Overview

ORSA is a *Free Software* project developed under *Linux* and written in **C++**. The main purposes of the ORSA project are:

- create a *common infrastructure* among the existing Celestial Mechanics programs;
- implement all the *state of the art* orbit integration algorithms, with concerns on **accuracy**, **performance** and interaction options;
- develop command line and graphical applications;
- support for *High Throughput Computing*, *LSF* and *GRID*;
- use an object oriented programming language (C++), CVS and auto-configuration tools to achieve a good design, rapid development and maintainability;
- keep the ORSA project under **GPL**.

The main components of the project are:

- **liborsa** the main numerical library, where all the base classes like *Units*, *Vector*, *Body*, *Interaction*, *Orbit*, *Integrator*, *Frame*, *Evolution* and *Universe* are implemented;
- **libxorsa** the graphical library, which implements all the components of the Graphical User Interface;
- **xorsa** the main graphical application, which provides most of the functionalities of both liborsa and libxorsa.

The project is developed using freely available libraries, like *FFTW*, *Qt*, readline and the *GNU Scientific Library*.

Basic concepts

We briefly describe here the concepts ORSA is based on. Most of them are represented as C++ classes, but this is just an implementation detail for programming–minded readers.
The **Universe** is the first class created by a program using ORSA, and keeps all the informations regarding interactions, units, bodies, evolutions and reference systems. Like other classes, the role of Universe is very intuitive and can be guessed by the class name. There are two type of universes: *Real* and *Simulated*. The Real Universe is of choice for investigations in the Solar System: the planets and asteroids can be easily imported from ephemeris files, and the time is in years, months and days. The Simulated Universe is the most general one, and allows the user to set all the bodies from scratch; in this case the time starts from zero and cannot be converted into a date.
The **Vector** class represents a 3D vector, and implements all the basic operations.
The **Body** class contains all the properties of a body, like mass, name, position and velocity.
The **Units** class provides a coherent implementation of physical quantities and constants. The base units used in the internal computations are defined at the creation of the Universe.

the ORSA applications. It implements all the basic physical classes described before, and also implements all the file import/export methods, the configurations–related classes, the implementation of the Analysis classes like the *Frequency Map Analysis*, *Lyapunov* and *Mean Motion Resonances*, and many other tasks.
This library uses other numerical libraries to perform part of the computations, like the *FFTW* library of the Fast Fourier Transforms, the *GNU Scientific Library* for minimization problems, and makes use of the C++ Standard Library for containers and algorithms.
The liborsa library provides support for *High Throughput Computing* (like Condor), *LSF* and *GRID*.

libxorsa

This is the graphical library, used only by the graphical applications. It implements more than fifty different widgets, like object browsers, editors and generators, plotting widgets, units and date conversion tools, 3D OpenGL viewers, and many others. This library makes an heavy use of the liborsa library, where all the data structures and all the basic methods are defined.
The libxorsa library is based on the Trolltech's Qt library, which allows, among the other benefits, to write multi-thread applications. This means that it is possible to perform a numerical integration while observing the same integration in a 3D viewer, or start more than one integration from the same application.

xorsa

This is the main application of the ORSA framework. With xorsa it is possible to perform all the operations allowed by the ORSA framework, as the base libraries and xorsa are developed concurrently.

Tutorials

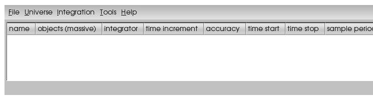
Together with the libraries and the main application, we have developed some tutorials, which allow to learn quickly all the basic functionalities of the framework. They are usually very short programs focusing on a particular task, like basic system numerical integration, file import, analysis tasks and others.

name, an interaction, a set of bodies, and
sampling period.

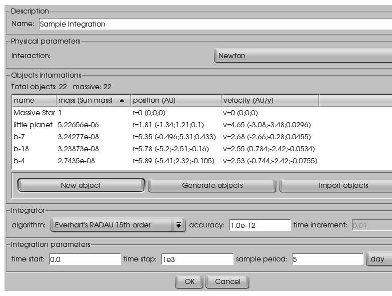
- When the integration is finished, but als
possible to plot the osculating elements
system in a 3D OpenGL viewer, with an
reference system, and other options.
- It is also possible to analyze the integrat
- The whole session can be saved as a sin
automatically compressed for a minima
restored for further investigations, or se

Gallery

Here we display some of widgets used in

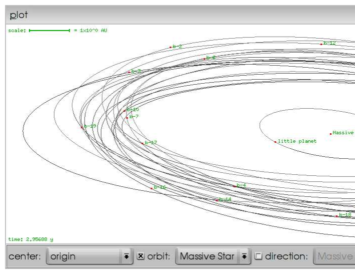


name	objects (massive)	integrator	time increment	accuracy	time start	time stop	sample perio

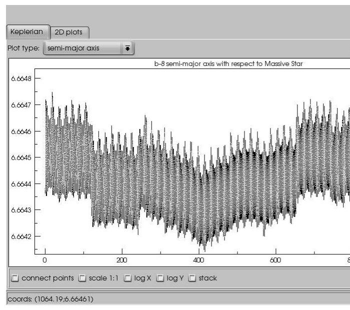

Description
Name: Sample Integration
Physical parameters
Interaction: Newton
Objects informations
Total objects: 22 massive: 22

name	mass (Sun mass)	position (AU)	velocity (AU/y)
Massive Star	1	r=0 (0;0;0)	v=0 (0;0;0)
little planet	5.22656e-06	r=1.81 (-1.34;1.21;0.1)	v=4.65 (-3.08;-3.48;0.0296)
b-7	3.24277e-08	r=5.35 (-0.496;5.31;0.433)	v=2.68 (-2.66;-0.28;0.0455)
b-18	3.23873e-08	r=5.78 (-5.2;-2.51;-0.16)	v=2.55 (0.784;-2.42;-0.0534)
b-4	2.7435e-08	r=5.89 (-5.41;2.32;-0.105)	v=2.53 (-0.744;-2.42;-0.0755)

Integrator
algorithm: Everhart's RADAU 15th order accuracy: 1.0e-12 time increment: 0.01
Integration parameters
time start: 0.0 time stop: 1e3 sample period: 5 day



center: origin [✓ orbit:] Massive Star direction: Massive



Plot type: semi-major axis

connect points scale 1:1 log X log Y stack
coords: (1064.19;6.66461)

Conclusions

ORSA is an evolving project whose goal is
ORSA is *Free Software*: feel free to use and
http://orsa.sourceforge.net.