SEREN - Version 1.5.1

David Hubber, Christopher Batty & Andrew McLeod

July 3, 2013

Contents

1 Overview

SEREN is a Smoothed Particle Hydrodynamics (SPH) code designed for solving self-gravitating hydrodynamical problems in astrophysics, particularly in the fields of star and planet formation. SEREN largely grew from DRAGON, the star formation SPH code written by Simon Goodwin at Cardiff University, although many routines have significantly diverged from the original DRAGON versions, or have been rewritten from scratch. SEREN has also been designed to be compatible with DRAGON in its features and file formats.

The basic elements of SEREN can be used to simulate any problem involving hydrodynamics and gravity, but SEREN also contains many specialized features for star formation problems. The main features present in SEREN 1.5.1 include :

- Smoothed Particle Hydrodynamics (Standard SPH or conservative 'grad-h' SPH)
- Self-gravitating SPH / N-body dynamics
- Isothermal, barotropic or polytropic equations of state
- Octal-spatial (Barnes-Hut) neighbour-searching and gravity tree
- Sink particles, using new accretion algorithm of Hubber et al. (2013)
- Different particle types (gas, inter-cloud, boundary, CDM and dust particles)
- 1, 2 or 3 dimensions
- Periodic boundary conditions (independent for each dimension) or spherical wall
- Ewald method for periodic gravity
- Hierarchical block timesteps, with neighbour-checking for safety
- 2nd order Runge-Kutta, Leapfrog-KDK and Leapfrog-DKD and integration schemes
- Artificial viscosity with Balsara switch, time-dependence and Keplerian pattern-matching
- Artificial conductivity with switches
- N-body evolution of sinks using 4th order Hermite integrator at termination of SPH
- Radiative cooling approximation of Stamatellos et al. (2007), and hybrid flux-limited diffusion method (Forgan et al. 2009)
- Ionising radiation using HEALPix (Bisbas et al. 2009)
- Hybrid SPH and 4th-order Hermite N-body algorithm (Hubber et al. 2013)
- Simple external background gravitational potentials
- Parallelized using OpenMP
- Parallelized using MPI (partial implementation, currently in beta-mode)
- Bash script to run automated batch tests
- Output compatable with Splash (Price 2007)

Features currently in development, or implemented but not full tested

- Wind feedback from high-mass stars (Ngoumou & Hubber)
- Multiple fluid components and independent EOSs (Hubber)

2 Using SEREN

2.1 Obtaining SEREN via github

SEREN is hosted on the github website (https://github.com/) which uses the *git* (http://git-scm.com/) version-control software, written by Linux-kernel author Linus Torvalds. The SEREN code is held in the github repository

https://github.com/dhubber/seren,

and a webpage describing the features of the code, tests and other information can be found at

http://dhubber.github.io/seren/.

Although you must register to join and use github, the SEREN code itself can be downloaded anonymously. One advantage of joining github is that it is easier for us to track usage of the code and that it is possible for users to give feedback, such as bug reports or suggested improvements, or share information about the code that can be useful to other users. Another advantage is that you can use the github desktop manager, which can make downloading and updating the code easier for those who prefer graphical interfaces rather than simple command line interfaces.

In order to download SEREN, you must first install *git* on your system. It is required that the user has *git* version 1.6 or later. If you have version 1.5 or older, it is recommended asking your computer administrator if he/she could update *git* to the latest version, since I have noticed that v1.5 does different things with regards to password authentication and therefore the instructions below will be invalid. For your own computers (e.g. laptops), *git* can easily be obtained with package managers such as apt, rpm, etc .. For Mac users, *git* can be obtained with fink or macports. *git* can also be downloaded directly from the *git* webpage (http://git-scm.com).

Before downloading the code, either from the command line or using the github programme, you should set the following important variables in order to track your local changes (i.e. what changes are your own, and which changes are made by the authros):

```
git config --global user.name "Your name here"
git config --global user.email "Your email address here"
```

This creates a file called .gitconfig in your home directory containing this information.

Once this is done, you can download the code by one of two ways

- Anonymous command line download
  First change into the directory you wish to download the code to. Next, copy the https clone address into the command line as

  git clone https://github.com/dhubber/seren.git

- Desktop application download First, register with github and download and install the github desktop application. Next, go to the SEREN repository webpage and click the 'Clone in desktop' option. Select the directory you wish to download the code into and then follow the links to the end.

2.1.1 Updating Seren and managing conflicts on the command line

The SEREN git repository can be updated quite easily with a few commands. First, if you have changed any files in the repository (e.g. most likely with the Makefile or the params.dat file), then you have to commit your changes to the local repository. This can be done easily using

```
git commit -am "Message"
```

where "Message" is some status message which is recorded in the git logs. This command allows git to know the changes you have made to the SEREN files so it can easily be merged with the new version's updates. The update can now easily be obtained with the same command as above :

```
git pull origin master
```

with the same password to be entered at the prompt. If you have changed any part of the SEREN files which have also been changed in a different way by the update, then there will be a conflict and the merging of the two versions cannot proceed automatically. In this case, you have to intervene manually and resolve the conflict by selecting which version (i.e. your altered version or the new update) you would like to use. To view which files have a conflict (plus other information about your local repository), simply type

```
git status
```

Once you have identified which files have conflicts, you have to open each one individually with a text editor, and edit the conflicted regions (which are clearly marked with both versions of the code in conflict) and save with the chosen version. Once ALL conflicted files have been modified, you can inform your local repository by commiting the new files by again typing

```
git commit -am "Another message"
```

All conflicts should now be resolved, and you are free to update to any later versions. Note that you must commit your changes locally and resolve any potential conflicts every time you want to update the code.

2.1.2 Other important commands

A selected list of important commands that will be needed from time to time :

git log : Outputs log of various commits to screen
git log --online : Less verbose version of 'git log'
git status : Status of local repository, including what has been modified or added, but not commited
git branch : Tells you which code branch you are currently on (should always say master)
git diff : Displays difference between local files and those in the repository
git gc --aggressive : Compresses parts of the git repository to reduce the overall size

2.2 Compiling and running SEREN

SEREN has been designed so to be compiled with GNU make. The user must specify a number of compiler options, which are set at the head of the Makefile (see Section 2.4 for more information). In order to compile, a compatible compiler must be specified in the first line of the Makefile. SEREN has been successfully tested on the following operating systems and compilers.

- GNU/Linux
  - f95 - NAG f95 compiler (Linux workstations)
  - g95 - g95 compiler (Linux workstations)
  - gfortran - GNU Fortran compiler (Linux workstations)
  - ifort - Intel Fortran compiler (Merlin cluster)
  - pgf90 - Portland group Fortran compiler (Coma cluster)
  - pgf95 - Portland group Fortran compiler (Iceburg cluster)
- Mac OS X (1.4, 1.5, 1.6 & 1.7)
  - g95 - g95 compiler
  - gfortran - GNU Fortran compiler
  - ifort - Intel Fortran compiler

Once all the other Makefile options have been set to their desired values, SEREN is compiled by GNU make with the command

make [-j N] seren

GNU make will compile the source code of SEREN and produce the executable program seren. The optional argument, -j N, allows parallel compilation on multi-core architecture, where N is the number of routines to be compiled in parallel at any one time. Prior to performing a simulation, the user must set all simulation parameters in the file params.dat (See section 2.6 for more information) and provide an initial conditions file in the appropriate format. To run SEREN, the user must type

./seren

SEREN will read in the default parameters file params.dat before performing the simulation.

2.2.1 Command-line arguments

SEREN has a number of optional command-line arguments that can be invoked to change the behaviour of the SEREN executable. The behaviour can depend on several factors, particularly what Makefile options have been invoked while compiling SEREN.

Table 1: List of all command-line arguments available in SEREN

Argument	**Behaviour**
-d, -D, - - debug	SEREN prints out the debug output column data format to screen and then exits without running any simulation. (N.B. The same information is printed to the runid.params file when a simulation is performed using SEREN)
- - diag	SEREN prints out the column data format that is used in the diagnostics file to the screen (enabled with DIAGNOSTIC_OUTPUT = 1 in the Makefile).
-h, -H, - - help	SEREN prints out all available command-line options
-m, -M	SEREN prints out the Makefile options used to compile the code to screen and then exits without running any simulation. (N.B. The same Makefile options are printed to the runid.params file when a simulation is performed using SEREN)
-s, -S, - - sinks, - - stars	SEREN prints out the column data format to screen for the sink files.
-v, -V, - - version	SEREN prints out current version number
paramsfile	SEREN reads the parameters file paramsfile instead of the default params.dat

2.2.2 Restarting simulations

If a simulation is terminated for some reason, then it can be restarted by simply running SEREN without any modification to the params.dat file. Each simulation generates a file runid.restart which contains the name of the last snapshot to be outputted. SEREN will search for this file, and if it exists, it will read the snapshot name contained and restart the simulation from that point. If you do not wish to restart the simulation from this point, but want a fresh run, then you should delete the runid.restart file. If you wish to restart a simulation from a different snapshot, you can delete the runid.restart file and alter some of the parameters in the params.dat file, such as the restart logical flag (See Section 2.6).

2.3 SEREN-MPI

The SEREN source code also contains an MPI version, SEREN-MPI. In order to compile SEREN-MPI, one must download and install an implementation of MPI that works on your system. To date, we have tested SEREN-MPI using the mpich2 (http://http://www.mpich.org) library at the development stage. SEREN-MPI has not been fully tested or debugged using other MPI implementations (e.g. openmpi). Therefore, it is recommended to use mpich2 if possible.

We note that SEREN-MPI is currently only a partial implementation of all features and is in the beta-testing stage. Although it has been shown to run for a limited number of test cases, it has not been through an exhaustive list of tests. Therefore bugs may still (and probably do) exist in the code that can only be identified and fixed through thorough usage, bug-reporting, fixing, updating and re-testing. We welcome any feedback from users who wish to trial SEREN-MPI to help us fix any problems rapidly.

2.3.1 Compiling and running SEREN-MPI

In order to compile SEREN-MPI, several extra options must be set in the Makefile. These options are

- MPIF90
- MPI
- MPI_LIBRARY

The default values to compile with MPI should be MPIF90 = mpif90, MPI = 1 and MPI_LIBRARY = mpich2. If the user wishes to compile also with OpenMP, then this flag should also be switched on. In this mode, the code will run on each local node parallelised with OpenMP, but parallelised using MPI to communicate between nodes. This mode could potentially have scaling advantages, although this will be determined with more testing of the code.

make or make seren

To run the code

mpiexec -n N ./seren-mpi

where N is the number of tasks that the MPI job will run on.

2.3.2 Combining data snapshots with MPI

Since MPI runs segments of the simulation on different CPU tasks (often on different nodes), it is more efficient during runtime for each task to write its own contribution to the overall simulation to a separate file. In order for these files to be meaningfully analysed with SPLASH or some other data analysis tool, they must be combined together to a single file per output. A python script has been written to easily accomplish this with minimal effort, called joinlots.py which is located in the joinsim sub-directory. It may be required to add the pathname of both the main SEREN directory and the joinsim sub-directory to your PATH environment variable. To join all snapshots for a simulation, we simply type :

joinlots.py runid

where runid is the run id of the simulation as chosen in the parameters file. (N.B. this currently only works with double precision simulations).

2.4 Makefile

The head of the Makefile contains the complete list of compilation variables that are available. Most variables have two or more possible values which must be entered in the Makefile. If an illegal value is entered, then make will halt during compilation, or the program will stop during runtime (see the routine /main/sanitycheck.F90). The Makefile is technically split into two separate files; Makefile, which contains the user options, and makefiletail.mk, which processes all the selected options to compile the code. The full list of all Makefile variables and possible options is given in the table below.

Table 2: List of all available Makefile options in SEREN

Variable	**Options**	
F90	f95 :	NAG f95 compiler (Linux)
	g95 :	free (not gnu) f95 compiler (Linux, Mac OS X)
	gfortran :	gnu f95 compiler (Linux, Mac OS X)
	pgf90 :	Portland Group compiler (Coma cluster)
	mpif90 :	Portland Group compiler (Coma cluster)
	ifort :	Intel Fortran compiler (Merlin cluster)
MPIF90	mpif90 :	MPI Fortran compiler
VERSION_NO		Version no. string
SRCDIR		Absolute path of main SEREN directory (default $(PWD)/src)
EXEDIR		Absolute path of location for SEREN executable (default $(PWD))
OPENMP	0 :	Compile as serial code
	1 :	Compile using OpenMP directives
MPI	0 :	Compile as serial code
	1 :	Compile with MPI directives
MPI_LIBRARY	mpich2	Name of MPI library used
COMPILER_MODE	0 :	No optimisation or debugging flags
	STANDARD :	Use standard optimisation flags (-O3 plus inline functions)
	FAST :	Use fast optimisation flags for increased speed. (Note that fast math optimisations are potentially unsafe and can lead to floating point errors, even in apparently bug-less code, and therefore should not be used without testing with the required compilation flags.)
	DEBUG :	Use all available debug flags (e.g. bounds-testing, floating point erros, etc..). Should be selected when debugging the code using gdb, or another debugger.
OUTPUT_LEVEL	0 :	Output/debug flags switched off
	1 :	Output/debug flags switched on to level 1
	2 :	Output/debug flags switched on to level 2
	3 :	Output/debug flags switched on to level 3
DIAGNOSTIC_OUTPUT	1 :	Print diagnostic information of conserved quantities (e.g. total mass, momentum, energy) to screen (if OUTPUT_LEVEL ¿ 0) and file ('run_id.diag')

Variable	**Options**	
	0 :	Do not compute diagnostic information
NDIM	1 : 2 : 3 :	One-dimensional Two-dimensional Three-dimensional
PRECISION	SINGLE : DOUBLE :	Single precision for main real variables Double precision for main real variables
INFILE_FORMAT	ALL : DRAGON : SEREN : ASCII :	Include routines to read all possible file formats Only include DRAGON-format reading routines Only include SEREN-format reading routines Only include column-ASCII format reading routine
OUTFILE_FORMAT	ALL : DRAGON : SEREN : ASCII :	Include routines to write all possible file formats Only include DRAGON-format writing routines Only include SEREN-format writing routines Only include column-ASCII format writing routine
PERIODIC	1 : 0 :	Periodic boundary conditions (Note : must be set to 1 if any of X_BOUNDARY, Y_BOUNDARY, Z_BOUNDARY or SPHERICAL_MIRROR are set to any value other than 0) No periodic boundary conditions
X_BOUNDARY	PERIODIC : WALL : 0 :	Periodic box in x-dimension Walls in LHS and RHS directions of x-dimension No periodicity in x-dimension
Y_BOUNDARY	PERIODIC : WALL : 0 :	Periodic box in y-dimension Walls in LHS and RHS directions of y-dimension No periodicity in y-dimension
Z_BOUNDARY	PERIODIC : WALL : 0 :	Periodic box in z-dimension Walls in LHS and RHS directions of z-dimension No periodicity in z-dimension
SPHERICAL_WALL	1 : 0 :	Spherical mirror to reflect particles that exceed some give radial distance No spherical mirror
CYLINDRICAL_WALL	1 : 0 :	Cylindrical mirror to reflect particles that exceed some given distance about z-axis No spherical mirror
GHOST_PARTICLES	1 : 0 :	Use ghost particles for periodic boundaries (experimental; advise not to use for now) No ghost particles; use relative position periodic wrapping
SPH_SIMULATION	1 : 0 :	Perform SPH simulation No SPH simulation

Variable	**Options**	
NBODY_SPH_SIMULATION	1 :	Perform hybrid N-body/SPH simulation
	0 :	No hybrid simulation
NBODY_SIMULATION	1 :	Perform N-body simulation
	0 :	No N-body simulation
SPH	STANDARD :	Use traditional SPH formulation (Monaghan 1992)
	GRAD_H_SPH :	Use 'grad-h' SPH formulation (Springel & Hernquist 2002; Price & Monaghan 2004)
	RTSPH :	Use Ritchie & Thomas (2001) SPH formulation
SPH_INTEGRATION	RK2 :	2nd order Runge-Kutta integration scheme
	LFKDK :	2nd order Leapfrog kick-drift-kick scheme
	LFDKD :	2nd order Leapfrog drift-kick-drift scheme
KERNEL	M4 :	M4 kernel (Monaghan & Lattanzio 1985)
	M4TC :	M4 kernel with modifed 1st derivative (Thomas & Couchman 1992)
	QUINTIC :	Quitic kernel (Morris 1996)
	QUINTICTC :	Quintic kernel with modified 1st derivative (c.f. Thomas & Couchman 1992)
	GAUSSIAN_3H :	Gaussian kernel truncated at 3 h
HFIND	NUMBER :	Determine h by number of neighbours
	MASS :	Determine h by total mass of neighbours
	H_RHO :	Determine h by iterating h-rho relation (as in 'grad-h' SPH')
	CONSTANT :	Use contant smoothing length
MINIMUM_H	1 :	Set a minimum smoothing length
	0 :	Allow any smoothing length
HYDRO	1 :	Hydro forces switched on
	0 :	No hydro forces
ENERGY_EQN	1 :	Activate energy equation
	0 :	Do not include energy equation in compliation
ENTROPY_EQN	1 :	Activate entropy equation
	0 :	Do not include entropy equation in compliation
ARTIFICIAL_VISCOSITY	MON97 :	Monaghan (1997) artificial viscosity
	AB :	Standard α-β viscosity (Monaghan & Gingold 1983)
	0 :	No artificial viscosity
VISC_TD	1 :	Use time-dependent value of α (Morris & Monaghan 1997)
	0 :	Constant value for α
BALSARA	1 :	Use Balsara switch (Balsara 1995)
	0 :	No Balsara switch
PATTERN_REC	1 :	Use Keplarian pattern-matching (Cartwright & Stamatellos 2010)
	0 :	Do not use pattern matching
ARTIFICIAL_CONDUCTIVITY	0 :	No artificial conductivity
	PRICE2008 :	Artificial conductivity with constant $\alpha_{\rm COND}$ (Price 2008)

Variable	Options	
	WADSLEY2008 :	Wadsley et al. (2008) conductivity
EXTERNAL_PRESSURE	0 : 1 :	No external pressure Simple external pressure formulation
RAD_WS	1 : 0 :	Activate radiative cooling scheme (Stamatellos et al. 2007) Do not include radiative cooling in compilation
FLUX_LIMITED_DIFFUSION	1 : 0 :	Switch on flux-limited diffusion for hybrid radiation scheme (Forgan et al. 2009). Only activated when RAD_WS = 1. No flux-limited diffusion
SINK_POTENTIAL_WS	1 : 0 :	Use gravitational potential from sink in radiative cooling calculations Ignore gravitational potential from sinks for cooling
AMBIENT_HEATING_WS	1 : 0 :	External ambient heating source (e.g. CMB) No ambient heating
SINK_HEATING_WS	STAR_HEATING : STAR_SIMPLE_HEATING : HDISC_HEATING : 0 :	
COOLING_HEATING	0 : EXPLICIT :	No cooling or heating terms added Explicit cooling/heating terms added to energy equation (Experimental)
IONIZING_RADIATION	0 : SINGLE_STATIC_SOURCE : MULTIPLE_SINK_SOURCES :	No ionizing sources Single static source of ionizing radiation (Bisbas et al. 2009) Sink particles act as sources of ionizing radiaiton (experimental)
STELLAR_WIND	0 : SINGLE_STATIC_SOURCE :	No wind sources Single static source of wind
EXTERNAL_FORCE	0 : PLUMMER : UDS : NFW1996 :	No external forces Plummer sphere potential Uniform density sphere potential Navarro, Frenk & White (1996) potential
SELF_GRAVITY	0 : KS : NBODY :	No gravitational forces computed Kernel-softened gravity for 2-body forces Newton's gravitational law for all 2-body forces
MEANH_GRAVITY	1 : 0 :	Use mean-h gravity (cf. Price & Monaghan 2007) Use default gravity
EWALD	1 : 0 :	Ewald periodic gravity switched on No Ewald corrections
SINKS	0 : SIMPLE : NO_ACC : SMOOTH_ACC :	No sinks Simple (i.e. only gravitating) sinks (Bate, Bonnell & Price 1995) Simple sinks with no accretion Sinks with smooth accretion (Hubber et al. 2013)

Variable	Options	
SINK_RADIUS	FIXED_ABSOLUTE :	Absolute value (in AU in params.dat file; same for all sinks)
	FIXED_HMULT :	Multiple of mean h at sink density (same for all sinks)
	HMULT :	Multiple of h at sink density (individual values for sinks)
SINK_REMOVE_ANGMOM	1 :	Deposit sink ang. mom. on nearby particles
	0 :	Sink particle retain ang. mom. of accreted particles
SINK_GRAVITY_ONLY	0 :	Consider all physical sources of gravity
	KS :	Sinks only source of gravity using Kernel-softened gravity for 2-body forces
	NBODY :	Sinks only source of gravity using Newton's gravitational law for all 2-body forces
NBODY_INTEGRATION	HERMITE4 :	4th-order Hermite integration scheme (Makino & Aarseth 1992)
	LFKDK :	2nd-order Leapfrog KDK integration scheme
BINARY_STATS	1 :	Calculate binary statistics and output to file
	0 :	No binary calculations
TREE	0 :	No tree (all quantities calculated by direct summation)
	BH :	Use Barnes-Hut tree (octal-spatial; Barnes & Hut 1985)
MULTIPOLE	0 :	No higher-order multipole terms
	QUADRUPOLE :	Include quadrupole moment terms in gravity calculations
	OCTUPOLE :	Include both octupole and quadrupole moment terms
MAC	GEOMETRIC :	Use standard Barnes-Hut geometric opening angle criterion (Barnes & Hut 1985)
	GADGET :	Use Gadget-style higher-order moment criterion (Springel et al. 2002)
	GADGET2 :	Use Gadget 2.0 moment (Springel 2005)
	EIGEN :	Use Eigenvalues of Q tensor to compute appropriate MAC (Hubber et al. 2010)
REORDER	PARTICLES :	Re-order particles in arrays according to tree-walk order
	0 :	No re-ordering
SORT	INSERTION :	Use insertion sort for sorting lists
	HEAP :	Use heapsort for sorting lists
TIMESTEP	ADAPTIVE :	Block timestep levels adjusted at resync
	FIXED :	Fixed block timestep levels, with maximum level set by the $\mathrm{dt\_fixed}$ parameter
	RESTRICTED :	Timestep levels can only take certain values (dt_fixed parameter times integer power of 2), but are readjusted at resync
CHECK_NEIB_TIMESTEP	2 :	Ensure neighbours have similar timesteps
	1 :	As option 2, but doesn't change timestep in middle of current step

Variable	**Options**	
	0 :	No neighbour timestep comparison
SIGNAL_VELOCITY_DT	1 : 0 :	Use signal velocity for Courant timestep Use velocity divergence for Courant timestep
NEIGHBOURLISTS	1 : 0 :	Store neighbour lists in memory Do not store neighbour lists in memory
KERNEL_TABLES	1 : 0 :	Tabulate kernels in arrays for quick lookup Use inline function calls for kernel functions
REMOVE_OUTLIERS	1 : 0 :	Remove outlying particles from the simulation (Experimental) No removal of outliers
TURBULENT_FORCING	1 : 0 :	.... No turbulent forcing
TIMING_CODE	1 : 0 :	Use custom subroutines to produce timing statistics No timing output
DIMENSIONLESS	1 : 0 :	Dimensionless simulation (all scaling variables set to zero) Scaled variables (units specified in params.dat)
TEST	FREEFALL : SPIEGEL : BINARY : PLUMMER : ENTROPY : 0 :	Freefall collapse test Spiegel (ref??) test Orbitting binary stars test Plummer sphere stability test Entropy core test No test flags

2.5 Debug flags

The SEREN Makefile contains a number of debug flags below the main options which can be switched on or off by uncommenting them or commenting them out. Most of the debug flags produce verbose output of each routine, and in some cases produce extra files with more important information. The full list of debugging options with additional output is shown in the table below.

Table 3: List of special debugging options available in SEREN

Variable	Options
IEEE_EXCEPTION_HANDLING	
DEBUG_DIV_A	
DEBUG_ACCRETE	
DEBUG_ALLOCATE_MEMORY	
DEBUG_BHTREEBUILD	
DEBUG_BHTREESTOCK	
DEBUG_BHTREEWALK	
DEBUG_BHTREEGRAVITY	
DEBUG_BINARY_PROPERTIES	Ouput binary properties to screen when calculated
DEBUG_BINARY_SEARCH	
DEBUG_BLOCK_TIMESTEPS	Outputs occcupation of timestep levels
DEBUG_COPY_PARTICLE_DATA	
DEBUG_CREATE_SINK	
DEBUG_CREATE_HP_SOURCE	
DEBUG_DENSITY	
DEBUG_DIV_V	
DEBUG_DUDTRAD	
DEBUG_ENERGY_EQN	
DEBUG_FOLIATE	
DEBUG_FORCES	Records individual grav, hydro, magnetic forces
DEBUG_FREEFALL	
DEBUG_GATHER_NEIB	
DEBUG_GET_NEIB	
DEBUG_GHOST_PARTICLES	
DEBUG_GRAD_H_SPH	
DEBUG_GRID_RESULTS	
DEBUG_HEAPSORT	
DEBUG_HERMITE4	
DEBUG_H_GATHER	
DEBUG_H_GATHER_DENSITY	
DEBUG_H_GUESS	
DEBUG_HP_IF	
DEBUG_HP_OUTPUT	Outputs various ionization properties to files
DEBUG_HP_SPLIT_ACTIVE_RAYS	
DEBUG_HP_WALK_ALL_RAYS	
DEBUG_HP_WALK_RAY	
DEBUG_INTEGRATE	
DEBUG_KERNEL	Outputs file 'kernel.dat'
DEBUG_MHD	
DEBUG_NBODYSETUP	
DEBUG_OUTPUT_STAR_DATA	
DEBUG_PARAMETERS	

Variable	Options
DEBUG_PLOT_DATA	Outputs regular debug files with snapshot files. Files are simple column-data files where the information of each column is written to the run_id.params file.
DEBUG_RAD	Outputs 'run_id.rad' file for RAD_WS tests
DEBUG_REDUCE_TIMESTEP	
DEBUG_REMOVE_OUTLIERS	
DEBUG_RSPH_OUTPUT	Outputs files in RSPH format
DEBUG_SINK_REMOVE_ANGMOM	
DEBUG_SINK_SEARCH	
DEBUG_SINK_TIMESTEP	
DEBUG_SKELETON	
DEBUG_SMOOTH_ACCRETE_PARTICLES	
DEBUG_SPH_UPDATE	
DEBUG_SWAP_PARTICLE_DATA	
DEBUG_TIMESTEP_SIZE	
DEBUG_TRACK_PARTICLE	Outputs single file ('track1.dat') which contains large amount of information (same as that ouputted by debug files including the time) of one single chosen particle (set by parameter ptrack in params.dat file) which is printed every timestep.
DEBUG_TREE_BUILD	
DEBUG_TREE_GRAVITY	
DEBUG_TREESTOCK	
DEBUG_TREEWALK	
DEBUG_TYPES	
DEBUG_VISC_BALSARA	
DEBUG_VISC_PATTERN_REC	
DEBUG_WRITE_MPI_TASK	

2.6 Parameter file

SEREN contains all simulation information in a single parameter file, called params.dat. The information contained in the parameters file in version 1.0 is shown in the following table.

Table 4: List of user parameters in SEREN

Variable	**Type**	**Description**
run_id	char(256)	Run identifier string
run_dir	char(256)	Output directory name
in_file	char(256)	Name of initial conditions file
in_file_form	char(50)	Format of initial conditions file
out_file_form	char(50)	Format of output snapshot files
restart	logical	Is this a restart or a new run?
com_frame	logical	Change to centre of mass frame?
rseed	int	Random number seed
ptrack	int	i.d. of tracked particle
sph_endtime	DP	End time of SPH simulation
nbody_sph_endtime	DP	End time of hybrid N-body/SPH simulation
nbody_endtime	DP	End time of N-body simulation
firstsnap	DP	Time of first snapshot
snaptime	DP	Snapshot time interval (in real time)
noutputstep	int	Screen output interval (in integer steps)
ntempstep	int	Temporary snapshot interval (in integer steps)
ndiagstep	int	Integer time interval between diagnostic output
nsinkstep	int	Sink output time interval (in integer time)
nsnapstep	int	Snapshot time interval (in integer time)
courant_mult	DP	Courant timestep multiplication factor
accel_mult	DP	Acceleration timestep multiplication factor
sink_mult	DP	Sink accel. timestep multiplication factor
nbody_timemult	DP	Timestep factor for N-body simulations
nlevels	int	Number of multiple timestep levels
dt_fixed	DP	Fixed ref. time for creating timestep levels
runit	char(256)	Length scaling unit
munit	char(256)	Mass scaling unit
tunit	char(256)	Time scaling unit
vunit	char(256)	Velocity scaling unit
aunit	char(256)	Acceleration scaling unit
rhounit	char(256)	Density scaling unit
sigmaunit	char(256)	Column density scaling unit
Punit	char(256)	Pressure scaling unit
funit	char(256)	force scaling unit
Eunit	char(256)	Energy scaling unit
momunit	char(256)	Momentum scaling unit
angmomunit	char(256)	Angular momentum scaling unit
angvelunit	char(256)	Angular velocity scaling unit
dmdtunit	char(256)	Accretion rate scaling unit
Lunit	char(256)	Luminosity scaling unit
kappaunit	char(256)	Opacity scaling unit
Bunit	char(256)	Magnetic field (B-field) scaling unit
Qunit	char(256)	Electric charge unit
Junit	char(256)	Current density unit

Variable	**Type**	**Description**
uunit	char(256)	Specific internal energy unit
dudtunit	char(256)	Rate of change of specific internal energy unit
tempunit	char(256)	Temperature unit
rscale	DP	Length scaling factor
mscale	DP	Mass scaling factor
periodic_min(1)	PR	Size of periodic box in x-dimension
periodic_max(1)	PR	Size of periodic box in x-dimension
periodic_min(2)	PR	Size of periodic box in y-dimension
periodic_max(2)	PR	Size of periodic box in y-dimension
periodic_min(3)	PR	Size of periodic box in z-dimension
periodic_max(3)	PR	Size of periodic box in z-dimension
rspheremax	PR	Radius of spherical wall
psphere	int	Mirror origin id (0 : co-ordinates origin; p : SPH particle; -s : sink particle)
pp_gather	int	Neighbours required to determine h
hmin	PR	Minimum allowed smoothing length
h_fac	PR	grad-h density-h iteration factor
boundaryeos	char(256)	Boundary particle equation-of-state
icmeos	char(256)	ICM particle equation-of-state
gaseos	char(256)	Gas particle equation-of-state
isotemp	PR	Temperature for isothermal, barotropic EOSs (K)
rhobary	PR	Adiabatic density for barotropic density (cgs units)
gamma	PR	Ratio of specific heats
mu_bar	PR	Mean gas particle mass (in a.m.u.)
Kpoly	PR	Polytropic constant
Pext	PR	External pressure
cooling_law	char(256)	Cooling law used
alpha	PR	α-viscosity value
beta	PR	β-viscosity value
alpha_min	PR	Minimum value of α
abserror	PR	Absolute error fraction in GADGET MAC
thetamaxsqd	PR	Maximum opening angle squared (Geometric MAC)
nbuildstep	int	Frequency of DRAGON tree builds (in integer time units)
rhosink	PR	Sink formation density (cgs units)
sinkrad	PR	Sink radius (in units of h or in AU depending on options)
nsearchstep	int	No. of integer timesteps between sink search
rho_search	logical	Calculate density for selecting sink candidates
potmin_search	logical	Only consider particles at potential minimum
hill_sphere_search	logical	Hill spheres of sinks must not overlap
energy_search	logical	Only create sinks from bound objects
thermal_search	logical	Only create sinks from thermally bound objects
div_v_search	logical	Only create sinks from converging objects
div_a_search	logical	Do not create sinks if particles are accelerating apart
timescale_search	logical	Compare timescales for sink formation
energy_accrete	logical	Only accrete bound particles
alpha_ss	PR	Sunyaev-Shakura viscosity parameter
smooth_accrete_frac	PR	Fraction of mass for instant accretion
smooth_accrete_dt	PR	Timestep fraction for instant accretion
f_accretion	PR	Fraction of accretion energy radiated as luminosity

Variable	Type	Description
feedback_tdelay	PR	Time delay between sink formation and feedback
feedback_minmass	PR	
rho_remove	logical	Remove particles below density threshold?
energy_remove	logical	Remove unbound particles from system?
rad_remove	logical	Remove distant particles?
rholost	PR	Density removal threshold
rad_lost	PR	Distance removal threshold
npec	int	No. of Predict-correct-evaluate iterations
nbody_frac	PR	Fraction of mass accreted before switching to N-body
ptemp0	PR	Disc temperature at r = 1 AU from star (K)
temp_inf	PR	Disc temperature at infinity (K)
ptemp_r0	PR	Temperature softening radius ($\ll$ 1 AU)
ptemp_q	PR	Temperature power law index
fcolumn	PR	Column polytrope factor
nionallstep	int	Integer steps inbetween HEALPix walk
f1	PR	Integration step accuracy variable
f2	PR	HEALPix resolution factor
f3	PR	Temperature smoothing parameter
f4	PR	Density interpolation parameter
Tneut	PR	Temperarure of neutral gas
Tion	PR	Temperature of ionized gas
Xfrac	PR	Fraction of hydrogen
mu_ion	PR	Mean gas particle mass for ionisied gas
a_star	PR	Recombination coefficient
N_LyC	PR	No. of ionizing photons per second
rstatic	PR(1:3)	Location of single static ionizing source
lmax_hp	PR	Maximum allowed number of HEALPix levels
M_loss	PR	Wind mass loss rate from source
v_wind	PR	Wind velocity from star
comp_frac	PR	..
turb_T	PR	..
turb_Ndt	PR	..
turb_min	PR(1:3)	..
turb_max	PR(1:3)	..

3 Generating initial conditions

SEREN contains a large number of small programs which can be used to generate initial conditions to run simulations. These programs are contained in the sub-directory /seren/ic and can be compiled. To compile any initial conditions program of some name ic_name, simply type

```
make ic_name
```

Some of the initial conditions programs require their own separate parameters file, a template of which can be found in the seren/datafiles sub-directory. These parameters files must be copied into the main seren run directory in order to be accessed by the initial conditions program. To run the initial conditions program, type

```
./ic_name
```

3.1 ic_BB

ic_BB sets-up the Boss-Bodenheimer test (Boss & Bodenheimer 1979), i.e. a uniform density sphere with an azimuthal density perturbation in solid-body rotation. Program reads in a unifrom density sphere of unit radius (centred at the origin), scales to the required density and radius, and then adds the azimuthal perturbation and a solid-body velocity field. The original Boss-Bodenheimer test considered simply an isothermal EOS, but many subsequent studies have used barotropic and other EOSs. Parameters read in from file BBparams.dat.

Required Makefile options :

- NDIM = 3
- SPH = STANDARD/GRAD_H_SPH
- HYDRO = 1
- GRAVITY = KS
- DIMENSIONLESS = 0

Variable	Type	Description
in_file	char(256)	Input filename (file containing uniform density sphere of unit radius)
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
mass	PR	Mass of cloud
munit	char(256)	Mass unit
rcloud	PR	Radius of cloud
runit	char(256)	Length unit
temp_cloud	PR	Temperature of cloud
angvel	PR	Angular velocity of cloud
angvelunit	char(256)	Angular velocity unit
mpert	integer	Order of azimuthal perturbation (usually mpert=2)
amp	PR	Amplitude of density perturbation (usually 0.1 or 0.5)

3.2 ic_binary

ic_binary sets up a binary system from two polytropes (or other self-gravitating structures) read in from files. Parameters read in from file binaryparams.dat.

Required Makefile options :

- NDIM = 3
- HYDRO = 1
- DIMENSIONLESS = 1

Variable	Type	Description
in_file1	char(256)	Input filename 1
in_file2	char(256)	Input filename 2
in_file_form1	char(256)	Input file 1 format
in_file_form2	char(256)	Input file 2 format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
abin	PR	Separation (semi-major axis) of binary
ecc	PR	Eccentricity of binary
corot	logical	Are stars co-rotating with binary orbit?

3.3 ic_core

ic_core creates a spherically symmetric density distribution for any given density function (as a function of radial distance). Currently only contains the distribution for a plummer-like density profile and a radial power-law density function. Requires the params file core.dat.

Required Makefile options :

- NDIM = 3
- HYDRO = 1
- SELF_GRAVITY = KS
- DIMENSIONLESS = 0

3.4 ic_jeans

ic_jeans sets-up the Jeans instability test (Hubber et al. 2006) which tests the ability of SEREN to resolve the Jeans gravitational instability. Program reads in a relaxed unit cube (with the cube placed in positive octant) and stretches the particle distribution to produce a 1-D sinusoidal density perurbation. Currently reads in parameters from the command line rather than via a separate parameters file.

Required Makefile options :

- NDIM = 3
- PERIODIC = 1
- PERIODIC_X = 1
- PERIODIC_Y = 1
- PERIODIC_Z = 1
- SPH = STANDARD/GRAD_H_SPH
- HYDRO = 1
- GRAVITY = KS
- EWALD = 1
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
in_file	char(256)	Input filename (File containing unit-uniform density sphere)
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
npert	int	No. of wavelengths
amp	PR	Amplitude of sinuosoidal perturbation

3.5 ic_KH

ic_KH creates the initial conditions for the Kelvin-Helmholtz instability test. Requires KHparams.dat file.

3.6 ic_lattice_cube

ic_lattice_cube creates a cubic-lattice distribution of particles with side-length length and ppd particles in each dimension. Therefore the total number of particles in the lattice is $\mathrm{ppd}^{\mathrm{NDIM}}$. In 1-D, the program produces a uniformly-spaced line of particles, in 2-D a square-grid of particles, and in 3-D a cubic lattice. The lattice extends from 0 to length in each dimension. Parameters are currently read in from the command-line.

Required Makefile options :

- NDIM = 1/2/3
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
ppd	integer	Particles per dimension in lattice (Must be a positive integer)
length	PR	Total length of lattice edge (For a unit cube, length = 1)
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format

3.7 ic_NTSI

ic_NTSI generates the initial conditions for the non-linear thin-shell instability (NTSI) test. Requires the parameters file NTSIparams.dat.

3.8 ic_plummer

ic_plummer generates the a Plummer sphere, either with stars, gas, cdm particles, or a mixture of the three. Requires the parameters file plummer.dat.

3.9 ic_polytrope

Creates a finite polytrope/infinite polytrope with surrounding medium from a uniform-density sphere of unit radius centred at the origin. For an isothermal polytrope (e.g. a Bonner-Ebert sphere), the inputted sphere is divided into 4 regions; the polytrope (self-gravitating gas), the gas envelope (self-gravitating gas), the surrounding inter-cloud medium (non-self gravitating gas) and a static outer-wall (boundary particles). The outer-three regions are optional depending on the parameters selected in polytrope.dat.

Required Makefile options :

- NDIM = 3
- THERMAL = ISOTHERMAL/POLYTROPIC/BAROTROPIC
- HYDRO = 1
- SELF_GRAVITY = KS
- DIMENSIONLESS = 0

Variable	**Type**	**Description**
in_file	char(256)	Input filename (File containing unit-uniform density sphere)
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
isocloud	logical	Flag true if isothermal polytrope (if true, gas_eos must equal isothermal)
etapoly	PR	Polytropic index
xi_bound	PR	Dimensionless cloud boundary (6.35 for a mariginally stable Bonner-Ebert sphere)
mpoly	PR	Mass of cloud
munit	char(256)	Mass unit (e.g. m_sun)
rho0	PR	Central density of cloud (Only if mflag = rho0)
rhounit	char(256)	Density unit
mflag	char(20)	Set the total mass (mass) or central density (rho0) of the polytrope
Kpoly	PR	Polytropic constant, or a_0^2 for isothermal polytrope
vunit	char(256)	Velocity unit (unit of isothermal speed of gas if isothermal polytrope is selected)
menvelope	PR	Mass of gas envelope around polytrope (distributed uniformly around the polytrope with the same density and pressure as the polytrope at its surface)
micm	PR	Mass of IcM envelope which surrounds gas (distributed uniformly around the polytrope/gas envelope with the same density and pressure as the polytrope at its surface)
hboundary	PR	Size of static boundary zone (in units of the mean smoothing length; should be 3 or 4 to ensure no edge effects occur for interior gas particles)

3.10 ic_radtest

ic_radtest creates the initial conditions to perform the Masunaga-Inutsuka test (Masunaga & Inutsuka ????) using the radiative cooling method of Stamatellos et al. (2007; RAD_WS option).

3.11 ic_random_cube

ic_random_cube creates a line, sheet or cube (depending on the dimensionality) of randomlly-placed particles. Distributes particles between 0 and length in each dimension. Parameters are currently read in from the command-line rather than a separate parameters file.

Required Makefile options :

- NDIM = 1/2/3
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
ptot	int	Total number of particles
length	PR	Total length of lattice edge
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format

3.12 ic_replicate_cubes

Loads in a unit cube (from 0 to 1 in each dimension) and creates nrepeat periodic replicas in each dimension. The larger cube is then scaled to a unit cube itself. Used to create large-relaxed uniform density fields from smaller files. Parameters are read in from the command-line rather than a separate parameters file.

Required Makefile options :

- NDIM = 1/2/3
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
in_file	char(256)	Input filename (File containing unit cube)
in_file_form	char(256)	Input file format
nrepeat	int	No. of replicas in each dimension (must be a positive integer)
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format

3.13 ic_RT

Generates initial conditions for Rayleigh-Taylor instability test. Prepares two layers of gas with different densities in hydrostatuc balance on top of each other with a sinusoidal density perturbation to seed the instability. A cubic grid of particles is generated rather than reading in a file. Parameters are read in from the file RTparams.dat.

Required Makefile options :

- NDIM = 2
- PERIODIC = 1
- PERIODIC_X = 1
- PERIODIC_Y = 1
- ENERGY_EQN = 1
- HYDRO = 1
- SELF_GRAVITY = 0
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
pertmode	char(20)	Perturbation mode (1=velocity, 2=boundary)
ppd1,ppd2	int	Particles per dimension
nlayers1,nlayers2	int	No. of layers of particles (in y-direction)
rho1,rho2	PR	Densities
Press1	PR	Pressure
acc_grav	PR	External y-gravitational acceleration
gamma	PR	Ratio of specific heats
xsize	PR	x..
amp	PR	Amplitude of y-velocity perturbation
lambda	PR	Wavelength of perturbation
pp_gather	PR	Required no. of SPH neighbours
hmin	PR	Minimum smoothing length
h_fac	PR	'grad-h' SPH factor

3.14 ic_sedov

Creates initial conditions for Sedov blast-wave test from inputted unit-uniform density sphere. Requires inputting a unit-sphere. A 'point-explosion' is added by giving the central particle and its neighbours a total energy of unity (weighted by the kernel from the centre, while the rest of the particles equally share an energy of total 10^{-6}. Parameters are read in from the file sedovparams.dat.

Required Makefile options :

- NDIM = 3
- PERIODIC = 0
- PERIODIC_X = 0
- PERIODIC_Y = 0
- PERIODIC_Z = 0
- HYDRO = 1
- ENERGY_EQN = 1
- SELF_GRAVITY = 0
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
in_file	char(256)	Input filename (File contains unit sphere)
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
rho0	PR	Density of sphere
radius	char(20)	Radius of sphere after rescaling

3.15 ic_shocktube

Generates initial conditions for general 2-part shocktube tests (e.g. Sod 1978). Reads in two relaxed cubic density distribution, creates periodic replicas in the x-direction to elongate the shocktube and sets particle properties to create the desired test problem. Parameters are read in from the file sodparams.dat.

Required Makefile options :

- NDIM = 1/2/3
- PERIODIC = 1
- PERIODIC_X = 1
- PERIODIC_Y = 1
- PERIODIC_Z = 1
- HYDRO = 1
- ARTIFICIAL_VISCOSITY = AB/MON97
- SELF_GRAVITY = 0
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
file1	char(256)	Input filename
file1_form	char(256)	Input file format
file2	char(256)	Input filename
file2_form	char(256)	Input file format
p1, p2	int, int	No. of particles in file 1, 2
n1, n2	int, int	No. of replicas for LHS/RHS
rho1, rho2	PR, PR	Density of LHS/RHS layers
Press1, Press2	PR, PR	Pressure for LHS/RHS
x1, x2	PR, PR	x
y1, y2	PR, PR	y
z1, z2	PR, PR	z
v1(1), v2(1)	PR, PR	vx
v1(2), v2(2)	PR, PR	vy
v1(3), v2(3)	PR, PR	vz
B1(1), B2(1)	PR, PR	Bx
B1(2), B2(2)	PR, PR	By
B1(3), B2(3)	PR, PR	Bz

3.16 ic_sphere

Creates a spherical distribution of particles of unit radius and centred on the origin containing an exact number of particles. First, loads in a unit cube and then iterates to find the radius that contains the correct number of particles. Finally the spherical cut is rescaled and placed at the origin. Will fail to find the required number of particles if the inputted unit cube has too few particles. Sphere parameters are read in from the file $\mathrm{sphereparams.dat}$.

Required Makefile options :

- NDIM = 3
- PERIODIC = 0
- PERIODIC_X = 0
- PERIODIC_Y = 0
- PERIODIC_Z = 0
- DIMENSIONLESS = 1

Variable	**Type**	**Description**
in_file	char(256)	Input filename
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
rcloud	PR	Required radius of sphere
nwant	int	Required number of particles in sphere

3.17 ic_vel_pert.F90

Adds a variety of perturbations to any inputted (spherical) density distribution. Requires parameters file velpert.dat.

Required Makefile options :

- NDIM = 3
- DIMENSIONLESS = 0

Variable	Type	Description
in_file	char(256)	Input filename
in_file_form	char(256)	Input file format
out_file	char(256)	Output filename
out_file_form	char(256)	Output file format
densmode	char(20)	Mode of density perturbation (not used yet)
amp	PR	Amplitude of azimuthal perturbation (not used yet)
mpert	integer	Azimuthal perturbation mode (not used yet)
fenhance	PR	Density enhancement factor (increase all particle masses by fenhance; used to make stable polytropes unstable)
vpower	PR	Turbulent velocity power spectrum index
eturb	PR	Ratio of turbulent to gravitational energy
ngrid	integer	No. of grid points for vel field (determines resolution of velocity field; must be a multiple of 2)
iseed1	integer	Random No. seed 1
iseed2	integer	Random No. seed 2
velradmode	char(20)	Radial velocity mode (energy, dvdr or none)
dvdr	PR	Radial velocity gradient
erad	PR	Ratio of radial kinetic to gravitational energy
velrotmode	char(20)	Rotational mode (energy, angmom, angvel or none)
angmom	PR	Total angular momentum (if velrotmode = angmom)
angmomunit	char(256)	angular momentum unit
angpower	PR	Angular velocity power law (angular velocity is a function of axial distance, $\omega \propto r^{angpower}$)
angvel	PR	Angular velocity
angvelunit	char(256)	Angular velocity unit
erot	PR	Ratio of rotational kinetic energy to gravitational energy

4 Running the SEREN bash test script

SEREN contains a bash script designed to run batches of tests of SEREN for development and debugging purposes. The script, and all related files for running the tests, is located in the /seren/testsuite sub-directory. In the testsuite directory, there is the test-seren.sh bash script and further sub-directories which contain files used by test-seren.sh when performing batch tests.

A test is launched from the command line as in the following example :

```
./test-seren.sh -gfortran -openmp -debug1 -test POLYRAD1-AB
```

The current list of command line options for the script are (**TBD**) :
The list of tests currently set-up for use with the test script are (**TBD**) :

Table 5: List of automated tests in SEREN

Test name	**Description**
ADSOD-3D	Classic SOD test of two initially static columns of gas in contact which then interact forming a shock. Gas is non-radiative so the energy equation is solved and no energy escapes the system (i.e. it is adiabatic).
BURRAU1	Burrau 3-body problem (Burrau 19??); also known as the Pythagorean problem. Three stars with masses 3, 4 and 5 placed on the corners of a right-angled triangle all with zero-velocity and allowed to evolve until the system dissolves into a single star and a binary star.
COL-3D	Two columns of uniform density gas collide supersonically to produce a dense shocked layer.
EIGEN1	Gravitational force accuracy using eigenvalue MAC
FIGURE8	Figure-8 3-body test for N-body integrator (????).
FREEFALL1	Free-fall collapse test.
GEO1	Gravitational force accuracy using geometric MAC
ISOFREEFALL1	Isothermal free-fall collapse test
KH-2D	2D Kelvin-Helmholtz instability test
NTSI-2D	2D Non-linear thin shell instability test
POLYRAD1	Masunaga & Inutsuka (????) collapse test
SEDOV-3D	Sedov blast wave test (Sedov 19??).
SHEAR-2D	2-D shearing layer test.
SIT1-AB	A variation of the Boss-Bodenheimer (1979) test. A uniform-density spherical cloud is given a sinusoidal azimuthal density perturbation and a solid-body rotationaal velocity field such that it collapses to form a dense filament with a star on each end and eventually bound binary system.
STATPOLY1	Relax a polytropic gas to hydrostatic balance.

5 Coding style of SEREN

5.1 Design philosophy of SEREN

SEREN is a highly modular code written in Fortran 90 which comprises of several layers of subroutine calls in performing basic simulations. Each subroutine is designed to perform one single task. If a long procedure consists of a number of independent steps (i.e. not using the same local variables), then it is broken down into a sequence of smaller subroutines. Also, each .F90 file contains one single subroutine (with the exception of sanitycheck.F90 which has two extra smaller subroutines for clarity).

For the benefit of anyone reading through the source code, or for those wishing to develop new routines, we discuss here in detail some of the more important coding conventions that are used in SEREN. We do not discuss the particular features of any one subroutine (since each routine is extensively commented), but focus on the style used in most subroutines of SEREN.

5.2 Macros

SEREN uses C-like macros throughout the source code, both for the clarity (by reducing the number of lines) and the efficiency and runtime speed of the code. Macros are strings (conventionally in upper case as in C) which are substituted for some user-defined value or expression by the *pre-processor*, i.e. before the compiler generates machine code from the source code. This can improve the runtime performance somewhat by removing common variable references.

Macros are defined in two separate locations in SEREN. Some are defined in the Makefile (e.g. NDIM). Most macros however are defined in the header file /headers/macros.h. In order to make use of the macros, we must import the file /headers/macros.h into the subroutine by way of the pre-processor command #include "macros.h". The majority of macros in SEREN are straight-forward numerical substitutions of important information, such as array sizes or physical constants.

5.2.1 Function-like macros

Function-like macros are macros that look like functions/subroutines by their syntax, but work by the substitution of a string of commands, rather than calling a subroutine elsewhere in memory (thereby eliminating the extra cost associated with a subroutine call). In SEREN, we use function-like macros as a compact and concise way of writing debugging information to the screen when in debug mode. For example, we define the debug1 macro in the following way.

```
#ifdef DEBUG1
    #define debug1(x) write (6,*) x
#else
    #define debug1(x)
#endif
```

If we wished to write debug information to screen (e.g. in order to indicate the current location in the code), we could write in long-hand:

```
#ifdef DEBUG1
    write(6,*) "Calculating smoothing lengths"
#endif
```

In SEREN, we can instead write the short-hand form

```
debug1("Calculating smoothing lengths")
```

If the DEBUG1 compiler flag is defined in the Makefile, then the debug1() macro is replaced with write(6,*) "Calculating smoothing lengths". If DEBUG1 is not defined in the Makefile, then debug1() macro is replaced with nothing. For subroutines (particularly those in development) that contain many debugging statements, these macros allow us to write code with more clarity and fewer lines. We use four levels of debug macros, which are all defined in /headers/macros.h.

5.3 Real variable types

Rather than hard-wiring in the precision of real variables in the source code, SEREN allows the user to specify the precision through one of the options in the Makefile (PRECISION). The precision is controlled by several lines in the module definitions (in modules.F90) :

```
    integer, parameter :: DP = selected_real_kind(p=15)
    integer, parameter :: SP = selected_real_kind(p=6)
#if defined(DOUBLE_PRECISION)
    integer, parameter :: PR = DP
#else
    integer, parameter :: PR = SP
#endif
```

The first two lines use the intrinsic selected_real_kind function to define the precision independent of the processor type (i.e. whether it is 32-bit or 64-bit). The conditional compilation section then defines the precision used in the code (i.e. PR) depending on the option selected in the Makefile. Any real variable in the code must be defined in the following way, e.g.

```
    real(kind=PR) :: drmag
```

If we wish to declare a double or single precision variable irrespective of the general precision (e.g. any summation variables in /main/diagnostics.F90 always use double precision), then we use DP or SP in place of PR.

If we wish to convert a variable to a real variable of required precision, we must specify the kind (i.e. PR, DP or SP) as a second parameter in the real function, e.g. to convert the integer variable i to a real variable of precision PR, we write

```
    ireal = real(i,PR)
```

5.4 Particle data arrays

SEREN mainly uses simple arrays to store particle data. However, data which are important in gravity calculations are stored differently. The position, mass and smoothing length information are grouped together in a single array, parray(1:NDIM+2,1:ptot). The position of particle p is stored in the elements parray(1:NDIM,p), the mass is stored in the element parray(MASS,p), and the smoothing length is stored in the element parray(SMOO,p) (See /seren/headers/macros.h for macro definitions).

5.5 Particle types

SEREN accomodates the following particle types:

- Static boundary particles (pboundary)
- Non-gravitating inter-cloud medium (IcM) particles (picm)
- Self-gravitating gas particles (pgas)
- Dark-matter particles (pcdm)
- Dust particles (pdust)
- Ion particles (pion)

- Sink particles (stot)

where the variable names indicate the number of each particle type present in the simulation. All data for the first three (boundary, IcM and self-gravitating gas particles) are stored in the main arrays, which contain ptot elements where $\mathrm{ptot} = \mathrm{pboundary} + \mathrm{picm} + \mathrm{pgas} + \mathrm{pcdm} + \mathrm{pdust} + \mathrm{pdust}$. The data is stored such that the first $\mathrm{pboundary}$ elements contain the information for boundary particles, the next picm elements contain the information for the IcM particles, and the next pgas elements contain the information for the gas particles, and the final pcdm elements contain the information for the cdm particles. Although provision has been made for their use in future versions of SEREN, dust and ion particles are not currently active.

The sink particles are stored in separate data structures, since they can have many additional properties that are not possessed by normal SPH particles and thus require their own data structures. We use Fortran types (equivalent to C structures) to hold sink data. The main array that contains each sink structure is called $\mathrm{sinkdata}$ and elements can be accessed using the Fortran % notation (e.g. the mass element of sink s is $\mathrm{sinkdata(s)\%m}$).

6 Units

Dimensionless units are used in numerical simulations so that all values are as close to unity as possible, to avoid having very large or very small values that may result in significant rounding errors. SEREN contains a flexible system of units which allows the user to select from a wide range of commonly used astrophysical units, or easily construct their own set of units. All variables related to units and scaling are determined in units.F90. Each quantity, X, has four scaling variables associated with it: Xunit, Xscale, X_SI and Xcgs.

- Xunit is a string which contains the name of the unit that the quantity X is measured in; e.g. for length units, runit may take the values pc, au, r_sun, etc. All Xunit strings are defined in the parameters file. The available options in version 1.0 of the code are given in the following table:
- Xscale is a real variable that allows us to convert between physical and code units. In order to convert any variable from physical to code units (where the physical variable is measured in units specified by Xunit), then we divide the physical unit by Xscale. Conversely, to convert any code variable to physical units, we multiply the code value by Xscale
- X_SI is a real variable that allows us to convert between the unit specified by Xunit and S.I. units. In order to convert from Xunit to S.I. units, we multiply the variable (in units of Xunit) by X_SI.
- Xcgs is a real variable that allows us to convert between the unit specified by Xunit and cgs units. In a similar way to converting to S.I. units, in order to convert from Xunit to cgs units, we multiply the variable (in units of Xunit) by Xcgs.

In a self-gravitating code like SEREN, we choose a set of units so as to make the gravitational constant G equal to unity. We are free to choose the values of rscale and mscale. The value of tscale is then set to ensure G $= 1$ in the new system of units. Therefore, the value of tscale can be obtained using

$$tscale \times t\_SI = \frac{(rscale \times r\_SI)^{3/2}}{\sqrt{G \times mscale \times m\_SI}} \tag{1}$$

where G is the gravitational constant in c.g.s. units, i.e. $\mathrm{G} = 6.67 \times 10^{-8}\,\mathrm{cm^3\,g^{-1}\,s^{-2}}$. All other scaling factors can be determined in a similar way using:

$$vscale \times v\_SI = \frac{rscale \times r\_SI}{tscale \times t\_SI} \tag{2}$$

$$ascale \times a\_SI = \frac{rscale \times r\_SI}{(tscale \times t\_SI)^2} \tag{3}$$

$$rhoscale \times rho\_SI = \frac{mscale \times m\_SI}{(rscale \times r\_SI)^3} \tag{4}$$

$$sigmascale \times sigma\_SI = \frac{mscale \times m\_SI}{(rscale \times r\_SI)^2} \tag{5}$$

$$Pscale \times P\_SI = \frac{mscale \times m\_SI}{rscale \times r\_SI \times (tscale \times t\_SI)^2} \tag{6}$$

$$fscale \times f\_SI = \frac{mscale \times m\_SI \times rscale \times r\_SI}{(tscale \times t\_SI)^2} \tag{7}$$

$$Escale \times E\_SI = \frac{mscale \times m\_SI \times rscale \times r\_SI}{(tscale \times t\_SI)^2} \tag{8}$$

$$momscale \times mom\_SI = \frac{mscale \times m\_SI \times rscale \times r\_SI}{tscale \times t\_SI} \tag{9}$$

$$angmomscale \times angmom\_SI = \frac{mscale \times m\_SI \times rscale^2 \times r\_SI^2}{tscale \times t\_SI} \tag{10}$$

$$dmdtscale \times dmdt\_SI = \frac{mscale \times m\_SI}{tscale \times t\_SI} \tag{11}$$

$$Lscale \times L\_SI = \frac{Escale \times E\_SI}{tscale \times t\_SI} \tag{12}$$

$$kappascale \times kappa\_SI = \frac{(rscale \times r\_SI)^2}{mscale \times m\_SI} \tag{13}$$

In MHD, we must also introduce the unit of charge and associated units such as magnetic field and current density. As with gravitational problems, we can scale the units of the magnetic field such that the permiability of free space, μ_0, is equal to unity. **to be completed**.

$$Jscale \times J\_SI = \frac{Qscale \times Q\_SI}{tscale \times t\_SI \times rscale^2 \times r\_SI^2} \tag{14}$$

Table 6: List of unit options in SEREN

Xunit	**Options**	**Description**
runit	mpc	megaparsecs
	kpc	kiloparsecs
	pc	parsecs
	au	astronomical units
	r_sun	solar radii
	r_earth	Earth radii
	km	kilometres
	m	metres
	cm	centimetres
munit	m_sun	solar masses
	m_jup	Jupiter masses
	m_earth	Earth masses
	kg	kilograms
	g	grams
tunit	gyr	gigayears
	myr	megayears
	yr	years
	day	days
	sec	seconds
vunit	km_s	kilometres per second
	au_yr	astronomical units per year
	m_s	metres per second
	cm_s	centimetres per second
aunit	km_s2	kilometres per second squared
	au_yr2	astronomical units per year squared
	m_s2	metres per second squared
	cm_s2	centimetres per second squared
rhounit	m_sun_pc3	solar masses per cubic parsec
	g_cm3	grams per cubic centimetre
sigmaunit	m_sun_pc2	solar masses per parsec squared
	g_cm2	grams per centimetre squared
Punit	Pa	pascals
	bar	bars

Xunit	Options	Description
	g_cms2	grams per centimetre per second squared
funit	N dyn	newtons dynes
Eunit	J erg GJ	joules ergs gigajoules
momunit	m_sunkm_s m_sunau_yr kgm_s	solar masses kilometres per second solar masses astronomical units per year kilomgram metres per second
angmomunit	kgm2_s gcm2_s	kilogram metres squared per second gram centimetres squared per second
angvelunit	rad_s	radians per second
dmdtunit	m_sun_myr m_sun_yr kg_s g_s	solar masses per megayear solar masses per year kilograms per second grams per second
Lunit	L_sun	solar luminosities
kappaunit	m2_kg cm_g	metre squared per kilogram centimetre per gram
Bunit	tesla gauss	tesla gauss
Qunit	C	coulombs
Junit	C_m2_s	coulombs per second per metre squared
uunit	J_kg erg_g	Joules per kilogram ergs per gramme
dudtunit	J_kg_s	Joules per kilogram per second
tempunit	K	kelvin

7 File formats

SEREN 1.5.1 uses both the DRAGON file format and the native SEREN file format for reading in initial conditions and writing out snapshots. Unlike in DRAGON, the format of the initial conditions file need not be the same as the format of the output snapshots. This is controlled by the two input parameters in the parameters file, in_file_form and out_file_form. The possible values for these parameters are :

- ascii - Simple (ASCII) column format
- dragon_form - Formatted (ASCII) DRAGON snapshot files
- dragon_unform - Unformatted (binary) DRAGON snapshot files
- seren_form - Formatted (ASCII) SEREN snapshot files
- seren_unform - Unformatted (binary) SEREN snapshot files (Not yet working)

As well as standard snapshot files, SEREN can also produce a simple ASCII output which is useful for debugging purposes. This can be enabled by using the -DDEBUG_PLOT_DATA compiler flag.

7.1 ASCII format

Seren can use a simple flexible ASCII column-format. The data is stored in columns with width $N_{\rm COLUMNS}$ and length N (where N is the total number of particles of all types). The data descriptor of each column is contained in a file labelled asciicolumns.dat (a template copy should be stored in the /datafiles sub-directory). The possible data descriptors currently enabled in SEREN are

- ptype - Particle type. The following particle types are available in SEREN :
  - -1 : sink/star
  - 0 : dead particle
  - 1 : gas
  - 6 : boundary particle
  - 9 : ICM particle
  - 10 : cold-dark matter particle
- x or y or z - Cartesian position coordinates
- vx or vy or vz - Cartesian velocity components
- h - Smoothing length
- m - Mass
- u - Specific internal energy
- temp - Temperature (in K)

The only constraint on the column order is that the first column must be ptype. Thereafter, the remaining columns can be arranged in any order. In the file containing the data, the data must match up to the chosen columns correctly, or the particle data will be read-in incorrectly. All physical quantities are measured in the units defined in the params.dat file. Due to the simplicity of this format, it contains no extra information (e.g. time, gamma, etc.), and therefore is perhaps not of long-term practical use, but should be suitable for those who wish to generate their own initial conditions from other programmes without learning all the complications of the other available formats.

7.2 Dragon format

To be written

7.3 Seren format

To be written

8 Structure of code

8.1 Basic directory structure

Subroutines in SEREN are not all contained in a single source directory, but are grouped in several sub-directories depending on their purpose. In version 1.0, the following sub-directories exist :

- /seren/src/advance - integration routines
- /seren/src/analyse - analysis routines
- /seren/src/BHtree - Barnes-Hut octal tree subroutines
- /seren/src/binarytree - Binary-number tree subroutines
- /seren/datafiles - Contains initial conditions parameters files
- /seren/docs - Contains latest version of the userguide
- /seren/src/gravity - subroutines that calculate gravitational forces
- /seren/src/headers - macro and modules files
- /seren/src/healpix - HEALPix ioniaztion routines
- /seren/src/ic - programs to generate initial conditions for regularly used configurations (e.g. relaxed rectangular cubes, spheres)
- /seren/src/io - subroutines that read and write files
- /seren/src/main - contains important and commonly used subroutines
- /seren/src/mhd - contains magneto-hydrodynamics routines
- /seren/src/nbody - N-body routines
- /seren/src/nbody_sim - N-body simulation subroutines
- /seren/src/nbody_sph_sim - Hybrid N-body/SPH simulation routines
- /seren/src/radiation - contains radiation transfer subroutines
- /seren/src/setup - contains subroutines called during initialization of SEREN .
- /seren/src/sinks - subroutines that search for, create and advance sink particles.
- /seren/src/sorts - subroutines for sorting lists
- /seren/src/sph - subroutines that perform important SPH functions
- /seren/src/sph_sim - SPH simulation routines (e.g. h-finding, neighbour searching)
- /seren/src/tests - test programs
- /seren/src/timestep - timestepping routines
- /seren/testsuite - bash script for running batch tests of seren

9 Variable conventions

In SEREN, the names of all commonly used local variables are kept as consistent as possible between different subroutines. Here we list the names of all common local integer and real variables.

9.1 Integer variables

c	Cell identifier
cc	Child cell identifier
i	Auxiliary counter (often used when looping over neighbour lists)
k	Dimension counter
l	Level counter (for BH tree and HEALPix)
p	Particle identifier
pp	Neighbour identifier
pp_pot	No. of potential neighbours (e.g. after a tree walk)
pp_templist(1:pp_limit)	Temporary list of neighbour identifiers
pp_tot	Total number of neighbours for particle p
s	Sink particle identifier
ss	Secondary sink counter

9.2 Real variables

dr(1:NDIM)	Relative position vector
drmag	Distance
drsqd	Distance squared
dr_unit(1:NDIM)	Unit position vector
hp	Smoothing length of particle p
hpp	Smoothing length of neighbouring particle pp
mp	Mass of particle p
mpp	Mass of neighbouring particle pp
ms	Mass of sink particle s
invdrmag	Reciprocal of distance, i.e. 1 / drmag
invdrsqd	Reciprocal of distance squared, i.e. 1 / drsqd
invhp	Reciprical of smoothing length of p, i.e. 1 / hp
invhpp	Reciprical of smoothing length of pp, i.e. 1 / hpp
qc(1:NQUAD)	Quadrupole moment tensor for cell c
rp(1:NDIM)	Position of particle p
rpp(1:NDIM)	Position of neighbouring particle pp
rs(1:NDIM)	Position of sink particle s
sound_p	Sound speed of particle p
sound_pp	Sound speed of neighbouring particle pp
up	Specific internal energy of particle p
upp	Specific internal energy of neighbouring particle pp
vp(1:NDIM)	Velocity of particle p
vpp(1:NDIM)	Velocity of neighbouring particle pp
vs(1:NDIM)	Velocity of sink particle s