Planet Simulator

Honors Project

by

Jochen K. Illerhaus

May 10, 2016

Supervisor:
Dr. Wojciech L. Golik

St. Charles (MO) Linendenwood University
School of Science
Institute of Mathematics

Contents

1 Abstract

TODO: WRITE THE ABSTRACT!!!

2 Introduction

2.1 Problem

The task was to write some software that would visualize the motion of bodies (planets) in a gravitational system (Solar System). The simulation was to show the motion of those bodies from the perspective of one one of them. This would than show the Apparent Retrograde Motions [1] of planets.

2.2 Personal Motivation

Computers have always fascinated me. Hence I enjoyed the chance to write a complicated and interesting peace of software. It was in the nature of the problem that I would have to work with computer graphics which is a part of computer programming that I particularly enjoy because I like seeing things move on my screen. I also have the dream to one day, write a computer game. This solar system project would allow me to work with some things important to fulfill that dream.

2.3 Prior Knowledge

I started to write my first programs four years ago initially using C++. Since than, I have been constantly improving my knowledge of computer programming and computer science although I never took any organized classes on computer programming. At the starting point of the project I was confident with C++, Java, Python, JavaScript, and Scilab (a clone of Matlab). I had done some data visualization and I had written some simple games. For their graphical parts, I had used the libraries: Swing (generic Java), Standard Windows API (C++), OpenGL through Glew (in both C++ and Java), Matplotlib (Python Anaconda), and I had worked with the direct manipulation of pixel data (in C++, Java, Python, and JavaScript). I had had three years of Physics in high school (3 semesters of them were calculus based) and I was rewarded with a membership in the German Society of Physicist for academic excellency in that subject. Those classes gave me a fundamental understanding of Newtonian Physics and Newtonian Gravity.

3 Programming Language

3.1 Selection of the Programming Language

I planed to select a programming language that I was already familiar with. On top of this, I wanted to write a "‘big"’ application, not just a simple visualization. Further more a I anticipated that the result would have to be runnable

[1]Apparent Retrograde Motion is the phenomenon of planets appearing to move backward when observed form another planets

without unusual external dependencies (that was a knockout for programs like Mathematica or Matlab). It would have been impractical to learn a new language just for this project. So, I had to choose form Python, Java and C++. Which one I picked didn't really matter, and I selected C++ for no particular reason.

3.2 Selection of the Graphics Library

I picked OpenGL (glew) as my graphics library knowing that this was not the easiest method to create graphics. There are many libraries, e.g. Matplotlib (for Python), available that offer easy tools that allow the creation of a simple plots in only a few lines. OpenGL on the other hand is a massive graphics library. The libraries API is as close to the hardware as it will get. When implemented correctly, OpenGL usually offers the best performance possible. OpenGL is commonly used for video games, computer aided design, movies, etc. This makes it an very important tool, hence the knowledge of its API is a valuable skill. So the main reason for selecting OpenGL was that I wanted to get familiar with it's API. As mentioned, I had worked with OpenGL befor but I had always used OpenGL 1.0 which does not use features like a programmable rendering pipeline or the exclusive use of buffer objects. I was looking forward to learn a new OpenGL based techniques. OpenGL requires the use of linear algebra to calculate transformations [2], manipulate positions, etc. In C++ the library OpenGL Mathematics (GLM) is often used for those tasks so I used it for this project as well.

4 Planet Motion

Obtaining the Planets positions as time progresses is obviously a key requirement. There are two options to do so that presents themselves. Firstly, planets move on conic sections that can be described in symbolic equations. A planets position at an arbitrary point in time may than be found by evaluating those symbolic expression. Second, some numerical method can be used solve a system of differential equations that describes the motion of those planets. Those differential equations would follow directly form a mathematical description of gravity (either Newton or Special Relativity). Given that my first idea would only involve evaluating some equations I was sure that this would be the easiest way but I was also curious to see whether or not my second thought would work the way I expected it to. Approximating the solution to a differential equation would also have the advantage that it would be a real simulation. Additionally to fulfilling the requirements of this task this would also allow answering questions like: What would happen if a gigantic comet crashed into Mars and slowed it down considerably? In the end I used both methods. I first wrote a solver (using Forward Euler Method) for the differential equation that follows form Newtons law of gravity and than implemented my first idea by reading in orbital elements and calculating the planets positions form them.

[2] It is also possible to use a mathematically much cleaner approach through Quaternions $\mathbb{H}$

4.1 Planet Motion as Solutions of Differential Equations

Gravity was first described by Newton through the equation:

$$\vec{F_G} = \gamma \frac{mM}{r^2} \hat{r} \tag{1}$$

Where $\vec{F_G}$ is the Gravitational force that body A exerts on body B, γ is the Gravitaionla Konstant ($\gamma \approx 6.6710^{-11} m^3 kg^{-1} s^{-2}$), m is the mass of B, M is the mass of A, r is the distance between the centers of mass of A and B, and $\hat{r}$ is a unit vector pointing form the center of mass of B to the center of mass of A. A fundamental part of Newtons mechanics is Newtons Second Law:

$$\vec{F} = m\ddot{\vec{x}} \tag{2}$$

Where $\vec{F}$ is the Force, m the mass and $\ddot{\vec{x}}$ the second derivative with respect to time of the position.

Form equations 1 and 2 follows that:

$$\ddot{\vec{x}} = \gamma \frac{M}{r^2} \hat{r} \tag{3}$$

This differential equation relates the second derivative of the position of body B to its position (hidden in the corresponding $r$and $\hat{r}$). It is of second order and non linear. To reduce this to a first order problem that can be solved with standardized methods, the substitution $\vec{v} = \dot{\vec{x}}$ can be used. This reduces the problem to a system of non linear first order differential equations:

$$\begin{cases} \dot{\vec{v}} = \gamma \frac{M}{r^2} \hat{r} \\ \dot{\vec{x}} = \vec{v} \end{cases} \tag{4}$$

The problem with 4 is that it only considers a system of two bodies. In practice it is often possible to ignore the influence of most gravitational bodies except for one central body. For example, a model of the solar system can reach a reasonable degree of precession without considering the gravitational influence that the planets have on each other. Once this approximation is accepted, each planet could be considered as a individual two body problem (sun and planet). But I wanted to go through a more accurate model that would allow the simulation of binary stars and other special cases that require the consideration of the gravitational influences of all bodies in the system. Since in those cases 4 has to hold pairwise for all bodies in the system it is a reasonable approach to simply sum all gravitational forces exerted on a given planet and than use the resulting force in a system of differential equations similar to 4. The resulting system of differential equations, that describes the motion of body A is:

$$\begin{cases} \dot{\vec{v}} = \gamma \sum_{i \in I} \frac{M_i}{r_i^2} \hat{r}_i \\ \dot{\vec{x}} = \vec{v} \end{cases} \tag{5}$$

Where I is the set of all bodies except A, M_i is the mass of body i, r_i is the distance from the center of mass of body A to the center of mass of body i, and $\hat{r}_i$ a unit vector pointing in the direction of that distance.

The delima with this system of differential equations is that the future positions can only be found if the future forces are known and the future forces can only be found if the future positions are known. My solution was to define a small time step Δt and assume that the gravitational forces would not change during that time step. In reality, the gravitational Forces change continuously but it turns out that a small Δt is close enough to obtain reasonably accurate results. With the definition of Δt time can be considered as an integer n such that $t = n\Delta t$. It follows form 5 that:

$$\vec{a_n} = \gamma \sum_{i \in I} \frac{M_i}{r_i^2} \hat{r_i} \quad ; \quad \vec{v_{n+1}} = \vec{a_n}(1 + \Delta t) \quad ; \quad \vec{x_{n+1}} = \vec{v_n}(1 + \Delta t) \tag{6}$$

Using 6, the planets positions can be obtained through iterating over n. Initial conditions ($\vec{v_0}$ and $\vec{x_0}$) have to be known and are input parameters.

Note This method is called *Forward Euler* and its error grows linearly with Δt [1]. Forward Euler is vary bad numerical method for the approximation of periodic solutions there are more advances methods [1] but Forward Euler is simple enough that I could develop the method myself and if a small enough Δt is picked the results are still reasonably accurate.

As Figure 1 shows, the choice of Δt influences both the precision of the approximation. I picked a very small Δt (exactly how small, is a parameter that is read in together with the other input data) [3].

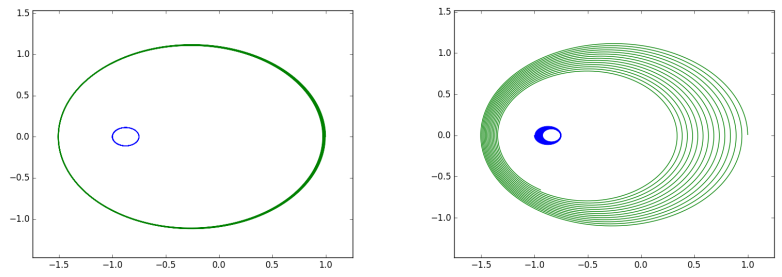


Figure 1: *Left:* The Orbits of two planets with $\Delta t = 0.001$ and 100000 steps
Right: The Orbits of two planets with $\Delta t = 0.02$ and 5000 steps
Both use the same initial conditions

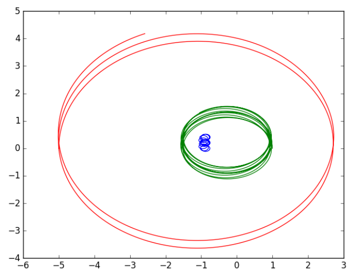


Figure 2: Orbits of 3 planets in the system ($\Delta t = 0.001$)

[3]A small Δt also goes with a high computational cost

4.1.1 Assumptions and Simplifications

The latest models for planetary motions take many effects into account. As described above I am only considering Newtonian Gravity. This leaves out many effects (especially General and Special Relativity). But since the numerical method used goes with a very high error, considering any of those effects does not appear to be helpful.

4.2 Planetary Motion Derived from Kepler Elements

Another idea was to consider the orbits of those bodies as conic sections. Because the visualization was only interested in planets the only important type of conic section would be the ellipse.

In the easiest case, planetary motion on elliptical orbits can be described in polar coordinates using the eccentricity (e) and length of the semi-major axis (a) of the Ellipse as parameters. The state vector ($\vec{s} = \langle\ r, \Theta, v, \omega\ \rangle$) can than be described as a function of time an converted to Cartesian coordinates. The result can than expresses the state of the body on a plain in $\mathbb{R}^3$ were the position and velocity are each element of $\mathbb{R}^2$ to rewrite the coordinates of the planet on the plain in terms of the global base, a simple matrix transformation can be applied to the position and velocity of the planet. This is a common concept (see [2] [3]).

5 Apparent Retrograde Motion

The main task of the project was visualizing the apparent retrograde motion of planets as seen form other planets. The effect is a consequence of planets having different orbital periods. Since the Gravitational force acts as a centripetal force (assuming circular orbits), the planets will have different tangential velocities, and path lengths, hence their years will be of different lengths:

$$F_G = F_c \Leftrightarrow \gamma \frac{mM}{r^2} = m\left(\frac{2\pi r}{T}\right)^2 \frac{1}{r} \Rightarrow T = \sqrt{\frac{4\pi^2}{\gamma M} r^3} \Rightarrow T \propto r^{3/2} \qquad (7)$$

This is not the velocity observed form Earth, but from the center of Gravity of the system. A observer on a planet will move together with that planet. In the best case this would allow the observer to see the sum of his/her own velocity vector $\vec{v}$ and the velocity of the observed planet $\vec{u}$. A second effect is caused because any observer at a single point can only observe a 2D projection of the 3D universe. Visible for the observer is only the projection vector $\vec{w}$ of the $\vec{v} + \vec{u}$ vector on a plain perpendicular to the connecting line form the observer to the observed object. Figure 3 illustrates this in two dimensions.
Note: Another type of the apparent retrograde motion of planets is caused by the planet rotating about its own axis. This effect is not considered in that project.

Apparent retrograde motion occurs when $\vec{w}$ switches directions. Figure 4 shows to extreme cases of apparent retrograde motion.

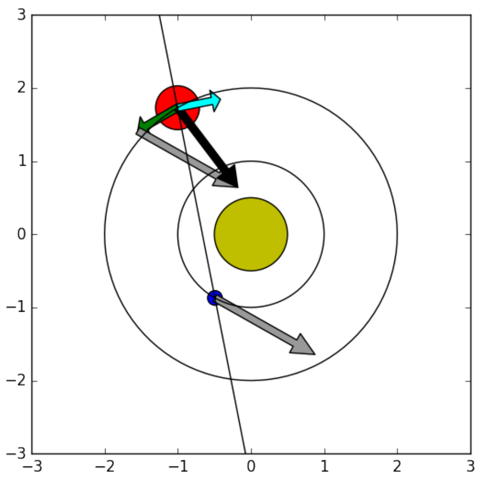


Figure 3: The velocities as seen form earth: gray: $\vec{v}$, green: $\vec{u}$, black: $\vec{v}+\vec{u}$, blue: $\vec{w}$

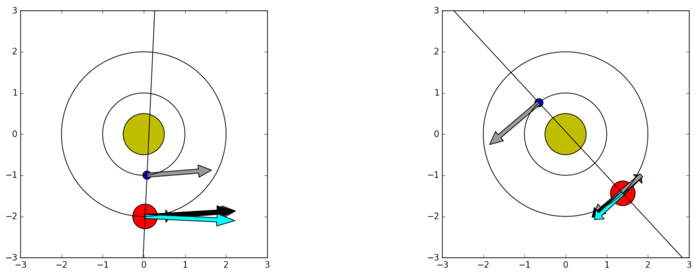


Figure 4: **Left:** the speed as observed from earth is at its positive extreme **Right:** the speed as observed from earth is at its negative extreme

6 User Manual

The user interface (UI) of this program is bad. It heavily relies on third party software (to edit input files) and on key events. On top of that, neither the input files nor the keys follow a clear nor concise system. The user interface of this program could be greatly improved in a future project.

6.1 General Operation

The User Interface (UI) consists of an visual output and file, keyboard, and mouse inputs. The Output is split into two areas. The main area of the window shows an overview of the current scene. In the lower right hand corner, a separate panel shows the current scene as it could be observed from the center of the current planet. Whichever planet is currently selected, and some of its properties, is outputted as text in the upper left hand corner. An arrow in the upper right hand corner points towards the current planet's position. A bar chart on the right hand side informed about the energy that is currently in the planetary system. The yellow bar showed the total amount of energy (gravitational + kinetic), the green bar represents the total kinetic energy of all bodies, and the red bar represents the total gravitational energy of all bodies. A body infinitely far away form all other bodies is considered to have a gravitational energy of 0 and the coordinate system is defined to have a kinetic Energy of 0. (Note: Each of these bars determines a reasonable scale on start up, those scales will not be equal).

6.1.1 Camera

The camera controls are inspired by the videogame Minecraft. Right-Mouse-Dragging allows to changing the Euler Angles of the direction the camera looks in. The camera position is set using the keyboard. The Keys *A* and *D* move the camera perpendicular to the projection of the view direction, *W* and *S* move the camera along the view direction, shift and spacebar move the camera up and down [4].

6.1.2 Observation Mode

Visualizing the apparent retrograde motion of planets is done in Observation Mode. The Observation Mode is toggled on/off with the the *J* key. In Observation Mode, the text on the left hand side expands and shows information about both: the current planet (at which the camera is stationed) and the observed planet (at which the camera is pointed). Observation Mode also shows the velocity vector of the observed planet projected such that it corresponds to the velocity as it would be perceived form earth. The length of this vector is also represented by a purple bar on the right hand side. The current planet can be changed with the TAB key (TAB jumps to the next planet TAB+B to the previous). The observed planet is change with CONTROL+TAB (CONTROL+TAB jumps to the next planet CONTROLE+TAB+B to the previous).

6.1.3 Key-Map

Here are all the functions of the keys:

- **W, A, S, D, SHIFT, SPACE:** Move main camera forward, left, backward, right, down, up **RIGHT_MOUS_BUTTON:** Slows down the speed at which W, A, S, D, SHIFT, SPACE move the camera trough space ("‘sneaking"’).
- **TAB, TAB+B:** Change the planet the on planet camera is on.
- **SHIFT+TAB, SHIFT+TAB+B:** Change the planet the on planet observes (observation mode only).
- **U:** Start/Stop simulation.

[4]this setup is usually controlled with the left hand where the baby finger is placed on shift, the ring finger on *A*, the middle finger alternates between *W* and *S*, the index finger controls *D* and the thumb is used for the space bar

- **K:** Show coordinate system / stop showing it.
- **+, -:** Increase/decrease the speed of the simulation (Warning: Too high speeds can lead to performance and precision issues).
- **F5:** Reload the scene from the file. Can be used as "‘Reset to initial conditions"’
- **F:** Move simulation forward by a single frame.
- **V:** Draw error symbolizing the velocities of the planets / stop drawing them.
- **J:** Enter/leave observation mode.
- **G:** Plot the gravitational force field as arrows / stop plotting it. Warning: A new scale for the arrows is determined every frame.
- **P:** Shows stationary randomly placed stars in the background / stops showing them.
- **T:** Draws 1000 data point long traces behind planets / stops drawing them.
- **PAGE_UP, PAGE_DOWN:** Increase/decrease the size of the on planet camera’s display.
- **F2** Increase the size of the window.
- **ESC** Leave the program

6.2 Input Files

The initial conditions and parameters are read from an input file. A file picker would be a useful improvement but currently a fill called "‘inputPlanetHorizen.txt"’ in the working directory is loded. All input files are UTF-8 encoded text files and will usually use the extension ’.txt’. The parsing of those files only catches view error and it is therefore essential that the input files are precisely correct. There are two fundamentally different protocols used which are distinguished by their first line: "‘#cartesian"’ files specify the initial state vectors [5] in Cartesian coordinates, "‘#kepler"’ files specify orbital elements of the orbits of those planets. Figures 5 and 6 show examples for those input files both specifying the solar system. Because the program has problems determining the velocities of planets correctly when given orbital elements (through a "‘#kepler"’ file), the use of those files is discouraged. In both file types, planets are assigned a name that has to consist of "‘A-Z"’, "‘a-z"’, "‘0-9"’, and/or "‘_"’. The files also allow three different scales for the planets radii. It is not practical to display e.g. the suns and the earths radius in proportion, hence it is possible to specify three scales "‘Scale_l"’, "‘Scale_m"’, and "‘Scale_s"’. In the coulomb size, each planet must specify which scale its radius is supposed to be multiplied by. This is done by entering one of the characters "‘l"’, "‘m"’, or "‘s"’. The visualization

[5]If $\vec{x}$ is the position, $\vec{v}$ the velocity of a body than the state vector $\vec{s}$ is defined as $\vec{s} = \langle\vec{x}, \vec{v}\rangle$

also does some lighting calculations (but no shadows) and therefore needs information about which of the specified bodies will act as a light source. This is specified by either true or false in the "‘isLight"’ column [6]. The Visualization does not handle units, therefore it is up to the user to make the units of the input parameters match with each other and the gravitational constant provided. It is advisable to pick one unit for distances, one for time, and one for mass. Then all parameters can be expressed in terms of those units [7].

7 Future Planes

There are a lot of functions that I want to add into my visualizations:

- Direct access to the Horizon System (JPL) to obtain orbital elements, state vectors and properties for various planets directly form JPL Solar System Mecanics .
- Improvements of the style of the the visualisation e.g through the use of texture, normal, and shadow maps or post processing effects.
- Change form the input files (from csv with a non standard header to some standard file system like xml or json)
- An actual graphical UI. Possibly using a mixture of OpenGL and Qt a graphical UI could massively improve user friendlessness.
- General improvements of the Key-Map. Especially reassigning the functions to new keys that are easier to remember.
- The main components of this program could be interfaced through Python and wrapped up in a python model to reduce development time without experiencing any major performance drawbacks.
- Optimizations of the rendering process, e.g. dynamically picking a reasonable vertex density would be helpful.
- Numerically finding the Lagrangian points of a given set of two planets.
- Sound Effects. Planetary motion is periodic just like sound. It might be possible to convert this motion into sound.
- Quaternions instead of linear algebra (a mathematically much nicer approach).
- Lagrangian Mechanics instead of Newtonian, could simplify the equations of motion.

[6]currently only one light source is supported, hence there is only one row can specify true in the "‘isLight"’ column

[7]Notice that none of the radii are relevant for the computations of position and velocity, hence may use a separate unit

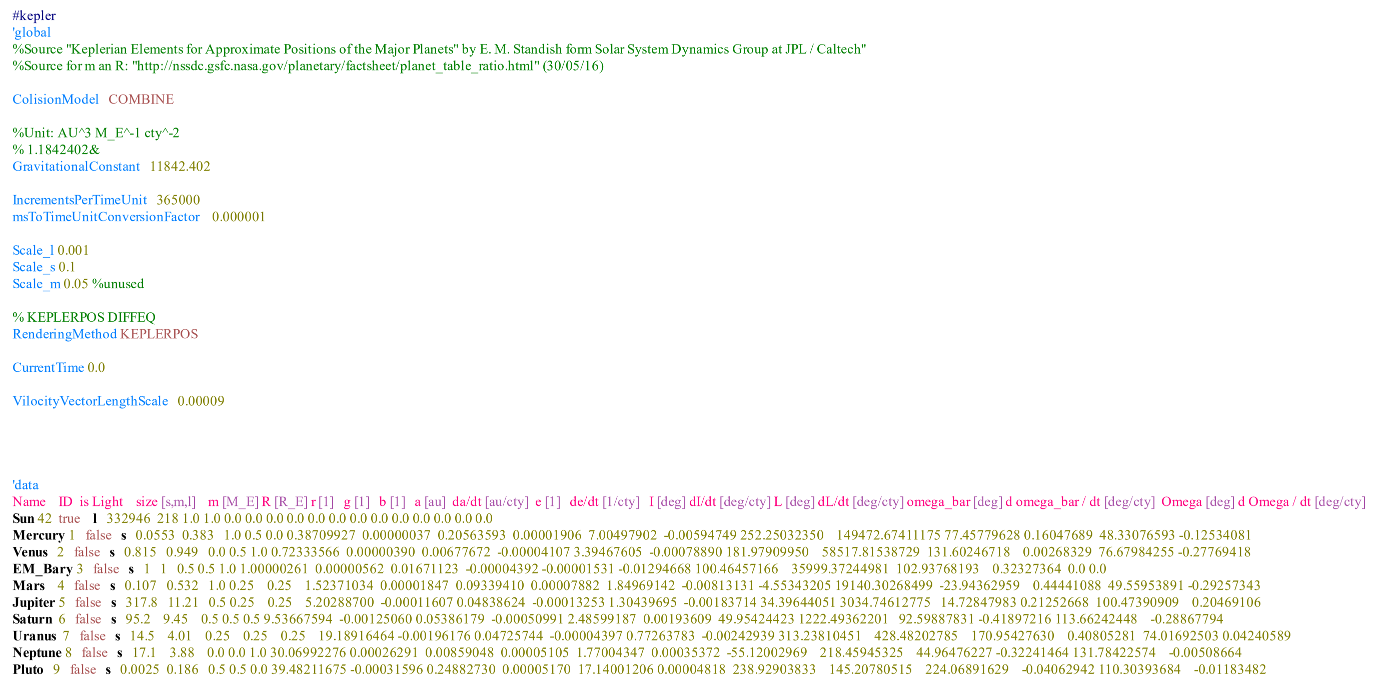
```
#cartesian
'global

%Source Horizen System JPL (Epoch: 2016-02-29)
ColisionModel   COMBINE
%in: AU^3*kg^-1*day^-2
GravitationalConstant   1.48807E-34
IncrementsPerTimeUnit   0.5
msToTimeUnitConversionFactor    1.0
% R[0] = 6.963E5, R[1] = 6371.01

Scale_l 0.0000001
Scale_s 0.00001
Scale_m 0.05 %unused

'data
Name    light?  size    m   R   r_x r_y r_z dx/dt   dy/dt   dz/dt   r   g   b
Sun true    l   1.988544E30 6.963E5 0.000000000000000E+00  0.000000000000000E+00  0.000000000000000E+00 0.000000000000000E+00  0.000000000000000E+00  0.000000000000000E+00 1.0 1.0 0.0
Mercury false   s   3.302E23    2440    6.007997289679312E-02 -4.255835367557072E-02 -4.533875709774438E-01 2.224809279232962E-02   -1.621098658757443E-03  5.140244930641504E-03  1.0  0.5 0.0
Venus   false   s   48.685E23   6051.8  1.130136857826883E-01  -1.637145466446817E-02   -7.184042445106410E-01  1.984529631441176E-02   -1.103074878163787E-03  3.072347239860358E-03   0.0 0.5 1.0
Earth   false   s   5.97219E24  6371.01 -9.285341858933795E-01  -1.066853474546717E-05  3.451590005165814E-01   -6.279483251621094E-03  2.814687460283703E-07   -1.618665531707757E-02  0.5 0.5 1.0
Mars    false   s   6.4185E23   3389.9  -1.513883941107460E+00  2.518641861230258E-02   -5.711216417718485E-01  5.461324847944407E-03  -3.833524345209925E-04   -1.189734360747706E-02  1.0     0.25    0.25
Jupiter false   s   1898.13E24  71492   -5.295296426803042E+00   1.135866072436046E-01  1.180496571154193E+00   -1.733980608854409E-03  6.793899467275298E-05   -7.015077038090188E-03  0.5 0.25    0.25
Saturn  false   s   5.68319E26  60268   -3.424902026580343E+00  2.998967494485082E-01   -9.409678580919255E+00  4.936976254820112E-03  -1.633519847802958E-04   -1.929864534391561E-03  0.5 0.5 0.5
Uranus  false   s   86.8103E24  25559   1.878924347699355E+01  -2.181882598133227E-01   6.760979273335092E+00   -1.360667466357249E-03   3.075151635370548E-05  3.510456144315725E-03   0.25    0.25    0.25
Neptune false   s   102.41E24   24766   2.801566510691893E+01  -4.272476479183284E-01   -1.060371610880382E+01  1.089820734799785E-03   -8.559630543769536E-05  2.948043135554971E-03   0.0 0.0 1.0
Pluto   false   s   1.307E22    1195    8.713966582456347E+00   8.884060807040053E-01   -3.186843322233723E+01  3.095506978437470E-03  -9.266927579566235E-04   1.706463064870472E-04   0.5 0.5 0.0
```

Figure 5: Contents of an "‘#cartesian"’ Input-File specifying the Cartesian state vectors of the Sun, Pluto and all major planets in the solar system



```
#kepler
'global
%Source "Keplerian Elements for Approximate Positions of the Major Planets" by E. M. Standish form Solar System Dynamics Group at JPL / Caltech"
%Source for m an R: "http://nssdc.gsfc.nasa.gov/planetary/factsheet/planet_table_ratio.html" (30/05/16)

ColisionModel   COMBINE

%Unit: AU^3 M_E^-1 cty^-2
% 1.1842402&
GravitationalConstant   11842.402

IncrementsPerTimeUnit   365000
msToTimeUnitConversionFactor    0.000001

Scale_l 0.001
Scale_s 0.1
Scale_m 0.05 %unused

% KEPLERPOS DIFFEQ
RenderingMethod KEPLERPOS

CurrentTime 0.0

VilocityVectorLengthScale   0.00009



'data
Name    ID  is Light    size [s,m,l]    m [M_E] R [R_E] r [1]   g [1]   b [1]   a [au]  da/dt [au/cty]  e [1]   de/dt [1/cty]   I [deg] dI/dt [deg/cty] L [deg] dL/dt [deg/cty] omega_bar [deg] d omega_bar / dt [deg/cty]  Omega [deg] d Omega / dt [deg/cty]
Sun 42  true    l   332946  218 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Mercury 1   false   s   0.0553  0.383   1.0 0.5 0.0 0.38709927  0.00000037  0.20563593  0.00001906  7.00497902  -0.00594749 252.25032350    149472.67411175 77.45779628 0.16047689  48.33076593 -0.12534081
Venus   2   false   s   0.815   0.949   0.0 0.5 1.0 0.72333566  0.00000390  0.00677672  -0.00004107 3.39467605  -0.00078890 181.97909950    58517.81538729  131.60246718    0.00268329  76.67984255 -0.27769418
EM_Bary 3   false   s   1   1   0.5 0.5 1.0 1.00000261  0.00000562  0.01671123  -0.00004392 -0.00001531 -0.01294668 100.46457166    35999.37244981  102.93768193    0.32327364  0.0 0.0
Mars    4   false   s   0.107   0.532   1.0 0.25    0.25    1.52371034  0.00001847  0.09339410  0.00007882  1.84969142  -0.00813131 -4.55343205 19140.30268499  -23.94362959    0.44441088  49.55953891 -0.29257343
Jupiter 5   false   s   317.8   11.21   0.5 0.25    0.25    5.20288700  -0.00011607 0.04838624  -0.00013253 1.30439695  -0.00183714 34.39644051 3034.74612775   14.72847983 0.21252668  100.47390909    0.20469106
Saturn  6   false   s   95.2    9.45    0.5 0.5 0.5 9.53667594  -0.00125060 0.05386179  -0.00050991 2.48599187  0.00193609  49.95424423 1222.49362201   92.59887831 -0.41897216 113.66242448    -0.28867794
Uranus  7   false   s   14.5    4.01    0.25    0.25    0.25    19.18916464 -0.00196176 0.04725744  -0.00004397 0.77263783  -0.00242939 313.23810451    428.48202785    170.95427630    0.40805281  74.01692503 0.04240589
Neptune 8   false   s   17.1    3.88    0.0 0.0 1.0 30.06992276 0.00026291  0.00859048  0.00005105  1.77004347  0.00035372  -55.12002969    218.45945325    44.96476227 -0.32241464 131.78422574    -0.00508664
Pluto   9   false   s   0.0025  0.186   0.5 0.5 0.0 39.48211675 -0.00031596 0.24882730  0.00005170  17.14001206 0.00004818  238.92903833    145.20780515    224.06891629    -0.04062942 110.30393684    -0.01183482
```

Figure 6: Contents of an "‘#kepler"’ Input-File specifying the orbital elements of the Sun, Pluto and all major planets in the solar system

8 Lessons Learned

I learned a lot during this project. I managed to widen my understanding of the following points:

- 3D graphics using the Open Graphics Library (OpenGL): I learned OpenGL 3.0, which means that the programmable pipeline as well as most buffer objects were new to me.
- The programming language C++: I had written object oriented programs before but during this project I massively improved at applying the concept of object orientation. In particular at deciding when to split up functionality between different objects.

During the project I also learned some entirely new concepts:

- Solving a differential equation numerical.
- Working continually on a supervised project. I had done projects of similar scale before, but I had usually not been supervised nor did I have a clear task to accomplish.

9 Declaration of Originality

I certify, as the author of this paper, that I was the person primarily involved in the study designs, implementation, analysis and manuscript preparation. I declare that the work presented in the paper is to the best of my knowledge and belief, original (except as acknowledged in the text).

Jochen K. Illerhaus

10 References

References

[1] A. Brinely Codd M. Fofaria T. Shah S. Amen, P. Bilokon. Numerical solutions of differential equations. *Imperial College London*, 2004.

[2] RenÃ© Schwarz. Keplerian orbital elements -> cartesian state vectors. *Rene Schwarz*, 2014.

[3] E. M. Standish. Keplerian elements for approximate positions of the major planets. *Solar Systems Dynamics Groupe (JPL).*