IAC-04-P.P.05

"DIGITAL ORRERY" A GRAPHICAL SOLAR SYSTEM SIMULATION AS AN EDUCATIONAL TOOL

MARK CUMMINS
ESA STUDENT OUTREACH PROGRAM, IRELAND
MARK.CUMMINS@BALLIOL.OXFORD.OX.UK

Figure 1.1. A mechanical orrery.

Abstract

The goal of this project was to construct a graphical simulation of the solar system, intended for use by the general public. Though several such simulations exist, none combine intuitive presentation with true physical simulation. This report outlines the development of the back end of the simulation. Our goal was to achieve a flexible, fast and accurate simulation suitable for real-time graphical display. Additionally, the simulation provides support for real-time modification of the simulated system to facilitate a richly interactive user experience. As part of the development process we design a tailored force determination algorithm that offers a significant speed advantage over direct approaches. We also investigate several numerical integration schemes and provide support for various appealing visualizations of our data. The resulting program is unique in combining intuitive 3D presentation with a flexible and accurate simulation engine, and we hope it will prove to be a useful educational tool for promoting deeper understanding of the behaviour of gravitational systems.

1. Introduction

The objective of this project was to produce a "digital orrery". Originally an orrery was a device that illustrated the motion of the planets and their moons by means of an arrangement of mechanical gears. Our aim is to create a similar simulation on a PC – a program that will display the motion of the solar system in a way that is attractive and appealing to a casual observer. We hope that this will prove to be an engaging educational tool for the exploration of the physics of gravitational systems.

1.1. **Existing Systems.** Not surprisingly, there already exist several programs that deserve the name "digital orrery". Many of these are Open Source. The two most prominent open source programs in this area are *Celestia* and *ORSA*. To explain how they differ requires a short introduction to how the simulation engine of a program of this type can be constructed. There are essentially two strategies.

One way is to make use of predefined paths for bodies in the system. This is the approach used by *Celestia*. The advantage of this approach is that it is computationally very cheap. The problem, however, is that the motion of the objects in the simulation is essentially hard-coded. It would not be possible, for example, to add a spacecraft and have its trajectory determined by the gravitational forces of the planets. While this is a faithful translation of the mechanical orrery into the digital domain, it seems somewhat a missed opportunity.

The other approach is to determine the motion of the bodies by direct numerical integration of the forces acting upon them. This is the approach used by *ORSA*. The advantage of this approach is that the user can be allowed to modify the system being simulated.

1.2. **Objective.** Given the above, where can we improve on what already exists? The answer lies in the user interface. *Celestia* provides a high-quality 3D display engine, much like high-end computer games. However, its simulation is somewhat limited. *ORSA*, on the other hand, provides a powerful and flexible simulation, but is geared toward scientific data analysis. Its presentation style resembles Matlab. As far as we are aware, there is no program available that provides a 3D game-like display driven by a numerical-integration based backend. This is what we set out to develop in this project.

The result should be useful as an educational tool. What would happen if the Moon were twice as massive? If a planet were added or removed from the solar system? If we orbited a binary star? Our program should allow the user to engage easily in this kind of exploration, and have the results presented immediately and in an appealing way.

2. Simulation Model

One of the first issues in designing a simulation is to decide exactly what aspects of the physical process to include in the model. This section outlines the major considerations.

2.1. **Newton vs. General Relativity.** There are two major theories of gravity – Newtonian and General Relativity. General relativity is certainly a better model, but under the conditions that exist within the solar system the predictions of the two theories are nearly indistinguishable. For our purposes the Newtonian theory is an excellent model of gravity. It is also far less computationally complex than general relativity. We thus constructed our simulation on a Newtonian framework.

2.2. **Tidal Effects.** The motion of planetary satellites is significantly affected by tidal interaction with the planet they orbit. These effects arise because the bodies are not rigid spheres of uniform density, as in the Newtonian scheme, so cannot truly be considered as point masses.

Directly determining tidal effects on orbital motion is a problem of significant complexity, requiring detailed knowledge of the form and internal physical properties of the bodies. The author is not aware of any simulation that currently handles these effects dynamically.

Semi-empirical corrections to the Newtonian field can be used [2], however these are essentially a fit to observed data, and need to be developed on a case-by-case basis.

For a simulation designed to run in real time, it was considered that the modeling of tidal effects was too computationally complex to include.

2.3. **Non-Gravitational Effects.** Several other effects, such as course correction by spacecraft and out-gassing by comets, were considered for inclusion in the simulation. Including these effects would comprise a periodic adjustment of the bodies velocity vector, and incorporates naturally and easily into the simulation. However, due to limited available project time, these effects remain to be coded.

3. Force Algorithm

The primary task of the backend is to calculate updated positions for the bodies in the simulation as the frontend requests them. Performing the calculation is a two-step process. First, we must determine the acceleration of each particle, by calculating the gravitational force acting upon it. This is the job of the force algorithm, discussed in this section. Then, knowing the acceleration, we must determine the positions and velocities of the particles at the next timestep. This is the numerical integration step, discussed in Section 4.

The force algorithm performs the first stage of the `Advance()` process - determining the acceleration of the particles at the current timestep. For the N-Body case, the acceleration of the i^{th}particle is given by:

$$\vec{a}_i = \sum_{j \neq i}^{N} \frac{GM_j}{|\vec{r}_{ij}|^3} \vec{r}_{ij}$$

Some of our numerical integration procedures also require the first derivative of acceleration, known as the *jerk*. This could be estimated from the difference in acceleration between timesteps, but it is more accurate to differentiate the above expression to get the explicit formula below:

$$\vec{j}_i = \sum_{i \neq j}^{N} \frac{GM_j}{|\vec{r}_{ij}|^3} \left(\vec{v}_{ij} - \frac{3\left(\vec{r}_{ij}.\vec{v}_{ij}\right)\vec{r}_{ij}}{|\vec{r}_{ij}|^2} \right)$$

We will now consider various algorithms for determining the acceleration and jerk of particles.

3.1. **Direct Summation.** The simplest force evaluation algorithm is to directly encode the above summation. We can eliminate a little work by noting that $|\vec{r}_{ij}|^3 = |\vec{r}_{ji}|^3$, and hence cut the number of calculations in half. However, we must still evaluate the force expression for every particle pair.

Thus the algorithm is unavoidably $O(n^2)$. Evaluating the force expression is costly, since for ever particle pair we must calculate$|\vec{r_{ij}}|^3$, or in components $\left(x^2+y^2+z^2\right)^{\frac{3}{2}}$. This involves an expensive square root calculation and as n grows it comes to dominate the running time of the entire program.

3.1.1. *Tweaks to Direct Summation.* We can tweak the direct summation process slightly to improve performance. We divide the bodies in the simulation into two groups - *interacting* and *non-interacting*. Non-interacting bodies are test particles that feel the pull of the gravitational field, but do not generate their own field. . Consider, for example, a simulation containing the planets and a spacecraft. It would be ridiculous to calculate the pull of the spacecraft on the planets. Force calculations become $O(n_i^2)$, where n_i is the number of interacting bodies only. With a carefully chosen threshold, the accuracy impact is slight.

3.2. **Hierarchical Force Determination.** The speed of the $O(n^2)$ algorithm is quite acceptable if our simulation contains only a small number of bodies. However, as our simulation grows it becomes the limiting factor on backend speed. A simulation containing most major solar system object requires 35 interacting particles. If our simulation is to handle this number of particles and still run smoothly on a typical desktop PC, we require a faster force determination algorithm.

For exact determination of forces, it is hard to do better than $O(n^2)$. However, if we permit a small degree of approximation, there are large gains to be made.

3.2.1. *Strategies.* In the $O(n^2)$ algorithm, often the strength of many of the interactions computed will be completely negligible. For example, the motion of the Earth is dominated by the influence of 3 or 4 other objects. However, we calculate the effect of all 35. Most of these other interactions are comparatively minuscule.

A natural strategy would be to discard or approximate many of these minor influences.

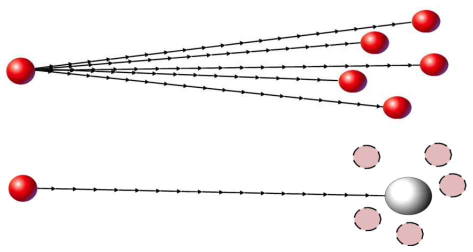

FIGURE 3.1. A remote group of particles can be replaced by a single particle.

Consider the effect on a particle of a *remote*, *compact* group of bodies. Instead of calculating the pull of each of these particles individually, we could replace the group by a single aggregate particle. What we mean by remote and compact will depend on how much error we are willing to tolerate in our results. In many cases the accuracy penalty involved can be very small.

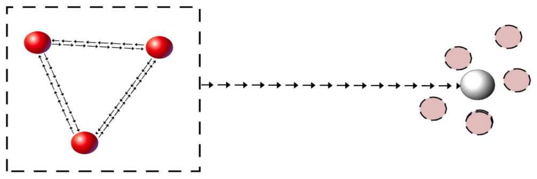

FIGURE 3.2. Group-Group interaction approximations

The above reasoning can be extended to interactions between groups. Say we have two groups, which are both compact and mutually remote. Instead of computing the pull of the remote group on each of the local particles, we instead compute the interaction between the two barycentres. This force can then be applied to all the particles in the group. Interactions within the local group are computed directly.

Since a group can be internally subdivided into further groups, this suggests a divide-and-conquer approach to computing the gravitational interactions. These techniques are well-developed in existing N-body simulators.

3.2.2. *Grouping Algorithm.* The approach outlined relies on identifying groups of particles within our simulation. Several properties could form the basis of a grouping algorithm: physical separation, velocity, local density, gravitational attraction, etc.

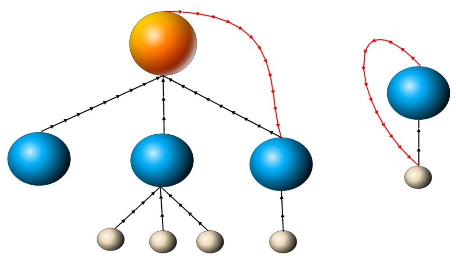

FIGURE 3.3. A greatest attractor tree on an example system. The graph is not connected due to the presence of binaries (e.g Pluto/Charon and Neptune/Triton within the solar system).

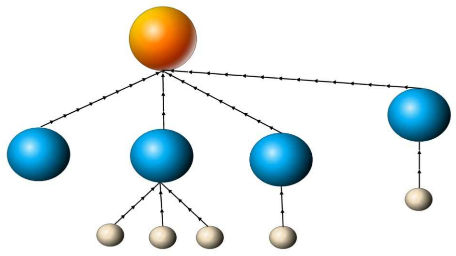

FIGURE 3.4. A modified greatest-attractor tree. This structure successfully groups our system.

One approach would be to examine these properties and group objects which are in some sense *close*. However, this leads to the problem of thresholding – to group things which are *close* we then need to define some type of cutoff where *close* becomes *far*.

We decided against pursuing this route, and instead tried to exploit the natural hierarchical structure of planetary systems. Consider the graph resulting from linking each body to that body which attracts it most strongly (Fig. 3.3). Many natural grouping show up in this graph. For example, moons are grouped with their planets, etc. However, the graph contains cycles and is not connected. We would like to generate a tree structure to which we can apply a recursive force-determination scheme along the lines outlined earlier, so these features need to be eliminated from the graph.

We can obtain a much more useful graph if we change the construction criteria so that each body links to that body *with greater mass* that attracts it most strongly *(*See Fig. 3.4). Given that there are no two bodies of the same mass (a physically reasonable assumption) it is simple to show that this graph cannot contain cycles and has a unique root. This tree gives us the kind of groupings we need for our force algorithm.

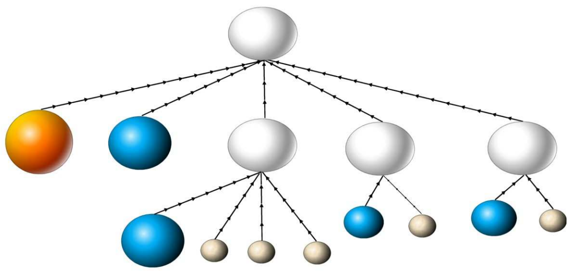

FIGURE 3.5. Our modified groupings tree. *PseudoBodies* are shown in white. Each node in the tree is the barycentre of the nodes below that point.

3.2.3. *PseudoBodies and Force Estimation.* For the approximation scheme, we need to know the mass, position and velocity of the barycentres of every group. Rather than calculate these each time, we insert new *pseudobodies* into our tree to record this information (See Fig.3.5).

Pseudobodies do not refer to any physical body, and are invisible to the front end. They simply record the barycentre information.

Once this tree has been constructed the operation of the algorithm is basically as outlines in Section 3.2.1, and is illustrated in Fig. 3.6. Staring on the level below the root, we calculate the interaction of all the bodies on that level using the $O(n^2)$ algorithm. We then examine the bodies on that level. If a body has no descendants, its acceleration has been completely determined and we need to do nothing more. Otherwise, we "trickle down" its acceleration to the bodies directly below it, then recursively call the evaluation function on that level.

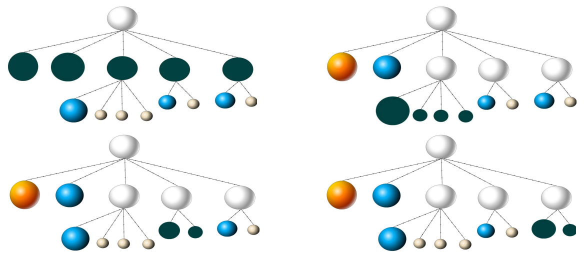

FIGURE 3.6. Interacting groups in the tree-force algorithm (shown in green).

3.2.4. *Performance and Review.* With only 35 bodies in the system we were doubtful if our algorithm would offer much benefit over direct summation. However, it actually resulted in a $\times 2.5$ speedup, with very modest accuracy penalty.

Our scheme appears to be successful in pragmatic terms, though asymptotically it's still $O(n^2)$. This is because, in the worst case, the grouping algorithm could determine that all the bodies in the system formed a single interacting group. With no partitioning of the system, we have a flat tree and no performance benefit. Pragmatically, however, the algorithm should offer performance gains on any system with a hierarchical structure.

That said, our grouping algorithm could be more aggressive. For example, within the solar system the Sun's attraction is so dominant that many objects have it as their greatest attractor. As a result our tree is not very deep, with many objects on the first level. Tweaking the algorithm to further partition large blocks like this offers scope for further performance enhancements.

A final word – while this algorithm was designed from first principles, a literature review unsurprisingly revealed that many similar algorithms exist [3, 4]. In particular, the force approximation scheme is almost identical to that described by Appel [5], which was later developed into the Fast Multipole Method.

We have not come across a grouping algorithm exactly like our own, though Hut and Eisenstein's HOP algorithm [6] bears many similarities.

4. Numerical Integrators

4.1. **Introduction.** The second step in advancing the simulation is the numerical integration step. Knowing the acceleration of bodies at t, we need to integrate to get their position and velocity at $t + \delta t$. There exist a large number of procedures for doing this. Even for the specific case of N-body simulations, which method to use is far from obvious.

In the absence of any clear favourite method, it was decided that we would attempt to implement and compare several methods. With a little care in the design of the program structure, the nature of the underlying numerical integration procedure can be changed without affecting the interface to front end. The change is also largely transparent to most of the backend methods.

4.2. **Leapfrog.** A very simple and effective method is the Leapfrog or Verlet method [10, 12]. The update-equations appear very similar to the basic Simple Euler method:

$$s_1 = s_0 + v_{\frac{1}{2}}\delta t$$
$$v_{\frac{3}{2}} = v_{\frac{1}{2}} + a_1\delta t$$

Though it is not obvious for the equations, leapfrog is a second-order scheme, which exhibits good conservation of energy and angular momentum. The method tends to perform especially well on orbits which are near-periodic, as in the solar system, due to the time symmetry of the scheme.

4.3. **Time Symmetry.** Some integrators have the property of time-symmetry. For a full discussion of time symmetric integrators and their properties, see [10, 11, 8]. Informally, if a time symmetric integrator is run in the forward time direction, and then time is reversed, the integrator will retrace its steps. Errors are made symmetrically, so that, except for the effects of rounding errors, the particle will return to its starting point.

Time symmetry is a basic property of the physical laws we are simulating. Integrators that mirror this property tend to have desirable characteristics. In particular, they tend to exhibit very good energy conservation. To understand why this occurs, consider a particle in a circular orbit . Since the orbit is symmetric, the forces experienced by the particle in the second half of its orbit are equal and opposite to those experienced during the first half. So travelling the second half of the orbit is essentially the same as travelling the first half of the orbit in the time-reverse direction. Time-symmetry means the errors made during the second half cancel with those made during the first half. As a result, errors are periodic, and do not accumulate between orbits. This can be clearly seen in Fig. 4.1

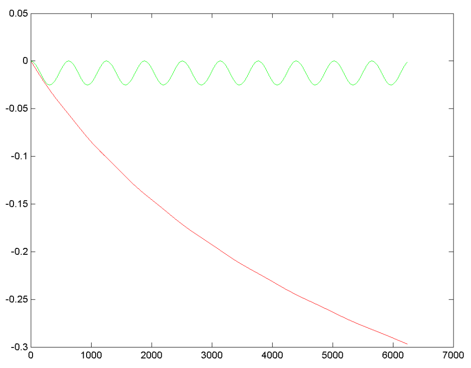

FIGURE 4.1. Energy profile for Simple Euler and Leapfrog for a particle in a circular orbit. The vertical scale of the Leapfrog error has been magnified by 10^7. Observe that the energy error for leapfrog is periodic.

The good performance characteristics of this class of integrators tends to be exaggerated by orbits which are exactly symmetric, such as our test case. The solar system, being composed of many near-periodic orbits, is also particularly suited to integration with time-symmetric schemes. However, even in more general cases they perform very well. The transient energy error tends to manifest itself, for a periodic orbit, as a small linear drift in the time of pericentre passage – for our purposes a relatively benign type of error.

4.4. **Hermite Predictor-Corrector Scheme.** Hermite scheme is a fourth-order predictor-corrector scheme [7, 9, 12]. Like the Leapfrog scheme, it is also time symmetric. The update equate equations are as follows:

The predictor step simply a truncated Taylor series:

$$s_{pred} = s_0 + v_0\delta t + a_0\frac{(\delta t)^2}{2} + j_0\frac{(\delta t)^3}{6}$$

$$v_{pred} = a_0\delta t + j_0\frac{(\delta t)^2}{2}$$

The force algorithm is now called to determine a and j at s_{pred}.This is known as the evaluator step. Finally we apply the corrector step:

$$v_1 = v_0 + \frac{\delta t}{2}(a_0 + a_{pred}) + \frac{(\delta t)^2}{12}(j_0 - j_{pred})$$

$$s_1 = s_0 + \frac{\delta t}{2}(v_0 + v_1) + \frac{(\delta t)^2}{12}(a_0 - a_{pred})$$

We can iterate the Evaluator-Corrector stage for higher accuracy. However, each iteration requires a force evaluation, and thus is expensive. Generally, when force evaluations are expensive, the single-iteration PEC scheme was found to give the best trade-off, though $P(EC)^2$ is preferable in situations where we want high accuracy with a small number of bodies.

With energy errors on the order of 10^{-9} over 100 years of simulated time, it was decided that the Leapfrog and Hermite integrators more than met the performance goals set out in our spec, and were chosen as the integrators for the program. Both implemented methods are available in the final program, and the existing program structures should support the implementation of further schemes.

5. Further Integrator Issues

5.1. **Variable and Individual Timesteps.** There are performance gains to be made by moving beyond a single fixed simulation timestep and allowing variable or individual timesteps. However, naively implemented, a variable timestep scheme will destroy the accuracy of an integrator that depends upon time symmetry. Recently methods have been developed that allow time-symmetry to be restored in the variable-timestep case [10, 11]. These scheme remains to be implemented in our simulation. Also, save for comets, orbits within the solar system are only moderately eccentric, so variable timesteps are not critically important.

5.2. **Time Control and the Front End.** One issue that proved tricky to resolve was the interaction of the backend and the frontend regarding time control. The frontend calls `Advance(T)` once per frame to update the position of bodies. The problem is that the frame-rate of the display is dependant upon the number of bodies in the field of view. If the front end called `Advance(T)` with a constant T per frame, then bodies would appear to move at varying rates depending on the number of bodies on-screen, even though their speed in the underlying simulation remains the same.

This suggests that the frontend should vary the T per frame to maintain a fixed ratio between simulated time and real time. However, this presents problems too. As explained in Section 5.1, varying the timestep interferes with time-symmetry and destroys the accuracy of the simulation. So we

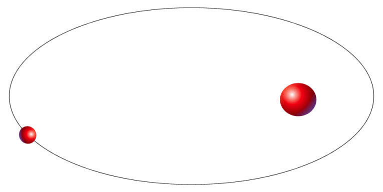

FIGURE 6.1. Orbit lines.

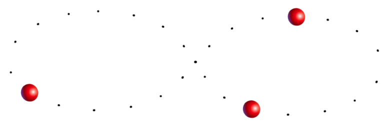

FIGURE 6.2. Some situations, such as this figure-of-eight orbit, cannot be well-approximated as a collection of two-body problems. In this situation, orbit lines are not drawn, and we revert to drawing trails only.

cannot link T directly to the integrator timestep δt.

Our saving grace is that the backend runs considerably faster than the frontend – generally one or two orders of magnitude faster. The solution we implemented was to have an internal timestep δt, perhaps 10 to 100 times smaller than the typical T per frame. Then, when the front end requests an advance of T, this actually causes the backend to take T div δt steps (each of length δt). Since $\delta t \ll T$, this is almost equal to the advance requested. This allows us to maintain an almost constant ratio between real time and simulated time, while allowing the backend to retain control over its own timestep.

6. Orbit Visualization

Another task of the backend is to support orbit visualization. To do this we determine an analytic equation for the orbit of the body, which allows us to plot an orbit line as in Fig. 6.1. In an N-body situation this will not be possible in general, since only 2-body problems are analytically solvable. However, in cases where the force from one body dominates, we can approximate our N-body problem by a collection of 2-body problems, which are then solvable analytically. This works well within the solar system – it is, after all, why Kepler's Laws are successful. For more complex situations where no good 2-body approximation exists (as in Fig. 6.2), we fall back upon plotting a trail of where the body has been recently.

7. Ephemerides

The simulation takes initial positions from the standard JPL DE406 ephemeris published by NASA. However, the JPL ephemeris contains no data for the natural satellites of the planets. While this data is available from other sources, it exits in a variety of different file formats and is difficult to access. Currently initial position data for the natural satellites is available in our program for a limited number of dates only.

8. Conclusions and Evaluation

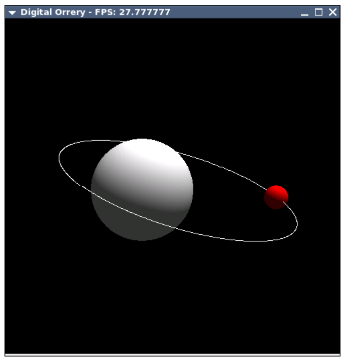

FIGURE 8.1. A screenshot of the final program.

8.1. **Review.** Development so far has provided the core functionality of our digital orrery. A flexible simulation engine is now complete, as is the rendering engine and the basics of the user interface. The final simulation is fast, flexible and accurate, and provides support for some interesting visualizations. We also believe that it is unique in offering the user an appealing graphical front end to a truly flexible underlying simulation.

8.2. **Further Developments.** While we have so far achieved a solid core of functionality, there is still some way to go in providing an easy to use educational tool. In particular, work remains to be done in developing the user interface. By nature this project is very open ended, and there remains plenty of scope for features to be added. Possible features for inclusion in future development could include visualization of the gravitational field and eclipse/conjunction prediction. Also, as noted in the body of the report, there are several avenues open for improving the core force algorithm and the integration schemes.

References

[1] Nobili, A. M. and Rosburgh, I.W., *Simulation of General Relativistic Corrections in Long Term Numerical Integration of Planetary Orbits*, Relativity in Celestial Mechanics and Astrometry, 1986

[2] Vitagliano,A., *Numerical Integration for the Real Time Production of Fundamental Ephemerides over a Wide Time Span*, Celestial Mechanics 66, 1996.

[3] Anderson,R.J., *Computer Science Problems in Astrophysical Simulation*, Proceedings of the Silver Jubilee Workshop on Computing and Intelligent Systems, 1993.

[4] Graps, A., *Particle Simulation Methods*, 1995 (Available at amara.com/papers/nbody)

[5] A. W. Appel, *An efficient program for many-body simulation*, SIAM Journal of Scientific and Statistical Computing, 1985.

[6] Eisenstein, D.J. and Hut, P., *HOP: A New Group-Finding Algorithm for N-Body Simulations*, Astrophys. J., 1998

[7] Kokubo, E., Keiko, Y., and Makino, J., *On the time-symmetric Hermite integrator for planetary N-body simulation,* Notices of the Royal Astronomical Society, 1998.

[8] Funato, Y., Hut, P., McMillan, S., and Makino, J., *Time-Symmetrized Kustaanheimo-Stiefel Regularization,* The Astrophysical Journal, 1996.

[9] Makino, J. and Aarseth, S.J., *On the Hermite Integrator with Ahmad-Cohen Scheme for Gravitational Many-Body Problems*, Astron. Society of Japan, 1992.

[10] Hut, P., Makino, J. and McMillan, S., *Building a Better Leapfrog,* The Astrophysical Journal, 1995.

[11] Hut et al., *Time Symmetrization Meta-Algorithms*, Computational Astrophysics, 1997.

[12] Hut, P., and Makino, J., *The Art of Computational Science*, forthcoming. *(Draft available at www.artcompsci.org)*

[13] Everhart, E., *An Efficient Integrator that uses Gauss-Radau Spacings*, Dynamics of comets: their origin and evolution, 1985.

[14] Aarseth, S.J., *Multiple Time Scales*, Academic Press, 1985.

[15] Press, W.H., Saul, A.T., Vetterling, W.T., and Flannery, B.P., *Numerical Recipes in C++*, Cambridge University Press, 2002.

[16] Burlisch, R. and Stoer, J., *Introduction to Numerical analysis*, Springer, 2002.

[17] Lee, S.E., *A Calendar Conversion Program*, http://www.scottlee.net/, 1995.

[18] Dybczynski, P.A., and Gray,B.J., *C++ translation of the JPL import code*, http://www.projectpluto.com/jpl_eph.htm, 2003.