

Orbiter: A Free Spacecraft Simulation Tool

Martin Schweiger
Department of Computer Science
University College London
www.orbitersim.com



2nd ESA Workshop on Astrodynamics
Tools and Techniques
ESTEC, Noordwijk
13-15 September 2004

Contents

- Overview
- Scope and limitations
- Some Orbiter features:
 - Time propagation
 - Gravity calculation
 - Rigid-body model and superstructures
- Orbiter Application Programming Interface:
 - Concept
 - Orbiter instrumentation
 - The VESSEL interface class
- New features:
 - Air-breathing engines: scramjet design
 - Virtual cockpits
 - New visual effects
- Connectivity and data exchange
- Summary and future plans
- Demonstration



Overview

- ❑ Orbiter is a real-time space flight simulation for Windows PC platforms.
- ❑ Modelling of atmospheric flight (launch and re-entry), sub-orbital, orbital and interplanetary missions (rendezvous, docking, transfer, swing-by etc.)
- ❑ Newtonian mechanics, rigid body model of rotation, basic (static) atmospheric flight model.
- ❑ Planet positions from public perturbation solutions. Time integration of state vectors or osculating elements.
- ❑ Developed since 2000 as an educational and recreational application for orbital mechanics simulation.
- ❑ Written in C++, using DirectX for 3-D rendering. Public programming interface for development of external module plugins.
- ❑ With an increasingly versatile API, development focus is beginning to shift from the Orbiter core to 3rd party addons.



Scope

- ❑ Launch sequence from surface to orbital insertion (including atmospheric effects: drag, pressure-dependent engine ISP ...)
- ❑ Orbital manoeuvres (alignment of orbital plane, orbit-to-orbit transfers, rendezvous)
- ❑ Simultaneous simulation of multiple spacecraft.
- ❑ Vessel-to-vessel approach and docking. Building of superstructures from vessel modules (including simple rules for updating the rigid-body model).
- ❑ Release and re-capture of satellites.
- ❑ Re-entry.
- ❑ Interplanetary transfers (including Hohmann orbits, slingshot manoeuvres)
- ❑ Atmospheric flight (aerodynamic flight model, airfoil definition, runway takeoff/landing, air-breathing engines)
- ❑ Simulation and testing of historical, present and hypothetical spaceflight concepts.



Limitations

- ❑ Stability of time propagation: numerical accuracy limited by frame refresh rate (physics engine and graphics subsystem competing for clock cycles)
- ❑ Flight model: no native support yet for radiation pressure, micro-drag at high altitude. Simple rigid-body model (no native support for tethers, internal mass distribution changes ...)
- ❑ Simple atmospheric flight model
- ❑ No collision detection
- ❑ No damage modelling
- ❑ No native multi-user support



Time propagation

Real-time simulation with time acceleration up to 10^4 and variable step length determined by processor speed, graphics load, simulation complexity etc.

Method 1:

Semi-analytic perturbation solutions for celestial bodies (VSOP87, ELP2000 ...)

Method 2:

Propagation of state vectors $\mathbf{r}(t_i) \rightarrow \mathbf{r}(t_{i+1})$, $\mathbf{v}(t_i) \rightarrow \mathbf{v}(t_{i+1})$ with 4th order Runge-Kutta.

Method 3:

Updating elements of osculating orbit from perturbations of the primary gravitational field.



Gravity calculation

Orbiter accommodates perturbations of the radial symmetry of gravitational potential sources in a single (polar) dimension using a harmonic series:

$$U(r, \phi) = \frac{GM}{r} \left[1 - \sum_{n=2}^N J_n \left(\frac{R}{r} \right)^2 P_n(\sin \phi) \right]$$

with Legendre polynomial P_n of order n , and perturbation coefficients J_n . Number of terms N is adjusted automatically as a function of distance r .

Earth	$J_2 = 1082.63$] x 10 ⁻⁶
	$J_3 = -2.51$	
	$J_4 = -1.6$	
	$J_5 = 0.13$	
Mercury	$J_2 = 60$	
Venus	$J_2 = 27$	
Mars	$J_2 = 1964$	
Jupiter	$J_2 = 14750$	
Saturn	$J_2 = 16450$	
Uranus	$J_2 = 12000$	
Neptune	$J_2 = 4000$	

A full spherical harmonics expansion of the field perturbations in both polar (ϕ) and azimuth direction (λ) is planned for a future version.



Rigid-body model, composite structures

Orbiter uses a simplified model of rigid body motion to construct superstructures by connecting vessels.

Given Euler's equation for angular motion, assuming diagonalised inertia tensors with PMI J_x, J_y, J_z :

$$J_x \dot{\omega}_x = M_x - (J_z - J_y) \omega_y \omega_z$$

$$J_y \dot{\omega}_y = M_y - (J_x - J_z) \omega_z \omega_x$$

$$J_z \dot{\omega}_z = M_z - (J_y - J_x) \omega_x \omega_y$$

we represent each vessel by 6 samples $p_{1,2} = (\pm x, 0, 0)$, $p_{3,4} = (0, \pm y, 0)$, $p_{5,6} = (0, 0, \pm z)$ with

$$x = \frac{1}{2} \sqrt{|-J_x + J_y + J_z|}$$

$$y = \frac{1}{2} \sqrt{|J_x - J_y + J_z|}$$

$$z = \frac{1}{2} \sqrt{|J_x + J_y - J_z|}$$

The samples p_i for each vessel are transformed into a common reference frame p_i' , given by the superstructure connectivity, and transformed PMI are constructed:

$$J'_x = m \sum_i y_i'^2 + z_i'^2$$

$$J'_y = m \sum_i x_i'^2 + z_i'^2$$

$$J'_z = m \sum_i x_i'^2 + y_i'^2$$

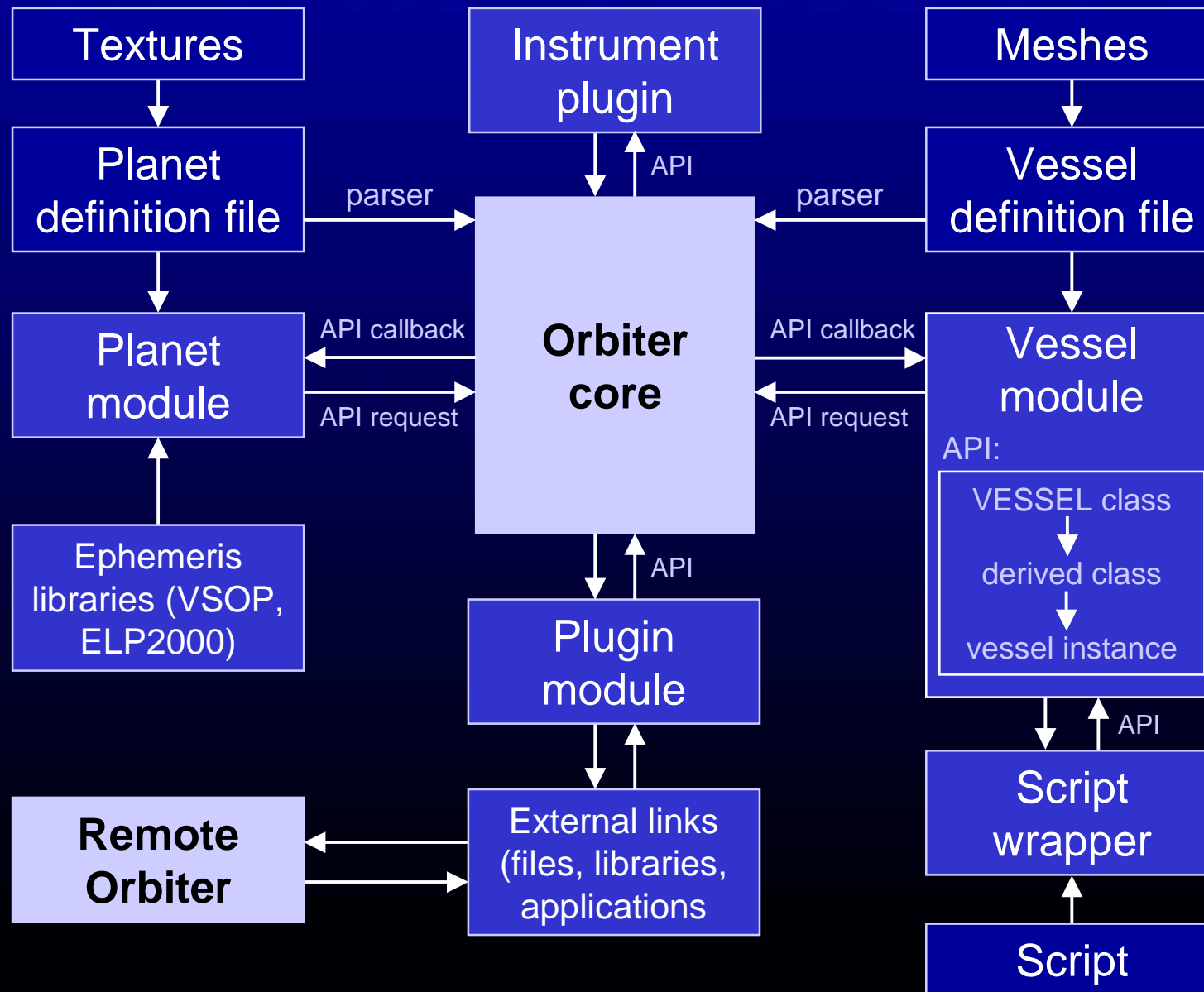
The PMI $J^{(S)}$ of the superstructure are then given by collecting all vessel contributions:

$$J^{(S)} = \sum_k J'^{(k)}$$

No off-diagonal elements are considered in $J^{(S)}$.



Orbiter module design and API



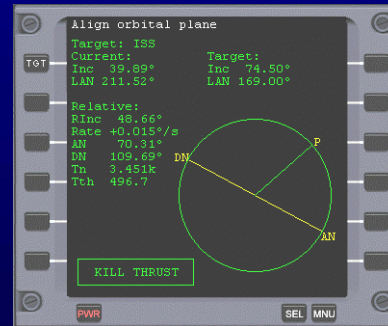
Orbiter instrumentation

Multifunctional display (MFD) concept: seamless extension of instrumentation functionality via plugin MFD modes.

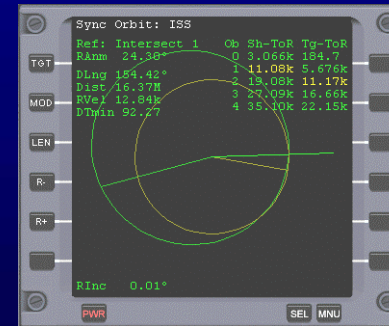
Generic instruments (selection):



Surface-relative and atmospheric parameters.



Align orbital plane with a target orbit at a node.



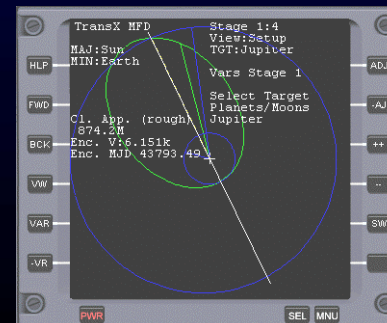
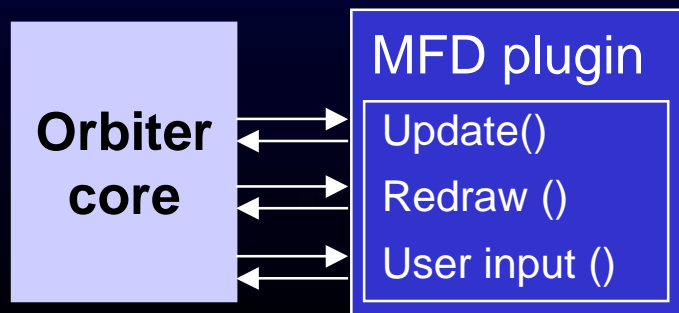
Rendezvous with target object.



Line up docking approach path.



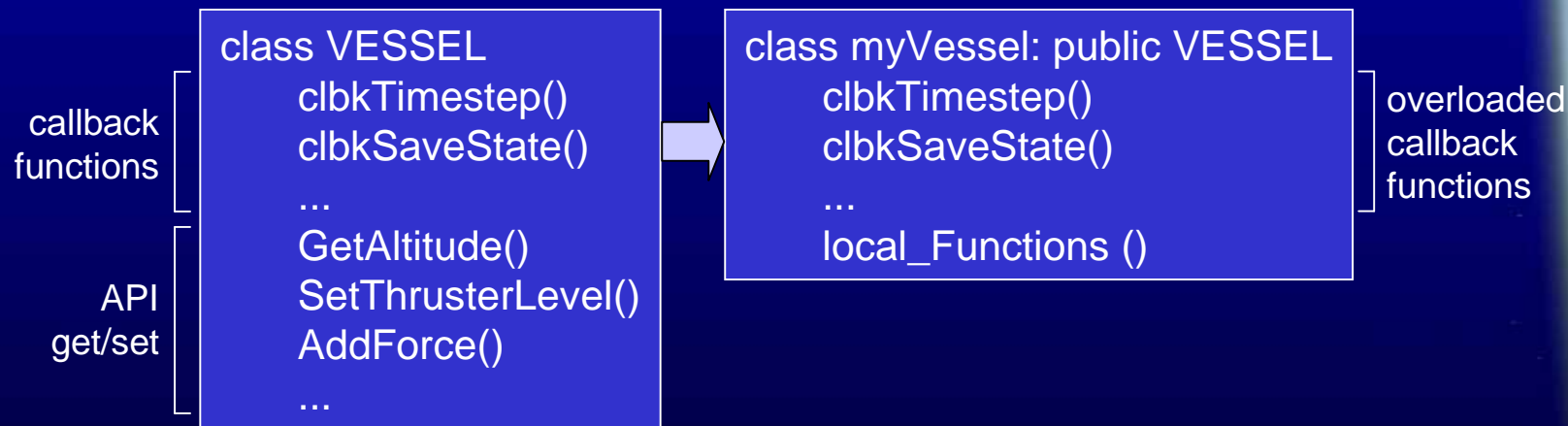
Drop-in instruments from plugin modules:



TransX MFD mode: interplanetary transfer calculation with patched cone approach. (courtesy Duncan Sharpe)

Orbiter API: The VESSEL interface

The VESSEL class is a generic interface between Orbiter and a vessel implementation.



Using Orbiter's built-in flight model:

Setup:

```
Define_thruster (r, d, f0, isp, ...)  
Define_airfoil (r, cL(...), cD(...), ...)
```

Local calculation, bypassing Orbiter's flight model:

Time step:

```
Get_Positions ()  
Get_Atmospheric_data ()  
(F, r) = local_Calculate_forces ()  
AddForce (F, r)
```

The vessel designer has a choice of using built-in flight models, or implementing a local model (using Orbiter as a visualisation framework only).



Air-breathing engines: scramjet (1)

Scramjet design is an example for implementing a feature entirely externally without native support in the Orbiter core.

Ideal scramjet: temperature and pressure relationships

Diffuser: isentropic
compression

$$T_d = T_\infty \left(1 + \frac{\gamma - 1}{2} M_\infty^2 \right) \quad p_d = p_\infty \left(\frac{T_d}{T_\infty} \right)^{\gamma/(\gamma-1)}$$

Combustion chamber:
isobaric expansion

$$T_b = \max(T_{b0}, T_d) \quad p_b = p_d$$

Exhaust nozzle:
isentropic expansion

$$T_e = T_b \left(\frac{p_e}{p_b} \right)^{(\gamma-1)/\gamma} \quad p_e = p_\infty$$

Jet engine propulsion thrust balance equation:

$$F = (\dot{m}_a + \dot{m}_f)v_e - \dot{m}_a v_\infty + (p_e - p_\infty)A_e$$

where \dot{m}_a and \dot{m}_f are the air and fuel mass rates, respectively, v_e and v_∞ are the exhaust and freestream velocities, and A_e is the exhaust cross section.



Air-breathing engines: scramjet (2)

Specific thrust is given by $\frac{F}{\dot{m}_a} = (1 + D)v_e - v_\infty$

where $D = \dot{m}_f / \dot{m}_a$ is the *fuel-to-air ratio*.

The amount of fuel burned in the combustion chamber must be adjusted so that the burner temperature limit is not exceeded. This leads to the following expression for D :

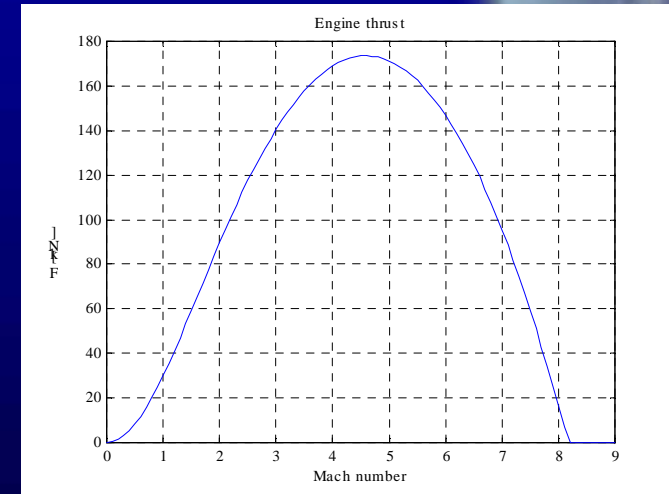
$$D = \frac{T_b - T_d}{Q/c_p - T_b}$$

where Q is a fuel-specific heating value and c_p is the specific heat at constant pressure, given by $c_p = \gamma R / (\gamma - 1)$.

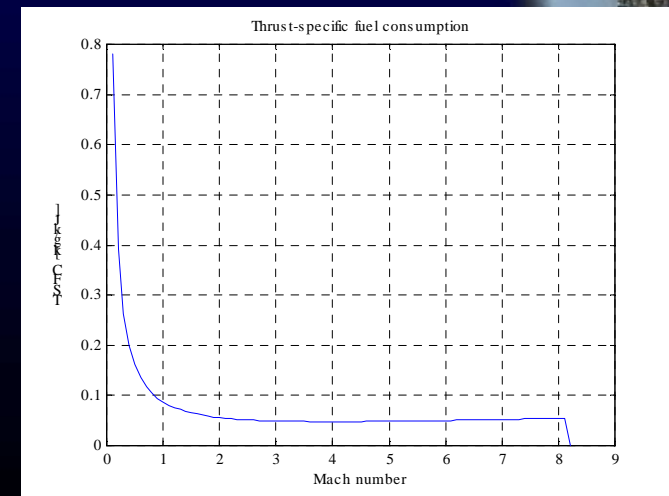
The exhaust velocity v_e can be obtained from the energy balance

$$c_p T_b = c_p T_e + v_e^2 / 2$$

Thrust vs. Mach number:



TSFC vs. Mach number:



Virtual 3-D Cockpit



- ❑ Support for 3-D virtual cockpit view
- ❑ Head rotation improves situational awareness
- ❑ “Eye-neck” offset generates movement parallax
- ❑ Camera reference point and rotation ranges defined by API calls
- ❑ Dynamic display updates
- ❑ Mouse-operated instruments
- ❑ Viewpoint-corrected HUD display



3-D artwork courtesy Roger Long

New visual effects (1)



- ❑ Improved rendering of atmospheric haze from high altitude
- ❑ Additional configuration parameters for colour distribution



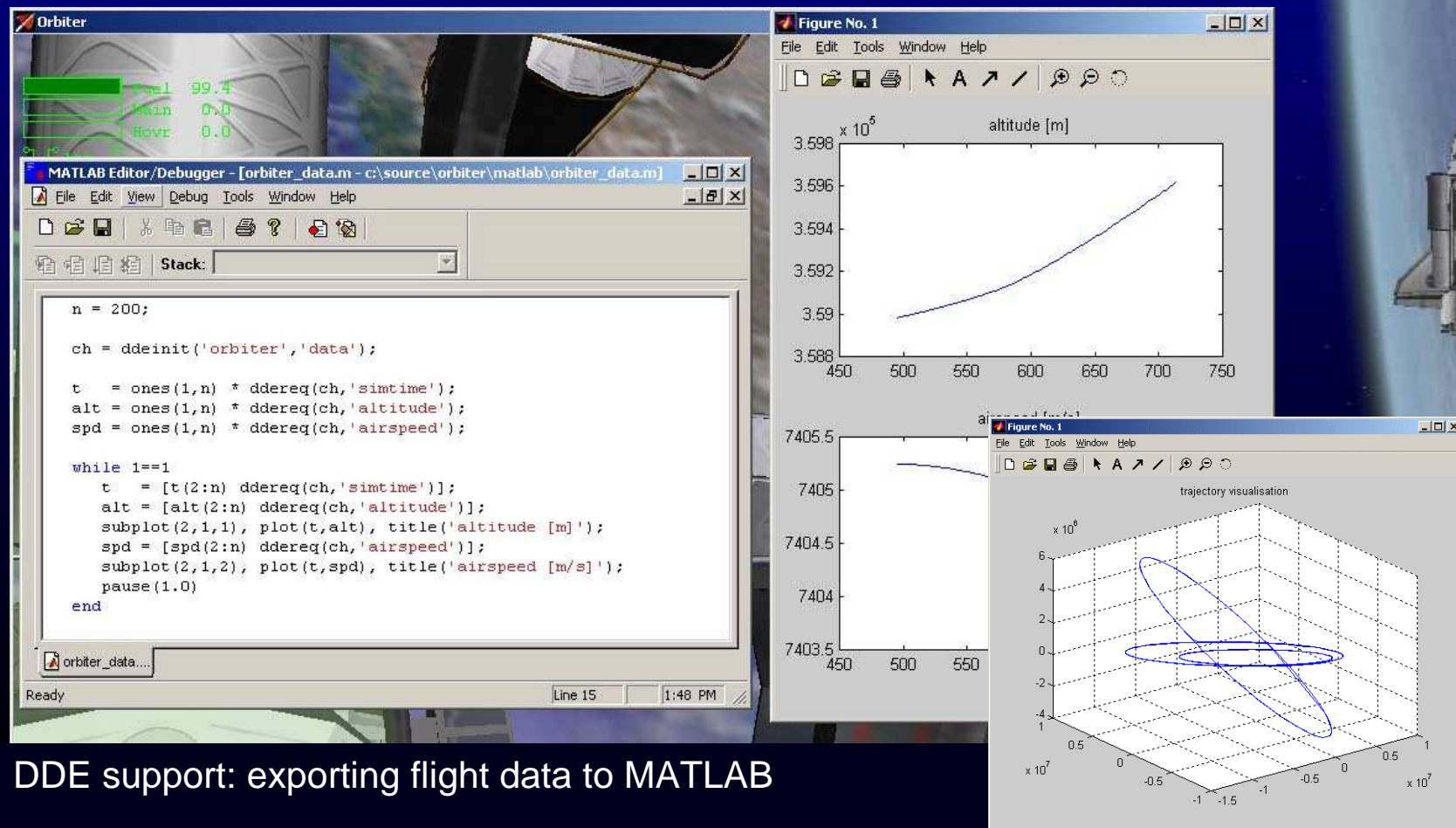
- ❑ Rendering of objects through atmosphere layers is now additive.



Integration and data exchange

Data logging: flight data can be extracted for analysis by:

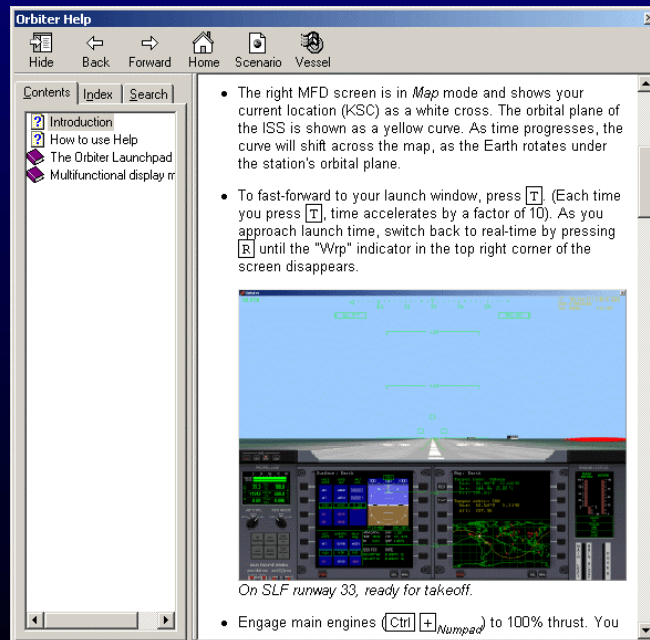
- ❑ Using built-in flight data logging facilities
- ❑ Writing custom data extraction modules using the API interface
- ❑ New: Using DDE (dynamic data exchange) protocol



DDE support: exporting flight data to MATLAB

Orbiter as a teaching tool

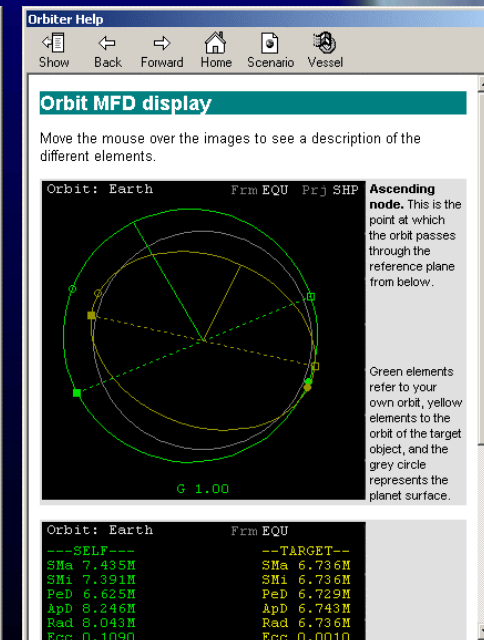
- ❑ New html-based help system (context-sensitive: scenario- and vessel-specific). Can be extended by 3rd party plugins.
- ❑ New "kiosk mode" for unsupervised use in public environments (limited simulation run time, automatic scenario selection).



Help system: scenario instructions



Help system: instrument layout and documentation



Help system: orbital mechanics primer

Summary

- ❑ Orbiter is an accessible tool for atmospheric, orbital and interplanetary space flight simulation.
- ❑ Combining a (moderately) accurate physics engine with 3-D rendering, its main application is as an educational or recreational tool.
- ❑ Diverse educational applications: maths, physics, astronomy, geography training, mission design/planning.
- ❑ The programming interface (API) is a versatile way to extend the core Orbiter functionality. Features not natively supported by the core can be added by external plugins.
- ❑ C++ API interface consists of more than 500 functions, methods, classes and callback hooks, including:
 - ❑ state vector updates for celestial bodies
 - ❑ spacecraft implementations
 - ❑ instrumentation
- ❑ Development of the core module is ongoing, and a growing set of 3rd party contributions is available.



Future developments

Some of the features planned for future releases include:

- Improvements of the flight model (stability of time integration, micro-drag, radiation pressure, atmospheric flight model).
- Use of Orbiter as a testbed for mission planning / trajectory optimisation.
- Damage and collision modelling.
- Multi-user support (simulation running on server continuously, clients connect temporarily).
- Planetary elevation modelling.



Acknowledgements

- Trajectory code Duncan Sharpe (TransX transfer trajectory plugin)
- Vessel code Radu Poenaru, Robert Conley
- 3-D Modelling Roger Long, Andrew Farnaby, Don Gallagher, Damir Gulesich, David Sundstrom, Jason Benson, Valerio Oss
- Planet textures James Hastings-Trew, Björn Jonsson, Dean Scott, Philip Stooke, Constantine Thomas, Robert Stettner, James Williams, Seth Hollingsead

The beta test team

The addon developer base

Web space sponsored by M6.net, avsim.com, and others



Resources

Orbiter main site (includes download links and related sites):

www.medphys.ucl.ac.uk/~martins/orbit/orbit.html

www.orbitersim.com

Contact:

martins@medphys.ucl.ac.uk

Orbiter is an un-funded, volunteer-based project.

Critique, suggestions and collaborations are very welcome!



< Orbiter Demonstration >

