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
• 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
• Orbiter as a teaching tool
• Summary and future plans
• Demonstration
Overview

- Orbiter is a real-time space flight simulation for Windows PC platforms.

- Modelling of atmospheric flight (launch and reentry), suborbital, orbital and interplanetary missions (rendezvous, docking, transfer, swing-by etc.)

- Newtonian mechanics, rigid body model of rotation, basic 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)
- 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, airbreathing engines)
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 $r(t_i) \rightarrow r(t_{i+1})$, $v(t_i) \rightarrow v(t_{i+1})$ 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 \( r \).

<table>
<thead>
<tr>
<th>Planet</th>
<th>( J_2 )</th>
<th>( J_3 )</th>
<th>( J_4 )</th>
<th>( J_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earth</td>
<td>1082.63</td>
<td>-2.51</td>
<td>-1.6</td>
<td>0.13</td>
</tr>
<tr>
<td>Mercury</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Venus</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mars</td>
<td>1964</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jupiter</td>
<td>14750</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saturn</td>
<td>16450</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uranus</td>
<td>12000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neptune</td>
<td>4000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A full spherical harmonics expansion of the field perturbations in both polar \( (\phi) \) and azimuth direction \( (\lambda) \) is planned for a future version.
Rigid-body model and 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 $J_x, J_y, J_z$:

\[
J_x \ddot{x} = M_x - (J_z - J_y) \omega_y \omega_z \\
J_y \ddot{y} = M_y - (J_x - J_z) \omega_z \omega_x \\
J_z \ddot{z} = M_z - (J_y - J_x) \omega_x \omega_y
\]

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

- Textures
  - Planet definition file
    - Planet module
      - Ephemeris libraries (VSOP, ELP2000)

- Instrument plugin
  - API callback
  - API request
  - API

- Orbiter core
  - API

- Meshes
  - Vessel definition file
    - Vessel module
      - API:
        - VESSEL class
          - derived class
            - vessel instance

- Plugin module
  - API

- External links (files, libraries, applications)

- Remote Orbiter
  - Script wrapper
  - Script

- External links (files, libraries, applications)
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:

- Orbiter core
  - MFD plugin
    - Update()
    - Redraw()
    - User input()

- 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.

```
class VESSEL
    clkTimestep()
    clkSaveState()
    ...
    GetAltitude()
    SetThrusterLevel()
    AddForce()
    ...
```

```
class myVessel: public VESSEL
    clkTimestep()
    clkSaveState()
    ...
    local_Functions()
```

Using Orbiter's built-in flight model:

Setup:
- Define_thruster \((r, d, f_0, isp, \ldots)\)
- Define_airfoil \((r, c_L(...) , c_D(...) , \ldots)\)

Local calculation, bypassing Orbiter's flight model:

Time step:
- Get_Positions()
- Get_Atmospheric_data()
- \((F, r) = \text{local}_\text{Calculate_forces}()\)
- 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
\[ T_d = T_\infty \left(1 + \frac{\gamma - 1}{2} M_\infty^2 \right) \]
\[ p_d = p_\infty \left(\frac{T_d}{T_\infty}\right)^{\gamma/(\gamma-1)} \]

Combustion chamber: isobaric expansion
\[ T_b = \max(T_{b0}, T_d) \]
\[ p_b = p_d \]

Exhaust nozzle: isentropic expansion
\[ T_e = T_b \left(\frac{p_e}{p_b}\right)^{(\gamma-1)/\gamma} \]
\[ p_e = p_\infty \]

Jet engine propulsion thrust 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 \( v_\infty \) 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

\[ c_p T_b = c_p T_e + v_e^2 / 2 \]
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.
New visual effects (2)

- Surface labels (launch sites, radio transmitters, user-defined)
- Surface shadow support for structures and vessels
Orbiter as a teaching tool (1)

- New "kiosk mode" for unsupervised use in public environments (limited simulation run time, automatic scenario selection).
Orbiter as a teaching tool (2)

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
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.
- 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.
- The API interface includes
  - 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).
- Damage and collision modelling.
- Multi-user support (simulation running on server continuously, clients connect temporarily).
- Elevation modelling of celestial bodies.
## Acknowledgements

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

The beta test team

The addon developer base

The sponsors M6.net, avsim.com
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

Critique, suggestions and collaborations are very welcome!
< Orbiter Demonstration >