Draft version November 26, 2018
Typeset using LaTeX **twocolumn** style in AASTeX62

DRAFT 1 Processing JUPITER Hydrodynamics Simulation Data for Visualisation in Paraview DRAFT 1

Evert Nasedkin[1]

[1] *ETH Zurich*

ABSTRACT

1. INTRODUCTION

This report examines the software developed to process JUPITER hydrodynamic simulation data to allow for visualisation using the open source program Paraview.

1.1. *JUPITER*

JUPITER is a 3D, nested mesh simulation program that solves the hydrodynamic and radiative transfer equations using a high order Godunov scheme. This allows a spatial resolution of about 0.8 Jupiter radii at the finest mesh level. The radiative transfer is calculated with the method of Commerçon et al. (2011), with Dirichlet boundary conditions on the boundaries between mesh levels. Thus JUPITER solves for mass and momentum conservation, along with total energy, accounting for coupling between thermal radiative energy (ϵ_{rad}). The governing equations for the program are therefore:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{1}$$

$$\frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v}) + \nabla P = -\rho \mathbf{v} \cdot \nabla \Phi + \nabla \cdot \bar{\boldsymbol{\tau}} \tag{2}$$

$$\frac{\partial E}{\partial t} + \nabla \cdot [(P\mathbb{1} - \bar{\boldsymbol{\tau}}) \cdot \mathbf{v} + E\mathbf{v}] = \rho \mathbf{v} \cdot \nabla \Phi - \rho \kappa_P c \left(\frac{B(T)}{c} - \epsilon_{rad} \right) \tag{3}$$

$$\frac{\partial \epsilon_{rad}}{\partial t} = -\nabla \cdot F_{rad} + \rho \kappa_P c \left(\frac{B(T)}{c} - \epsilon_{rad} \right) \tag{4}$$

The density is given by ρ, E is the total gas energy (U + K), $\mathbf{v}$ is the gas velocity, P is pressure, Φ is the gravitational potential and T is temperature. κ_P is the Planck opacity from eqn. 6 and $B(T)$ is the thermal blackbody radiation power, given by $4\sigma T^4$. c is the speed of light and σ is the Stephan-Boltzmann constant. $\mathbb{1}$ is the identity and $\bar{\boldsymbol{\tau}}$ is the stress tensor:

$$\bar{\boldsymbol{\tau}} = 2\rho\nu \left(\bar{\boldsymbol{D}} - \frac{1}{3} (\nabla \cdot \mathbf{v}) \mathbb{1} \right) \tag{5}$$

Corresponding author: Evert Nasedkin
evertn@student.ethz.ch

The Planck Opacity is defined as

$$\kappa_P = \frac{\int_0^\infty \kappa_\nu B_\nu(T)\, d\nu}{\int_0^\infty B_\nu(T)\, d\nu} \tag{6}$$

where ν is the kinematic velocity and $\bar{\mathbb{D}}$ is the strain tensor. Finally F_{rad} is given by:

$$F_{rad} = -\frac{c\lambda}{\rho \kappa_R} \nabla \epsilon_{rad} \tag{7}$$

κ_R is the Rosseland mean opacity, and λ is a flux limiter to smooth the transition between optically thick and thin regions.

The equation of state of the system is taken to be

$$P = (\gamma - 1)\, \epsilon \tag{8}$$

where $\epsilon = \rho c_v T$ and the adiabatic index $\gamma = 1.43$.

JUPITER can then be supplied with a given set of initial conditions and system properties, and will calculate each of the hydrodynamic fields at each time step to explore planet-disk interactions and related physics.

1.2. *Adaptive Mesh Refinement*

TALK TO JUDIT TO MAKE SURE THIS SECTION IS CORRECT

The nested mesh system used in JUPITER is an example of adaptive mesh refinement (AMR) techniques. In general, AMR improves computational speed for a given maximum resolution, as it reduces the number of mesh elements in the grid while maintaining the high resolution in the region of interest as seen in fig. 1. In JUPITER, the mesh is not truly adaptive, as the refinement regions are predefined in the region of the planet embedded in the circumstellar disk. Other AMR techniques could refine the mesh only in regions of steep density or velocity gradients. However, as circumstellar are not a static-flow environment, the location of these gradients changes over time, and it would be difficult to manage a constantly changing grid. In contrast, JUPITER uses a static grid, and allows the flow to evolve throughout the grid over time.

Boundary conditions must be defined for each of the mesh regions, and this is achieved by interpolating the

results of a coarse mesh to provide the boundary conditions for the next level. The results of the fine level are then used to improve the results of the coarse mesh in the region. This is iterated for each mesh refinement level, with the edge length in each dimension being half that of the previous, coarser level.

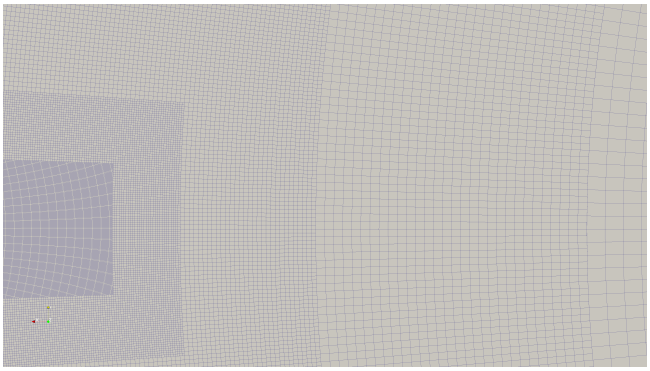

Figure 1. Nested mesh levels around the location of the planet from a JUPITER simulation.

1.3. *Outputs*

As JUPITER relies on parallel computation, each core will output the hydrodynamic fields in its mesh region. These can be stitched together to create output files for each hydrodynamic field over the whole disk.

The first output file is a descriptor file that provides simulation properties, along with the positions in spherical coordinates of each of the vertices of the mesh for each mesh level. For this report *phi* represents the azimuthal angle, and *theta* the polar angle from the z-axis, and coordinates are generally ordered (ϕ, r, θ). The file is structured as follows: TALK TO JUDIT ABOUT WHAT ELSE IS IN THE DESCRIPTOR FILE

```
[line 1]
[line 2]
[line 3]
[line 4]
[line 5]
[line 6]
[N_φ N_r N_θ]
[line 8]
[φ axis]
[r axis]
[θ axis]
```

This structure is then repeated sequentially for each mesh refinement level, from coarse to fine. The first and last two components of each axis are ghost cells, and should not be read in. The number of components along each axis (e.g. N_ϕ) takes this accounts for this, however it assumes zero index counting. Thus if there are 681 ϕ components, the descriptor file will read 680, and the total axis in the file will contain 685 elements.

Each hydrodynamic field is stored in a data file, with a data point for each cell of the mesh. Note that while the descriptor file stores the locations of the vertices of the cells, all of the field data is defined at the barycentre of a given cell. The data is stored as binary doubles, and is ordered such that for a given coordinates (ϕ_i, r_j, θ_k), i is iterated the fastest and k the slowest. For vector data (velocities), each component of the vector is ordered as a scaler, with the azimuthal velocity listed first in its entirety, followed by the radial component, followed by the polar component. This ordering is also user adjustable in JUPITER, and may change. All data is stored in code units, and must be converted to physical units after read in.

Many of these hydrodynamic fields cannot be processed for visualisation with existing software (e.g. RadMC3D), so a conversion tool was necessary to allow for visualisation in Paraview.

2. PROCESSING TOOLS

All of the tools described on this section are available on Github: https://github.com/nenasedk/JUPITER_VTKFileConversion

2.1. *VTK File Structure*

Paraview uses the VTK file format to store data for visualisation. While a modern xml format exists, the legacy format is simpler to use and has much more extensive documentation, and was thus chosen for this project. Future work could explore the usage of a VTK format developed for adaptive mesh grids.

The legacy VTK format is structured into 5 parts:

1. The file version and identifier, which must be exactly `# vtk DataFile Version x.x` For this project we are using VTK version 2.0.

2. The header, one line terminated by `\n` to describe the data.

3. File format. Either `ascii|binary`

4. Dataset structure. This describes the geometry and topology of the dataset, and consists of the word `DATASET` followed by a keyword description of the data. We use an Unstructured Grid to describe the data. An unstructured grid requires both a list of coordinates and a list of connections to fully specify the locations and connections of each cell in the grid.

Following the keyword is the list of coordinates used to describe the data. This initiated by line `POINTS n datatype`, where n is the number of coordinates and datatype is a C type (we use doubles). The JUPITER coordinate grid is listed in order ϕ, r, θ, with ϕ iterating the fastest and θ the slowest. Each coordinate vector is converted to a Cartesian grid before being written to file, so the VTK file will contain the x,y,z components of each vertex in the mesh in CGS units, centred on the star. Each mesh level is appended subsequently from coarsest to finest.

After the list of coordinates, the cells are listed. This section begins with the line `CELLS m` where m is the number of cells. Each row begins with the number of vertices of the cell, followed by the index of the coordinates of each vertex of the cell. We use a hexahedral cell type (cell type 8), and a description of the algorithm to compute the indices is found in sec. 2.2.

Following the list of cells is the list of cell types for each cell. This is started by the line `CELL_TYPES m` where m is the number of cells. All of the cells used in JUPITER are hexahedral (type 12).

5. Dataset attributes. This can be either `POINT_DATA`, where each datapoint is located at a mesh vertex, or `CELL_DATA`, where each datapoint is located at the centre of a cell. All JUPITER hydro fields are `CELL_DATA`. Additionally, each datapoint can be either a `SCALAR` or `VECTOR`, the latter of which is used to store velocities.

An example file is included in Appendix A.

2.2. *Data Processing Methods*

Requirements

To run the python conversion script the user must have installed at minimum:

- Python 2.7 or greater
- numpy package
- astropy package
- vtk package v8.1.1

Overview

To process the outputs of the JUPITER simulations into the VTK fileformat, a python class was written. The class has a range of user settable functions, allowing the user to write a script to wrap the class for a given conversion job, or use the interactive command line interface written for the class. In general, the user must provide the input and output directories, the output number, the number of mesh refinement levels, the hydrodynamic field to convert, the orbital radius of the planet in AU and the mass of the star in solar masses.

The program flow is as follows:

1. Setup directories and filenames, and set science variables.
2. Read in the coordinates from the descriptor file. From the axes read in, build a grid of 3D coordinates (ϕ_i, r_j, θ_k), and create an additional grid with the coordinates converted to Cartesian.
3. For each mesh level, define the boundaries of the next finest level. Cells (NOT grid points) within this region of interest (ROI) will be ignored.
4. Compute the indices of each vertex of each cell. This is achieved using a stride calculation in the `ComputeStructuredCell()` function:
   - Label the indices from 0 to 7, counter clockwise and from bottom to top. Thus the 'bottom' level of each hexahedral cell consists of indices 0-3, the top from 4-7, with index 4 directly above index 0.
   - The axis are iterated *phi* fastest and θ slowest. Therefore index 1 will be one greater, but index 3 will be a full ϕ axis farther along than index 0. Likewise, index 4 will be an $r - \phi$ plane greater than index 0.
   - Iterate through each axis, incrementing the counters. Any cells with one or more vertices within the ROI will be skipped, but their index in the total list of cells is stored.
5. Read in the data from the descriptor file, reshaping vector (velocity) data as necessary. Remove the datapoints with the indices of the removed cells.
6. Write the data to a binary or ascii VTK file. If the output file already exists, the user can choose to overwrite the file, or append new data to the existing file.

Usage

To use the command line interface simply use `python Convert.py` Within the Convert.py script, the user can change whether to output to a binary or ascii VTK file, as well as the location of the input data from JUPITER. This data must be in a folder labelled output, and the default path to this directory is to use the current working directory.

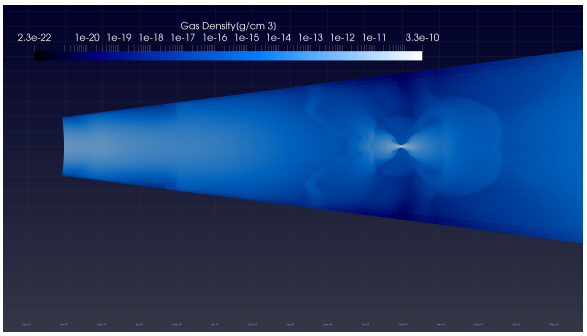

Figure 2. Cross section of gas density of the circumplanetary disk of a planet embedded at 50 AU in the circumstellar disk of a 1 solar mass star.

A second interface has been developed to allow for easier scripting of inputs. To use this, use:
`python script_convert.py [first] [grid level]\ [radius] [mass] -l [last] -b [binary/ascii] -d\ [dir] [field list]`

Or without all optional tagged arguments:
`python script_convert.py first [grid_level]\ [radius] [mass] [field list]`

Tagged arguments can be placed in any order, but the field list must be the last argument passed.

- `[first]` The first simulation output to process.
- `- l [last]` The last simulation output to process. Assumes that all integers between first and last exist and should be processed.
- `[grid_level]` The mesh refinement level.
- `[radius]` The orbital radius of the planet in AU.
- `[mass]` The mass of the star in solar masses.
- `[-b binary/ascii]` A string (b)inary or (a)scii to set the output file type. Binary is recommended.
- `[-d dir]` Directory where the output folder from JUPITER is located
- `[field_list]` A list of space separated strings denoting which hydro fields should be converted.

REFERENCES

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33

Bertin, E., & Arnouts, S. 1996, A&AS, 117, 393

Corrales, L. 2015, ApJ, 805, 23

Ferland, G. J., Porter, R. L., van Hoof, P. A. M., et al. 2013, RMxAA, 49, 137

Hanisch, R. J., & Biemesderfer, C. D. 1989, BAAS, 21, 780

Lamport, L. 1994, LaTeX: A Document Preparation System, 2nd Edition (Boston, Addison-Wesley Professional)

Schwarz, G. J., Ness, J.-U., Osborne, J. P., et al. 2011, ApJS, 197, 31

Vogt, F. P. A., Dopita, M. A., Kewley, L. J., et al. 2014, ApJ, 793, 127

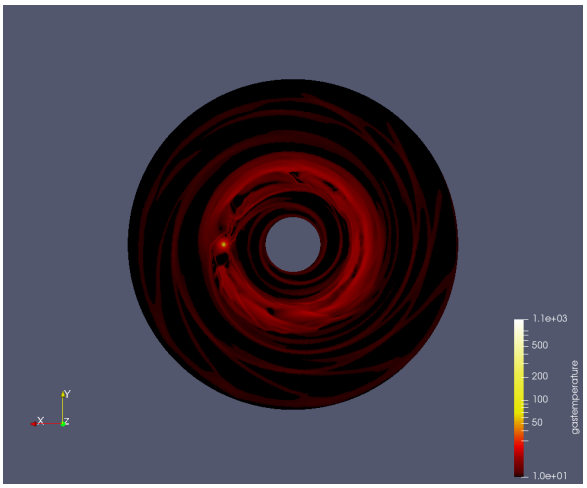

Figure 3. Temperature of the midplane of the circumstellar disk.

APPENDIX A

An example of a simple VTK file:

```
# vtk DataFile Version 2.0
Jupiter Simulation Data
ASCII
DATASET UNSTRUCTURED_GRID
POINTS n double
p0x p0y p0z
...
p(n-1)x p(n-1)y p(n-1)z

CELLS m
nVert0 i0 i1 i2 ... i()nVert0-1)
...
nVert(m-1) i0 i1 i2 ... i(nVert(m-1)-1)

CELL_TYPE m
type0
..
type(m-1)

CELL_DATA m
SCALARS name double
LOOKUP_TABLE default
s0
...
s(m-1)
```