The PENNANT Mini-App

Charles R. Ferenbaugh
Los Alamos National Laboratory
cferenba@lanl.gov

Version 0.4 – January 2013

PENNANT is an unstructured mesh physics mini-app designed for advanced architecture research. It contains mesh data structures and a few physics algorithms adapted from the LANL rad-hydro code FLAG, and gives a sample of the typical memory access patterns of FLAG.

1 Building and running the code

1.1 Building

A simple `Makefile` is provided in the top-level directory for building the code. Before using it, you may wish to edit the definitions of CXX and CXXFLAGS to specify your desired C++ compiler and flags, and to choose between optimized/debug and serial/OpenMP builds. Then a simple "`make`" command will create a `build` subdirectory and build the `pennant` binary in that directory.

PENNANT has been tested under GCC 4.6.1, PGI 11.10, and Intel 12.1.2. Building under other compilers should require only minor changes.

1.2 Running tests

Several test problems are provided in subdirectories under the `test` directory. The command line

`pennant` *testname*`.pnt`

is used to run a test.

The available test problems are listed in Table 1. The smaller problems run quickly and are useful for debugging and regression tests; gold standard files are provided (see next section). The larger tests take longer to run and are suitable for timing tests.

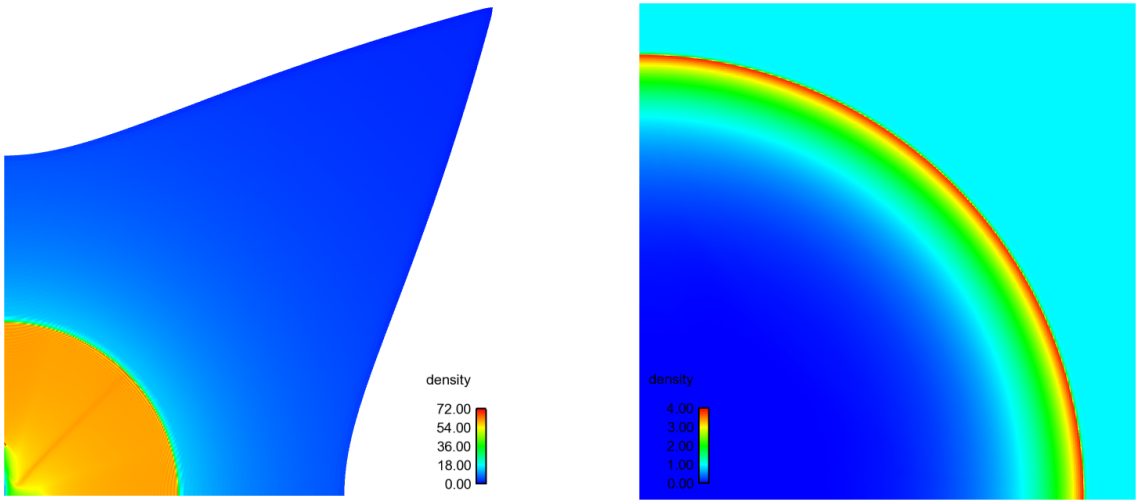
1.3 Test inputs and outputs

Each test problem directory contains two input files. The `.pnt` file is a small text file containing input parameters for the test. The `.gmv` file contains the input mesh in LANL GMV format; its name is specified as one of the entries in the `.pnt` file.

PENNANT generates output files of two kinds. The `.xy` output file is a text file containing the per-zone values of zone density, energy, and pressure. (It is modeled after a similar file generated by FLAG.) For the smaller tests, a gold standard file *testname*`.xy.std` is provided for reference.

Table 1: Test problems provided with PENNANT.

name	# zones	zone type
Leblanc problems [1]:		
leblanc	900	square
leblancbig	57600	square
Noh problems [5]:		
nohsmall	40	triangle/quad
nohsquare	32400	square
nohpoly	22801	mostly hexagons
Sedov problems [6]:		
sedovsmall	81	square
sedov	2025	square
sedovbig	72900	square



(a) (b)

Figure 1: Final state of (a) `nohsquare` and (b) `sedovbig` problems, colored by zone density.

There are also several graphics output files in Ensight Gold format: the main file has suffix `.case`, and it refers to auxiliary files with suffixes `.geo`, `.ze`, `.zp`, and `.zr`. These can be viewed by the proprietary Ensight[1] viewer, or by open-source viewers such as ParaView[2] and VisIt[3]. Sample outputs are shown in Figure 1.

For users who would like to generate additional test cases of varying sizes (e.g., for scaling studies), the utility script `gmvrect.py` is provided in the `tools` directory. This generates rectangular meshes containing square zones, with user-specified dimensions. Usage is:

```
Usage:  gmvrect.py NZX [NZY [LENX [LENY]]]
where nzx, nzy   = number of zones in x, y directions
```

[1] `http://www.ensight.com`
[2] `http://www.paraview.org`
[3] `https://wci.llnl.gov/codes/visit/home.html`

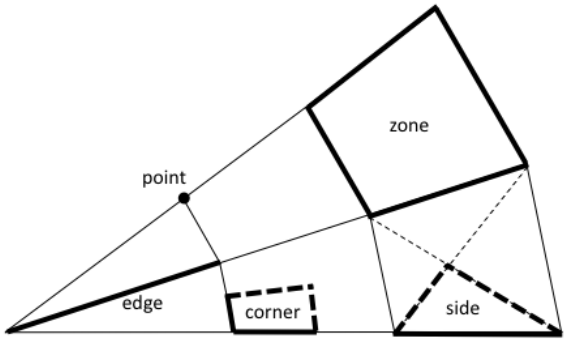


Figure 2: PENNANT terminology for mesh entities.

```
                (no default for nzx; default nzy = nzx)
lenx, leny = total length in x, y directions
                (default for both = 1.0)
```

Note that changing the number of zones in a problem typically requires a corresponding change to the `dtinit` parameter in the input file. As a rule of thumb, if the resolution of the problem is increased by a factor of r, `dtinit` must decrease by a factor of r.

2 Data structure details

2.1 Mesh data structures

PENNANT is designed to use standard finite-volume meshes similar to those used by many common physics solvers. In particular, PENNANT supports 2-D unstructured meshes composed of arbitrary polygons.

The PENNANT mesh data structures are a subset of those used by FLAG. These are implemented in the `Mesh` class. FLAG supports 1-, 2-, and 3-D meshes with various geometries; for simplicity, PENNANT is restricted to the 2-D, cylindrical geometry case.

The PENNANT terminology for entities within a mesh is shown in Figure 2. The basic mesh elements in 0, 1, and 2 dimensions are called *points, edges,* and *zones* respectively. PENNANT also uses two types of sub-zone entities. Within any given zone, a *side* is a triangle whose vertices are two consecutive boundary points of the zone together with the zone center. A *corner* is a quadrilateral whose vertices are one boundary point of a zone, the midpoints of the two adjoining edges, and the zone center.

For each entity type, the first letter of its name is used as an identifier for variables associated with it. For example, `px` is an array of point coordinates, and `zvol` is an array of zone volumes. There are scalar variables of the form `numX` which give the number of `X`'s in the problem: `nump` is the number of points, and so on.

PENNANT also stores various mapping arrays between different entity types. Most of these are side-based, as shown in Figure 3. Given a side s, the following mapping arrays are available:

- `mapsz` gives the zone z of which s is a subregion.
- `mapse` gives the edge e on the boundary of s.

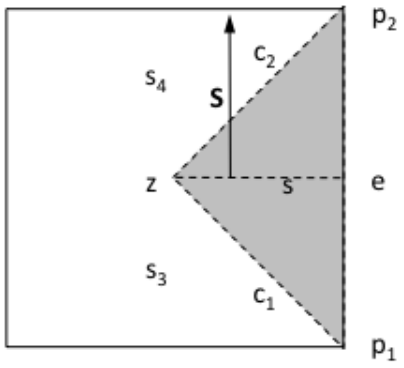


Figure 3: Various side map arrays supported by PENNANT.

- `mapsp1` and `mapsp2` give the two mesh points p_1 and p_2 on the boundary of s. It is assumed that the mesh is oriented according to a right-hand rule, so that the edge from p_1 to p_2 is always in a counter-clockwise direction relative to the zone.
- `mapsc1` and `mapsc2` give the two corners c_1 and c_2 which overlap with s, where c_1 is before c_2 in a counter-clockwise traversal of the zone.
- `mapss3` and `mapss4` give the two sides s_3 and s_4 on either side of s, where s_3 is before s and s_4 is after it in a counter-clockwise traversal of the zone. [4]

There are also two edge-based arrays `mapep1` and `mapep2`, which give the two endpoints of an edge e; and two corner-based arrays `mapcz` and `mapcp`, which give the associated zone and point, respectively, of a corner c.

The `Mesh` class also has methods for computing other geometry-related variables, such as edge and zone centers, lengths, volumes, and surface vectors. The surface vector for a side s, shown by the vector $\mathbf{S}$ in Figure 3, is used by force computations in the hydro algorithms described later.

2.2 Chunk processing

The PENNANT `Mesh` class has been set up to support computation on chunks of the mesh in parallel. This is used to implement an OpenMP version of the code and should lend itself to other task-based approaches as well.

The maximum chunk size is controlled by the input file parameter `chunksize`. If `chunksize` is larger than the total number of sides in the mesh, the entire mesh is treated as a single chunk (this is the default).

In the current chunking approach, the list of points is simply divided into chunks of size `chunksize` (except for the final, leftover chunk). The list of sides is handled similarly, except that the size of each individual chunk is rounded down slightly if necessary so that each zone has all of its sides in the same

[4] If you're wondering why there isn't a `mapss2`: FLAG uses s_2 to denote the side that is across the edge e from side s. This variable isn't currently needed by PENNANT, so I didn't implement it or include it in the figure. It may be implemented in a future PENNANT release.

chunk. For each chunk, the *first* and *last* indices of the chunk are stored. (Note that *last* is actually one index beyond the end of the chunk, in a similar manner to STL iterators, so that the sides in a side chunk are those with $s_{first} \leq s < s_{last}$.) The total numbers of point and side chunks (`numpch` and `numsch`, resp.) are also stored.

Chunk processing for other mesh entities is tied to either point or side chunking, as appropriate. Zone and corner chunks correspond to side chunks, while boundary point chunks correspond to point chunks. (Edge chunking is not needed; the code has been written in such a way that there are no loops over edges.)

Then, nearly all of the routines in the main hydro cycle have been modified to take as input first and last indices of the appropriate mesh entity. This allows the hydro processing to be divided into four phases, two on point chunks, and two more on side chunks. Within each phase, all chunks are independent and can be processed in parallel. See `Hydro::doCycle()` for the complete code flow.

A few of the helper routines, particularly in the `QCS` class, use scratch arrays the size of the chunk currently being processed. The prefix `s0` is used for an array with one entry per side in the current chunk, with prefixes `c0` and `z0` used similarly for corners and zones respectively.

3 Physics details

3.1 Basic hydro algorithms

PENNANT provides a subset of the compatible Lagrangian staggered grid hydrodynamics (SGH) algorithms implemented in FLAG and described in [3]. These are implemented in the `Hydro` class. An outline of the main steps is given in Table 2.

The PENNANT hydro algorithm is a *Lagrangian* method, meaning that the computational mesh moves with the material as the problem state advances. This implies that the mass and material type within each zone are constant throughout the problem, but the zone's position and shape will change over time.

It is also a *staggered-grid* method, meaning that mesh positions and related variables (velocity, acceleration, etc.) are stored on points, while most state variables (density, energy, pressure, etc.) are stored on zones. Therefore, the calculation must frequently use values of zone-based variables to compute point-based results, or vice-versa. In Table 2, such results are shown in **bold**. (Note that this is true for five of the 11 steps shown.)

To facilitate these calculations, many of the calculation loops are done over sides, and some intermediate variables are stored on sides. This works since each side can be easily correlated to its corresponding zone and points using the mapping arrays in the `Mesh` object.

PENNANT hydro uses a *predictor-corrector* time integration method. Each cycle can be broken into two steps, shown in the table. The cycle begins with all problem state defined for the beginning of the timestep. In the predictor step, some variables are advanced to the middle of the timestep, in order to compute half-advanced point acceleration values. In the corrector step, the new accelerations are then used to advance all variables to the end of the timestep.

To implement the predictor-corrector scheme, it is necessary to store multiple values of some of the problem variables. This is done using the following notation convention:

- suffix 0 = the beginning of the timestep ("cycle n")
- suffix p = half-way through the timestep ("cycle n + 1/2")
- no suffix = completion of the timestep ("cycle n + 1")

Table 2: Basic data flow for FLAG/PENNANT hydrodynamics.

step	main inputs	main outputs
Predictor step:		
1. Update mesh	point velocity, position	point position (half-advanced)
1a. Update mesh geometry	point position	**side and zone volume; zone density; side surface vector**
2. Compute point masses	zone density, volume; side mass fraction	**point mass**
3. Update thermodynamic state	zone density, specific energy, work rate	zone pressure, sound speed
4. Compute forces	side surface vector; zone pressure	side and **point force**
4a. Apply boundary conditions	point force, velocity	point force, velocity (constrained)
5. Compute accleration	point force	point acceleration
Corrector step:		
6. Update mesh	point acceleration, velocity, position	point velocity, position (fully advanced)
6a. Update mesh geometry	point position	**side and zone volume**
7. Compute work	point position, velocity; side force	**zone work, work rate, total energy**
8. Update zone state variables	zone volume, mass, total energy	zone density, specific energy

Some examples are shown in Table 3. Note that some entries in the table are blank, since not all quantities are needed at all times.

3.2 Material model

PENNANT provides finite-volume, arbitrary-polygon cells with a gas material model, implemented in the `PolyGas` class. This class includes code to compute a simple gamma-law gas equation of state, and to compute the resulting pressure-based forces.

3.3 Subzonal pressures

PENNANT provides the Temporary Triangular Subzoning (TTS) algorithm described in [4, 7]. This is implemented in the `TTS` class. This prevents certain kinds of distortions of zones, such as "hourglassing," by estimating a pressure for each side, and adding a force to each side based on the difference between the zone and side pressures.

Note to FLAG users: The FLAG implementation of TTS contains, in addition to the subzonal pressure treatment, an artificial viscosity algorithm based on the subzonal pressures; the artificial viscosity is not part of the standard TTS description in the references. Only the subzonal pressure part of TTS is implemented in PENNANT.

Table 3: Examples of PENNANT variables and their dependence on timestep.

	Part of time step		
quantity	begin	middle	end
point coordinate	px0	pxp	px
point velocity	pu0		pu
point acceleration		pap	
point force		pf	
point mass		pmaswt	
zone center coordinate		zxp	zx
zone mass			zm
zone volume	zvol0	zvolp	zvol
zone density		zrp	zr
zone specific energy			ze
side volume		svolp	svol
side surface vector		ssurfp	

3.4 Artificial viscosity

PENNANT provides the tensor artificial viscosity algorithm of Campbell and Shashkov, described in [2]. This is implemented in the `QCS` class. (The symbol q is traditionally used to denote artificial viscosity, hence the `Q` prefix on the class name.) Artificial viscosity is a fictitious term commonly introduced into fluid flow equations to correctly handle shock regions with large discontinuities in the problem state variables.

Acknowledgements

Thanks to Mikhail Shashkov and the ASCR "Mimetic Methods for PDEs" project, and the ASC Hydrodynamics project, for providing support for this work. Thanks also to the Lagrangian Applications Project members who have contributed to the FLAG code; parts of the PENNANT code and documentation are adapted from their work.

References

[1] D.J. Benson. Momentum advection on a staggered mesh. *J. Comput. Phys*, 100:143–162, 1992.

[2] J. Campbell and M. Shashkov. A tensor artificial viscosity using a mimetic finite difference algorithm. *J. Comput. Phys*, 172:739–765, 2001.

[3] E.J. Caramana, D.E. Burton, M. Shashkov, and P.P. Whalen. The construction of compatible hydrodynamics algorithms utilizing conservation of total energy. *J. Comput. Phys.*, 146:227–262, 1998.

[4] E.J. Caramana and M.J. Shashkov. Elimination of artificial grid distortion and hourglass-type motions by means of Lagarangian subzonal masses and pressures. *J. Comput. Phys.*, 142:521–561, 1998.

[5] W. F. Noh, Errors for calculations of strong shocks using an artificial viscosity and an artificial heat flux. *J. Comput. Phys.*, 72:78, 1987.

[6] L. I. Sedov, Similarity and Dimensional Methods in Mechanics. Academic Press, New York, 1959.

[7] K.B. Wallick. Temporary triangular subzoning (TTS), in REZONE: A method for automatic rezoning in two-dimensional lagrangian hydrodynamics problems. Technical Report LA-10829-MS, Los Alamos National Laboratory, Los Alamos, NM 1987.

A Version Log

0.4 January 2013. First open-source release. Fixed a bug in QCS and added some optimizations. Added Sedov and Leblanc test problems, and some new input keywords to support them.

0.3 July 2012. Added OpenMP pragmas and point chunk processing. Modified physics state arrays to be flat arrays instead of STL vectors.

0.2 June 2012. Added side chunk processing. Miscellaneous minor cleanup.

0.1 March 2012. Initial release, internal LANL only.

B Copyright and License Information

LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL LOS ALAMOS NATIONAL SECURITY, LLC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.