SPHysics GUIDE - Chapter 5.
HOW TO CHANGE SPHysics FOR YOUR APPLICATION

Introduction

When people start using the SPHysics code, we often get asked if the code can do a particular function that is not included in the demonstration cases. The answer we give is normally yes, but the particular functionality required may require some re-coding. We do not normally propose to do this re-coding ourselves unless the application area coincides closely with our own area and current projects, or there is a bug. The reason behind this is that SPHysics is primarily a research code and we have released what we have found useful for our own research. As the code is research oriented, it is up to the user to adapt the code and the subroutines to their satisfaction.

This short section is aimed at helping those people who want to change the code for their own purposes. Figure 5.1 displays the main structure of the code. Here, we list which subroutines in the code you should examine for possible modification. **Important Note**: if you create any new subroutines for the main source code, you must include the names of these new files in the “make files” used for compiling the code which are written in subroutines `tocompile_win_ifort`, `tocompile_ftn95`, `tocompile_gfortran`, `tocompile_ifort` in `SPHYSICSgen_2D/3D.f`. Read Section 3.2.2.3 to see where each of the subroutines is compiled.

1. **Changing the motion of moving objects (forced motion)**
   `movingObjects_2D/3D.f` controls the calling of `movingGate_2D/3D.f`, `movingPaddle_2D/3D.f`, `movingWedge_2D/3D.f` and `rigid_body_motion_2D/3D.f`. If the motion you desire is not covered by these subroutines, then you must create your own.
2. **Changing the boundary conditions.**
   Boundary conditions are treated in each `celij` & `self` subroutines. Any modification to the boundary conditions should be done in these subroutines.
3. **Changing the timestepping algorithm**
   The timestepping is performed in all of the step subroutines: `step_predictor_corrector_2D/3D.f`, `step_verlet_2D/3D.f`, `step_symplectic_ 2D/3D.f`, `step_Beeman_2D/3D.f`. These subroutines then call subroutines ac which control the sweep across the particles (or 2h grid) for each (part of the) timestep.
4. **Changing the kernel calculation**
   The smoothing kernel and its derivatives are calculated in the kernel subroutines: `kernel_gaussian_2D/3D.f`, `kernel_quadratic_2D/3D.f`, `kernel_cubic_2D/3D.f and kernel_Wendland_2D/3D.f`. In version 2.0 of SPHysics, these can now optionally be corrected for lack of complete support in subroutine `kernel_correction` (see Section 1.9).
5. **Changing the viscous formulation**
   The viscous terms are all calculated in the viscosity subroutines which are called

from `celij` & `self`: `viscosity_artificial_2D/3D.f`, `viscosity_laminar_2D/3D.f`, `viscosity_laminar+SPS_2D/3D.f`. For the SPS turbulence model, the shear stresses are zeroed in subroutine ac and then defined for the next timestep in subroutine `correct_SPS_2D/3D.f`.

6. **Loading in data files and setting useful parameters**
   If you wish to examine and modify what data SPHysics loads initially, all the useful data is imported in subroutine `getdata_2D/3D.f` Furthermore, all the useful parameters that remain the same throughout the simulation are calculated here such as the kernel normalization factors, etc. All global variables are stored in the common blocks contained in `common.2D/3D`.
7. **Zeroing variables**
   Many variables that are evaluated throughout the timestep, such as the accelerations, `ax, ay, az` are zeroed initially in the different ac subroutines: `ac_2D/3D.f`, `ac_Conservative_2D/3D.f`, `ac_Shepard_2D/3D.f`, `ac_MLS_2D/3D.f`, `ac_KGC_2D/3D.f`, `ac_KC_2D/3D.f`.
8. **Changing the input geometry**
   At present, SPHysics is limited to generating a few simple geometric structures both in 2-D and 3-D such as boxes, planar beaches, triangular moving wedges, square floating objects. Generating the geometry is controlled by the code `SPHYSICSgen_2D/3D.f`. As explained in Section 3.2, the input case files can be used to generate a mixture of these basic options. If you wish to modify or add new options, you will need to edit and modify `SPHYSICSgen_2D/3D.f`. Here, we try to give you some indications which subroutines to change:
   (i) main geometric container shape: `box, beach`
   (ii) static obstacles: `trapezoid, wall, obstacles`
   (iii) filling the particles: `fill_part` (& maybe `fluid_particles`)
   (iv) forced motion objects: `gate, wavemaker, RaichlenWedge_Particles, fill_part`
   (v) free-motion objects (floating): `FloatingBody_Particles, fill_part`

As described in Section 4.10, a complex geometry generator is now provided for 3-D applications (see http://wiki.manchester.ac.uk/sphysics/index.php/Contributors) making the creation of new geometries and loading in CAD files more accessible.

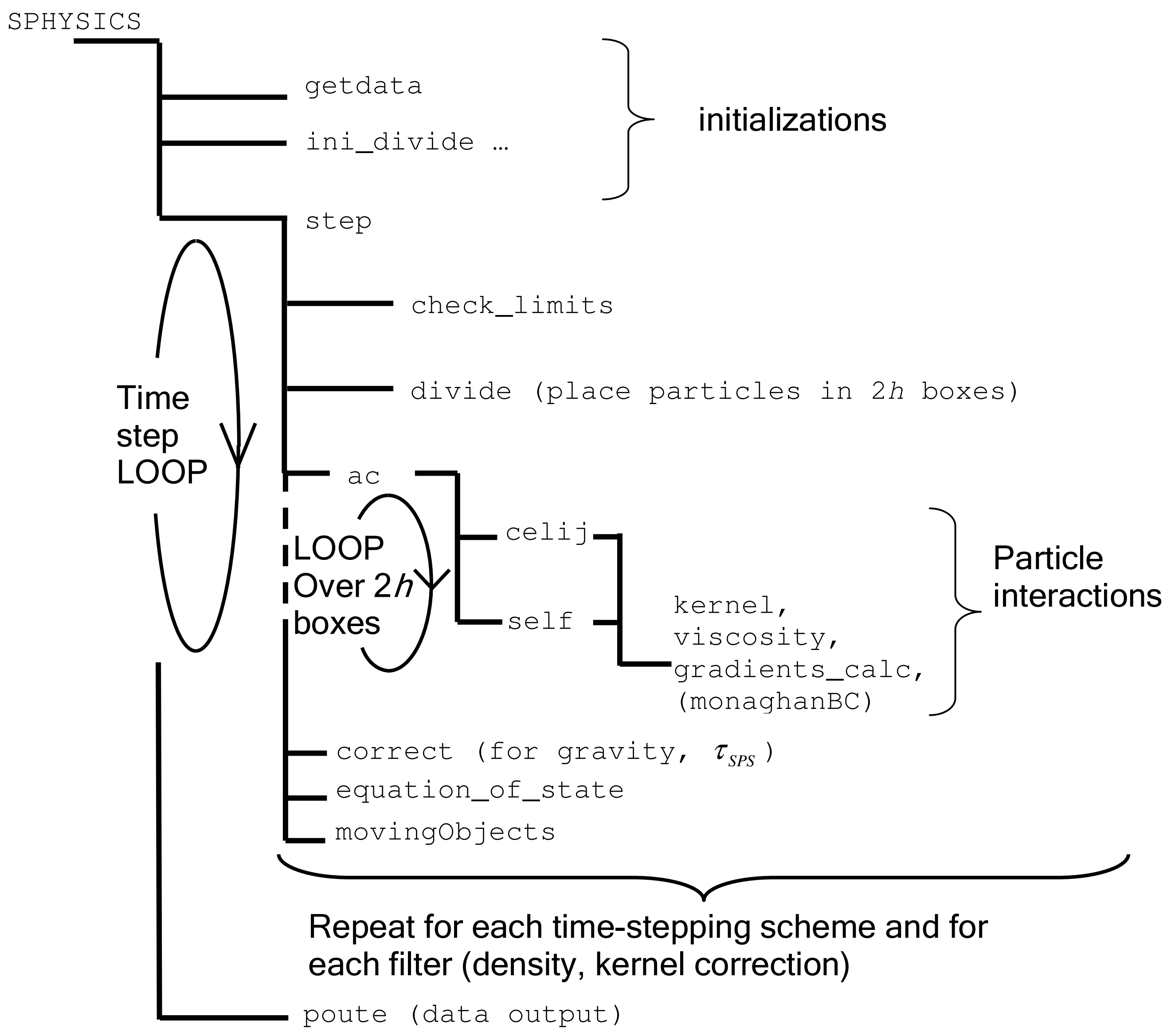
SPHYSICS
getdata
ini_divide …
initializations
step
Time step LOOP
check_limits
divide (place particles in 2h boxes)
ac
LOOP Over 2h boxes
celij
self
kernel, viscosity, gradients_calc, (monaghanBC)
Particle interactions
correct (for gravity, τ_SPS)
equation_of_state
movingObjects
Repeat for each time-stepping scheme and for each filter (density, kernel correction)
poute (data output)

Figure 5.1 Outline of code structure

Here we present a table of the main variables (or those with less than obvious names) and the counterpart in equations:

SPHysics variable	**SPH quantity**
ax(i), ay(i), az(i)	$\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t} = \frac{\mathrm{d}\vec{v}_a}{\mathrm{d}t}$
ar(i)	$\frac{\mathrm{d}\rho_a}{\mathrm{d}t}$
aTE(i)	$\frac{\mathrm{d}e_a}{\mathrm{d}t}$
cbar	$\overline{c_{ab}} = (c_a + c_b)/2$

cs(i)	c_a

deltaptb(j,1), deltaptb(j,2)
 	
deltapsb(j,1), deltapsb(j,2)
	$\vec{t}_b \cdot (\vec{r}_{b+1} - \vec{r}_b)/\lvert\vec{r}_{b+1} - \vec{r}_b\rvert$ & $\vec{t}_b \cdot (\vec{r}_b - \vec{r}_{b-1})/\lvert\vec{r}_b - \vec{r}_{b-1}\rvert$
	$\vec{s}_b \cdot (\vec{r}_{b+1} - \vec{r}_b)/\lvert\vec{r}_{b+1} - \vec{r}_b\rvert$ & $\vec{s}_b \cdot (\vec{r}_b - \vec{r}_{b-1})/\lvert\vec{r}_b - \vec{r}_{b-1}\rvert$
	(for boundary particles only)

dudx_CSPH(i), dudy_CSPH(i), …	$\frac{\partial u_a}{\partial x_a}, \frac{\partial u_a}{\partial y_a}, \ldots$
drx, dry, drz	$x_{ab} = (x_a - x_b),\ y_{ab} = (y_a - y_b),\ z_{ab} = (z_a - z_b)$
duxp, duyp, duzp	$u_{ab} = (u_a - u_b),\ v_{ab} = (v_a - v_b),\ w_{ab} = (w_a - w_b)$
frxi, frxj, frzi, frzj	$\frac{\partial W_{ab}}{\partial x_a}, \frac{\partial W_{ba}}{\partial x_b}, \frac{\partial W_{ab}}{\partial z_a}, \frac{\partial W_{ba}}{\partial z_b}$
fxbp, fybp, fzbp	Boundary forces $\mathbf{f} = (f_x, f_y, f_z)\mathbf{n}$
pm(j)	m_b
pr(i)	$\frac{P_a}{\rho_a^2}$
pVol(j)	$V_b = \frac{m_b}{\rho_b}$
p_v	$\left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} + \Pi_{ab}\right)$ or $\left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} + \Pi_{ab} + Rf_{ab}\right)$
rhop(i)	ρ_a
rhop_sum	$\sum_b \rho_b W_{ab} \frac{m_b}{\rho_b} = \sum_b m_b W_{ab}$
rr2	r_{ij}^2
sum_wab	$\sum_b W_{ab} \frac{m_b}{\rho_b}$
up(i), vp(i), wp(i)	$\mathbf{v}_a = \vec{v}_a = (u_a, v_a, w_a)$
ux(i), vx(i), wx(i)	$\sum_b \frac{m_b}{\overline{\rho}_{ab}} \vec{v}_{ba} W_{ab}$ (XSPH correction)
xnb(j), ynb(j), znb(j)	$\mathbf{n}_b = \vec{n}_b = (n_x, n_y, n_z)_b$ (Boundary normals)
xsb(j), ysb(j), zsb(j)	$\mathbf{s}_b = \vec{s}_b = (s_x, s_y, s_z)_b$ (Boundary tangent **s**)
xtb(j), ytb(j), ztb(j)	$\mathbf{t}_b = \vec{t}_b = (t_x, t_y, t_z)_b$ (Boundary tangent **t**)
xp(i), yp(i), zp(i)	$\mathbf{r}_a = \vec{r}_a = (x_a, y_a, z_a)$
Wab	$W_{ab} = W(\lvert\vec{r}_a - \vec{r}_b\rvert)$
bigUdot, bigVdot, bigWdot, OmegaX/Y/Zdot, bigMass	$\frac{\mathrm{d}\mathbf{V}}{\mathrm{d}t}$, Ω, M (for rigid body dynamics, eq 2.21)
bigU, BigV, bigW, bigOmegaX	$\mathbf{V} = \vec{V} = (U, V, W)$, $\mathbf{\Omega} = (\Omega_x, \Omega_y, \Omega_z)$