HAAGA-HELIA
ammattikorkeakoulu

Making an observation geometry tool for the international Rosetta mission data analysis and visualisation

Touko Haunia

Thesis work
Degree programme in Business ICT
2014

HAAGA-HELIA
ammattikorkeakoulu

Tietojenkäsittelyn koulutusohjelma

Tiivistelmä

Raportin palautuksen tai esityksen päivämäärä

Tekijä tai tekijät Touko Haunia	**Ryhmätunnus tai aloitusvuosi** 2009S
Raportin nimi Making an observation geometry tool for the international Rosetta mission data analysis and visualisation	**Sivu- ja liitesivumäärä**
Opettajat tai ohjaajat Jukka Juslin	

Tässä opinnäytetyössä tehdään työkalu Ilmatieteen Laitoksen avaruustutkimuksen käyttöön. Työkalu on suunniteltu erityisesti Euroopan Avaruusjärjestön Rosetta-luotainta varten, ja se kirjoitetaan C++ -kielellä.

Ilmatieteen Laitos kehittää simulaatiomalleja mallintamaan aurinkotuulen vaikutusta taivaankappaleiden magneettikentässä ja ilmakehässä. Simulaatiot perustuvat aina parhaaseen oletukseen, joten aidon in-situ mittausdatan saaminen on tieteen kannalta elintärkeää. Rosetta-luotain tuottaa tätä mittausdataa komeetta 67P/Churyumov-Gerasimenkolta vuodesta 2014 eteenpäin.

Opinnäytetyössä kehitetty työkalu auttaa yhdistämään luotaimen paikkatiedon, instrumenttien mittausdatan sekä simulaatiomallit yhdeksi kuvaajaksi, jolloin tiedon analysointi on helpompaa. Opinnäytetyössä ei keskitytä tutkimuksellisiin yksityiskohtiin, vaan kerrotaan korkeammalla tasolla työkalun vaatimuksista, luotaimesta sekä sen paikkatietojen hankinnasta.

Työkalu hyödyntää Yhdysvaltain Avaruushallinnon kehittämää ohjelmaa luotaimen rata- ja asentotietojen laskemiseen. Lasketut ratatiedot muutetaan yleisesti simulaatioissa käytettyyn VTK – tiedostoformaattiin.

Opinnäytetyön lopussa esitellään työkalulla tehtyjä kuvaajia, sekä esimerkkejä sen käytöstä simulaatiomallien, in-situ mittausten ja ratatietojen yhdistämiseen.

Asiasanat
Avaruustutkimus, C++, aikasarja-analyysi, aurinkotuuli

HAAGA-HELIA
University of Applied Sciences

Abstract

Date of presentation

Degree programme in Business ICT

Authors Touko Haunia	**Group or year of entry** 2009S
The title of thesis Making an observation geometry tool for the international Rosetta mission data analysis and visualisation	**Number of report pages and attachment pages**
Advisor(s) Jukka Juslin	

In this thesis work, an observation geometry tool is made for the Finnish Meteorological Institute. The main purpose of the tool is to support data analysis activities in European Space Agency Rosetta mission, and it is written in C++.

Finnish Meteorological Institute is developing simulation models to simulate effects of the solar wind in the magnetospheres and atmospheres of celestial bodies. Simulations are always based on best estimates, and therefore in-situ measurements are important assets in model development. The Rosetta mission will produce this data from the comet 67P/Churyumov-Gerasimenko.

The tool will help to combine simulation models with the ephemeris and instrument data of the spacecraft, creating a unique 3D plot where it is easy to see how the measurements from the spacecraft differ from the simulation-predicted data. This thesis work does not focus on scientific issues, but rather gives a higher level description of the requirements and implementation of the tool.

The tool utilizes software called SPICE toolkit from the US Space Agency, which is used to calculate ephemeris and attitude data. The calculated data is converted into the VTK common data format, which is often used in simulations.

At the end of the thesis work, an example plots and use cases are presented. During the development, some further improvements were also identified, which are discussed.

Key words
Space research, C++, time series analysis, solar wind

Index

1 Introduction

In this thesis work, an observation geometry tool is made for the Finnish Meteorological Institute. The main purpose of the tool is to support data analysis activities in European Space Agency Rosetta mission, and it is written in C++.

Finnish Meteorological Institute is developing simulation models to simulate effects of the solar wind in the magnetospheres and atmospheres of celestial bodies. Simulations are always based on best estimates, and therefore in-situ measurements are important assets in model development. The Rosetta mission will produce this data from the comet 67P/Churyumov-Gerasimenko.

The tool will help to combine simulation models with the ephemeris and instrument data of the spacecraft, creating a unique 3D plot where it is easy to see how the measurements from the spacecraft differ from the simulation-predicted data. This thesis work does not focus on scientific issues, but rather gives a higher level description of the requirements and implementation of the tool.

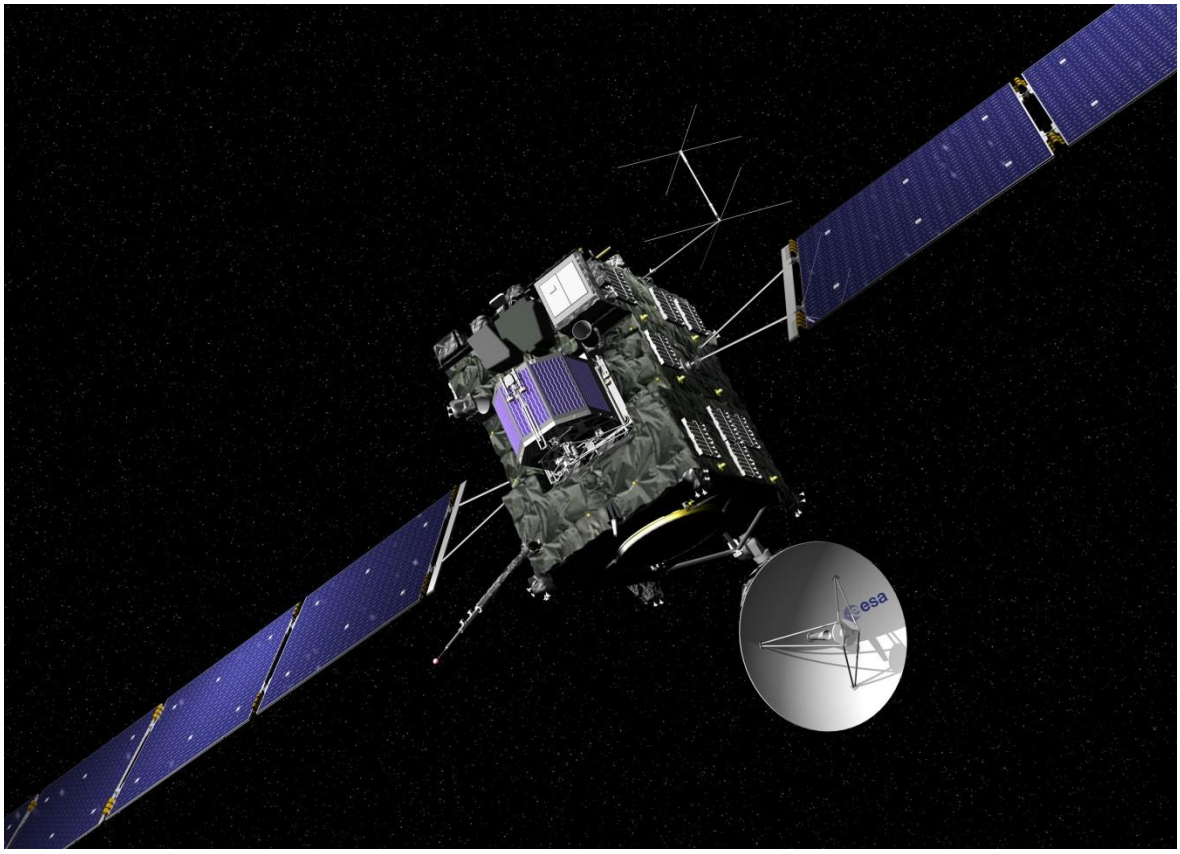
The tool utilizes software called SPICE toolkit from the US Space Agency, which is used to calculate ephemeris and attitude data. The calculated data is converted into the VTK common data format, which is often used in simulations.

At the end of the thesis work, an example plots and use cases are presented. During the development, some further improvements were also identified, which are discussed.

1.1 Rosetta -mission

Rosetta mission was selected in 1993 as a European Space Agency cornerstone mission. It was launched in 2004 with an Ariane 5 launcher from French Kourou, and arrived to the comet 67P/Churyumov-Gerasimenko during the summer of 2014, after a long voyage through interplanetary space. It consists of an orbiter and a lander called Philaea, both of which host a number of scientific instruments. Rosetta is managed by ESA, and built by a European consortium of space industry companies. Instruments are provided mainly by research institutes, who also manage the instrument operation and data-analysis.

Figure 1, Rosetta spacecraft (© ESA – J. Huart)



1.2 Scientific goals

Comets are fundamentally interesting to planetary science due to their formation during the early solar system. By studying comets, we can understand more about the early evolution of our solar system and possibly even learn how life formed on Earth. All previous missions to comets have been so-called fly-by missions with very high relative

velocity between the target body and the observing spacecraft. Instead of a fly-by, Rosetta will rendezvous with the comet and conduct in-situ measurements, which will give scientists unparalleled access to valuable information of the composition and properties of a comet. This also allows the tracking of outgassing volatiles through different activity phases of the comet as it approaches the sun and becomes more active.

Initial phases of the mission consist of careful observations with priority on spacecraft safety. Later stages see increased risks as the spacecraft is slowly manoeuvred closer to the comet nucleus to conduct more accurate scientific measurements.

1.3 Instruments

Rosetta is a platform for many instruments inside the orbiter and the lander. FMI has been participating in many of them, including plasma instruments LAP and ICA, and particle instrument COSIMA. FMI is also the principal investigator organization for the permittivity instrument PP on the lander.

2 Background

In this chapter, an explanation of the required calculations is presented. Basic celestial dynamics are explained and it is discussed how they can be calculated programmatically. Detailed description of the software implementation is given in the SPICE chapter of this thesis.

2.1 Observation geometry

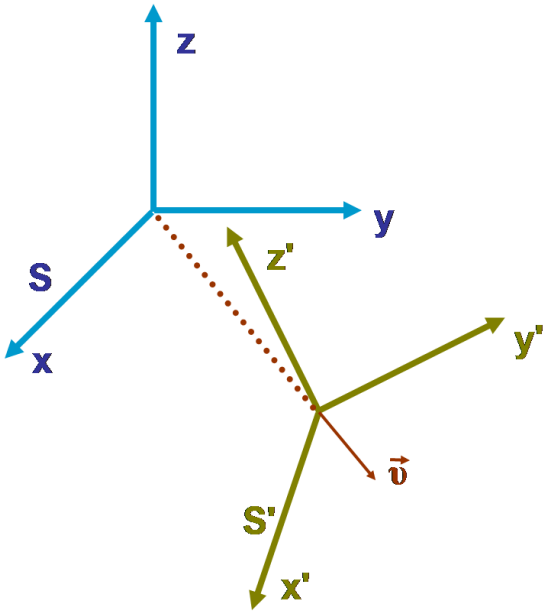
The term observation geometry is used when referring to the geometrical ancillary information of a certain observation e.g. an image or data. It is often critical to the analysis and calibration processes since derived properties are often dependent on external events such as target illumination, which is then determined with the help of precise observation geometry. An observation geometry is usually described in three vectors: position vector, velocity vector and a unit vector describing the orientation of the spacecraft. The presented tool is mainly meant to calculate these observation geometries.

2.2 Reference Frames and Rotation Matrices

In a three-dimensional space, an element has to be described with relations to other elements. To allow the precise calculation of trajectories and positions of elements in space, a system of reference frames has been developed. As there is no constant reference element in space, such as Earth's gravity or magnetic field, all physical vectors are described in a reference frame in order to calculate the kinematics and dynamics of elements. An example of this would be a position vector in an Ecliptic J2000 frame, which fixes the axes of the vector in a certain orientation of the Earth with relation to the Sun, in a certain point in time.

These frames and subsequent vectors can be transformed between different inertial and fixed frames, with the help of rotation matrices. In the following figure, a relation between two reference frames is described. After the figure, a definition of a rotation matrix is described.

Figure 2. Relation between two frames of reference



Let's describe the blue frame as **S** defined by the unit vectors **x**, **y** and **z** and the green frame as **S'** defined by unit vectors **x'**, **y'** and **z'**. Now if we want to describe an arbitrary vector **v** in the **S** frame instead of **S'**, it needs to be calculated with the so-called rotation matrix, which we describe here as **C**.

$$C = \vec{S}\, x\, \overrightarrow{S'}$$

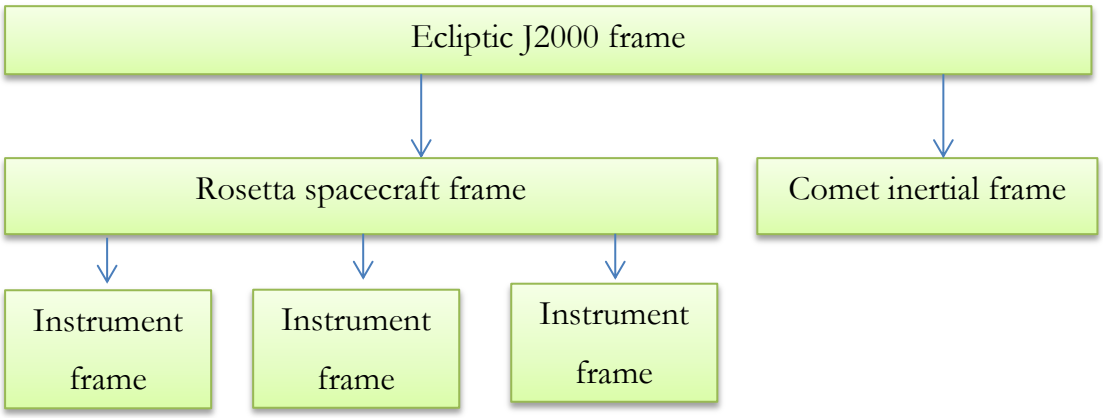
$$= \begin{bmatrix} x \cdot x' & x \cdot y' & x \cdot z' \\ y \cdot x' & y \cdot y' & y \cdot z' \\ z \cdot x' & z \cdot y' & z \cdot z' \end{bmatrix}$$

As we see, the rotation matrix **C** contains all the information to describe the orientation of two reference frames with respect to each other, and to transform vector coordinates between frames. We can now calculate the orientation, or attitude in spacecraft terminology, with the following equation:

$$v_2 = C v_1$$

2.2.1 Frame hierarchy

In the Rosetta spacecraft, a number of different frames are defined which are ordered in a so called frame hierarchy. This means that lower level frames are defined in relation to the upper level frames, so in order to calculate the orientation of the hierarchically lower frame, knowledge of all the upper frames is required. The topmost frame is usually fixed to an ecliptic frame in interplanetary missions. For scientific purposes, other frames can be used as well such as the so called CSO frame, which inertially fixes the frame axes in the motion of a specific celestial body relative to the Sun. This kind of fix benefits space weather research as it fixes one axis on the direction of the solar wind. Below is illustrated the simplified frame hierarchy of the Rosetta mission.

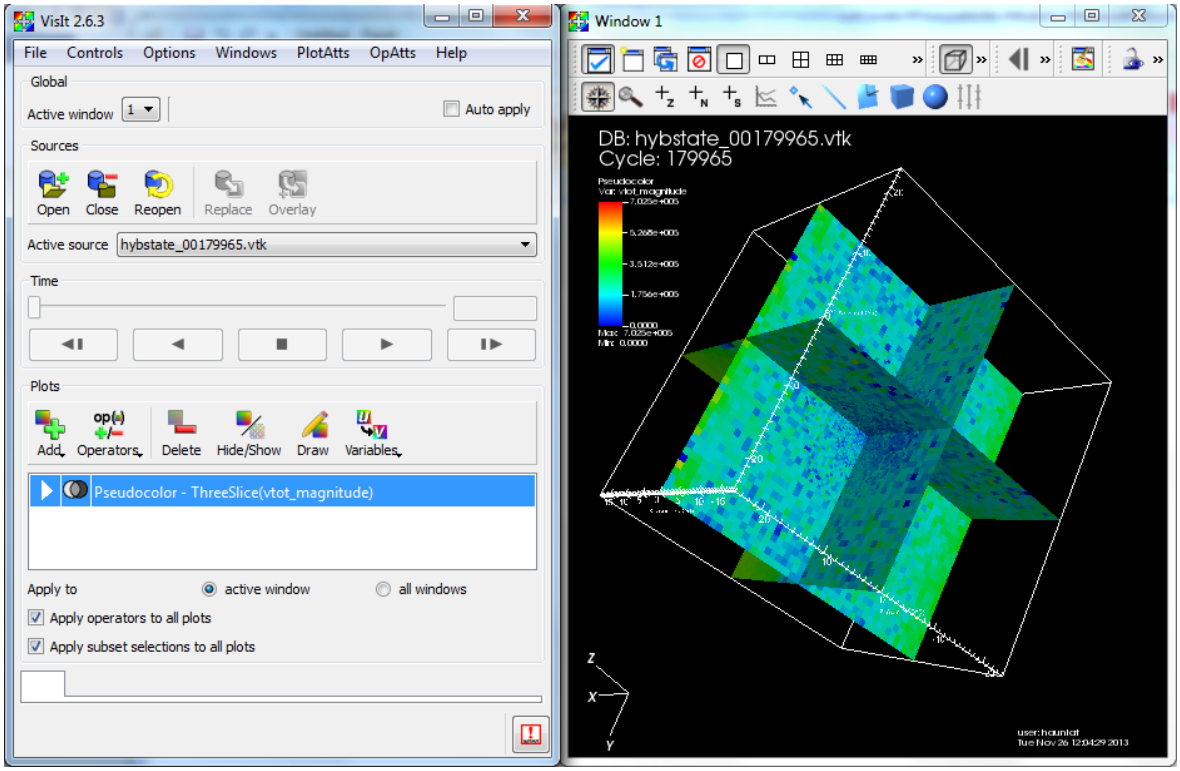


2.3 VisIt visualization software

VisIt is an open source visualisation tool created by the Lawrence Livermore National Laboratory in the United States. It is widely used in particle physics and high-energy physics, and supports many different file formats including Silo, VTK and NetCDF [https://wci.llnl.gov/codes/visit/].

VisIt is also the most widely used tool at the Finnish Meteorological Institute to visualise space weather phenomena. As seen in Figure 4, it has an intuitive graphical user interface which allows powerful manipulation of data and simulations. In the pseudocolor plot of Figure 4, an interaction between solar wind and a comet can be seen, visualised by the total velocity of the particles.

Figure 3. VisIt software



3 Requirements

The software requirements were identified in several meetings with scientists working with instrument data and simulation models. In the table below, a list of high-level re-

quirements are identified. No detailed requirements specification document was made as this was considered unnecessary.

Require-ment ID	Requirement description	Importance	Done?
REQ-OGT-0010	The tool shall calculate the position, velocity and orientation of an object relative to another object in a known reference frame	High	Yes
REQ-OGT-0011	The tool shall output the calculated results in a VTK file format	High	Yes
REQ-OGT-0020	The calculation shall be based on SPICE kernels	High	Yes
REQ-OGT-0030	The user shall be able to select the reference frame	Medium	Yes
REQ-OGT-0031	The user shall be able to select the used kernels	Medium	Yes
REQ-OGT-0040	The user shall be able to configure and run the tool via command line interface	Medium	Yes
REQ-OGT-0041	The user shall be able to configure the tool via a configuration file	Medium	Yes
REQ-OGT-0050	The user shall be able to output the calculated results in a CSV file format	Low	Yes
REQ-OGT-0060	The tool shall provide a well-defined API for integration with other code	Medium	Yes
REQ-OGT-0070	A user manual shall be provided for the tool	Medium	Yes

4 Implementation

The implementation was done with Eclipse CDT C/C++ IDE on a Linux environment. No special version control was used during implementation phase as the software was written by a single developer. The software is made of six classes, which are called by the following identifiers.

- ObsGeomImpl
- ConfigHandler
- CsvGenerator
- MessageHandler
- ObservationGeom
- VTKWriter

The purposes of these classes are described in this section. When applicable, example usage of the class is also presented.

4.1 ObsGeomImpl class

A decision was made to utilise a separate implementation class in order to support future integration of the software to other systems, and allows it to be used as a module. In other words, the implementation class acts as a simple API, which allows the abstraction of the main program execution and the retrieval of MessageHandler object and the ConfigHandler object (more of these later). With this abstraction, it is very simple to export the source code into another project and use it from there.

An example use case of this API would be the programmatical generation of arguments list which modifies the configuration settings and then calls the implementation class constructor. Below is an example usage of this API.

```
// Run the implementation
ObsGeomImpl obsGeomImpl(ac, av);

// Retrieve pointer to the messaging class
MessageHandler* obsGeomMessages = obsGeomImpl.getMessageHandler();

// Read error messages
obsGeomMessages->dumpErrorMessages();
```

4.2 ConfigHandler class

The ConfigHandler class uses the boost::program_options library (see chapter 5.1 for details) to parse user inputs and perform configuration via the command line interface or a separate configuration file. The library allows the setting of permanent configurations in the configuration file and temporarily overriding these configurations via the command line. This approach has proven useful when testing different parameters rapidly, and it makes bash scripting possible. The configuration options are separated to those that are visible with the help-command, those that are available but no visible, and those that are only available from the command line, such as the configuration file location. Configurations are saved in runtime memory during initialisation, where they can be retrieved at any time with simple getter methods. Below a metakernel file location is retrieved from the configuration and then loaded into the SPICE toolkit runtime.

```
   // Load the kernels into spice runtime memory.
   furnsh_c(config->getFurnshFileLocation().c_str());
```

4.3 CsvGenerator class

The CsvGenerator class generates standard Comma Separated Value (CSV) files using the fstream object. It operates simply by iterating the calculations and saving the data in ASCII format.

4.4 MessageHandler class

The MessageHandler class is responsible for brokering error, warning and debug messages during application execution. Depending on message severity, the message can be displayed to the user immediately, or collected into a separate runtime memory,

where it can be later retrieved. Below is an example how to initialise the tool with the help of the MessageHandler class.

```
1    // Initialise main program implementation object, with configuration
2    // arguments passed to the constructor
3    ObsGeomImpl obsGeomImpl(ac, av);
4
5    // Retrieve configuration and message handles from implementation object
6    ConfigHandler*   config = obsGeomImpl.getConfigHandler();
7    MessageHandler* message = obsGeomImpl.getMessageHandler();
8
9    // Check that the retrieved pointers are valid
10   if (!config || !message) {
11      std::cout << "Fatal error during initialisation!" << std::endl;
12   }
13
14   if (!obsGeomImpl.initialise()) {
15      message->reportError("Fatal error during initialisation!");
16      return false;
17   }
18
19   message->reportDebug("Observation geometry tool completed successfully");
```

4.5 ObservationGeom class

The ObservationGeom class is the main geometry calculation class, which includes methods for all the necessary SPICE calculations to support the Rosetta mission data analysis. This class can easily be reimplemented to support other science missions as well. Calculation methods are split based on the calculations done, so that a single method can be called to make a single calculation, for example an attitude calculation in a specific frame. The calling function sends a pointer to the data locations where the method will then insert the results.

4.6 VTKWriter class

The VTKWriter class uses a library called ”visit_writer” to convert data into legacy VTK data format. The library is written in C, so some specific C-related syntax had to be implemented. The VTK is written as a point mesh, with each point as a spacecraft location. Pointdata can then be any other set of observation geometry information, which is inherently fixed to the spacecraft location. Below is a code snippet from this class, which calls the visit_writer library and generates the VTK based on previously calculated arrays or geometry data.

```
1    // Information to pass to the visit_writer C-function
2    int nvars = 12;
3    int vardims[] = {3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1};
4    const char *varnames[] = {"ScXAxis", "ScYAxis", "ScZAxis",
5                              "InstrXAxis", "InstrYAxis", "InstrZAxis",
6                              "Velocityvec", "Earthvec", "Sunvec",
7                              "Velocitymag", "Distance", "Time"};
8    float *vars[] = { (float*)m_sc_x_axis, (float*)m_sc_y_axis,
9                      (float*)m_sc_z_axis,
10                     (float*)m_instr_x_axis, (float*)m_instr_y_axis,
11                     (float*)m_instr_z_axis,
12                     (float*)m_sc_obs_vel, (float*)m_sc_earth_pos,
13                     (float*)m_sc_sun_pos,
14                     (float*)m_sc_obs_vel_mag, (float*)m_sc_obs_pos_mag,
15                     (float*)m_et, };
16
17   // Write the data into a VTK file with visit_writer
18   // NOTE! Doubles may lose some precision when converting to floats
19     // (64-bit vs. 32-bit)
20   std::string vtkFileName = config->getNetcdfFileName();
21   write_point_mesh(vtkFileName.c_str(), 0, iterations,
22    (float*)point_location, nvars,
23       vardims, varnames, vars);
```

5 Libraries

In this work, several different C/C++ libraries were used. They are introduced below.

5.1 Boost::program_options

Boost is a library package widely used in C++ programming that provides a vast amount of different libraries from parallel computing to graphical interface. In this thesis, a library called program_options is used to provide configuration capability. Program_options is developed by Vladimir Prus, and it uses advanced C++ features such as template classes and smart pointers to deliver a flexible and robust library for configuring the software from a configuration file and from command line [http://www.boost.org/doc/libs/1_55_0/doc/html/program_options.html]. In Figure 3, --help argument is given to the tool to show available configuration parameters. Figure 7 shows an example of the configuration file that program_option parses.

Figure 4, command line configuration

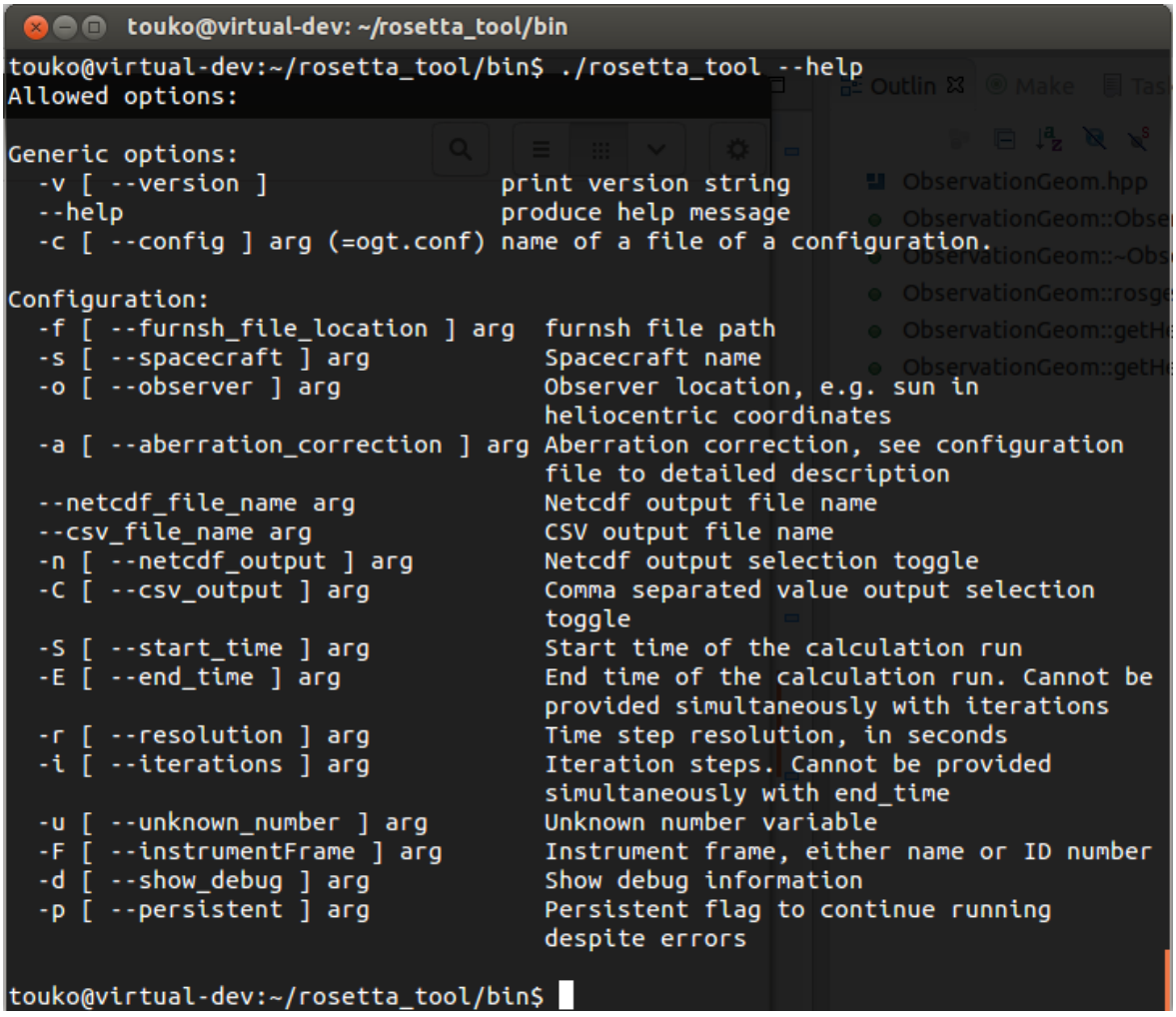


Figure 5, configuration file

```
133
134  ; Configure output file name for netcdf and csv. Standard file types
135  ; are .nc for netCDf and .csv for CSV files.
136
137  netcdf_file_name = ros_ephemeris.nc
138  csv_file_name    = ros_ephemeris.csv
139
140  ; This parameter determines which output is desired. Default is NetCDF
141  ; format, but .csv is also available. These can be on simultaneously,
142  ; so that the program gives both files. NOTE! This can be slow since
143  ; the calculations are run twice.
144
145  netcdf_output = TRUE
146  csv_output    = FALSE
147
148  ; Start and end-times of the time-series calculation. Either a start-time and
149  ; iterations count can be provided or start and end-time, so that iterations
150  ; will be calculated according to the time bounds and resolution.
151  ;
152  ; Resolution is in seconds. Time is in UTC, YYYY-MM-DDThh:mm:ss.s format.
153
154  start_time = 2007-07-16T19:20:30.45
155  resolution = 86400
156
157  ; Provide either end_time or iterations. Values on both triggers an error.
158  end_time   = 0
159  iterations = 100
160
161  ; Available dimensions (data columns). FALSE = not available,
162  ; TRUE = available. The order of dimensions in the output file is not
163  ; configurable at the moment. NOTE! Cosima instrument FOV is almost directly
164  ; along the boresight of the SC Z-axis, therefore the SC Z-axis can be
165  ; assumed to be the instrument boresight. Here other axises are disabled
166  ; by default.
167
168  scSunPosX = true
169  scSunPosY = true
170  scSunPosZ = true
```

5.1.1 Changes to program_options

By default, program_options uses hashtag (#) to identify comment lines in the configuration file. This is fine for stand-alone software, but in this case, a semicolon (;) is much better identifier for a comment line, since it is the default for PHP configuration file. Since the software might be used as a web service in the future, a modification was made to the library in order for it to parse semicolons as comments instead of hashtags. Configuration file with semicolons can be seen in a previous chapter, in Figure 7.

5.2 NASA SPICE

NASA has been developing its own open-source ancillary data system for a long time, and the result of this development work is the SPICE toolkit. It is available in several languages, including C. The C implementation is called CSPICE and it provides the functionality to calculate all the geometric needs in this thesis.

Spacecraft location and orientation in space can be initially described with three vectors: position and velocity vectors relative to an observing body, and a unit vector of the spacecraft attitude in a specific reference frame.

As spacecraft dynamics problems generally involve the use of several reference frames, it is necessary to know how to describe reference frame relations and transform vector coordinates between frames [De Ruiter 2013, p. 11]. The SPICE toolkit addresses these issues by introducing kernel files.

5.2.1 Kernels overview

SPICE toolkit does not have any knowledge of celestial bodies on its own, so a mechanism to introduce this data to the toolkit is required. SPICE kernel files are low-level ancillary data files that can be either in ASCII or binary format, and are fully portable. There are 11 different types of kernels available [http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/08_intro_to_kernels.pdf], most common of which are the position and attitude kernel files, namely the SPK and CK kernels. In spaceflight terminology, attitude means the orientation of the spacecraft in space. These kernels can be loaded into the SPICE toolkit either individually or in batches in so-called metakernels, which include the kernel references that link together sets of kernels related to some event, mission or planning activity. Kernels in ASCII format are simple text files that include variable assignments and some relevant metadata in the form of comments. Usually the user doesn't have to look into the kernels themselves since all the information is extracted by the toolkit.

5.2.2 Producing kernels

ASCII (text) kernels can be produced with a regular text editor, taking into account several syntax limitations. Most notable of these are:

- Only printable characters (ASCII values 32-126)
- Each line must be terminated by appropriate EOL
- 132 characters max. line length
- max. 32 characters variable names
- Variable names cannot include space, period, parenthesis, equals sign or tab

The text kernel syntax is otherwise very simple, consisting only of several markers that separate data blocks from metadata (comment) blocks. An example of this is below.

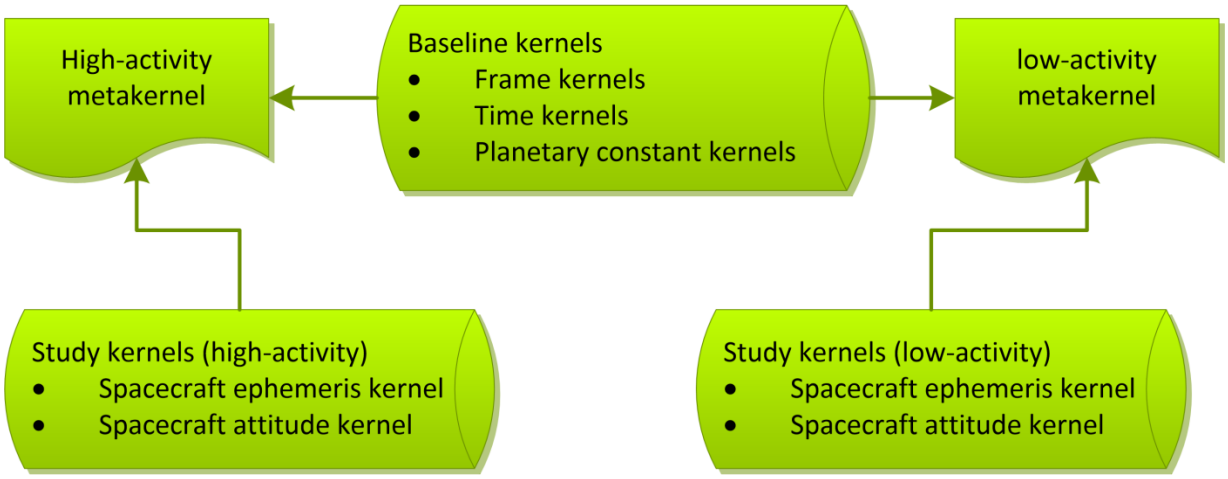
```
1 \begindata
2
3 NAME          =  'Rosetta'
4 DATAPOINT     = ( 0, 0.345, 0.356 )
5
6 \begintext
7
8 Comments go here without any special notation. Everything after the begintext
9 marker is considered metadata, until another begindata marker initializes
10 another data block.
11
12 \begindata
13
14 DATAPOINT += ( 0, 0.346, 0.360 )
```

Binary kernels are generated with utility programs, or directly via the toolkit API. This is useful when integrating SPICE with planning software. Binary kernels also include describing metadata in a somewhat readable manner; however it is not intended to read this information by directly opening the binary file. There are utility tools for extracting the metadata.

5.2.3 Metakernels

A metakernel, as stated before, tells the SPICE toolkit to load all the kernels in the metakernel file. These metakernels can be used, for example, in a trajectory study for high comet activity trajectory, as in Figure 7.

Figure 6. Using metakernels in trajectory studies



5.2.4 SPICE Toolkit example

In the following example code, position and velocity vectors are calculated for a spacecraft relative to the Sun. It assumes message and config objects are retrieved earlier.

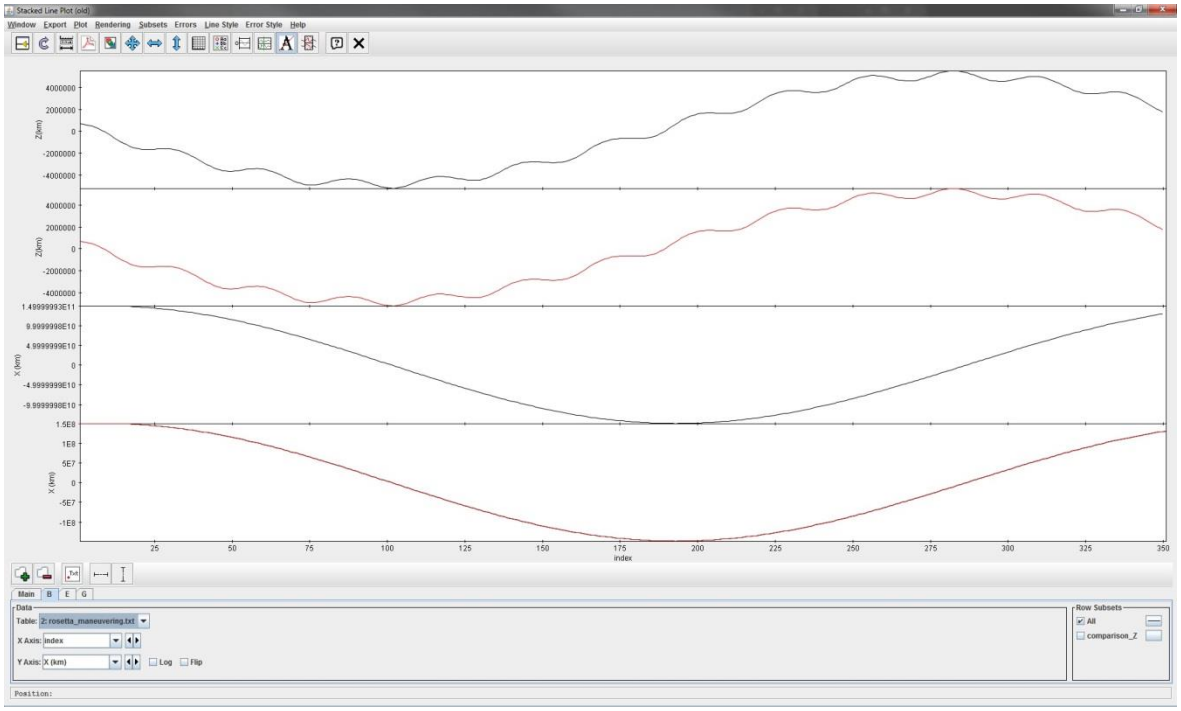
```
#include "SpiceUsr.h"

    // Configure spice error messages to show traceback and long messages
    errprt_c (  "SET", 100, "NONE, TRACEBACK, LONG" );

    // Load the kernels into spice runtime memory.
    furnsh_c(config->getFurnshFileLocation().c_str());

    // Resolve spacecraft name provided by user to NAIF ID number
    int spacecraftID = 0;
    int IDFound = 0;
    bodn2c_c(config->getSpacecraft().c_str(), &spacecraftID, &IDFound);

    // If resolving did not find the ID (implicit cast to bool).
    // Usually the SPICE routine takes care of reporting.
    if (!spacecraftIDFound) message->reportDebug("spacecraft name not found");

    // Get the startTime as an UTC string
    string inputEphemerisString = config->getStartTime();

    // Convert UTC to ephemeris time
    utc2et_c(inputEphemerisString.c_str(), &et);

    // Spice variables
    SpiceDouble   state[6];
    SpiceDouble   lt;

    //
    // S/c position and velocity relative to the Sun in ecliptic frame
    // can be computed with a single call to spkezr_c. The call returns a
    // 6-element state vector, the first three elements of which contain
    // position and the last three -- velocity. The magnitudes can be
    // computed by calling vnorm_c for each of the two vectors.
    //
    spkezr_c( config->getSpacecraft().c_str(), et, config->
    getReferenceFrame().c_str(), config->getAberrationCorrection().c_str(),
    "SUN", state, &lt );

    vequ_c  ( &state[0], sc_sun_pos );
    *sc_sun_pos_mag = vnorm_c ( sc_sun_pos );

    vequ_c  ( &state[3], sc_sun_vel );
```

6 Verification and validation

It is important to verify the accuracy of the calculated results compared to a reference. Since the core calculations are made with an external toolkit, it is left to validate that the toolkit is used correctly and no data-loss occurs. This is achieved by comparing results from a reference implementation of SPICE toolkit into the implementation described here. The reference implementation chosen was the "WebGeocalc" tool by NASA, which is a simple GUI interface to the toolkit.

To validate all output parameters, two tests were conducted and the outputs compared with TOPCAT and WinMerge programs. The first Earth-Sun state vector test validates distance, relative velocity and position parameters. Second test calculated the orientation of Rosetta spacecraft relative to the Ecliptic J2000 reference frame, thus validating frame transformation algorithms.

Figure 7. Comparing values to a reference implementation



7 Use cases

Several different use cases were identified and are described below. Users of the tool can be either from the scientific community or from education. Since it requires knowledge of orbital calculations and has a command-line interface, most of the users will likely be in the scientific community.

7.1 Spacecraft data-analysis

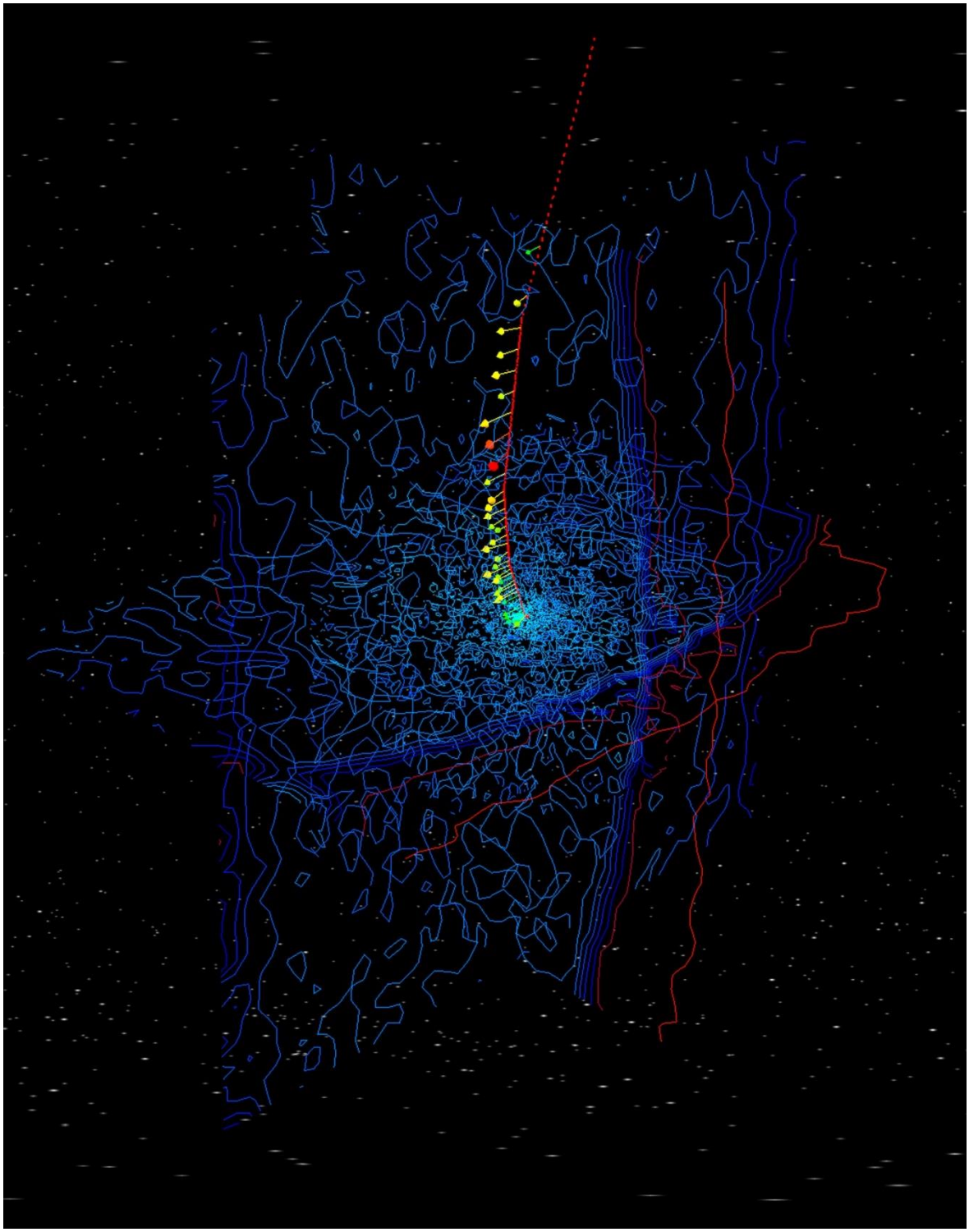
The tool can be used for regular mission analysis. In this use case, it is combine simulation model with instrument data and trajectory. In Figure 7, the sliced contour plot (blue and red) indicates simulated data of solar wind interaction with a comet. As seen from the plot, solar wind is flowing from the right-hand-side of the plot, and creates a

bow-shock region as it encounters particles and plasma surrounding the comet nucleus, often also referred as the coma.

An approach trajectory is calculated for the Rosetta spacecraft, indicated by red dots. These are data-points for which the precise location is calculated with the same time-step. As can be seen from the plot, spacecraft velocity w.r.t. to the comet is slowing as it approaches closer, indicated by the density of the red dots increasing.

Here, the magnetic field measurements are taken from the simulation model as no real data was available at the time. Magnetic field orientation is indicated by vectors pointing out from the spacecraft location, and the magnitude by vector colour. Once real data becomes available, it can be plotted in the same window on top of the simulated data, giving instant insight on the accuracy of the underlying model.

Figure 8. Simulation model with Rosetta trajectory (red dots) and magnetic field measurements (arrows)



7.2 Deep space mission analysis

The tool can be used in conjunction with flight dynamics tools in deep space mission analysis. A common workflow for this would be to export a flight dynamics scenario (such as a spacecraft orbit and a manoeuvre) into SPICE kernels and then use this tool to convert the scenario into VTK format. This allows deeper analysis of the trajectory with possible simulations that are not compatible with the used flight dynamics tool.

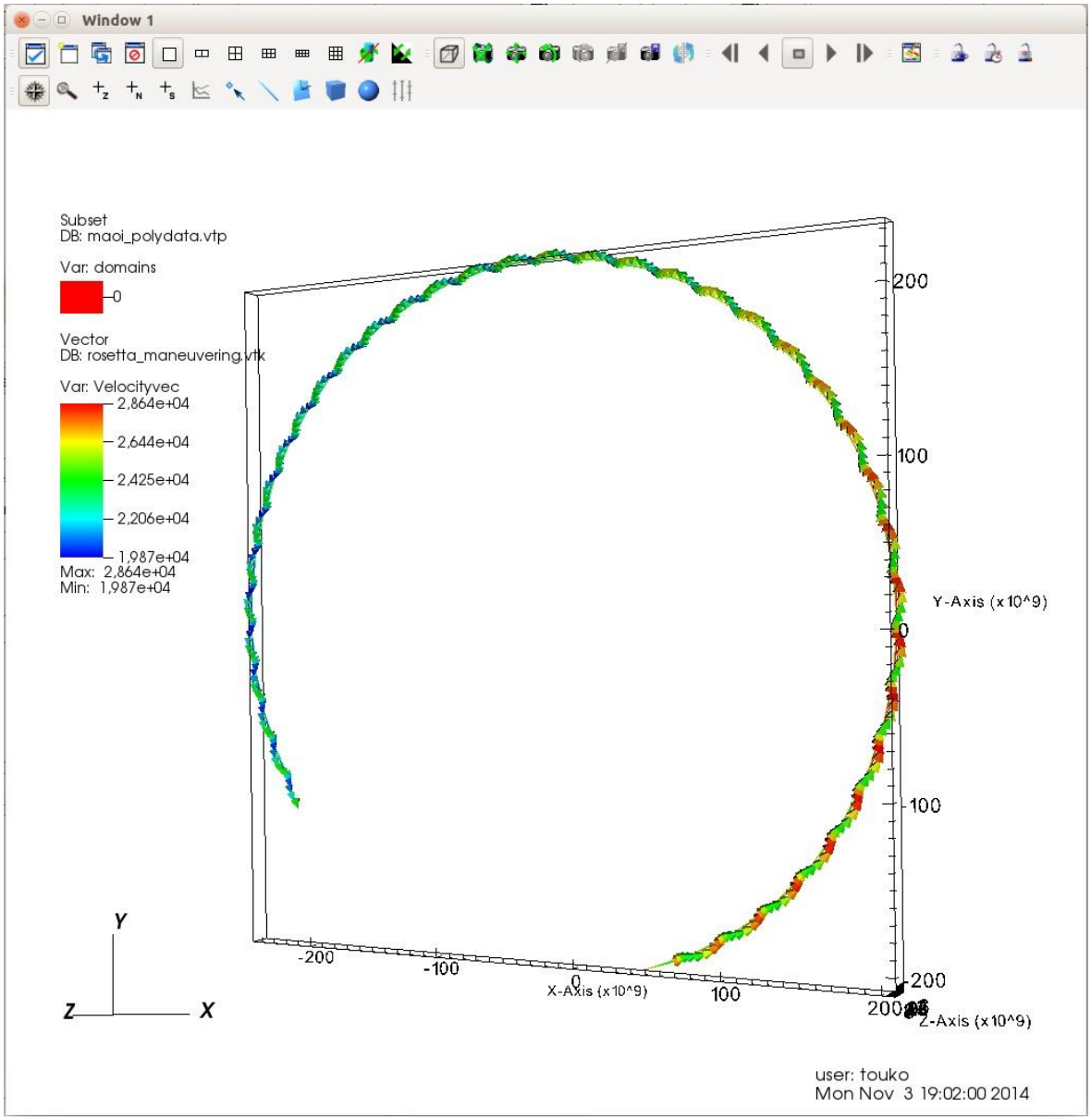
7.3 Asteroid hunting

New SPICE kernels are generated by the NASA HORIZONS system once orbital parameters for newly found asteroids are released. These asteroid kernels can then be assembled into different metakernels to study potential asteroid missions. Asteroids are interesting targets especially for smaller and commercial missions for their business potential as a raw materials source. With this tool, it is easy to generate visualization of these trajectories.

7.4 Education

By inserting uncommon observer-target pairs it is easy to demonstrate orbital dynamics. For example, setting observer as the sun and target as a Phobos, the Mars moon orbital period and distances can be easily comprehended.

Figure 9. Phobos relative to the Sun



8 Discussion

In the discussion section, general usefulness and future developments of the tool is presented. Three development suggestions were identified.

8.1 Usefulness of the tool

As there are already several implementations for SPICE toolkit, none of them is able to provide as much configuration freedom and lack VTK export. At FMI, the reliance on ready-made ephemeris sets meant limitations in data-analysis capabilities. This especially amplified when new simulation models were made for Rosetta. The ability to

generate VisIt compatible trajectories is of great benefit to the simulation developers and users.

8.2 Learning outcome

The purpose of this thesis was to familiarize the author with the SPICE toolkit and the C++ programming language. Indeed the outcome of this thesis was the rather deep knowledge on the generation of this type of spacecraft trajectory files and their usage to calculate positions, attitudes and other parameters required in observation geometry calculations.

8.3 Development suggestions

The tool is good as it is as a utility tool, but there is great potential in growing it further with new interfaces and functionality to improve its usability. Below is listed some development suggestions for further work.

8.3.1 Graphical User Interface

The current interface is purely based on a configuration file and a command-line interface. In the future, a GUI interface is inevitable if the configuration options and functionality grows. The GUI could also be integrated into VisIt, so that the interaction between data and visualization becomes more natural, or even real-time.

The tool was written with a simple API to be used as a module in other software. This feature can also be used in the development of the GUI, for example with the C++ Qt framework. However, interoperability with any programming language is achieved by running the tool programmatically, as all configuration parameters can be adjusted with arguments.

8.3.2 ESA PSA interface

Currently the interface to retrieve instrument data is not clearly defined. Future developments could see integration to the ESA Planetary Science Archive (PSA) database

services, which would allow the automatic retrieval of calibrated instrument data. This is especially important in the validation of the simulation models.

8.3.3 Simulation model database interface

The current implementation of the tool does not interact with simulations, but expects the user to do it manually in VisIt. Future development could include interfaces to several Simulation Model Databases (SMDB's) which would allow more streamlined plot generation and less manual work.

9 References

NASA, 2013. Introduction to Kernels. URL: http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/08_intro_to_kernels.pdf Accessed 4.11.2014

De Ruiter, 2013. Attitude Dynamics and Control: An Introduction. p. 11

Prus, V. 2004. Chapter 21. Boost.program_options. URL: http://www.boost.org/doc/libs/1_55_0/doc/html/program_options.html Accessed 20.11.2014

LLNL, 2014. VisIt software. URL: https://wci.llnl.gov/codes/visit/ Accessed 20.11.2014

10 Appendices

Appendix A. Source Code.