 fig, ax = plt.subplots()
15 ax.loglog(k, Pk_matter)
16 ax.set_xlabel(r'$k \ [h\, \mathrm{Mpc}^{-1}]$')
17 ax.set_ylabel(r'$P(k) \ [(h^{-1}\, \mathrm{Mpc})^3]$')
18 plt.tight_layout(); plt.savefig('power_spectrum.png', dpi=150)
```

Listing 20.5: Computing the matter power spectrum with Pylans3.

Appendix A

Complete Annotated Parameter File

The following parameter file includes all commonly used parameters with brief annotations. Parameters marked **[required]** must be present; all others have defaults visible in `parameters-usedvalues` after the first run.

```
%--- File paths -----
InitCondFile      box.ic      % [required] IC file, no extension
OutputDir         ./output    % [required] must exist before run starts
SnapshotFileBase  snap        % base name for snapshot files
EnergyFile        energy.txt   % global energy log
CpuFile           cpu.txt      % CPU timing log
InfoFile          info.txt     % general info log
TimingsFile       timings.txt  % per-step timing
CpuTimeBetRestartFile 12.0    % wall hours between restart file writes

%--- File formats -----
ICFormat          2           % 1=legacy, 2=block binary, 3=HDF5
SnapFormat        2           % same options; HDF5 recommended

%--- Output times -----
TimeBegin         0.0         % start time (or scale factor for cosmo)
TimeMax           1.0         % end time
TimeBetSnapshot   0.1         % interval between snapshots [code time]
TimeOfFirstSnapshot 0.0       % time of first snapshot
TimeBetStatistics 0.05        % interval for energy.txt entries
OutputListOn      0           % 1: use OutputListFilename instead
OutputListFilename outputs.txt % list of output times (one per line)
NumFilesPerSnapshot 1         % >1 splits snapshot across files
NumFilesWrittenInParallel 1

%--- CPU budget -----
TimeLimitCPU      86400       % wall-clock limit in seconds
ResubmitOn        0           % 1: resubmit job automatically
ResubmitCommand   my_resubmit_script

%--- Memory -----
PartAllocFactor   1.5         % memory allocation per particle
                               % use 5.5 for simulations with SFR
BufferSize        150         % MB for communication buffer
TreeAllocFactor   0.8

%--- Unit system -----
UnitLength_in_cm  3.085678e21 % 1.0 kpc/h
```

```

UnitMass_in_g          1.989e43      % 1e10 Msun
UnitVelocity_in_cm_per_s 1e5          % 1 km/s
% Derived: time unit = 0.978 Gyr, G in code units computed
      automatically

GravityConstantInternal 0              % 0: use physical G; set to 1.0 for
      % dimensionless test problems

%--- Cosmological parameters (only for ComovingIntegrationOn = 1) ----
ComovingIntegrationOn  0
Omega0                  0.3
OmegaLambda             0.7
OmegaBaryon             0.046
HubbleParam            0.7
BoxSize                 1.0          % [required] box side length [
      code length]

%--- Boundaries and gravity -----
PeriodicBoundariesOn    1              % 1=periodic, 0=isolated
ErrTolForceAcc          0.005          % tree opening criterion (
      smaller=more accurate)
ErrTolIntAccuracy       0.025          % timestep accuracy eta
MaxSizeTimestep         0.01          % maximum allowed timestep [
      code time]
MinSizeTimestep         0.0            % minimum (0 = no limit)
MaxRMSDisplacementFac   0.25          % PM-related timestep criterion
      (cosmo only)
CourantFac              0.15          % CFL safety factor

TreeDomainUpdateFrequency 0.05        % how often to rebuild tree
      domain

%--- Gravitational softening (one entry per particle type 0-5) -----
SofteningGas            0.2
SofteningHalo           0.2
SofteningDisk           0.2
SofteningBulge          0.2
SofteningStars          0.2
SofteningBndry          0.2
SofteningGasMaxPhys     0.2
SofteningHaloMaxPhys    0.2
SofteningDiskMaxPhys    0.2
SofteningBulgeMaxPhys   0.2
SofteningStarsMaxPhys   0.2
SofteningBndryMaxPhys   0.2

%--- SPH parameters -----
DesNumNgb               64            % target neighbour number; kernel
      -dependent
      % check parameters-used values for
      kernel default
MaxNumNgbDeviation      1              % tolerance on Nngb
ArtBulkViscConst        0.8           % artificial viscosity alpha
      % (0.05 for smooth flows, 0.8-1.5
      for shocks)

%--- Star formation and cooling -----
CoolingOn               0             % 1: enable radiative cooling (needs TREECOOL)

```

```

StarformationOn      0      % 1: enable SFR (needs COOLING and SFR flag)
MaxSfrTimescale      1.5    % star formation timescale t_0* [code time]
FactorSN              0.1    % SN energy fraction
FactorEVP             1000.0  % evaporation parameter
TempSupernova         3.0e8   % SN ejecta temperature [K]
TempClouds            1000.0  % cold cloud temperature [K]
CritOverDensThresh    57.7    % density threshold for SFR (in units of
      rho_crit)
CritPhysDensity        0.0    % min physical density for SFR [H/cm^3];
      0=auto

%--- Galactic winds (requires WINDS flag) -----
WindEfficiency         2.0
WindEnergyFraction      0.5
WindFreeTravelMaxTimeFactor 0.025
WindFreeTravelDensFac   0.5

%--- Thermal conduction (requires CONDUCTION flag) -----
ConductionEfficiency    0.1    % fraction of Spitzer conductivity
MaxSizeConductionStep   0.005  % max conduction timestep [code time]

%--- Physical viscosity (requires NAVIERSTOKES flag) -----
ShearViscosityTemperature 5.0e7 % T_visc in code temperature units [K]
FractionSpitzerViscosity 1.0    % fraction of Spitzer viscosity

%--- MHD (requires MAGNETIC flag) -----
ArtificialMagneticDissipationConstant 0.2

%--- Stop condition -----
StopAfterNSteps        0      % 0 = run to TimeMax; N>0 = stop after N steps

```

Listing A.1: Complete annotated OpenGADGET3 parameter file.

Appendix B

Complete Config.sh Flag Reference

This appendix lists all `Config.sh` flags available in the tutorial version of OpenGADGET3 , grouped by category. Only one kernel flag may be set at a time. Flags marked [**incompatible**] cannot be used together.

Table B.1: Complete `Config.sh` flag reference.

Flag	Description
<i>Geometry</i>	
PERIODIC	Periodic boundary conditions
LONG_X=N	Stretch box by factor N in x
LONG_Y=N	Stretch box by factor N in y
LONG_Z=N	Stretch box by factor N in z
TWODIMS	2D (particles in $z = 0$ plane)
<i>Gravity</i>	
NOGRAVITY	Disable all self-gravity
EVALPOTENTIAL	Compute and output Φ
OUTPUTACCELERATION	Write acceleration vector to snapshot
<i>SPH kernels (choose one)</i>	
STANDARD_KERNEL	Cubic spline M4, C^2
QUINTIC_KERNEL	Quintic spline M6, C^4
WENDLAND_C2_KERNEL	Wendland C^2
WENDLAND_C4_KERNEL	Wendland C^4 (recommended)
WENDLAND_C6_KERNEL	Wendland C^6
<i>SPH solver</i>	
USE_MESHLESS_FINITE_MASS	Replace SPH with MFM [incompatible with MAGNETIC]
<i>Artificial dissipation</i>	
ARTIFICIAL_CONDUCTIVITY	Thermal conductivity at contacts
TIME_DEP_ART_VISC=2	Time-dependent α_{visc}
TIME_DEP_ART_COND=2	Time-dependent α_{cond}
ARTIFICIAL_MAGNETIC DISSIPATION	Artificial resistivity
TIME_DEP_MAGN_DISP=2	Time-dependent resistivity
<i>Boundary particles</i>	
SPH_BND_PARTICLES	Enable ID=0 boundary particles
SPH_BND_PARTICLES_ALLOW_DRIFT	Allow boundary particles to move

Flag	Description
SPH_BND_PARTICLES_ALLOW_CONDUCTION	Enable conduction at boundaries
<i>MHD</i>	
MAGNETIC	MHD equations
MU0_UNITY	$\mu_0 = 1$ (Gaussian-like units)
NO_BALSARA_LIMITER	Disable Balsara viscosity limiter
DIVBCLEANING_TRICCO	Powell–Tricco divergence cleaning
<i>Transport physics</i>	
CONDUCTION	Thermal conduction
CONDUCTION_CONSTANT	Constant κ (vs. Spitzer $T^{5/2}$)
NAVIERSTOKES	Physical viscosity
NAVIERSTOKES_CONSTANT	Constant η
<i>Sub-grid physics</i>	
COOLING	Radiative cooling (needs TREECOOL)
SFR	Star formation (needs COOLING)
WINDS	Kinetic stellar winds
GENERATIONS=N	Max stellar generations per gas particle
<i>Miscellaneous</i>	
CONSTANT_GRAVITY_Y=V	Constant gravitational acceleration in y
OUTPUTTIMESTEP	Write individual particle timesteps to snapshot
OUTPUTPOTENTIAL	Write gravitational potential (same as EVALPOTENTIAL)

Appendix C

GADGET Binary File Format

C.1 Format 2 Block Structure

Each block in a Format 2 snapshot file has the following structure:

```
/* Label record */
int32  size_label = 8;
char   label[4];           /* e.g. "POS ", "VEL ", "ID  " */
int32  size_data + 8;      /* size of next record + 8 */
int32  size_label = 8;

/* Data record */
int32  size_data;           /* size in bytes of the data */
char   data[size_data];    /* the actual data */
int32  size_data;
```

Listing C.1: Format 2 block structure in C.

C.2 Reading Format 2 in Python

```
1 import struct, numpy as np
2
3 def read_block(f, dtype, n_elements):
4     """Read one labelled block from a Format 2 GADGET file."""
5     # Skip label record
6     f.read(4 + 4 + 4 + 4)
7     # Data record
8     nbytes = struct.unpack('i', f.read(4))[0]
9     data = np.frombuffer(f.read(nbytes), dtype=dtype)
10    f.read(4) # trailing size
11    return data.reshape(n_elements, -1) if data.size > n_elements else
        data
12
13 with open('snap_010', 'rb') as f:
14     # HEAD
15     f.read(4+4+4+4 + 4)
16     header = f.read(256)
17     f.read(4)
18     npart = struct.unpack_from('6i', header, 0)
19     mass_h = struct.unpack_from('6d', header, 24)
20     time = struct.unpack_from('d', header, 72)[0]
```

```
21     boxsize = struct.unpack_from('d', header, 144)[0]
22
23     N_gas = npart[0]
24     pos    = read_block(f, np.float32, N_gas)    # shape (N_gas, 3)
```

Listing C.2: Low-level Format 2 reader in Python.

Appendix D

Quick Reference Card

D.1 Essential Commands

```
# Environment (run once per session)
module load compiler openmpi
export SYSTYPE=Cast

# Get the code
git clone https://gitlab.lrz.de/MAGNETICUM/Hydro-OpenGadget3

# Get tutorial files
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Config.sh
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/box.param
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/runme.sh
wget https://www.usm.uni-muenchen.de/~dolag/Hydro/Hydro/Python/g3read.py

# Compile
cd Hydro-OpenGadget3 && make -j && cd ..

# Run
mkdir output
sbatch runme.sh

# Monitor
squeue -u $USER
tail -f slurm-*.out

# Python analysis
python3 show.py
ffplay -loop 0 basic_animation.mp4

# Restart
# mpirun ... Gadget3 box.param 1 (from checkpoint)
# mpirun ... Gadget3 box.param 2 (from snapshot)
```

Listing D.1: Essential commands for every tutorial.

On Table D.1 Config.sh changes per tutorial.

Table D.1: Required `Config.sh` changes per tutorial.

Tutorial	Enable	Disable
T0	PERIODIC, NOGRAVITY, WENDLAND_C2	—
T1	WENDLAND_C4	PERIODIC, NOGRAVITY
T2	PERIODIC, NOGRAVITY, WENDLAND_C2, TWO DIMS	—
T3	Vary kernel	—
T4	PERIODIC, NOGRAVITY, WENDLAND_C2, TWO DIMS, LONG_X=40	—
T5	WENDLAND_C4, EVALPOTENTIAL	PERIODIC, NOGRAVITY
T6	PERIODIC, NOGRAVITY, WENDLAND_C6, LONG_X=20	—
T7a	PERIODIC, NOGRAVITY, WENDLAND_C4	LONG_X/Y/Z
T7b	PERIODIC, NOGRAVITY, TWO DIMS, SPH_BND_*, LONG_X=30	—
T8a	PERIODIC, NOGRAVITY, WENDLAND_C4, SPH_BND_*, CONDUCTION, LONG_X=20	—
T8b	PERIODIC, NOGRAVITY, SPH_BND_*, LONG_X=30, LONG_Y=1, LONG_Z=1	—
T9a	PERIODIC, NOGRAVITY, TWO DIMS, SPH_BND_*, NAVIERSTOKES	—
T9b	+ARTIFICIAL_CONDUCTIVITY or +USE_MESHLESS_FINITE_MASS	—
T10	PERIODIC, NOGRAVITY, WENDLAND_C4, MAGNETIC, MU0_UNITY, LONG_X=100	—
T11	WENDLAND_C4, COOLING, SFR, GENERATIONS=4	PERIODIC, NOGRAVITY

D.2 Common Pitfalls

Forgot to recompile after Config.sh change Always make `-j` after any flag change.

Output directory does not exist Create `./output` before submitting.

Wrong particle type in read_new Gas = type 0, DM = type 1, Stars = 4.

Simulation hangs Usually too-small softening (timestep $\rightarrow$ 0) or memory overflow (`PartAllocFactor` too small for SFR runs).

NaN in snapshot Check `ArtBulkViscConst`: too small for a strong-shock simulation causes instability.

TREECOOL not found Copy it to the run directory: `cp Hydro-OpenGadget3/TREECOOL`

Bibliography

- Bagla, J. (2002). TreePM: A code for cosmological N-body simulations. *Journal of Astrophysics and Astronomy*, 23:185–196.
- Barnes, J. and Hut, P. (1986). A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449.
- Brio, M. and Wu, C. (1988). An upwind differencing scheme for the equations of ideal magnetohydrodynamics. *Journal of Computational Physics*, 75:400–422.
- Dehnen, W. and Aly, H. (2012). Improving convergence in smoothed particle hydrodynamics simulations without pairing instability. *Monthly Notices of the Royal Astronomical Society*, 425:1068–1082.
- Evrard, A. (1988). Beyond N-body: 3D cosmological gas dynamics. *Monthly Notices of the Royal Astronomical Society*, 235:911–934.
- Gingold, R. and Monaghan, J. (1977). Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389.
- Hahn, O. and Abel, T. (2011). Multi-scale initial conditions for cosmological simulations. *Monthly Notices of the Royal Astronomical Society*, 415:2101–2121.
- Hopkins, P. (2015). A new class of accurate, mesh-free hydrodynamic simulation methods. *Monthly Notices of the Royal Astronomical Society*, 450:53–110.
- Kennicutt, R. (1998). The global Schmidt law in star-forming galaxies. *The Astrophysical Journal*, 498:541–552.
- Lucy, L. (1977). A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82:1013–1024.
- Morris, J. and Monaghan, J. (1997). A switch to reduce SPH viscosity. *Journal of Computational Physics*, 136:41–50.
- Power, C., Navarro, J., Jenkins, A., Frenk, C., White, S., Springel, V., Stadel, J., and Quinn, T. (2003). The inner structure of LCDM haloes – I. a numerical convergence study. *Monthly Notices of the Royal Astronomical Society*, 338:14–34.
- Price, D. (2008). Modelling discontinuities and Kelvin–Helmholtz instabilities in SPH. *Journal of Computational Physics*, 227:10040–10057.
- Roediger, E., Brüggen, M., Simionescu, A., et al. (2013). Viscosity, pressure waves, and dumbbell galaxies: a unifying explanation for parsec-scale radio morphology. *Monthly Notices of the Royal Astronomical Society*, 436:1721–1740.

- Sedov, L. (1959). *Similarity and Dimensional Methods in Mechanics*. Academic Press, New York.
- Spitzer, L. (1962). *Physics of Fully Ionized Gases*. Interscience, New York, 2nd edition.
- Springel, V. (2005). The cosmological simulation code GADGET-2. *Monthly Notices of the Royal Astronomical Society*, 364:1105–1134.
- Springel, V. and Hernquist, L. (2002). Entropy-conserving smoothed particle hydrodynamics. *Monthly Notices of the Royal Astronomical Society*, 333:649–664.
- Springel, V. and Hernquist, L. (2003). Cosmological smoothed particle hydrodynamics simulations: a hybrid multiphase model for star formation. *Monthly Notices of the Royal Astronomical Society*, 339:289–311.
- Tricco, T., Price, D., and Bate, M. (2016). Constrained hyperbolic divergence cleaning in smoothed particle magnetohydrodynamics with variable cleaning speeds. *Journal of Computational Physics*, 322:326–344.
- Tully, R. et al. (2014). The Laniakea supercluster of galaxies. *Nature*, 513:71–73.