Extensible, Reusable, and Reproducible Computing: A Case Study of PySPH

Prabhu Ramachandran

1 Department of Aerospace Engineering, IIT Bombay, Powai, Mumbai, India 400076
E-mail: prabhu@aero.iitb.ac.in

Abstract.
In this work, the Smoothed Particle Hydrodynamics (SPH) technique is considered as an example of a typical computational research area. PySPH is an open source framework for SPH computations. PySPH is designed to be easy to use. The framework allows a user to implement an entire simulation in pure Python. It is designed to make it easy for scientists to reuse their code and extend the work of others. These important features allow PySPH to facilitate reproducible computational research. Based on the experience with PySPH, general recommendations are suggested for other computational researchers.

Keywords: SPH, Reproducible Computing, Scientific Computing

1. Introduction
Reproducibility and the ability to use and extend the research of others has always been a cornerstone of scientific research. However, even a cursory sampling of most modern publications in computational research will show that this does not hold true in practice. Research papers are often hard to extend, reuse, or reproduce. Results may not be reproducible either because some parameters are mistyped or not specified at all. The authors cannot be blamed entirely as it is easy to overlook some details of a particular computation. There is little interest or incentive in making computational results reproducible. There are several strategies to overcome these but they typically require a fair bit of effort and some knowledge of software engineering. Donoho et al. [1] provide an overview of reproducible computing along with several potential benefits and objections to its adoption by the scientific computing community.

Many researchers are not experts in software engineering, yet high performance code is also desirable. It would be ideal to have an environment where the user of a tool could write high-level code that is numerically well-behaved, mathematically consistent, and representative of the physics without worrying about high-performance or requiring expertise in software engineering principles.

In this paper we consider the Smoothed Particle Hydrodynamics (SPH) technique as an example of a typical computational research area. The SPH method was first proposed by Gingold and Monaghan [2] and Lucy [3] to simulate astrophysical problems. The method employs a Lagrangian formulation for the continuum equations and discretizes field properties into particles carrying them. This results in a Lagrangian particle method that does not require a mesh. This makes the method very general purpose. Since its initial formulation in astrophysics, it has been adapted to simulate a wide variety of problems including incompressible free-surface fluid flow [4] and solid mechanics [5]. An excellent review of SPH and its various applications is provided by Monaghan [6].

While there are some open source implementations of SPH available, they are not very easy to extend and are often tailor-made for one particular domain. In the astrophysics community GADGET-2 [7]
is one such open source package. However, this is designed only for astrophysical simulations and is implemented in C. While C is a popular language, it is more difficult than a modern programming language like Python. Modifying the source code in order to extend it, requires a reasonable amount of experience. This makes extending such codes a little more difficult. SPHysics is an open source package but is designed for incompressible flows. JOSEPHINE [8] is another recent package but implemented in Fortran 90. These packages are not very easy to extend and are tailor-made for specific domains.

Over the last several years, my group has been working on an open source framework for SPH called PySPH (http://pysph.bitbucket.org). PySPH is implemented in Python which is a powerful, easy-to-learn programming language. PySPH automatically generates high performance code from high-level Python code. Good software development practices are used for testing the code on a regular basis on multiple platforms. Finally, simple tools are used to automate computational tasks. This can be used to facilitate easy reproduction of results by others. This requires additional effort. These practices make it easy to extend, reuse, and reproduce any results produced from the framework without sacrificing performance. Some of the general techniques and approaches that have been used to achieve this are discussed so as to make this useful to a general audience.

The following section provides a very brief introduction to the SPH method. Thereafter, some numerical implementation details are discussed. This is followed by an overview of PySPH with a discussion on how it has been made easy to use, reuse, and extend. The last section provides a set of general recommendations to other computational scientists.

2. Smoothed Particle Hydrodynamics

2.1. Mathematical details

Consider the identity,

\[ f(x) = \int_{-\infty}^{\infty} \delta(x - x') f(x') dx', \] (1)

where \( \delta(x) \) is the Dirac delta distribution. The SPH scheme approximates the Dirac delta using a kernel \( W(x) \) and this results in,

\[ f(x) \approx \int_{-\infty}^{\infty} W(x - x') f(x') dx'. \] (2)

The kernel, \( W \), is typically, compact, smooth, and symmetric. The kernel is parametrized on the parameter \( h \) which determines its spatial extent. In an SPH scheme, one represents the function \( f \) using a collection of \( N \) particles that together approximate the function. Each particle located at \( x_j \) has a functional variation determined by \( W(x - x_j) \) and a weight, \( f(x_j) \). The function can therefore be discretized as,

\[ f(x) \approx \langle f(x) \rangle = \sum_{j=1}^{N} W_h(x - x_j) f(x_j) \Delta x_j \] (3)

Clearly, since \( W \) is compact, the summation does not need to include all particles but only particles in the neighborhood of \( x \). This is written as,

\[ f(x) \approx \langle f(x) \rangle = \sum_{j \in \mathcal{N}(x)} W_h(x - x_j) f(x_j) \Delta x_j, \] (4)

where \( \mathcal{N}(x) \) is the set of neighboring particles at \( x \).

Given that the particles may move, \( \Delta x_j \) is not constant. An elegant way to estimate it, is to use the particle density. If a particle has mass, \( m_j \) and density \( \rho_j \), then one can write the SPH approximation of the function as,

\[ \langle f(x) \rangle = \sum_{j \in \mathcal{N}(x)} \frac{m_j}{\rho_j} W_h(x - x_j) f(x_j). \] (5)
Given that the functions can be represented with a collection of particles, one can easily define derivatives by using the derivative of the smooth kernel. For example, in one-dimension, one could write the derivative of the function at particle $i$ as,

$$\frac{\partial f_i}{\partial x_i} = \sum_{j \in N(x)} \frac{m_j}{\rho_j} \frac{\partial W_{ij}}{\partial x_i} \left( f_j - f_i \right).$$  \hspace{1cm} (6)

When solving a partial differential equation using SPH, a Lagrangian formulation is typically used. For example, for the conservation of mass, the governing equation is,

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \vec{v}. \hspace{1cm} (7)$$

With the SPH method, this equation would be written as,

$$\frac{d\rho_i}{dt} = -\rho_i \sum_{j \in N(x)} \frac{m_j}{\rho_j} \vec{v}_{ji} \cdot \nabla_i W_{ij}. \hspace{1cm} (8)$$

Using a similar approach, the governing differential equations for a variety of problems may be solved. The partial differential equations are thus converted to ordinary differential equations that may be integrated using a standard integrator. More details are available from the review by Monaghan [6].

2.2. Numerical details
Given the governing differential equations, one may discretize them as discussed in the previous section. This results in a set of coupled ordinary differential equations similar to that in equation (8). Given a particular differential equation that is suitably discretized using an SPH scheme, the typical steps involved in an SPH computation are as follows:

- Discretize the field variables into particles carrying the properties of the flow.
- For each differential equation, compute the right-hand sides (for example see equation (8)) as follows:
  - For each particle $i$, find the neighbors of the particle.
  - Compute the acceleration from each of the neighboring particles, $j$, on the particle, $i$.
- Integrate the differential equations.

From the above it can be seen that there are a few key aspects in an SPH computation:

(i) Find nearest neighbors of a given particle.
(ii) Find the interactions between individual particles.
(iii) Integrate the differential equations.

All SPH implementations allow researchers to specify these. The differences between SPH schemes mostly lies in how the interactions between particles is specified.

3. PySPH
As discussed in the previous section, it is important to be able to easily specify the interactions between particles and also specify how the equations of motion are to be integrated. In the following sections, some details of the PySPH implementation are discussed.

The PySPH framework allows a user to specify the interactions between particles and the integration schemes, using Python. Python [9] is a powerful and easy to use programming language. It is increasingly being used as the first introductory programming language in the top US universities [10]. It is also well known for its increasing popularity with scientists [11].
The use of Python makes it very easy to define the SPH interactions. However, Python is not designed for high-performance computing and one must use a low-level language for this. PySPH generates this code automatically from the high level Python code. PySPH also makes use of OpenMP and MPI to parallelize the code automatically. The user does not need to write any code to use the parallelization. Thus, PySPH allows a user to specify the interactions easily without sacrificing performance.

The goal of PySPH is to make it very easy for anyone to use and extend it to simulate a wide variety of SPH problems. This must be done without sacrificing correctness or performance. In order to do this, good software engineering practices must be employed. The next sections focus on a few key aspects: ease of use, accuracy and testing, reusability, and reproducibility.

3.1. Ease of use

There are several things that make PySPH easy to use.

- The use of Python makes it very easy to create new simulations and extend the existing framework.
- It is distributed under a liberal BSD license which allows it to be reused by anyone without restriction.
- The sources are version controlled using Git which allows others to easily use and contribute to the development of PySPH. The sources are currently available from [http://pysph.bitbucket.org](http://pysph.bitbucket.org).
- It is cross-platform and runs on Linux, OS X, and Windows.
- The documentation on how to install and use PySPH is available online at [http://pysph.readthedocs.org](http://pysph.readthedocs.org).
- When using PySPH to solve specific problems, once a user writes the Python code to solve the problem, no changes need to be made to use multiple CPU cores or multiple computers.
- About 30 useful examples are provided with PySPH. These examples can be easily run and modified by users.
- A built-in 3D viewer using Mayavi [12] for the solutions generated by PySPH is also provided. Fig. [1] provides a screenshot of the viewer.

In order to solve a typical problem with PySPH a user simply has to implement the following:

(i) Generate the initial particles with suitable properties.
(ii) Specify the inter-particle interactions, which are called “Equations” in PySPH.
(iii) Specify the integrator.
(iv) Perform any post-processing to produce results.

PySPH already implements several different SPH schemes for which the particle interaction equations are already defined. These simply need to be specified. Equations may be easily defined by the user if they are not already implemented. As a simple example, consider the continuity equation discretized in equation (8). In PySPH, one could write this equation in pure Python as follows,

```python
class ContinuityEquation(Equation):
    def initialize(self, d_idx, d_arho):
        d_arho[d_idx] = 0.0

    def loop(self, d_idx, d_arho, s_idx, s_m, VIJ, DWIJ):
        vijdotwij = DWIJ[0]*VIJ[0] + DWIJ[1]*VIJ[1] + DWIJ[2]*VIJ[2]
        d_arho[d_idx] += s_m[s_idx]*vijdotwij
```

XXVII IUPAP Conference on Computational Physics (CCP2015) IOP Publishing
Journal of Physics: Conference Series 759 (2016) 012094
doi:10.1088/1742-6596/759/1/012094
Figure 1. PySPH viewer showing a simulation of the dam-break past an obstacle.

Note that $DW_{IJ}$ is the gradient of the kernel and $VI_{J} = v_{i} - v_{j}$. These are automatically provided by PySPH.

Once the user has written the code to solve a problem, PySPH internally generates high-performance parallel code. This is then executed. The high-performance code is generated by the use of Cython which generates C++ code. This is automatically compiled and called from Python.

PySPH is much easier to use than a C/C++/F90 package since it does not require the user to explicitly compile anything. Users can simply run their Python scripts and the framework takes care of the rest. Extending an example is equally simple and a minimal amount of code needs to be written. We find that even undergraduate students with little experience are able to use the framework.

The approach of generating code makes it easy for users to perform SPH simulations. It also allows the PySPH developers to work on other high-performance backends transparent to the users. A GPU implementation is currently under development.

3.2. Accuracy and Testing

As an open source package that is released often, with a large number of features and an emphasis on ease of use, it is important that PySPH be free of errors and works correctly on multiple platforms. The project makes use of unit and integration tests to ensure this. There are about 250+ tests written for the project. These tests include unit tests that test the functionality of individual pieces of code as well as the entire framework. For example, one of the tests runs every example that ships with PySPH to ensure that the code successfully runs and the high-performance code is generated without error. Another test runs an example in serial and parallel, and ensures that the results are the same. The existence of these tests is extremely convenient as they immediately let the developers know if something is incorrect.

PySPH also makes use of continuous integration. This is a common software development practice used in many software projects. Every time the source code is changed and pushed to the PySPH source code repository, the entire test suite runs automatically on multiple platforms. The free services provided
by [http://shippable.com](http://shippable.com) and [http://codeship.com](http://codeship.com) are used to run the tests on Linux and [http://appveyor.com](http://appveyor.com) is used to run the tests on Windows. The tests are run on both Python 2.7.x and Python 3.4.x. Test result badges are displayed on the main page of the PySPH website. This makes it very easy for both the developers and users to see the current status of the software.

The combination of version control, online hosting, and continuous integration is very powerful, and allows us to develop reliable software relatively easily.

### 3.3. Reusability, and Reproducibility

As discussed above, by virtue of being written in Python and being freely available, PySPH enables reuse. Once a new set of equations is implemented in PySPH, it may be used by anyone.

In any area of numerical computation, there are a standard set of benchmark problems that are solved each time a new scheme is implemented. In PySPH, many standard benchmarks are implemented in such a way that a writer of a new scheme can easily reuse all the code in the existing example. For instance, if one considers the standard Lid-Driven-Cavity benchmark problem that is routinely solved (see [13] and [14]), PySPH provides a standard implementation and also includes all the post-processing to compare the results with well-established results. If a new SPH scheme is created and one wishes to compare the results with those of other implementations, one simply needs to subclass the standard example and change just the equations that are used. One does not need to re-write the initial particle distribution or the post-processing. This makes it extremely easy to compare the different schemes with minimum effort.

As can be seen, PySPH is not merely a package which is implemented in an easy to use programming language. It is also developed with the users and developers in mind, and strives to make it easier to reuse as much as possible. This is important as it allows anyone to verify the computational results produced. Furthermore, any improvements to the PySPH examples immediately benefit the entire community. This readily facilitates the comparison of schemes.

The reusability of PySPH naturally makes it easy to perform reproducible research in the area of SPH. By using PySPH, it is easy for other researchers to build on previous work. It is easier to develop new schemes as users can focus on the new scheme without having to implement the benchmark problems or worrying about performance of the scheme.

In any publication there are several computations performed. A single plot may involve tens or hundreds of simulations. In order to reproduce such results, it is imperative to make it easy for any researcher to re-run the simulations and generate the plots. A simple way to do this is to automate the generation of the plots in a publication. The use of Python and the design of PySPH make this easy to do. For a paper that is currently under preparation, we use all the above features of PySPH. In addition, we use other libraries in Python that facilitate automation. With this we are able to automate the generation of every figure in our publication. For example, we could run the following command:

```
$ python automate.py
```

inside our publication directory and this will run simulations for a few days and produce every single plot that is used in the paper. The details of this will be discussed in a future publication.

The features discussed above are individually not very significant, however, together they facilitate reproducible research in SPH, which has not been done before. It is my belief that the general principles that are behind the PySPH project may be applied to any software developed for computational science.

### 4. Concluding remarks

In this paper, the SPH method has been used as a prototypical computational scheme. The PySPH framework has been discussed with a focus on its ease of use, software engineering, and reuse. It has also been mentioned that with additional effort to automate results in publications, one can easily perform reproducible computational research.
Based on the lessons learned in the development of PySPH, I have the following general recommendations to others developing software for computational science:

- Make research codes and software open source with a convenient license.
- Use a modern version control system and host the software online to facilitate easy collaboration and contribution.
- Make every effort to make the package easy to use and install.
- Generate code for high-performance.
- Test the code regularly and use continuous integration services on multiple platforms.
- The examples are just as important as the source and it is a good idea to make the examples reusable, complete, and self-contained.
- Make it easy to automate the generation of results.
- Think about reuse and reproducibility when designing and writing code.

If these general recommendations are adopted more computational research can be made reproducible.

Acknowledgements
I thank Kunal Puri and other developers who have helped develop PySPH. I thank Kadambari Devarajan and an anonymous referee whose comments have helped improve this article.

References
[1] Donoho D L, Maleki A, Rahman I U, Shahram M and Stodden V 2009 Computing in Science and Engineering 11 8–18 ISSN 1521-9615
[2] Gingold R A and Monaghan J J 1977 Monthly Notices of the Royal Astronomical Society 181 375–389
[3] Lucy L B 1977 The Astronomical Journal 82 1013–1024
[4] Monaghan J J 1994 Journal of Computational Physics 110 399–406
[5] Gray J, Monaghan J J and Swift R 2001 Computer Methods in Applied Mechanics and Engineering 190 6641–6662
[6] Monaghan J J 2005 Reports on Progress in Physics 68 1703–1759
[7] Springel V 2005 Monthly Notices of the Royal Astronomical Society 364 1105–1134
[8] Cherfils J, Pinon G and Rivoalen E 2012 Computer Physics Communications 183 1468 – 1480 ISSN 0010-4655
[9] van Rossum G et al. 1991– The Python programming language http://www.python.org/
[10] Guo P 2014 Python is now the most popular introductory teaching language at top u.s. universities http://goo.gl/
[11] Perkel J M 2015 Nature 518 125–126
[12] Ramachandran P and Varoquaux G 2011 Computing in Science and Engineering 13 40–51
[13] Ghia U, Ghia K N and Shin C T 1982 Journal of Computational Physics 48 387–411
[14] Adami S, Hu X and Adams N 2013 Journal of Computational Physics 241 292–307 URL http://linkinghub.elsevier.com/retrieve/pii/S002199911300096X