Draft version December 19, 2018
Typeset using LaTeX **twocolumn** style in AASTeX62

Processing JUPITER Hydrodynamics Simulation Data for Visualisation in Paraview.

Evert Nasedkin[1]

[1] *ETH Zurich, Institute for Particle and Astrophysics*

ABSTRACT

At the present, no standard procedure exists for visualising the 3D outputs of JUPITER hydrodynamic simulations. Therefore, we have developed a tool that converts the hydrodynamic fields calculated on a nested mesh into a VTK file format which can be visualised using standard open source software.

1. INTRODUCTION

This report examines the software developed to process JUPITER hydrodynamic simulation outputs to allow for visualisation using the open source program Paraview Ahrens et al. (2005).

1.1. *JUPITER*

JUPITER is a 3D, nested mesh, simulation program that solves the hydrodynamic and radiative transfer equations using a high order Godunov (1959) scheme described in Szulágyi et al. (2016) Szulágyi et al. (2014) and De Val-Borro et al. (2006). 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, B. 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 and radiative energy (ϵ_{rad}). The governing equations for the program are therefore as in Szulágyi et al. (2016):

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

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

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

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

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 tensor and $\bar{\boldsymbol{\tau}}$ is the stress tensor:

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

where ν is the kinematic velocity and $\bar{\mathbb{D}}$ is the strain tensor. The Planck Opacity is defined as in Bitsch et al. (2013):

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

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, also defined in Bitsch et al. (2013).

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. These must then be further processed for visualisation and study.

1.2. *Adaptive Mesh Refinement*

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

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

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.

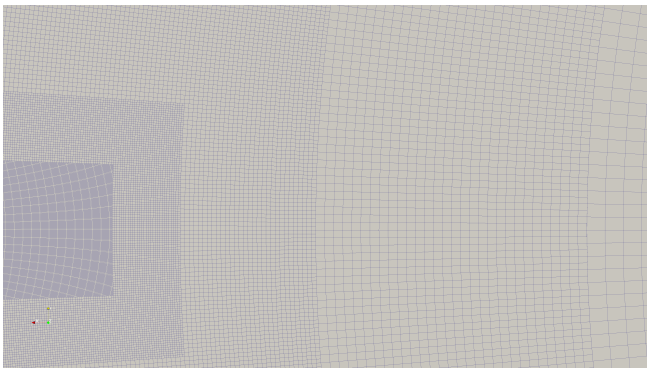

Figure 1. Nested mesh levels around the location of the planet from a JUPITER simulation.

1.3. *Outputs*

As JUPITER relies on parallel computation, each computational 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 ϕ represents the azimuthal angle, and θ the polar angle from the z-axis, and coordinates are generally ordered (ϕ, r, θ). The file is structured as follows:

```
[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 lines without descriptions are undocumented lines used within the JUPITER simulations, and are not relevant for this report. 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 n ϕ components, the descriptor file will read $n-1$, and the total ϕ axis in the file will contain $n+4$ 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 scalar, 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. This section describes the format as in Visual Toolkit Organization (2009). 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. Although the JUPITER grid is based in spherical coordinates, for a small enough grid size a Cartesian grid with hexahedral cells provides a 'close-enough' approximation for practical purposes.

   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, astropy and the python VTK package v8.1.1.

Overview

To process the outputs of the JUPITER simulations into the VTK file format, 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 other primary requirement of the tool was to be at most of O(n) time complexity, as the mesh can contain millions of elements. This restricts which algorithms can be used throughout the program.

The program flow is as follows:

1. Set-up 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 as shown in Fig. 2.
   - The ϕ axis is iterated fastest and θ the 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.

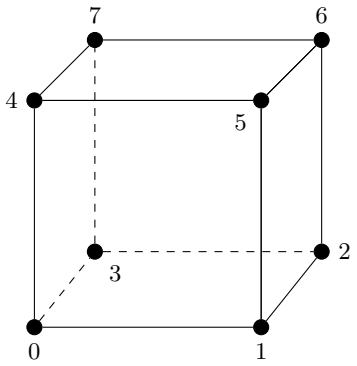

Figure 2. Labelling of hexahedral cell indices.

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 outputXXXXX where the X's denote the simulation output number with leading zeros. The default path to this directory is the current working directory.

A second interface has been developed to allow for easier bash scripting. To use this, enter the following into the command line or a bash script:

```
python script_Convert.py [first] [grid level]\
[radius] [mass] [-l [last]] [-v]\
[-b [binary/ascii]] [-d[dir]]\
[-f [field list]]
```

Or without all of the optional tagged arguments:

```
python script_Convert.py first [grid_level]\
[radius] [mass] [-f [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.
- `[-v]` Include if velocities should be planet centred.
- `[-b [binary/ascii]]` A string (b)inary or (a)scii to set the output file type. Binary is recommended, and is the default value.
- `[-d [dir]]` Directory where the output folder from JUPITER is located
- `[-f [field_list]]` A list of space separated strings denoting which hydro fields should be converted.

2.3. *Technical Challenges*

Several technical challenges needed to be overcome to implement this data processing tool.

Cell Counting—To visualise data in Paraview, cells cannot overlap, but the output from JUPITER contains overlapping cells within the ROI around the planet. This creates challenges when indexing the vertices of cells, as the axss length is not constant. However, Paraview does allow grid points to occur without being used. This allows us to store the full set of mesh points in the VTK file, and use a standard stride method with constant length axes to index cell vertices. Any cells with a vertex within the ROI are discarded, with the cell number being noted to allow for removal of the datapoint.

Velocity Vectors—The conversion of the velocity vectors from the initial data file to the VTK file was not straightforward. In the initial data file, the velocity vectors are stored component-wise, with the full list of azimuthal velocity components, followed by the radial and finally the polar velocities. These velocities have the azimuthal velocity of the planet subtracted, so we are in the co-rotating frame of the planet. These velocities are then converted from spherical to Cartesian coordinates using the following set of transformations:

$$\dot{x} = \dot{r}\sin\phi\sin\theta + r\dot{\phi}\cos\phi\sin\theta + r\dot{\theta}\sin\phi\cos\theta \quad (9)$$

$$\dot{y} = -\dot{r}\cos\phi\sin\theta + r\dot{\phi}\sin\phi\sin\theta - r\dot{\theta}\cos\phi\cos\theta \quad (10)$$

$$\dot{z} = \dot{r}\cos\theta - r\dot{\theta}\sin\theta \quad (11)$$

Where a dot denotes the time derivative of the coordinate. Note that the direction of the y component is inverted from the standard transformation.

3. PARAVIEW VISUALISATION

3.1. *Paraview*

Paraview is an interactive visualisation program that allows the user to visualise large datasets,with the processing done either in serial on a local machine or distributed on a server. As with the VTK file format, Paraview is

an open source project primarily developed by Kitware, with additional funding and support from the Sandia and Los Alamos National Laboratories. While the full scope of Paraview's visualisation tools is extensive, and detailed in the user guide Ahrens et al. (2005), we will outline some of the common techniques relevant to this report.

Once a VTK file has been produced by the conversion tool, it can be opened in Paraview. The user may then select a suitable colour map for the data. Best practices should be followed when producing publication-ready visualisations. Colour hue and brightness should be considered to allow for accessible images that can convey the content of the data. A large range in brightness and hue will allow small variations in the data to be more visible. The luminosity should be consistent across the colour range, so as not to emphasise certain features, unless these are the features in the data that should be emphasised. Many pre-made colour palettes are available in Paraview, including standard matplotlib palettes commonly used in python or matlab visualisations, allowing for standardisation across various tools. Opacity scaling, and logarithmic colour scales are also useful tools when visualising complex data.

In addition to colour, Paraview also allows the user to select various ways of representing the data in a 3D render. The default is to view the surface of the object, additional options can represent the data as points in space, a volume render, a wireframe or other options. These can be used to develop a better understanding of the 3 dimensional nature of the data.

Following the initial rendering, **filters** can be applied to the data. Paraview currently includes around 120 different filters, several commonly used ones are summarised here.

Slice Extract a 2D slice from 3D data given an orientation and position.

Clip Given an orientation and position, remove all cells 'outside' of the cut.

Threshold Remove all cells below a minimum, or above a maximum threshold.

Glyphs Visualise vector data as arrows or other objects for flow visualisation. Typically set to show every nth vector.

Stream Tracer Compute particle trajectories in a flow and visualise using streamlines. The user should set an initial point, with a small radius, and select whether to integrate forwards or backwards from that point (or both). For speed, an RK2 integrator is suggested. Typically only a small number of streamlines are necessary to visualise the flow.

Tube Replace the streamlines generated by a Stream Tracer filter with 3 dimensional tubes, for easier viewing.

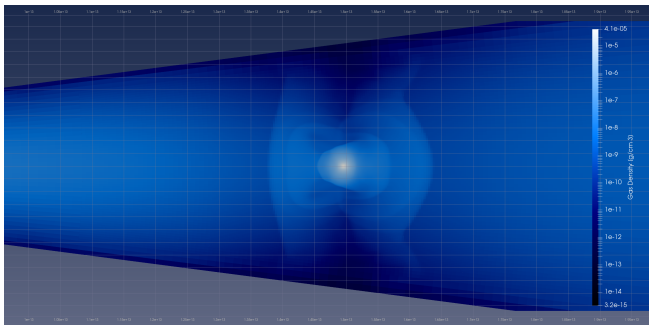

Figure 3. Cross section of gas density of the circumplanetary disk of a 1 Jupiter mass planet embedded at 50 AU in the circumstellar disk of a 1 solar mass star.

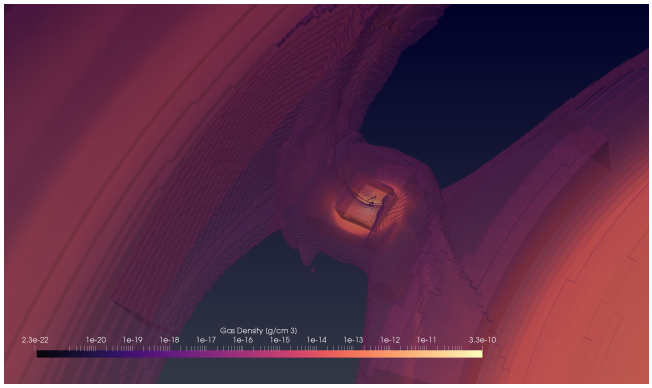

Figure 4. 3D density structure of the circumplanetary disk.

All of these filters can be combined in various ways to generate interesting and informative visualisations. Velocity vectors can be plotted over density fields; multiple slices can show the 3D structure of the temperature field. Additional tools include axis visibility, with customisable labelling, background settings (solid colour, gradient or image), alternate render views (slice view, charts, etc.) animations and more.

In addition to Paraview, the python VTK library allows for scripting of these visualisations, which could be used to automate and standardise visualisations for a project.

3.2. *Results*

With the conversion tool developed, we can present several examples of possible visualisations made using Paraview Ahrens et al. (2005). Fig. 3 shows a cross section of the gas density in the region of the circumplanetary disk along with the 3D density structure in Fig. 4.

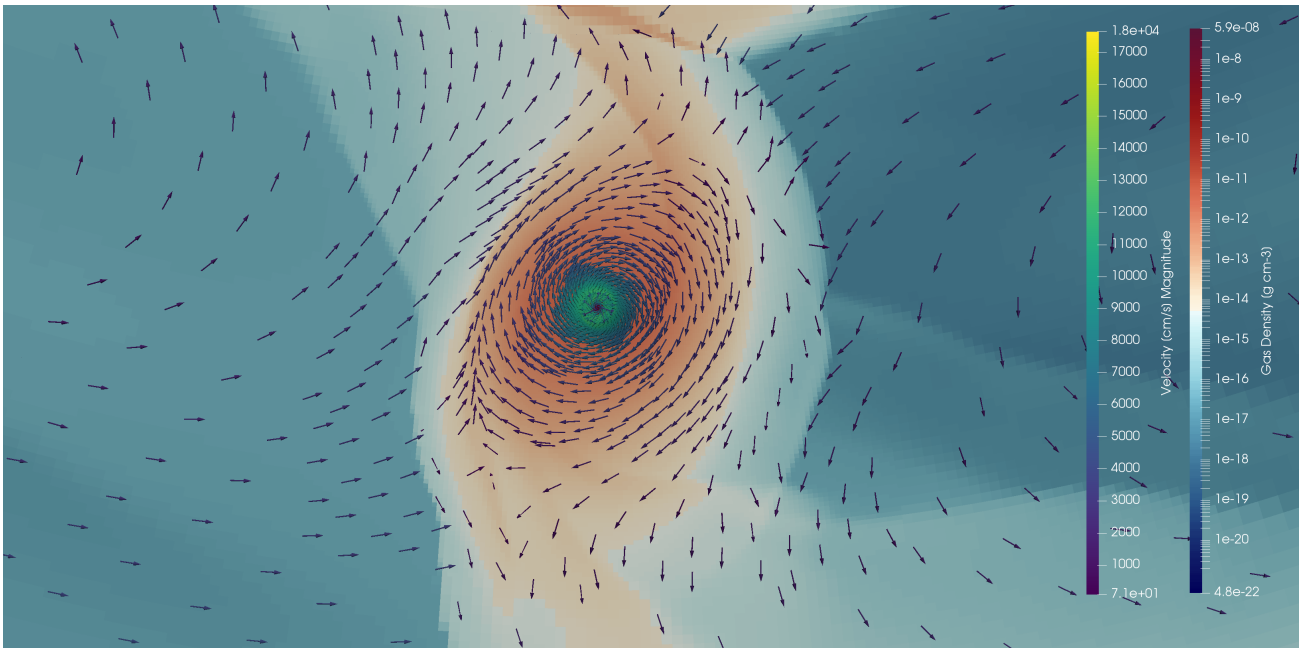

Figure 5. Velocity vectors spiralling out from the circumplanetary disk.

This shows the dens regions near to the forming planet, and a potential location for satellite formation. Fig. 8 shows the gas temperature of the mid-plane of the disk, coloured to a blackbody spectrum.

Fig. 5 shows the gas velocity in the circumplanetary disk, showing that the gas is spiralling away from the planet. This demonstrates that the circumplanetary disk is *not* an accretion disk, but rather a de-accretion disk, as gas that flows in from above and below the planet spirals away in the disk. This flow is also shown in Fig. 6. In addition, Fig. 7 shows the location in the temperature profile where the infowing gas shocks as it approaches the dense planet. Fig. 9 shows the increase in internal energy of the gas near the planet as it is adiabatically compressed.

More complicated renders – particularly when rendering 3D volumes – are very computationally expensive, and it is recommended to set up a Paraview server dedicated to rendering the dataset. Other possibilities include 3D time series animates, image generation using python scripts, streamline and flow visualisation and more.

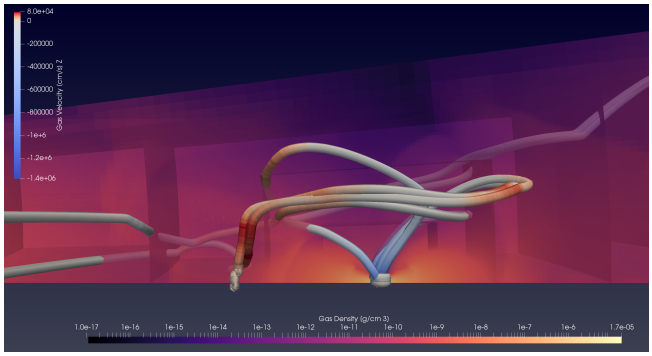

Figure 6. Inflow of gas from above onto an accreting planet.

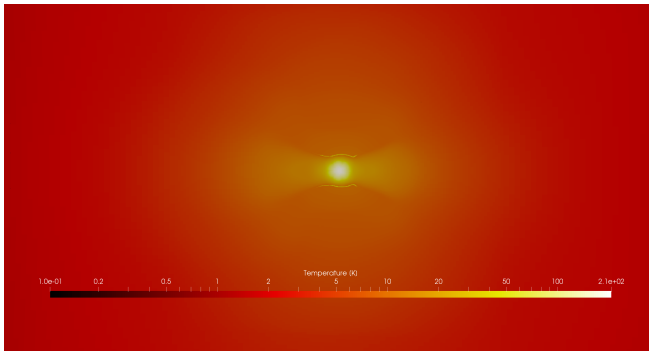

Figure 7. Shock front of gas flow near to the planet.

4. CONCLUSIONS

We have successfully developed a data processing tool to allow JUPITER simulations results to be visualised using Paraview or other VTK software. This will allow further exploration of the physics of circumstellar and circumplanetary disks, as the increasingly complex 3D results require adequate visualisation to understand and study.

5. ACKNOWLEDGEMENTS

I would like to sincerely thank my supervisor Dr. Judit Szulágyi for her guidance, mentorship and patience during this project.

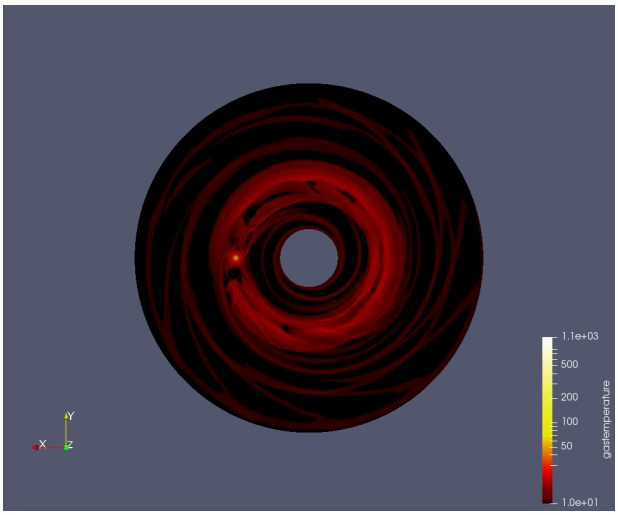

Figure 8. Temperature of the mid-plane of the circumstellar disk.

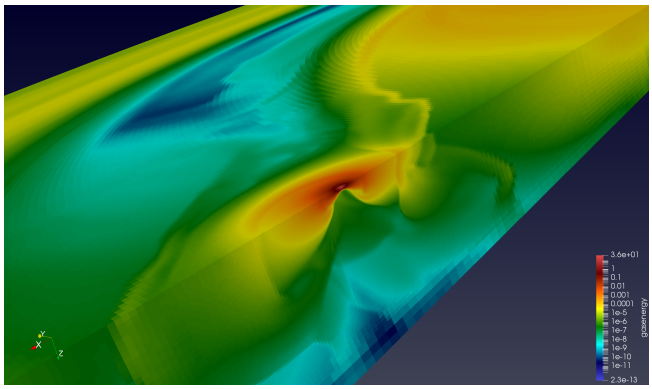

Figure 9. Total internal energy of the gas near the planet.

I would also like to acknowledge the Kitware Inc. for supporting the open source platforms of the Visualization ToolKit (VTK) and Paraview, upon which this work relies.

REFERENCES

Ahrens, J., Geveci, B., & Law, C. 2005, ParaView: An End-User Tool for Large Data Visualization (Elsevier)

Bitsch, B., Crida, A., Morbidelli, A., Kley, W., & Dobbs-Dixon, I. 2013, A&A, 549, 124

Commerçon, B., Teyssier, R., Audit, E., Hennebelle, P., & Chabrier, G. 2011, A&A, 529, A35. https://doi.org/10.1051/0004-6361/201015880

De Val-Borro, M., Edgar, R. G., Artymowicz, P., et al. 2006, Monthly Notices of the Royal Astronomical Society, 370, 529

Godunov, S. K. 1959, 47 (89), 271

Szulágyi, J., Masset, F., Lega, E., et al. 2016, 10, 1

Szulágyi, J., Morbidelli, A., Crida, A., & Masset, F. 2014, Astrophysical Journal, 782, arXiv:arXiv:1312.6302v2

Visual Toolkit Organization. 2009, 1

APPENDIX

A. LEGACY VTK FILE FORMAT

An example of a simple VTK file Visual Toolkit Organization (2009):

```
# 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)
```